Nikola Mrkšić

Semi-supervised Learning Methods for Data Augmentation

Computer Science Tripos

Trinity College

May 15, 2013

Proforma

Name:	Nikola Mrksic
College:	Trinity College
Project Title:	Semi-supervised learning methods for data augmentation
Examination:	Computer Science Tripos, Part II, 2013
Word Count:	11926
Project Originator:	Dr Sean Holden
Supervisor:	Dr Sean Holden

Original Aims of the Project

The original goal of this project was to investigate the extent to which data augmentation schemes based on semi-supervised learning algorithms can improve classification accuracy in supervised learning problems. The objectives included determining the appropriate algorithms, customising them for the purposes of this project and providing their Matlab implementations. These algorithms were to be used to develop a robust system for achieving data augmentation in arbitrary application areas. For evaluation purposes, a general framework for assessing the quality of data augmentation achieved was to be constructed.

Work Completed

The project met and exceeded all of the success criteria. A survey of theoretical results underlying data augmentation has been conducted. Full, general implementations of Bayesian Sets, Spy-EM and Roc-SVM algorithms, as well as their proposed extensions have been implemented. A general scheme for achieving data augmentation in binary and multi-class classification has been developed and successfully applied to the three application areas proposed. An evaluation framework for assessing the quality of data augmentation was implemented and used to give statistical significance to the results obtained.

Special Difficulties

None.

Declaration

I, Nikola Mrksic of Trinity College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

I give permission for my dissertation to be made available in the archive area of the Laboratory's website.

Signed

Date

Contents

Acknowledgements

This project was completed under supervision of Dr Sean Holden, whom I would like to thank for his invaluable assistance, support and guidance throughout my studies. I would also like to thank Dr Arthur Norman, Dr Simone Teufel, Professor Jon Crowcroft and Professor Katherine Heller for all the helpful advice and suggestions they provided.

Chapter 1

Introduction

In this chapter, the task of data augmentation is presented and framed as a semi-supervised machine learning problem. The necessary theoretical assumptions are outlined and a proof of utility of unlabelled data for improving classification is provided.

1.1 Supervised Learning

Supervised learning algorithms generate a function $f : X \to Y$ that maps potential inputs to desired outputs, also known as labels. f maps any $x_i \in X$ to a specific label $y_i \in Y$. The algorithm is trained using a list of input-output pairs (x_i, y_i). If the set of labels Y is finite, the problem is known as classification; otherwise, the problem is known as regression.

The basic assumption of supervised learning is that the (x_i, y_i) pairs are independent and identically distributed random variables drawn from an underlying probability distribution on $X \times Y$. The function f obtained by learning from the training data tries to capture the probability density function of this distribution. The main condition required for supervised learning to work (that is, to be able to generalise from a finite training set to infinitely many unseen test cases) is that the *smoothness assumption of supervised learning* holds. Informally, it requires that if the points x_1 and x_2 are *close*, their corresponding outputs (labels) y_1 and y_2 should be *close* as well.

1.2 Semi-supervised Learning

Classifier performance is highly dependent on the nature of the data to be classified. Across different application areas, the prediction quality correlates with the amount of labelled data used for training the classifier. In applications ranging from text classification and speech recognition to different biomedical applications, gathering unlabelled data is usually cheap and straightforward: large amounts of text can be gathered online, speech can be recorded automatically and protein sequences can be acquired at industrial speeds. This training data has to be labelled either by a human expert or through a set of physical experiments, making the acquisition of a *sufficiently* large labelled training set expensive, time consuming, and often infeasible as a result.

Due to the abundance of the unlabelled data available, we are interested in how this data can be used to improve the accuracy of the classifiers produced. This is the task of semi-supervised learning.

Semi-supervised learning algorithms use two different data sets for training: $X_l = ((x_1, y_1), ..., (x_l, y_l))$, the set of inputs with labels available and $X_u = (x_{l+1}, ..., x_{l+u})$, the set of unlabelled data. Given these sets as input, the algorithm tries to build the same prediction function as in the case of supervised learning. Unlabelled data can boost prediction quality only if the distribution of $p(x)$ observed from the unlabelled data carries information applicable to the inference of $p(y|x)$.

The *semi-supervised smoothness assumption* formalises this notion:

If x_1 and x_2 are points *close* to each other in a *high density region*, their corresponding outputs y_1 and y_2 should be *close* as well. By transitivity, points from the same *cluster* are likely to have their respective outputs clustered as well.

1.3 Data Augmentation

The performance of supervised learning algorithms improves with the size of the data set used for training the classifier. In application areas containing a limited amount of labelled, but a large amount of unlabelled data, *data augmentation* schemes are used to create larger (*augmented*) training data sets by incorporating some of the unlabelled examples into the labelled data set.

For each of the classes (labels) present in the data set, unlabelled examples most likely to belong to that class are identified. These examples, with their anticipated labels, become part of the labelled training set. Then, this *augmented* data set is used to retrain the classifier (for example, a Support Vector Machine, presented in Chapter 2), in the hope that the data introduced may reduce the *classification error* of the classifier built.

In general, constructing data augmentation schemes which result in both simple and fast algorithms is a highly non-trivial task, as successful strategies vary greatly with the nature of data present in different applications [12].

Under certain assumptions about the nature of the data model, namely that all of the examples were generated using the same *mixture model* and that there exists a one-to-one correspondence between the generative components of this model and the labels present in the data set, it is possible to show that unlabelled data does *carry information* about the parameters of this mixture model. A full discussion and the proof of this claim are given in Appendix A.

This result does not yield a mechanism for achieving a reduction in classification error using unlabelled data, which is what the principle goal of data augmentation is. In general, it is not always the case that this reduction can be achieved: given an infinite amount of labelled data, an optimal classifier can be built without using any unlabelled data.

Building upon the result derived in Appendix A, one can show that with a finite amount of labelled and an infinite amount of unlabelled data, definite improvement can be achieved [3]: the classification error approaches the optimal one exponentially fast as the size of the labelled set increases. Little is known about the case when the sets of labelled and unlabelled data are both finite. Theoretical arguments relevant for understanding the fundamental limits of data augmentation are investigated in Chapter 3 and Appendix A.

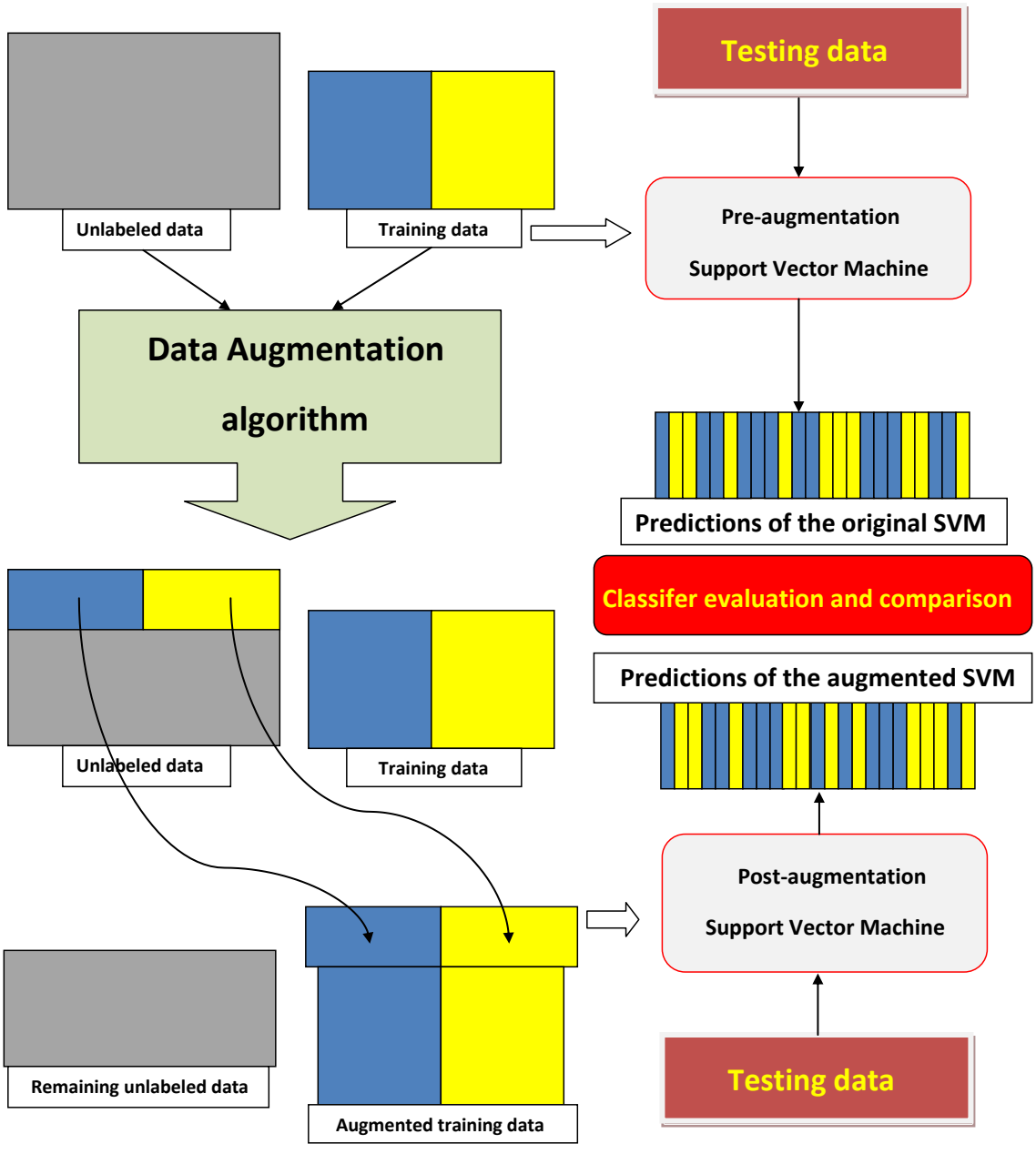


Figure 1.1: Flow diagram of the data augmentation process. Additional training examples are extracted from the unlabelled data and used to re-train the SVM.

Chapter 2

Preparation

This chapter presents most of the research and planning conducted before the implementation of this project started. It contains my starting point, the rationale behind choices made in the implementation, an outline of the software engineering paradigm used and a description of the algorithms used for classification.

2.1 Starting Point

At the start of the project, I had very limited knowledge of machine learning. All the algorithms used and the computational learning theory presented were learned and absorbed throughout the course of this project. The original project idea proposed the use of the Bayesian Sets algorithm for achieving data augmentation. The theoretical results presented, as well as the use of *Spy-EM* and *Roc-SVM* algorithms are my own refinements to the project proposal, identified through a thorough survey of the associated literature.

I had no knowledge of Matlab prior to starting work on this project, and no knowledge of any experimental methods used in the subsequent evaluation.

2.2 Requirements Analysis

The project proposal outlined the idea of using different semi-supervised machine learning algorithms to achieve data augmentation across different application areas. The project consisted of three principle components:

1. Understanding the theoretical boundaries of the reduction in classification error achievable through data augmentation.
2. Identifying and implementing algorithms and statistical techniques applicable to data augmentation.
3. Establishing a uniform criterion for measuring the success of different data augmentation schemes across different application areas and implementing a system which would empirically determine the extent to which data augmentation can aid classification.

2.2.1 Theoretical Background

The first step in refining the project proposal was to identify the theoretical results that could allow us to more effectively determine which algorithms are likely to be most effective at boosting classifier accuracy. To this extent, a broad literature survey was conducted [3, 4, 7, 11, 12]. Understanding the material required familiarisation with the relevant aspects of computational learning theory, including the probably approximately correct (PAC) learning framework and the use of Vapnik Chervonenkis (VC) dimensions. Results most relevant to my work are presented in Chapter 3 and Appendix A.

2.2.2 Algorithms for Achieving Entity Set Expansion

The task of *entity set expansion* deals with extracting additional examples of each class from the unlabelled data. Initially, the following algorithms were proposed for achieving entity set expansion:

- The Bayesian Sets algorithm [6] was envisioned as the principle set expansion method; it was to act as the representative of the *ranking methods*.
- Spy Expectation Maximisation algorithm (*Spy-EM*[10, 11]) was proposed as an extension, acting as the representative of the *classification methods*.

Further consideration of the related work revealed another *potential extension*: the *RocSVM* algorithm [8], a method based on a paradigm akin to *Spy-EM*, but applicable to real-valued input data (unlike *Spy-EM*, which supports only binary data). Implementing *RocSVM* was incorporated into the project goals, as it promised the potentially most powerful [8] and general approach to performing entity set expansion.

2.2.3 Choosing the Benchmark Classifier

For the sake of meaningful evaluation of the quality of data augmentation achieved, a *uniform measure* of the improvement in classification accuracy achieved through data augmentation had to be established. The simplest, most elegant solution was to use the same classifier across all application areas.

A survey of relevant literature [1] suggested that support vector machines (SVMs) could provide the benchmark classifier. SVMs are applicable to most classification and regression problems. Using SVMs across all application areas was a rational decision for the sake of this project: we are interested in improving the performance of *arbitrary classifiers* through data augmentation, not in building optimal classifiers through domain specific knowledge.

The alternative approach was to use classifiers specifically tailored for each of the application areas considered. This would provide insights into using very specific classifiers in very specific applications, restricting our ability to compare the performance of a data augmentation scheme across different application areas.

SVMs are a very powerful tool applicable to a wide range of problems in machine learning. In addition to the practical consideration of allowing consistent evaluation, using *SVMs* was beneficial for the following reasons:

- A technique applicable to improving SVM performance in any setting would be highly reusable: it would provide a wrapper method for boosting the performance of any (existing and new) supervised learning application which has a body of unlabelled data available to it.
- SVM performance is well understood and reliable library implementations exist (*LIBSVM*, SVM^{light}). Choosing SVMs as our means of classification minimised the risk of this component becoming a bottleneck in developing the final data augmentation evaluation system.

The principal measure of data augmentation quality will be the improvement in classification accuracy achieved by training the SVM using the augmented data set produced by our data augmentation system (Figure 1.1).

2.3 Application Areas Considered

The extent to which data augmentation can aid classification was investigated in three different application areas. The first two deal with *two-group classification* of instances consisting of *binary features*, whereas the third data set consists of *continuous features* and deals with *multi-class classification*.

2.3.1 Text Classification

Reuters21578 text categorisation collection contains Reuters news documents from 1987, labelled manually by Reuters personnel. The total number of categories is 672, but many of them occur very rarely.

For the purpose of the experiments, a subset of this data set representing feature vectors of the two most frequent labels was used. We are dealing with binary classification of feature vectors consisting of discrete binary data: each vector contains 18933 binary features, each indicating the presence of some word in the given article.

Both categories are abundant in the training data, with an approximate $3 : 2$ class ratio. The training set T_r consists of 4108 articles and the testing set T_s consists of 1660 articles; the unlabelled set U is sampled from T_r. The class distributions in the training and testing sets are the same. Consequently, T_r, T_s and U have the same class distributions.

2.3.2 Prediction of Biomolecular Activity for Drug Design

KDD Cup 2001, Task 1: 'Binding to Thrombin' data set, produced by DuPont Pharmaceuticals Research Laboratories, is related to drug design.

