Inter-class relationships in text classification

Thesis

Submitted in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

by

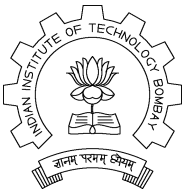
Shantanu Godbole

Roll No. : 00429403

Advisors

Dr. Sunita Sarawagi

Dr. Soumen Chakrabarti



K. R. SCHOOL OF INFORMATION TECHNOLOGY

INDIAN INSTITUTE OF TECHNOLOGY - BOMBAY

MUMBAI - 400 076

2006

Approval Sheet

Thesis entitled "**Inter-class relationships in text classification**" by **Shantanu Godbole** is approved for the degree of **DOCTOR OF PHILOSOPHY**.

Examiners

_____

_____

_____

Supervisors

_____

_____

Chairman

_____

Date: 7 December 2006

Place: IIT Bombay

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY, INDIA

CERTIFICATE OF COURSE WORK

This is to certify that Mr. Shantanu Godbole was admitted to the candidacy of the Ph.D. Degree on January 16, 2001 after successfully completing all the courses required for the Ph.D. Degree programme. The details of the course work done are given below.

Sr. No.	**Course Code**	**Course Name**	**Credits**
1.	IT 620	Seminar	4
2.	IT 606	Mobile Computing	6
3.	CS 610	Information retrieval and mining for hypertext and the web	6

I.I.T Bombay
Date:

Dy. Registrar (Academic)

Abstract

Text classification is an active research area motivated by many real-world applications. Even so, research formulations and prototypes often make assumptions that are not suitable for deployment. For example, in many real applications, the set of class labels keeps evolving, continual user feedback must be integrated into the classifier, and test documents may come from a population statistically different from the training distribution. The main aim of our work is to build solutions for these problems using the idea of exploiting inter-class relationships.

We learn noisy, approximate, and probabilistic mappings between related classes across label-sets in a semi-supervised framework we call cross-training. We exploit the notion of confusion between closely related classes, study its effect on label hierarchies, and present an algorithm for scaling up training of multi-class classifiers. We design discriminative, multi-label classifiers that are robust in the face of significant overlap, in terms of word distributions, between related classes. In many real applications, the set of labels is not predefined but must be constructed from vague specifications and a study of the corpus. Moreover, the label-set has to keep evolving as the corpus changes. We propose an algorithm that supports such temporal evolution by detecting classes in unseen data not defined during training. Our algorithm detects such classes using new notions of coverage of label-sets, support and confidence in a classification setting, and abstractions to represent documents. To enable continual interactive learning and to incorporate human input, we present a framework for active learning that combines terms and documents in a symmetric manner, reducing cognitive burden on the trainer.

We conclude by proposing a new architecture for next-generation text classification platforms that embodies the ideas and contributions in this dissertation. To summarize, our work fills in conspicuous gaps between research prototypes and industry requirements, by exploiting one central idea: class labels are mutable variates just like words, documents and their assigned labels.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivating applications

Text classification is the task of learning models of categorised collections of documents. These models are applied to new documents and one or more categories are assigned by the system. Numerous text mining applications today use some form of text classification and this has fueled extensive research in the area. Various techniques like complex information extraction, feature construction through tagging and shallow parsing, feature engineering, and representation choices are routine steps in setting up classifiers. Efficient training and application, performance tuning, and building understandable classifiers, are continuing fields of text classification research. A few notable applications of text classification are outlined below.

Document (email) filtering and routing is a very important application in large corporate settings. Spam filtering is perhaps the most common application that impacts all of us, with Bayesian or rule-based spam filtering being a necessary component in all client and server mail software. Web directories are an invaluable source of well categorised information on a broad variety of topics on the web, and though manually created for now, there are many applications using them for better information presentation and navigation. News filtering and other personalisation applications also

use some form of text classification. The vision of the semantic web has also brought about a variety of advances in terms of standardisation and toolkits to deal with ontologies and markup languages. The latest surge of applications on the web include social tagging of a variety of content and this poses many unsolved challenges to the study of classification.

Several real-life projects reported at the Operational Text Classification workshops [LGM+03] describe applications that span law, journalism, libraries and scholarly publications. Surprisingly, automated batch-mode techniques did not perform well in these settings. Substantial human involvement was required before a suitable feature set and even label-set could be defined. Many classifiers including simple statistical models and well-tuned rule-based systems were used in these systems. With lots of systems engineering the accuracy attained in such systems was regularly reported to be as high as 90–95%.

1.2 Past research in text classification

We summarise some of the important work in text classification research in this section. One of the most significant developments in the last 10 years has been the use of support vector machines (SVMs) for text classification by Joachims [Joa98]. SVMs are discriminative classifiers well founded in statistical learning theory [Vap95] and are accepted to be the most accurate. Newer work in text generation models like Latent Dirichlet Allocation (LDA) [BNJ02] and the Aspect model [Hof99] give intuitive explanation of the document generation process. Text classification has been studied in a variety of settings including in hierarchies, on-line learning, ensemble learning and so on. Koller et al. [KS97] and Chakrabarti et al. [CDAR98] studied classification of documents organised in hierarchies using discriminative and generative models re-

spectively. Klinkenberg et al. [KJ00] studied concept drift, when the notion of a class as determined by its documents changes over time. The machine learning wisdom of using ensembles of weak classifiers to give strong competitive classifiers has been applied for text by Schapire et al. [SS00] in their Boostexter system. A setting often encountered in real-life is non-availability of labeled training data. For such settings McCallum et al. [MN98b] proposed using unlabeled data for better parameter estimation using the EM algorithm. In case human feedback is available, Cohn et al. [CGJ95] proposed active learning to get instances labeled by humans to assist classifier training.

The missing link: One important aspect common to all the above pieces of work is that the set of classes (label-set) used in the problem is taken for granted. The label-set is assumed to be fixed and unchanging. We believe there is a lot to be gained in treating classes as important entities in the system in addition to documents and terms. Studying properties of sets of classes is one aspect that has been left out in previous work. The main idea we propose in this thesis is that studying the label-set, relationships between its classes, and evolution of the label-set as a whole is very important in approaching a variety of challenging problems in text classification. We elaborate this in the next section, after outlining the assumptions of existing text classification setups.

1.3 Assumptions in text classification setups

The text classification process typically requires a set of documents tagged with a pre-specified set of classes. This training data typically goes through some pre-processing steps to convert the data into a form directly usable by a variety of classification algorithms. The models learned on training data are used to make classification predictions on unseen data; system accuracy is measured by the cor-

rectness of these predictions. Chapter 2 presents background material on training, validation, testing, pre-processing, representation, and evaluation in more detail.

Almost all text classification research assumes some fixed, simple feature representation (such as bag-of-words), and at least a partially labeled corpus. Statistical learners also depend on the deployment data to be reasonably related to the training data. Many of these assumptions do not hold in real-life applications. Discrimination between labels can be difficult unless features are engineered and selected incorporating extensive human knowledge. The semantics behind labeling is liable to be ill-conceived or at least obscure. There is often no labeled collection to start with, or the label-set may not be specified up front, and must evolve with the user's understanding of the application.

The set of problems we work on arise from various assumptions underlying text classification systems. Relaxing these assumptions in turn leads to a variety of different problems and our work focuses on proposing novel solutions to these. Our main approach in tackling these problems is identifying and exploiting various kinds of inter-class relationships; this forms the central idea of our work. Some important assumptions in text classification are outlined below:

- The size of training data for each class is known to directly impact the learnability of the class. Hence all classification systems assume the existence of labeled training data.

- The training and unlabeled (test) distributions of documents are assumed to be similar. This is a valid assumption to make as it enables learning of classification models that can be applied to new documents. The train and test distribution of document streams is susceptible to change, either gradually or in bursts, and systems need to adopt to these settings. An example is the news domain

where new stories are generated over time corresponding to real world events. In a classification setting, new classes may be introduced over time or existing classes may become irrelevant.

- Another simplistic assumption in existing work is that classes are assumed to be disjoint concepts. Ideally, all possible classes can be pre-specified such that it is possible to come up with a representation of documents where discriminating any class from the others is possible. Text documents rarely conform to rigid class boundaries and are not generated by picking a class first and then sampling class conditional word distributions. Authors typically mix concepts into a single document, and different parts of the document may belong to different classes. Thus, documents are realistically multi-labeled (belong to more than one class).

- Most existing research employs the bag-of-words (BOW) model to represent documents. This model considers the occurrence of all tokens independent of each other and is immune to phrases, treating 'New' and 'Delhi' as two features instead of 'New Delhi' as one. In practice, this model is surprisingly found to be about as good as other NLP based representations. The BOW model may work well in terms of just classification accuracy, but there is a lot of value in pre-processing text documents to identify phrases, named entities, visual properties of web documents and so on. Identification and exploitation of meta-level features enables us to propose solutions to other interesting problems. Interpretability and use of complex features is often as important as accuracy.

1.4 Novel applications - challenges and solutions

In this section, we look at some novel applications which result from clear information needs of users in various settings. The main focus of our work was tackling these

problems in a systematic way by treating classes and label-sets as important entities and not take them for granted. We identify four relationships between classes. In general this facet of relationships between classes and promoting the status of label-sets in text classification systems has not been explored in the literature.

We propose four relationships namely, *mappings*, *confusion*, *overlap*, and *coverage*. Learning mappings between related label-sets is an important problem to solve in e-commerce and taxonomy maintenance applications. Scaling large multi-class problems is essential to scale up various existing interesting prototypes to realistic scales and we exploit confusion between classes for this. Multi-labeled classification is a reality often ignored in text classification research and we aim to enhance the performance of multi-labeled classification by countering overlapping boundaries of related classes. We also look at the problem of bootstrapping text classification systems in their early stages of construction. One aspect of this problem is tracking temporal evolution of label-sets as they are being defined and use the coverage of existing label-sets to detect this evolution. The other aspect is that of leveraging human expertise in the form of document and term labeling using active learning principles in an interactive setting; this naturally leads to feature engineering considerations. We describe these problems in detail in the rest of this section.

1.4.1 Learning mappings between classes

The vision of the semantic web [BLHL01] is to universally facilitate information interchange between entities on the web by providing machine understandable semantics. A large effort on this front is in the direction of schema standards and ontology specifications which hope to understand and link various kinds of documents. Content creators on the web have no notion of a single catalog of universal labels to tag their content. News producers, content management sites and blogs all have different, but

sometimes related sets of labels or tags to annotate content. General as well as domain specific topic directories (like DMOZ and Yahoo!) have evolved similar yet different taxonomies of labels. Vendor and distributor catalogs in e-commerce are dynamic, evolving, and need to be mapped onto each other. It is unclear if universal standard taxonomies can emerge outside domain specific settings, and even for those settings there is a need for consolidating legacy data into these standards. Text classification could potentially help tag web content with unambiguous semantic annotations.

Documents are inherently conglomerations of subjective, ill-specified concepts. We believe that any kinds of *mappings* between content-based taxonomies will be *complex*, *uncertain* and *noisy*. One use of such mappings is to explore if better classifiers can be constructed for a taxonomy B, if label assignments of documents in another related taxonomy A are known or can be inferred. We introduce a general semi-supervised learning framework called *cross-training* which can exploit knowledge of such label assignments. Cross-training introduced in Chapter 3 generalises existing generative (like naive Bayes) and discriminative (like SVMs) classification algorithms, while comparing favorably with their baseline accuracy. Other benefits of cross-training include experience with encoding heterogeneous features for learning algorithms and a better understanding of different kinds of mappings between taxonomies [SCG03].

1.4.2 Scaling multi-class classification problems

The phenomenon of *confusion* between classes in a classification setting leads to mis-classification within a group of related classes. This is the result of either obscure or insufficient specification of the label-set or falls out of an insufficient feature selection and representation mechanism. The occurrence of certain keywords in documents does not always follow our intuitive separation between class concepts. The bag-of-words model typically employed to represent documents further leads to overlapping

concept clouds in term-dimensional vector space. Such *confusion* between concepts leads to erroneous classification not only in computer systems, but also for humans - controlled studies involving humans classifying content show high disagreement between human labelers [LYRL04].

One way to mitigate the effect of confusion in large text collections has been to organize classes into hierarchies. The hope in such a hierarchical organization is that different features will be active in different parts of the hierarchy, and it should be possible to build high performance classifiers using such a hierarchical class structure. Multi-class classification results comparing such hierarchies (Pachinko machine classifiers [KS97]) against flat classifiers built only on leaf nodes have proved inconclusive except in special cases. Hierarchies suffers from multiplication of errors at each level as classification proceeds down from general to specific classes. In Chapter 4, we develop the notion of *confusion* amongst classes and present an efficient algorithm called *GraphSVM* for constructing a scalable multi-class classifier on leaf nodes that turns out to be competitive or even better than multi-class classifiers while being significantly better in training time and memory requirements [GSC02].

1.4.3 Enhancing multi-labeled classification

Another problem related to *confusion* and *overlap*ping class boundaries is that of multi-labeled classification. Most applications require the ability to classify documents into one out of many (> 2) classes as often it is not sufficient to talk about a document belonging to a single class. Based on the granularity and coverage of the set of classes, a document is often about more than one topic. A document describing the politics involved in the sport of cricket, could be classified as `Sports/Cricket`, as well as `Society/Politics`. When a document can belong to more than one class, it is called multi-labeled. Multi-labeled classification is a harder problem than just choosing

one out of many classes. In addition to estimating the classes a document belongs to, the additional problem is to determine how many classes are relevant for the document in hand. The problem is compounded because closely related classes often have overlapping class boundaries because they share many common features which would otherwise help discriminate the classes.

In Chapter 5 we present algorithms which use existing discriminative classification techniques as building blocks to perform better multi-labeled classification. We propose two enhancements to existing discriminative methods. First, we present a new algorithm which exploits correlation between related classes in label-sets of documents; this is similar to our work on cross-training in Chapter 3. Next, we present two methods of improving the margin of SVMs for better multi-labeled classification. We show experimental results comparing various multi-labeled classification methods.

1.4.4 Bootstrapping text classification systems

An important set of challenges in development and deployment of text classification systems is bootstrapping new systems as and when they are being constructed. We observe that such bootstrapping can happen at various levels and we study these problems and propose novel solutions. Bootstrapping label-sets could be required when the data as well as the user's understanding of the application evolves over time. Learning in the presence of very limited labeled data (labeled documents) also poses the problem of bootstrapping classifiers by incorporating human labeling.

One of the major assumptions in text classification research [Joa98, NLM99, ZY03] is that statistical learners assume the deployment data to be reasonably related to the training data. This assumption does not hold in many real life systems. Thus, an important challenge in building text classification systems is recognising that the constitution of unlabeled data changes over time. Often new classes are introduced

and need to be detected and folded into the system. We call this the **evolving label-set** problem. We propose the notion of *coverage* that looks at whether a given label-set captures the spread of topics that documents in a corpus are about.

For example, consider a classification problem with n classes, where the classes are documents about certain countries (*India*, *US*, *UK*, ...). Over a period of time, a new country's documents (say Australia) are introduced into the system. The evolving label-set problem is to detect such (one or more) new classes, propose a cohesive set of documents for training the new classes, get user for validation about these fitting in with the label-set, and fold these new classes into the classification system. The existing label-set here does not cover all the countries that the given set of documents are about and the label-set needs to be expanded to increase its *coverage*. Such problems occur especially when a nascent classification system is built from scratch and the set of labels evolves over time with the user's understanding of the application. We design algorithms for identifying new classes in both generative and discriminative settings in Chapter 6. We introduce the notion of *abstractions* to capture the importance of terms not encountered during training, and also to provide a representation that more intuitively reveals the classification criteria to the user. We also present a method for automatically triggering detection of a new class in unlabeled data by the system.

There is much scope for building machine learning tools which engage the user in an active dialog to acquire human knowledge about features and document labels. When such supervision is available only as label assignments, *active learning* provides clear principles [CGJ95, TK00, FSST97] and strategies for maximum payoffs from the dialog. We wish to extend the active learning paradigm significantly to include both feature engineering and document labeling conversations, exploiting rapidly increasing computing power to give the user immediate feedback on her choices. One of our main goals is to amplify human effort through machine learning, thus scaling up data sets

that can be practically processed.

1.4.5 Next generation text classification platforms

Almost all machine learning text classification research assumes some fixed, simple class of feature representation (such as bag-of-words), and at least a partially labeled corpus. Discrimination between labels can be difficult unless features are engineered and selected with extensive human knowledge. Several real-life applications in various domains note that batch-mode techniques are not satisfactory; substantial human involvement is required before a suitable feature set, label-set, labeled corpus, rule base, and resulting system accuracy is attained. However, not all the techniques used in commercial systems are publicly known, and few general principles can be derived from these systems.

We propose a new architecture for next-generation text classification platforms that embodies the ideas and contributions in this dissertation. Our work fills in conspicuous gaps between research prototypes and industry requirements, by exploiting the central idea that class labels are mutable variates just like words, documents and their assigned labels. We feel the time is ripe to architect such a next generation platform for text classification that addresses some of these differences in perception between the research and industry communities. The focus of our work has been to promote the status of classes and label-sets in text classification settings by discovering and exploiting inter-class relationships and also exploring the problem of bootstrapping nascent text classification systems. In Chapter 7 we propose a architecture for a next-generation platform that is built upon three main entities, (1) document and classification models, (2) feature engineering components, and (3) label-sets, and the interactions between these three. We would not like to take classes for granted and would like to structure our applications around label-sets and their relationships. All

our work explained above neatly fits into the architecture of such a proposed platform.

We have had good initial success in building a prototype of such a platform called *HIClass* (for Hyper Interactive text Classification) described in Chapter 7. *HIClass* is an interactive workbench that provides a tight interaction loop between human experts and statistical classifiers. We extend SVMs to naturally absorb human inputs in the form of feature engineering, term inclusion/exclusion and term and document labels. In the past, such actions were performed through ad hoc means and as a distinct processing step before classification construction. We make these more effective by (1) providing the user easy access to a rich variety of summaries about the learned model, the input data and aggregate performance measures, (2) drawing the user's attention to terms, classes or documents in greatest need of inspection, and (3) helping the user assess the effect of every choice on the performance of the system on test data. These are some of the interaction mechanisms a full-fledged next-generation text classification platform should contain and we discuss other such requirements in detail.

1.5 Point wise summary of contributions made

In this report, we explore and devise a range of algorithms to solve real-world text classification problems. The main focus of our work is exploiting inter-class relationships; this has been a neglected area of text classification research. We identify four relationships between classes: (1) we learn and exploit *mappings* between label-sets to help build better classifiers and taxonomy maintenance tools, (2) we exploit *confusion* and present techniques to handle scalability issues in large text collections, (3) we present algorithms to overcome *overlap* to enhance discriminative multi-labeled classification, and, (4) we introduce the notion of *coverage* of label-sets

to detect temporal evolution of label-sets by detecting new classes in unlabeled data that were not present during training. We also focus on problems related to bootstrapping text classification systems with challenges of lack of well defined label-sets which evolve over time (point 4 above) and low availability of labeled training data to learn models from. We significantly extend document and term level active labeling conversations to harness human expert labeling knowledge while exerting little cognitive load on the user.

Our work promotes the importance of classes as mutable entities in text classification systems along with documents and features. Our work revolves around treating classes as first level entities and exploiting different inter-class relationships. We present an architecture for general-purpose text classification platforms as we feel the time is ripe to bridge the gap between academia and industry perceptions of text classification systems, actively integrate human knowledge, and relax many of the standard assumptions.

1. We present a general semi-supervised framework called cross-training in Chapter 3 to learn noisy, approximate, and probabilistic mappings between label-sets.

2. We design generative algorithms for cross-training called EM2D with variants. EM2D builds upon NB and decisively learns better classification models. EM2D results in better classifiers for both taxonomies and finds interesting mappings between the constituent classes. The results in terms of accuracy improvement are good.

3. We design a discriminative cross-training algorithm called SVM-CT. The use of SVM-CT for augmenting and maintaining taxonomies has definite potential; some examples of mappings between label-sets for taxonomy maintenance tools are shown in the appendix.

4. Closely related classes have overlapping class boundaries and confuse classifiers. This results in lower accuracy and forces construction of time-consuming, resource intensive classifiers. In Chapter 4, we exploit this notion of *confusion* among similar classes.

5. We study confusion matrices in detail and presented an algorithm for automatically generating a hierarchy of classes from a given flat set based on classification similarity of classes. This output is in the form of a dendrogram of classes; there are some examples in the appendix. We present a method to create hierarchical classifiers using these dendrograms.

6. We present the *GraphSVM* algorithm to tackle the problem of scalability of efficient multi-class classifiers like SVMs. *GraphSVM* builds on the confusion matrix of a fast moderately accurate classifier like NB and produces efficient SVM ensembles with lesser training data by exploiting the confusion between classes. It performs as well or better than SVMs trained on all data but is many times faster and needs much less memory.

7. Discriminating between closely related classes is hard as they often share important features, and their class boundaries are fuzzy, and *overlap* with each other. In Chapter 5, we propose algorithms for better multi-labeled classification with discriminative methods. Our algorithms performed better than usual methods on different accuracy measures and we gave interpretable mappings between correlated classes within label-sets.

8. We present an algorithm to exploit correlation between classes along the lines of cross-training. We also present algorithms to improve the separating surface of discriminative learners by altering data close to decision boundaries.

9. Bootstrapping text classification systems in the presence of limited training data is an important area of research. We propose algorithms to deal with temporal evolution of label-sets, document and term labeling, and feature engineering in Chapter 6.

10. Topics covered in a classification system change with time and new classes need to be discovered and folded into the system. We propose the novel idea of using *abstractions* for better representation of documents to help the user get an idea of the coverage of the label-set and estimate fit of proposed new classes into the existing label-set.

11. We propose generative and discriminative methods based on notions of support and confidence respectively to propose a candidate set of unlabeled documents for consideration for adding a new class in the label-set. We also propose heuristics for the system to automatically trigger on detection of new class candidates in unlabeled data streams. Our algorithms do well in terms of precision of suggested classes and also show low triggering error rates.

12. We present a multi-class multi-labeled framework for active learning of documents using linear additive models like SVMs. We present many aids for reducing the expert user's cognitive load; these aids include bulk labeling, ranked list of suggested labels, and conflict checking.

13. We introduce the novel idea of active learning on terms (instead of documents). In the initial stages of classifier construction when very few documents are available, our method is able to identify a good set of features for human labeling from the unlabeled pool of data.

14. We present a new OLAP-like interface for the user to browse term-document-

class matrices of learned models. The user can drill down on documents for influence of certain features on certain classes or vice-versa. Anywhere in this interface, the user can engineer features using add/delete/ignore directives for certain features which are obviously indicative (or not) of certain classes. This feature engineering component is an interesting way for incorporating rule-bases and human knowledge into classification models like SVMs.

15. We propose a new architecture for next-generation text classification platforms that embodies the ideas and contributions in this dissertation. Our work fills in conspicuous gaps between research prototypes and industry requirements, by exploiting one central idea: class labels are mutable variates just like words, documents and their assigned labels. We see how this naturally gives rise to the many applications we have worked on in different parts of this report.

16. We describe our experience in building *HIClass*, an interactive text classification workbench along the lines of such a platform. Our experience has been extremely fruitful and we could achieve many of the desirable characteristics we aimed for. We provided heavy interaction with experts while exerting a low cognitive load, and could bootstrap classification system very well after starting with very little training data. We were able to facilitate interaction between humans and systems, leveraging human expert knowledge and the data processing power of machines.

1.6 Organisation of the report

We present some background material common to all our work in Chapter 2 where we review document corpora, representations, classifiers, and evaluation techniques. We present the cross-training framework in Chapter 3 where we introduce the notion

of mappings between classes across taxonomies. In Chapter 4 we present the notion of confusion between classes and use it to study hierarchies and design scalable efficient training techniques for multi-class classifiers. We present overlap as the third kind of inter-class relationship in Chapter 5 and we design better discriminative methods for multi-labeled classification by overcoming overlap. In Chapter 6 we explore issues in bootstrapping text classification systems. We introduce the notion of coverage of a label-set to track changes in distribution of unlabeled data by detecting new classes introduced in the system. We present active-learning based methods (on documents as well as features) to bootstrap classifiers built on very little training data. We propose a broad architecture for next-generation text classification platforms in Chapter 7 and describe our initial experience with building such a system. We include related work, a discussion, and a summary with every chapter. Finally, we conclude this report in Chapter 8 and point out some avenues for future work.

Chapter 2

Background on text classification

In this chapter we review some background material for studying text classification problems. This material and notation is common to all systems developed and used in our work. It is required background to reading most work in text classification literature.

2.1 Problem setting

The text classification task is typically one of learning models for a given set of classes and applying these models to new unseen documents for class assignment. The given set of classes is called the **label-set** and for each class in the label-set a set of representative documents is provided as training data. Machine learning algorithms use this training data from all classes to construct various types of statistical or rule-based models. These models are applied to unseen documents which are presented for label assignment. We note here that there exist another kind of leaning models called *lazy learners*, like k-nearest neighbour classifiers, that don't actually learn any models but assign labels to unseen documents based on the label assignments of their neighbours. In the rest of this thesis we use classification and categorisation synonymously. Similarly, the terms *class*, *label*, and *category* are used interchangably.

The label-set is either a flat set of classes as in simplistic email routing. A more intuitive and managable organisation for large sets of classes is a hierarchy or taxonomy of classes. A taxonomy is typified by large web directories like Yahoo! [1] or DMOZ [2]. These taxonomies represent all world knowledge at the root of the tree-structure and specialise into topics like Sports, Science, News, Recreation at the first level. Further specialisation at the second level has classes like Football and Cricket under Sports, Physics and Chemistry under Science and so on. Such taxonomies can be arbitrarily deep although most researchers prefer dealing with a small number of levels or even a flat organisation of classes and ignore the taxonomy.

The label-set is denoted C, with individual classes denoted c_1, c_2, ..., c_n or simply c for a total of $|C| = n$ classes. Unless otherwise stated we deal with a flat set of classes. The simplest classification problem involves differentiating a class of interest from uninteresting documents. Here $n = 2$. When $n > 2$ we have a multi-class classification problem. Here documents can belong to one of n classes but every document can belong to only one class. When a document can belong to more than one class at a time, we have a multi-labeled classification problem. For example, a document talking about politics in the sport of cricket today, belongs to the Politics and Cricket classes. Note that multi-labeled problems always have $n > 2$ and are thus also multi-class problems. However not all multi-class problems need to be multi-labeled and are usually not unless otherwise stated.

2.1.1 Pre-processing

Typically while building text classification systems, the pre-processing steps include various complex information extraction steps. Feature engineering through var-

[1] `http://www.yahoo.com`
[2] `http://dmoz.org`

ious natural language processing (NLP) tasks like named-entity (NE) tagging, shallow parsing, part-of-speech (POS) tagging, and phrase-level chunking is common. The choice of representation for text documents is important before invoking any learning method. Efficient training and application, performance tuning, and building understandable classifiers are all continuing fields of text classification research. We look at some more details of pre-processing in Section 2.2.

2.1.2 Types of classifiers

The two broad types of classification methods are discriminative and generative. Discriminative methods like SVMs or logistic regression (LR) [ZY03] are two-class classifiers that find separators between documents of two classes in some space of representations. Other discriminative models include maximum entropy methods [NLM99] and boosted decision trees in the ADABoost framework [FS99]. Generative methods are typified by naive Bayes (NB), the Aspect model [Hof99], Latent Dirichlet Allocation [BNJ02], and the more recent BayesANIL [RCKB05]. Discriminative methods are widely accepted to be more accurate, but generative methods provide intuitive text generation models and have been used in a variety of applications.

The industry has also made significant advances in the development and deployment of real-world high-performance text classification systems [LGM+03] using combinations of rule-based, hand-tuned, and statistical techniques. However, not all the techniques used in commercial systems are publicly known, and few general principles can be derived from these systems.

Outline: We have described the text classification problem setting in Section 2.1. Next in Section 2.2 we look at details of any text corpus or collection of documents. We review the features used in the text classification setting in Section 2.2.1 and

Section 2.2.2 and look at various representation issues in Section 2.2.3. Next in Section 2.3 we look at the actual classification algorithms used for learning models of classes and take a look at training and prediction procedures. In particular we look at the naive Bayes (NB) classifier in Section 2.3.1 and support vector machines (SVMs) in Section 2.3.2. Section 2.4 contains details on standard methods of evaluation used for text classification systems. In Section 2.4 we look at common evaluation methodologies based on various measures outlined in Section 2.4.1. Some commonly used benchmark text datasets are described in Section 2.4.2.

2.2 Document Corpus

The corpus of documents containing the training and unlabeled data is denoted D. The training documents are denoted T and unlabeled documents are denoted U, hence $D = T \cup U$. An individual document is denoted d or the i^{th} document is denoted d_i. Next, we look at features of documents used in classification algorithms following which we take a look at representation issues.

2.2.1 Feature extraction

Every document d contains all the words, spaces, markups, and tags which occur in it. Consider any news article or a random page from the web. Not all the words in the document are useful and in vanilla text classification tasks only the text portions of the page is taken into account. Other applications including search rely heavily on taking the markup or link structure of documents into account but in text classification tasks we usually only consider the text portions of the page; there are specialised applications that account for other content.

Considering only text portions of a page involves ignoring all markup, tags, and whitespaces. Common stop-words (like a, an, the, for) which are not useful in distin-

guishing one page from another are also ignored. The standard stop-word list used is the SMART list [3]. Stop-words are domain specific and the SMART list contains stop-words for the general English language.

Following stop-word removal, all text tokens on the page are taken as individual features to be considered for classification. Every word in the write-up of the page now becomes a feature. Another common operation in extracting features is the operation of stemming where words like 'achieve', 'achiever', 'achieved', are all contracted to their stemmed form of 'achiev'. The Porter stemmer [Por80] is available in many forms freely.

The above approach of converting pages to a set of features is called the bag of words (BOW) model where words like 'New' and 'Delhi' will occur as separate features instead of the intuitive single feature 'New Delhi'. A lot of natural language processing (NLP) research is devoted to detecting such phrases or named-entities (NEs) in text documents. This is a first step in moving from the BOW model to more intelligent models for feature selection in documents. Part-of-speech (POS) tagging is also employed to generate POS n-grams as other kinds of features. All these NLP derived feature sets have been used in text classification but surprisingly none have been found to significantly improve over the simple BOW model.

2.2.2 Feature selection

It is seen that in-spite of stop-word removal and stemming, not all features extracted are useful in classification tasks. In fact a lot of unnecessary features harms the learnability of models. Hence feature selection is a standard technique used to reduce the number of features. Feature selection typically employs some statistical measures over the training corpus and ranks features in order of the amount of in-

[3] `ftp://ftp.cs.cornell.edu/pub/smart/`

formation (correlation) they have with respect to the class labels of the classification task at hand.

The typical measures used to rank feature lists are mutual information with the class label, information gain, and other such measures. After the feature set has been ranked, the top few features are retained (typically order of hundreds or a few thousand) and the others are discarded. Typical large text corpora contain tens to hundreds of thousands of unique features.

Such feature selection is not needed for all classification algorithms as some classifiers are capable of feature selection themselves. However for some other classifiers feature selection is mandatory and a large number of bad features harms the classifier accuracy significantly.

2.2.3 Representation

Once features are extracted from documents, each document is converted into a document vector. Documents are represented in a vector space; each dimension of this space represents a single feature and the importance of that feature in that documents gives the exact distance from the origin. Documents are thus points (vectors) in a $|V| = v$ dimensional vector space where V denotes the vocabulary of all features.

Representation of features is very important as it forms the basis for most classification algorithms. Algorithms work on these document vectors and the classification task hopes to distinguish documents of one class from the other, so there must be significant differences in the vectors of different documents. The simplest representation of document vectors uses the binary event model, where if a feature $j \in V$ appears in document d, then the j^{th} component d_j is 1 otherwise it is 0. This can be replaced by term frequency (TF) where d_j is the count of the number of times j occurs in the document. Since a TF representation can reorient the document vector unfairly in

directions of features with lots of occurrences, the TF representation is modified by the 'importance' or rarity with which the feature j occurs in all documents of the corpus. If $DF(j)$ is the document frequency of j occurring in all documents D, then for the j^{th} feature, the d_j^{th} component of d is $d_j = TF(j) * log\frac{|D|}{DF(j)}$ (many variants exist). This is known as the TFIDF representation of documents.

The final issue in representing document in vector space is accounting for different sizes of documents. There has been extensive work in normalising document vectors for fair representations across the corpus [LK02]. These normalisation approaches yield minor differences in performance with respect to each other, and the results are very dataset dependent. The most commonly used length normalisation is the Euclidean length normalisation where are document vectors are normalised to unit norm by the L_2 distance metric. Comparative studies have been done with binary, TF, and TFIDF representation along with no, L_1, and L_2 unit normalisation. Normalising vectors seems to most significantly affect text classification performance especially with discriminative classifiers.

2.3 Classifiers

In this section we review some of the algorithms commonly used for text classification. Once document vectors of the entire corpus are ready as per pre-processing steps described in the previous section, a variety of learning algorithms can be applied. The two broad types of classification methods are generative and discriminative methods.

Generative models: Generative methods include naive Bayes (NB), Latent Dirichlet Allocation (LDA), the Aspect model, and BayesANIL. All generative methods model some kind of probability of generating the corpus i.e. $Pr(d|c)$ or the probability of generating a document given a class.

Discriminative methods: This is very different from discriminative methods typified by support vector machines (SVMs), logistic regression (LR), decision trees, ensembles of decision trees (like ADABoost). Discriminative methods directly try to learn models for predicting $Pr(c|d)$ or finding the class most appropriate for the document at hand.

Generative models are moderately accurate but very efficient and their power comes in providing very good document generation models which find wide spread application. Discriminative methods are accepted to be the most accurate but require inefficient training methods to be invoked. Other kinds of classifiers like highly tuned rule-based systems are very effectively used in the industry where maintainability and interpretability of the rules is of prime concern. Such rule bases have been developed over long period of careful tuning but all this knowledge is not publicly available.

We review NB classifiers in Section 2.3.1 and SVMs in Section 2.3.2 as common representative classifiers. A lot of our work uses these two classifiers and they also find common application in the literature.

2.3.1 Naive Bayes (NB)

In NB classification for a single label set C,

$$\begin{aligned} \Pr(c|d) &= \frac{\Pr(c,d)}{\Pr(d)} = \frac{\Pr(c)\Pr(d|c)}{\Pr(d)} \\ &\propto \Pr(c)\Pr(d|c) \propto \pi_c \prod_{t\in d} \theta_{c,t}^{n(d,t)}, \end{aligned} \tag{2.1}$$

where $c \in C$ is the label, d is the test document, t occurs $n(d,t)$ times in d, π_c is the fraction of documents tagged c (also called the *prior* probability of c), and $\theta_{c,t}$ are multinomial probability parameters [MN98a], estimated from training documents as

$$\theta_{c,t} = \frac{\lambda + \sum_{d\in D_c} n(d,t)}{\sum_{\tau\in T}\left(\lambda + \sum_{d\in D_c} n(d,\tau)\right)}, \tag{2.2}$$

where T is the vocabulary or feature set, D_c is the set of training documents marked with label c, and $0 < \lambda \leq 1$ is the *Lidstone's smoothing* parameter [Ris95] ($\lambda = 1$ corresponds to the well-known Laplace's smoothing). Having estimated model parameters from training data, the goal is to find the best class $\arg\max_c \Pr(c) \Pr(d|c)$ for test documents.

The open source toolkit called Rainbow [McC98] is very widely used for its implementation of NB and a few other classifiers. Rainbow is a very good bag-of-words processing toolkit that has many options for document parsing, vector creation, feature selection, representation choices, summary statistics, and many implementations of different classifiers. We use Rainbow extensively for document pre-processing and feature selection.

2.3.2 Support Vector Machines (SVMs)

Suppose we are given a vector representation of D documents. Each vector used TFIDF representation of features normalised to unit L_2 norm. Each document vector is associated with one of two labels, $+1$ or -1. The training data is thus $\{(d_i, c_i), i = 1, \ldots, n\}, c \in \{-1, +1\}$.

A linear SVM finds a vector $\mathbf{w}$ and a scalar constant b such that for all i, $c_i(\mathbf{w} \cdot d_i + b) \geq 1$, and $\|\mathbf{w}\|$ is minimized. This optimization corresponds to fitting the thickest possible slab between the positive ($c = +1$) and negative ($c = -1$) documents. In case the training samples are not linearly separable, it is possible to trade off the slab width for the number of misclassified training instances. A more complete and theoretical introduction to SVMs can be found in Burghes' tutorial [Bur98] and Vapnik's book [Vap95].

If the data has more than two labels, it is common to create an *ensemble* of yes/no SVMs, one for each label. During training, a document marked c is a positive example

for the SVM associated with c, and a negative example for all other SVMs. This is called the "one-vs-others" ensemble approach. During testing, for a test document d, each SVM evaluates its regression function; the SVM corresponding to label c evaluates $\mathbf{w}_c \cdot d + b_c$. The label chosen is $\arg\max_c(\mathbf{w}_c \cdot d + b_c)$ (other policies can also be used).

Other policies can be used instead of one-vs-others. One-vs-one for example trains all possible pairs [Kre99] of classes against each other resulting in $\binom{n}{2}$ binary SVMs. The winning class is chosen by majority voting. Error correcting output codes [DB95] and DAGSVM [PCST00] are other methods for decomposing a multi-class problem into many 2-class problems. Detailed studies have found these approaches to be very similar in performance [HL01], hence one-vs-others is usually used widely because of ease of implementation and clear semantics. We also use one-vs-others in all our work unless otherwise stated. Such decomposition is also needed for other discriminative techniques like logistic regression.

Inducing a SVM classifier involves a complex, iterative numerical optimization. Several implementations of SVM are publicly available, including Sequential Minimum Optimization (SMO) [Pla98, DPHS98], SVMlight [Joa], and LibSVM [CL]. We have used both in our work.

This issue of decomposing multi-class classification problems into multiple binary ones are common to most discriminative methods which are 2-class classifiers by definition. Regularised least squares and logistic regression [ZY03] are other commonly used discriminative classifiers which perform comparably in accuracy to SVMs.

2.4 Evaluation

In this section we review standard evaluation procedures used in text classification research and industry systems. We look at the standard evaluation methodology,

followed by a look at various measures in Section 2.4.1, and conclude by describing commonly used benchmark datasets in Section 2.4.2.

The common metric of interest in evaluating text classification systems is how accurate the system is i.e. what fraction of documents belonging to known classes are correctly assigned those classes. The standard way of doing this is to take a labeled portion of the corpus as the training data, and use the remaining fraction of labeled data as test data. The system is trained on this training data and evaluation performance is measured on the test data - the known labels of test data are hidden and compared against label predictions from the system. A typical train-test split of a labeled corpus is in the ratio $70 : 30$.

Sometimes a part of labeled data is also held aside for tuning purposes. The system is trained on training data, tuned on this held aside data called validation data, and tested against the test data. A typical ratio for training, validation, and test data is $60 : 20 : 20$.

Experiments are often repeated with different random splits into train, validation, and test datasets. Usually the corpus is randomly split and evaluated from 5 to 30 times and the mean and variance of the accuracies is reported. Techniques like cross-validation and k-fold validation help guard against randomness in particular data splits and make the results more sound. k-fold validation involves splitting the data in k parts, using $(k-1)$ parts for training and the remaining part for testing. This is repeated k times for each part as the test set one at a time. Average results of k runs are reported. Refer to Haykins [Hay00] for details on cross-validation.

When comparing two algorithms, experiments are performed multiple times and statistical tests like the paired t-test are used to qualitatively judge whether one algorithm is better than the other.

2.4.1 Measures

There are many measures used to evaluate various aspects of text processing and information retrieval systems. See Rijsbergen's classic book [Rij79] for details of many of these measures. We introduce the most commonly used measures in the text classification context below.

For a binary classification problem, there is a positive class and a negative class. A 2-by-2 confusion matrix shown in Figure 2.1 gives the number of documents predicted correctly and incorrectly into the two classes.

Predicted → True ↓	+ve	-ve
+ve	a	b
-ve	c	d

Figure 2.1: 2-by-2 confusion matrix

In this confusion matrix, $(a + b)$ are the number of documents which truly belong to the positive class of which a were correctly predicted to be positive (true positives) and b were predicted negative (false negatives). $(c + d)$ documents belong to the negative class of which c were incorrectly predicted positive (false positive) and d were correctly predicted negative (true negative). Based on this:

$$Accuracy = (a + d)/(a + b + c + d) \tag{2.3}$$

$$Precision = a/(a + b) \tag{2.4}$$

$$Recall = a/(a + c) \tag{2.5}$$

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{2.6}$$

$$\tag{2.7}$$

In multi-class classification problems, results need to be averaged over confusion matrices of all n classes in the label-set. The two methods of averaging are common to all the above measures. Micro-averaged precision (similarly recall and F1) is the average of all precision values weighted by the number of positive instances in each class $(a + b)$. This measure is heavily influenced by the performance of highly populated classes in the case of skewed instance distribution in a multi-class setting. Macro-average precision (similarly recall and F1) is the simple average of all precision values without weighting. This measures ignores the skew in the instance distribution and treats all classes equally. Both types of averaging have their own uses for maximisation depending on the application.

In multi-labeled classification problems, these evaluation measures have a slightly different interpretation and we review these in the relevant chapter in Section 5.4.

2.4.2 Benchmark datasets

Standard benchmark datasets are used in all text classification research for evaluation. In this section we review some of the common benchmark datasets we have used in our work and describe their specific characteristics. Any specific processing done on the datasets is described at the start of the relevant experimental sections. Amongst standard pre-processing steps we apply stop-word removal from the SMART list, apply word stemming using the Porter stemmer [Por80], and select features with high mutual information (or other statistical measures like information gain).

Reuters-21578

The Reuters-21578 Text Categorization Test collection[4] is the most popularly used benchmark dataset. It contains 135 classes distributed in a set of SGML files. The

[4] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

dataset is a multi-class multi-labeled dataset and the standard Mod-Apte train-test split is popularly used in the literature. All SGML tags are ignored and only the title and body of the message is taken into account. Various researchers have used many standard subsets of this dataset. In particular people have used only the most populous 10 classes or classes with at least a minimum specified number of training documents.

20 Newsgroups

The 20-newsgroups (20NG) dataset[5] is a collection of $18,828$ news wire articles from 20 Usenet groups. The older version of the dataset had nearly $1,000$ articles in each group, but the newer version we use has duplicate posts removed and has most post headers removed. This dataset is not pre-processed into training and testing sets. We randomly chose 70% of the documents for training and the remaining 30% for testing. This is repeated 10 times and average numbers are reported. The corpus contained around $75,000$ words. All HTML tags are skipped and all header fields except subject and organization of the posted article are ignored.

Patents (WIPO)

The Patents dataset is the *wipo–alpha* collection of an English language collection of patent applications. These are classified into a hierarchy of classes with subclasses and groups. We take all 114 sub-classes of the top level (A to H) using the given train/test split. This is a multi-labeled dataset. Only the text in the title and abstract of each patent application is used for classification purposes.

[5] `http://www.ai.mit.edu/~jrennie/20_newsgroups/`

RCV1

The RCV1 [LYRL04] dataset[6] is a collection of one year of Reuters news stories from August '96 to August '97. There are more than 800,000 stories and the suggested train-test split for this dataset is to use the first 12 days stories for training and the remaining for testing. However different applications and researchers have customised this split as appropriate for the problem at hand.

The news stories are organized into three unrelated label-sets or taxonomies: *regions*, *topics*, and *industries*. The *topics* and *industries* label-sets are hierarchical while *regions* is flat. Stories in this dataset are assigned labels from all three label-sets and multi-labeling within a label-set is common. These stories are distributed as a set of XML files and only the main title and main body of the news story is considered for classification.

Webcrawls

For some of our work with web related applications we required real-life data crawled from the web. Web data has different characteristics from controlled benchmark datasets; they are more noisy, hyper-links can be spammy, and a lot of junk pages like browser-dependent ones get crawled. We describe 5 such datasets we crawled from the Yahoo! and Dmoz directories.

Our examples were collected from the Dmoz and Yahoo! directories. Their intersection had 110926 documents, less than 10% of either's total size. Like A&S [AS01] we selected five data sets: Autos, Movies, Outdoors, Photo, and Software. However, their sub-topics and training examples were not available to us. Therefore, for each data set, in each of the two taxonomies, we picked immediate children as labels such that there were at least 10 URLs in common with a label of the other taxonomy. We

[6] `http://trec.nist.gov/data/reuters/reuters.html`

then added in a few additional labels from each taxonomy. Finally we went back to the original Dmoz and Yahoo! sources to collect all URLs within the chosen label sets; some 70–80% of the fetches succeeded.

Table 2.1: Sizes of various document and label sets in our collected data

Data set	$\lvert D_A - D_B \rvert$	$\lvert A \rvert$	$\lvert D_B - D_A \rvert$	$\lvert B \rvert$	$\lvert D_A \cap D_B \rvert$
Autos	3589	31	3138	24	184
Movies	8003	33	11420	27	1222
Outdoors	8739	26	1540	39	181
Photo	2895	8	438	22	95
Software	9851	51	2383	25	264

Table 2.1 shows various properties of our crawled data sets. We felt uncomfortable about the small test sets, but related work reported intersections of similar small sizes. Given the size of the web such small intersection sizes can in fact be expected from manually created directories. Further, we also found human labeling (based on page text alone) to systematically reject pages with relatively unreliable text.

Chapter 3

Learning mappings between classes

3.1 Introduction

In this chapter we introduce the first inter-class relationship we propose in this thesis. We look at discovering and exploiting mappings between sets of classes or label-sets. Mappings between label-sets have interesting application in the web and e-commerce domains. We propose to use mappings to enhance accuracy of existing classifiers. We also show how mappings can be useful for data integration and taxonomy maintenance.

An important step in information interchange and integration of software systems is to be able to identify various kinds of mappings between domain specific ontologies, taxonomies, and label-sets. This is an important step toward realising the vision of the semantic web. The Web has evolved without central editorship and it is unclear if universal standards will emerge outside specific application segments. Even for those segments, there is a need to consolidate legacy data into content organized as per agreed-upon standards; standards that are far from static.

Since content creators have no notion of a universal catalog of labels, news sites, content management sites, blogs, all have different, but sometimes related sets of labels to annotate content. Topic directories have evolved different taxonomies of labels while

vendor and distributor catalogs in e-commerce are dynamic and evolving.

A few examples will illustrate the current scenario. Consider the Yahoo! and Dmoz directories. Both cover the Web and have evolved to similar taxonomies, but show non-trivial differences. Among other artifacts, *taxonomy inversion* is rampant in the `Regional` categories; what one calls `Reference.Education.Colleges_and_Universities.Asia.India`, the other might call `Regional.Asia.India.Education`; in fact, they sometimes coexist in the same taxonomy! Other relationships are also common: `Dmoz.Recreation.Outdoors.Speleology` overlaps `Yahoo.Recreation.Outdoors.Caving`, but there are important non-overlapping sub-topics.

As another example, an e-commerce site which consolidates catalogs of goods and services may want to organize them according to the (still evolving) codes being developed in cooperation between ECCMA and UNSPSC (see `http://www.eccma.org/unspsc/`). Meanwhile, vendors may have their own custom/legacy codes which generally evolve over time.

Documents are inherently conglomerations of subjective, ill-specified concepts. Any kind of *mappings* between content-based taxonomies will be *complex*, *uncertain* and *noisy*. Text searching, ranking, and mining tools must exploit any available relationships, even probabilistic ones, between diverse taxonomies and tag sets. One use of such mappings is to explore if better classifiers can be constructed for a taxonomy B, if label assignments of documents in another related taxonomy A are known or can be inferred.

Outline: In this chapter, we introduce a general semi-supervised learning framework called *cross-training* which can exploit knowledge of such label assignments. In the rest of this chapter, classes c_A form label-set A which is attached to documents D_A (similarly c_B and D_B for B). Section 3.2 introduces the generative cross-training

framework that builds upon NB classifiers and the Expectation Maximisation (EM) algorithm. Section 3.3 introduces cross-training in a discriminative setting using SVMs. Section 3.4 shows detailed experiments where cross-training matches or outperforms baseline accuracy on several real world learning tasks and shows us how to learn relationships between taxonomies. We discuss related work in Section 3.5, look at some interesting aspects of cross-training in Section 3.6, and summarise in Section 3.7.

3.2 Generative cross-training

Generative classifiers try to model the process by which a corpus of training documents is generated. They use this model to predict labels for test documents. A naive Bayes (NB) classifier posits that a document is generated by first fixing a label from a label-set by invoking a (typically multinomial) prior distribution on labels, and then creating the document by invoking a term (feature) distribution conditioned on the label just chosen. Our aim in generative cross-training is to learn a model that examines the generation process of documents from two label-sets simultaneously. We want to explore probabilistic mappings between topics in two label-sets and to this end we want to see if label assignments of documents in one taxonomy help us learn a better generative model for a second taxonomy.

In Section 3.2.1 we look at a method based on Expectation Maximisation (EM) for one label-set called EM1D. EM1D is a simple semi-supervised approach for learning a generative classification model from a small amount of training data and a large pool of unlabeled data. In Section 3.2.2 we extend this EM formulation based on one label-set to learn label-pair assignments to documents, one label from each label-set. This is our main algorithm for learning label-pairs and is called EM2D for our 2-d EM formulation. Between EM1D and EM2D, we propose a stratified version of EM1D in

Section 3.2.3 that uses A-labels in simpler ways than EM2D by setting up an EM1D instance for each $\alpha \in A$.

3.2.1 Expectation maximization (EM1D)

The NB classifier needs each training document to be marked with one label. Can we make use of additional documents with no label information (such as the test documents themselves or a pool of unlabeled documents) or partial label information (e.g., that a document was generated from one among a restricted subset of labels)?

A classic approach to estimating distributions over missing values is Expectation Maximization (EM) [DLR77]. Nigam et al. [NMTM00] use EM to induce a document classifier starting from a few labeled and many unlabeled documents (Figure 3.1). Because this algorithm is designed for only one label set, we will call it **EM1D**.

1: **Input:** Training and unlabeled set of documents
2: **Output:** EM model parameters using semi-supervised learning
3: Use labeled documents to induce a naive Bayes classifier with parameters Θ
4: **while** model Θ has not stabilized to satisfaction **do**
5: set up new model parameters Θ'
6: collect contributions from labeled documents to Θ'
7: **for** each unlabeled document d **do**
8: **E-step:** calculate the class probabilities $\Pr(c|d, \Theta)$ based on current parameters
9: **M-step:** if term t occurs $n(d, t)$ times in d, let d "contribute" a fractional term count of $\Pr(c|d)\, n(d, t)$ to the next estimate $\theta'_{c,t}$
10: **end for**
11: Re-estimate new cluster model parameters Θ'
12: $\Theta \leftarrow \Theta'$
13: **end while**

Figure 3.1: Using standard EM ("EM1D") for semi-supervised learning of document labels.

3.2.2 EM2D: Cross-trained naive-Bayes

EM1D is a simple classic semi-supervised formulation of EM in a classifier learning setting. EM1D does not have any notion of two label-sets. Our main motivation behind cross-training is to see whether we can learn better models for two taxonomies if they share some mappings and similarities between their respective topics.

We extend Nigam et al.'s EM algorithm to EM2D by creating a 2d grid of class labels taken from the product set $C = A \times B$. We assume a standard mixture model [DLR77] for document generation. First the label pair (c_A, c_B) is picked with probability $\Pr(c_A, c_B)$, and then a conditional term distribution $\Pr(d|c_A, c_B)$ is sampled to generate the document. Thus,

$$\Pr(d) \quad = \quad \Pr(c_A, c_B) \Pr(d|c_A, c_B). \tag{3.1}$$

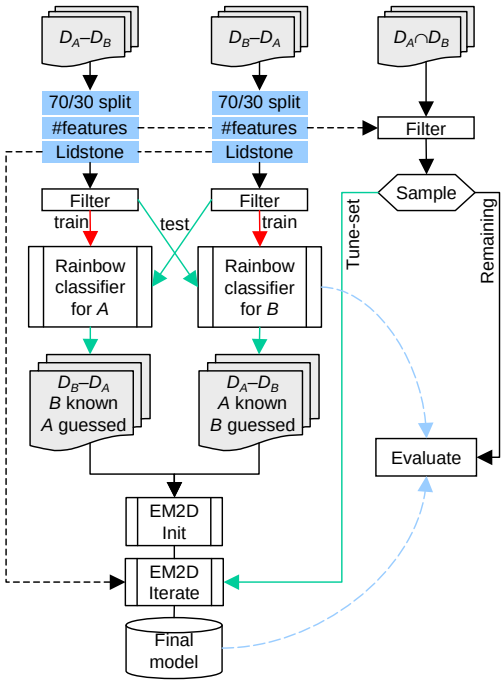


Figure 3.2: Design and evaluation of EM2D.

We assume the term distribution to be multinomial, extending parameters $\theta_{c,t}$ to

$\theta_{c_A,c_B,t}$. Likewise, parameters π_{c_A,c_B} express the prior probability of a document being generated from label-pair (c_A, c_B).

Thus, each document belongs to exactly one cell of this grid [1]. However, in the mapping scenario in Section 3.2, each training document comes with exactly one label, which determines either the row or the column where the training document belongs, but not both. Thus, each document d identifies a subset $C_d \subset C$ to which it potentially belongs, and for $\gamma \notin C_d$, we are given that $\Pr(\gamma|d) = 0$. We force this constraint in the E-step shown in Figure 3.1, limiting the contributions from a training document to its correct row or column, and scaling the E-variables to add up to 1 over the row or column.

Initialization

The EM algorithm [DLR77] guarantees only a locally optimum solution to the E and M variables. It is important to start the iterations from a reasonably good initial estimate of Θ. In EM2D, we have two resources at our disposal to achieve good initialization.

The first option is to train two naive Bayes classifiers to generate guessed labels. $D_A - D_B$ is the set of training documents for A which have A-labels but are not labeled with B-labels. Similarly, $D_B - D_A$ is the training corpus for B. Both these document sets and their corresponding label-sets are used in inducing two naive Bayes classifiers. These classifiers are used to *guess* B-labels for $D_A - D_B$ and guess A-labels for $D_B - D_A$. These basic naive Bayes classifiers are trained on a $70-30$ train-validation split. They help us choose an initial number of features in decreasing order of information gain and an initial value of the Lidstone parameter λ in Equation (2.2). These initial steps are shown near the top of Figure 3.2.

[1] It is possible that a document belongs to more than one class in a single taxonomy; handling such cases is left to future work.

The second option is to use the tune-set of fully-labeled documents to seed the initial Θ distribution. Fully labeled documents are our test documents which have A-labels as well as B-labels. These are denoted $D_A \cap D_B$. However, this intersection set is generally rather small. Using *only* the tune-set would generally fail to populate all the cells of the label grid adequately. It is probably best to use both options in the rare case that fully-labeled data is available.

Update rules

Suppose a training document d has $\alpha = c_A(d)$ known but $c_B(d)$ unknown. Then $\sum_{c_B} \Pr(\alpha, c_B|d, \Theta) = 1$. Using the standard multinomial model in Equation (2.1), we can write

$$\Pr(\alpha, \beta|d, \Theta) \;=\; \frac{\pi_{\alpha,\beta} \prod_{t \in d} \theta_{\alpha,\beta,t}^{n(d,t)}}{\sum_{\beta} \pi_{\alpha,\beta} \prod_{t \in d} \theta_{\alpha,\beta,t}^{n(d,t)}}. \tag{3.2}$$

This completes the specification of the E-step, although some care is required to preserve numerical precision. For the M-step, we set

$$\pi'_{\alpha,\beta} \;=\; \frac{1}{|D|} \left[\begin{array}{l} \sum_{d:c_A(d)=\alpha} \Pr(\alpha, \beta|d, \Theta) \\ +\quad \sum_{d:c_B(d)=\beta} \Pr(\alpha, \beta|d, \Theta) \end{array} \right], \tag{3.3}$$

which is simply the *expected fraction* of documents occupying label cell (α, β). Likewise, we set

$$\theta'_{\alpha,\beta,t} \;=\; \frac{\left[\begin{array}{l} \lambda + \sum_{d:c_A(d)=\alpha} n(d,t) \Pr(\alpha, \beta|d, \Theta) \\ \quad + \sum_{d:c_B(d)=\beta} n(d,t) \Pr(\alpha, \beta|d, \Theta) \end{array} \right]}{\sum_{\tau} \left[\begin{array}{l} \lambda + \sum_{d:c_A(d)=\alpha} n(d,\tau) \Pr(\alpha, \beta|d, \Theta) \\ \quad + \sum_{d:c_B(d)=\beta} n(d,\tau) \Pr(\alpha, \beta|d, \Theta) \end{array} \right]} \tag{3.4}$$

This expression closely resembles Equation (2.2), except that again, contributions to term counts are weighted by the probability of each document occupying label cell (α, β), like in step 7 of Figure 3.1.

Damping: In the EM2D setup, different documents have different quality and extent of label information. Tune-set documents plug into exactly one known (α, β) slot and presumably have the most reliable label information. Training documents have one label pinned by human input, which is assumed to be reliable, but the other label is not as reliable. In the zero-label setting, test documents have neither label known, but may still help the classifier gain accuracy by participating in the EM iterations "completely floating" over the label grid.

In the update equations above, we have given one vote to each document. However, it is common [NMTM00] to use a *damping factor* $L \leq 1$ to scale down the contribution of documents whose labels we consider less reliable. It is as though a fully-labeled document is worth one vote, but a singly labeled document is worth only $L = 0.5$, say. Thus, L can be thought of as an instance scaling mechanism like in boosting. It does not invalidate the theory of EM in any way. The best value of L can be set by cross-validation.

Early stopping: The NB generative model is a very crude approximation to reality. Therefore, maximizing data likelihood using EM may not improve classification accuracy in all cases. It is common to use a tune-set to stop EM iterations in case classification accuracy over (cross-) validation data is found to drop [NMTM00].

Deployment: The half-label setting is simple. Given a test document d with $c_A = \alpha$ known, we simply find $\Pr(\alpha, \beta|d)$ for all candidates $\beta \in B$, and report the best. The zero-label setting gives us at least two distinct options: EM2D with guesses and EM2D with model aggregation.

EM2D with guesses (EM2D-G): To classify to target taxonomy B, we first apply an A-classifier to the test document. The guessed A-label now lets us deal with

the zero-label test instance as if it were a mapping problem. Obviously, we should use the best possible A-classifier.

EM2D model aggregation (EM2D-D): After EM2D iterations are over, we use the final values of the E-variables to prepare a new classifier for target taxonomy B. More specifically, each training document $d \in D_A - D_B$ has associated E-values $\Pr(\beta|d, \alpha)$. Just like in EM, we let d "contribute" its term counts in proportion to this probability to label β. Documents in D_B contribute fully to their respective labels in B. The resulting "aggregated" classifier for B is used to classify test instances.

3.2.3 Stratified EM1D

If EM2D improves upon the accuracy of single-taxonomy learners, that could be attributed to multiple reasons. Let B be the target taxonomy in this discussion. The mapping of A-labeled documents to B may improve simply because of the extra documents in $D_A - D_B$, not because these documents are A-labeled. Whether this is the case can be easily determined by calibrating EM2D against EM1D (run with B as target labels) with the documents in $D_A - D_B$ thrown in as unlabeled documents.

Between EM1D and EM2D there are options which let us use the A-labels, but in ways simpler than EM2D. We set up an EM1D instance for each $\alpha \in A$. The B-labeled documents are shared across all such EM1D instances. A-labeled documents bearing the label α become unlabeled documents for the instance corresponding to α. The pseudo-code is shown in Figure 3.3. We call this **Stratified-EM** and this is very different from EM1D where there is just one label-set and one pool of unlabeled test documents.

If EM2D beats both EM1D and Stratified-EM, we can conclude that the mutual "corrections" of term distributions in EM2D are somehow vital to its higher accuracy.

1: **Input:** Training and unlabeled set of documents
2: **Output:** Per-class EM model parameters using unlabeled data
3: **for** each label $\alpha \in A$ **do**
4: train a B classifier Θ^{α} using EM1D with $D_B - D_A$ as the labeled set and $\{d \in D_A - D_B | c_A(d) = \alpha\}$ as the unlabeled set.
5: **end for**
6: **for** each test document labeled $(c_A, ?)$ **do**
7: use the EM1D model Θ^{c_A} to predict c_B
8: **end for**

Figure 3.3: Stratified EM to exploit A-labels while classifying for B.

3.3 Discriminative cross-training

The classifiers discussed thus far aim to fit a class-conditional generative distribution $\Pr(d|c)$ (or $\Pr(d|c_A, c_B)$), and use Bayes rule to estimate $\Pr(c|d)$ (or $\Pr(c_A|d, c_B)$ etc.). In contrast, discriminative classifiers seek to directly fit a regression function from the document to scores for label(s).

In this section we will discuss cross-training using two discriminative classifiers. The first new approach uses Support Vector Machines (SVMs), which have been reported to do well for text data [DPHS98]. This approach called SVM cross-training (denoted SVM-CT) is described in Section 3.3.1. The second existing approach [AS01] combines generative and discriminative aspects and is reviewed in Section 3.3.2.

3.3.1 SVM-CT: SVM-based cross-training

If A-labels are good predictors of B-labels, one way to enhance a purely text-based SVM learner for B is to allocate, over and above a column for each token in the training vocabulary, $|A|$ extra columns, one for each label in A. A document $d \in D_B - D_A$ is submitted to a text-based SVM ensemble for A, called $S(A, 0)$, which gives it a score $\mathbf{w}_{c_A} \cdot d + b_{c_A}$ for each class $c_A \in A$.

These scores can be inserted into the $|A|$ new columns, either as-is, or after some simple transformation, such as taking the sign of the score, or converting the largest score to $+1$ and the rest to 0 or -1 (we use the latter option in our experiments), and scaling ordinary term attributes by a factor of $f (0 \leq f \leq 1)$ and scaling these *label attributes* by a factor of $1 - f$. Document vectors are always scaled to unit L_2 norm.

The parameter f, which can be chosen through cross-validation on a tune-set, decides the relative importance of label and term attributes in the SVM kernel evaluations. We evaluated f from 0 to 1 in steps of 0.05 and set $f = 0.95$. These cross-trained SVMs are denoted by **SVM-CT**.

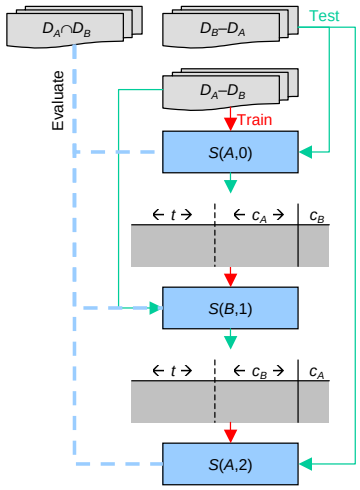


Figure 3.4: Data flow diagram for cross-training SVMs.

Documents in $D_B - D_A$ thus get a new vector representation with $|T| + |A|$ columns where $|T|$ is the number of term features. They also have a supervised B label. These are now used to train a new SVM ensemble $S(B, 1)$. The document tables and how they are used to train and test SVMs are shown in Figure 3.4. We can obviously repeat the process iteratively in a ping-pong manner, each classifier providing synthetic columns for the other. The complete pseudo-code is shown in Figure 3.5.

Our experiments show that SVM-CT does outperform SVM, making effective use

1: **Input:** Training data sets for label-sets A and B
2: **Output:** Iteratively cross-trained SVM models for A and B
3: Represent each document as a vector d in term space and $\|d\| = 1$
4: Build one-vs-rest SVM classifiers $S(A, 0)$ and $S(B, 0)$ for $D_A - D_B$ and $D_B - D_A$ using text tokens only
5: **for** $i = 1, 2, \ldots$ **do**
6: **for** each document $d \in D_B - D_A$ **do**
7: Apply $S(A, i-1)$ to d, getting a vector $\gamma_A(d)$ of $|A|$ scores (see text)
8: Concatenate vectors d and $\gamma_A(d)$ into a single training vector with label $c_B(d)$, with relative term-label weight determined by f and maintaining $\|d\| = 1$
9: Add this vector into the training set for a one-vs-rest SVM classifier $S(B, i)$
10: **end for**
11: Similarly, use $S(B, i-1)$ to get $\gamma_B(d)$ and induce a new one-vs-rest SVM classifier $S(A, i)$ for all $d \in D_A - D_B$
12: **end for**

Figure 3.5: Cross-training SVMs.

of the label attributes (although there is little improvement beyond the first ping-pong round). SVM-CT is also better than the generative cross-training methods in about half the cases. SVM (which uses text alone), in turn, is much better than the baseline NB classifier. Moreover, inspecting the components of $\mathbf{w}$ along the label dimensions derived by SVM-CT gives us some interesting insights into various kinds of mappings between the label sets A and B. We will return to these observations in Section 3.4.

3.3.2 The A&S mapping algorithm

Agrawal and Srikant (A&S) [AS01] proposed a hybrid generative/discriminative classification algorithm by enhancing the prior estimation of NB in Equation (2.1). Let the target label-set be C and the source label set be S (to be consistent with their notation). In the mapping setting, classifying document d entails finding $\arg\max_c \Pr(c|d, s)$, where the source label $s \in S$ is supplied and the target label c is sought.

Given d and s are fixed, $\arg\max_c \Pr(c|d, s) = \arg\max_c \Pr(c|s)\underline{\Pr(d|c, s)}$, which

A&S approximate as $\Pr(c|s)\underline{\Pr(d|c)}$, using the conditional independence assumption shown underlined (which is theoretically debatable, but seems to work in practice). All that remains is to propose parametric forms for $\Pr(c|s)$ and $\Pr(d|c)$. $\Pr(d|c)$ is modeled exactly as in Equation (2.1), i.e., $\Pr(d|c) \propto \prod_{t \in d} \theta_{c,t}^{n(d,t)}$. All $\theta_{c,t}$ are pre-estimated by a C-trained classifier which has no knowledge of S-labels. (This is the generative part.)

The key innovation of A&S is to propose a parametric form for $\Pr(c|s)$ depending on inter-label relations. Let N_c be the number of C-labeled documents in the training set for C. As in EM2D, A&S use a C-trained classifier to guess classes of S-labeled documents; let $G(s,c)$ be the number of documents with source label s that this classifier assigns to target label c. The overall score uses a tuning parameter $R \geq 0$ and is given by

$$\begin{aligned} \Pr(c|d,s) &\propto N_c \; G(s,c)^R \prod_{t \in d} \theta_{c,t}^{n(d,t)}, \quad \text{or} \qquad (3.5) \\ \log \Pr(c|d,s) &= \text{constant} + \log N_c + R \log G(s,c) + \\ &\qquad \sum_{t \in d} n(d,t) \log \theta_{c,t}. \end{aligned}$$

Note that (once the $\theta_{c,t}$s are fixed) R is the only tunable parameter here. $R = 0$ coincides with standard NB on the master labels. Taking logs, we see that (like SVM) A&S is also a linear discriminant learner. A&S use a tune-set to set the best value of R, which can be chosen in two ways.

Random sampling: A fraction (varying between 10% and 90% in our experiments) of the fully-labeled documents is sampled to create a tuneset. The remaining documents are used as the test set. A range of choices for $R \in \{0, 1, 3, 10, 30, 100, \ldots\}$ is evaluated against the tuneset. Average of the accuracy is reported over dozens of such samples.

Active learning: The system repeatedly samples the fully-labeled documents. For each sample d, it varies R to see if R makes any difference to the estimated C-label. If it does, d is placed in the tune-set; otherwise it is put in the test/calibration set. A&S report that 5–10 actively chosen samples are adequate to pick a suitable R.

3.4 Experiments

In this section we present experiments performed in a variety of settings for testing our cross-training algorithms. We used the 5 web-crawled datasets described in Section 2.4.2 for the main experiments with EM2D and SVM-CT. All our algorithms were coded in a few thousand lines of simple C++. A&S, EM2D, and variants were run on 1.3GHz Pentium3 servers with 1–3GB of RAM. The models fit easily in tens of megabytes of RAM. We scanned the documents sequentially and did not need to hold document vectors in memory. The SVM implementation we used did load document vectors into memory. A&S, EM2D and its variants generally trained faster than SVM and SVM-CT.

In Section 3.4.1 we see the baseline naive Bayes accuracy for the 5 datasets used. All further accuracy numbers are with respect to naive Bayes's accuracy. In Section 3.4.2 we compare NB and SVMs with the cross-trained variation SVM-CT; we also see some interesting mappings learned between classes across taxonomies. In Section 3.4.3 we review the performance of EM2D against NB and the 1D EM variants presented in Section 3.2. We see the sensitivity of the initial guesses in EM2D in Section 3.4.3 and look at performance in asymmetric scenarios in Section 3.4.3. We compare our methods to A&S in Section 3.4.3 and present results with trying to classify zero-labeled documents in Section 3.4.3.

3.4.1 The naive Bayes baseline

We used naive Bayes (NB) classifier in the Rainbow package [McC98]. We created two Rainbow classifiers, one for A labels using $D_A - D_B$, the other for B labels using $D_B - D_A$.

Apart from providing a strawman, NB runs are used to set the Lidstone parameters and the feature sets for A and B. Consider the classifier for A. We first created a random 70%/30% train-test split of $D_A - D_B$. Rainbow ingested the 70% training subset and listed features in decreasing order of information gain (w.r.t. the labels). In an outer loop, we chose from λ_A between 0.1 and 1 in steps of 0.1. In an inner loop, we chose a prefix V_A of the feature list of size 10% through 90% in steps of 10% (similarly for B). We then used the 30% validation data to pick the best values for $\lambda_A, \lambda_B, V_A, V_B$. Finally, the NB baseline is obtained by subjecting the held-out $D_A \cap D_B$ to these optimized Rainbow classifiers. Table 3.1 shows various accuracy statistics.

All other generative cross-training algorithms used these optimized values of λ and V. In particular, EM2D used $V_A \cup V_B$ as the feature set, and the average of λ_A and λ_B.

Feature selection and the choice of λ matters a great deal for most data sets. Given high-dimensional data like text, feature selection would likely be helpful for any learning method, but the benefit from tuning λ is large mainly because the naive Bayes model results in terrible estimates of the joint distribution, and any "fix" to the innumerable θs is likely to help. Whereas the two classifiers can each optimize V_A, V_B, λ_A and λ_B in an unconstrained manner, EM2D is stuck with a single feature set and a single value of λ, which puts it at a disadvantage.

Table 3.1: Naive Bayes baseline accuracy with optimized choices of $|V|$, the number of features, and λ, the smoothing parameter. A is Dmoz and B is Yahoo!. Percent accuracy is shown for 70/30 cross validation and the unseen $D_A \cap D_B$ test set.

Data set		$\lvert V\rvert$	λ	70/30	$D_A \cap D_B$
Autos	A	10000	0.1	39.3	46.5
	B	10000	0.2	59.2	65.6
Movies	A	8385	0.5	44.6	43.0
	B	64434	0.2	50.9	41.0
Outdoors	A	2142	0.2	79.8	77.1
	B	813	0.5	68.0	78.0
Photo	A	27969	0.5	68.7	40.9
	B	325	1.0	49.6	35.5
Software	A	40000	0.1	40.0	47.8
	B	17000	0.1	58.4	54.3

3.4.2 Performance of SVM-CT

We used SVMLight [Joa99] in one-vs-rest ensemble mode, with a linear kernel and default settings for all parameters. Documents were represented as unit vectors and f was set to 0.95 as explained in Section 3.3.1.

Figure 3.6 compares the accuracy of SVM and SVM-CT with the NB baseline. In most cases, SVM beats NB. This is consistent with folk wisdom that SVMs generally perform better than NB on text classification tasks. More interesting is the observation that SVM-CT has higher accuracy than SVM, which shows that it is possible for SVM-CT to exploit additional information from label-derived columns.

We made two additional studies of SVM-CT. First, we checked that the average magnitude of $\mathbf{w}$ for ordinary term features was always lower than the average magnitude of $\mathbf{w}$ for label-derived features. Recall that $|\mathbf{w}_t|$ is a measure of how strongly

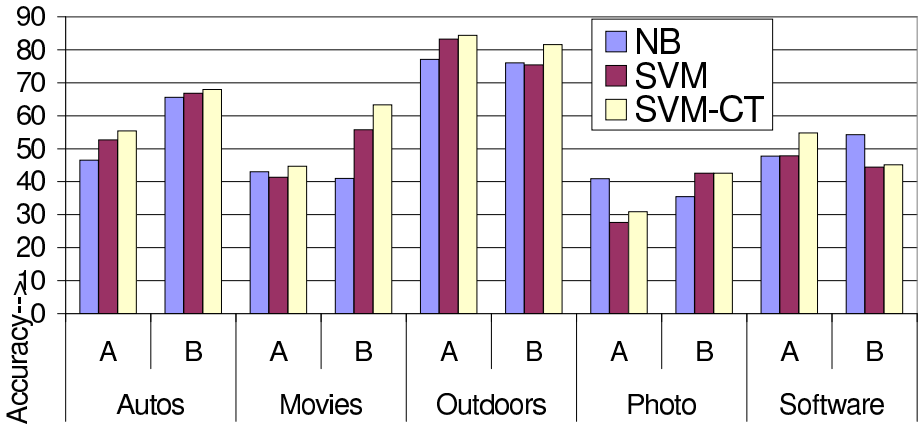


Figure 3.6: Comparative evaluation of NB, SVM and SVM-CT (cross-trained SVM).

the feature t can influence the decision of the SVM, i.e., the sensitivity of the SVM to feature t.

Second, we tabulated the B (respectively, A) labels corresponding to the highest and lowest $\mathbf{w}$ values of various A (respectively, B) classifiers. We wanted to observe the mappings learned between the classes in the two taxonomies using cross-trained SVMs [2]. During cross-training, the label information was transformed into a vector of 1 and -1 values as mentioned in Section 3.3.1. In addition, a new dimension called none-of-the-above (NOTA) was introduced, whose value was set to 1 when all label scores obtained from B (respectively, A) were negative and all label dimensions were set to -1. The purpose of NOTA is explained shortly.

The results are shown in Table 3.2. We show some Dmoz (respectively, Yahoo!) class labels along with the Yahoo! (respectively, Dmoz) class labels which had the greatest positive and negative influence in predicting the said Dmoz (respectively, Yahoo!) class. All positive couplings are very meaningful; some negative couplings are fairly intriguing too.

The `Outdoors` dataset for both taxonomies contains the class `ScubaDiving` which

[2]See `http://www.it.iitb.ac.in/~shantanu/ctdemo/` for examples

Table 3.2: Dmoz and Yahoo topic mappings learned with cross-trained SVMs

Dataset	`Dmoz.`	Maps to `Yahoo.`	Weight
Autos	News&Magazines	News&Media	0.147
		Volkswagen	−0.156
Movies	Genres/Western	Titles/Western	0.242
		Titles/Horror	−0.052
Outdoors	Scuba Diving	Scuba	5.878
		Snowmobiling	−0.647
Photo	Techs&Styles	Pinhole Ph'graphy	2.796
		3D	0.964
		Panoramic	0.921
		Organizations	−1.184
Software	Accounting	NOTA	0.156
		Screen Savers	0.103
		OS/Unix	−0.171
Dataset	`Yahoo.`	Maps to `Dmoz.`	Weight
Autos	Corvette	Chevrolet	0.981
		Parts&Accessories	−0.266
Movies	SciFi&Fantasy	Series/Star-Wars	1.123
		Reviews	−0.824
Outdoors	Scuba	Scuba Diving	4.822
		Wildlife	−0.437
Photo	Pinhole Ph'graphy	Techs&Styles	0.4842
		Photographers	−0.270
Software	OS/MSWindows	OS/MSWindows	0.018
		NOTA	−0.001
		OS/Unix	−0.008

maps to it's namesake class in the other taxonomy with a large positive component along $\mathbf{w}$. Such one-to-one mappings are symmetric and expected. Even when there is no direct one-to-one correspondence between the labels, or there is a containment relationship, as between `Yahoo.Movies.Genres.SciFi-Fantasy` and `Dmoz.Movies.Series.StarWars`, SVM-CT seems capable of extracting that information. On the other hand when the `Dmoz.Software.Accounting` class really has no relevant class in the Yahoo! taxonomy, the synthetic NOTA class indicates this with a high $|\mathbf{w}_{\text{NOTA}}|$.

One interesting case is the mapping of `Dmoz.Photo.Techniques&Styles`. `Yahoo.Photo`, on the other hand contains separate classes for each technique like Pinhole Photography, 3D Photography, Panoramic Photography, etc. The Dmoz to Yahoo! mapping in this case gives high positive weights to most of these *child* classes as seen in the Figure. This *parent-child*, or *one-to-many* mapping emerges in-spite of our assumption of flat taxonomies and is instructive.

Another interesting mapping is from `Yahoo.Software.OS.MSWindows` to multiple high positive weights to classes in the Dmoz taxonomy. Here, the NOTA class can be interpreted as clearly separating all the Windows related classes above it from the Unix related classes below it within `Dmoz.Software`.