Drugs are usually small organic molecules. The first step in designing a new drug is to identify and isolate the receptor to which the drug should bind; in this case, the Thrombin site. Subsequently, many small molecules are tested to identify those able to bind to that site: these are known as the *active compounds*.

The data set contains examples of both active and inactive compounds: a list of binary attributes is provided for each molecule, indicating whether its structure possesses a certain three-dimensional property. Our goal is to build a classifier able to identify active compounds given only the molecules' structural properties.

This task is no different from the text classification one: it deals with binary classification of examples consisting of binary attributes. However, it is much more challenging for the following reasons:

1. There is a great imbalance between active and inactive compounds in the training set: only 42 of the 1909 examples represent the active ones.
2. The number of attributes is *very large*: $139,351$ binary features are used to represent each compound.
3. The data set exhibits *selection bias*: the distribution of active and inactive compounds in T_r and U differs from the distribution encountered in the testing data set T_s.

2.3.3 Supervised Learning for Theorem Proving

In this problem, supervised learning is used to guide an automated theorem prover. The data set was available from a recent research project in the Computer Laboratory [2]. The prover can use five different heuristics to try to prove a first-order logic statement. Given attributes of these statements such as the number of variables per logical connective or the average number of times that variables are repeated in the statement, we want to help the prover decide which of the five heuristics would be best for tackling that given statement [2].

The training examples represent statements labelled with the numerical identifier of the heuristic which proved them the fastest. The unlabelled data is the set of problems that none of the heuristics could solve within a certain time limit. A major distinction from the first two application areas is that the attributes describing the examples to be classified are no longer binary, or even discrete: they are now represented by continuous (real-valued) data.

Examples are now classed into one of five different categories. Our goal is to achieve a reduction in the classification error by using examples from U to augment each of the five labelled sets used for training.

There were three (new) major challenges associated with this task:

1. The attributes are *continuous*, not binary, making the *Spy-EM* algorithm inapplicable. Bayesian Sets provide no efficient algorithm for real-valued data [6]. Thus, *Roc-SVM* was the only admissible method: implementation of all project extensions was a *prerequisite* for tackling this problem.

2. Instances belong to five different categories: in order to consider this application area, a *multi-class* classification scheme had to be developed.

3. Unlabelled data does not have the same distribution as the training data: U is the set of statements that the theorem prover *did not prove*. It is questionable whether these examples contain any information applicable to discriminating between the five heuristics.

2.4 Project Methodology

The target system consisted of four principle components (Figure 2.1). Project goals revolved around the application of the three data augmentation schemes to the three application areas proposed, and different classification schemes had to be used for binary and multi-class classification. The exact work schedule was highly unpredictable as the implementation of the more advanced schemes depended on success in earlier stages.

The ambitious nature of the goals presented required careful planning to ensure that the primary success criteria were fulfilled notwithstanding potential complications with the more advanced goals such as the *Roc-SVM* algorithm or the augmentation of the theorem prover data.

Roc-SVM and *Spy-EM* can be very computationally demanding: processing the enormously large biomedical data set promised to be borderline intractable. Thus, having the entire system able to run at least some of the data augmentation schemes early on was instrumental for obtaining all the results in time.

Taking all of these requirements into account, the development paradigm chosen for this project was *evolutionary prototyping* [5].

2.4.1 Evolutionary prototyping

Evolutionary prototyping is a form of software prototyping that builds each prototype in a quality manner: the process includes a requirements specification, design documentation and thorough testing. In each development cycle, only the *well understood* requirements are implemented. Subsequently, the prototype is used experimentally to shed more light on the remaining requirements (those less understood), as well as to identify new requirements.

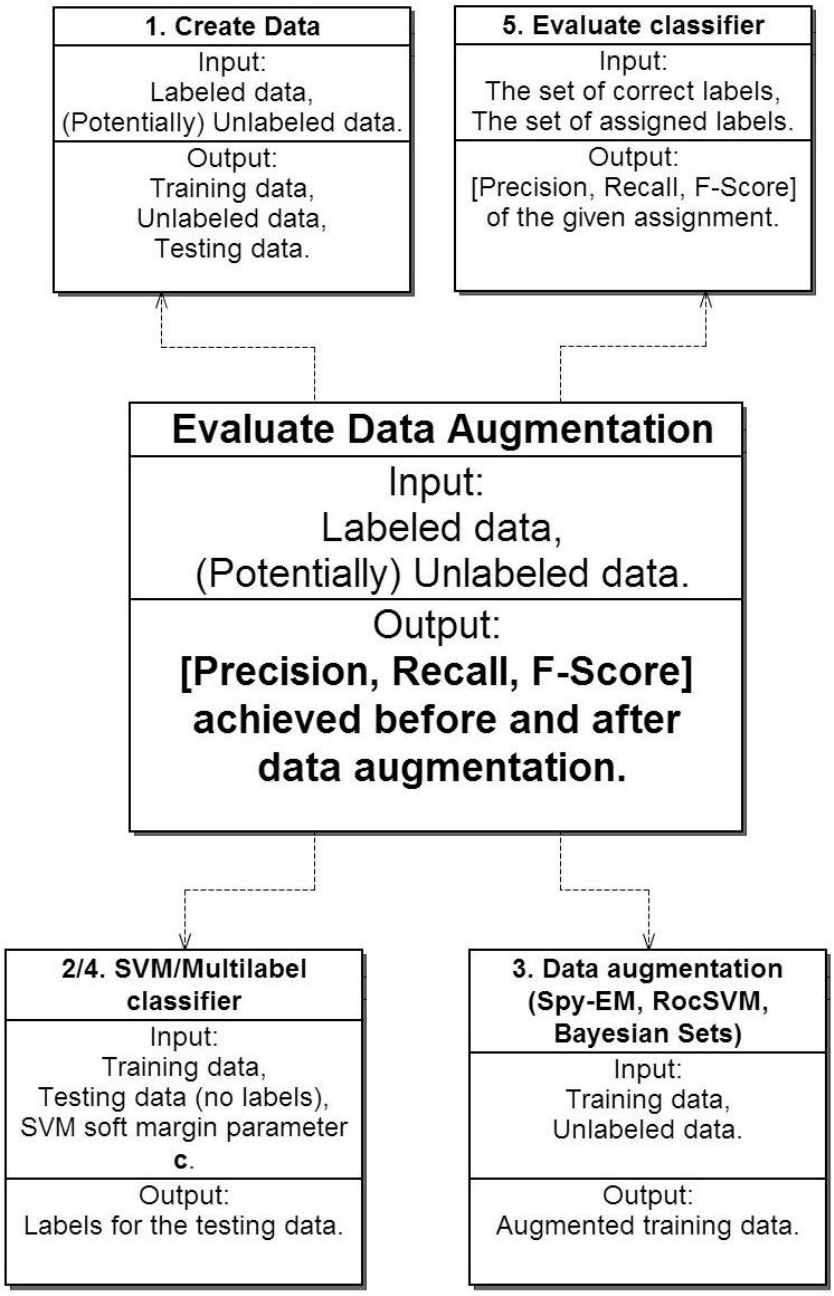


Figure 2.1: Component diagram of the data augmentation evaluation system.

Evolutionary prototyping delivers incremental *functional systems*, with each iteration providing a system meeting a set of *well-defined* requirements. This paradigm was ideal for the needs of this project, as it catered perfectly to our need to start obtaining results about simpler schemes and simpler application areas while work on the more complicated ones was still under way.

More importantly, the constant requirements feedback warranted by this method proved to be instrumental in providing enough versatility and early warning signals to streamline my efforts towards a smooth and successful completion of all the success criteria and all of the proposed extensions.

2.5 Relevant Algorithms

This section presents two of the algorithms used as building blocks for the data augmentation schemes presented in the Implementation. Existing implementations of support vector machines and Naive Bayesian classifiers available from Matlab's Machine Learning toolbox were used in this project.

2.5.1 Naive Bayes Classifier

In statistics and machine learning, classification refers to the problem of identifying to which category (of the given set of categories) a new observation (also known as *instance* in machine learning) belongs to, based on the training data which consists of observations for which the categories (classes) are known. Instances are characterised by a vector of explanatory variables known as *features*.

The probability model for the classifier is represented by the conditional distribution $\Pr(C \mid F_1, ..., F_n)$; C represents the set of classes and $F_1 \dots F_n$ stand for the features of the examples.

The Naive Bayes classifier makes the assumption of *conditional independence between features* of the observations: for any features F_i and F_j, it must hold that $\Pr(F_i \mid C, F_j) = \Pr(F_i \mid C)$. Even though this is not the case in most data sets, it is a common assumption and it often results in very good performance. This assumption is useful as it can be used to simplify the probability model.

The expression for conditional probability can be rewritten using Bayes' theorem:

$$\Pr(C \mid F_1 \dots F_n) = \frac{\Pr(C) \Pr(F_1 \dots F_n \mid C)}{\Pr(F_1 \dots F_n)} \tag{2.1}$$

The denominator represents the normalisation constant, which can be omitted.

Repeatedly unfolding the definition of conditional probability (and omitting all the normalisation factors) yields:

$$\frac{\Pr(C)\Pr(F_1 \dots F_n)}{\Pr(F_1 \dots F_n)} \propto \Pr(C)\Pr(F_1 \mid C)\Pr(F_2 \dots F_n \mid F_1, C)$$

$$\propto \quad \Pr(C)\Pr(F_1 \mid C)\Pr(F_2 \mid C, F_1)\Pr(F_3 \dots F_n \mid F_1, F_2, C)$$

$$\propto \quad \Pr(C)\Pr(F_1 \mid C)\Pr(F_2 \mid C, F_1)\Pr(F_3 \mid C, F_1, F_2) \dots \Pr(F_n \mid F_1 \dots F_{n-1}, C)$$

Applying the assumption of conditionally independent features to the final expression yields:

$$\Pr(C \mid F_1 \dots F_n) = \frac{\Pr(C)\Pr(F_1 \dots F_n \mid C)}{\Pr(F_1 \dots F_n)} \propto \boxed{\Pr(C)\prod_{i=1}^{n}\Pr(F_i|C)} \tag{2.2}$$

The *prior* for this model, $\Pr(C)$, can be estimated by observing the frequency of each class in the training data available. The conditional probability distributions of each feature, $\Pr(F_i \mid C)$ depend on the joint probability distribution $\Pr(F_i, C)$, as $\Pr(F_i \mid C) = \Pr(F_i, C)/\Pr(C)$. These can be estimated from the training data by dividing the number of occurrences of each *(feature, class)* pair with the (already estimated) prior of that class (the *bag of words* approach).

Naive Bayes classifiers have been successfully used in a wide range of applications. Their popularity stems from their practicality: the decoupling of features allows one to estimate each feature's probability distribution independently and thus avoid the *curse of dimensionality*, which occurs with many data models that scale exponentially with the number of features.

2.5.2 Support Vector Machines

Support vector machines are a very popular supervised learning model used for two-group classification and regression analysis. [15]

Given a set of training instances, the SVM training algorithm builds a model that assigns new examples into one category or the other. The SVM model represents instances as points in a high-dimensional space, mapped in such a way that the

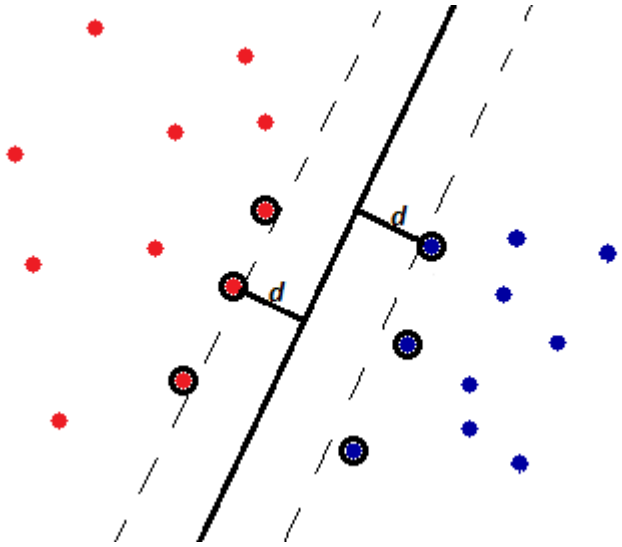


Figure 2.2: The optimal decision line is as far from both classes' points as possible [1].

points representing instances of the two categories are separated from each other by a gap that is as wide as possible. SVMs construct a hyperplane (or a set of hyperplanes) in that high-dimensional space in order to split the two classes into two distinct subspaces (Figure 2.2). Subsequently, new examples are classified according to the subspace of the model space that they are mapped to.

Support vector machines are a direct product of work done in *computational learning theory*. If f denotes the target classifier, let $L(f((x)), y)$ be the function that measures its error of prediction (the *loss function*): this can be the square error function, the negative log marginal likelihood function, etc. Then, one can obtain the parameters of the optimal decision function by minimising the *expected error of classification* on all data:

$$R(f) = E[L(f(x), y] = \int_y \int_x L(f(x), y) \Pr(\mathbf{x}, y) dx dy \tag{2.3}$$

As $\Pr[\mathbf{x}, y]$ is not available, $R(f)$ can not be computed. Instead, it is replaced with the sum of errors across all training data (the *empirical risk function*).

Out of all classifiers that separate the training data, SVMs choose the optimal one by *maximising* the *minimal* distance from the set of all training points (irrespective of class membership) to the separating hyperplane (Figure 2.2). This principle is known as *empirical risk minimisation*. A relatively small number of data points known as *support vectors* are required to determine the exact position of the optimal decision hyperplane.

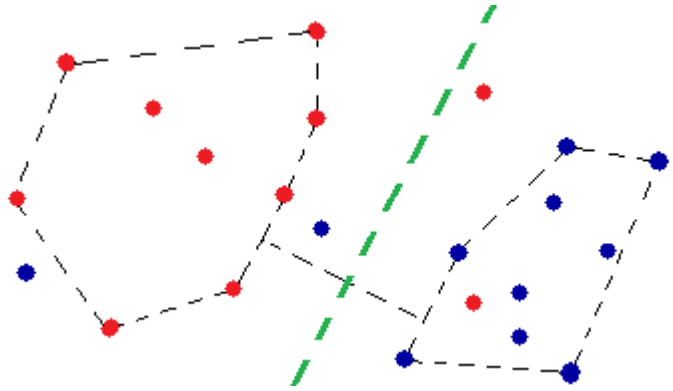


Figure 2.3: Soft margin SVM: here, outlier points are 'ignored'; the decision plane still maximises the distance from the convex hulls of the *reduced* sets of the two classes [1].

In real world data sets, it is rarely the case that the separating hyperplane can perform error-free classification of the training data. There are two different approaches to tackling these *non-separable* data sets:

Soft margin classification: if the data is not linearly separable, we might still try to split the majority of the data points *as cleanly* as possible, using the same approach as for the separable data. If no separating hyperplane exists, the *soft margin method* will allow *some of the examples* to be misclassified (Figure 2.3).

Non-linear SVMs are the alternative approach. These use the *kernel trick* to map the initial (non-separable) data points onto a higher-dimensional space where they become separable (Figure 2.4). This mapping is achieved by a non-linear kernel function such as the Gaussian radial basis function or the hyperbolic tangent.