Many such mappings are further reported visually in the Appendix of this thesis. For each of the 5 datasets, we have given part of the graph generated by keeping classes in A and B as nodes and mapped weights between them as directed edges. Only edges greater than 0.5 are retained and interesting parts of the graph are are shown. The full graphs and an interactive demonstration can be viewed at `http://www.it.iitb.ac.in/~shantanu/ctdemo/`.

3.4.3 Performance of EM2D

EM2D for mapping

Figure 3.7 shows the accuracy of EM2D in comparison with NB. EM2D is significantly better than NB with a maximum gap of 30% for the Movies dataset and average gap of 10%. This is reassuring, but in this section we wish to analyze carefully *why* this is the case.

There are two potential sources of information from $D_A - D_B$ which may improve the accuracy of classifying into B given d and c_A. The first is simply the addition of a bunch of documents, even if they are not labeled with B-labels and even if we ignore

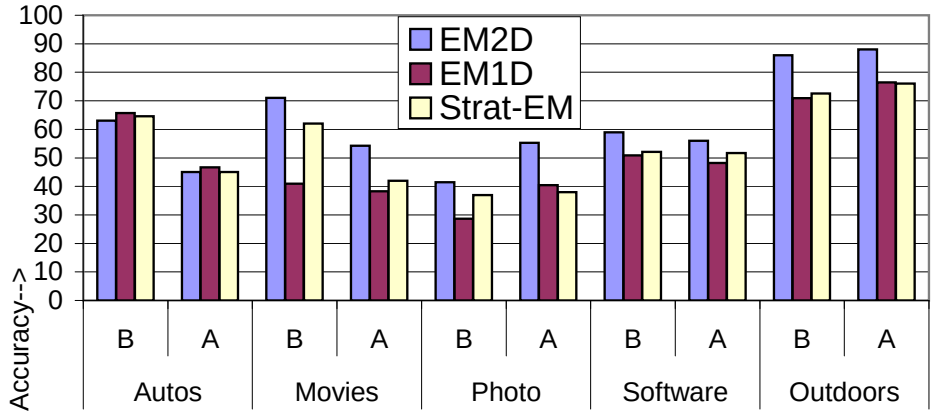


Figure 3.7: EM2D vis-a-vis Stratified-EM1D, EM1D, and NB. For EM1D we used its best damping parameter, $L = 0.01$.

their A-labels. If this were the only source of extra information, EM1D should be able to match EM2D, which is clearly not the case. Therefore, knowledge of A-labels of specific documents is vital.

As we discussed in Section 3.2.3, A-labels can be used by Stratified-EM, which simply creates one instance of EM1D for each distinct label $\alpha \in A$. Figure 3.7 also shows that with only two minor exceptions, EM2D beats both EM1D and Stratified-EM. This despite the fact that each EM1D has denser data, lowering the variance of the parameter estimates compared to EM2D. These measurements help us establish that

- There is information available in cross-training which EM1D cannot exploit, and
- A relatively straight-forward extension to EM1D, Stratified-EM, does not work as well as EM2D.

Sensitivity to initial guesses of labels

EM2D, like EM1D, finds locally optimum values of the total data likelihood. Hence the final accuracy is sensitive to the initial assignment of half-labeled data in the 2D label grid.

Given the baseline classifiers trained on A (using documents in $D_A - D_B$) and B (using documents in $D_B - D_A$), it is natural to initialize EM2D by submitting documents in $D_A - D_B$ to the B-classifier and vice versa. We may use a "hard" assignment to the best guess, or a "soft" or fractional assignment based on the probabilities emitted by the baseline classifiers.

However, these are not the only options. How will EM2D behave if each document in $D_A - D_B$ is assigned *uniformly* over each label in B? In general, how sensitive is EM2D to perturbations and errors in the initial E-estimates?

To test EM2D's resilience, we randomly picked a fraction q of documents (with A-labels, say) and replaced their guessed scores for B-labels with a uniform distribution smeared over all B-labels. The remaining fraction $1 - q$ of documents are added to the EM2D system as before. Thus, $q = 1$ corresponds to full uniform assignment.

Obviously, the effect of smearing a fraction q depends on the accuracy of the guesses in the first place. Therefore we repeat the smearing experiments for varying level of starting guess accuracy. (We fake different guess accuracies by random flips in guesses. Note that these "flips" are distinct from the "smear.")

In Figure 3.8 we show the change in accuracy with increasing fraction of smeared guesses on the Movie dataset, with two different settings of guess accuracy: the default as shown in Table 3.1 and a second setting where 70% of the guesses have been pre-flipped to a random label (this results in guesses of very poor quality). These plots show that

- When the guesses are reasonably accurate, uniform assignment is worse than assignment based on guessed probabilities, which makes eminent sense.
- EM2D can handle limited ($q = 0.15$ to $q = 0.20$ for this data) smearing, beyond which accuracy starts to drop.
- When the accuracy of guesses is too poor, smearing a fraction of the guesses

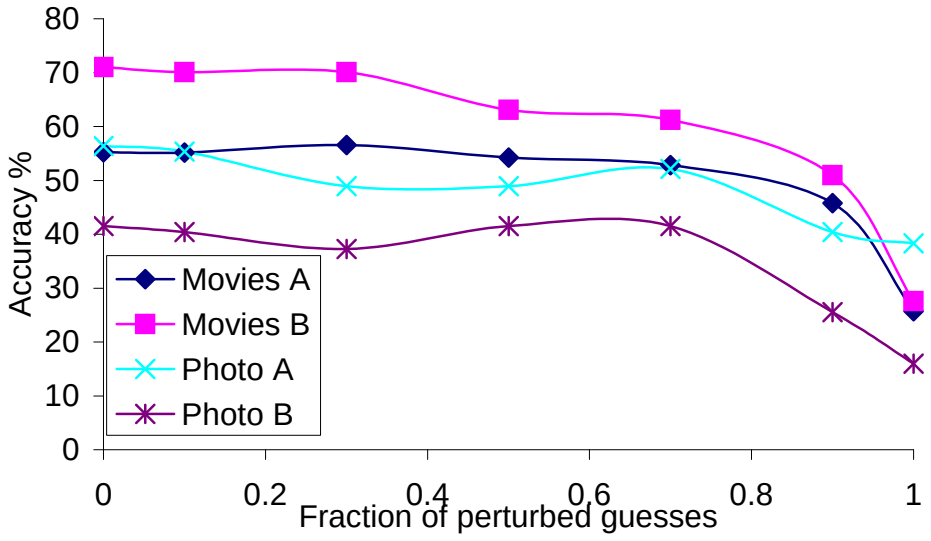


Figure 3.8: The effect on EM2D of smearing the initial guesses of a fraction of half-labeled training documents. The y-axis shows final EM2D accuracy.

($q = 0.10$ in this case) can improve accuracy. This was somewhat unexpected, and made sense only in hindsight.

Highly asymmetric scenarios

All the data sets we collected have relatively balanced sizes of $D_A - D_B$ and $D_B - D_A$. How well can EM2D do in highly unbalanced settings, especially w.r.t. the sparsely-populated taxonomy?

To answer this question, we (arbitrarily) picked B as the taxonomy to be decimated, and sampled $D_B - D_A$ down to 300 documents. (Actually, we decimated to 200, 300, and 5% of the original. Results were similar.) $D_A - D_B$ was left unchanged.

The small size of $D_B - D_A$ led to a poor baseline B-classifier. Therefore, the guessed B-labels for the documents in $D_A - D_B$ had a large error rate. Because information flow is bidirectional in EM2D, poor B guesses reduced overall accuracy. We propose three fixes for this problem:

- Taking our cue from Figure 3.8, we smear documents in $D_A - D_B$ over all B-labels. Documents in $D_B - D_A$ continue to use the A guesses.
- Like A&S, we use a tune-set sampled from $D_A \cap D_B$. Specifically, we sampled 5% of fully-labeled documents.
- We set the damping factor L (Section 3.2.2) so as to restore the relative weights of $D_A - D_B$ and $D_B - D_A$ to the same ratio as in the original dataset. This would mean $L \approx 0.05$ on documents in $D_A - D_B$.

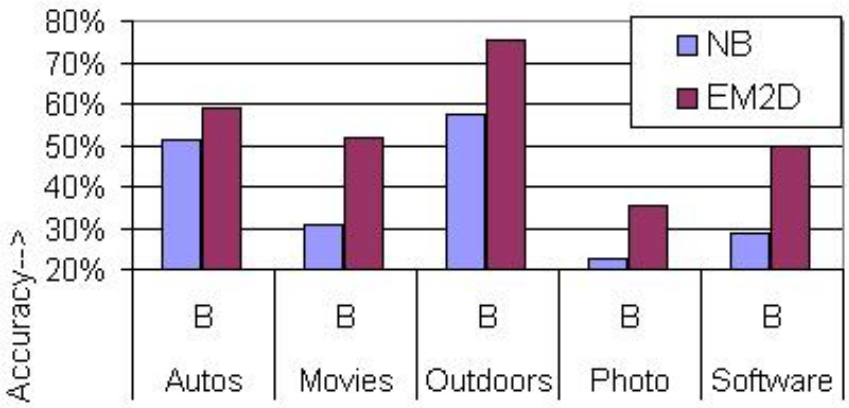


Figure 3.9: EM2D on a small sample of 300 documents from $D_B - D_A$.

Figure 3.9 shows that the accuracy gain of EM2D over NB in highly asymmetric settings can in fact be higher (average 11.4%) than in more balanced data (average 10% in Figure 3.7), provided EM2D is initialized properly. Nigam et al.'s experience [NMTM00] seems to corroborate that the gains from semi-supervised learning are larger when labeled data is limited.

Comparison with the A&S algorithm

For our A&S implementation, we fixed the feature set to $T_A \cup T_B$ as found by Rainbow, and also fixed the λ parameter to one that gave the best accuracy for Rainbow for each of A and B prediction.

Making a fair comparison between A&S and EM2D involves exposing to EM2D at least the fully-labeled tune-set that A&S uses. In fact, it is very difficult to compare the active-learning version of A&S with EM2D in a principled way, because A&S inspects fully-labeled documents (not the labels, but the text) outside the tune-set as well. (EM2D is not designed for active learning.) Therefore, we focused on the randomly sampled tune-set paradigm only, because that could be used with both A&S and EM2D.

In addition, given the large skew between half-labeled and fully-labeled populations, we used damping to re-scale them to the same effective size (see Section 3.2.2 for more details).

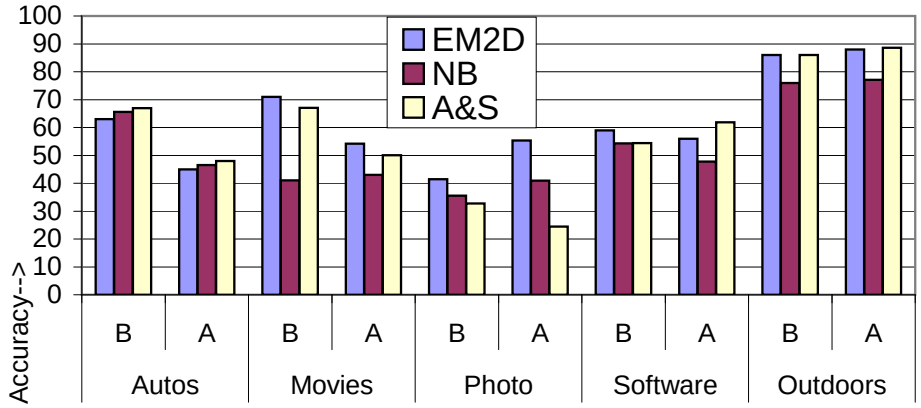


Figure 3.10: Accuracy of the A&S algorithm compared with EM2D for 10% and 90% tuneset (T) and A&S active learning (AL).

In Figure 3.10 we present the accuracy of A&S with 10% and 90% randomly sampled tuneset as well as a tuneset of size 10 picked by active learning (AL) from the entire test set of fully-labeled documents. Broadly, A&S and EM2D are comparable, but EM2D edges over A&S by a maximum of 20% and an average of 4% for the 10% tuneset and 2% for the 90% tuneset. When EM2D loses to A&S, the gap is very small.

EM2D for classifying zero-label documents

We used a custom crawled Bookmarks dataset specifically for the experimental setting explained here. The Bookmarks data set was created mainly to study zero-labeled classification. We collected and inspected a dozen or so bookmark files published on the Web. We found it very common for bookmark authors to collect URLs into coherent topics. Usually, these topics had strong correspondence with one or few topics in Yahoo!/Dmoz. However, the number of URLs per topic was small (say 3–20), exactly the scenario we painted at the outset. We derived sample bookmark topics B from these bookmark collections, and populated them from Yahoo! (A) URLs, and removed them from D_A. The dataset had 47247 documents in A with 154 classes and 365 documents in B with 7 classes with an intersection size of 1289.

In this scenario, we are required to finally produce a classifier for B which does not depend on the test instances being labeled with A labels. In Section 3.2 we discussed two methods for deploying EM2D in this setting: EM2D-D, a model aggregation method, and EM2D-G which is essentially EM2D, except that the A-labels are supplied as guesses from an A-classifier.

Figure 3.11 shows the accuracy for EM2D-D, EM2D-G, EM1D and NB, for various sizes of labeled training sets, and two choices of the damping factor L discussed at the end of Section 3.2. These numbers are for the Bookmark data set. The accuracy values were averaged over three random choices of the training set for each choice of training set size.

As the fraction of training data is increased, the benefit of semi-supervised learning reduces, which is obvious. The damping factor does essential damage control when there are many labeled documents, but can hurt when the labeled set is very small. These observations corroborate with earlier EM1D results by Nigam et al. [NMTM00].

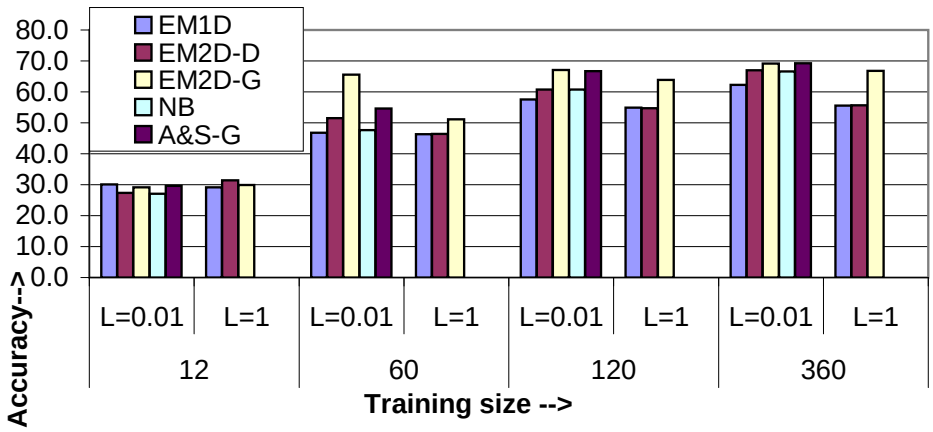


Figure 3.11: EM2D with guessing is the best methods for classifying zero-label documents. NB accuracy is shown only once for each size of the training set, because it does not change with L.

Unlike Nigam et al., EM1D could improve beyond NB only for the smallest training sets in our case. One possible reason is that the unlabeled Yahoo! dataset, from which EM1D adds instances, is significantly different, and has many more irrelevant classes, compared to the initial labeled data in our Bookmark dataset. Nigam et al.'s experiments drew unlabeled and labeled documents from the same distribution.

Finally, we were surprised to see that EM2D-G performed better than EM2D-D. Recall that EM2D-D is really a 1d classifier, which should reduce data sparsity and improve the reliability of its parameter estimates compared to EM2D. Despite this benefit, model aggregation appears to hurt. Even a noisy guess at the A-label, followed by a row-conditioned classification, outperforms the aggregated model.

3.5 Related work

In recent years, EM-like semi-supervised learning has been enhanced in several ways and applied to a number of settings. Extending beyond EM, Liu et al. [LLYL02] and Yu et al. [YHC02] consider the realistic situation where, apart from labeling only

a few samples, the user is also unlikely to spend the effort to mark negative samples. Their EM-like algorithms can work on a set of positive examples P and a mixed pool of samples M which may contain both positive and negative instances.

Cross-training is related to *multi-task* learning or *life-long* learning, in which information (features, models, etc.) from one learning task is used for another. Thrun [TO96] discusses clustering learning tasks (not instances) and picking, for a given task, other related tasks. This information influences the distance function in a nearest-neighbor classifier. Caruana [Car97] discusses how to use multi-task learning in neural networks, and Baxter [Bax00] provides a PAC analysis. Cross-training is a two-task setting with no instance submitted to more than one task. The similarity between tasks falls out naturally as we estimate $\pi_{\alpha,\beta}$.

A recent approach to semi-supervised learning (which might appear superficially similar to cross-training) is *co-training*, proposed by Blum and Mitchell [BM98]. In co-training, two learners use *disjoint* subsets of attributes, and assign labels from only *one* taxonomy. Each learner picks unlabeled training instances that it is most confident about classifying correctly, and makes it a labeled training instance for the other learner. Co-training and cross-training are quite different things: two label sets are central to our formulation, and our approach depends on modeling a single term distribution conditioned on a pair of labels.

Independently, Zhang et al. [ZL04] have recently proposed the co-bootstrapping framework for integrating taxonomies and they also present mappings between the source and master taxonomies. Like SVM-CT their approach also uses clues in the source taxonomy to enhance classification in the master taxonomy, but their experiments use only URL and description information from publicly available directory sites unlike our methods which also take the full text of the pages into account. Rajan et al. [RPG05] have very recently proposed a maximum likelihood approach for

integrating taxonomies. They exploit the hierarchical structure of taxonomies unlike our assumption of flat label-sets and obtain natural mappings between the source and master taxonomies. They additionally insert source classes in appropriate places in the master taxonomies, even creating new classes when required.

The Glue system [DMDH02] is also similarly motivated to cross-training in the semantic web domain task of mapping ontologies. The approach they follow is to find the most similar class in the related ontology by aggregating classification decisions of one set of classes on documents of the other. Our approach is different and more complex than this setting.

3.6 Discussion

Aggregating information about placement of documents in other taxonomies is seen to be useful in training better classifiers for the taxonomy at hand. For generative methods like EM2D, this model aggregation significantly helps in training because class conditional word statistics are now drawn from a richer source of information. For similar taxonomies this results in more stable and smooth parameter estimates from larger training data. EM2D is built upon NB classifiers which are the simplest generative classifier. We would like to investigate the application of cross-training to other generative models, specifically the newer generative models like LDA and BayesANIL.

Discriminative methods benefit only slightly from cross-training. SVMs are already high accuracy learners and it is difficult to further boost their accuracy, yet modest improvements do result. The other interesting result of discriminative cross-training is the discovery of *mappings* between taxonomies. The Appendix, Chapter 8.1, shows extensive examples of mappings learned between parts of our Dmoz and Yahoo datasets

used in Section 3.4.2. Both these results can be understood by seeing that the new label dimensions we add during cross-training have high signal compared to particular term features when strong mappings between source and target classes exist.

It is known that adding noisy redundant features has a regularising effect on linear classifiers [SD99]. The label dimensions added in the case of closely related mappings between label-sets is one form of adding features that are redundant and noisy with respect to the target class label. This could be leading to a regularising effect on learning the label-sets. This phenomenon needs further theoretical study and investigation.

3.7 Summary

In this chapter, we explored the notion of *mappings* across label-sets in a general semi-supervised framework called cross-training. We leveraged noisy, incomplete mappings between label-sets to improve classification in both generative and discriminative settings. Through a detailed experimental study using real-life and semi-synthetic data from Yahoo! and DMOZ, we show that for both discriminative and generative models, cross-training is decisively better than the best baseline classifier we could induce on either of the label-sets alone.

More reassuring are the observations that show our approach to compare favorably with the best existing approach, while providing a more sound foundation. Inspecting the components of learned hyper-planes in SVM-CT along the label dimensions gives us some interesting insights into various kinds of mappings [SCG03, GS04] between the label-sets A and B. Detailed examples are provided in the Appendix. These mappings could prove useful for ontology maintenance tools.

Chapter 4

Scaling multi-class classification problems

4.1 Introduction

In the previous chapter we looked at exploiting mappings between sets of classes. In this chapter we propose the second relationship between classes, that of confusion within a label-set. We develop the notion of confusion to deal with large datasets in the form of hierarchies. Our main contribution is an algorithm based on the notion of confusion that helps in training multi-class classifiers with significant speedups without loss in accuracy.

Handling very large data collections and issues of scale have become very important in the context of the web. Text classification systems have also matured to the point that their large scale deployment is now possible. The availability of very large controlled data corpora like RCV1 (described in Section 2.4.2) has further sparked interest in large scale text classification systems. Web directories contain a few million high quality manually classified pages and these are a ripe target for training and deploying large scale systems. The scalability and efficiency of classifiers at this scale remains an important research area.

An important phenomenon we come across when dealing with large amounts of data or a very large number of classes is that of *confusion*. Sparse training data, obscure notions of classes and insufficient class separation, all lead to loss in accuracy of learning algorithms due to documents being classified into wrong though related classes. Confusion is the common phenomenon of mis-classification into related classes. This is captured very well in a confusion matrix that plots the true versus predicted classes of instances [God02] in a quickly understandable summary. Confusion occurs when the classifier cannot distinguish the true target class from a very similar adversary and mis-classifies the instance into the other class.

We focus on this notion of confusion in this chapter. We next look at exploiting confusion to create hierarchies as a natural answer to the problem of organising large amounts of data with many classes. We also look at problems associated with training multi-class classification systems with many classes.

4.1.1 Hierarchical methods

One way to mitigate the effect of confusion in large text collections has been to organize classes into hierarchies. The hope in such a hierarchical organization is that different features will be active at different parts of the hierarchy, and it should be possible to build high performance classifiers using this *is-a* relationship between classes. A typical hierarchy like Dmoz has first level children comprising of broad ranging topics like *Sports*, *Recreation*, *Science*, and *Shopping*. Within *Sports*, the sub-classes could be *Football*, *Hockey*, and *Volleyball*. BOW classifiers depend upon features (textual tokens, n-grams, phrases) like *game*, *player*, and *referee* to distinguish between *Sports* and other classes at the top level. Within *Sports*, these features lose discriminating power since all sub-classes will equally likely contain these words. Other features like *batsman*, *volley*, or *stick* will now become important for discrimination at

the second level in the hierarchy.

Confusion between classes in a classification setting is the result of either obscure or insufficient specification of the label-set or falls out of an insufficient feature selection and representation mechanism. The occurrence of certain keywords in documents of classes does not always follow our intuitive notion of separation between classes. The bag-of-words model typically employed to represent documents further leads to overlapping concept clouds in the term-dimensional vector space. Such *confusion* between concepts leads to erroneous classification not only in computer systems, but also for humans; controlled studies involving humans classifying content show as high as 30% disagreement between reviewers [LYRL04]. Confusion results in loss of accuracy and it's effect can be easily seen in a multi-class consufion matrix. We exploit this notion of confusion for construction and deployment of hierarchies of classes in Section 4.2. Experimental results show the problem of multiplication of errors across levels in hierarchical classification in Section 4.2.2.

4.1.2 Non-hierarchical methods

Support Vector Machines (SVMs) [Vap95] are a kind of *discriminative* classifier which have shown superb performance for classifying text and other data. They are accurate, robust, and quick to apply to test instances. We have seen a brief overview of support vector machines used in text classification in Section 2.3.2. The elegant theory behind the use of large-margin hyperplanes to separate two classes cannot be extended easily to separate N mutually exclusive classes. The most popular approach to multi-class classification is the one-vs-others approach, the other approaches being pair-wise decomposition, error correcting output codes, and DAGSVM, all reviewed earlier. All these decomposition approaches require training an ensemble of classifiers and training every classifier takes time quadratic to the number of training instances.

For very large datasets this presents a scaling up problem.

Generative classifiers, in contrast, are essentially independent of the number of classes as far as training time is concerned. A popular generative classifier for text like naive Bayes trivially scales with the number of classes as they process each document only once. Also, NB classifiers train much faster than SVMs owing to their extreme simplicity, but in terms of accuracy, the linear SVM has decisively outperformed NB owing to the latter's high bias in assuming attribute independence.

We would like to devise a method that achieves the best of both worlds: scalability of NB classifiers w.r.t. number of classes and accuracy of SVMs. We do this in two stages. In the first stage we use the fast multi-class NB classifier to compute a confusion matrix, which is used to reduce the number and complexity of two-class SVMs that are built in the second stage using the one-vs-others approach. During testing, we first get the prediction of a NB classifier and use that to selectively apply only a subset of the two-class SVMs, as indicated by the confusion matrix. On standard benchmarks, our algorithm is 3 to 6 times faster than multi-class SVMs, and has superior scalability in terms of memory requirements and training set size. In terms of accuracy, the method is better than NB classifiers and comparable or superior to SVMs.

Our main contribution in this chapter is the *GraphSVM* algorithm that efficiently scales up training of high-accuracy multi-class classifiers in Section 4.3. In Section 4.4 we see that this algorithm turns out to be competitive or even better than the normal multi-class classifiers [GSC02] in terms of accuracy while being significantly better in terms of training time and memory requirements. We review related work in Section 4.5 and summarise our work in Section 4.7.

4.2 Automatic construction of hierarchies for large-scale data organisation

In this section we look at topic hierarchies in detail. We look at methods of automatically constructing topic hierarchies based on classifiability properties of the data at hand. We use a confusion matrix of a NB classifier to generate a dendrogram of classes and use this dendrogram to create a simple two level hierarchy. We study the greedy hierarchical classification process of such a hierarchy and note some interesting results.

The genesis of our approach lies in the class relationships derived from a confusion matrix, easily generated from a fast classifier like NB. This can be obtained using a held-out validation dataset. For example, in Figure 4.1 we show an example of a confusion matrix built on the 20-newsgroup dataset (details in Section 2.4.2) using an NB classifier. The rows show actual classes and the columns show predicted classes.

Classname		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
alt.atheism	1	251	6	1	3	32	1	1	2	1	2	0	0	0	0	0	0	0	0	0	0
soc.religion.christian	2	9	277	0	1	6	0	0	1	0	0	0	0	0	1	2	2	0	0	0	1
sci.space	3	3	1	273	1	0	1	2	0	1	1	9	0	0	1	2	3	0	0	1	1
talk.politics.misc	4	2	0	3	213	24	3	0	17	3	0	0	0	0	0	0	1	0	1	33	0
talk.religion.misc	5	88	36	2	23	132	0	1	0	0	0	0	0	0	0	0	2	0	1	15	0
rec.autos	6	0	0	0	3	1	272	0	0	0	7	1	2	1	6	4	1	0	0	2	0
comp.windows.x	7	1	1	2	1	0	1	246	0	2	2	30	5	3	1	1	2	1	1	0	0
talk.politics.mideast	8	0	3	1	18	0	0	0	275	0	1	0	0	0	0	0	0	0	1	1	0
sci.crypt	9	1	0	1	2	1	0	3	0	284	0	3	0	1	0	0	1	0	0	3	0
rec.motorcycles	10	0	0	0	1	0	4	1	0	0	286	1	2	0	1	2	1	0	0	1	0
comp.graphics	11	0	1	2	1	1	0	10	1	2	0	243	23	7	3	3	3	0	0	0	0
comp.sys.ibm.pc.hardware	12	0	0	0	0	0	2	7	0	1	0	5	243	23	12	3	1	3	0	0	0
comp.sys.mac.hardware	13	0	0	1	1	0	2	1	0	0	0	7	10	260	8	9	1	0	0	0	0
sci.electronics	14	1	0	1	0	1	5	2	0	2	0	7	13	13	245	6	3	0	1	0	0
misc.forsale	15	0	1	4	2	0	12	1	0	0	4	1	19	10	8	233	1	0	1	1	2
sci.med	16	0	1	5	0	1	1	0	0	0	1	2	0	2	7	2	275	0	1	1	1
comp.os.mswindows.misc	17	1	0	2	0	1	1	58	1	3	0	38	71	17	3	6	0	97	1	0	0
rec.sport.baseball	18	2	1	1	0	0	0	0	0	0	0	4	0	0	0	1	1	0	282	1	7
talk.politics.guns	19	0	0	0	9	5	1	0	0	1	0	0	0	0	1	0	0	1	1	281	0
rec.sport.hockey	20	0	1	0	0	0	1	0	0	0	2	0	0	1	1	0	0	0	3	0	291

Figure 4.1: 20-newsgroups confusion matrix

The matrix clearly shows that different classes have different degrees of confusion with other classes. Some classes, like `rec.sport.hockey`, are well separated from the rest, whereas others like `comp.os.ms-windows.misc` are easily confused with others.

The mis-classifications of a class are usually limited to a small subset of classes. In fact, in most cases, the rows and columns of a matrix can be rearranged manually as shown in Figure 4.2 so as to reveal clusters of classes that confuse with each other. These appear as blocks along the diagonal of the confusion matrix. Not surprisingly, in many cases, these clusters are formed of classes whose names can be immediately recognized as forming natural hierarchies. The confusion matrix provides a domain-independent method of deriving this relationship.

Classname		1	2	5	19	8	4	11	17	12	13	7	15	14	6	10	18	20	9	16	3
alt.atheism	1	251	6	32	0	2	3	0	0	0	0	1	0	0	1	2	0	0	1	0	1
soc.religion.christian	2	9	277	6	0	1	1	0	0	0	0	0	2	1	0	0	0	1	0	2	0
talk.religion.misc	5	88	36	132	15	0	23	0	0	0	0	1	0	0	0	0	1	0	0	2	2
talk.politics.guns	19	0	0	5	281	0	9	0	1	0	0	0	0	1	1	0	1	0	1	0	0
talk.politics.mideast	8	0	3	0	1	275	18	0	0	0	0	0	0	0	0	1	1	0	0	0	1
talk.politics.misc	4	2	0	24	33	17	213	0	0	0	0	0	0	0	3	0	1	0	3	1	3
comp.graphics	11	0	1	1	0	1	1	243	0	23	7	10	3	3	0	0	0	0	2	3	2
comp.os.mswindows.misc	17	1	0	1	0	1	0	38	97	71	17	58	6	3	1	0	1	0	3	0	2
comp.sys.ibm.pc.hardware	12	0	0	0	0	0	0	5	3	243	23	7	3	12	2	0	0	0	1	1	0
comp.sys.mac.hardware	13	0	0	0	0	0	1	7	0	10	260	1	9	8	2	0	0	0	0	1	1
comp.windows.x	7	1	1	0	0	0	1	30	1	5	3	246	1	1	1	2	1	0	2	2	2
misc.forsale	15	0	1	0	1	0	2	1	0	19	10	1	233	8	12	4	1	2	0	1	4
sci.electronics	14	1	0	1	0	0	0	7	0	13	13	2	6	245	5	0	1	0	2	3	1
rec.autos	6	0	0	1	2	0	3	1	0	2	1	0	4	6	272	7	0	0	0	1	0
rec.motorcycles	10	0	0	0	1	0	1	1	0	2	0	1	2	1	4	286	0	0	0	1	0
rec.sport.baseball	18	2	1	0	1	0	0	4	0	0	0	0	1	0	0	0	282	7	0	1	1
rec.sport.hockey	20	0	1	0	0	0	0	0	0	0	1	0	0	1	1	2	3	291	0	0	0
sci.crypt	9	1	0	1	3	0	2	3	0	0	1	3	0	0	0	0	0	0	284	1	1
sci.med	16	0	1	1	1	0	0	2	0	0	2	0	2	7	1	1	1	1	0	275	5
sci.space	3	3	1	0	1	0	1	9	0	0	0	2	2	1	1	1	0	1	1	3	273

Figure 4.2: 20-newsgroups re-organized confusion matrix.

We would like to automate this re-organization of classes into clusters of similar classes [GSC02]. We want to automatically generate topic hierarchies from this given flat organisation of classes. Each class is represented by a row in the confusion matrix. For each class, it's respective row is converted to a normalized N dimensional vector that denotes how much the class confuses with other classes. We then use a distance measure like the Euclidean L_n or the KL-distance measure to compute distance between the classes. These distances are used to cluster the classes using a hierarchical agglomerative clustering (HAC) algorithm. The output of HAC is a dendrogram that we analyze to determine the clusters that provide the maximum inter-class separation.

The dendrogram for the confusion matrix at hand is given in Figure 4.3.

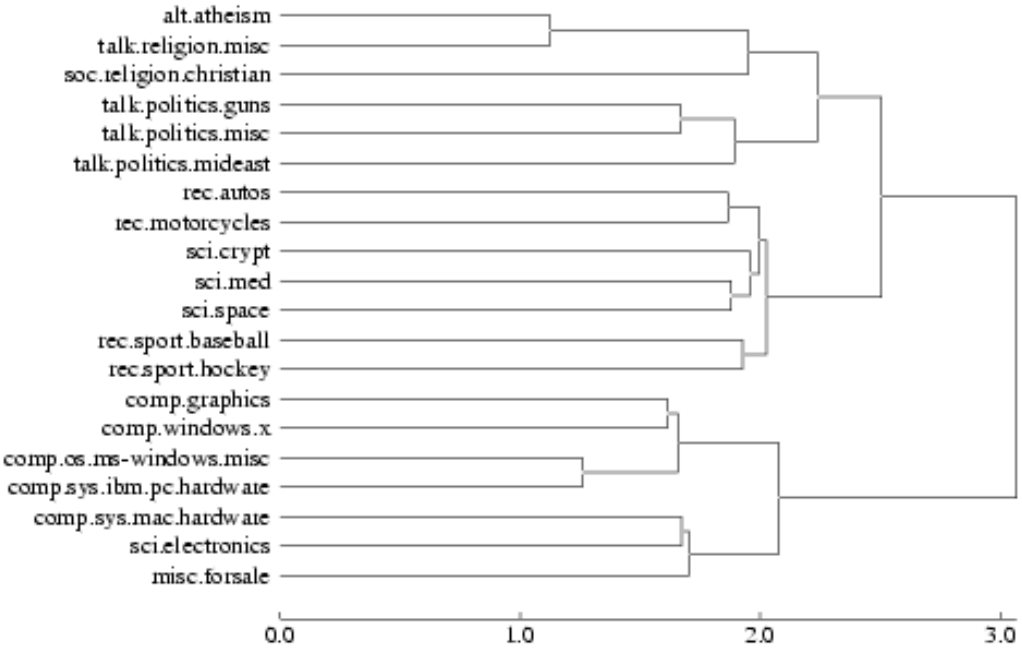


Figure 4.3: Dendrogram for 20-Newsgroups

Note that from the root of the dendrogram, as we proceed to the left of the figure beyond a base distance of 1.0, the number of clusters merging per unit distance change drastically. We can plot the distances at which clusters merge against the merge numbers for the dendrograms of each dataset. Cluster merge distances for the 20-newsgroups, Reuters, and ODP datasets are given in Figure 4.7.

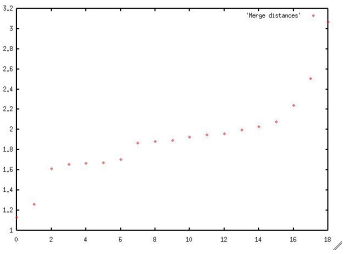


Figure 4.4: 20-newsgroups

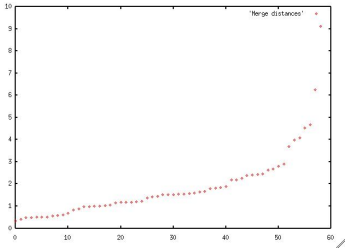


Figure 4.5: Reuters-21578

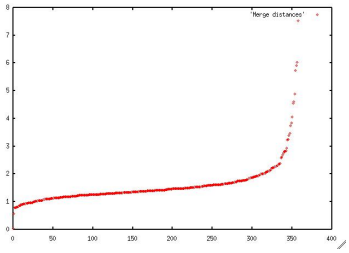


Figure 4.6: Dmoz dataset

Figure 4.7: Merge distances for the datasets plotted against cluster merge number

We clearly note a point of inflection in the cluster merge distance plots. Beyond this

point of inflection, there are a very small number of highly dissimilar clusters. This can be explained by the nature of Ward's algorithm for hierarchical agglomerative clustering. Consider some vector space with an underlying pattern of clusters. Very similar clusters are consolidated first as the algorithm proceeds. Toward the end of the HAC run, we will reach a stage when huge clusters are left to be merged but the distance between them is very large. This indicates that these clusters are not similar to each other. This analogy can be applied to our clustering of classes in confusion space. After the point of inflection, only highly dissimilar groups of classes are left to be merged.

For the ODP dataset, we see this point to occur at merge number 335. Since there are 359 classes, we are left with 22 top level clusters of classes largely dissimilar to each other. Similarly, the 20-newsgroups dataset gives us 5 groups and the Reuters dataset gives us 8 groups. These are the groups we use for the multi-level classifiers detailed in Section 4.2.1. We provide sample dendrograms in the appendix. For the 20-newsgroups dataset, we see from the manually arranged Figure 4.2 and the automatically generated clustering using Figure 4.3 that generated clusters in both assignments are very similar. This automated method is thus a method for automatically generating two-level hierarchies from the confusion matrix. We will use these constructed two-level hierarchies for classification next.

4.2.1 Hierarchical multi-class classification

We propose to exploit the clustering of classes to prune the number and complexity of two-class classifiers needed for a leaf-level one-vs-others SVM ensemble. An obvious approach is to arrange the clusters in a two-level tree hierarchy and train a classifier at each internal node. We have already seen above how such a two-level hierarchy can be automatically constructed. If we restrict to NB classifiers, Mitchell [Mit98] has

shown that if the same feature space were used for all the classifiers and no smoothing is done, the accuracy would be the *same* as that of a flat classifier. In practice, each classifier has to deal with a more easily separable problem, and can use a independently optimized feature set; this should lead to slight improvements in accuracy, apart from the gain in training and testing speed. We propose to use a combination of NB and SVM classifiers at the two levels.

We first build a top-level classifier to discriminate amongst the top-level clusters of labels, called the Level 1 (L1) classifier. This top-level classifier could either be a NB or SVM classifier. Even with SVM, the training time will be smaller since the number of classes is reduced, although each two-class SVM will still need all documents. At the second-level (L2) we build multi-class SVMs within each cluster of classes. The total number of SVMs at the second level will be close to N but the number of classes per SVM is significantly reduced.

Each L2 classifier can concentrate on a smaller set of classes that confuse with each other. For a generative classifier like NB, we expect this to result in better feature selection and thus enable finer distinctions amongst the confusing classes [CDAR98]. For SVMs, the spread of the negative ('others') class is reduced, which we expect will make separability easier and/or better.