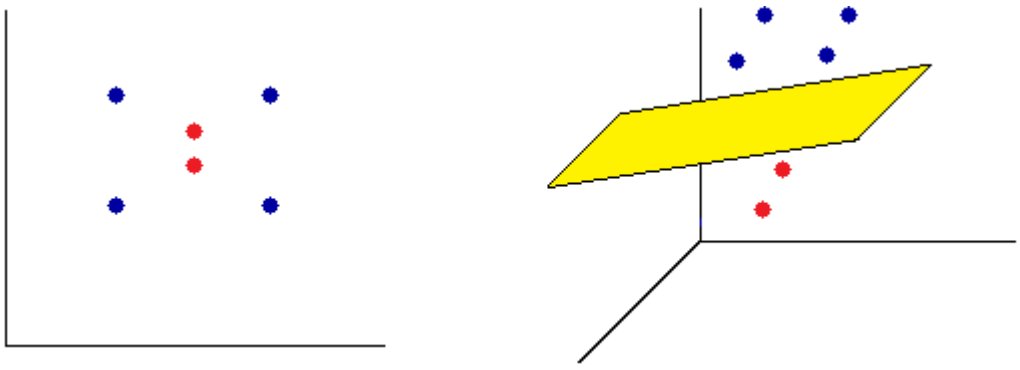


Figure 2.4: Using the *kernel trick* to separate an inseparable 2D training set [1].

2.6 Languages and Tools

Matlab was deemed to be the most suitable language for implementing the algorithms required for this project. Furthermore, Matlab is omnipresent in machine learning, so this decision facilitated subsequent reuse of these algorithms (of which there are no open-source implementations available) by the research community.

Throughout the course of the project, a number of additional tools were used:

1. C++ was used to pre-process the data sets used.
2. Subversion, Tortoise SVN and Google Drive were used for version control.
3. TeXnicCenter was used to write the dissertation.
4. GeoGebra, MS Word and Paint were used to draw the diagrams for the dissertation.
5. Microsoft Excel was used for part of the evaluation work.

Chapter 3

Implementation

This chapter presents the data augmentation system created, focusing on the partially supervised algorithms developed for the problem of entity set expansion and their subsequent use for data augmentation. Theoretical arguments behind the implementation choices made are analysed and the specific requirements of each of the application areas are considered.

3.1 Overview

The implementation work undertaken for this project consisted of solving two separate problems: *entity set expansion* and *data augmentation*.

Entity set expansion: given a set S of seed entities of a particular class S and a set D of candidate entities, determine which of the entities in D belong to S. In other words, *expand* the set S based on the given seeds.

This part of the project consisted of identifying and implementing algorithms for achieving entity set expansion, as well as understanding the theoretical assumptions required in order for these algorithms to work. Three different algorithms were implemented: Bayesian Sets, *a ranking method*, and *Spy-EM* and *Roc-SVM*, which are *classification methods*.

Data augmentation: given a supervised machine learning problem with unlabelled data available to it, label some of the unlabelled examples and add them to the training set so that the use of this new, augmented data set boosts the accuracy of the classifier produced.

The target system had to use the set expansion algorithms to achieve data augmentation and then measure how successful this augmentation was across three different application areas.

The two problems are interdependent. Entity expansion could be regarded as a stand-alone problem; however, the challenges faced in using these algorithms for data augmentation provided valuable feedback for the subsequent direction of their development, in accordance with the evolutionary prototyping paradigm adopted. For the sake of clarity, the work on the problem of entity set expansion will be covered first. Subsequently, the way in which these algorithms were used to achieve and assess data augmentation will be presented.

3.2 Partially Supervised Learning

Partially supervised learning algorithms are a special class of semi-supervised learning algorithms. They learn from a set of *positive examples* P and a *mixed set* of *unlabelled examples* U (hence the term PU learning). U is implicitly assumed to contain both positive and negative examples. The key characteristic of PU learning is that *no negative training examples* are available for learning. This is in contrast to the typical classification setting, in which the training data contains instances of *every possible class*.

Partially supervised learning algorithms create binary classifiers that decide whether an instance belongs to the positive set P or not. These algorithms are ideal building blocks for our data augmentation scheme, as the classifiers built can be used to extract the *hidden positives* from U. If the training set T_r consists of different classes P_i ($T_r = \bigcup_i P_i$), each of these is separately treated as the positive set and expanded using examples from U.

My major concern was that the knowledge about the negative sets (for class i, $N_i = T_r \setminus P_i$) was not used for entity set expansion. These negative sets will be utilised during the later data augmentation stages to *purify* the augmented sets, by minimising the number of wrongly labelled examples introduced.

In the face of imbalance in the distribution of negative data in the training and testing sets (as is the case with our biomolecular data and many real world data sets), the use of negative data can degrade the quality of classification [9]. In these settings, PU learning offers a superior approach to traditional binary

classification [9]. To ensure that our data augmentation scheme was as general as possible, entity set expansion was treated as a partially supervised problem.

3.2.1 Theoretical Foundations

Prior to deciding to base data augmentation on partially supervised learning, a detailed investigation of relevant work was conducted: [3, 4, 7, 11, 12]. The theoretical results used to justify the decision to use PU learning for achieving data augmentation are presented in *Appendix A*:

Partially supervised classification is treated using the *probably approximately correct* (*PAC*) framework to show that a classifier with an *arbitrarily low* classification error can be produced given *sufficiently large* sets of positive and unlabelled examples. This result is by no means trivial, as *no labelled negative data* is introduced at any point. Indeed, it comes as a surprise that a classifier deciding membership of P can approach *perfect precision* and *perfect recall* without using *any negative examples* for training.

Detailed discussion and proofs of these claims are given in *Appendix A*. The theorem presented there yields sufficient conditions on the sizes of P and U required so that partially supervised learning can (in theory) produce an *arbitrarily accurate* classifier with *very high probability*. This result provides the theoretical foundation to all the approaches to entity set expansion employed in this project. Together with the theoretical grounding of the utility of unlabelled data (also presented in Appendix A), it completes the computational learning theory argument showing that data augmentation, as framed and implemented in this project, promises (at least in theory) to be a well-founded scheme that can aid classification through the use of unlabelled data.

These proofs rely on many assumptions. It is unclear whether (or if at all) these assumptions hold in any of the application areas proposed. The theory presented yields no constructive (or efficient) method for implementing PU classification or using the constructed classifiers to achieve data augmentation.

Henceforth, the discussion moves away from the theoretical aspects and focuses on the algorithms used, the data augmentation scheme developed and the results obtained in the suggested application areas. As will be shown in the experiments, schemes powered by PU learning can be successful at data augmentation, thus providing empirical support for the theoretical arguments presented.

3.3 Spy Expectation Maximisation Algorithm

The *Spy-EM* algorithm was first proposed for the task of partially supervised classification by Liu et al. in [11]. It is a heuristic technique based on Naive Bayes classifiers and the *expectation-maximisation* (*EM*) algorithm [11].

Expectation-maximisation is an iterative method for finding *maximum likelihood parameters* for statistical models based on incomplete data. The expectation step fills in the missing values using estimates of their expected values based on existing data and the current estimates of the parameters. The maximisation step calculates the parameters most likely to have generated the values obtained in the expectation step. The two steps are repeated until the parameters converge.

In partially supervised classification, available data corresponds to the positive set P and the missing data corresponds to U. Let c_1 and c_2 denote the positive and the negative *class*.[1] Obviously, $\Pr[c_1 \mid x_i] = 1$ for all $x_i \in P$, but no information is available about $\Pr[c_1 \mid x_i]$ if $x_i \in U$. The expectation step will estimate $\Pr[c_1 \mid x_i]$ for each $x_i \in U$ (line 5), and the maximisation step will determine the conditional feature distribution $\Pr[f_j \mid c_1]$ and the prior $\Pr[c_1]$ most likely to have generated the values assigned to $\Pr[c_1 \mid x_i]$ in the expectation step (line 7).

Algorithm 1.1 *Spy-EM* Step 1: Initialisation via expectation-maximisation

Input: P and U, the sets of positive and unlabelled examples.

1: **function** INITIAL-EM(P, U)

2: $C \leftarrow \text{TrainNB}(P, U)$ ▷ Train the initial Naive Bayes classifier

3: **while** parameters of C change **do**

4: **for each** unlabelled example $x_i \in U$ **do**

5: $\boxed{\Pr[c_1 \mid x_i]} \leftarrow \text{posterior}(C, x_i)$ ▷ new probability that $x_i \in P$

6: **for each** feature f_j of x_i **do**

7: ***Recalculate*** $\Pr[f_j \mid c_1]$ and $\Pr[c_1]$ for the updated $\boxed{\Pr[c_1 \mid x_i]}$

8: **end for**

9: **end for**

10: ***Retrain*** C using new values of $\Pr[f_j \mid c_1]$ and $\Pr[c_1]$

11: **end while**

Output: C, the classifier obtained via expectation maximisation.

[1] Focus on the positive class c_1. All results for c_2 follow directly, as $\Pr[c_1 \mid x_i] = 1 - \Pr[c_2 \mid x_i]$.

Naive Bayes classifiers can be used as the basis of the *EM* algorithm, as shown in Algorithm 1.1. The initial values for the prior $\Pr[c_1]$ and the conditional probabilities $\Pr[f_j \mid c_1]$ are estimated from the training data, as explained in the Preparation.

Initially, all the unlabelled data is treated as negative, and *EM* iterates until the assignment of labels to U converges. The original paper ([11]) claims that the algorithm converges quickly. In our experiments, expectation maximisation took no more than 4-5 iterations to converge.

Initial-EM, shown in Algorithm 1.1, can be used as a stand-alone classifier. Given *easily separable* data sets, it exhibits good performance. However, the method is highly biased towards positive samples. To achieve better performance, the method should balance the influence that the positive and the (unknown) negative set have on classification: the first step of any PU algorithm is to extract a set of *reliable negatives* N from U.

The key trick employed by *Spy-EM* is to move a randomly sampled subset of P, the *spy set* SP, into the mixed set U. If we use *EM* on $P \setminus SP$ as the positive set and $U \cup SP$ as the negative set, the values of posterior probabilities assigned to *the spy elements* (SP) will be in the same range as those assigned to positives originally hidden in U. These hidden positives represent the instances we want to extract for use in the later augmentation steps.

Elements with dissimilar posteriors are *more likely* to be the *actual negatives* and should be put into N, the set of reliable negatives. The remaining set of unlabelled examples $M = U \setminus N$ will contain a higher proportion of positives than U did. Thus, *ExtractRN* (Algorithm 1.2) decomposes U into sets M and N, both of *higher purity* (the proportion of positives/negatives) than U (Figure 3.1).

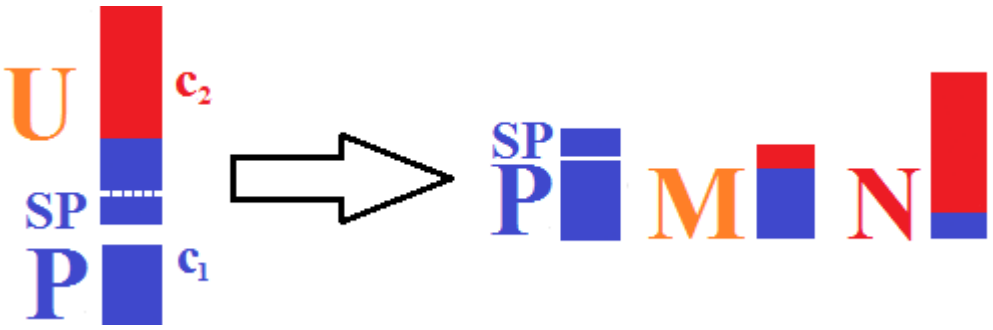


Figure 3.1: Extracting the set of *reliable negatives* N from the unlabelled set U.

Algorithm 1.2 *Spy-EM* Step 2: Extracting reliable negatives

Input: positive set P, unlabelled set U, sampling rate s.

1: **function** ExtractRN(P, U)
2: $SP \leftarrow \text{sample}(P, s)$
3: $U' \leftarrow U \cup SP$
4: $P' \leftarrow P \setminus SP$
5: $C \leftarrow$ Initial-EM(P', U') ▷ Use *EM* to build the classifier
6: **for each** $x_i \in U'$ **do**
7: $\boxed{\Pr[c_1 \mid x_i]} \leftarrow \text{posterior}(C, x_i)$ ▷ *EM* posterior probabilities
8: **end for**
9: *Choose threshold* **t** *using posteriors assigned to spy positives in EM*
10: $N \leftarrow \emptyset$, $M \leftarrow \emptyset$
11: **for each** $x_i \in U$ **do**
12: **if** $\Pr[c_1 \mid x_i] \leq t$ **then** $N \leftarrow N \cup x_i$
13: **else** $M \leftarrow M \cup x_i$
14: **end for**

Output: the set of reliable negatives N and the remaining mixed set M.

Choosing the threshold **t** (line 9) used for decomposing U is non-trivial. To extract all spy positives from U' into M, we should set **t** to $\min\{\Pr[c_1 \mid s] \mid s \in SP\}$. In a noiseless setting, this is a reasonable approach, but most real-world data sets include some outliers which cause the value of **t** to be unacceptably low. In our experiments, choosing **t** so that $l\%$ of the examples have $\Pr[c_1 \mid x] \leq \mathbf{t}$ proved to work well for any l between 5 and 20 per cent: varying l changed the number of iterations required for convergence, but had no effect on the final classifier produced.

Given the positive set P, the reliable negatives set N and the remaining mixed set M, the *EM* algorithm is employed once again to obtain the final Naive Bayes classifier. Posteriors assigned by this classifier are used to determine which of the $x \in M \cup N$ actually belong to c_1. These are the *hidden positives* that will be used to achieve data augmentation.

As shown in Algorithm 1.3, the posterior probabilities of examples in both M and N are allowed to change between subsequent *EM* iterations (whereas the posteriors of P remain fixed). This is the reason why the *noise level* l can be set to any value between 5 and 20 per cent: hidden positives wrongly placed into N

will have their posteriors corrected by the *EM* algorithm after a sufficient number of iterations.

Algorithm 1.3 *Spy-EM* Step 3: Building the final classifier

Input: positive set P, reliable negatives set N, mixed set M.

1: **function** FINAL-EM(P, N, M)

2: **for all** $x_i \in P$ **let** $\Pr[c_1 \mid x_i] \leftarrow 1$ ▷ posteriors of P **remain fixed** in *EM*
3: **for all** $x_i \in N$ **let** $\Pr[c_1 \mid x_i] \leftarrow 0$ ▷ **not fixed** in *EM*

4: $C \leftarrow$Initial-EM(P, N) ▷ Build the first classifier without considering M

5: **for each** example $x_i \in M$ **do**
6: $\Pr[c_1 \mid x_i] \leftarrow$posterior($C$, x_i) ▷ estimate posteriors for M using C
7: **end for**

8: $C_f \leftarrow$ EM($\mathbf{Pr}[c_1 \mid \mathbf{x}]$) ▷ Run *EM* with the new posteriors for M and N

Output: C_f, the final classifier for extracting *hidden positives* from $N \cup M = U$.

The final run of *EM* produces a sequence of classifiers, the last of which is *not necessarily the optimal one*. Further discussion of the stopping criteria used can be found in [11].