4.2.2 Evaluation

We evaluate the accuracy and training time for solving the multi-class problem using a two-level hierarchy. We compare four methods: Flat multi-class NB classifiers (MCNB), flat multi-class SVMs (MCSVM) using the one-vs-others approach, NB classifiers at both the levels (L1 and L2) of the hierarchy (Hier-NB) and NB classifier at L1 followed by SVMs at L2 (Hier-SVM).

We notice from Figure 4.8 and Figure 4.9 that while the training times for Hier-

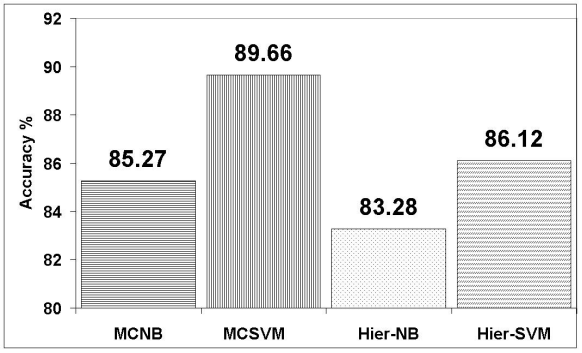


Figure 4.8: Accuracy of different hierarchical methods for 20NG

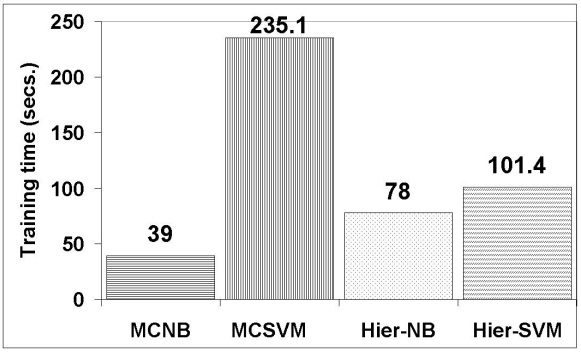


Figure 4.9: Training time of different hierarchical methods for 20NG

Table 4.1: Level-wise comparisons for 20newsgroups data

Method	Accuracy (in %)
MCNB	85.27
MCSVM	89.66
NB-L1	93.56
SVM-L1	95.39
NB-L2 with NB-L1	89.01
NB-L2 with Perfect-L1	88.41
SVM-L2 with NB-L1	92.04
SVM-L2 with Perfect-L1	91.65

NB and Hier-SVM are lower than for MCSVM, the accuracy is also reduced. The accuracy for Hier-NB is even lower than that of MCNB. Hier-SVM has lower accuracy than MCSVM though it is slightly better than MCNB. Also, training time of Hier-SVM is less than half that of MCSVM.

We show in Table 4.1, a comparison of accuracy of the two levels separately for both NB and SVM classifiers. For L2, we show two kinds of accuracy, first on documents that get correctly classified to their correct group by L1, and second the absolute accuracy of L2 assuming a perfect L1 classifier. As expected, all the L1 and L2

classifiers are individually more accurate than the original flat classifier, as noted in [CDAR98] and as we expected in Section 4.2.1. Even though NB-L2 has an accuracy of 89.01%, combining with the NB-L1 accuracy of 93.56% leaves us with a resultant accuracy of Hier-NB of 83.28%. Although NB-L1 and NB-L2 are both individually better than MCNB (85.27%), we see that compounding of classification errors leaves Hier-NB worse off than MCNB. Similarly, SVM-L2 with a NB classifier at L1 (Hier-SVM) has an accuracy of 92.04%. The accuracy of Hier-SVM still drops down to 86.12% which is slightly better than MCNB, but is worse than the MCSVM accuracy of 89.66%. If we replace NB-L1 with SVM-L1 having 95.39% accuracy, the overall accuracy for both Hier-NB and Hier-SVM improves slightly, but the training time gets worse.

Classification results on such hierarchies (Pachinko machine [KS97]) against a flat classifier built on leaf nodes have proved inconclusive except in special cases. Hierarchies suffer from multiplication of errors at each level as classification proceeds down. There is no conclusive comparison in [CDAR98, KS97] which decisively states whether a hierarchical classification scheme is better than a flat one for NB classifiers. These previous studies have either restricted the number of features at each node in the hierarchy or have tried to equalize the number across the flat and hierarchical schemes. It is not clear whether by attempting to equalize the number of features, one of the classification schemes is getting compromised.

The main reason for the low accuracy of the hierarchical approaches is the compounding of errors at various levels. Even though the accuracy at both levels is higher than that of a flat classifier, the product of their accuracies falls short of the accuracy of a flat classifier. Increasing the levels beyond two is expected to worsen this compounding effect. This led us to design a new algorithm *GraphSVM*, that attempts to ensure that the first-stage classification will make the overall process fast, but in-

accuracy of the first-stage classifier will not jeopardize overall accuracy. We describe *GraphSVM* in the next section.

4.3 Efficiently training multi-class classifiers in non-hierarchical setups

In this section, we represent class confusion in a general way using a graph, which may connect a class with any other class, instead of restricting confusion to a hierarchy of disjoint groups as in the previous approach.

As in the previous approach, we start with the confusion matrix obtained by a fast multi-class NB classifier M_1. For each class i, we find the set of classes $F(i)$ such that more than a threshold (t) percentage of documents from each class in $F(i)$ gets mis-classified as class i. For example, for the confusion matrix in Figure 4.1, we find that for the class `alt.atheism` and with a threshold of 3%, $F($`alt.atheism`$) =$ {`talk.religion.misc`, `soc.religion.christian`}.

Next, for each node i with non-empty $F(i)$, we train a multi-class classifier $M_2(i)$ to distinguish amongst the classes in $\{i\} \cup F(i)$. These classifiers are constructed using a more accurate and possibly slower method like SVMs. During testing, we first classify a document d using M_1. If the predicted class for d is i, we feed it to $M_2(i)$, if any, and get a refined prediction j. We denote this method *GraphSVM*.

In the above example, when a test instance is predicted as `alt.atheism` by M_1, we get the prediction refined by a one-vs-others SVM, $M_2(i)$, between the classes `alt.atheism`, `talk.religion.misc` and `soc.religion.christian`. The prediction of $M_2(i)$ is returned as the final answer for the test instance.

4.4 Experiments

Our aim in creating *GraphSVM* was to combine the speed of NB with the accuracy of SVMs. *GraphSVM* builds on NB but we hope it will perform better in terms of accuracy while being significantly more efficient than training full-fledged SVM ensembles. We compare *GraphSVM* to multi-class NB (MCNB), multi-class SVMs (MCSVM) and the hierarchical approach with a NB classifier at L1 and SVMs at L2 (Hier-SVM). We compare the algorithm on accuracy, training time and scalability with respect to size of the training set.

For the experiments in this section we used the 20-newsgroups dataset described in Section 2.4.2 and the Reuters-21578 dataset described in Section 2.4.2. All experiments were performed on a 1.4GHz P4 machine with 512MB RAM, running Linux. *Rainbow* [1] was used for feature selection, text processing and experiments involving NB classifiers. *SVMLight* [2] was used for all experiments involving SVMs.

The ODP dataset: Additional experiments were also performed on a very large custom crawl of the DMOZ directory in the year 2000. This crawl contained more than 350 classes and $180,000$ documents. This was a noisy dataset with lots of junk pages crawled from the internet; there were about $650,000$ unique features in this dataset, though only a few thousand were used after feature selection. The appendix of this thesis gives some parts of the dendrograms we derived using the methods outlined in this chapter. Dendrograms for the ODP dataset as well as the 20-newsgroup and Reuters-21578 datasets are given in the appendix.

[1] `http://www.cs.cmu.edu/~mccallum/bow/`
[2] `http://svmlight.joachims.org/`

4.4.1 Accuracy and training time

In Figures 4.10 and 4.11 we show the accuracy and training time for the four methods on the 20ng and Reuters datasets.

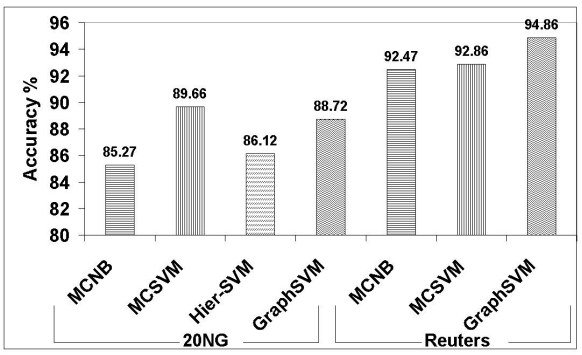


Figure 4.10: Accuracy comparison for all methods for both datasets

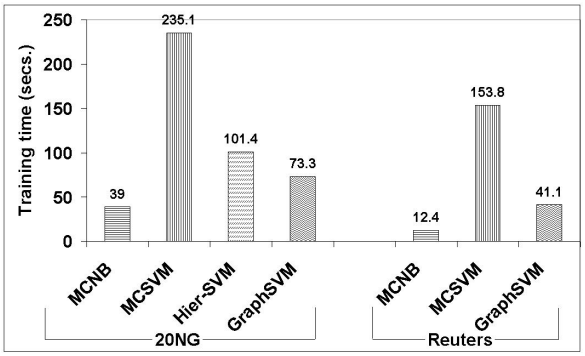


Figure 4.11: Training time comparison for all methods for both datasets

In Figure 4.10, for the 20NG dataset, we observe that the accuracy of *GraphSVM* at 88.72% is slightly smaller than the MCSVM accuracy of 89.66%. The accuracy of *GraphSVM* is higher than the previous Hier-SVM accuracy of 86.12%. For the Reuters dataset, *GraphSVM* has the highest accuracy of 94.86% while MCSVM and MCNB have accuracies of 92.86% and 92.47% respectively.

From Figure 4.11 we observe that *GraphSVM* has the fastest training time among the approaches involving SVMs, and for both the datasets, is more than a factor of 3 faster than MCSVM. As expected, MCSVM is the slowest to train.

4.4.2 Scalability with number of classes

We evaluated the training time of the different approaches with increasing number of classes. We started with 5 randomly picked classes, and added 5 randomly picked classes at a time for both the datasets. In Figures 4.12 and 4.13 we observe that the gap between the training time of *GraphSVM* and MCSVM increases as the number of

classes is increased.

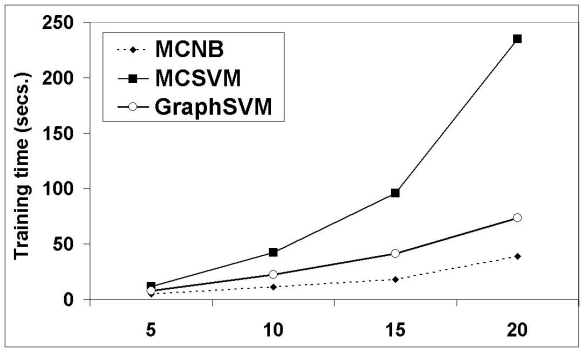


Figure 4.12: Training time vs. Number of classes for 20NG

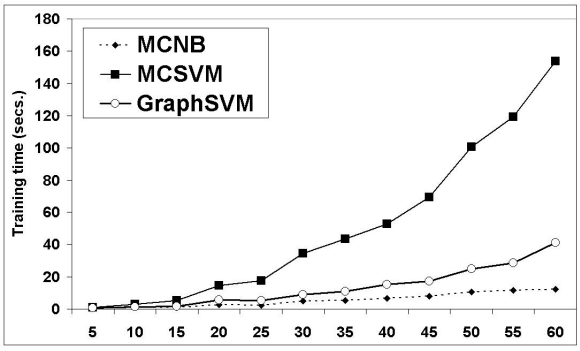


Figure 4.13: Training time vs. Number of classes for Reuters

Figures 4.14 and 4.15 show that in all cases *GraphSVM* continues to maintain the high accuracy vis-a-vis MCSVM and MCNB. For the 20NG dataset (Figure 4.14), *GraphSVM* maintains an accuracy of within 1% of MCSVM and for the Reuters dataset (Figure 4.15) *GraphSVM* is on an average, 3% better than MCSVM.

We notice a large dip in Figure 4.15 for MCNB and MCSVM. The Reuters dataset is highly skewed in the distribution of instances per class. Figure 4.15 additionally shows the total number of test instances over which the micro-averaged accuracy values are reported. For 25 classes there are only 166 test instances. The 7% difference between *GraphSVM* and MCNB and MCSVM is due to only 11 additional instances correctly classified by *GraphSVM*. Since these 25 classes are thinly populated, most of the mis-classifications seen in the confusion matrix are larger than the threshold and contribute to edges in the *GraphSVM* algorithm. These mis-classifications are corrected by the more focused SVMs in the *GraphSVM* method. The graph smoothens out after 30 classes when there are a larger number of instances and the accuracy of *GraphSVM* is consistently better.

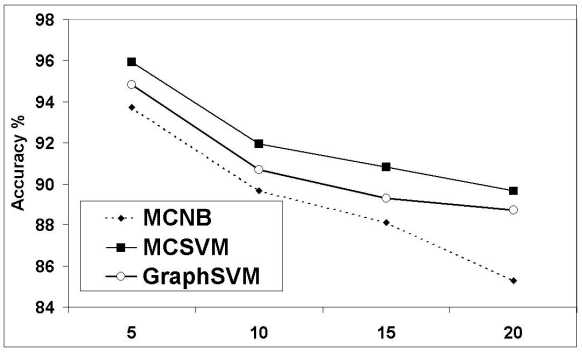


Figure 4.14: Accuracy vs. Number of classes for 20NG

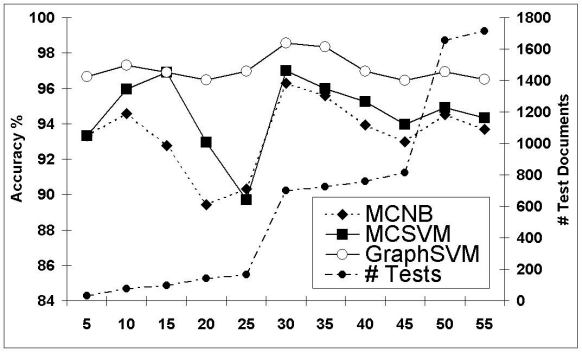


Figure 4.15: Accuracy vs. Number of classes for Reuters

4.4.3 Scalability with training set size

Training time: In Figure 4.16, the size of the training set was varied from 10% to 70% of the whole data, while keeping the relative train-test ratio constant at 70:30. We observe that the training time of *GraphSVM* is nearly linear in the training set sizes, while for multi-class SVMs the training time increased super-linearly with training set size. This causes the gap between the two methods to become more prominent for larger datasets.

Accuracy: In Figure 4.16 we show the corresponding accuracy values against varying percentages of training set sizes for 20NG. We observe that as the accuracy of MCSVM increases with increasing number of training instances and *GraphSVM* closely tracks the increase and is always more accurate than MCNB.

Maximum memory: In Figure 4.16 the percentage of training documents is plotted against the maximum memory required to train any SVM model in the *GraphSVM* and MCSVM approaches. In both cases, multiple one-vs-others SVMs are learned, but the size and heterogeneity of the negative ('others') class varies largely leading to different memory requirements. In MCSVM, this negative class contains the entire dataset

apart from the positive class, whereas it is greatly pruned in the *GraphSVM* approach. We notice that *GraphSVM* requires less than one-fourth the memory required by MCSVM.

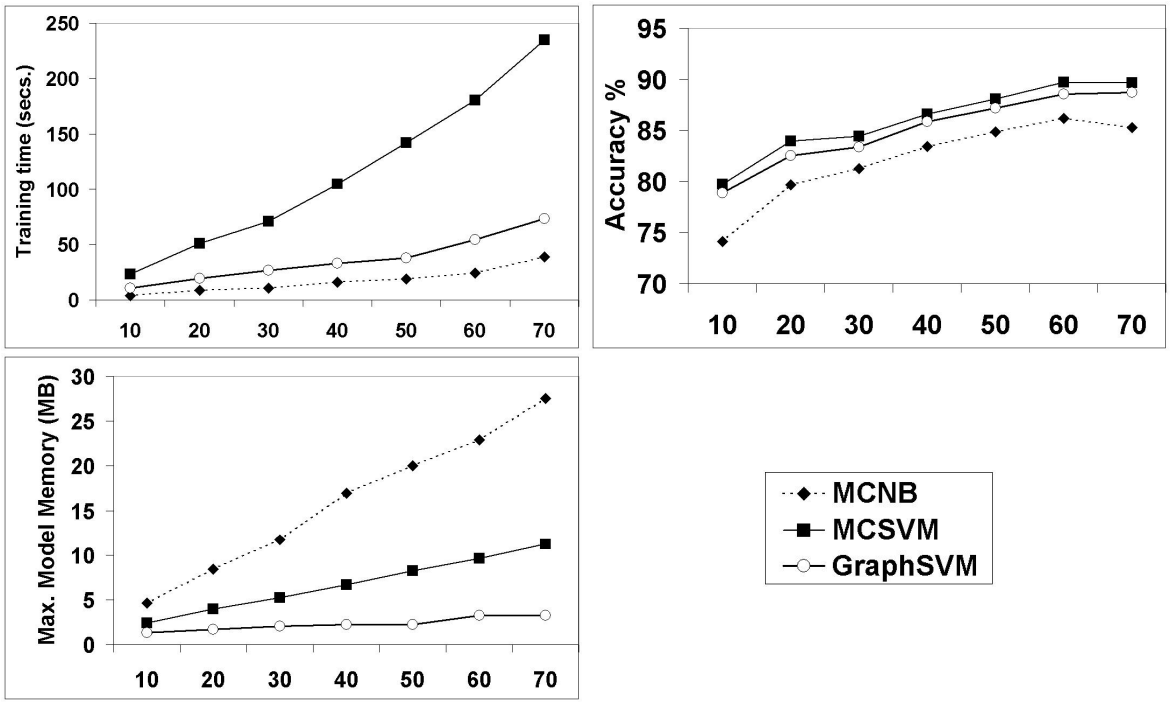


Figure 4.16: Training time, accuracy and maximum model memory with varying training set sizes for the 20NG dataset

4.4.4 Effect of the threshold parameter

An important parameter of *GraphSVM* is the threshold used to decide what SVMs to create in the second stage.

In Table 4.2 we show the accuracy and training time for different values of the threshold (t). We see that a threshold of 3% to 7% is appropriate for both these datasets because of the base accuracy of the NB classifier chosen to get the confusion matrix.

If the threshold value is kept unnecessarily high, we will hardly have any graph to construct, assuming the base classifier has a decent accuracy. On the other hand, even

Table 4.2: Accuracy and Performance

Method	Threshold t	Training time (secs)	Accuracy in %
Reuters			
MCSVM	-	158	92.86
GraphSVM	0.07	36	94.91
GraphSVM	0.05	41	94.86
GraphSVM	0.03	46	94.86
GraphSVM	0.01	65	95.33
20NG			
MCSVM	-	235	89.66
GraphSVM	0.07	63	87.92
GraphSVM	0.05	73	88.72
GraphSVM	0.03	73	88.52

for a moderately accurate base classifier, a very low threshold (say 2% or less) will make a densely connected graph. In that case, *GraphSVM* will approach MCSVM.

As seen in Section 4.4.1 and Section 4.4.2, *GraphSVM* is more accurate than MCSVM and MCNB for the Reuters dataset and also requires less training time. The scalability results for the Reuters dataset w.r.t. training set size are the same as that for the 20NG dataset in Section 4.4.3, viz. *GraphSVM* is highly scalable w.r.t. training time, accuracy and memory requirements as compared to MCSVM. The detailed results are omitted here for space constraints.

4.5 Related work

A general framework for solving multi-class problems using a collection of two-class problems is to associate it with a coding matrix M where each class is a row

and each two-class classifier is a column [DB95, ASS00]. Each element a_{ij} is $+1$,-1 or 0, where $+1$ denotes the class i serving as a positive class in classifier j, -1 denotes i serving as a negative class in j and 0 denotes that class i does not participate in classifier j. The coding matrix for one-vs-others will have N columns with $+1$ on the diagonal and -1 in all other entries. For Max-Wins [Kre99], the coding matrix will have $\binom{N}{2}$ columns and each column will have a single $+1$ and -1 and the rest of the entries 0. Error Correcting Output Codes (ECOC) classifiers are proposed in [DB95] where the $+1$s and -1s are chosen in such a way that the different rows are maximally separated. During testing, the outcome of each classifier is treated as a vector and compared with each row. The row to which it is closest is returned as the predicted class. Ghani [Gha00] reports experiments with a number of ECOC classifiers to solve large multi-class text classification problems using NB. They report significant improvement in accuracy with the ECOC method on the Industry section dataset. Rifkin et al. [RR01] repeated similar experiment with Support Vector Machines as the base classifier. On the standard text benchmarks, they found that for SVMs the various ECOC classifiers did not provide any accuracy improvement over the simple one-vs-others method. Platt et al [PCST00] present a modification of the testing procedure of the Max-Wins algorithm which reduces the number of kernel evaluations during testing. They arrange the various two-class classifiers in a DAG structure (DAGSVM), that is used to order the application of various two-class SVMs during testing. This method does not affect the training time and the accuracy is comparable.

4.6 Discussion

GraphSVM partitions a classification task between NB and SVMs such that SVMs are only invoked on small subsets of classes that get mis-classified by the NB classifiers.

We can claim that *GraphSVM* will not be worse than MCNB provided that the training data is representative of the test data. *GraphSVM* can choose not to use any second stage SVM refinements in the rare case where the SVM classifier is found to be worse than a NB classifier on a validation dataset. Compared to MCSVM, the main reason *GraphSVM* may be worse is high positive values of the threshold t. We can always decrease the threshold, at the expense of increasing the training time, to match MCSVM accuracy as shown in Section 4.4.4. In most cases, we expect the strengths of NB and SVMs to combine to give performance better than both individually.

The main property that we rely on for accuracy is that the set of classes that a class confuses with, using a NB classifier, should be the same for the training and test set. The relative distribution of the confusion matrix is not required to remain unchanged, provided entries that were previously below the threshold t do not suddenly increase beyond it.

The benefit of *GraphSVM* will be greatest when the mis-classification of the first stage are spread across a small number of classes. The worst case is when mis-classifications of a class are uniformly distributed over many classes. In this case, the algorithm will reduce to multi-class SVMs. In most practical datasets that is rarely the case.

4.7 Summary

We have described *GraphSVM*, an effective framework for extending discriminative classifiers like SVMs to handle data with a large number of classes, accurately and efficiently. *GraphSVM* estimates a measure of affinity between classes which depends upon the severity of confusion between these classes by a fast classifier like NB. This confusion indicates a clustering of classes which is used to limit the number and

complexity of the SVMs that need to be trained for multi-class categorization.

GraphSVM beats the accuracy of multi-class NB decisively even though it builds on a NB classifier. *GraphSVM* outperforms SVMs w.r.t. training time and memory requirements. It matches or even exceeds the accuracy of multi-class SVMs. *GraphSVM* is very simple to understand and requires negligible coding, but it can be of substantial utility while dealing with very large classifiers with tens of thousands of classes and millions of instances. In future work, we would like to explore using SVMs with the positive set containing more than one class. The composition of this positive set of related candidate classes is as yet unexplored.

Chapter 5

Enhancing multi-labeled classification

5.1 Discriminative multi-labeled classification

In the previous chapter we studied the phenomenon of confusion between related classes. We turn our attention in this chapter to the related idea of overlap between related classes. The facet of text classification most significantly affected by overlapping class boundaries is multi-labeled classification and we present three discriminative algorithms for it.

The task in multi-labeled classification is to assign all appropriate class labels to documents of interest. In multi-labeled classification a document can have up-to k out of n labels where $1 \leq k < n$. Most typical applications require the ability to classify documents into one out of many (> 2) classes but often it is not sufficient to talk about a document belonging to a single class. This could either be due to a document comprising of multiple parts, each from a different class, or the classes themselves could be overlapping concepts. For example, a document describing the politics involved in the sport of cricket, could be classified as `Sports/Cricket`, as well as `Society/Politics`. When a document can belong to more than one class, it is

called multi-labeled. Multi-labeled classification is harder than choosing one out of many classes because the number of labels for each document needs to be determined in addition to dealing with overlapping class concepts.

Discriminative multi-class classification techniques, including SVMs, have historically been developed to assign an instance to exactly *one* of a set of classes that are assumed to be disjoint. In contrast, multi-labeled data, by its very nature, consists of highly correlated and overlapping classes. For instance, in the Reuters-21578 dataset, there are classes like *wheat–grain*, *crude–fuel*, where one class is almost a parent of the other class although this knowledge is not explicitly available to the classifier. Such overlap among classes hurts the ability of discriminative methods to identify good boundaries for a class. We devise two techniques to handle this problem in Section 5.3. We exploit strong mutual information among subsets of classes to "pull up" some classes when the term information is insufficient. In the next section, we present a new method to directly exploit such correlation among classes to improve multi-label prediction.

5.1.1 SVMs for multi-labeled classification

Most discriminative classifiers, including SVMs, are essentially two-class classifiers. We have seen in Section 2.3.2 various methods of decomposing multi-class problems into binary classification problems. When the one-vs-others method of creating binary ensembles is used for multi-labeled classification special care is warranted. Usually for each of the classifiers in the ensemble, the positive side contains the class of interest and the negative side contains the remaining $(n-1)$ classes. In multi-labeled classification since document have multiple labels, the same document could occur on both the positive as well as negative sides of the binary classifiers. Such duplicates need to be removed from the negative side.

The interpretation of one-vs-others changes here to an ensemble of A-vs-notA classifiers. For each label c_i, the positive class includes all documents which have c_i as one of their labels and the negative side includes all documents which do not have the label c_i. During application, the set of labels associated with a document d is $\{c_i\}$, such that $\mathbf{w}_{c_i} \cdot d + b_{c_i} > 0$. This is the basic SVM method (denoted SVM) that serves as a baseline against which we compare other methods.

Text classification with SVMs has the problem of all classifiers in an ensemble rejecting an instance. In the modified one-vs-others, all constituents of the ensemble emit a $(\mathbf{w}_{c_i} \cdot d + b_{c_i})$ score; for multi-labeled classification we admit all classes in the predicted set, whose score $\mathbf{w}_{c_i} \cdot d + b_{c_i} > 0$. However, in practice, we find that a significant fraction of documents get negative scores by all the classifiers in the ensemble. One technique of selecting good thresholds other than 0 to counter this problem are Yang's rank, proportion, and score-based thresholding techniques [Yan01].

In this chapter, we present algorithms which use existing discriminative classification techniques as building blocks to perform multi-labeled classification. We propose two kinds of enhancements to limitations of the basic SVM method outlined above. First, we present a new algorithm which exploits correlation between related classes in the label-sets of the corpus in Section 5.2. This algorithm combines text features and information about relationships between classes by constructing a new kernel for SVMs with heterogeneous features. In Section 5.3, we present methods of improving the margin of SVMs for better multi-labeled classification in the presence of overlapping class boundaries. We present experiments in Section 5.4 comparing various methods. We review related work in Section 5.5 and summarise in Section 5.7.

5.2 Combining text and class membership features

The first opportunity for improving multi-labeled classification is provided by the co-occurrence relationships of classes in label sets of documents. We propose a new method for exploiting these relationships. If classification as class C_i is a good indicator of classification as class C_j, one way to enhance a purely text-based SVM learner is to augment the feature set with $|C|$ extra features, one for each label in the dataset. The cyclic dependency between features and labels is resolved iteratively.

Training: We first train a normal text-based SVM ensemble $S(0)$. Next, we use $S(0)$ to augment each document $d \in D$ with a set of $|C|$ new columns corresponding to scores $\mathbf{w}_{c_i} \cdot d + b_{c_i}$ for each class $c_i \in C$. All positive scores are transformed to $+1$ and all negative scores are transformed to -1. In case all scores output by $S(0)$ are negative, the least negative score is transformed to $+1$. The text features in the original document vector are scaled to $f(0 \leq f \leq 1)$, and the new "label dimensions" are scaled to $(1 - f)$.

The differential scaling of term and feature dimensions results in a new kernel function applied to the documents. The kernel function in linear SVMs gives the similarity between two document vectors, $K_T(d_i, d_j) = \frac{\langle d_i \cdot d_j \rangle}{|d_i||d_j|}$. When document vectors are scaled to unit L_2 norm, this becomes simply the $cos\theta$ of the angle between the two document vectors, a standard IR similarity measure. Scaling the term and label dimensions sets up a new kernel function given by

$$K(d_i, d_j) = f \cdot K_T(d_i, d_j) + (1 - f) \cdot K_L(d_i, d_j) \tag{5.1}$$

where K_T is the usual dot product kernel between terms and K_L is the kernel between the label dimensions. The tunable parameter f is chosen through cross-validation on a held out validation set. The label dimensions interact with each other independent

of the text dimensions in the way this kernel is set up. Just scaling the document vector suitably is sufficient to use this kernel and no change in code is needed.

Documents in D thus get a new vector representation with $|T| + |C|$ columns where $|T|$ is the number of term features. They also have a supervised set of labels. These are now used to train a new SVM ensemble $S(1)$. We call this method *SVMs with heterogeneous feature kernels* (denoted *SVM-HF*). The complete pseudo-code is shown in Figure 5.1. This approach is directly related to our previous work on Cross-Training [SCG03] detailed in Section 3.3.1 where label mappings between two different taxonomies help in building better classification models for each of the taxonomies.

1: **Input:** Labeled training set of documents T
2: **Output:** SVM models with class correlation information
3: Represent each document $d \in T$ as a vector d with $\|d\| = 1$
4: Build one-vs-others SVM classifier $S(0)$ using text tokens only
5: **for** each document $d \in T$ **do**
6: Apply $S(0)$ to d, getting a vector $\gamma_C(d)$ of $|C|$ scores
7: Concatenate vectors d and $\gamma_C(d)$ into a single vector scaled as per Equation (5.1)
8: Maintain unit L_2 norm: $\|d\| = 1$
9: Add this vector into the training set of $S(1)$
10: **end for**
11: Induce a new one-vs-rest SVM classifier $S(1)$ for all $d \in T$

Figure 5.1: SVMs with heterogeneous feature kernels

Testing: During application, all test documents are classified using $S(0)$. For each document, the transformed scores are appended in the $|C|$ new columns with appropriate scaling. These document are then submitted to $S(1)$ to obtain the final predicted set of labels.

5.3 Improving the margin of SVMs

In multi-labeled classification tasks, the second opportunity for improvement is provided by tuning the margins of SVMs to account for incomplete label-sets attached with documents. As we saw in Section 5.1, the Reuters-21578 dataset's $wheat - grain$ and $fuel - oil$ like class sets are overlapping and inspection reveals that label-sets for documents seem incomplete. Discriminative methods work best when classes are disjoint. Owing to such incomplete label-sets with overlapping concepts, the documents are best treated as 'partially labeled'. Therefore, it is likely that the 'others' set includes instances that truly belong to the positive class also. We propose two mechanisms of removing these examples from the large negative set which are very similar to the positive set. The first method does this at the document level, the second at the class level.

5.3.1 Removing a band of points around the hyperplane

The presence of very similar negative training instances on the others side for each classifier in an SVM ensemble hampers the margin, and re-orients the separating hyperplanes differently than if these points were absent. If we remove these points which are very close to the resultant hyperplane, we can train a better hyperplane with a wider margin. The algorithm to do this consists of two iterations:

1. In the first iteration, train the basic SVM ensemble.
2. For each SVM trained, remove those negative training instances which are within a threshold distance (band) from the learned hyperplane. Re-train the ensemble.

We call this method the *band-removal* method (denoted BandSVM). When selecting this band, we have to be careful not to remove instances that are crucial in defining the boundary of the others class. We use a held-out validation dataset to

choose the band size. An appropriate band-size tries to achieve the fine balance between large-margin separation, achieved by removing highly related documents which most likely should have been assigned the +ve label, and over-generalization, achieved by removing points truly belonging to the negative class.

5.3.2 Confusion matrix based "others" pruning

Another way of countering very similar positive and negative instances, is to completely remove all training instances of 'confusing' classes. Confusing classes are detected using a confusion matrix quickly learned over held out validation data using any moderately accurate yet fast classifier like naive Bayes [GSC02]. The confusion matrix for a n-class problem is n-by-n matrix M, where the ij^{th} entry, M_{ij}, gives the percentage of documents of class i which were misclassified as class j. If M_{ij} is above a threshold β, we prune away all confusing classes (like j) from the 'others' side of i when constructing a i-vs-others classifier. If the parameter β is very small a lot of classes will be excluded from the others set. If it is too large, none of the classes may be excluded resulting in the original ensemble. β is chosen by cross-validation. This method is called the *confusion-matrix based pruning* method (denoted *ConfMat*). This two-step method is specified as:

1. Obtain a confusion matrix M over the original learning problem using any fast, moderately accurate classifier. Select a threshold β by cross-validation.

2. Construct a one-vs-others SVM ensemble. For each class i, leave out the entire class j from the 'others' set if $M_{ij} > \beta$.

ConfMat is faster to train than *BandSVM*, relying on a confusion matrix given by a fast NB classifier, and requires only one SVM ensemble to be trained. The user's

domain knowledge about relationships between classes (e.g. hierarchies of classes) can be easily incorporated in *ConfMat*.

5.4 Experiments

In this section we present the results of our experiments with the various discriminative methods for multi-labeled classification. We used the Reuters-21578 and Patents datasets described in Section 2.4.2 and Section 2.4.2 for our experiments. The datasets were pre-processed by doing stemming, stop-word removal and we only considered tokens which occurred in more than one document at least once. The top 1000 features by mutual information were retained in both datasets. We used a subset of 30 most populated classes of Reuters-21578 for testing statistical significance but also used all the 135 classes for general experiments. Similarly we used the F hierarchy for significance tests for Patents and also used all 114 classes in other experiments. The significance tests were done by 10 random train-test splits on the specified label-sets, repeating the experiments, and performing the paired t-test on the results.

All evaluation measures used in our experiments are slightly different but in the same spirit as those outlined in Section 2.4.1. The measures we use are on a per instance basis and the aggregate value is an average over all instances. For each document d, let L be the true set of labels, S be the predicted set of labels. Accuracy is measured by the Hamming score which symmetrically measures how close L is to S. Thus, Accuracy$(d) = |L \cap S|/|L \cup S|$. The standard IR measures of Precision (P), Recall (R) and F_1 are defined in this setting as $P(d) = |L \cap S|/|S|$, $R(d) = |L \cap S|/|L|$, and $F_1(d) = 2P(d)R(d)/(P(d) + R(d))$.

All experiments were performed on a 2-processor 1.3GHz P3 machine with 2GB RAM, running Debian Linux. *Rainbow* [1] was used for feature and text processing

[1] `http://www.cs.cmu.edu/~mccallum/bow/`

and SVMLIGHT [2] was used for all SVM experiments. In Section 5.4.1 we look at the overall accuracy and $F1$ comparisons of all methods. In Section 5.4.2 we inspect some results that highlight correlation between classes within the label-set discovered by *SVM-HF*. In Section 5.4.3 we look at the different methods with respect to the number of labels predicted for documents compared to the true number of labels and note that our proposed methods performed much better than the baseline.

5.4.1 Overall comparison

Table 5.1 and Table 5.2 show the overall comparison of the various methods on the Reuters-21578 and Patents datasets. Table 5.1 shows comparison on all 135 classes of Reuters-21578 as well as results of averaging over 10 random train/test splits on a subset of 30 classes. Table 5.2 shows the comparison for all 114 subclasses of Patents and average over 10 random train/test splits on the F class sub-hierarchy. For both datasets we see that *SVM-HF* has the best overall accuracy. SVM has the best precision and ConfMat has the best recall. We also observe that BandSVM and *SVM-HF* are very comparable for all measures.

Table 5.1: The Reuters-21578 dataset

	30 class subset				All 135 classes			
Method	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
SVM	82.02	**92.65**	82.47	87.26	81.26	**92.41**	82.45	87.15
ConfMat	76.16	81.64	**88.00**	84.7	80.92	87	**88.37**	87.68
BandSVM	83.18	89.87	87.41	88.63	81.73	88.44	87.54	**87.99**
SVM-HF	**84.25**	91.56	86.94	**89.19**	**82**	88.66	87.27	**87.96**

We performed a directional *t–test* of statistical significance between the SVM and *SVM-HF* methods for the 30 class subset and the F sub-hierarchy. The accuracy

[2] `http://svmlight.joachims.org`

Table 5.2: The Patents dataset

Method	F class sub-hierarchy				All 114 subclasses			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
SVM	66.65	**73.65**	67.57	70.48	42.47	**56.76**	43.37	49.16
ConfMat	66.62	69.70	**70.63**	70.16	41.67	53.40	**51.65**	**52.51**
BandSVM	67.30	72.09	68.90	70.45	43.30	55.24	48.61	51.70
SVM-HF	**68.86**	72.06	69.78	**70.90**	**44.41**	55.35	49.84	**52.45**

and F_1 scores of *SVM-HF* were 2% better than SVM, being a small but significant difference at 95% level of significance.

5.4.2 Interpreting co-efficients

Inspection of the components of $\mathbf{w}$ along the label dimensions derived by *SVM-HF* reveals interesting insights into various kinds of mappings between labels. The signed components of $\mathbf{w}$ along the label dimensions roughly represent the amount of positive or negative influence the dimension has in classifying documents. As an example for the Reuters-21578 dataset, the label dimension for *grain* (+8.13) is highly indicative of the class *grain*. *Wheat* (+1.08) also has a high positive component for *grain*, while *money-fx* (−0.98) and *sugar* (−1.51) have relatively high negative components. This indicates that a document classified as *wheat* is a positive indicator of the class *grain*; and a document classified as *sugar* or *money-fx* is a negative indicator of the class *grain*.

5.4.3 Comparing number of labels

Table 5.3 shows the size of the true set of labels L, and the predicted set S. We fix $|S|$ to be $1, 2, 3$ for each $|L| = 1, 2, 3$. For instance, for $|L| = 1$, $|S| = 1$ for 99% of the instances for the SVM method, and only 1% of the instances are assigned $|S| = 2$.

For singleton labels, SVM is precise and admits only one label whereas other methods admit a few extra labels.

Table 5.3: Percentage of instances with various sizes of S for L=1,2,3 with 30 classes of Reuters. Here, 68% of all test instances in the dataset had L=1; 22% had L=2; 8% had L=3; others had L greater than 3.

	L=1			L=2			L=3		
Corresponding S=	1	2	3	1	2	3	1	2	3
SVM	0.99	0.01	0	0.5	0.5	0	0.52	0.35	0.13
ConfMat	0.83	0.14	0.03	0.27	0.63	0.1	0.17	0.3	0.48
BandSVM	0.89	0.09	0.01	0.32	0.64	0.03	0.22	0.3	0.43
SVM-HF	0.94	0.06	0.01	0.34	0.63	0.02	0.3	0.26	0.39

When $|L| = 2, 3$, we see that SVM still tends to give lesser number of predictions, often just one in half the cases. Compared to SVM, the other methods have a high percentage of instances in the $|L| = |S|$ column; around 63% for two labels and around 40% for three labels. One reason for this is the way one-vs-others is resolved. All negative scores in one-vs-others are resolved by choosing the least negative score and treating this as positive. This forces the prediction set size to be 1 and the semantics of least negative is unclear. The percentages of documents assigned all negative scores by SVM is 18% for 30 classes of Reuters-21578, while ConfMat, BandSVM, and *SVM-HF* assign all negative scores to only 4.94%, 6.24%, and 10% of documents respectively.

5.5 Related work

There has been very limited research in multi-labeled classification compared to other areas of text classification. This is a harder problem than vanilla text classification because of the added problem of estimating the number of classes that documents belong to. Lack of many benchmark datasets has also restricted attention though in

the past few years many multi-labeled datasets have become available. Most notable among related work in this area is the category ranking work [CS03, EW01] and attempts at using generative models for multi-labeled classification [McC99, Hof99].

Cramer et al. [CS03] propose a one-vs-others like family on on-line topic ranking algorithms. Ranking is given by $\mathbf{w}_{c_i} \cdot x$ where the model for each class $\mathbf{w}_{c_i}$ is learned similar to perceptrons, with an update of $\mathbf{w}_{c_i}$ in each iteration, depending on how imperfect ranking is compared to the true set of labels. Another kernel method for multi-labeled classification tested on a gene dataset is given by Elisseeff et al. [EW01]. They propose a SVM like formulation giving a ranking function along with a set size predictor. Both these methods are topic ranking methods, trying to improve the ranking of all topics. We ignore ranking of irrelevant labels and try to improve the quality of SVM models for automatically predicting labels. The ideas of exploiting correlation between related classes and improving the margin for multi-label classification are unique to our work.

Positive Example Based Learning–PEBL [YHC02] is a semi-supervise learning method similar to BandSVM. It also uses the idea of removing selected negative instances. A disjunctive rule is learned on features of strongly positive instances. SVMs are iteratively trained to refine the positive class by selectively removing negative instances. The goal in PEBL is to learn from a small positive and a large unlabeled pool of examples which is different from multi-labeled classification.

Multi-labeled classification has also been attempted using generative models, although discriminative methods are known to be more accurate. McCallum [McC99] gives a generative model where each document is probabilistically generated by all topics represented as a mixture model trained using EM. The class sets which can generate each document are exponential in number and a few heuristics are required to efficiently search only a subset of the class space. The Aspect model [Hof99] is another

generative model which can be naturally employed for multi-labeled classification, though no current work exists. Documents are probabilistically generated by a set of topics and words in each document are generated by members of this topic set. This model is however used for unsupervised clustering and not for supervised classification.

Recently Zhu et al. [ZJXG05] have also worked on exploiting correlation between classes using maximum entropy classification. Their motivation is similar to that of our *SVM-HF* method and they have shown slightly better results on two benchmark datasets. This work confirms the idea of exploiting correlation between related classes for better multi-class.

5.6 Discussion

Multi-labeled classification is more natural than assigning documents to just one class in real-world settings. However the success of various methods in the multi-labeled classification setting is limited because of the additional overhead of assigning a correct number of labels because documents potentially belong to all classes in the label-set. Semantically it is further unclear whether the noise/junk class should be treated as a class in itself for modeling purposes.

In our experience at inspecting standard benchmark datasets like Reuters-21578 and the various web datasets, we feel that the given labels to documents are incomplete. There are many cases in the Reuters-21578 dataset where documents are assigned *wheat* but not *grain*. The latter class intuitively seems to subsume the former but this is not part of the specification of the label-set. In such cases where the semantics of classes is unclear we have seen lots of cases where documents seem to have incomplete label-sets. In such a case we feel discriminative multi-labeled classification methods can achieve higher accuracy levels if the training data is cleaner and complete.

5.6.1 Recent approaches using graphical models

We have been recently involved in some related work where we have attempted solutions to multi-labeled classification using graphical models [Chi05]. We want to exploit the correlation between class labels. As *Graphical Models* [JGJS99] provide a natural formalism to handle correlated random variables, we model the joint distribution of the class labels for a given document using a graphical model where the nodes are binary random variables corresponding to each class.

Let N be the total number of classes and $\vec{X} = \{X_1, X_2, \ldots, X_N\}$ be continuous random variables corresponding to the output scores of one-vs-others SVMs for each class. Let $\vec{Y} = \{Y_1, Y_2, \ldots, Y_N\}$ be binary random variables with states 1/0 indicating whether the class is related to the current document or not. A specific assignment of values to each of the above random variables will be denoted in small case like $\vec{x} = \{x_1, x_2, \ldots, x_N\}$ and $\vec{y} = \{y_1, y_2, \ldots, y_N\}$.

For a given document, given the SVM ensemble output scores $\vec{X}$, the task is to predict the most likely assignment of values to the binary random variables $\vec{Y}$. We learn a conditional random field (CRF), which is a conditional model $Pr(\vec{Y}|\vec{X})$, for the above task. The logistic score of the current label given the SVM output score vector was used as a state feature. The various binary combinations of adjacent true class labels were used as edge features.

The structure of the above CRF represents the various statistical dependencies that exist among the class labels. Structure learning for undirected graphical models is known to be hard. However, it is relatively easy to learn the structure of directed graphical models like Bayesian Networks. Hence, we resort to learning a directed graphical model and later *moralizing* it into an undirected model. We use *WinMine*, a tool developed by Microsoft to learn the structure of the directed model from the class

labels of training instances. Since the above graphical model could have loops, we use Loopy Belief Propagation (LBP) for inferencing. The perceptron algorithm was used to train the parameters of the model. The model was tested on the Reuters-21578 dataset with 10, 18, and 90 classes and results show a significant improvement over baseline SVM predictions [Chi05].

On a related note, more recently Ghamrawi et al. [GM05] proposed the use of graphical models for multi-labeled text classification. They use a factor model on the terms and class labels to learn the class co-occurrence relationships. There are some key differences with our approach. Since SVMs are known to be robust classifiers in the term space, we do not use the terms in the document directly. Instead, we use the SVM ensemble output scores to learn the inter-class dependencies.

5.7 Summary

We have presented methods for discriminative multi-labeled classification. We have presented a new method (*SVM-HF*) for exploiting co-occurrence of classes in label sets of documents using iterative SVMs and a general kernel function for heterogeneous features. We have also presented methods for improving the margin quality of SVMs (BandSVM and ConfMat). We see that *SVM-HF* performs 2% better in terms of accuracy and F_1 than the basic SVM method; a small but statistically significant difference. We also note that *SVM-HF* and BandSVM are very comparable in their results, being better than ConfMat and SVM. ConfMat has the best recall, giving the largest size of the predicted set; this could help a human labeler in the data creation process by suggesting a set of closely related labels.

In future work, we would like to theoretically understand the reasons for accuracy improvement in *SVM-HF* given that there is no extra information beyond terms and

linear combinations of terms. Why should the learner pay attention to these features if all the information is already present in the pure text features? We would also like to explore using these methods in other application domains.

Chapter 6

Bootstrapping text classification systems

6.1 Introduction

In the previous chapters we looked at three relationships between classes, namely mappings, confusion, and overlap. We proposed methods to exploit or overcome these to help in building better applications. We now turn our attention to another important aspect of text classification systems, that of development of classifiers and label-sets in typical new application settings. We focus on the problems of bootstrapping feature sets, document sets, and label-sets while building new text classification systems.

6.1.1 Problem setting

When building new text classification systems, there is initially very limited training data available. The set of classes is usually undefined and users may rely on unsupervised discovery techniques like clustering and even latent semantic indexing to group documents into related sets. Once some such grouping has been done, these groups of documents may be inspected to reveal themes of commonality and names

of classes and classification criteria may be deduced. It is quite standard to proceed cautiously and assign very few training documents to each of these new classes. Classification algorithms are expected to operate on this training set and we need to help the system bootstrap as fast as possible and reach respectable performance levels.

In such settings, and even in the case when the label-set is specified upfront, there is a constant need to revisit the coverage of the label-set at hand. The label-set represents the breadth of knowledge the classification system is aware about and hence defines the coverage or scope of topics. New classes often get introduced into the system that were not present earlier and these classes need to be folded in. There is also very limited training data in terms of labeled documents when building new classifiers. Our focus in this chapter is to study the bootstrapping of classifiers at three different levels. We look at the problem of temporally evolving label-sets by tracking addition of new classes. Our approach is to present candidate new classes to the user whose judgment decides whether the class is to be folded in or not. We also look at bootstrapping classes or concepts by human feedback on document and term level labeling interactions.

6.1.2 Outline

We look at the problem setting in detail in Section 6.1.1. This chapter is organised in two independent halves. First, in Section 6.2 we look at the evolving label-set problem of discovering new classes in unlabeled data. Second, in Section 6.3 we present active learning based mining techniques for incorporating human feedback via document and term labeling to bootstrap the accuracy of classifiers.

Tracking temporal evolution of label-sets in text classification systems: In the first part of this chapter in Section 6.2 we develop the notion of coverage of a label-set. We propose to use the extent to which training data covers unlabeled

data as a method of tracking changing distributions in unlabeled data. In particular we present methods to detect new classes (evolving label-sets) in unlabeled data that were not present in training data. This is another important problem that needs to be handled when building new classifiers for text based systems from scratch.

We introduce the problem and identify three main challenges in Section 6.2.1. We propose the novel idea of abstractions for document representation in Section 6.2.2, propose generative and discriminative methods for selecting new class documents in Section 6.2.3 and Section 6.2.4 respectively. Section 6.2.5 contains heuristics for automatically triggering on detection of new classes. We present experiments in Section 6.2.6, related work in Section 6.2.7, a discussion in Section 6.2.8, and we summarise in Section 6.2.9.

Leveraging human feedback by document and term labeling conversations: In the latter half of this chapter, we look at active-learning based techniques to incorporate human knowledge to guide classifier construction. We wish to extend the active-learning paradigm so that we can process many classes and documents efficiently while exerting little cognitive load on the expert user. We also look at feature engineering steps that can be incorporated at this stage. We explain these solutions using SVMs as a representative linear-additive model.

We explain the linear-additive classification model used for this part in Section 6.3.1. We look at active learning on documents in multi-class multi-labeled settings in Section 6.3.2 and in Section 6.3.3 we introduce the new notion of active learning on terms to bootstrap classifiers when nearly no training data is available. We present some experimental results in Section 6.3.4 and discuss related work in Section 6.3.5. We discuss other aspects of the problem in Section 6.3.6 and summarise in Section 6.3.7.

6.2 Temporal evolution of label-sets

In the previous chapters we have looked at direct relationships between classes; mappings, confusion, and overlap. In the present chapter we look at an indirect relationship between classes. We develop the notion of coverage of a label-set. We propose to use the extent to which training data covers unlabeled data as a method of tracking changing distributions in unlabeled data. In particular we present methods to detect new classes (thus evolving label-sets) in unlabeled data that were not present in training data.

Text classification systems are typically trained on a corpus of training data. The learned models are applied to the incoming stream of unlabeled data for label assignment. An inherent assumption in text classification systems is that training data and unlabeled deployment data follow similar distributions and belong to the same set of underlying classes or concepts. An important practical challenge is that this assumption does not hold in real-life applications. In this chapter, we focus on the problem of new classes being introduced into the system; those that were not defined or relevant during training.

Example settings: Such new classes need to be detected from the unlabeled data and folded into the system. We call this the **evolving label-set** problem. Such a scenario is common in directory systems like Dmoz that manually classify ever-changing web-pages. For example, a directory of scientific disciplines would need to add "bio-informatics" as it emerged as a new discipline, or add an industry type "cell-phones" when they started becoming popular.

Another example is in the news domain where new kinds of news stories about recent events need to be detected and classified. This example is well studied and

called *novelty detection* in the topic detection and tracking (TDT) track at the TREC [1] conferences. The novelty detection task aims for on-line clustering of news stories; its goal is to tag novel news stories as interesting and spawn new events for these stories. On the other hand, the evolving label-set problem in a classification setting requires that new classes be carefully spawned only if they fit into the existing label-set. We review some of the novelty detection work and point out differences to the evolving label-set problem in Section 6.2.7.

Consider a classification problem with n classes, where the classes are documents about certain countries (*India*, *US*, *UK*, ...). Over time, a new country's documents (say Australia) are introduced into the system. The evolving label-set problem is to detect such (one or more) new classes, propose a cohesive set of documents for training the new classes, get user's validation about these fitting in with the label-set, and fold these new classes into the classification system. Such problems occur especially when a nascent classification system is built from scratch, the entire set of labels is not known beforehand, and the user's understanding of the label-set evolves over time.

Contributions: We present algorithms for identifying new classes in both discriminative and generative settings. We introduce the notion of abstractions so as to capture the importance of terms not encountered during training, and also to provide a representation that more intuitively reveals the classification criteria to the user. We perform experiments on three very different types of real-life taxonomies and show that our methods achieve 60–90% precision of discovering unlabeled documents comprising a new class. Our proposed methods for automatically detecting presence of a new class also shows low error rates of about 15%. We also make the surprising discovery that while 2-class discriminative methods like SVM are more accurate than generative

[1] `http://trec.nist.gov`

ones, as far as the discovery of new classes is concerned they fare poorly with respect to state-of-the-art generative methods. We attribute this to the notion of model support inherent to the generative models we used.

An important assumption in the rest of this chapter is that our methods detect only one new class in unlabeled data at a time. If more than one class is introduced in the unlabeled data, for a small number of such additions, we have found successive detection of one class at a time to work well. If many new classes are expected in the unlabeled data, we feel this is outside our problem definition. It is not clear that classification is the best tool to use when training and unlabeled data have drastically different distributions. Such a setting would fall under novelty detection in news stories where new concepts are continuously being introduced in the system.

Problem setting: We envision a scenario where a separate module for new class detection continuously monitors unlabeled documents as they arrive into a text classification system for label assignment. The system diagram is shown in Figure 6.1. When the module called Class-Detector gathers enough evidence of an emerging new class, it sends a trigger to the administrating user. Alternately, the user could periodically query the Class-Detector for the presence of a new class. The Class-Detector then presents to the user a ranked list of documents that could comprise a new class. The user can then choose to add a new class to the classification system with an initial labeled set filtered from this ranked list.

Our methods for tackling the evolving label-set problem *do not interfere* with the working of the main classifier. The main classifier can be any high-performance well-tuned algorithm; a popular choice being Support Vector Machines (SVMs) [Joa98]. As we will see later in Section 6.2.6 the underlying learning model that works best for the detecting new classes can be very different from the optimal main classifier for

label assignment.

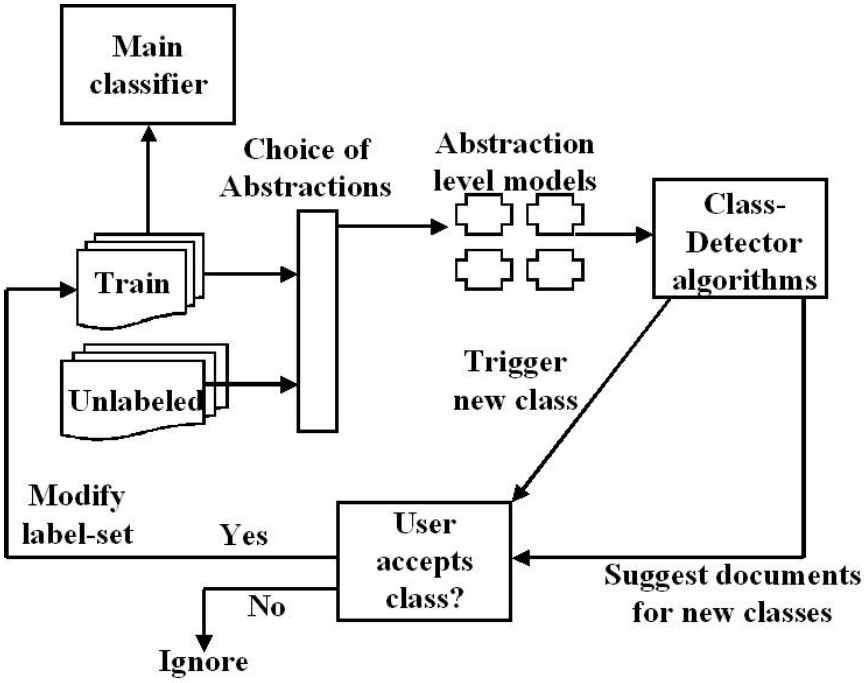


Figure 6.1: System overview

6.2.1 Challenges

We expect to find a new class when there are a significant number of unlabeled documents that do not fit the existing class structure and which are themselves coherent enough to be grouped into a class. Converting this intuition into a robust procedure involves attacking three main problems identified next.

Dealing with unseen tokens

One of the important issues in the evolving label-set problem is **dealing with unseen features**. It is likely that documents of new classes will contain terms that have not been seen during training, and not all of these are important for classification. Further, even normal unlabeled documents contain new terms in abundance, thereby eliminating the possibility of depending on frequency of new terms to detect new

classes. We look at this problem in detail in Section 6.2.2 where we introduce the notion of *abstractions*.

Finding representative documents

The **document selector** that picks a ranked list of documents comprising a possible new class. Separating out documents forming a possible new class from mis-classified documents of existing classes on the basis of being "misfits" in the classification model, is extremely challenging, particularly in the presence of multi-labeled documents. If the selected documents contains several of these mis-classified documents (say more than 50%), the user may get confused about the nature and scope of the proposed new class. In fact, for state-of-the-art one-vs-others SVM ensembles, we have observed that as many as 30% of the unlabeled documents are rejected by all the binary classifiers, making them hard to separate from valid new class documents. We therefore explore generative models to capture the degree of fit of unlabeled documents. We further discuss the problem in Section 6.2.1.

Any new class discovered by our methods needs to *f*it into the label-set. In the above example, the Australia class will be a good fit given the user's understanding of the country-wise nature of the label-set. The challenge is to extract a cohesive set of unlabeled documents to train the new Australia class. This new class should have good training documents such that class-specific parameter estimates are robust and do not spoil the existing multi-class setup. In a bad case it is possible that a wrong new class spoils parameters of the original set of classes.

We employ two techniques to extract a cohesive set of new class documents from the unlabeled data in Section 6.2.3. One technique is to perform Hierarchical Agglomerative Clustering (HAC) on some representation of the training and unlabeled documents. The unlabeled documents of the new class should be sufficiently different

from existing training documents. Another technique we use is to iteratively train SVMs with documents which could not be learned and properly classified in the existing SVM ensemble and hence could belong to a new class.

Automatically triggering detection of a new class

Our system can work in either of two modes. In the first case, the user maintaining the text classification system can periodically probe the system about the nature of unlabeled documents, the system analyzes the unlabeled documents, and proposes a new class from the unlabeled data in hand. Given such a candidate proposed class, it is possible for the user to evaluate whether it fits in the given label-set or not.

The second case is a far more challenging problem where the system automatically detects the fact that a new class is present in unlabeled data. Every document which is mis-classified by the existing classifier potentially defines a new class. Since the existing model did not learn to classify such a document, it is possible that the document contains a new set of words not seen before or combines existing features and yet charts out a completely new concept. More practically, such documents are usually noisy or multi-labeled and are hence mis-classified. The challenge is to collect a group of such documents which form a cohesive set. If this set has enough consistent divergence from other unlabeled documents, the system should be able to trigger detection of a new class. In Section 6.2.5, we propose some methods for evaluating candidate new classes and propose a method to automatically trigger new classes as they occur in the unlabeled data. The **new class trigger** algorithm that decides if there is enough consistent divergence in the unlabeled set to define a new class. We look at this problem closely in Section 6.2.1.

6.2.2 Abstractions for dealing with unseen tokens

The main challenge in the evolving label-set problem is that of dealing with new terms in unlabeled data that were not previously seen during training. These new terms will usually define the scope and coverage of the newly introduced classes. Standard feature selection metrics like mutual information, information gain and other statistical measures, rank features in the training set according to their usefulness in classification. However, there is no mechanism except smoothing by which features not present in the training data can be accounted for. We want to collect all new tokens which together help in defining a new class.

In a supervised setting, the importance of terms is established either explicitly using statistical metrics like information gain, or implicitly, in the classifier via term weights (as in SVMs). Such metrics that depend on labeled data are not applicable here. We depend on indirect methods to establish term importance via a notion of term **abstractions** that assigns importance to a family of terms together. Abstractions indicate various properties of the term based on the way it is used in the documents. Examples of abstractions are: Named-Entity (NE) tags, part-of-speech (POS) tags, formatting features, visual clues in HTML documents, and match with external dictionaries or keyword ontologies. For example, a classification system based on regions would assign high importance to location NE tags. Additionally, abstractions help a user better interpret the criteria used for defining new classes. Abstractions can be clubbed together to form *abstraction sets*. The user can choose a small number of candidate abstraction sets for a label-set by inspection, domain knowledge, or through validation experiments in an off-line training phase. Some examples will make the notion of *abstractions* clear.

Consider an example scenario where the evolving label-set problem occurs. In the

text classification system in Section 6.2, documents are classified according to countries (India, USA, UK, ...). Over time, documents pertaining to Australia, are introduced in the system. These documents will contain words like Sydney and Melbourne which will either not be present in the training data, or their presence will be randomly spread out across all classes. These documents will be mis-classified as the classifier has no existing notion of the class Australia.

Intuitively, in this example, we know that looking at place names is a good indicator of the class of the document. Mumbai very likely implies India, Washington implies USA, and Sydney should imply Australia. *Groupings of tokens based on some external knowledge*, like the location Named-Entity (NE) tag here, are called *abstractions*. Abstractions group features under some human-understandable concept.

In the above example, the location abstraction captures the central theme of the label-set as understood by a human expert. An interesting representation of documents is to look at one (or more) abstraction at a time and ignore all other tokens. We find this helpful in dealing with the evolving label-set problem. We can work with a rich set of abstractions. NE tags like location, person name, organization name, have been found useful in novelty detection [YZCJ02]. Text mining for authorship attribution and gender classification has found part of speech (POS) n-grams to be valuable. The IR and web communities have found visual clues and properties in HTML pages useful; these include words belonging to titles, table headers and hyper-links, which are treated differently from normal text. Custom dictionaries and keyword ontologies are useful in domain-specific text mining tasks. NE tags, POS tags, visual properties, and topical dictionaries are all examples of abstractions. Abstractions can be clubbed together to form *abstraction sets*. The user can choose candidate abstraction sets for a label-set by inspection and domain knowledge. In the above example, we would choose the location, person, and organization NE tags and some combinations of these.

Example: We highlight the importance of abstractions in understanding a label-set with an example in Table 6.1. With the full vocabulary, for three classes, we show the four most indicative features used in classification. We also show indicative features when only the organization name NE tag is seen for the documents.

Table 6.1: Indicative features for a label-set

Full vocab
bank,issue,warrant,fee oil,crude,million,refinery compuserve,service,subscribe,cost
Organization names
Commonwealth Bank of Australia, Commerzbank, Central Bank, Fleet Financial Group Gulf Oil, Chinese Petroleum Corp, Esso Australia Ltd, Natural Gas Corporation Europe Online, Compuserve, First Data Corp, AOL

The full vocabulary makes it hard for us to judge what the classes in the label-set are though we can estimate that the classes are broadly about commerce, oil, and computers. However, looking at only the organization names (the organization name abstraction), we immediately understand that the label-set is about industry types and the classes pertain to a kind of banking, oil companies, and computer data companies respectively. Indeed, this label-set is taken from the *Industries* dataset described in Section 6.2.6. If new classes are discovered in unlabeled data based on the full vocabulary, it is not clear that the user will be able to judge the nature or constitution of the proposed class. On the other hand, the organization name abstraction will definitely help the user in understanding and identifying a new industry type.

6.2.3 Generative methods for selecting new class candidates

In this section we propose generative and discriminative methods for selecting unlabeled documents for a likely new class. First, we present an algorithm for generative classifiers based on the notion of *support*. Following this, we present algorithms for discriminative classifiers based on the notion of classification *confidence*.

Generative methods

Generative models for text such as naive Bayes, Latent Dirichlet Allocation (LDA) [BNJ02], the Aspect model [Hof99], and *BayesANIL* [RCKB05] model the process of generation of documents and document features (*e.g.* words) from classes. The simplest generative model is naive Bayes outlined in Section 2.3.1. It is a very fast and moderately accurate classifier used for text classification tasks. Due to its independence assumptions it is notorious for giving very highly skewed estimates of $Pr(c|d)$; either very close to 0 or very close to 1. However the rankings it gives to these estimates over all classes are known to be very good and this classifier is very widely used in many applications where accuracy is not the sole aim.

As we will see later in this section we develop methods which rely on joint probability estimates $Pr(c, d)$ which are output by the new generative model *BayesANIL* [RCKB05]. We could try and use naive Bayes, LDA, or the aspect model in what follows; such experimentation is left for future work. *BayesANIL* is a also a Bayesian model that assumes conditional independence of words (features) from classes, given documents. It has been shown to be capable of estimating uncertainties associated with the labeling process. Given a corpus of documents $d \in \mathcal{D}$ and some of the documents labeled with class labels $c \in \mathcal{C}$, the model estimates the joint distribution $Pr(c, d)$ of training documents d and class labels c by using a generalization of the EM algorithm. The $Pr(c, d)$ estimate from *BayesANIL* can be interpreted as a measure of *support* for membership

of document d in class c. The marginalized probability $Pr(d) = \sum_{c \in \mathcal{C}} Pr(c, d)$ can be interpreted as a measure of *support* of how well document d fits into the existing label-set defined by classes $\mathcal{C} = \{c_1, c_2 ... c_n\}$.

BayesANIL provides for folding in feature evidence from unlabeled documents, which is especially important, given that some features are often poorly represented in the labeled set. This folding-in is enabled by setting the parameter λ in *BayesANIL* to a non-zero value. In our experiments, we used $\lambda = 0.001$. We chose *BayesANIL* over other generative models, because in empirical experiments, the estimates output were indicative of support for documents from the model even in the presence of noisy, approximate and incomplete labeling. We use this measure *of support* for the problem of detecting documents pertaining to classes beyond those already provided; documents with low support for membership in any of the existing classes are determined as documents of a candidate new class.

In general, we could make use of any generative model that (1) provides an estimate of the joint distribution $Pr(c, d)$ or the *support* for membership of each document in each class and (2) provides for folding in feature evidence from unlabeled documents. Folding in feature evidences from unlabeled documents is an important consideration in the evolving label-set problem. We want to move beyond the usual smoothing techniques for terms not seen in the training data. As we have pointed out, unseen terms are central to determining whether a new class exists in the unlabeled data. We now discuss algorithms for selecting documents that potentially form a new class.

SortPrD

A simple method of selecting documents belonging to a new class is to select documents with the lowest $\Pr(d)$ values. We call this method *SortPrD*. This selection tries to directly capture the lowest support documents; those with uniformly low

support for being generated by any existing class. In practice we will see that in addition to the new class documents, a lot of noisy and multi-labeled documents also tend to get low model support.

PrDNewClass

Another method for suggesting new class documents is to seed an $(n+1)^{th}$ class by documents with the lowest $Pr(d)$ values, re-train the generative model for $(n+1)$ classes, and select unlabeled documents with the highest $Pr(c_{n+1}, d)$. We call this method *PrDNewClass*. Since sorting based on $Pr(d)$ is not perfect, it is likely that documents selected by both these methods will include documents of existing classes that were mis-classified either due to noise or since they were multi-labeled. Next, we propose an algorithm that avoids this limitation.

GenSupp

For the original n-class label-set, we project all training and unlabeled documents in an n-dimensional *support space* where the components for a document d along the n dimensions are the n values of $\Pr(c, d)$. For training documents these $\Pr(c, d)$ values could be pre-computed once during the training phase and stored. We then use a hierarchical clustering (HAC) algorithm to group similar documents. We measure the distance between any two documents d and d' by the average KL-distance between their $\Pr(c, d)$ and $\Pr(c, d')$ n-dimensional scores. If d_is and d'_is are the document projections in the support space, then distance between d and d' is given by:

$$dist(d, d') = \frac{\sum_{k=1}^{n} d_k \ln(\frac{d_k}{d'_k}) + \sum_{k=1}^{n} d'_k \ln(\frac{d'_k}{d_k})}{2} \tag{6.1}$$

We tried single-link, complete-link, group-average, and Ward's method[EHW86] as cluster combination strategies, and found Ward's method to work best. We grew

the dendrogram till we had a large number (say $5n$) of small clusters. Since $Pr(d)$ scores give the probability of generating the document from the model, we expect the lowest $Pr(d)$ values to be assigned to the new class or other noisy, multi-labeled documents. We found this to be true empirically. To get tight sub-clusters from these candidates, we chose clusters which had the lowest average value of $Pr(d)$ of its constituents. Figure 6.2 gives the complete GenSupp algorithm (for Generative model method based on Support). We require each candidate cluster to have a minimum number of unlabeled documents (say 5) to guard against outliers and to be able to define a new class. We also require clusters to be *pure* where the fraction of unlabeled documents is at least $p\%$; this ensures that the new class lies in an area of the support space where there are no (or few) training documents in the vicinity. We chose p as 20% in our experiments.

1: **Input:** $Pr(c, d)$ scores for all training and unlabeled docs
2: **Output:** List of cohesive docs which possibly form a new class
3: $PureClusterSet = \{\phi\}$
4: Project all training and unlabeled docs in n-dimensional space on $Pr(c_i, d)$ scores
5: Perform HAC using Ward's method and average KL-distance from Equation (6.1)
6: Grow the dendrogram to $5n$ clusters
7: **for all** Clusters c **do**
8: If c has a minimum threshold number of unlabeled documents and has a minimum fraction of unlabeled vs. labeled docs: add c *to*$PureClusterSet$
9: **end for**
10: Select one cluster c' from $PureClusterSet$ with the lowest average value of $Pr(d)$ of documents in it
11: Select fixed number of unlabeled documents from c' sorted by lowest $Pr(d)$ as output

Figure 6.2: *GenSupp* algorithm

6.2.4 Discriminative methods for selecting new class candidates

Discriminative classifiers like SVMs do not model the probability of generating a document. Therefore we do not have equivalents of $\Pr(d)$ values for selecting candidate documents for a new class. We assume a standard one-vs-others binary ensemble for multi-class classification as outlined in Section 2.3.2 and design two algorithms that rely on the "rejection" scores of the SVM ensemble.

NotaSVM – NOTA-based method

Let NOTA denote the set of unlabeled documents that are rejected by all the binary SVMs (NOTA stands for None Of The Above classes). Some of these documents are possibly self-similar, coherent and belong to a candidate new class; others may be off-topic, noisy, or multi-labeled. Un-tuned SVMs are known to produce a significant fraction (up to 30%) of such NOTA predictions for standard text benchmarks like Reuters-21578. In one-vs-others, NOTA documents are resolved by assigning them to the class giving the least negative score. We train a $(n+1)^{th}$ binary SVM with the NOTA set as the positive class and the known training data as the negative class. We expect this SVM to prefer documents of the new class and accordingly select candidate new class documents in decreasing order of their scores from the $(n+1)^{th}$ binary SVM. We call this algorithm *NotaSVM* and note that it is similar in spirit to *PrDNewClass* described in Section 6.2.3. The complete algorithm is given in Figure 6.3.

DisConf

We now propose a second algorithm along the lines of the *GenSupp* algorithm for generative models. The *DisConf* algorithm (Discriminative method based on Confidence) is designed to be the HAC-based discriminative counter-part of the *GenSupp*

1: **Input:** n-class SVM one-vs-others ensemble
2: **Output:** Ranked list of unlabeled documents in order of belonging to new class
Require: n-class SVM one-vs-others ensemble
Require: n-class outputs for training and unlabeled data
3: Seed a $(n+1)^{th}$ class with NOTA documents returned by the n-class ensemble
4: Re-train new $(n+1)$ class SVM one-vs-others ensemble and apply it to unlabeled documents
5: Rank unlabeled documents classified positively by the $(n+1)^{th}$ binary SVM by distance from separator

Figure 6.3: *NotaSVM* algorithm

algorithm. In *GenSupp* we represented documents in the $Pr(c, d)$ space. In this case, we represent each document by the n-dimensional vector of projection scores from the SVM ensemble. These scores indicate the prediction *confidence* of each classifier. The distance between two documents is the Euclidean L_2 distance metric.

An important difference between *DisConf* and *GenSupp* is that in *DisConf* we cannot choose a tight cohesive cluster based on the lowest average value of a ranking function like $Pr(d)$. Instead, we need to apply a heuristic like choosing a cluster which has the most negative average value of document projections. Remember that for a document d, most of the prediction scores in its confidence vector will be negative. As we will see in the experiments in Section 6.2.6, this is not a very good heuristic for choosing candidate new classes. The algorithm is same as Figure 6.2, except for two differences (1) each document is projected in the n-dimensional space of the SVM ensemble's output confidence scores instead of the $Pr(c_i, d)$ scores and (2) the L_2 distance metric is used instead of KL-distance. The complete algorithm is shown in Figure 6.4.

1: **Input:** n-class SVM one-vs-others ensemble
2: **Output:** Cohesive cluster of documents as a candidate new class
3: Project all training and unlabeled docs in n-dimensional space on the SVM ensemble's output confidence scores
4: Perform HAC using Ward's method and the L_2 distance metric
5: Grow the dendrogram to $5n$ clusters
6: Choose clusters to have the smallest fraction ($p\%$) of training documents and more than $(1 - p\%)$ of unlabeled documents
7: Choose the cluster with the most negative average value of it's n-dimensional document confidence scores
8: Select these documents as a candidate new class for user validation

Figure 6.4: *DisConf* algorithm

6.2.5 Automatically triggering new classes

In the previous section we saw how to select candidate documents for a new class once a new class is detected or requested by the administrator. We now consider the problem of detecting if indeed a new class exists in the unlabeled data. In general, for a classifier there will be several unlabeled documents not classified in any of the existing classes (NOTA documents in case of SVMs) or having very low probability of being generated by the learned model (low $Pr(d_j)$ in *BayesANIL*). Typically, most of them are due to noisy and mis-classified multi-labeled documents and automatically triggering if there is a new class amongst them is a hard problem.

We approach the problem using *BayesANIL*'s notion of support using its $Pr(c, d)$ scores as follows. Let $T = \{T_1, T_2, \ldots, T_n\}$ be the training documents for the original n-class label-set. We keep aside a set $V = \{V_1, V_2, \ldots, V_n\}$ of documents for measurement. Another set $U = \{U_1, U_2, \ldots, U_n\}$ are treated as unlabeled documents. We introduce a fake class and change the original label-set by adding T_{n+1} which is a cohesive set of documents in U_i found by HAC as in *GenSupp*. T_{n+1} is a fake class as it's documents are chosen from some U_i; for every such U_i, it's corresponding class T_i

already exists in the original label-set. We re-train this $n+1$ class document collection using *BayesANIL* and find the value of two measures MV and MT.

$$MV = \sum_{d \in V} Pr(c_{n+1}, d) \tag{6.2}$$

$$MT = \sum_{d \in T_{n+1}} Pr(c_{n+1}, d) \tag{6.3}$$

MV measures the support for the validation set V from the newly added class, and MT measures the support of the newly added class for itself. We perform this experiment n times, every time adding documents of an existing class as a fake class. This gives us n prototype values which we store as MV_i and MT_i, where $i = 1 \ldots n$. These MV and MT vectors capture the range of possible values when fake classes are introduced into the label-set. Other similar measures are also possible; in particular $\max_{c_i} \sum_{d \in V_{c_i}} Pr(c_{n+1}, d)/|V_{c_i}|$ should also work well.

We expect that if we really detect a new class from the unlabeled data, then it's corresponding MT_{n+1} value should be higher that all previous MT_i's. Since the fake classes always had a corresponding class in T, these documents in T_{n+1} share the probability mass of $d \in T_i$ for some T_i. A real new class will take away some probability mass from all classes in T and $MT_{n+1} > MT_i \ \forall i = 1 \ldots n$. By a converse argument we should get $MV_{n+1} < MV_i \ \forall i = 1 \ldots n$ for a genuine $(n+1)^{th}$ class because a genuine new class will not have any support for the n-class documents $d \in V$.

In Table 6.2 we added the class Australia to our running example of country-wise classification. The original label-set without Australia contained the classes - *USA*, *UK*, *Canada*, etc. Australia was hidden in the unlabeled data and had to be discovered. We measured MV_i and MT_i by adding fake classes from *USA*, *UK*, *Canada* etc. The last row shows values of MV_{n+1} and MT_{n+1} when the Australia class is really inserted

Table 6.2: Discovering Australia

True class	MV_i	MT_i
CANA	0.00001526	0.000073
CHINA	0.00001521	0.000093
FRA	0.00001556	0.000077
GFR	0.00001530	0.000111
INDIA	0.00001548	0.000083
NETH	0.00001571	0.000075
RUSS	0.00001607	0.000062
SAFR	0.00001580	0.000071
UK	0.00001621	0.000043
USA	0.00001655	0.000093
AUSTR	**0.00001509**	**0.000121**

into the label-set using *GenSupp*. We see that MV_{n+1} and MT_{n+1} are respectively minimum and maximum compared to the prototypes. The differences in values are small, but the evaluation is on a constant V.

6.2.6 Experiments

In this section we present the results of our experimental evaluation for the evolving label-set problem. We used the RCV1 dataset described in Section 2.4.2 for our experiments. We selected the 20 most populous classes in each of the three taxonomies. The 4 top-level *topics* classes CCAT, ECAT, MCAT, GCAT were not chosen since these top-level classes in the hierarchy will not play a part in the evolving label-set problem. As we are dealing with the evolving label-set problem, we used the news stories of the first two days. The first day's stories were taken as training data and the second day's stories were taken as unlabeled data for all our Class-Detector experiments. This resulted in 4525 documents for regions, 11637 for topics, and 1571

for industries. This selection gave us good varied datasets in terms of variety of classes and sizes for the experiments.

We experimented with the full vocabulary (global G), the location (L), organization (O), and person name (P) named-entity (NE) tags, and the singular noun (N) part-of-speech (POS) tag. As mentioned in Section 6.2.2 we also used other NE and POS tags as abstractions and also tried pairs of abstractions together as abstraction sets. We report results with $\{G,P,L\}$ for *regions* and $\{G,O,P\}$ for *topics* and *industries*. We found these to be the most appropriate and understandable abstractions for these datasets. In practice, abstraction sets can be chosen from a variety of NE tags, POS tags, visual properties, and custom dictionaries, depending on the problem domain and user expertise.