Most of the work in *Spy-EM* consists of training Naive Bayes classifiers required for expectation maximisation. The complexity of training a Naive Bayes classifier is $O(n')$, where n' represents the number of non-zero elements in the *feature matrix* of the training data. Assuming that the number of iterations required for *EM* to converge is bounded, as was the case in our experiments, it follows that the complexity of *Spy-EM* is at most $O(n \times f)$, n being the number of examples and f the number of features. In two of our application areas, the data sets are sparse, with each feature vector containing a limited number of non-zero attributes. Consequently, the complexity of *Spy-EM* is closer to *O(n)*.

The apparent simplicity of the pseudo-code presentation style used in this Dissertation is somewhat misleading. *Appendix B* contains representative samples of the Matlab source code written. Due to the machine learning nature of the project, no user interface was required: all code was written in Matlab and all results were obtained by running the algorithms in the Matlab environment.

The following code snippet is the part of the *Spy-EM* code that builds the final classifier (after the set of reliable negatives has been extracted):

```
% labelsEM: the list of labels assigned by the EM algorithm:
labelsEM = zeros(1, ...
    length(positiveSet)+length(negativeSet)+length(unlabeledSet));
labelsEM(positiveSet) = 1; % fix the priors of positives to 1.

if(¬isempty(negativeSet) )% if the negative set is NOT empty:
    labelsEM(negativeSet) = binarizePosteriors( posteriors( ...
        (length(positiveSet)+1) : ...
        (length(positiveSet)+length(negativeSet)) ,2) );
    disp('Negative set not empty, all is well.')
end

% the negative examples, as well as those from U, can be relabelled:
posteriorsU = PN_NBClassifier.posterior(FeatureMatrix(unlabeledSet, :));

labelsEM(unlabeledSet) = binarizePosteriors ( posteriorsU(:,2) );

oldLabels = zeros(size(labelsEM));   numIterations = 0;

% Finally, run EM: retrain the classifier using new labels, binarize ...
    the new posteriors, reset positives to 1; repeat until convergence:

while ( not(isequal(oldLabels, labelsEM)))

    oldLabels = labelsEM; numIterations = numIterations + 1;

    if(numIterations > 30) % enforce termination after many iterations
        disp('Spy-EM failed to converge!')
        break;
    end

    EM_NBClassifier = NaiveBayes.fit(FeatureMatrix, labelsEM, ...
        'Distribution', 'mn'); %multinomial distribution (bag of words).

    posteriorProbabilities = EM_NBClassifier.posterior(FeatureMatrix);

    labelsEM = binarizePosteriors(posteriorProbabilities(:,2));
    labelsEM(positiveSet) = 1;

end

finalClassPosteriors = posteriorProbabilities(:,2);
```

3.4 Rocchio-SVM Algorithm

Spy-EM was originally designed for text classification, where feature vectors consist of binary indicators stating whether certain words appear in a document or not. In this setting, the priors required for Naive Bayes classification can be estimated from the training data. By design, *Spy-EM* is unable to perform classification if feature vectors consist of continuous attributes, as is the case with the theorem prover data set. An alternative scheme applicable to such data sets can be implemented using the *Roc-SVM* algorithm.

Roc-SVM is a partially supervised learning algorithm first proposed by Li and Liu in [8]. As in *Spy-EM*, a set of *reliable negatives* N is extracted from U and used to build the final classifier. Instead of expectation-maximisation, *Roc-SVM* uses *support vector machines* to build that classifier.

To extract reliable negatives, *Roc-SVM* employs the *Rocchio classifier*. The method first computes the *centroids* (also known as the *prototype vectors*) of P and U. Then, for all the unlabelled examples $x \in U$, it computes the *cosine similarity* between their feature vectors $\vec{d_x}$ and the two centroid vectors. All instances closer (more similar) to the negative centroid become members of the set of reliable negatives N.

Algorithm 2.1 *Roc-SVM* Step 1: Rocchio classification

Input: P and U, the sets of positive and unlabelled examples.

function ROCCHIO(P, U)

$\vec{c^+} := \alpha \frac{1}{|P|} \sum_{\vec{d} \in P} \frac{\vec{d}}{||\vec{d}||} - \beta \frac{1}{|U|} \sum_{\vec{d} \in U} \frac{\vec{d}}{||\vec{d}||}$

$\vec{c^-} := \alpha \frac{1}{|U|} \sum_{\vec{d} \in U} \frac{\vec{d}}{||\vec{d}||} - \beta \frac{1}{|P|} \sum_{\vec{d} \in P} \frac{\vec{d}}{||\vec{d}||}$

$N \leftarrow \emptyset$

for the feature vector $\vec{d_x}$ of every unlabelled example $x \in U$ **do**

  if sim($\vec{c^+}$, $\vec{d_x}$) $\leq$ sim($\vec{c^+}$, $\vec{d_x}$) **then** ▷ sim($\vec{c}$, $\vec{d}$) $= \frac{\vec{c} \cdot \vec{d}}{||\vec{c}|| ||\vec{d}||}$

    $N \leftarrow N \cup \{x\}$

end for

Output: the set of reliable negatives N.

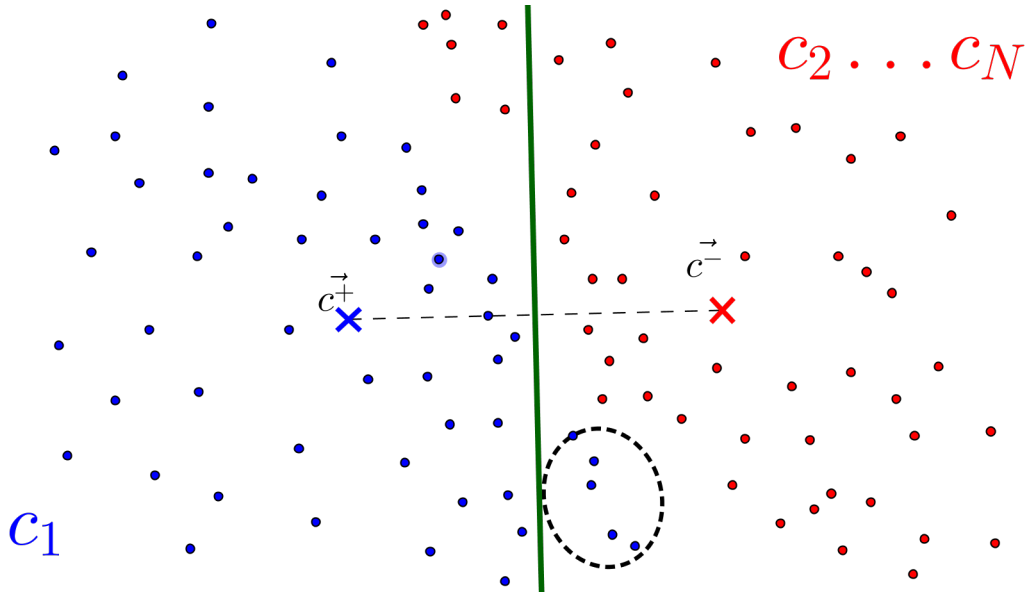


Figure 3.2: Rocchio classifiers are not able to remove all positives from N [8].

Rocchio classification could be used as a stand-alone method for extracting N. This method performs classification using a separating hyperplane equidistant from the two centroids. If the actual decision boundary is not represented by *that specific hyperplane*, Rocchio misclassifies many of the examples.

Lower purity of the reliable negatives set does not degrade *Spy-EM* performance because additional iterations of *EM* relabel the positives present in N. *Roc-SVM* builds the final classifier using support vector machines, which can be much more susceptible to noise than *EM* is. Hence, purging as many positives from N is of paramount importance for subsequent *SVM* performance.

P contains examples from the positive class c_1. Conversely, U may include negative examples of many different classes: $c_2 \ldots c_N$. Thus, the centroid vectors used in Rocchio are not representative of the negative classes; they tend to pull some of the hidden positives into N (Figure 3.2).

K-means clustering can be used to separate the negative set obtained using Algorithm 2.1 into K clusters, each (hopefully) representing *distinct* negative classes. Rocchio classification is then re-employed to identify instances closer to at least one of the new negative centroids than to the positive set P. These examples become the final negative set N. This method, shown in Algorithm 2.2, outperforms the stand-alone Rocchio classifier [8].

Algorithm 2.2 *Roc-SVM* Step 2: K-means clustering for negatives extraction
Input: P and U, the sets of positive and unlabelled examples.

function ROCCHIO WITH K-MEANS(P, U)
  $RN \leftarrow$ Rocchio(P, U)
  Randomly choose k initial cluster centres from RN ▷ K = 10 was used in the experiments
  Perform *K-means clustering* to separate RN into $N_1 \dots N_k$
  for $i = 1$ to k **do**
    $$\vec{p_i} := \alpha \frac{1}{|P|} \sum_{\vec{d} \in P} \frac{\vec{d}}{||\vec{d}||} - \beta \frac{1}{|N_i|} \sum_{\vec{d} \in N_i} \frac{\vec{d}}{||\vec{d}||}$$
    $$\vec{n_i} := \alpha \frac{1}{|N_i|} \sum_{\vec{d} \in N_i} \frac{\vec{d}}{||\vec{d}||} - \beta \frac{1}{|P|} \sum_{\vec{d} \in P} \frac{\vec{d}}{||\vec{d}||}$$
  end for

  $N \leftarrow \emptyset$
  for the feature vector $\vec{d_x}$ of every example $x \in RN$ **do**
    if $\exists \vec{n_i} \forall \vec{p_j} [\text{sim}(\vec{d_x}, \vec{n_i}) \geq \text{sim}(\vec{d_x}, \vec{p_j})]$ **then**
    $N \leftarrow N \cup \{x\}$ ▷ x is closer to *some* negative than to *all* positives
  end for

Output: the set of reliable negatives N.

K-means Rocchio extracts a negative set of high purity by enforcing *very rigid criteria* for including examples into N. This strictness can cause the resulting negative set to become too small to warrant a good classifier [8]. If so, a method similar to expectation-maximisation is employed to *iteratively grow* N:

The first SVM is trained using only P and N and used to extract $W_1 \subseteq M$, the set of examples in M likely to be negative. The next SVM is trained using P and the *new negative set* $N \cup W_1$. This step is repeated until the (final) SVM built can extract no new negatives from the remaining mixed set M_i. The final mixed set $M_i \subseteq M$ represents the set of hidden positives identified by *Roc-SVM*.

This iterative method has its drawbacks. As SVMs can be very sensitive to noise, an inclusion of a small number of positives from M into N in any iteration can cause an avalanche effect: the algorithm may pull many more similar positives into the negative set by the time it terminates. Consequently, the performance of the classifiers produced may fall into a downward spiral.

We can determine whether this method went awry by observing how the final SVM performs on the positive set. If it classifies more than 5% of examples in P as negatives, that means that too many positives were pulled into the negative set. If so, we revert to the first SVM, trained using only P and N (lines 12-14).

Algorithm 2.3 *Roc-SVM* Step 3: Building the final classifier

Input: positive set P, mixed set M, reliable negatives set N.

1: **function** ROC-SVM(P, M, N)

2: $i := 1$, $N_i \leftarrow N$, $M_i \leftarrow M$

3: $C_i \leftarrow$ TrainSVM(P, N_i) ▷ use only P and N to build the first SVM

4: $W_i \leftarrow \{x \in M_i \mid \text{SVMClassify}(C_i, x) = -1\}$ ▷ *negatives* C_1 identifies in M

5: **while** $W_i \neq \emptyset$ **do** ▷ until no more negatives can be extracted from M

6: $N_{i+1} \leftarrow N_i \cup W_i$ ▷ move new negatives to the negative set

7: $M_{i+1} \leftarrow M_i \setminus W_i$ ▷ ...and remove them from the mixed set

8: $i \leftarrow i + 1$

9: $C_i \leftarrow$ TrainSVM(P, N_i) ▷ build the next SVM using the expanded N

10: $W_i \leftarrow \{x \in M_i \mid \text{SVMClassify}(C_i, x) = -1\}$ ▷ generate $W_i \subseteq M_i$ using C_i

11: **end while**

12: $p \leftarrow \frac{|\{x \in P \mid \texttt{SVMClassify}(C_i, x) = 1\}|}{|P|}$ ▷ Final SVM's *recall* on set P

13: **if** $p \geq 0.95$ $C_{\texttt{f}} := C_i$ ▷ if recall is high, use the final SVM

14: **else** $C_{\texttt{f}} := C_1$ ▷ otherwise, revert to the first SVM

Output: the final classifier $C_{\texttt{f}}$.

Roc-SVM can perform entity set expansion in data sets with real-valued attributes and tends to outperform *Spy-EM* on binary data sets [8]. These improvements come at the expense of efficiency. *Roc-SVM* trains a sequence of SVMs in order to build the final classifier. SVM training has a quadratic and a cubic component. Estimating the exact complexity is hard, but we can expect training times of the order of (at least) $O(n^2)$, n being the number of training examples. Assuming a *constant* number of iterations in Algorithm 2.3, *Roc-SVM's* best-case complexity is $O(n^2)$, still vastly inferior to *Spy-EM's* $O(n)$.

Roc-SVM was not only the most computationally demanding algorithm considered, but also the one most difficult to implement. Issues related to the termination of SVM training, the handling of dangerously small clusters and the uncertainty in choosing the optimal value for the SVM soft margin parameter marked the implementation of this algorithm. The following code snippet illustrates some of the subtleties I had to deal with to make this scheme work. *Appendix B* contains the full *Roc-SVM* implementation.

```
% ...after Rocchio identifies the first reliable negatives set and ...
    after k-means clustering has been applied to partition it:

for i = 1:numClusters
   [positiveCentroids(i, :), negativeCentroids(i, :)] = ...
      calculateCentroids(FeatureMatrix, PositiveSet, ...
      NegativeCluster{i}, alpha, beta);
end

% Compute the cosine similarities between clusters and entries:
for i = 1:numClusters

    % Calculate the dot product of the i-th centroid with all entries
    similarityMatrixPositive(i, :) = (sum(FeatureMatrix .* ...
       repmat(positiveCentroids(i,:), size(FeatureMatrix, 1), 1), 2));
    similarityMatrixNegative(i, :) = (sum(FeatureMatrix .* ...
       repmat(negativeCentroids(i,:) , size(FeatureMatrix, 1), 1), 2));

    % Divide the dot product by the norms of the respective examples:
    similarityMatrixPositive(i, :) = ( similarityMatrixPositive(i, :) ...
       ./  featureNorms ) / norm(positiveCentroids(i,:);
    similarityMatrixNegative(i, :) = ( similarityMatrixNegative(i, :) ...
       ./ featureNorms ) / norm(negativeCentroids(i,:);
end

% Extract the final set of negatives using these similarities:
maxSimilarityPositive(1:numEntries) = max( ...
    similarityMatrixPositive(:, 1:numEntries));
maxSimilarityNegative(1:numEntries) = max( ...
    similarityMatrixNegative(:, 1:numEntries));

% Examples with a positive value of this difference belong to the RN set:
similarityDifference = maxSimilarityNegative - maxSimilarityPositive;
similarityDifference(PositiveSet) = -1; %... and positives do not!

reliableNegatives = sort(LabelToArray(similarityDifference));
```

3.5 Bayesian Sets

Bayesian sets [6] are an information retrieval inspired algorithm for entity set expansion. Unlike *Spy-EM* and *Roc-SVM*, which build classifiers to decide membership of the positive set P, Bayesian sets *rank* all examples of the data set D by their *likelihood* of belonging to P. Formally, given the *query set* $Q \subset D$, the goal of the algorithm is to order all items in D by their *similarity* with elements of Q, so that the elements of Q hidden in D feature highly in the ranking produced.