We used a custom developed named-entity tagger [Ram05] for finding the P, L, O abstractions. We note that this tagging was imperfect and noisy, yet our ClassDetector methods worked well. We used SVMLight [2] for our experiments with SVMs, and a Java implementation of BayesANIL [RCKB05]. For HAC we used Peter Kleiweg's clustering software [3] with our own implementation of KL-distance. All experiments were run on a dual-processor Pentium Xeon server running Debian Linux with 2GB RAM.

We wanted to compare the various algorithms for document selection with each other; *GenSupp*, *SortPrD*, *PrDNewClass* from Section 6.2.3, and *DisConf*, *NotaSVM* from Section 6.2.4. Section 6.2.6 presents the precision results of these comparisons. We also compared the baseline text classification accuracies of simple discriminative and generative methods in Section 6.2.6. Finally we wanted to evaluate the MT and MV measures for triggering new classes. Section 6.2.6 presents the false positive and

[2] `http://svmlight.joachims.org`

[3] `http://www.let.rug.nl/~kleiweg/clustering/clustering.html`

false negative trigger rates.

Selecting new class documents

For each of the three datasets, we hid one class from the training data (first day stories) but retained it in the unlabeled data (second day stories). The training data thus had 19 classes and the unlabeled data had 20 classes. We checked if our algorithms could detect this new class from the unlabeled data and suggest a good set of documents comprising this class for user inspection. Our algorithms present a ranked list of suggestions and we measure the precision of these suggestions. Precision is the ratio of correctly suggested new-class documents to the total number of suggestions. We used 20 suggestions for the reported experiments; results with varying number of suggestions were similar. In our opinion, 20 is a good number for the user to be able to judge the existence of a new class fitting into the existing label-set. The average number of documents of the hidden class was always more than 20 for our particular datasets. For each taxonomy, we report the average precision over all 20 class-detection experiments hiding each class one-by-one. We report results for the following four methods:

- *GenSupp*, *SortPrD*, and *PrDNewClass* from Section 6.2.3,
- *DisConf* and *NotaSVM* from Section 6.2.4.

In Figures 6.5, 6.6, and 6.7 we show the precision values for the *regions*, *topics* and *industries* datasets with three abstractions each. The precision values in each dataset are averaged over 20 experiments. These graphs reveal interesting results about the various methods and the role of abstractions.

First, when we compare the generative method *GenSupp* with the *SortPrD* baseline, we find that *GenSupp* is either better or at par in seven out of nine dataset-

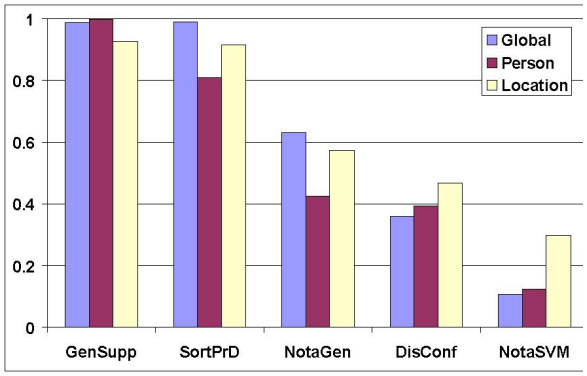


Figure 6.5: Regions

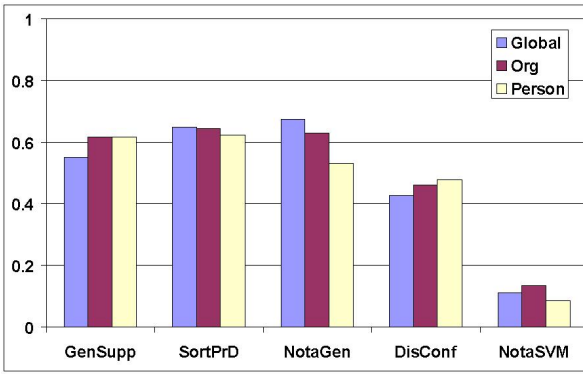


Figure 6.6: Topics

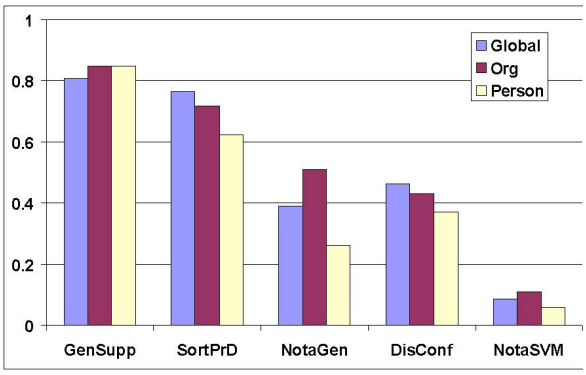


Figure 6.7: Industries

Figure 6.8: Selecting documents – Y-axis represents average precision

abstractions combinations. This illustrates that while the Pr(d) scores are valuable for detecting new classes, they by itself do not suffice, and it is important to account for coherency of the selected documents in defining a possible new class. The only exception is the *topics* taxonomy where we see that *SortPrD* outperforms or is marginally better than *GenSupp*. The characteristic of this dataset is that it is hierarchical and inherently multi-labeled. A document with a leaf label, is assigned all labels on the path from the root to the leaf. Our choice of 20 classes in this dataset contained five such parent-child pairs. Such multi-labeling paired the parent and child labels together and the documents of the hidden classes were already present in the original label-set. BayesANIL considered this noisy labeling and automatically assigned low Pr(d) scores

to these documents leading to better performance of *SortPrD* over *GenSupp*.

We also note that *PrDNewClass* did worse than *GenSupp* and *SortPrD* in most cases. *PrDNewClass* seeds a new $(n+1)^{th}$ class with documents sorted on $\Pr(d)$ scores which are not very precise. This new class learned by BayesANIL contains a lot of labeling noise and suggestions based on $\Pr(c_{n+1}, d)$ end up with lower average precision than other generative models.

Second, we find that abstractions do play an important role in some of the taxonomies. For the industries dataset, the O and P abstractions provide higher precision than G which includes all terms. For regions, the person name abstraction P provides slightly higher precision than G for the *GenSupp* method. We investigated why location name L was not the best abstraction for this dataset. We found that the dataset was highly skewed in class distribution. The USA class in the dataset accounted for about half of the documents and these USA documents were also multi-labeled. Hence, since common location names were already seen in the dataset, L did not prove to be as good as P for this dataset.

Third, in all three cases the discriminative methods (*DisConf* and *NotaSVM*) performed significantly worse than the generative methods. Even for the discriminative methods we find that abstractions matter. *GenSupp* performs better than *NotaSVM* but is not as good as any of the generative methods. When we inspected the results of *DisConf*, we found that there were high-precision clusters present in the results of the hierarchical clustering, but we were unable to pick those clusters for suggesting documents. We discuss this in detail in Section 6.2.8. This was a somewhat surprising finding of our experiments because discriminative methods like SVMs are popularly believed to out-perform generative methods for text classification tasks. In the next section, we show that this holds for our dataset too.

Baseline accuracy

We evaluated the baseline classification performance of the two classifiers we used in our studies. Figure 6.9 shows the micro-average $F1$ results for all the three datasets (Reg for *regions*, Top for *topics*, and Ind for *industries*) for their chosen three abstractions.

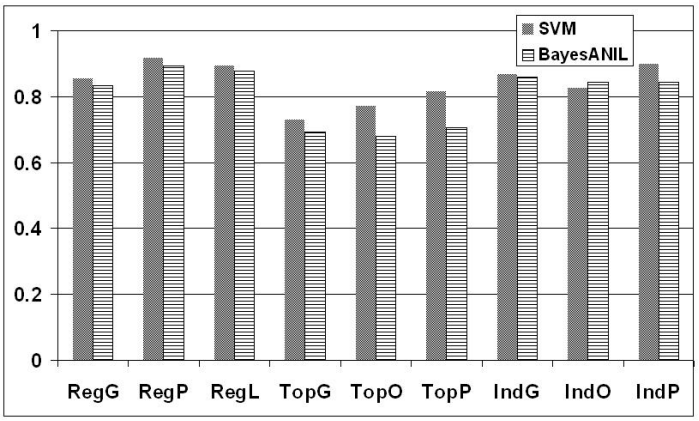


Figure 6.9: Micro-average F1 for SVM and BayesANIL

SVM outperforms BayesANIL in classification accuracy for nearly all dataset and abstraction combinations. It is interesting to see that in the case of the *topics* dataset, TopO (*topics* and the Organisation NE tag) and TopP (*topics* and person name) actually do better classification than TopG (*topics* and the full vocabulary). This affirms our faith in the notion of abstractions – the correct abstractions capture most of the information in the label-set.

Triggering new classes

We report experiments for detecting new classes according to the MT and MV measures outlined in Section 6.2.5. For the *regions* dataset, we experimented with G and L. For each of the 20 classes, we hid the class in the training data, introduced it in the unlabeled data, and determined the fake-class prototype values of MT and MV.

Table 6.3: False Negative and False Positive rates. All numbers are out of 20 runs. Lower numbers are better in both cases.

	False Negatives				False Positives			
	Reg		Ind		Reg		Ind	
	G	P	G	O	G	P	G	O
MT	8	3	14	7	5	3	8	5
MV	9	7	13	5	6	5	7	4

In Table 6.3, we report the number of false positives and false negative triggers based on the MT and MV heuristics. We triggered new class detection 20 times and we tabulate the error rates. We see that MT is a better measure than MV. MT_{n+1} is higher than all fake MT_i values more number of times because the fake classes have low support for documents determined to belong to them but which actually come from an existing class. MV performed comparatively poorly. We also see that abstractions perform better than the full vocabulary in triggering new classes and have lesser false negatives. *GenSupp* generated very few false alarms (3 of 20 best) for the correct abstractions in both the taxonomies.

6.2.7 Related work

Our work is related to the work on Topic Detection and Tracking (TDT) [APL98, ALJ00, YZCJ02] but the problem setting and approaches are different. The aim of TDT is to monitor an on-line feed of news stories and to detect the first occurrence of a new real world event reported in the news. This is called First Story Detection (FSD) and these stories are tracked further using threshold based vector space similarity.

Allan et al. [APL98] introduced the important concept of *surprising* features(words);

features with occurrences temporally far away from their previous occurrences. Most popular techniques at new event detection and tracking (Allan et al. [APL98, YZCJ02]) involve a single pass clustering algorithm with well-tuned novelty detection thresholds. Incoming stories are compared to prototypes of existing events and if more than a threshold away, these stories spawn new events. Allan et al. also showed [ALJ00] that quality of information organisation tasks like FSD is bounded by quality of the underlying event tracking system.

Topic conditioned novelty detection [YZCJ02] by Yang et al. breaks the TDT problem into two parts. They first use a supervised learning algorithm to classify incoming documents in pre-defined broad topic categories (like airplane accidents, terrorism). Following this, they compute weights for all terms and NEs in documents, which are used for topic-specific stop-word removal before FSD. The FSD algorithm used here is the usual comparison to average event vectors with highly tuned thresholds. The term weighting of normal tokens and NEs is central to our work because Yang et al. want to choose topic-level discriminators for topic conditioning, and want a heuristic indicator for the usefulness of certain types of NEs for FSD. Topic-conditioned binning helps them in topic-specific stop-word removal. Our idea of abstractions is related but our binning is across classes and we want to capture abstractions over the entire label-set to discover new classes.

Some systems also explore the use of NE tags [GDH04, YZCJ02] to define more meaningful similarities between documents. This is related our notion of abstractions, but abstractions are more general and not limited to NE tags. *Newsjunkie* [GDH04] is a system for personalizing news feeds based on information novelty measures. Among news articles about an event, the authors find one news article, which contains the most new information with respect to the seed story about the event. They adopt this batch pre-processing method to an on-line version which compares incoming stories with a

sliding window of recent stories. They recognize that a NE-only representation of news stories sometimes works better than the using the full vocabulary. This intuition matches ours, but the two settings are very different. They use NEs to find novel information within a class of stories.

In summary, most work on TDT needs to rely on unsupervised clustering techniques using word-based or NE tags-based similarity and cannot handle multi-labeled (multi-event) stories. The classification setting has multi-class multi-labeled data with a limited number of base classes known in advance. This makes it possible to project documents in a space that better captures their grouping as far as the set of classes in concerned and our problem is to detect new classes. These two setting are very different. The classification criteria requires greater coherence and we depend on user judgment to decide whether a new proposed class $f$its into an existing label-set.

Retrospective event detection work by Yang et al. [YPC98] is also related to our problem of evolving label-sets. Retrospective event detection is used for FSD in a batch off-line mode. They use HAC using group average combination of clusters, with temporal clues and constraints to exploit serial order in news stories. Stories are ordered chronologically and partitioned into non-overlapping consecutive buckets. Each bucket is clustered using HAC till a reduction factor is reached. All buckets are then combined and the current clustering is taken as the partitioning of the next clustering. This iterates till the number of top level clusters is reached. Periodically, each top level bucket is decomposed and reclustered to maintain events across partition boundaries. Yang et al. also give an on-line single pass clustering algorithm with incremental IDF calculation, history window decay with time, and detection thresholding. Retrospective event detection has a strong temporal component which is absent in the classification setting. This makes the two problems different.

Concept drift in classification is another related field of work, but it is quite dif-

ferent from our setting where the set of labels itself changes over time. In concept drift, the distribution of indicative words, and pattern of any *one* class changes over time. A method of dealing with concept drifts in SVMs on text documents is given by Klinkenberg et al. [KJ00]. They use a moving window of training data and iterative train SVMs to capture the time-varying nature of each class. SVMs output $\zeta\alpha$-estimates of their generalisation error during training [Joa98]. These $\zeta\alpha$-estimates are used to pick the best window size that trades-off fast adaptivity (rapidly changing concepts) and good generalization (when drift is slow). An interesting future work for us would be discovery of new classes in the face of concept drifts of existing classes.

6.2.8 Discussion

The evolving label-set problem is not the only kind of challenge that arises from the assumption of training and unlabeled data following a similar distribution. The constitution of unlabeled data changes over time in multiple ways. Meanings of constituent classes change from what they were during training. Work on concept drift tries to address this issue by continually updating class specific models giving more importance to recent representatives of the class.

Classes may loose relevance or simply get dropped from the unlabeled data. This system level detection can be achieved by monitoring the predicted occurrence rates for all classes and identifying classes which do not attract any documents. It is not easy to say whether this happens due to a badly trained model for such classes or whether the class has lost its relevance in time. Removing such classes from the system might be detrimental to the system since it cannot be known whether the class may regain relevance in the future.

In this chapter we have focused on identifying new classes being introduced into the system one at a time. It is possible to detect a small number of new classes by

detecting them one by one. If a very large number of new classes are introduced into the system after training it is not clear whether the problem remains one of classification and other techniques may be needed to handle such cases. Performance guarantees in such cases will be hard to estimate.

The main result in our study of the evolving label-set problem is that generative methods perform better than discriminative methods in selecting a good candidate set of documents for the likely new class. This, in-spite of discriminative methods having higher baseline accuracy is an intriguing observation. To explain this phenomenon we inspected a lot of results and discuss some explanations here. The Hierarchical Agglomerative Clustering (HAC) in *DisConf* used the L_2 distance metric, whose performance was good such high-precision clusters did exist. However, our cluster picking heuristic failed to pick these high precision clusters. This shows us that one-vs-others SVM output scores, though having similar values for similar documents, do not have large variability and the *DisConf* heuristic for choosing clusters fails. We tried another heuristic in *DisConf* that selected clusters whose average of the least negative (for NOTA as well as non-NOTA) scores was lowest. In the case of NOTA predictions in one-vs-others SVMs, the class with the least negative distance from separator is chosen as the winner. This heuristic too did not perform much better than the original heuristic.

To verify whether good clusterings existed, Figure 6.10 compares precision of *GenSupp* with a hypothetical method (DisCheat) that simply chose the cluster that had the highest ratio of new class documents. This method upper bounds our discriminative methods with their cluster picking heuristics. We can see that good clusters do exist, sometimes having higher precision than *GenSupp* but our heuristics cannot pick these clusters.

This shows us that it is hard for heuristics based on distances from separator

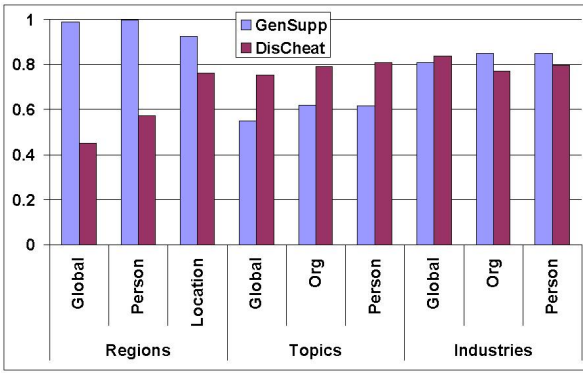


Figure 6.10: Best possible discriminative results

of SVMs to distinguish between new class documents and noisy, multi-labeled misclassified documents. This also shows that the measure of support is more important for the evolving label-set problem than that of confidence. The heuristic for picking clusters based on lowest average support and purity works very well in the case of *GenSupp*. Discriminative models do not provide such a measure [AZ04] and hence are not very useful for detecting evolving label-sets.

It is possible to get multi-class probability estimates from SVMs by using certain isotonic regression techniques [ZE02]. Following this it should be possible to apply the heuristics of *GenSupp* to choose clusters. However it is not clear if these multi-class probability estimates fit in with the notion of model support for documents which works so well for *GenSupp*. This investigation is left for future work.

6.2.9 Summary

We have introduced the evolving label-set problem and presented generative and discriminative methods for dealing with this problem in text classification systems. We introduced the notion of abstractions, which helps the user in understanding label-sets. We use abstractions as a basis for checking the existence of new classes in unlabeled data. We presented the *GenSupp*, *SortPrD*, and *PrDNewClass* algorithms which use

state-of-the-art generative models, and the *NotaSVM* and *DisConf* algorithms using discriminative classifiers. An interesting result of our experiments was that though discriminative models were better in text classification performance, generative models outperformed them in Class-Detector experiments.

We saw that though *DisConf* could discover new class clusters, the unsuitability of SVM scores prevented us from choosing these clusters. It will also be interesting to see applicability of our methods for novelty detection and TDT which are similar yet different tasks. Another opportunity for future work is presented by the triggering problem which we would like to study more formally and characterize the existence of a new class by looking at unlabeled data.

In future work, we would like to integrate evolving label-set detection in working text classification systems and workbenches like HIClass [GHSC04]. We have considered the introduction of one class at a time. This needs to be extended to detect more than one class at time. Preliminary results with successive detection of one class at a time are satisfactory but there is room for improvement.

6.3 Bootstrapping training documents and feature sets

In the previous section we looked at bootstrapping label-sets in nascent text classification tasks by tracking temporal evolution of label-sets. In this section, we present methods for bootstrapping documents and feature sets, again in the context of new text classification tasks where very limited training data is available. We study interactive text classification systems with special focus on providing mechanisms to aid the user in giving expert input in the form of document and term labeling decisions. We present an active-learning based paradigm with significant novel additions

in terms of kinds of data handled and aids that help the user with very little cognitive load.

6.3.1 Interactive text classification systems

We present the broad architecture of an interactive text classification system in Figure 6.11. This architecture focuses on document and term labeling (feature engineering) active-learning conversations between the system and a human expert. The three main modes of interaction are (1) Document-level interaction where the system chooses documents for which it needs human judgment to help it learn better models, suggest labels, and check consistencies and conflicts (2) Word-level interaction where the system suggests discriminative terms for human review, accepts engineered features into the classification models, and add/drop features for certain classes (3) Model and data exploration where the user can inspect the learned models and drill down into documents, terms, and see a variety of accuracy summaries. In order to construct such a system, we first choose a classification model amenable to the above design requirements. Next, we present the detailed document and feature level bootstrapping operations to help take a small amount of training data to good accuracy levels.

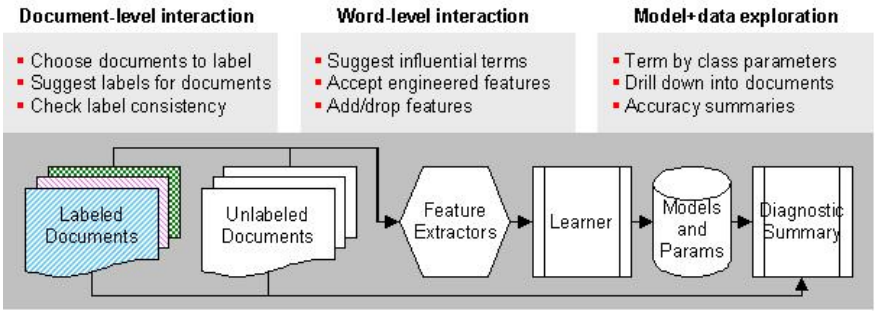


Figure 6.11: Architecture of an interactive text classification system - bootstrapping documents and features

The first step in the design of interactive systems is to choose a flexible classification model that (1) suits state-of-the-art automated learners and (2) can be easily inter-

preted and tuned by the user. A document is a bag of features. Usually, features are words after minor processing like stemming and case-normalization. But the user can also (dynamically) define features to reflect domain knowledge. E.g., month names or currency names may be conflated into synthetic features. On the other hand, the user may notice harmful conflation between "blood bank" and "bank", and define "blood bank" as a single compound feature. We will continue to use *term*, *word* and *feature* interchangeably where no confusion can result. Documents are represented as unit vectors.

Labeled documents can be associated with more than one class in general. We choose **linear additive** classifier models, where each class c is associated with a set of weights $w_1^c, \ldots w_T^c$ corresponding to the T terms in a vocabulary. Each document is represented by a vector of non-negative weights $\vec{x} = (x_1, \ldots, x_T)$,each component corresponding to a feature. The classifier assigns a document all class labels c for which $\mathbf{w}^c \cdot \vec{x} + b_c \geq 0$ where b_c is a scalar per-class bias parameter. As documents vectors have only non-negative components, both magnitude and sign of components of w^c give natural interpretations of salience of terms.

The linear-additive model generalizes a number of widely-used classifiers, including NB, maximum entropy, logistic regression, and SVMs. We focus on SVMs in our work for their high performance and ease of use. Given documents d_i with labels $y_i \in \{-1, +1\}$, a two-class linear SVM finds a vector $\mathbf{w}$ and a scalar constant b, such that for all documents $y_i(\mathbf{w}_c \cdot d_i + b) \geq 1$, and $||\mathbf{w}_c||$ is minimized.

When the application demands more than two classes, one can (1) rewrite the above optimization slightly, with one $\mathbf{w}$ vector per class, so that the discriminant $\mathbf{w}_{c_j} \cdot d_i + b_j$ is largest for the correct class c_j; or (2) build an ensemble of SVMs, each playing off one class against another ("one-vs-one"), and assigning the document to the class that wins the largest number of matches; or (3) build an ensemble of SVMs,

as many as there are classes, each predicting an yes/no label for its corresponding class ("one-vs-rest" or "one-vs-others"). In practice, all these approaches are comparable in accuracy [HL01]. We use one-vs-others as it is easily extended to make multi-labeled prediction and is efficient.

6.3.2 Active learning on documents

The system starts with a small training pool of labeled documents L and a large pool of unlabeled documents U. Assume that the number of class labels is k and each document can be assigned multiple labels. We train k one-vs-others SVMs on L. Our goal during active learning is to pick some unlabeled documents about whose predictions the classifier is *most uncertain*. Various measures are used for calculating uncertainty with SVMs [TK00]. However, these assume binary, single-labeled documents. We extend these to the multi-class, multi-labeled setting as described next.

Uncertainty

Each unlabeled document gets k discriminant values, one from each SVM in the one-vs-others ensemble. We arrange these values on the number line, and find the largest gap between adjacent values. A reasonable policy for multi-labeled classification using one-vs-others SVMs is that discriminant values to the right of the gap (larger values) correspond to SVMs that should be assigned a positive label to the document and the rest should be negative.

We need this policy because, in our experience with one-vs-others ensembles, as many as 30% of documents may be labeled negative by all members of the ensemble. For single label classification, it is common to pick the maximum discriminant even if it is negative. Our policy may be regarded as an extension of this heuristic to predict multiple labels.

With this policy, we declare that document to be most uncertain whose this largest gap is the smallest among all documents. When documents are restricted to have one label, this reduces to defining certainty (confidence) in terms of the gap between the highest scoring and the second highest scoring class.

Bulk-labeling

The user could label these uncertain documents one by one. But experience suggests that we can do better: often, many of these document are quite similar, and if we could present tight clusters that the user can label all at once, we can reduce the cognitive load on the user and speed up the interaction.

We pick the u most uncertain documents and compute pairwise vector-space similarity between documents in the uncertain set, and prepare for the user a cluster/subset of fixed size (set by the parameter s) that has the largest sum of pairwise similarities.

When showing these uncertain clusters to the user, we also provide an ordered list of suggested labels. The ordering is created by taking the centroid of each uncertain cluster and finding its similarity to the k centroids of positive training data of the k classes. Figure 6.12 summarizes the active bulk-labeling process for documents.

An alternative is to use the existing classifier itself to propose suggestions based on the confidence with which the documents in the uncertain cluster are classified into various classes. However, we feel keeping the same suggestion list for all documents in each uncertain cluster reduces the cognitive load on the user. Also, empirically we found in the initial stages this provides better suggestion than the SVMs.

The user provides feedback to the system by labeling all documents in an uncertain cluster in one shot. The labeled documents are inspected by a conflict check module for consistency. We defer discussion of this topic due to lack of space. Once the user confirms the labels,the newly labeled documents are removed from U and added to L.

1: **Input:** A labeled pool L and an unlabeled pool U of documents
2: **Output:** Set of documents and suggestions to present to user for manual labeling
3: **while** user wants to continue with active labeling **do**
4: Train a A-vs-not$A$ SVM ensemble on T
5: *Calculate uncertainty on all documents in U:*
6: **for all** documents $d \in U$ **do**
7: Get k scores by applying the k SVMs to d. Find the largest gap in score values.
8: **end for**
9: Sort the $|U|$ gaps in ascending order and add top u to the uncertain set.
10: Select the s most similar documents from top u
11: *Suggesting ranked list of labels for the group s:*
12: **for all** k classes **do**
13: Find similarity between centroid of s and centroid of positive training data of class k
14: **end for**
15: Sort these distances in a suggested list of classes
16: Present s and the ranked list of k suggestions to the user for active labeling
17: Accept multi-labeled suggestions for all documents in s. Check for conflicts
18: Add these s documents to L with user provided labels and remove from U
19: **end while**

Figure 6.12: The algorithm for active learning on documents

The system then iterates back to re-training the SVM ensemble.

6.3.3 Term level active learning

In interactive classifier construction, users often find it easier to bootstrap the labeled set using trigger terms (that they already know) rather than tediously scrutinize lengthy documents for known triggers. We demonstrate this with an example from the Reuters-21578 dataset shown in Table 6.4. We trained two SVMs using the *interest* class in Reuters-21578; the first trained with a single document per class and the second trained with 50 documents per class. For each SVM, we report some terms

corresponding to the maximum positive weights in the table. The SVM using more data contains terms like "rate" "fe" (foreign exchange), "pct" (percent), and "interest": that a user can readily recognize as being positively associated with the label *interest* that are missing from the first SVM.

Table 6.4: Training 'interest' with 1 and 50 training documents

Num labeled=1		Num labeled=50	
Term	w	Term	w
forecast	0.40	rate	2.08
bank	0.29	fe	1.97
noon	0.20	pct	1.65
account	0.20	market	1.26
oper	0.14	custom	1.01
market	0.14	interest	0.92
england	0.09	forecast	0.92
		stg	0.87
		bank	0.83

We allow a direct process of proposing trigger terms within the linear additive framework. We believe such manual addition of terms will be most useful in the initial phases to bootstrap a starting classifier which is subsequently strengthened using document-level active learning. We propose a mechanism analogous to active learning on documents to help a user spot such terms. SVMs treat labeled terms as mini-documents whose vector representation has a 1 at the term's position and 0 everywhere else, resulting in standard unit length document vectors.

We develop a criterion for term active learning that is based on the theoretically optimum criterion of minimizing uncertainty on the unlabeled set but avoids the exhaustive approach required to implement it [CGJ95, TK00, FSST97] by exploiting the

special nature of single-term documents.

Consider adding a term t whose current weight is w_t in the trained SVM. For terms not in any of the labeled documents $w_t = 0$. Suppose we add t as a "mini-document" with the user-assigned label y_t. Let the new SVM weight vector be w'. Since the term t is a mini-document whose vector has $x_t = 1$ and $\forall t' \neq t, x_{t'} = 0$, we can assume that in the new w' only w_t is changed to a new w'_t and no other $w_{t'}$ is affected significantly. This is particularly true for terms that do not already appear in the labeled set. From the formulation of SVMs, $y_t(w'_t + b) \geq 1$.

If the current w_t is such that $|w_t + b| \geq 1$ then adding t will probably not have any affect. So we consider only those ts where $|w_t + b| < 1$. Adding t with a label $+1$ will enforce $w'_t + b = 1$ i.e., $w'_t = 1 - b$ and with a label of -1 will make it $w'_t = -1 - b$. For each possible value of $y_t = c$, we get a new value of $w'_t(c)$. Thus we can directly compute the new uncertainty of each unlabeled document x by computing the *change* in the distance from separator value as $(w'_t(c) - w_t)x_t$, since uncertainty is inversely proportional to distance from the separator. Let $Pr(c, t)$ be the probability that the term t will be assigned to class c, as our weighing factor. We estimate $Pr(c, t)$ by the fraction of documents containing term t which have been predicted to belong to class c. We then compute the weighted uncertainty $WU(t)$ for a term t as $WU(t) = \sum_c U(c, t)Pr(c, t)$ and then select the term with the smallest $WU(t)$ for labeling. Other details and approximate variants can be found in [Har04]. This gives us a way to compute the total uncertainty over the unlabeled set without retraining a SVM for each candidate term.

6.3.4 Experiments

We have experimented with several text classification tasks ranging from well-established benchmarks like Reuters-21578 and 20-newsgroups to more noisy classification

tasks, like the Outdoors dataset, chosen from Web directories [SCG03]. It is difficult to quantify the many ways in which *HIClass* is useful. Therefore we pick a few measures like the benefits of active learning with terms and document to report as performance numbers. We also present some results which quantify the cognitive load on the user and try to show how *HIClass* eases the user's interaction and labeling process.

HIClass consists of roughly 5000 lines of C++ code for the back-end and 1000 lines of PHP scripts to manage front-end user interactions. The front-end is a web browser, readily available on any user's desktop. XML is used to pass messages between the front-end and the server back-end. LibSVM [CL] is used as the underlying SVM classifier.

All our development and experiments were done on a dual-processor P3 server running Debian Linux and with 2GB RAM. We report numbers for fixed settings of some of our system parameters. Further experiments can be found in [Har04]. Unless otherwise stated, the number of initial documents per class is set to 1, the number of documents selected for bulk labeling is 5 and the number of uncertain documents over which we pick similar clusters (the parameter u of Section 6.3.2) is set to 75.

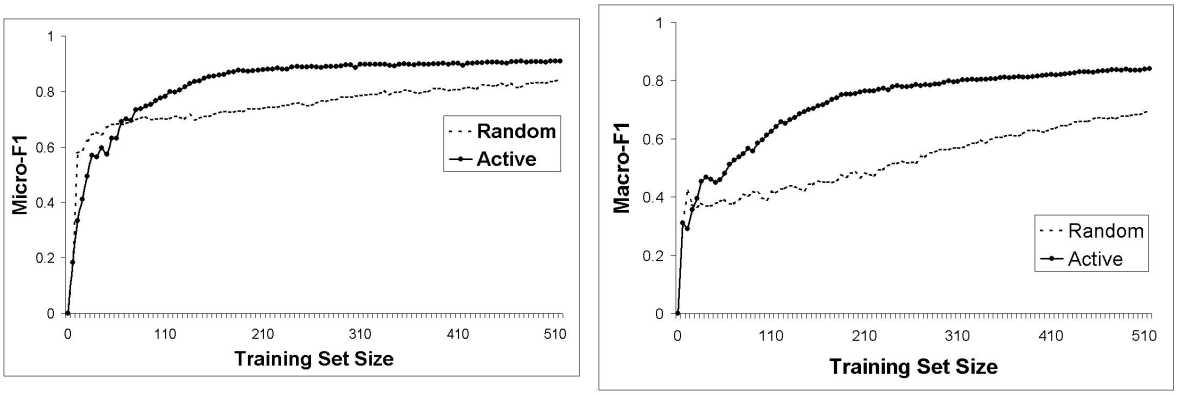


Figure 6.13: Reuters-21578 - Micro and Macro-averaged F1 on held-out test data while increasing training set size, randomly versus using document level active learning.

Document-level active learning

We now show how active learning on documents can reduce the number of documents for which the user needs to provide labels in a multi-class, multi-labeled setting. We started with one document in each class and added 5 documents in each round. All graphs are averaged over 30 random runs. Figure 6.13 compares the micro and macro averaged $F1$, of selecting 5 documents per round using active learning and using random selection for Reuters-21578 (similar results with other datasets omitted due to lack of space). We see that active learning outperforms randomly adding documents to L and reaches its peak $F1$ levels faster.

Reducing labeling effort

We next show the effectiveness of the two techniques that we proposed in Section 6.3.2 for reducing the effort spent for labeling a document. For lack of space we only show results with Reuters-21578 in this sub-section.

Quality of Suggestions: We quantify the quality of suggestions provided to the user by the average rank of the true labels in the suggested list. We see in Figure 6.14 that even in the initial stages of active learning the true classes on an average are within rank 4 whereas the total number of possible classes is 20 for this dataset. We also see that the suggestions with u fixed at 75 are better than at 10 as expected.

Bulk-labeling: We quantify the benefit of bulk-labeling by measuring **inverse similarity**, defined as the number of true distinct labels in a batch of s documents as a fraction of the total number of document-label pairs in the batch. So, if $s = 5$ and each document in a batch has one label and all of them are the same, then the inverse similarity is $\frac{1}{5}$.

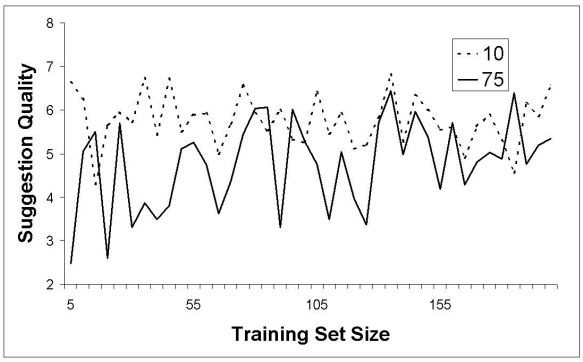


Figure 6.14: Reuters - Quality of suggestion measured as the rank at which correct labels are found in the suggested labels

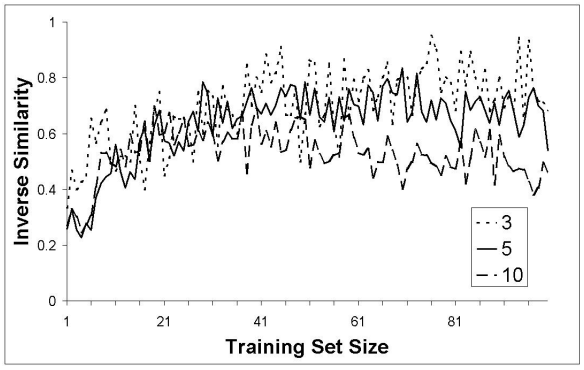


Figure 6.15: Reuters - Benefits of bulk-labeling measured as inverse similarity defined in Section 6.3.4

It is reasonable to assume that the cognitive load of labeling is proportional to the number of distinct labels that the user has to assign. Thus, Figure 6.15 establishes that our chosen set of similar documents reduce cognitive load by a factor of 2. The benefits are higher in the initial stages because then there are several documents with high uncertainty to choose from. With higher number of documents per batch, the benefits get larger.

We cannot set s to be very high because there is a trade-off between *reducing effort per label* by bulk labeling similar documents and *increasing number of labels* by possibly including redundant documents per batch. If we calculated labeling cost in terms of *number of documents* to be labeled, the optimum strategy is to label the most uncertain single document per batch. But the effort the user has to spend in deciding on the right label for rapidly changing document contexts will be high. The right trade-off can only be obtained through experience and will vary with different classification tasks and also the user's experience and familiarity with the data.

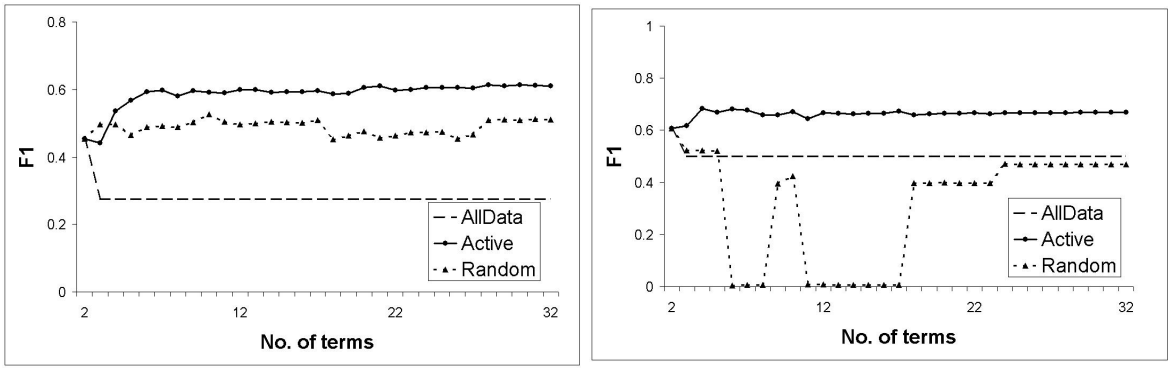


Figure 6.16: Adding labeled terms in score order Reuters (left) and 20-newsgroups (right)

Term-level active learning

Our goal here is to evaluate the efficacy of training with labeled terms. We take all available labeled documents for a class and train a one-vs-rest SVM for that class. All single-term documents that are predicted as positive or negative with very large margins (above $b/3$ here) are labeled with the predicted class and the rest are not labeled. We then start with a SVM trained initially with a single labeled document on each side and keep adding these collected labeled terms in order of the magnitude of their weights (the AllData method). We also evaluate the performance of our term level active learning described in Section 6.3.3. However, we use an approximation algorithm which is less time-intensive and computationally efficient. We select terms with higher values of $f(t)$ where $f(t) = (\sum_{i \in pos} x_{i_t} - \sum_{i \in neg} x_{i_t}) * (N - (pos - neg)(b + w_t))$ where N is the total number of unlabeled documents, and pos and neg refer to number of positive and negative unlabeled documents.