Bayesian Sets take a probabilistic model approach, assuming all examples are independently and identically distributed from some statistical model $P(\mathbf{x} \mid \theta)$, where θ represents the model parameters. In this setting, examples belonging to the same concept class (in this case Q) must have been generated using the same set of parameters θ_Q.

The marginal probability $P(\mathbf{x} \mid Q, \theta_Q)$ can be used to estimate $\Pr[\mathbf{x} \in Q]$. θ_Q is unknown, so we average across the possible values of θ according to $P(\theta)$, the *prior* density on θ, to obtain $P(\mathbf{x} \mid Q)$. To account for the biasing effect of longer examples (longer documents, proteins, etc.) being less frequent and thus having a lower marginal probability, *normalise* the scoring metric to obtain:

$$\texttt{score}(\mathbf{x}) = \frac{P(\mathbf{x} \mid Q)}{P(\mathbf{x})} = \frac{P(\mathbf{x}, Q)}{P(\mathbf{x})P(Q)} \approx \frac{Pr[\mathbf{x} \in Q]}{\Pr[\mathbf{x} \notin Q]}. \tag{3.1}$$

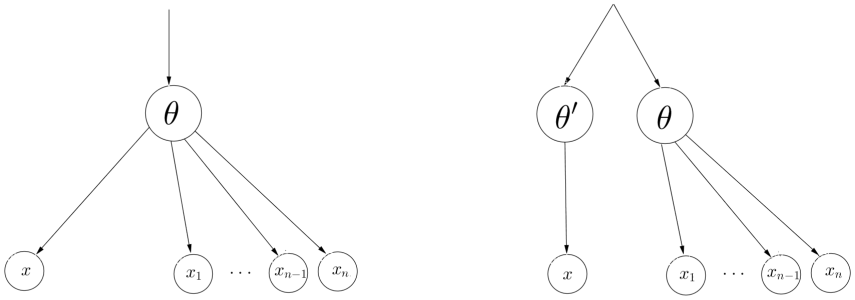
Using basic rules of probability, $P(\mathbf{x})$ and $P(\mathbf{x} \mid Q)$ can be expressed as:

$$P(\mathbf{x}) = \int P(\mathbf{x} \mid \theta) P(\theta) d\theta, \tag{3.2}$$

$$P(Q) = \int \prod_{\mathbf{x_i} \in Q} P(\mathbf{x_i} \mid \theta) P(\theta) d\theta,$$

$$P(\theta \mid Q) = \frac{P(Q \mid \theta) P(\theta)}{P(Q)},$$

$$P(\mathbf{x} \mid Q) = \int P(\mathbf{x} \mid \theta) P(\theta \mid Q) d\theta. \tag{3.3}$$



$P(x, Q)$ $P(x)P(Q)$

Figure 3.3: the `score` function compares the probabilities of the data being generated by the two graphical models given above [6].

The scoring metric reflects the ratio between $\Pr[\mathbf{x} \in Q]$ and $\Pr[\mathbf{x} \notin Q]$ (equations 3.2-3.3, Figure 3.3). As such, this score can not be normalised to obtain $\Pr[\mathbf{x} \in Q]$; this would require integration over all θ, which would be intractable.

Nevertheless, `score` provides an adequate measure of likelihood, since $\forall \mathbf{x_1}, \mathbf{x_2} \in D$. $\Pr[\mathbf{x_1} \in Q] \leq Pr[\mathbf{x_2} \in Q] \Rightarrow \texttt{score}(\mathbf{x_1}) \leq \texttt{score}(\mathbf{x_2})$. Therefore, it can be used to impose the desired ordering between the elements of D. Once the scoring function has been implemented, the Bayesian Sets algorithm is trivial:

Algorithm 3.1 *Bayesian Sets algorithm*

Input: set of items D, query set $Q = \{\mathbf{x_i}\} \subset D$.

1: **function** BAYESIAN SETS(Q, D)

2: **for each** $\mathbf{x} \in D$ **do**

3: ***compute*** $\texttt{score}(\mathbf{x}) = \dfrac{P(\mathbf{x} \mid Q)}{P(\mathbf{x})}$

4: **end for**

Output: the list of all elements in D, sorted by decreasing scores.

Intractability and difficulties in choosing the probability models and priors are recurring themes in most approaches based on Bayesian inference, including this one. If we limit our discussion to data sets featuring binary attributes, that is, if for all $x_i \in D$ it holds that $x_{i,1} \ldots x_{i,n} \in \{0, 1\}$, we can choose the probability models and the priors in a way that will make the integrals in equations 3.2 and 3.3 analytical.

Model the marginal probability $P(\mathbf{x} \mid \theta)$ using the *Bernoulli distribution*:

$$P(\mathbf{x} \mid \theta) = \prod_{j=1}^{n} \theta_j^{x_{i,j}} (1 - \theta_j)^{1 - x_{i,j}} \tag{3.4}$$

The distribution $P(\theta)$ is said to be a *conjugate prior* for the likelihood $P(\mathbf{x} \mid \theta)$ if the posterior probability $P(\theta \mid \mathbf{x})$ belongs to the *same family* of probability distributions as the prior. The use of conjugate priors can vastly simplify the task of inference. To this end, model the prior $P(\theta)$ using the conjugate prior of the Bernoulli distribution, given by the *Beta distribution*:

$$P(\theta \mid \alpha, \beta) = \prod_{j=1}^{n} \frac{\Gamma(\alpha_j + \beta_j)}{\Gamma(\alpha_j)\Gamma(\beta_j)} \theta_j^{\alpha_j - 1} (1 - \theta_j)^{\beta_j - 1}, \tag{3.5}$$

where α and β represent the *hyperparameters* of this distribution. In the experiments, these were set to $\alpha = c \cdot \mathbf{m}$, $\beta = c \cdot (1 - \mathbf{m})$, where $c = 2$ represents the *Dirichlet coefficient* and $\mathbf{m}$ represents the *mean vector* of all $\mathbf{x} \in D$.

From these assumptions, it follows that $P(\theta \mid \mathbf{x})$ is a Beta distribution as well. Its hyperparameters, α' and β', can be expressed as closed-form expressions of α, β and $\{x_{i,j}\}$. The integrals in equations 3.2 and 3.3 are now analytical, making *exact inference* possible. As shown in [6], the use of Bernoulli and Beta distributions for $P(\mathbf{x} \mid \theta)$ and $P(\theta)$ reduces the computation of the list of *log scores* $s_1 \ldots s_N$ to a single (sparse) matrix multiplication:

$$\mathbf{s} = c + \mathbf{X}\mathbf{q}, \qquad s_i = \log(\texttt{score}(\mathbf{x_i})), \tag{3.6}$$

where $\mathbf{X}$ is the N by n *feature matrix* of D, and c and $\mathbf{q}$ are auxiliary values computed in linear time using the hyperparameters and the feature matrix $\mathbf{X}$:

$$c = \sum_{i=1}^{n} \log(\frac{\alpha_i + \beta_i}{\alpha_i + \beta_i + N} \frac{\beta'_i}{\beta_i}), \qquad q_i = \log(\frac{\alpha'_i \beta_i}{\alpha_i \beta'_i}). \tag{3.7}$$

For sparse feature matrices $\mathbf{X}$, the multiplication in equation 3.6 can be implemented *very efficiently*, lowering the computational complexity of the algorithm to the order of *O(N)*, where N is the number of examples in D. Even though Bayesian Sets have the same asymptotic complexity as *Spy-EM*, there is practically *no overhead* associated with their execution, making them (by far) the fastest entity expansion algorithm implemented in this project.

My Matlab implementation of Bayesian Sets manages to abstract away from the complexity of the underlying mathematics. Unlike the previous two algorithms, understanding the mathematical assumptions behind Bayesian Sets constituted the bulk of the effort in their implementation. After the `score` function calculation is well understood, the Matlab code can be made very elegant:

```
function scoreVector = BayesianSet(FeatureMatrix, entitySetIndices, ...
    concentrationParameter)

% FeatureMatrix : binary sparse matrix representing feature vectors.
% entitySetIndices : indices of the set to be expanded.
% concentrationParameter: Dirichlet concentration parameter.
% scoreVector: the row vector of log likelihoods for all examples.

if nargin < 3
    concentrationParameter = 2;
end

clusterSize = length(entitySetIndices);
numEntries = size(FeatureMatrix,2);

% clusterFeatures: the total number of non-zero features in the data set.
clusterFeatures = sum(FeatureMatrix(:,entitySetIndices),2);

% meanVector: the mean vector of features accross all examples (the ...
    probability that the given feature appears in an entry).
meanVector = full(sum(FeatureMatrix,2)) / numEntries;

% alpha,beta : Beta hyperparameters of the prior distribution (column ...
    vectors which have values that represent how often specific ...
    features appear in the examples).

alpha = concentrationParameter * meanVector;
beta  = concentrationParameter * (1 - meanVector);

q = log(1 + clusterFeatures ./ alpha) - log(1 + (clusterSize - ...
    clusterFeatures) ./ beta);

c = sum(  log( (alpha + beta) ./ (alpha + beta + clusterSize) )  +  ...
    log( (beta + clusterSize - clusterFeatures) ./ beta )  );

% the computation of the final score vector reduces to a single ...
    multiplication of a vector with a sparse matrix:

scoreVector = c + q' * FeatureMatrix;
```

The following table provides the top twenty results I obtained by applying Bayesian Sets to the MovieLens data set (1643 films described by binary attributes). The query set $Q = \{172, 181, 210\}$ consists of two Star Wars films and an Indiana Jones film (all directed by George Lucas). Bayesian Sets return a range of sci-fi films akin to Star Wars, but the Indiana Jones film is not even in the top twenty results, although its influence (presumably) manifests itself through the presence of films such as the African Queen and Operation Dumbo Drop, which have little to do with the two Star Wars films present in the query.

Film ID	Score	Film Name
181	**9.58508**	**Return of the Jedi (1983)**
50	9.58508	Star Wars (1977)
172	**9.34084**	**Empire Strikes Back, The (1980)**
271	7.99097	Starship Troopers (1997)
498	7.14069	*African Queen, The (1951)*
897	5.20461	Time Tracers (1995)
450	5.20461	Star Trek V: The Final Frontier (1989)
449	5.20461	Star Trek: The Motion Picture (1979)
380	5.20461	Star Trek: Generations (1994)
373	5.20461	Judge Dredd (1995)
230	5.20461	Star Trek IV: The Voyage Home (1986)
229	5.20461	Star Trek III: The Search for Spock (1984)
228	5.20461	Star Trek: The Wrath of Khan (1982)
227	5.20461	Star Trek VI: The Undiscovered Country (1991)
222	5.20461	Star Trek: First Contact (1996)
82	5.20461	Jurassic Park (1993)
62	5.20461	Stargate (1994)
121	5.01092	Independence Day (ID4)
110	4.40121	*Operation Dumbo Drop (1995)*
1239	4.35433	Cutthroat Island (1995)

Users searching for films akin to Star Wars will be satisfied with these results. There exists *no correct answer* to a query: different users might prefer different rankings, due to the potential semantic ambiguity of the query sets or the specific user's perception of similarity. Therefore, measuring the quality of the ranking produced is a highly non-trivial task, as discussed in subsequent sections.

3.5.1 Iterative Bayesian Sets

As presented in the Evaluation, data augmentation via Bayesian Sets proved to be inferior to *Spy-EM* and *Roc-SVM* in one of the application areas. In accordance with the evolutionary prototyping paradigm, I formulated and implemented a final extension to Bayesian Sets during the last stages of the project.

Blind (pseudo) relevance feedback is a technique for achieving automatic query expansion in information retrieval systems. As used in this problem, the technique applies Bayesian Sets to obtain the ranking of D for the given query set. Then, the top K results in the ranking are added to the query set Q. This process is repeated until a new iteration identifies no new elements to be added to the query set. In the experiments, $K = 30$ and $K \in (0.05N, 0.2N)$ were attempted.

Algorithm 3.2 *Iterative Bayesian Sets*

Input: set of items D, query set $Q = \{\mathbf{x_i}\} \subset D$.

1: **function** ITERATIVE BAYESIAN SETS(Q, D)

2: $i \leftarrow 1,\ Q^0 \leftarrow \emptyset,\ Q^1 \leftarrow Q$ ▷ auxiliary variables, the initial query

3: **while** $Q^i \neq Q^{i-1}$ **do** ▷ until the query converges

4: $R_i[1 \dots N] \leftarrow$ Bayesian Sets(Q^i, D) ▷ ranking of D using the i-th query

5: $Q^{i+1} \leftarrow Q^i \cup \{R[1], R[2] \dots R[K]\}$ ▷ add the top K elements to the query

6: $i \leftarrow i + 1$

7: **end while**

Output: $R_f[1 \dots N]$, the ranking of D produced using the final query.

In some application areas, as shown in the Evaluation, pseudo relevance feedback managed to improve data augmentation achieved using Bayesian Sets by a substantial margin.

3.6 Data Augmentation

Support vector machines were used as the *benchmark classifier* for all three tasks. The SVM is first trained using T_r, the original training data. Then, the algorithm creates the augmented training set T_r^+ by labelling examples from U, the unlabelled set. The new data set is then used to train another SVM. The success of data augmentation is reflected through the *difference in the classification error* between these two classifiers.

As discussed in the Preparation, SVMs support two-group classification of examples with both discrete and continuous features. As such, they can be applied to the first two tasks directly. Task III boasts five different classes, but SVMs can be used as building blocks of a multi-class classification scheme as well.

The operation of the data augmentation scheme is the same across all three tasks (Algorithm 4.1, Figure 3.4). For each of the K classes in the training set, the algorithm identifies which examples in U are likely to belong to that class.

Algorithm 4.1 *Data augmentation*

Input: Training set $T_r = T_{r_1} \cup \ldots \cup T_{r_K}$, unlabelled set U.

1: **function** AUGMENTDATA($T_{r_1} \ldots T_{r_K}, U$)

2: **for each** class $i \in (1, K)$ **do** ▷ K = 2 for Tasks I, II; K = 5 for Task III
3: $T_{r_i}^+ \leftarrow$ Entity Set Expansion (T_{r_i}, U) ▷ T_{r_i} is used as the positive set
4: **end for**

5: $S \leftarrow \emptyset$
6: **for each** class $i \in (1, K)$ **do** ▷ find all duplicates
7: **for all** $j \in [1, K]$ **do**
8: **if** $i \neq j$ **then** $S \leftarrow S \cup \{T_{r_i} \cap T_{r_j}\}$
9: **end for**
10: $T_{r_i}^+ \leftarrow T_{r_i}^+ \setminus S$ ▷ remove duplicates
11: **end for**

Output: $T_{r_1}^+ \ldots T_{r_K}^+$, the augmented training sets for each class.

The *hidden members* of each of the classes are identified using the three entity set expansion algorithms implemented. The set of training examples belonging to the ith class T_{r_i} acts as the positive set to be expanded using the unlabelled

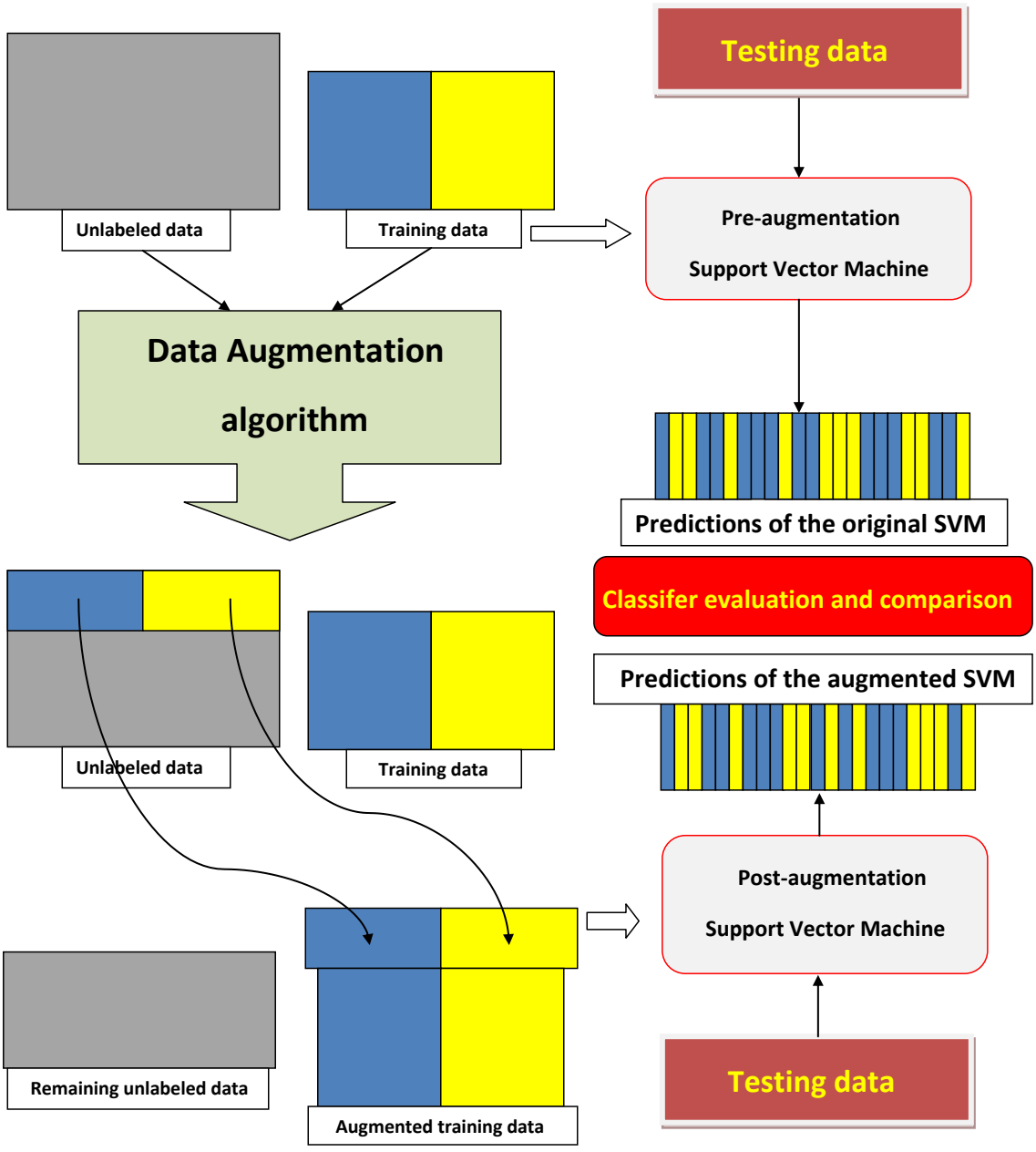


Figure 3.4: Flow diagram of the data augmentation process.

set U. The newly identified members of the ith class are then added to T_{r_i} in order to obtain the augmented set $T_{r_i}^+$.

The success of data augmentation depends on achieving a good trade-off between creating a (substantially) larger training data set and introducing too much noise (wrongly labelled examples) into the training set. In our experiments, erring on the conservative side (adding fewer elements in order to achieve higher purity of the augmented data set) proved to be the superior approach.

PU learning makes no use of the negative classes. Once the augmented data set T_r^+ has been created, I tried to extract some marginal utility from the fact that instances of all classes are available: to maximise the purity of the augmented set, all examples identified as members of multiple classes are removed from T_r^+ (lines 5-11). As shown in the Evaluation, removing these *unreliable examples* from the augmented data set immensely improved the subsequent classification error.

Of the three methods used to achieve entity set expansion (line 3), there is not much to say about the use of *Spy-EM* and *Roc-SVM* algorithms: they produce binary classifiers that decide membership of each of the K classes present in the training set T_r. These classifiers identify a subset of U to be used for augmenting each of the classes in the training set.

Unlike *Spy-EM* and *Roc-SVM*, Bayesian Sets do not provide binary decisions about class membership. Instead, for each of the K classes, they provide a ranking of all elements in U by their likelihood of belonging to that class. As these likelihoods do not represent the actual probabilities, they can not be used to determine a clear cut-off criterion d_i for the number of elements from the ith ranking to be included into $T_{r_i}^+$. Making an informed decision proved to be very difficult. Two different methods for setting the value of d_i were attempted:

1. If the class distributions in T_r and U are the same, the number of elements of each class in U can be reliably estimated from the class distribution observed in T_r.

2. Set each d_i to the number of elements of that class included into the augmented data set $T_{r_i}^+$ by *Spy-EM*. As shown in the Evaluation, this ad-hoc approach was useful for comparing Bayesian Sets to *Spy-EM*.

Different variations of this algorithm were implemented for applying each augmentation scheme to each of the data sets. The following code snippet shows how these methods are used to obtain statistically significant comparisons between the three schemes when used to augment the data sets of Tasks I and II:

```
function [ResultsBayes, ResultsSEM, ResultsRocSVM, ...
    entitySetExpansionQuality] = assessDataAugmentation(TrainFM, ...
    TrainLabels, TestFM, TestLabels, kkt_threshold, max_iter, ...
    dist_bias, bidirection_pull, useBayes, useSEM, useRocSVM)

% dist_bias: parameter used to change the positives to negatives ...
    ratio in the reduced training set constructed.
% For example, dist_bias = 2 means that P:N ratio of ReduxTrainFM ...
    constructed is twice that of TrainFM. 0.5 means the opposite.

ResultsSVM = zeros(13,7); % 1-3 reduced sets, 4 original, 5-7 first ...
    redux with 3 different augmentations, 8-10 second, 11-13 third.

disp('starting algorithm')

SVMCLassifer = svmtrain(full(TrainFM), TrainLabels ...
    ,'kktviolationlevel', kkt_threshold, 'options', ...
    statset('MaxIter', max_iter, 'Display', 'final' ));

Results = svmclassify(SVMCLassifer, full(TestFM));
ResultsSVM(4,:) = quality(Results, TestLabels);

% Now, we need to create redux sets with their labels and the ...
    remaining unlabeled set: its feature matrix and its (true) labels ...
    required to assess the quality of entity set expansion.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sample = 0.1; % first redux

[reduxFM, reduxLabels, reduxAugLabels, reduxAugLabelsNegative] = ...
    createRedux(TrainFM, TrainLabels, sample, dist_bias);

[newPosBayes, newPosSEM, newPosRocSVM] = augmentData(TrainFM, ...
    reduxAugLabels, LabelToArray(reduxAugLabelsNegative), useBayes, ...
    useSEM, useRocSVM, kkt_threshold, max_iter);

% Augment both classes; remove examples belonging to both augmented sets:
if(bidirection_pull==1)

    [newNegBayes, newNegSEM, newNegRocSVM] = augmentData(TrainFM, ...
        reduxAugLabelsNegative, 1:sum(reduxAugLabels), useBayes, ...
        useSEM, useRocSVM, kkt_threshold, max_iter);

    interBayes = intersect(newPosBayes, newNegBayes);
    interSem = intersect(newPosSEM, newNegSEM);
    interRocSVM = intersect(newPosRocSVM, newNegRocSVM);
```

```
32      newPosBayes = setdiff(newPosBayes, interBayes);
33      newNegBayes = setdiff(newNegBayes, interBayes);
34      newPosSEM = setdiff(newPosSEM, interSem);
35      newNegSEM = setdiff(newNegSEM, interSem);
36      newPosRocSVM = setdiff(newPosRocSVM, interRocSVM);
37      newNegRocSVM = setdiff(newNegRocSVM, interRocSVM);
38
39  end
40
41  % Train the (baseline) classifier using the reduced training set:
42  SVMClassifierRedux10 = svmtrain(full(reduxFM), ...
        reduxLabels,'kktviolationlevel', kkt_threshold, 'options', ...
        statset('MaxIter', max_iter, 'Display', 'off' ));
43
44  Results = svmclassify(SVMClassifierRedux10, TestFM);
45
46  ResultsSVM(1,:) = quality(Results, TestLabels);
47
48  clear SVMClassifierRedux10;
49
50  % Then, train a SVM using each of the new augmented data sets.
51  % ......
52  % ......
```

The use of these methods was constantly plagued by the lack of RAM memory. As explained in the Evaluation, the same samples of T_r had to be used to ensure that the comparisons between the three methods had statistical significance. Thus, all three variations of T_r^+ had to be present in memory simultaneously.

Different optimisations were implemented to relieve the memory requirements. Matlab's 'single' and 'sparse' data types were used extensively and constant manual garbage collection was enforced to pre-empt the use of virtual memory by Matlab. As a result, most schemes' execution and evaluation completed relatively quickly and gracefully.

The only exception was the *Roc-SVM* scheme: it could not handle the biomedical data set using the 6GB of RAM on my laptop. By my estimates, using an extra 4GB would have sufficed, but my laptop can support no more than 8GB. The use of substantial amounts of virtual memory made the execution *very slow*: *Roc-SVM* augmentation took 34 hours to terminate. In stark contrast, Bayesian Sets executed in a matter of minutes and the *Spy-EM* scheme took less than an hour.

The execution and evaluation of these methods was performed in the Matlab environment. Additional code samples can be found in *Appendix B*.

3.6.1 Multi-class Classification

In Task III, each example $\mathbf{x}$ is represented by a set of (real-valued) properties of a first order logic statement and belongs to one of the K (five) classes. To achieve data augmentation, we first need to construct f, the classifier deciding which heuristic should be used to attempt to prove an unseen statement $\mathbf{x}'$. Having built that function, *Roc-SVM* can be used for attempting data augmentation, that is, reducing the classification error of f.

Two different methods that use SVMs to create multi-class classification schemes have been implemented: *one-vs-all* and *all-vs-all*.

In *one-vs-all classification*, K different classifiers (SVMs) are built. Given a new example $\mathbf{x}$, $f_i(\mathbf{x})$ decides membership of class i (whether $\mathbf{x}$ belongs to class i or not). f_i is trained using T_{r_i} (training examples of class i) as the positive set and $T_r \setminus T_{r_i}$ (the remaining examples) as the negative set. Classification is performed using:

$$f(\mathbf{x}) = \underset{i \in 1 \ldots K}{\operatorname{argmax}} f_i(\mathbf{x}). \tag{3.8}$$

All-vs-all classification schemes build $\frac{K(K-1)}{2}$ different classifiers to decide preference between all pairs of classes. Let $f_{i,j}$ denote the SVM used to determine preference between classes i and j, trained using T_{r_i} as the positive set and T_{r_j} as the negative set. $f_{i,j}(\mathbf{x}) = 1$ if $\mathbf{x}$ is more likely to belong to class i; otherwise, $f_{i,j}(\mathbf{x}) = -1$. As the final class of an example $\mathbf{x}$, choose:

$$f(\mathbf{x}) = \underset{i}{\operatorname{argmax}} \left(\sum_{j \neq i} f_{i,j}(\mathbf{x}) \right). \tag{3.9}$$

In the experiments, all-vs-all classification managed to outperform the baseline algorithm (always choosing the most frequent class from T_r), whereas one-vs-all did not. As a result, the all-vs-all classification scheme based on support vector machines was used as *the benchmark classifier* for Task III. Samples of the code implementing the multi-class augmentation scheme can be found in *Appendix B*.

Chapter 4

Evaluation

This chapter evaluates the performance of the PU algorithms and the data augmentation schemes implemented. The system designed to provide statistically significant insights into the extent to which data augmentation improves classification is presented and used to analyse the three application areas suggested.

4.1 Overview

Testing machine learning algorithms is an inherently hard task [1]. In this chapter, I will present a detailed analysis of the performance measures for the schemes implemented. Functional correctness of these algorithms is hard to establish: to this end, the first part of this chapter is devoted to the results obtained by applying our PU algorithms to the task of entity set expansion. These results are in line with those obtained in [8, 10, 11].

The second part of the chapter analyses the extent to which data augmentation can reduce the classification error. The experiments were designed to facilitate direct statistically significant comparisons between the data augmentation schemes. The task of applying each of these schemes to the three data sets was considered.

As far as I am aware, this is the first time that these algorithms have been used to attempt data augmentation. The results obtained clearly show that this endeavour has been a major success.

4.1.1 Performance Measures

In order to reason about the quality of the classifiers produced, both for entity set expansion and in Tasks I - III (the application areas), five standard measures were employed. The classifier produced is used to classify all examples in the test set: let TP denote the number of true positives (positives examples classified as such), TN the number of true negatives, FP of false positives (negatives classified as positives) and FN of false negatives. Then, define:

1. *Accuracy*: the proportion of examples (both positives and negatives) classified correctly:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} . \tag{4.1}$$

2. *Classification error*: the proportion of examples classified incorrectly:

$$ERR = \frac{FP + FN}{TP + TN + FP + FN} = 1 - ACC . \tag{4.2}$$

3. *Precision* (also known as positive predictive accuracy): the proportion of positives among all elements classified as positive:

$$PR = \frac{TP}{TP + FP} . \tag{4.3}$$

4. *Recall* (also known as sensitivity): the proportion of positive examples correctly identified as such:

$$RC = \frac{TP}{TP + FN} . \tag{4.4}$$

5. *F-score*: harmonic mean of precision and recall. This metric unifies precision and recall, penalising low values of either of them:

$$F = \frac{2\ PR\ RC}{PR + RC} = \frac{2\ TP}{2\ TP + FP + FN} . \tag{4.5}$$

All of these measures range between 0 and 1, with higher values representing the better ones. *Precision* and *F-Score* are the principal measures we will use to compare the quality of the classifiers produced.

Table 4.1: Entity Set Expansion in 20Newsgroups data set

Positives in U	**Bayesian Sets**			**Spy-EM**		
	Precision	Recall	F-Score	Precision	Recall	F-Score
90%	0.759	0.846	0.800	0.778	0.866	0.820
67%	0.678	0.893	0.771	0.676	0.891	0.769
20%	0.326	0.946	0.485	0.325	0.942	0.483

4.2 Entity Set Expansion

The differences in entity set expansion quality between *Spy-EM* and *Roc-SVM* are relatively minor [8]. Therefore, consideration of *Roc-SVM* will be delayed until the next section. The performance of *Spy-EM* and Bayesian Sets will be compared in three different data sets:

1. *20Newsgroups* text categorisation collection. This data set was used for testing and verifying that the implementation of *Spy-EM* produced results comparable to those presented in [11].