In Figure 6.16 we show the resulting accuracy on 8 classes of Reuters and 3 classes of the 20-newsgroups dataset. Active learning on terms clearly works as expected though the gains are small. This is to be expected since SVMs are trained with very few terms instead of entire documents. Random selection performs much worse in

both the datasets. This confirmed our intuition that term-level active learning is best viewed as a bootstrapping technique followed by document-level active learning.

6.3.5 Related work

Most earlier work on applying active learning to text classification [TK00, MN98b] assumes a single binary SVM whereas our proposed scheme is for multiple one-vs-othersSVMs and for multi-labeled classification. Active learning has also recently been applied to the problem of selecting missing attributes of labeled instances whose values should be filled in by the user [LMG03]. This is different from our setting of term active learning because our goal is to add terms as additional labeled instances. The notion of labeling terms is used in [ALM+03] for building lexicons of terms related to a concept. So the goal there is not to assign documents to categories but to exploit the co-occurrence patterns of terms in documents to categorize terms.

Recently Raghavan et al. [RMJ05] have worked on interactive feature selection in active learning systems. They show that feature re-weighting works better than selective sampling and use this notion to acquire human expert opinion of terms rather than documents. Working with features instead of whole documents takes much less time and interleaving both speeds up the active learning cycle. This is consistent with our motivation and observations with *HIClass* where we use active learning on terms to bootstrap the classifiers in their initial stages when enough labeled documents are not available for training.

6.3.6 Discussion

Our aim in building the *HIClass* interactive workbench was to aid in bridging the chasm between industry and academia perceptions of text classification systems. The main guiding design principle behind *HIClass* was to have a working system that could

be built from scratch and taken up to high accuracy levels with human guidance. We used active learning on terms and documents to bootstrap the classification models as fast as possible. It was important to combine human understanding of real world concepts with the processing power of the machine through a variety of features like data, model, performance summaries, feature engineering, a document labeling assistant capable of bulk labeling and conflict checking, among others.

Our own experience with this version of *HIClass* was very fruitful and insightful. This leads us to think about a generic text classification research workbench that something like *HIClass* can become. Apart from the features already incorporated in *HIClass* it would be interesting to extend it to include at least some of the work mentioned throughout this report.

6.3.7 Summary

We have described *HIClass*, an interactive workbench for text classification which combines the cognitive power of humans with the power of automated learners to make statistically sound decisions. The system is based on active learning, starting with a small pool of labeled documents and a large pool of unlabeled documents. We introduce the novel concept of active learning on terms for text classification. We describe our OLAP-like interface for browsing the term-class matrix of the classifier cast as a linear additive model. The user can tune weights of terms in classes leading to better, more understandable classifiers. *HIClass* provides user continuous feedback on the state of the system, drawing her attention to classes, documents, and terms which would benefit by manual tuning.

Chapter 7

Next-generation text classification platforms

7.1 Introduction

In the previous chapters we looked at a number of novel algorithms to tackle a variety of problems encountered in real-world text classification settings. The common underlying thread in our work has been the exploitation of inter-class relationships. We discovered mappings between topics in related label-sets to build better classifiers as well as propose ontology maintenance tools. We exploited confusion between related classes to tackle the problem of scaling multi-class classifiers using hierarchical and non-hierarchical methods. We also proposed new kinds of features sets called abstractions to develop methods for tracking temporal evolution of label-sets in text classification systems. We developed better discriminative multi-labeled classification algorithms by countering overlap between class boundaries. Finally, we looked at various ways to bootstrap classifiers from training documents and feature sets when text classification systems are being built from scratch.

The main focus of this thesis has been to look at labels and label-sets as important entities in the text classification setup. All our work exploits inter-class relationships

or the structure of label-sets in text classification systems. The set of classes has usually been assumed fixed in earlier text classification research and only documents and features have received attention. In this chapter, we present a unified view of all our work and propose a high level design architecture for next generation text classification platforms.

One of the workbenches widely used in the research community today is the BOW toolkit [McC98]. The toolkit is well designed for the research community, allowing quick access to feature selection options, classification models, accuracy calculations and the like. Researchers are happy to use the toolkit since they can customise various components and are usually interested in some off-the-shelf capabilities it offers. However it is not suitable for large-scale deployment in operational settings. It assumes a rigid data format, class structure, lacks any interaction to incorporate human expertise, and is restrictive in the kinds of text mining applications it can be applied to. We believe significant extensions to such toolkits are needed in the near future for text classification systems. We feel the time is appropriate to design a generic text classification platform. Such a platform will provide basic system primitives to construct text classifiers from scratch and should aid in bringing much needed standardisation to the way systems are modeled and evaluated today in research as well in real-life settings.

Outline: We outline the main entities of the platform, namely, classes, documents, and terms, in Section 7.2. For all these entities, we highlight their role in the proposed platform and relate them to the various pieces of work we have presented earlier in this report. We outline a broad architecture in Section 7.3. The main components of the platform, in addition to the three entities and their interactions, are the *data summaries* module outlined in Section 7.3.1 and the *interaction UIs* outlined

in Section 7.3.2. We recount our experiences in building *HIClass*, an interactive text classification workbench modeled on the lines of the platform proposed in Section 7.3.3. We summarise in Section 7.4.

7.2 Platform entities and their interactions

There are many desirable features of a general-purpose text classification platform. In order to propose an architecture for such a platform, we first need to define the primary entities and their interactions. Document and classification models are the first important kind of entity to design for. Features are the second main entity in the platform. Recall that the main focus of our work has been to treat classes as important entities and we would like the proposed platform to define their characteristics and relationships with other entities. We would like classes (and label-sets) to be the third entity in the platform.

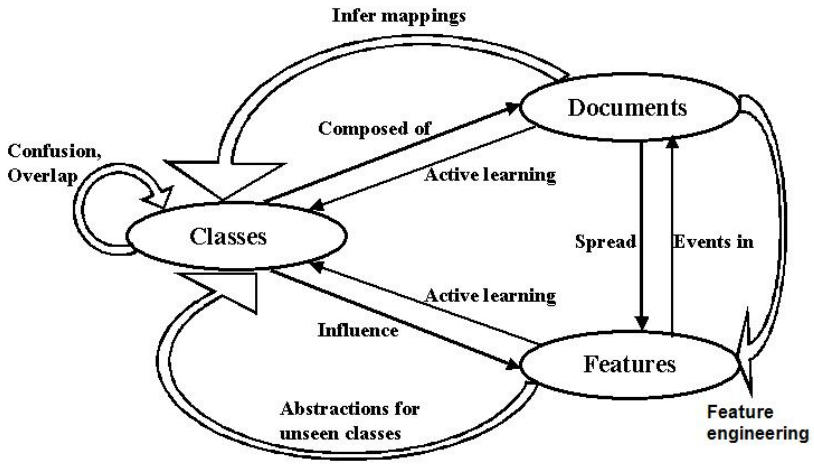


Figure 7.1: Entities and their interactions

Figure 7.1 shows classes, documents, and features as the three main entities and their interactions in future text classification platforms. The thin straight arrows in the figure depict basic relations between the entities and the curved thick arrows show

interactions that enable us to build various standard and novel applications. Features are events in documents and could be simple text tokens as in a BOW model or could be more complicated events detected by NLP tools. Documents are collections of features and their co-occurrence statistics give rise to many interesting applications on the term-document matrix. Classes give rise to features in a generative sense and classes are defined as collections of documents.

Both features and documents are important in determining the meanings of classes and hence both these participate in active learning steps to help build up models of classes with the help of human labeling expertise. Interaction of documents with classes at the correct level of abstraction gives rise to applications like learning mappings between label-sets as we saw in Chapter 3. Classes interacting with each other give rise to relationships like confusion and overlap. Exploiting these allowed us to handle issues of scalability and multi-labeled classification. Interaction of appropriately designed features involving label-sets helped us handle the temporal evolution of label-sets. We see that these interactions between the three main entities is a driving principle to consider when building next-generation text mining systems.

In the rest of this section, we study these entities in detail. We look at various document and classification models in Section 7.2.1 and the feature design and engineering modules of the platform in Section 7.2.2. We look at classes, label-sets, and labeling interactions in Section 7.2.3.

7.2.1 Document and classification models

We discuss desirable properties of document and classification models that form the main entities in the proposed platform. We begin by discussing document models, document representation, and event models among other issues. Following this we discuss various classification models that will be needed in the platform.

Document models

Documents are traditionally treated as a bag-of-words in most text classification packages in the research community. Bag-of-words has become a standard model for text classification research primarily because implementation and evaluation is very easy and some extent of standardisation has occurred in the kinds of pre-processing done across systems making published results somewhat comparable. Steps like stemming, stop-word removal, and selecting features by counts or statistical measures (details in Section 2.2.2) is common. The exact number of features selected by statistical measures or by corpus document counts has no standardisation whatsoever and usually optimizes an accuracy measure through validation.

An important step in any next generation platform will be to push for standardisation in feature selection mechanisms and have all options ready and implemented for the administrator to pick and choose. Since researchers and practitioners choose document models customised to specific tasks at hand, such standardisation may not be achievable, but common terminology and inter-operable parameters need to be defined and supported by the platform. This general standardisation issue is out of our current scope.

Apart from the BOW model, a variety of research has found great value in using various types of features detected by NLP tools. NE recognition has been used in many applications and we have also used NE tags to define abstractions in tackling the evolving label-set problem in Section 6.2. Generally, the notion of abstractions use some higher level properties of text instead of tokens to select features. This notion extends to various alternate representations of documents including NE tags, POS tags, and HTML properties of pages. The pre-processing module of the platform should provide a choice of various mechanisms including the above to the administrator.

An important part of selecting document models for the platform is to ensure the chosen model is amenable to active learning. This is important to be able to incorporate human feedback in the form of document labeling for uncertain documents to help the classifier learn better classification models. The most important module when interactively bootstrapping text classification systems is the active learning module where human expert decisions are leveraged to grow the training set by including useful unlabeled examples (details in Section 6.3.2). From an implementation point of view we need to remember while designing the system that individual documents will need to be displayed to the user in many different ways. These could be showing the original document, showing various abstractions of the document, showing summaries, presenting various parts/features of the document differently and so on. These simple requirements impact storage level considerations and the general disk and memory representation data structures. From general experience the sparse vector format and an inverted index suffice for representing documents internally and externally; it is amenable to all classification models, it can display the original document in various representation, and is simple and efficient.

Classification models

We saw in Section 2.3 that generative and discriminative classifiers are the two broad types of models used in research systems. Many industry-grade applications report good text classification performance with well-tuned rule-based systems. The choice of a classification model is based on the application. The main task of classification models selected in the platform will be to respond to batch-mode signals like 'train', 'predict', 'tune' using different data subsets. The models will also need to expose learned parameters in various forms. For discriminative classifiers like SVMs, distance of a document from various hyperplanes, or signs and weights of features in learned

hyperplanes will be needed. Generative models need to expose different probability estimates like $Pr(d|c)$, $Pr(c|w)$ and so on.

It is well known that SVMs outperform all competitor models in terms of system accuracy, but accuracy is rarely the only end goal. Training SVMs is roughly quadratic in the number of training instances in binary classification problems, and we need to train many binary SVMs for multi-class multi-labeled problems. Our approach to tackling the problem of scalability is to exploit the notion of confusion between classes using *GraphSVM* in Section 4.3. We saw how we could combine the speed of NB with the accuracy of SVMs for learning scalable multi-class models without sacrificing accuracy. A moderately accurate but one-pass training classifier like NB allowed us to group confusing classes together for further accurate classifier learning with SVMs.

The cross-training framework introduced in Chapter 3 also showed very different results with generative and discriminative models. EM2D, the generative cross-training variant, benefited in accuracy from 2d transfer of model information by learning Bayesian parameters over two related label-sets. On the other hand, SVM-CT, the discriminative variant, on the other hand had meagre accuracy improvements since the SVM baseline already had high accuracy, but we could infer insightful mappings between the label-sets, which can assist taxonomy maintenance. Related to cross-training was our experience with multi-labeled classification in the presence of overlapping class boundaries. We were able to design better discriminative methods to exploit labeling correlation and co-occurrence.

While building an interactive text classification workbench [GHSC04], we had success with **linear additive** models like SVMs as the base classification model. This allowed us great leverage in designing, tuning, and interpreting features. We incorporated the document-level and term-level active learning techniques described in Section 6.3 to bootstrap the construction of these classifiers. Each class c is associated

with a set of weights $w_1^c, \ldots w_T^c$ corresponding to the T terms in a vocabulary. Each document is represented by a vector of non-negative weights $\vec{x} = (x_1, \ldots, x_T)$, each component corresponding to a feature. The classifier assigns a document all class labels c for which $\mathbf{w}^c \cdot \vec{x} + b_c \geq 0$ where b_c is a scalar per-class bias parameter. As documents vectors have only non-negative components, both magnitude and sign of components of w^c give natural interpretations of salience of terms. The linear additive model generalizes a number of widely-used classifiers, including naive Bayes (NB), maximum entropy, logistic regression, and support vector machines (SVMs). Our implementation focused on SVMs.

It is clear that multiple classification systems must co-exist within the platform. We propose to use linear additive models as the platform's base classification method. We have seen how generative and discriminative models have different advantages in different settings. For scalability we saw how inexpensive NB classifiers can help efficient training of SVMs in Section 4.3. From cross-training we saw different kinds of gains in the generative and discriminative variants. In Section 6.2 we saw newer generative models working very well to discover new classes in unlabeled data via the notion of abstractions.

In summary, we need a uniform sparse vector representation of documents. We need inexpensive generative models for inferring the confusion matrix thus giving a general picture about the ease of separating/predicting the classes. We need other generative models for a host of high level applications like proposing class mappings, suggesting ontology changes, and tracking the evolution of label-sets. For the base classification sub-system we propose linear additive models like high performance SVMs. In the next section we take a detailed look at characteristics the feature engineering module of the platform should have to develop high performance systems.

7.2.2 Feature engineering

We now review various feature engineering options in the proposed platform. Generating word count features from the BOW representation is very simple. However, in Section 6.2 we also saw the immense value that simple NLP driven features bring, in terms of understanding and improving the capability of the system. There are some very simple examples where we saw that feature engineering beyond the BOW model is essential to enhance the accuracy of the system. Fast evaluation over a variety of test data enables a user to easily identify limitations of a trained model and perhaps the associated feature set. Most users, on inspection of the set of scores of existing features and classes, will be able to propose a number of modifications to the feature set. Some of these modifications may not impact performance on the available test set but could be beneficial in improving the robustness and performance of the classifier in the long run. For the Reuters-21578 dataset, close inspection of some of the terms shown to have a high positive weight for the class *crude* reveals the following:

- "Reagan" is found to be a positive indicator of the class *crude* though proper names should be identified and treated differently.

- "Ecuador" and "Ecuadorean" reveal insufficient stemming.

- "World bank" and "Buenos Aires" should always occur together as a bi-gram; "Union", a high weight term for *crude*, should be associated with "Pacific Union" in *crude*, but as "Soviet Union" in other classes.

- Month names, currencies, date formats, proper nouns should be recognized and grouped into appropriate high-level features to indicate concepts like time, money, and parts of speech respectively.

A general text classification platform should be designed to allow close supervision of features through feature engineering interaction. Ideally a human expert should be able to incorporate domain knowledge by introducing and designing types of features at the aggregate level (like designing lexical analysers for currency value). There should also exist some facility to tweak individual features, as in some of the examples above. While an operation like named-entity recognition will take care of some of these, there will be some mistakes which the user should be able to correct through an appropriately designed user interface.

In Section 5.3 we saw that similar classes overlap with each other and share many features. Say we are classifying documents in a general web directory. The words 'match' and 'referee' are important in distinguishing Sports classes from Science classes but these words are useless within all Sports sub-classes. We designed algorithms that include new kinds of features that capture correlation between document label-sets for better multi-labeled classification. This applied to our feature design in discriminative cross-training as well.

Another important contribution of our previous work was the notion of abstractions to capture high level features and properties of words beyond just word token and named-entities. In Section 6.2 we saw the use of abstractions to help solve the evolving label-set by providing better document representations as well as allowing the user to *understand* the label-set and decide if it needed to be augmented with a proposed new class.

In our work interactive text classification workbench, we provided an OLAP-like interface to browse class-document-feature matrices and tune the contributions of different features. Our approach is to provide *discard*, *positive*, *negative*, and *keep* semantics to weights for features learned by linear SVMs with very few training examples. The discard operation removes the feature in question from all documents of

the class in question. Forcing a feature to have positive (negative) weight keeps the feature only in the positive (negative) document set of that class's SVM and retrains. Details of this user interaction appear with screen shots in the next section. Coupled with the term level active learning discussed in Section 6.3.3, we found it was possible to significantly boost accuracy of our classifiers in an intuitive manner with human guidance in spite of having very limited training data.

We see that feature engineering is a very important aspect of the proposed platform. Choice of a correct feature set is important, but we also saw how keeping alternative representations and feature sets has its merit in discovering relationships between classes and tackling the evolving label-set problem. These situations are expected in real-world settings and when building a system from scratch. The design and implementation of this feature engineering component should have facilities for efficiently switching between representations and leveraging human feedback.

7.2.3 Labels and labeling

The main focus of our work has been to promote class labels as first level entities at par with features and documents in text classification settings. In this section, we look at labels, label-sets and human labeling interactions together as an important part of the proposed platform.

Our system must support evolution of label-sets. In Section 6.2, we introduced *abstractions* to aid the user in determining whether a suggested group of unlabeled documents formed a valid new class and *fit* into the existing label-set. We are not aware of other systematic approaches to this problem in the classification setting, and as seen in Figure 6.1, the human interaction step is crucial before augmenting label-sets and retraining the classification models.

We note that expansion of label-sets is not the only kind of evolution label-sets

could undergo once we start considering classes as first-class entities in the system. Classes could lose relevance and hence may need to be dropped from the label-set or folded into a catch-all "negative" class. A first-cut approach would be to track distribution of classes in unlabeled data streams over time. Merging classes, splitting classes, and re-organising hierarchies form the full spectrum of evolution of label-sets. We are not aware of work that looks at this problem from the abstract level of considering classes as entities and considering their relationships with each other. Our work on detecting and folding in new classes is a first step in this direction.

In Chapter 3 we studied the cross-training framework and saw how mappings learned by the discriminative variant can be used to design taxonomy maintenance tools. Giving the user access to such mapping graphs between related label-sets allows the user to redesign, augment, and re-organise the label-set and this is an important component of the proposed platform. In a Web directory setting, these mappings have tremendous potential to aid editors. Some example mappings we learned between Dmoz and Yahoo! appear in the Appendix. Similarly, from our work in Section 4.2, we can automatically suggest hierarchical organisations of the label-set at hand and allow the user to inspect similarity between classes through dendrograms and other statistics. Again we show sample dendrograms output by our methods for various datasets in the appendix.

The proposed text classification platform should contain the many class and label-set level operations like the ones mentioned above. In addition we would like to see heavy interaction with human experts while exerting as little cognitive load on the users as possible. Several projects reported at the annual Operational Text Classification workshops [LGM+03] describe applications spanning law, journalism, libraries and scholarly publications in which automated, batch-mode techniques were not satisfactory; substantial human involvement was required before a suitable fea-

ture set, label system, labeled corpus, rule base, and resulting system accuracy were attained. However, not all the techniques used in commercial systems are publicly known, and few general principles can be derived from these systems. There is much scope for building machine learning tools which engage the user in an active dialog to acquire human supervision about features, document labels, and coverage of label-sets.

When human supervision is available only as document (and term) label assignments, *active learning* has provided clear principles [CGJ95, TK00, FSST97] and strategies for maximum payoffs from the dialog. In Section 6.3.2 we studied the active learning paradigm in detail. Our contribution was an extensible document level active learning framework that handled multi-class multi-labeled settings and exerted very little cognitive load on the user through aids like bulk-labeling and conflict checking. We wished to extend the active learning paradigm significantly to include feature engineering thus exploiting rapidly increasing computing power to give the user immediate feedback on her choices. Section 6.3.3 introduced our idea of term level active learning and our initial experiments showed this to be a promising approach. Some screen-shots of our implemented system showing these interaction dialogs appear in the appendix.

The human interaction discussed above depends on support from the document models and feature engineering module discussed previously. For document and term level active learning we need a variety of data statistics to help the user make effective labeling decisions. When inspecting coverage of label-sets and evaluating fit of proposed new classes, our system shows document summaries via various abstractions as shown in Figure 6.1.

Another aspect of human interaction is the OLAP-like interface we implemented for the user to inspect how classes, documents, and features impact each other. The user can seamlessly browse the current data and classification models at hand and

suggest feature engineering tweaks discussed in the previous section. Screen shots and details appear in the next section.

7.3 Architecture

In Figure 7.2 we present a simple architecture of text classification systems. There is a pool of documents which represents the content at hand that can either be stored on disk, or could come from data streams or the web. There are standard pre-processing steps applied to this document corpus, followed by an appropriate choice of token models, representation methods, and labeling systems. Classification models are chosen to operate on train-validation-test splits, and classifiers are learned and stored. Standard choices for all of the above have been described in Chapter 2.

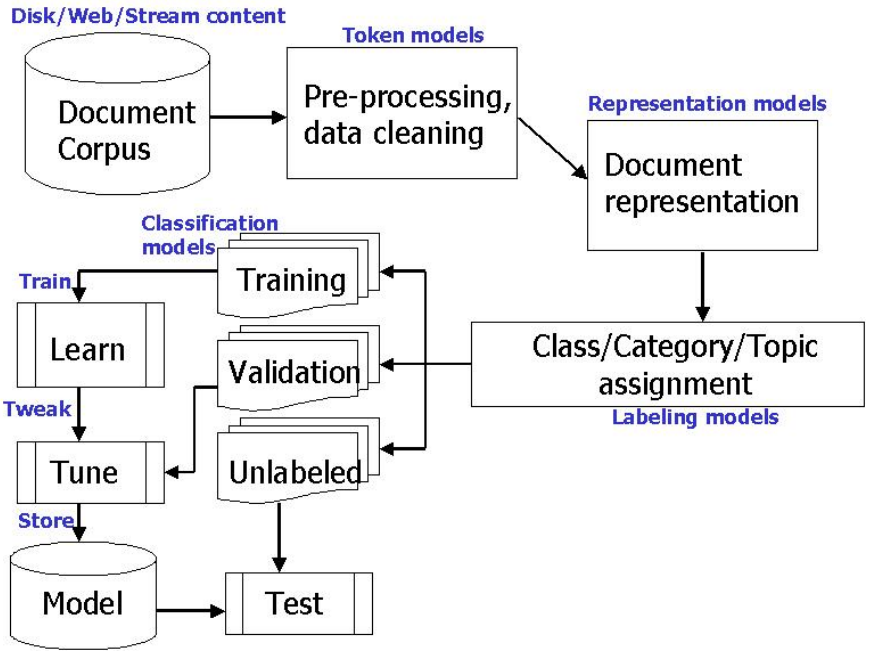


Figure 7.2: The standard text classification setup

Various applications fit this setting where training data for classes is available and we need to learn models to predict classes for new documents. Spam classification, email routing, news filtering, web directory maintenance are all well known examples of

text classification. The differences between these applications come from the structure of the label-set, the type of classification sought, or the kind of data available for use.

Text classification research has usually considered the label-set to be fixed and has rarely exploited rich information present in the way classes interact amongst themselves and with documents and feature sets. We saw a host of problems we could address because we treated classes equally important to documents and features. While Figure 7.2 can serve as a basic architecture for a next-generation text classification platform, there are two crucial components that need to be integrated. The first is a rich variety of data, model, and performance summaries that need to be designed with the implementation of the system. The second are the user interaction mechanisms that will incorporate human feedback on a host of tasks like document labeling, feature engineering, inspecting dendrograms, and maintaining label-sets.

In the rest of this section, we look at important characteristics of these data summary and interaction UI components. We conclude with a description of some of our experiences with implementing a prototype text classification workbench modeled along the lines of our proposed platform called *HIClass*.

7.3.1 Summaries

Interactive evaluation is a very important part of constructing classification systems. After spending considerable amount of time tweaking and fine-tuning systems, the user would like to know the resultant end to end performance of the system. In the research community, evaluation on benchmark datasets is a required component of any work. The industry also reports numbers to reflect the state of their systems. However there is a lack of any standardisation in experimental methodology and evaluation. The proposed platform should make a wide variety of rich data, model, performance summaries available for user inspection. These statistics can draw the user's attention

to classes or areas requiring further tuning or attention.

Quantitative measures like accuracy, F1 and their variants will reveal the state of the system. Class distribution statistics can draw the user's attention to classes that would benefit by further attention. Term distribution statistics will help the user in feature engineering tasks by guiding feature selection. Qualitative summaries include many of the tools used in our work presented in earlier chapters. Mappings between label-sets help the user better organise label-sets and provide a handle into pooling data from different sources. Dendrograms and confusion matrices give an overview of the general state of the classification system.

Quantitative evaluation: Quantifying text classification systems usually follows the route of performing classification experiments on (subsets of) benchmark datasets and reporting measures like $F1$ and accuracy or their many variants on held out labeled data. However researchers have used many variants of standard datasets like Reuters-21578 thus making comparison across systems meaningless. No standard feature selection methodology is defined or specified and there are many ways in which experiments are reported by different researchers.

The proposed platform implementation should have several data and model summaries in understandable graphical form for the user. At any time, the user should be able to draw up evaluation summaries on held out test data. Graphs for micro-average and macro-average precision, recall, F1, and accuracy should be provided. Per class distribution statistics can draw the user's attention to problematic, insufficiently trained classes that need further tuning.

Qualitative evaluation: In Section 4.2 we saw how confusion matrices are a good representation of the classifier accuracy and the mistakes it is making. They help

determine relatedness between classes and give a snapshot of system accuracy. Based on them we saw how to construct dendrograms that further give a hierarchical organisation of classes and are a good corpus visualisation tool.

In Section 3.3 we saw the SVM-CT algorithm for cross-training that generates interesting mappings between related label-sets. These mappings give interesting insights about the structure of label-sets and make for good ontology maintenance tools. The platform should provide all these qualitative summaries to the user.

7.3.2 Interaction UIs

We envision the proposed platform will have a variety of well-designed user interfaces (UIs) for increased interaction with human experts. We have already seen the benefits available for text classification systems when they can interact with human experts. In Section 6.2 we saw how we could detect temporal evolution in label-sets via expert validation from human experts. Our problem there was to detect new classes in unlabeled data for which the system was not trained. We presented methods to automatically select candidates for such new unseen classes but we also argued how the user was required in the loop since only she could judge the coverage of the existing set of labels; using abstractions we proposed a new representation of documents which forcefully brought out the classification criteria for the user.

In Section 6.3 we saw how human document and term labeling conversations could provide tremendous help in bootstrapping text classification systems being built from scratch. We presented document and term-based active learning algorithms that took human feedback on confusing documents and terms to bootstrap classifiers built on very little training data. An important aspect in such interaction dependent techniques is to leverage machine learning to reduce the cognitive load on users. We presented some techniques like bulk labeling, ranking suggestions, OLAP-like inter-

face for browsing and tweaking learned models, and conflict checking to ease the user's task as much as possible.

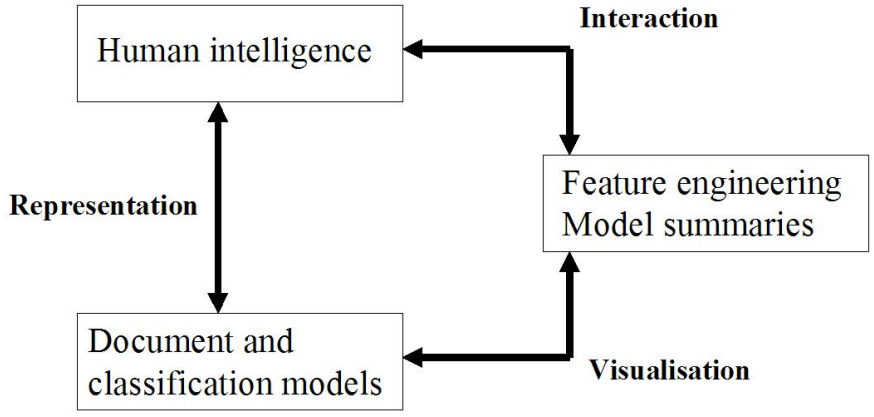


Figure 7.3: Central interactions

The main focus of the interaction UIs is again modeled along the interactions between the three main entities in the platform viz. classes, documents, and terms. In Figure 7.3 we see these central interaction between these entities and note how all user interaction can be designed to be driven from this figure. One box in the figure represents document and classification models which we discussed in detail earlier. These models capture human intelligence since they are modeled by the system architect, and in the text classification setting, inspection of these models is the way humans understand the system. The models are visualised via a host of summaries and feature engineering modules for tuning the models; an example is our OLAP-like interface for browsing, drilling down, and tuning class-document-term matrices.

The user interfaces come into picture when the summary modules are presented to the user for action. Such action is in the form of active labeling for documents and terms, but it is also in terms of inspecting and accepting classes proposed for inclusion in the label-set when dealing with temporal evolution of label-sets. Yet another kind of interaction is the rich variety of tools other parts of our work has produced in the form of confusion matrices, dendrograms, mappings between and across label-sets among

others. Human intelligence or domain knowledge is captured in these interactions and the system benefits by being able to learn better models.

In the next section, we discuss our initial experiences with building a prototype working text classification workbench that was modeled closely along the ideas discussed in this chapter for designing next-generation text classification platforms.

7.3.3 Initial experience

We developed a highly interactive text classification workbench called *HIClass* (for Hyper-interactive text Classification) along many of the desirable principles outlined in this chapter that next-generation text classification platforms should be modeled on. *HIClass* [GHSC04] focused on the feature engineering and active learning modules discussed in Section 6.3. We used SVMs as the underlying linear additive classification model. We implemented *HIClass* in a few thousand lines of C for the core back-end algorithms. User interaction was web-based in a browser front-end coded using PHP scripts with XML used to pass messages to and from the back-end. The back end server process ran on a small Linux system. Next, we see some screen-shots that capture some of the interesting modes of interaction between the system and the human expert.

Figure 7.4 shows the document labeling assistant built on active learning principles described in Section 6.3.2. The left pane shows system navigation options. The right pane contains a group of unlabeled documents presented to the expert user for labeling. There is a ranked list of suggestions next to each document snippet. The user can also inspect the entire document to determine its class.

Figure 7.5 shows the influence of a particular term ('soviet' here) across different classes. Occurrence of the term in representative documents of various classes is highlighted.

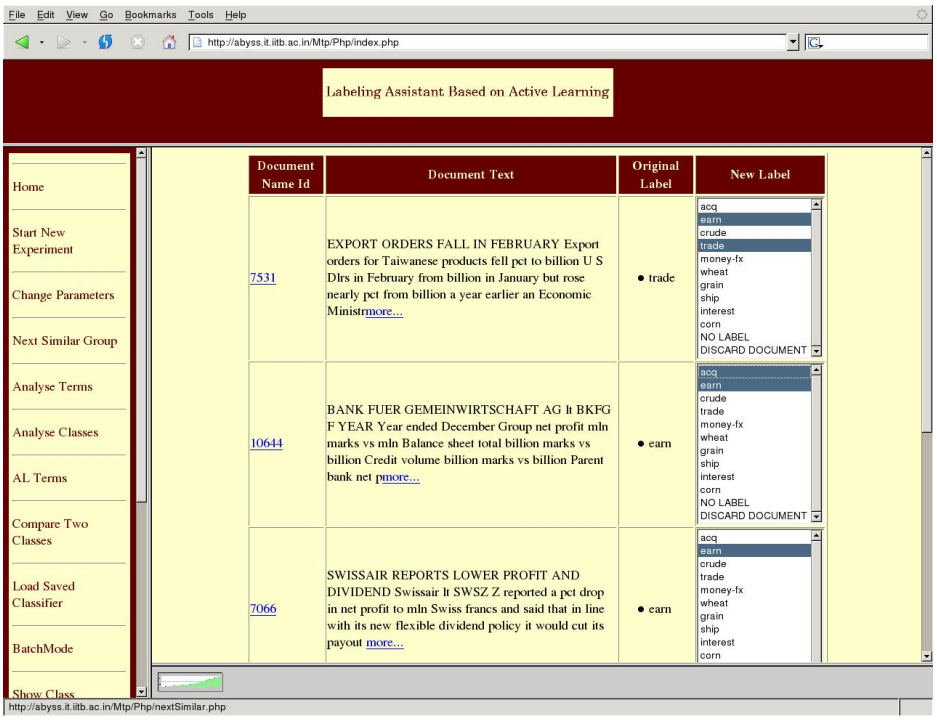


Figure 7.4: *HIClass* screenshot - Document labeling assistant

Figure 7.6 shows the influence of a single term ('billion' in this case) in various classes in the OLAP-like interface. Unlike the previous screen-shot, in this mode of interaction, the user can engage in some of the feature engineering options in the system. This interface is presented in the term based active learning mode (described in Section 6.3.3) where the user can tweak/label how discriminative the term is for different classes. The user can force the term to have positive or negative weights for classes, or can chose to ignore the term from consideration when building a particular class's model.

Figure 7.7 shows the OLAP-like interface from a class-driven context. The most indicative positive and negative terms for a particular class ('corn' here) are shown. This is the feature engineering mode of the system, where after learning on very few documents, the system exposes its models to the user for tweaking. Here again the user can set certain terms to have positive/negative/no influence on the said class

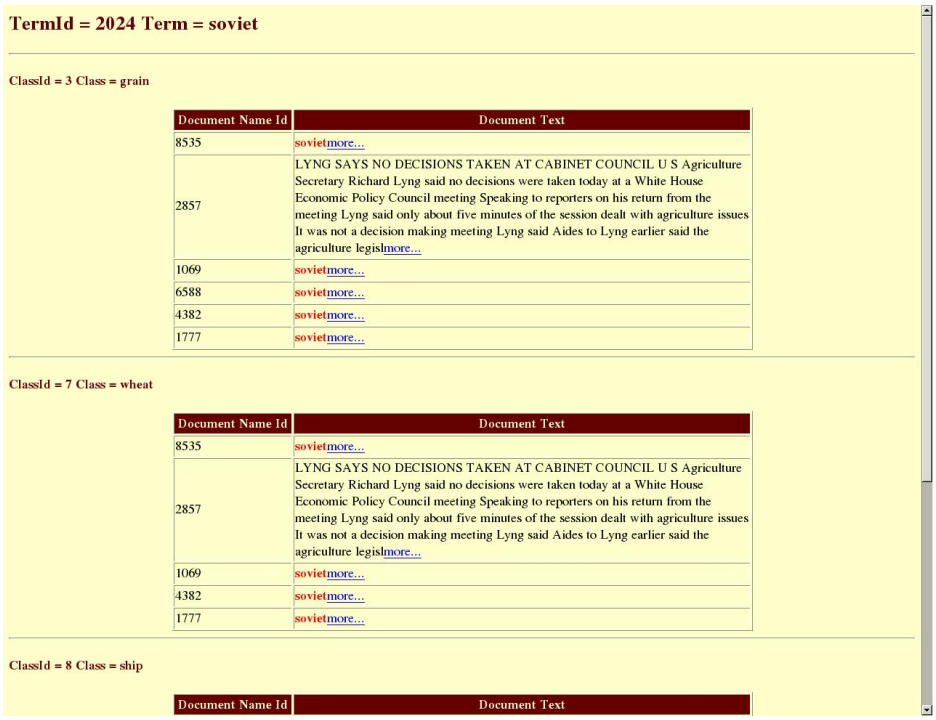


Figure 7.5: *HIClass* screen-shot - Term evidences across different classes

from intuition or domain knowledge.

Our OLAP-like interface for browsing classes, documents, and features is our novel contribution to interactive text classification systems. In addition to providing high level summaries of the corpus, various details shown in the interface include learned weights for features, identifying features within documents in a visually appealing manner, effects of features across various classes, various terms having positive or negative associations for a class, and so on. The tweaking of these features has been discussed in Section 7.2.2. Overall we had a very fruitful experience developing and using *HIClass*. We modeled *HIClass* broadly along the lines of our proposed platform and hence we could leverage many operations like feature engineering and bootstrapping classifiers.

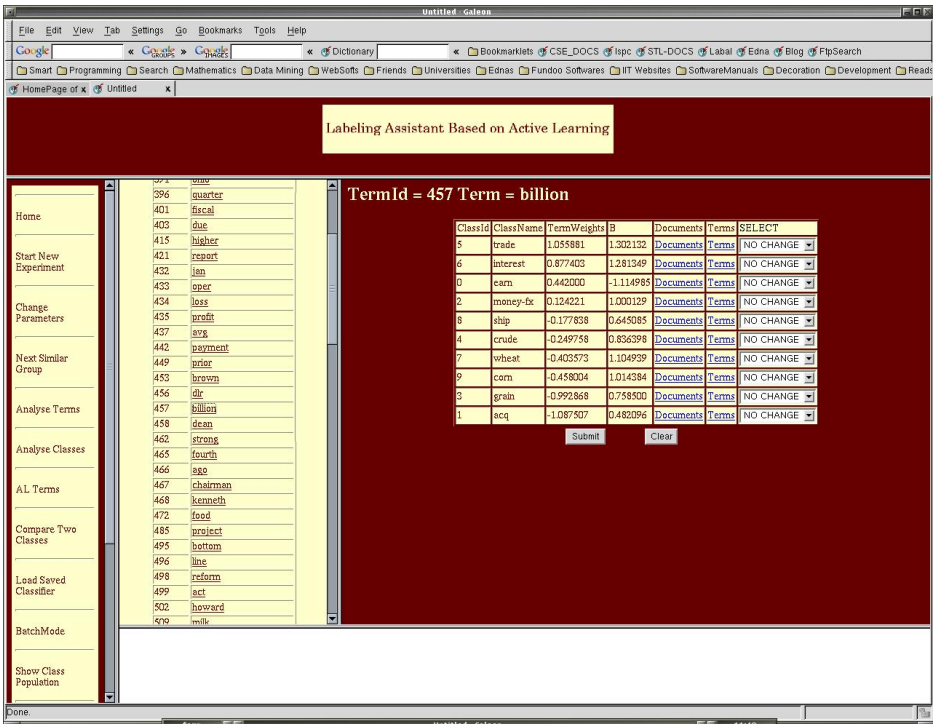


Figure 7.6: *HIClass* screen-shot - Term 'billion' influencing different classes

7.4 Summary

A generic text classification platform is hard to design. Significant effort along the steps outlined in this chapter will be essential before bridging the gaps between industry and academia perceptions of text classification systems. In general, most text classification research and practice concentrates on specific sub-problems and tackles particular assumptions. We are not aware of major work in the area of designing platforms for text classification. Aspects of such systems have been discussed in isolation, but to our knowledge, this is the first attempt at proposing such a platform that combines many interesting applications (developed as part of our work) as integral components of the platform. Our main contribution in proposing such a platform is promoting classes as first-level entities along with features and documents in text classification systems.