2. *Reuters21578* text categorisation collection, used in Task I.

3. *KDD Cup 2001*, Task 1 (Binding to Thrombin) data set, used in Task II.

As explained in the Implementation, finding a cut-off criterion for Bayesian Sets is non-trivial. In this section, the second approach is taken: the element count set by *Spy-EM* is reused. Even though this boundary does not represent the ideal cut-off for Bayesian Sets, it provides a good setting for comparing the purity of the expanded entity sets produced by these two algorithms.

Precision and recall are measures best suited to assess the quality of entity set expansion; accuracy and classification error will be used to assess the subsequent classifiers produced. All three data sets were attempted with several different proportions of positives left in the unlabelled set U, as shown in the tables. The fewer positives are left in U, the worse the subsequent performance is, as shown by the results obtained.

The results of 20Newsgroups and Reuters21578 experiments reveal stark similarities between the performance of these algorithms. With fewer positives left in U and more of them in P, the performance of the two algorithms converges, showing that Bayesian Sets can match and even improve on *Spy-EM's* perfor-

Table 4.2: Entity Set Expansion in Reuters21578 data set

Positives in U	**Bayesian Sets**			**Spy-EM**		
	Precision	Recall	F-Score	Precision	Recall	F-Score
90%	0.664	1.000	0.798	0.663	0.999	0.797
67%	0.587	1.000	0.739	0.586	0.998	0.738
20%	0.282	1.000	0.440	0.282	1.000	0.440

mance. Given a cut-off criterion specifically tailored for Bayesian Sets, it is not unreasonable to believe that they would be capable of outperforming *Spy-EM*.

In the three experiments with the biomolecular data set, Bayesian Sets exhibit performance inferior to that of *Spy-EM*. The performance of both schemes is somewhat anomalous: both precision and recall *decrease* with the number of positives revealed to the algorithms. However, the difficulty of the data set must be taken into account when evaluating these results. The data set contains a very small number of positive instances: if only 10% of these (4 of them) are revealed, more than half of the positives in U (24 of them) are extracted. As shown later, SVMs do not come close to replicating this performance (on their own) despite taking much longer to train.

Table 4.3: Entity Set Expansion in KDDCup2001 data set

Positives in U	**Bayesian Sets**			**Spy-EM**		
	Precision	Recall	F-Score	Precision	Recall	F-Score
90%	0.264	0.632	0.372	0.286	0.684	0.403
67%	0.194	0.655	0.299	0.225	0.759	0.347
20%	0.060	0.556	0.109	0.072	0.667	0.130

4.3 Setting up the Experiment

In order to assess the quality of data augmentation and make statistically significant comparisons between different methods, the experiment setup must ensure that data augmentation via different methods is attempted using the same data sets. Figure 4.1 shows a scheme of the system implemented to evaluate data augmentation in Tasks I, II and III.

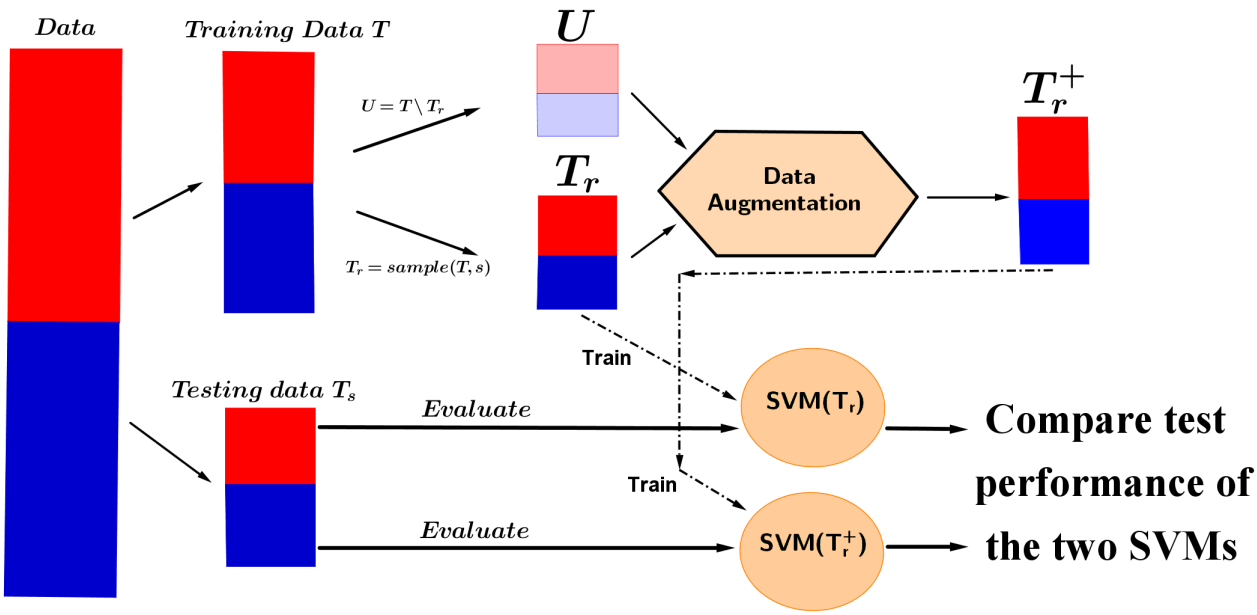


Figure 4.1: The experiment used to evaluate the quality of data augmentation.

The body of examples is first split into the training set T and the testing set T_s. The training set is decomposed into the *reduced training set* T_r and the unlabelled set U, with sampling rates of $s = 10\%$, $s = 33\%$ and $s = 80\%$ used to extract T_r in the experiments (Figure 4.1).

The *pre-augmentation* SVM is trained using the reduced training set T_r. The accuracy and F-Score achieved by this classifier on the testing set T_s act as the **baseline** for classifier performance in the experiments: if the data augmentation schemes aid classification, the performance of the classifiers produced must improve on the baseline classifier.

All three schemes attempt to augment T_r using examples from U, each of them producing a separate augmented data set T_r^+. These data sets are used to train their respective *post-augmentation* SVMs. The differences in performance between the post-augmentation SVMs and the baseline (pre-augmentation) SVM represent the success of the data augmentation achieved in the given experiment.

The **ceiling** for the classifier performance *in all experiments* (for *any sampling rate* s) is determined by training an SVM using the whole training set ($s = 100\%$, $T_r = T$, $U = \emptyset$) and evaluating it using T_s. The post-augmentation SVMs are not likely to outperform this one, as it is trained using the best training set that can be obtained from T_r and U.

4.3.1 Establishing Statistical Significance

If applicable to the specific data set, each of the data augmentation schemes (Bayesian Sets, *Spy-EM*, *Roc-SVM*) is applied to the *same instances (samples)* of T_r and U. The accuracy of the classifiers produced is evaluated against the *same testing set* T_s. As a result, confident assertions about the differences in performance between these three methods can be made.

As long as the testing set T_s is sufficiently large ($|T_s| > 30$ suffices), the *Central Limit Theorem* warrants that the distribution of the errors in that sample can be approximated by a Gaussian distribution. If so, the *Z-Score test for comparing learned hypotheses* can be used to establish the statistical significance of the differences between classifiers. Let $err_{T_s}(C)$ denote the testing error of classifier C evaluated on T_s. Then, the difference in the classification performance between two classifiers C_1 and C_2 is reflected by their *z-score*:

$$\boxed{z = \frac{d}{\sigma_d}} \qquad where \qquad d = |err_{T_s}(C_1) - err_{T_s}(C_2)|$$

$$\sigma_d = \sqrt{\frac{err_{T_s}(C_1)(1 - err_{T_s}(C_1)) + err_{T_s}(C_2)(1 - err_{T_s}(C_2))}{|T_s|}} \tag{4.6}$$

The *z-score* represents the distance from the sample mean to the population mean in units of the standard error: each value of z corresponds to some p such that $P[Z > z] = p$, where $Z \sim N(0, 1)$. For any z, the value of the corresponding p can be computed using a standard table of the cumulative distribution of $N(0, 1)$.

This *p-value* represents the probability (significance level) of the *null hypothesis* which assumes that the two samples (classifications of C_1 and C_2) are drawn from the same underlying distribution. When the *p-value* is less than 0.01, the null hypothesis is rejected. If so, the differences between classification accuracies of C_1 and C_2 are due to an underlying cause with a certainty of $100(1 - p)\%$. Thus, if $p < 0.01$, these differences have *statistical significance*.

Having implemented this evaluation system, we are finally in a position to make statistically relevant claims about the extent to which this data augmentation scheme reduces classification error in the three application areas suggested.

Table 4.4: Data augmentation in Reuters21578 data set

$\|\mathbf{T_r}\| : \|\mathbf{T}\|$	**Baseline SVM**		**Bayesian Sets SVM**		***Spy-EM* SVM**		***Roc-SVM* SVM**	
	Accuracy	F-Score	Accuracy	F-Score	Accuracy	F-Score	Accuracy	F-Score
10%	0.869	0.793	0.556	0.595	**0.922**	**0.904**	**0.905**	**0.859**
33%	0.899	0.854	0.721	0.705	**0.918**	**0.895**	**0.908**	**0.868**
80%	0.939	0.915	0.885	0.851	**0.940**	**0.919**	**0.945**	**0.924**
Ceiling (100%):	Accuracy: **0.946**				F-Score: **0.926**			

4.4 Text Classification

The results of data augmentation with the Reuters21578 data set are shown in Table 4.4. *Spy-EM* and *Roc-SVM* achieve *formidable performance*: both improve on the *baseline* in *all experiments*, usually by a substantial margin. The worse the pre-augmentation SVM performance is and the more unlabelled data is available, the greater the *relative improvement* in classification accuracy is.

The *Z-score test* reveals that the differences between the classifiers produced by *Spy-EM* and *Roc-SVM* are not statistically significant ($p > 0.01$ in all experiments). The differences between baseline SVMs and the ones produced by these two schemes are statistically significant in the first two experiments ($p < 0.01$). This means that the two schemes achieve statistically significant improvements in classification accuracy, with neither of them being the clear favourite.

In cases when only a small amount of unlabelled data is available and the extent to which performance can be improved is negligible (as in the third experiment), these two schemes still manage to improve performance. This *non performance decreasing property* is highly elusive: in such settings, data augmentation schemes usually construct SVMs inferior to the pre-augmentation ones. The reason why these two schemes consistently improve accuracy is that, as explained in the Implementation, the algorithms *purify* the augmented set T_r^+ by removing those elements that appear in the augmented sets of both classes: $T_r^+ = T_r^+ \setminus \{T_{r_1}^+ \cap T_{r_2}^+\}$.

Without purification, the performance of the *Spy-EM* augmentation scheme deteriorates: the scheme actually degrades SVM accuracy in the third experiment (Table 4.5). The *Roc-SVM* scheme was originally designed to perform more con-

Table 4.5: Reuters21578 data augmentation without the *purification step*.

$\lvert\mathbf{T_r}\rvert : \lvert\mathbf{T}\rvert$	**Baseline SVM**		**Bayesian Sets SVM**		***Spy-EM* SVM**		***Roc-SVM* SVM**	
	Accuracy	F-Score	Accuracy	F-Score	Accuracy	F-Score	Accuracy	F-Score
10%	0.855	0.774	0.624	0.642	**0.916**	**0.898**	**0.879**	**0.819**
33%	0.907	0.868	0.783	0.747	**0.934**	**0.916**	**0.929**	**0.899**
80%	0.933	0.906	0.869	0.830	0.922	0.894	**0.942**	**0.921**
Ceiling (100%):	Accuracy: **0.946**				F-Score: **0.926**			

servative set expansion than *Spy-EM*: it exhibits avid performance even without purification, boosting accuracy in all experiments. The differences between these versions of *Spy-EM* and *Roc-SVM* become statistically significant ($p > 0.01$ in 2/3 experiments).

Bayesian Sets using the number of elements included by *Spy-EM* as the cut-off criterion exhibit vastly inferior performance to *Spy-EM* and *Roc-SVM*: their post-augmentation SVMs perform below the baseline in all experiments. Iterative Bayesian Sets were designed in an attempt to improve the original scheme. As shown in Table 4.6, the new scheme greatly outperforms the original one, even constructing an SVM which reaches the ceiling performance in the third experiment. However, it falls short of matching the *consistent quality* or the *extent* of data augmentation achieved by *Spy-EM* and *Roc-SVM*. In fact, the *Z-Score test* reveals that the differences in performance between pre-augmentation SVMs and the ones produced by Iterative Bayesian Sets are *not statistically significant*.

Table 4.6: Reuters21578 data augmentation, regular versus Iterative Bayesian Sets.

$\lvert\mathbf{T_r}\rvert : \lvert\mathbf{T}\rvert$	**Baseline SVM**		**Bayesian Sets SVM**		**Iterative BS SVM**	
	Accuracy	F-score	Accuracy	F-Score	Accuracy	F-Score
10%	0.853	0.766	0.722	0.562	**0.856**	**0.772**
33%	0.915	0.877	0.705	0.621	0.910	0.870
80%	0.932	0.905	0.823	0.758	**0.946**	**0.926**
Ceiling (100%):	Accuracy: **0.946**			F-Score: **0.926**		

Table 4.7: Data augmentation in the biomolecular activity prediction data set.

$\lvert\mathbf{T_r}\rvert : \lvert\mathbf{T}\rvert$	**Baseline SVM**			**Bayesian Sets SVM**			***Spy-EM* SVM**		
	Precision	Recall	F-Score	Precision	Recall	F-Score	Precision	Recall	F-Score
10%	0.143	0.007	0.013	0.182	0.120	**0.145**	0	0	0
33%	0.158	0.020	0.036	0.167	0.113	**0.135**	0.118	0.073	**0.091**
80%	0.191	0.027	0.047	0.165	0.107	**0.130**	0.135	0.080	**0.100**
Ceiling (100%):	Precision: **0.294**			Recall: **0.033**			F-Score: **0.060**		

4.5 Biomolecular Activity Prediction

The goal of this task was to build a classifier identifying active compounds given information about the 3D structure of their molecules. As the number of negatives vastly exceeds the number of positives in the data set, high accuracy no longer warrants that the classifier performs the task well: a function classifying all examples as negatives has a high accuracy while being of no use for this task. Instead, the precision and recall of the positive class are the values to be optimised. The *F-Score* is the ideal metric for comparing the classifiers produced.

Table 4.7 shows the results achieved using the augmentation schemes based on Bayesian sets and *Spy-EM*. Data augmentation via *Roc-SVM* proved to be intractable for this data set given the computational resources at my disposal.

Given the complexity of this data set (139,351 binary features), the amount of data available (1909 examples) does not suffice for the SVM to learn enough about this problem. All baselines, as well as the ceiling of the SVM performance are incredibly poor. This is caused primarily by the *selection bias* present in the data set: T_r contains 2.2%, whereas T_s contains 23.6% of positives. As a result, the SVMs trained are extremely '*reluctant*' to assign positive labels and their performance can benefit immensely from the addition of extra positives into T_r.