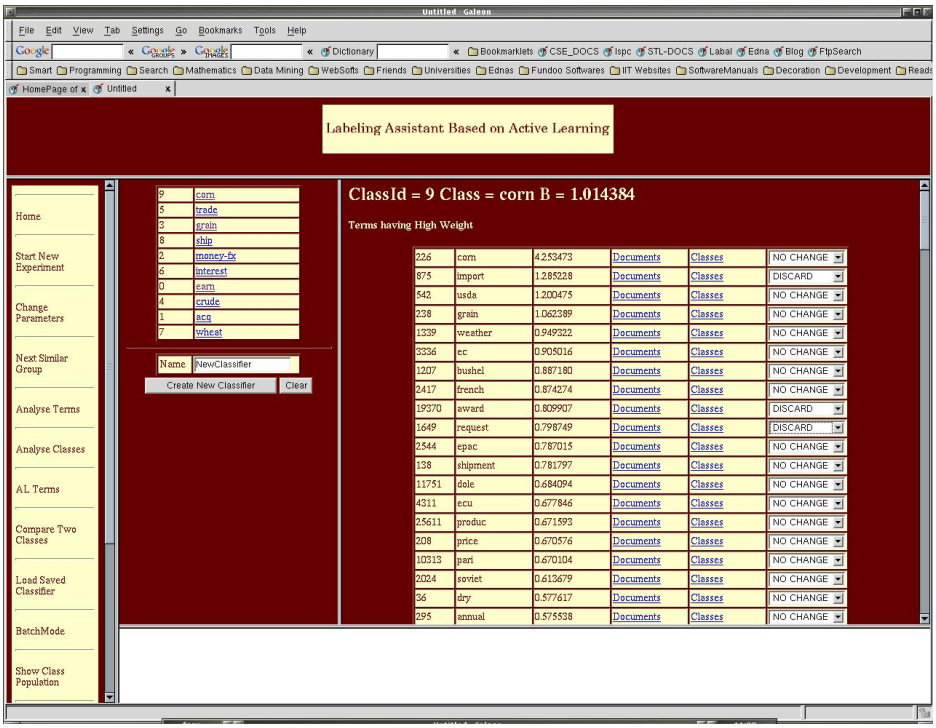


Figure 7.7: *HIClass* screen-shot - Class 'corn' influenced by different terms

Our aim in building an interactive text classification workbench along the lines of such a proposed next-generation platform was to have a working, tunable classifier that could be built from scratch and taken up to high accuracy levels with human guidance. To achieve this, we used active learning on terms and documents to bootstrap classifiers. We combined human understanding of real world concepts with the processing power of the machine through interactive data/model summaries, feature engineering, and cognitive tools like bulk labeling and conflict checking. Our experience with building the *HIClass* workbench modeled along these lines was fruitful.

Chapter 8

Summary and Conclusions

In this thesis we have presented a range of novel algorithms for solving real-world text classification problems arising in different situations. The main focus of our work has been exploiting the notion of inter-class relationships in text classification systems. We proposed four inter-class relationships and developed algorithms based on them for different tasks. We learned mappings between label-sets to build better classifiers and developed taxonomy maintenance tools. We exploited confusion between related classes to handle scalability issues in large-scale multi-class classification. We overcame the problem of overlapping class boundaries by proposing enhancements to discriminative classifiers for multi-labeled classification. We introduced the notion of coverage of label-sets to track temporal evolution in label-sets by detecting new classes in unlabeled data not present during training.

We also looked at different issues in helping bootstrap text classification systems in their early stages of construction when very limited training data is available. Along with the above work of dealing with temporal evolution of label-sets, we also presented an interactive framework where document labeling and term labeling/feature engineering conversations with expert users were equipped with aids to reduce cognitive labeling load and quickly increase accuracy of the system.

This led us to recognise the importance of classes as first level entities in text processing systems. Traditionally the set of classes has been assumed to be fixed, unchanging, and inter-relationships have been ignored. Our work revolves around treating classes as first level entities along with documents and terms and exploiting different inter-class relationships. We concluded by presenting a broad architecture for next generation text classification platforms as we feel the time is ripe to bridge the gap between academia and industry perceptions of text classification systems, actively integrate human knowledge, and relax many of the assumptions usually employed in building systems.

8.1 Future work

Our work leads to some interesting avenues of future work that we would like to explore. We would like to theoretically understand cross-training better and devise formal ways of studying related label-sets. We would like to extend our work in detecting evolving label-sets to larger scales and devise ways to track other kinds of evolution in label-sets apart from detecting new classes. The most exciting direction of work is the idea of building next-generation text classification platforms which could be used for research as well as real world deployment. Studying this under a formal framework thus leading to guarantees about the platform is a promising line of work.

Appendix

We show some diagrams in this section to supplement the work presented throughout this thesis. Mappings between topics for the 5 datasets of Chapter 3 are shown from Figure 1 to Figure 5 as an addendum to the results presented in Section 3.4.2. These mappings are drawn between the Yahoo! and Dmoz datasets and positive links above the threshold of 0.2 are shown plotted using GraphViz[1]. The full versions of these mappings can be found at `http://www.it.iitb.ac.in/~shantanu/ctdemo/`.

Chapter 4 presented the automatic construction of topic hierarchies using dendrograms derived from NB classifiers as a preliminary step in Section 4.2. Parts of dendrograms for various datasets are shown from Figure 6 to Figure 10.

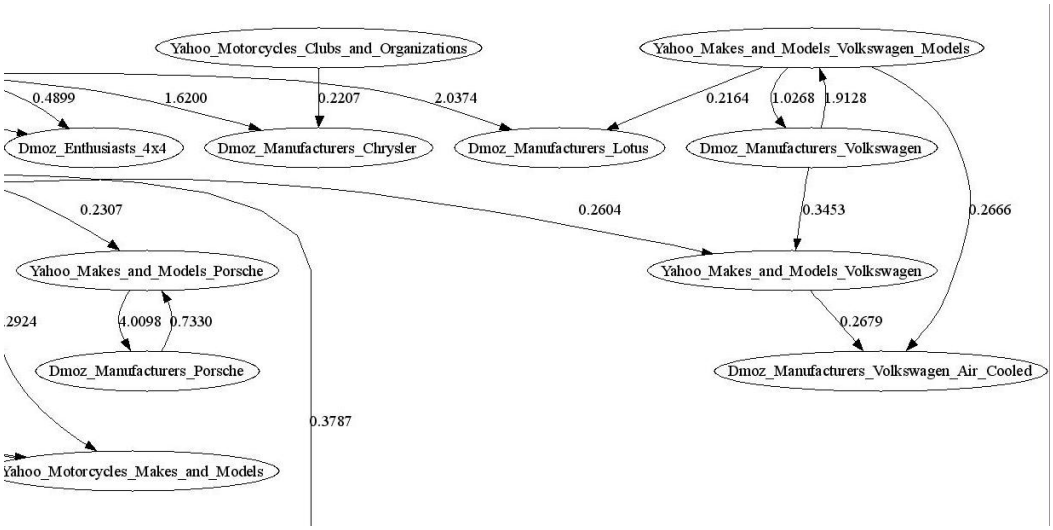


Figure 1: Mappings for Autos

[1] `http://www.graphviz.org/`

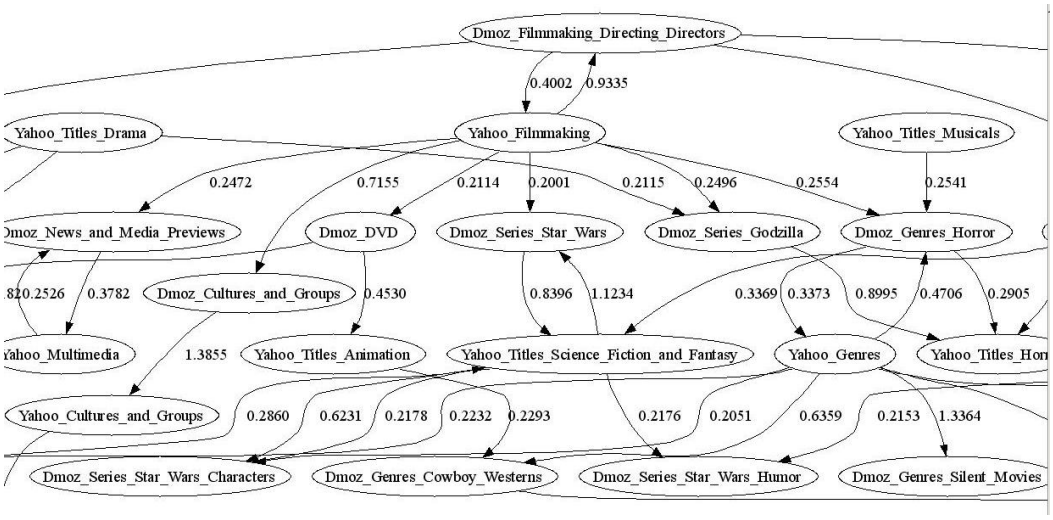


Figure 2: Mappings for Movies

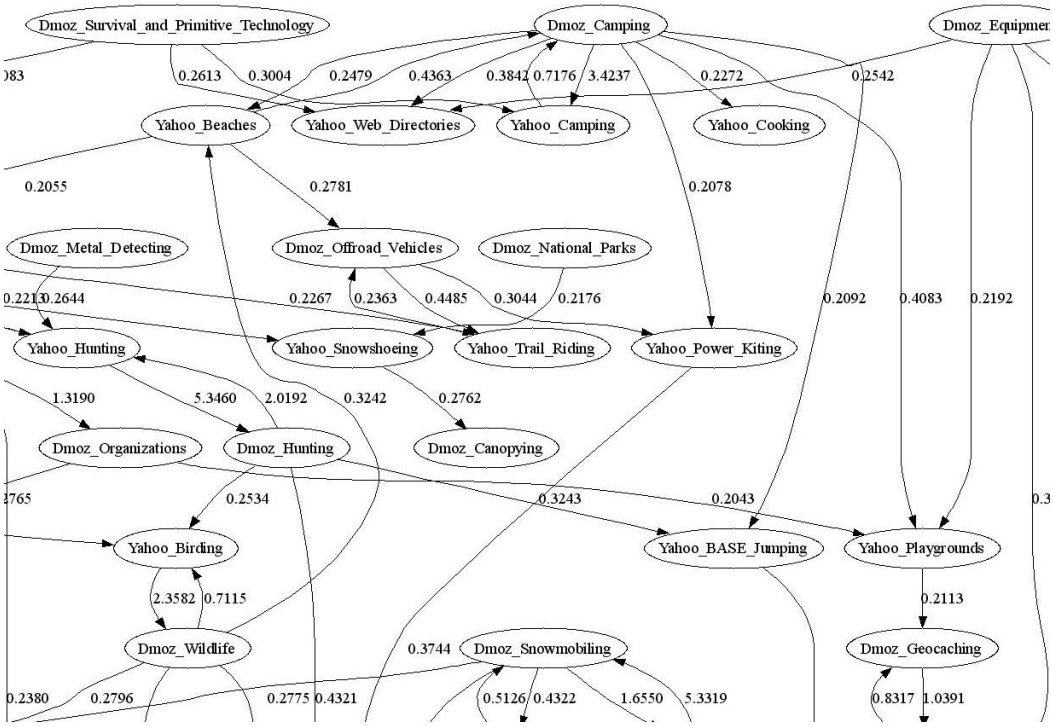


Figure 3: Mappings for Outdoors

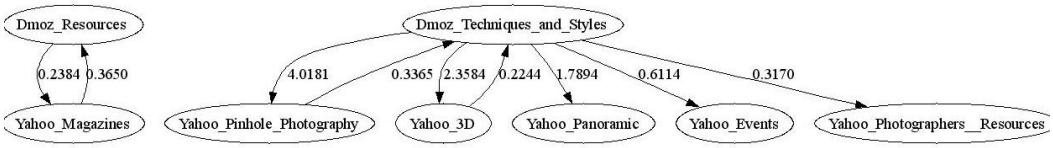


Figure 4: Mappings for Photography

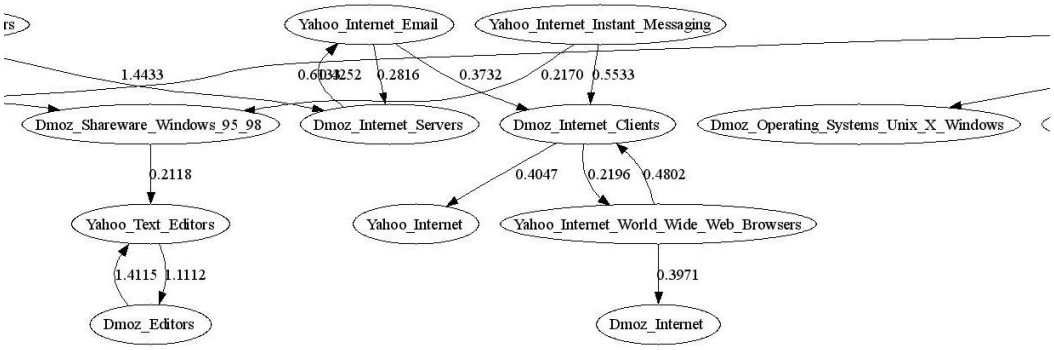


Figure 5: Mappings for Software

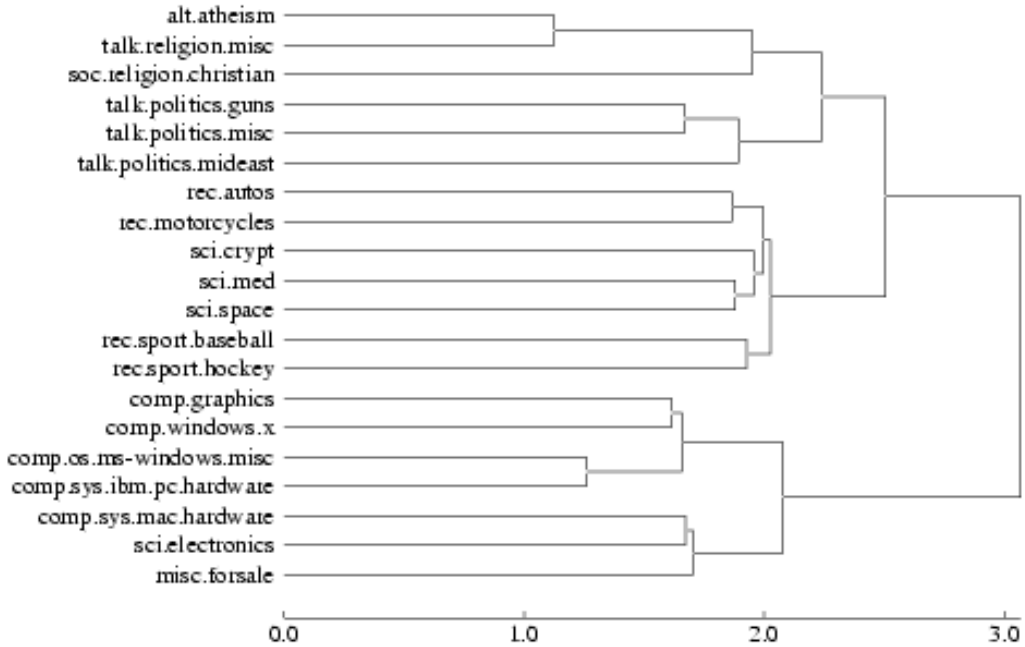


Figure 6: Dendrogram for 20-Newsgroups

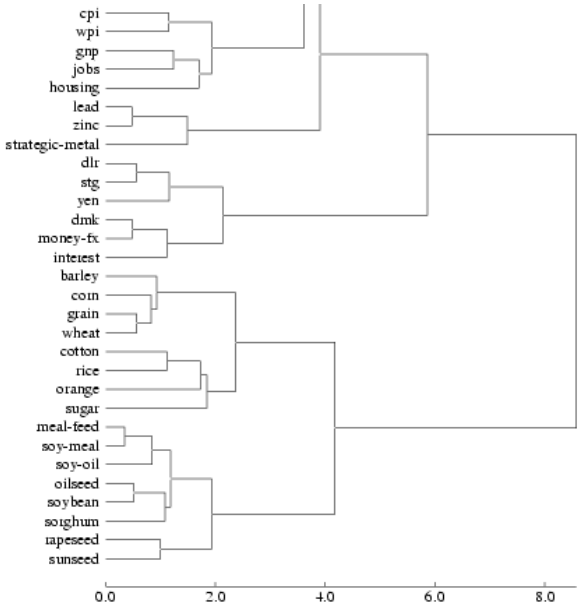


Figure 7: Partial dendrogram for Reuters-21578

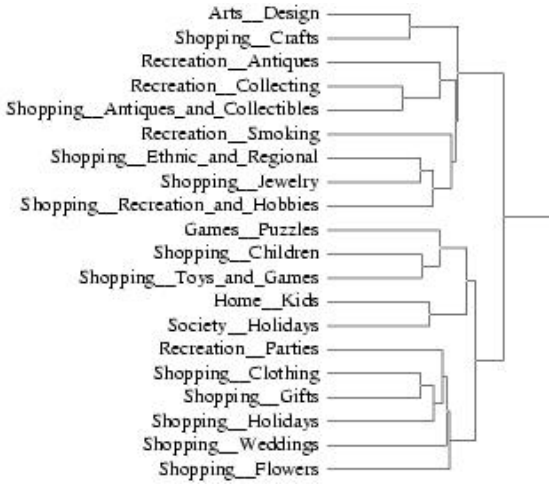


Figure 8: Partial dendrogram for Dmoz 1

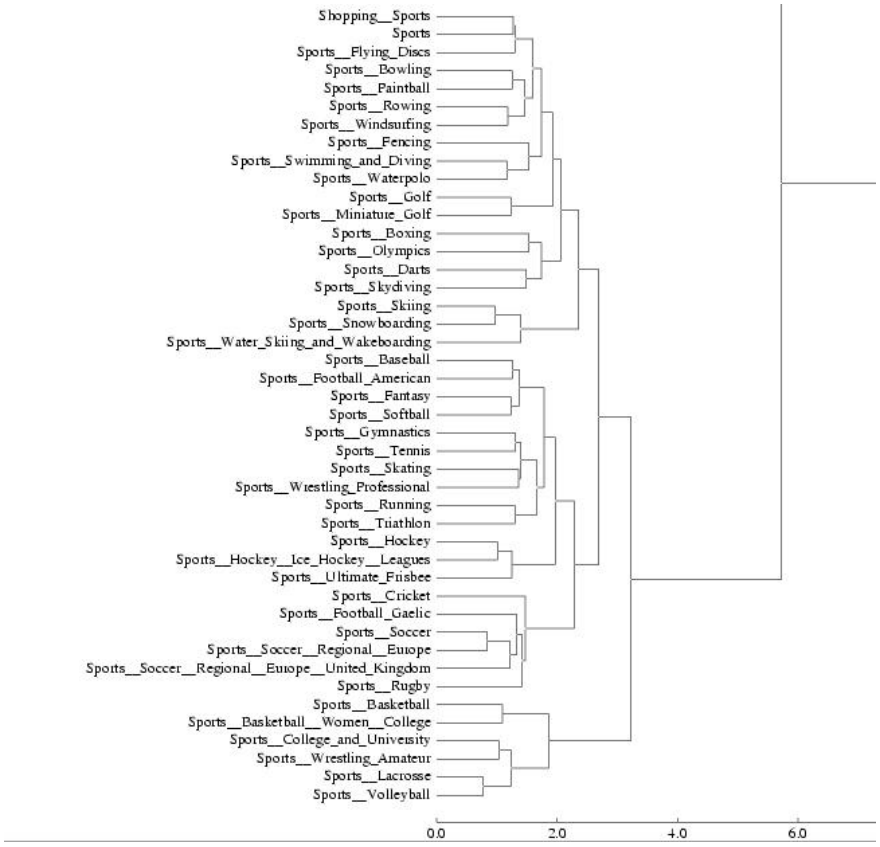


Figure 9: Partial dendrogram for Dmoz 2

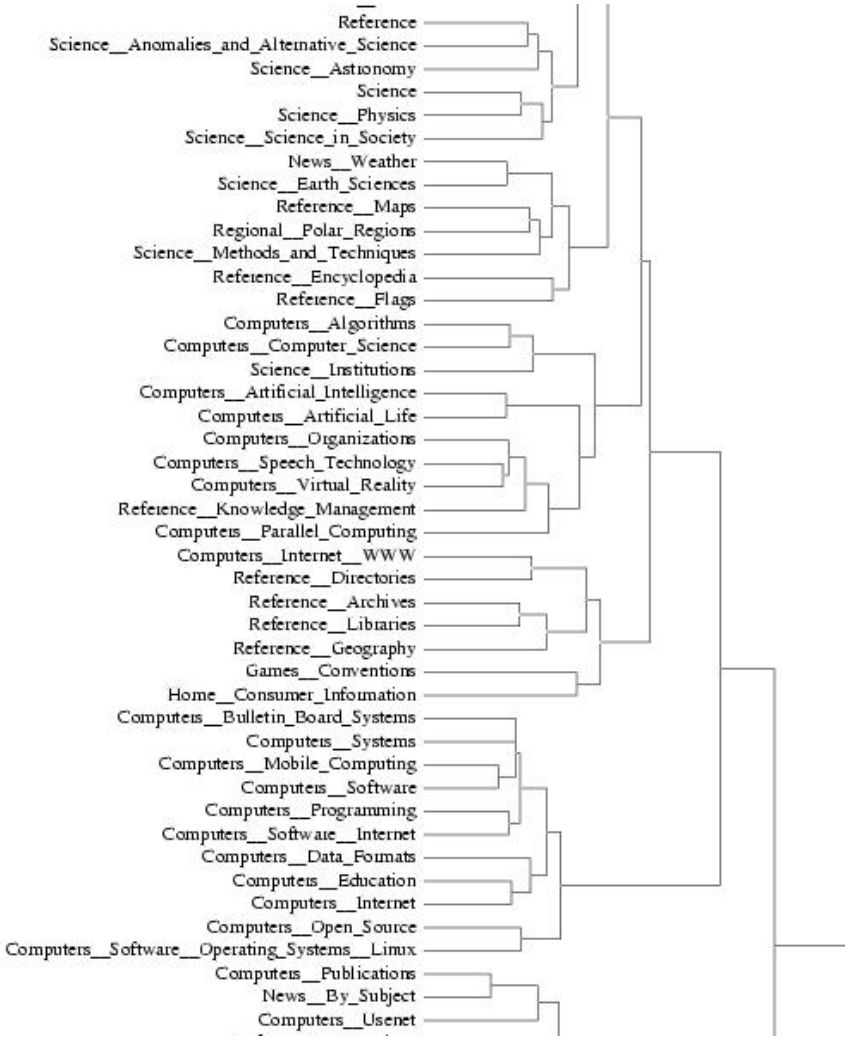


Figure 10: Partial dendrogram for Dmoz 3

Glossary of Acronyms

1. SVM: Support Vector Machine (Section §1.2)
2. NB: Naive Bayes (Section §1.2)
3. *HIClass*: Hyper-interactive text classification (Section §7.3.3)
4. *BayesANIL*: BayesANIL [RCKB05]
5. EM: Expectation Maximisation (Section §2.3.1)
6. EM1D: 1-dimensional EM (Section §3.2.1)
7. EM2D: 2-dimensional EM (Section §3.2.2)
8. SVM-CT: Cross-trained SVM (Section §3.3.1)
9. A&S: Agrawal et al.'s algorithm [AS01] (Section §3.3.2)
10. *GraphSVM*: Graph-based method for scaling SVMs (Section §4.3)
11. *BandSVM*: SVM with band based negative set pruning (Section §5.3.1)
12. *ConfMat*: SVM with confusion matrix based negative set pruning (Section §5.3.2)
13. *SVM-HF*: SVM with heterogeneous feature kernel (Section §5.2)
14. CRF: Conditional Random Field (Section §5.6)

15. TDT: Topic Detection and Tracking (Section §6.2.7)

16. FSD: First Story Detection (Section §6.2.7)

17. HAC: Hierarchical Agglomerative Clustering (Section §6.2.3)

18. *SortPrD*: Sort $Pr(d)$ values method for selecting new class documents (Section §6.2.3)

19. *PrDNewClass*: New class seeded by lowest $Pr(d)$ values (Section §6.2.3)

20. *GenSupp*: Generative method based on support (Section §6.2.3)

21. *NotaSVM*: SVM method for selecting new class documents from all rejects (Section §6.2.4)

22. *DisConf*: Discriminative method based on confidence (Section §6.2.4)

23. NLP: Natural Language Processing (Section §7.2.2)

24. PEBL: Positive Example Based Learning [YHC02]

25. DAGSVM: Directed Acyclic Graph based ensemble for SVMs [PCST00]

26. LBP: Loopy Belief Propagation (Section §5.6

References

[ALJ00] James Allan, Victor Lavrenko, and Hubert Jin. First story detection in TDT is hard. In *Proc. of CIKM*, 2000.

[ALM+03] Henri Avancini, Alberto Lavelli, Bernardo Magnini, Fabrizio Sebastiani, and Roberto Zanoli. Expanding domain-specific lexicons by term categorization. In *Proc. of SAC*, 2003.

[APL98] James Allan, Ron Papka, and Victor Lavrenko. Online new event detection and tracking. In *Proc. of SIGIR*, 1998.

[AS01] Rakesh Agrawal and Ramakrishnan Srikant. On integrating catalogs. In *Proc. of WWW*, 2001.

[ASS00] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proc. of ICML*, 2000.

[AZ04] Rie K. Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. In *Technical Report RC23462, IBM T.J. Watson Research Center*, 2004.

[Bax00] John Baxter. A model of inductive bias learning. In *Journal of Artificial Intelligence Research, 12:149–198*, 2000.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. In *Scientific American, 284(5):34–43*, 2001.

[BM98] Avirm Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of COLT*, 1998.

[BNJ02] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. In *Proc. of NIPS 14*, 2002.

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Car97] Rich Caruana. Multitask learning. In *Machine Learning, 28:41–75*, 1997.

[CDAR98] Soumen Chakrabarti, Byron Dom, Rakesh Agrawal, and Prabhakar Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. In *VLDB Journal, 7*, pages 163–178, 1998.

[CGJ95] David Cohn, Zoubin Ghahramani, and Michael Ivan Jordan. Active learning with statistical models. In *Proc. of NIPS*, 1995.

[Chi05] Manoj Kumar Chinnakotla. Graphical models for multi-labeled classification. In *Technical Report, IIT Bombay*, 2005.

[CL] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. Software available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[CS03] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. In *Journal of Machine Learning Research, 1025–1058*, 2003.

[DB95] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. In *Journal of Artificial Intelligence Research, 2:263–286*, 1995.

[DLR77] Alfred Paul Dempster, Noam Mitch Laird, and David Blei Rubin. Maximum likelihood from incomplete data via the EM algorithm. In *Journal of the Royal Statistical Society, B(39):1–38*, 1977.

[DMDH02] Anhai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic Web. In *Proc. of WWW*, 2002.

[DPHS98] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proc. of CIKM*, 1998.

[EHW86] A. El-Hamdouchi and Peter Willett. Hierarchic document clustering using ward's method. In *Information Processing and Management*, pages 149–156, 1986.

[EW01] A. Elisseeff and J. Weston. Kernel methods for multi-labelled classification and categorical regression problems. In *Technical report, BIOwulf Technologies*, 2001.

[FS99] Yoav Freund and Robert Schapire. A short introduction to boosting. In *Japanese Society for AI 14(5), 771-780. 11*, 1999.

[FSST97] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. In *Machine Learning, vol. 28, 133-168*, 1997.

[GDH04] Evgeniy Gabrilovich, Susan Dumais, and Eric Horvitz. Newsjunkie: providing personalized newsfeeds via analysis of information novelty. In *Proc. of WWW*, 2004.

[Gha00] Rayid Ghani. Using error-correcting codes for text classification. In *Proceedings of ICML*, 2000.

[GHSC04] Shantanu Godbole, Abhay Harpale, Sunita Sarawagi, and Soumen Chakrabarti. Document classification through interactive supervision of document and term labels. In *Proc. of ECML/PKDD*, 2004.

[GM05] Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *Proc. of CIKM*, 2005.

[God02] Shantanu Godbole. Exploiting confusion matrices for automatic generation of topic hierarchies and scaling up multi-way classifiers. In *Technical report, IIT Bombay*, 2002.

[GS04] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Proc. of PAKDD*, 2004.

[GSC02] Shantanu Godbole, Sunita Sarawagi, and Soumen Chakrabarti. Scaling multi-class support vector machines using inter-class confusion. In *Proc. of SIGKDD*, 2002.

[Har04] Abhay Harpale. Hiclass: Hyper-interactive text classification. Master's thesis, School of IT, IIT Bombay, 2004.

[Hay00] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. McGraw Hill, 2000.

[HL01] C. Hsu and C. Lin. A comparison of methods for multi-class support vector machines. In *Technical report, Department of Computer Science and Information Engineering, NTU Taiwan*, 2001.

[Hof99] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proc. of UAI*, 1999.

[JGJS99] Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[Joa] Thorsten Joachims. Svmlight. Software available from `http://svmlight.joachims.org/`.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proc. of ECML*, 1998.

[Joa99] Thorsten Joachims. *Advances in Kernel Methods: Support Vector Learning, Editors B.Scholkopf, C.J.C. Burgess, and A.J. Smola*, chapter Making large-scale SVM learning practical. MIT Press, Cambridge, MA, USA, 1999.

[KJ00] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *Proc. of ICML*, 2000.

[Kre99] Ulrich Kressel. *Advances in Kernel Methods: Support Vector Learning, Editors B.Scholkopf, C.J.C. Burgess, and A.J. Smola*, chapter Pairwise classification and support vector machines. MIT Press, Cambridge, MA, USA, 1999.

[KS97] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proc. of ICML*, 1997.

[LGM+03] David D. Lewis, Rayid Ghani, Dunja Mladenic, Isabelle Moulinier, and Mark Wasson. In *3rd Workshop on Operational Text Classification (OTC), in conjunction with SIGKDD*, 2003.

[LK02] Edda Leopold and Jorg Kindermann. Text categorization with support vector machines. how to represent texts in input space? *Machine Learning*, 46(1-3):423–444, 2002.

[LLYL02] Bing Liu, Wee-Sun Lee, Philip S. Yu, and Xingdong Li. Partially supervised classification of text documents. In *Proc. of ICML*, 2002.

[LMG03] Dan Lizotte, Omid Madani, and Russell Greiner. Budgeted learning of naive-bayes classifiers. In *Proc. of UAI*, 2003.

[LYRL04] D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. In *Journal of Machine Learning Research, 361–397*, 2004.

[McC98] Andrew McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. Software available from `http://www.cs.cmu.edu/~mccallum/bow/`, 1998.

[McC99] A. McCallum. Multi-label text classification with a mixture model trained by em. In *Proc. of AAAI'99 Workshop on Text Learning*, 1999.

[Mit98] Tom Mitchell. Conditions for the equivalence of hierarchical and flat Bayesian classifiers. Technical note, Online at `http://www.cs.cmu.edu/~tom/hierproof.ps`, 1998.

[MN98a] Andrew McCallum and Kamal Nigam. A comparison of event models for naive Bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, 1998.

[MN98b] Andrew K. McCallum and Kamal Nigam. Employing EM in pool-based active learning for text classification. In *Proc. of ICML*, 1998.

[NLM99] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *Proc. of IJCAI-99 Workshop on Machine Learning for Information Filtering*, 1999.

[NMTM00] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[PCST00] John Platt, Nello Cristianini, and John Shawe-Taylor. Large margin dags for multiclass classification. In *Proc. of NIPS*, 2000.

[Pla98] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In *Microsoft Research Technical Report MSR-TR-98-14*, 1998.

[Por80] Martin Fenton Porter. An algorithm for suffix stripping. In *Program, 14(3):130–137*, 1980.

[Ram05] Ganesh Ramakrishnan. *Bridging chasms in text mining through Word and Entity Associations*. PhD thesis, IIT Bombay, 2005.

[RCKB05] Ganesh Ramakrishnan, Krishna Prasad Chitrapura, Raghu Krishnapuram, and Pushpak Bhattacharya. A model for handling approximate, noisy or incomplete labeling in text classification. In *Proc. of ICML*, 2005.

[Rij79] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[Ris95] E. S. Ristad. A natural law of succession. In *Research report CS-TR-495-95, Princeton University*, 1995.

[RMJ05] Hema Raghavan, Omid Madani, and Rosie Jones. Interactive feature selection. In *Proc. of IJCAI*, 2005.

[RPG05] Suju Rajan, Kunal Punera, and Joydeep Ghosh. A maximum likelihood framework for integrating taxonomies. In *Proc. of AAAI*, 2005.

[RR01] Ryan Rifkin and Jason D. M. Rennie. Improving multi-class text classification with the support vector machine. In *AI Memo, AIM-2001-026, MIT*, 2001.

[SCG03] Sunita Sarawagi, Soumen Chakrabarti, and Shantanu Godbole. Cross-training: Learning probabilistic mappings between topics. In *Proc. of SIGKDD*, 2003.

[SD99] Marina Skurichina and Robert P. W. Duin. Regularisation of linear classifiers by adding redundant features. In *Pattern Analysis and Applications, Volume 2, Issue 1*, pages 44–52, 1999.

[SS00] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. In *Machine Learning, 39(2/3):135–168*, 2000.

[TK00] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. In *Proc. of ICML*, 2000.

[TO96] Sebastian Thrun and Jeremy O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *Proc. of ICML*, 1996.

[Vap95] Vladimir Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1995.

[Yan01] Yiming Yang. A study of thresholding strategies for text categorization. In *Proc. of SIGIR*, 2001.

[YHC02] H. Yu, J. Han, and Kevin C. Chang. Pebl: Positive example-based learning for web page classification using svm. In *Proc. of SIGKDD*, 2002.

[YPC98] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Proc. of SIGIR*, 1998.

[YZCJ02] Yiming Yang, Jian Zhang, Jaime Carbonell, and Chun Jin. Topic-conditioned novelty detection. In *Proc of SIGKDD*, 2002.

[ZE02] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proc. of SIGKDD*, 2002.

[ZJXG05] Shenghuo Zhu, Xiang Ji, Wei Xu, and Yihong Gong. Multilabelled classification using maximum entropy method. In *Proc. of SIGIR*, 2005.

[ZL04] Dell Zhang and Wee Sun Lee. Learning to integrate web taxonomies. In *Journal of Web Semantics, 2(2)*, pages 131–151, 2004.

[ZY03] J Zhang and Yiming Yang. Robustness of regularized linear classification methods in text categorization. In *Proc. of SIGIR*, 2003.

Publications and Tools and Utilities Resulting from the Work

Publications

1. Text Classification with Evolving Label-sets
   Shantanu Godbole, Ganesh Ramakrishnan, Sunita Sarawagi, in Proceedings of The Fifth IEEE International Conference on Data Mining (ICDM), 2005, New Orleans, Louisiana, U.S.A.

2. Document classification through interactive supervision of document and term labels
   Shantanu Godbole, Abhay Harpale, Sunita Sarawagi, Soumen Chakrabarti, in Proceedings of The 15th European Conference on Machine Learning (ECML) and the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), 2004, Pisa, Italy.

3. HIClass: Hyper Interactive text Classification by interactive supervision of document and term labels
   Shantanu Godbole, Abhay Harpale, Sunita Sarawagi, Soumen Chakrabarti, in Proceedings of The 15th European Conference on Machine Learning (ECML) and the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), 2004, Pisa, Italy.

4. Cross-Training: Exploiting probabilistic mappings between topics
   Sunita Sarawagi, Soumen Chakrabarti, Shantanu Godbole, in Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2003, Washington D.C., USA.

5. Discriminative methods for multi-labeled classification

Shantanu Godbole, Sunita Sarawagi, in Proceedings of The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2004, Sydney, Australia.

6. Scaling multi-class Support Vector Machines using inter-class confusion
Shantanu Godbole, Sunita Sarawagi, Soumen Chakrabarti, in Proceeding of The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002, Edmonton, Canada.

Tools and Utilities

1. Tool for generating dendrograms given a corpus with a flat set of classes
2. Browsable interface to show mappings between DMOZ and Yahoo! datasets used in Chapter 3. Online at `http://www.it.iitb.ac.in/~shantanu/ctdemo/`
3. Common experimental framework for text classification experiments reporting a variety of commonly used measures
4. Module to display new classes in unlabeled data not present during training
5. Abstraction based novel representation of documents to understand classification criteria
6. *HIClass* - an interactive text classification workbench containing many of the algorithms and tools described in Chapter 6 and Chapter 7

Platform for Implementation

Most of the algorithms developed as part of our work were implemented in C/C++ programs and Perl scripts. Most experiments were performed on a dual 1.3GHz Xeon server running Debian Linux. In some cases CPU intensive experiments of scale were performed on a high-end DEC Alpha server. We used open source software and libraries wherever possible.

Acknowledgement

December 7, 2006

I would like to start by acknowledging my advisors *Sunita Sarawagi* and *Soumen Chakrabarti* for being a constant source of inspiration and awe, without whose support none of this work would have been possible. I would also like to thank Prof. S. Sudarshan and Prof. N. L. Sarda who were in my research progress committee (RPC) and whose timely guidance and comments proved very valuable. I would also like to thank all my professors at IIT Bombay for teaching me all that I know; most importantly to be good and honest. I would like to thank my very large group of friends here at IIT Bombay - my wingmates, batchmates, department juniors, and many more.

In particular I would like to thank Ganesh Ramakrishnan, my friend, study partner, and luckily now, colleague at my new workplace for the innumerable discussions, criticisms, and inspiration he instilled into me. I would also like to thank my graduate colleagues and friends; Srinath, Vikram, Raghu, Nitin, Manoj, Sameer, Rahul, Ashish, Punit, Shruti, and Sumana for being a great bunch of folks to learn with and have chai with. My stay at IIT was made special by the many groups I was part of, in particular, SIGFood - thanks for all the fish. A special thanks to Francesco Bonchi, my Italian friend, whom I have met on three continents in the world; we have four more to go my friend.

I was an Infosys Research Fellow for most of the duration of my graduate studies. I would sincerely like to thank Infosys Technologies, Bangalore for their financial support. Last, but not at all the least, I would like to thank my parents and Sucheta for standing by me in everything.

Shantanu Godbole

Index