In all experiments, both Bayesian Sets and *Spy-EM* raise SVM performance *far above the ceiling value*: Bayesian sets induce at least a *three-fold improvement* of the SVM's *F-Score* in all experiments. The main reason for the enormous scale of this improvement is that adding more positives into T_r mitigates the disastrous effects that the selection bias had on the subsequent testing error.

Wrongly labelled examples introduced into an already small training set can completely ruin subsequent SVM performance. This is the case with the *Spy-EM* scheme in the first experiment: the post-augmentation SVM reverted to labelling all examples as negatives.

The *Z-Score test* confirms that both schemes improve on the baselines' and on the ceiling SVM's performance. The difference between the classifiers produced is statistically significant, and Bayesian Sets are the clear favourite.

The performance of the augmented SVMs is still far from impressive. Nonetheless, the *magnitude* of the *relative improvement in performance* achieved by these data augmentation schemes attests to their ability to reduce classification error in difficult problems without utilising any domain-specific knowledge.

4.6 Supervised Theorem Proving

Data augmentation in this task could only be attempted using the *Roc-SVM* algorithm. Table 4.8 summarizes the results achieved. As there are five different classes, accuracy is the only metric appropriately reflecting the differences in quality between the classifiers produced.

Table 4.8: Data augmentation in the supervised theorem proving data set.

$\lvert\mathbf{T_r}\rvert : \lvert\mathbf{T}\rvert$	**Baseline SVM**	***Roc-SVM* SVM**
	Accuracy	Accuracy
5%	**0.335**	**0.346**
10%	0.316	**0.369**
20%	0.313	**0.361**
33%	**0.345**	*0.313*
50%	0.259	**0.333**
80%	0.326	*0.201*
Ceiling (100%):	0.320	

The results achieved by the baseline and ceiling SVMs are far from impressive, barely improving on the accuracy of 0.298 achieved by a classifier which always

picks the most frequent class. The insufficient amount of training data is the primary cause for such weak performance.

Counter-intuitively, the SVM accuracy does not grow monotonously with the size of the training set used. The explanation for this variability in performance is that the *all-vs-all* voting scheme implemented uses no meaningful mechanism for resolving ties, which occur frequently as only five different classes exist.

The *Roc-SVM* scheme achieves *substantial improvements* in SVM performance when U is significantly larger than T_r (experiments 1, 2, 3). When U is not large enough, the impact *Roc-SVM* has is somewhat arbitrary: it may either degrade (experiments 4,6) or boost SVM performance (experiment 5). Thus, applying the *Roc-SVM* augmentation scheme makes sense if the amount of unlabelled data *greatly exceeds* the amount of labelled data.

The *Z-Score test* reveals that the differences in performance between baseline SVMs and the ones produced by the *Roc-SVM* scheme are statistically significant. The underlying multi-class classifier (based on SVMs) is susceptible to noise introduced during augmentation, so the post-augmentation SVMs do not always improve on the performance achieved by the pre-augmentation ones.

An additional data set from the original work [2] was available for the purposes of evaluation. Examples in U' consist of features of FOL statements that none of the heuristics could prove within a specified time limit. As these statements do not belong to any of the five classes, data augmentation using subsets of U' only degraded the accuracy of the SVM produced (Table 4.9). The more of the examples from U' were introduced, the more the classification accuracy suffered.

Table 4.9: Augmentation using statements not proven by the five heuristics.

$\lvert\mathbf{T_r}\rvert : \lvert\mathbf{U}\rvert$	**Baseline SVM**	***Roc-SVM* SVM**
	Accuracy	Accuracy (average)
1 : 10	**0.3601**	0.321
1 : 2		0.334
1 : 1		0.337
4 : 1		0.340

Chapter 5

Conclusion

The principle goal of this project was to investigate the extent to which *data augmentation* using semi-supervised learning algorithms can improve classification performance. The final data augmentation schemes substantially improve classification accuracy in all of the application areas considered.

The initial success criteria consisted of implementing the Bayesian Sets and *Spy-EM* algorithms and using them for data augmentation in different application areas. The project itself was highly underspecified since two of the data sets were not available until the later stages of the project. In accordance with the evolutionary prototyping paradigm adopted, the data augmentation system was built incrementally, developing progressively harder augmentation schemes and evaluating them using progressively harder data sets until all the success criteria were satisfied.

During the early stages of the project, Bayesian Sets were envisioned as the chief instrument for achieving entity set expansion. Further research indicated that partially supervised learning algorithms such as *Spy-EM* could be used to build data augmentation schemes with formal grounding in computational learning theory (*Appendix A*).

The *RocSVM* algorithm was identified as the only method capable of operating on data sets with real-valued attributes; it was incorporated into the project goals as the final extension. The implementations (*Appendix B*) of these entity set expansion algorithms produced results comparable to those obtained in [6, 8, 10].

The data augmentation schemes built use these algorithms to *augment* the *training data* of the support vector machines used to perform classification. A substantial amount of effort was spent refining Bayesian Sets and establishing an appropriate *cut-off criterion* for their rankings.

The final system built to evaluate these schemes was designed to provide *statistically significant* results about the extent to which data augmentation affects classifier performance. It was successfully implemented and applied to the three application areas proposed:

1. **Text classification:** the schemes based on *Spy-EM* and *Roc-SVM* achieved substantial improvements in classification, both of them producing SVMs with performance levels comparable to the ceiling performance. Bayesian Sets proved to be inadequate for this data set, not managing to improve accuracy even after pseudo relevance feedback was used to improve their original augmentation scheme.

2. **Biomolecular activity prediction:** Bayesian Sets and *Spy-EM* achieved *formidable results*, improving classification accuracy by *an order of magnitude*. Not only did these schemes effectively utilise the unlabelled data to expand the training set, but they managed to mitigate the negative effects that the *selection bias* had on classification, thus confirming the claims given in [9]. In this experiment, Bayesian Sets revealed their underlying potential by outperforming *Spy-EM* using a very ad-hoc cut-off criterion.

3. **Supervised theorem proving:** this part of the project dealt with a *continuous* data set consisting of five different classes. A multi-class classification scheme based on SVMs was implemented and the *Roc-SVM* augmentation scheme successfully improved its classification accuracy using unlabelled data drawn from a separate subset of training data. Subsequently, the system built revealed that the body of unlabelled examples available from the original research project could not be used to achieve data augmentation in the original work.

In addition to providing an open-source library for achieving entity set expansion and data augmentation in arbitrary data sets, work on this project generated some interesting ideas for further research. With the benefit of hindsight and the knowledge gained through work on this project, I would consider including the following two ideas into the original list of project goals:

Spy Bayesian Sets: Bayesian Sets provide the most efficient scheme out of the ones implemented. However, their performance is inhibited by the absence of a clear criterion for establishing the cut-off point in the rankings produced. The use of *spy positives* for extracting *reliable negatives* proved to be instrumental for the success of *Spy-EM* and *Roc-SVM*. Incorporating this idea into (Iterative) Bayesian Sets may yield an entity set expansion algorithm superior to all algorithms considered in this project.

Further evaluation: the results obtained in this project clearly show that the augmentation schemes developed reduce the classification error when support vector machines are used as the underlying classifiers. To establish the applicability of these schemes to real world supervised learning problems, a thorough investigation of their impact on application-specific classifiers tailored to each of the application areas should be conducted.

Bibliography

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, New York, NY, USA, 2006.

[2] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving: learning to select a good heuristic. *Submitted to Journal of Automated Reasoning*, 2013.

[3] V. Castelli and T. Cover. On the exponential value of labeled samples. *Pattern Recognition Letters 16:105-111*, 1995.

[4] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, USA, 2006.

[5] Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software, Volume 9 Issue 5, September 1992*, 1992.

[6] Zoubin Ghahramani and Katherine A. Heller. Bayesian sets. *Advances in Neural Information Processing Systems 18*, 2006.

[7] Benjamin Letham, Katherine Heller, and Cynthia Rudin. Growing a list. *Submitted to ICML*, 2013.

[8] Xiao-Li Li and Bing Liu. Learning to classify text using positive and unlabeled data. *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[9] Xiao-Li Li, Bing Liu, and See-Kiong Ng. Negative training data can be harmful to text classification. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[10] Xiao-Li Li, Bing Liu, Lei Zhang, and See-Kiong Ng. Distributional similarity vs. pu learning for entity set expansion. *Proceedings of the ACL 2010 Conference Short Papers*, 2010.

[11] Bing Liu, Wee Sun Lee, Philip S Yu, and Xiao-Li Li. Partially supervised classification of text documents. *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[12] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Learning to classify text from labeled and unlabeled documents. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[13] Leslie Valiant. A theory of the learnable. *Machine Learning, 20*, 1984.

[14] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications, 16*, 1971.

[15] Vladimir Vapnik and Corinna Cortes. Support-vector networks. *Machine Learning, 20*, 1995.

Appendix A

Computational learning theory

This appendix contains the theoretical arguments used to justify the use of partially supervised learning to achieve data augmentation, as attempted in this project. It contains proofs of two major arguments. Informally:

1. Partially supervised learning can yield *arbitrarily* good classifiers.
2. Unlabelled data carries information relevant to the task of classification.

Furthermore, this appendix introduces the relevant concepts in computational learning theory, such as the PAC learning framework and the VC dimension.

A.1 Theoretical Foundation of Partially Supervised Learning

This section provides a theoretical foundation of partially supervised learning, as framed in Chapter 3.2.

Let X denote the space of feature vectors of possible instances and let $Y = \{0, 1\}$ denote the set of possible outcomes: 1 for positives, 0 otherwise. Assume that all of the examples $(x, y) \in X \times Y$ are independently and identically distributed random variables drawn from some fixed probability distribution over $X \times Y$.

Let $\Pr_D[A]$ denote the probability of event A if the examples are drawn from some probability distribution D. For a finite set of examples L, let $\Pr_L[A]$ denote the probability of event A if the examples are uniformly drawn from L.

The input of the algorithm consists of the positive set P with n_1 elements drawn from $D_{X|Y=1}$, the conditional distribution of all positives, and the unlabelled set U with n_2 elements drawn from D_X, the marginal distribution on X.

The learning algorithm produces the classifier by choosing a function f from the class of functions $F : X \to \{0, 1\}$. The task of learning is equivalent to the task of minimising the expected error of f, that is the probability that the chosen classifier f makes a wrong prediction.

The best possible classifier of all $f \in F$ is the one that minimises the expected error, that is the probability of making wrong predictions:

$$
\begin{aligned}
\Pr[f(X) \neq Y] &= \Pr[f(X) = 1 \wedge Y = 0] + \Pr[f(X) = 0 \wedge Y = 1] \\
&= \Pr[f(X) = 1] - \Pr[f(X) = 1 \wedge Y = 1] + \Pr[f(X) = 0 \wedge Y = 1] \\
&= \Pr[f(X) = 1] - \Pr[Y = 1] + \Pr[f(x) = 0 \wedge Y = 1] + \Pr[f(X) = 0 \wedge Y = 1] \\
&= \Pr[f(X) = 1] - \Pr[Y = 1] + 2\Pr[f(x) = 0 \wedge Y = 1] \\
&= \Pr[f(X) = 1] - \Pr[Y = 1] + 2\Pr[f(x) = 0 \mid Y = 1]\Pr[Y = 1]
\end{aligned}
$$

The first thing to note is that $\Pr[Y = 1]$ is constant. Therefore, minimising the expression given above involves simultaneously minimising $\Pr[f(X) = 0 \mid Y = 1]$ and $\Pr[f(X) = 1]$. However, (X, Y) is drawn from D, which we know nothing

about. Hence, to obtain a method of choosing the best classifier, we need to transform the task of minimising the error across D to one that deals only with the sets P and U. Intuitively, we can only hope to find a *good* approximation to the best classifier given limited knowledge of D (obtained through P and U).

We shall focus on the *noiseless case*: classification settings such that there exists $f_t \in F$ such that $Y = f_t(X)$ for every example (X, Y) drawn from D. The noisy case, where all labels Y are flipped with some probability and the target classification function $f_t \notin F$ is presented in [11].

We want to relate the task of finding a *good* classifier $f \in F$ to the task of minimising $\sum_{i=1}^{n_2} f(X_i)$ (where $X_i \in U$) while maximising $\Pr_P[f(X) = 1]$ at the same time. Therefore, we need to find conditions on the sizes of P and U, n_1 and n_2, under which the function that chooses f by minimising $\sum_{i=1}^{n_2} f(X_i)$ generalises well, that is achieves *good* expected precision and recall across all examples drawn from D.

Formally, we can set this problem in the *probably approximately correct* (PAC) framework [13]: if the target function is any $f_t \in F$, the learning algorithm A needs to produce a function $f \in F$ such that, given enough training data, $\forall \epsilon > 0, \delta > 0. \Pr[E[f(X) \neq f_t(X)] > \epsilon] \leq \delta$. If A can do this for any $f_t \in F$, then A represents a *PAC learning algorithm* for F. However, before we can formally reason about the class of functions F, we need to provide a formal measure of how complicated these classes can be.

Vapnik Chervonenkis (VC) dimension is a measure of the capacity of the probability model used for classification. It is defined as the cardinality of the largest set of examples that the algorithm can *shatter*. An algorithm can *shatter* a set of training examples $T = \{(x_1, y_1) \ldots (x_n, y_n)\}$ if for all potential label assignments to $y_1 \ldots y_n$ there exists some $f \in F$ which makes no classification errors when evaluating T. If T is the greatest set that can be shattered using elements of F, then n represents the VC dimension of F.

The VC dimension reflects the power of the classifier used. A high VC dimension means that a classifier can model complex classifiers but might overfit, whereas a low VC dimension reduces the risk of overfitting but also restricts what the classifier can model. The VC dimension does not depend on the training set used, but solely on the maximum *number* of data points that the model can classify without making an error. Hence, it is an *inherent property* of the function class used to represent the classifiers produced by the learning algorithm. The utility

of VC dimensions stems from the fact that they can be used in conjunction with the training error to obtain a *probabilistic upper bound* on the subsequent testing error for the classifier, that is the classification error on new, unseen data. [14]

For the function class of our classifier $F : X \to \{0, 1\}$ it must hold that $VC(F) \leq \log_2(F)$, as there are two possible label assignments to each $x_i \in X$. Similarly, the Naive Bayes classifier has a VC dimension of no more than $2m+1$, where m represents the number of words used in the classifier [11]. The VC bounds presented henceforth are *not exact*; they are used as an illustration of the rate at which sample sizes have to grow so that the chosen classifier f becomes arbitrarily close to the ideal one, f_t.

Let A' be the algorithm that chooses f by minimising $\sum_{i=1}^{n_2} f(X_i)$. The conditions for A' to be a *PAC learning algorithm* for the *concept class* represented by $F : X \to \{0, 1\}$ are given by the following theorem [11]:

Theorem 1 *Let F be a permissible[1] class of functions of VC dimension d. Let $f_t \in F$ be the target function and let $\nu = E[f_t(X)]$. Let the sets P and U consist of n_1 and n_2 random variables drawn from $D_{X|Y=1}$ and D_X, respectively. Take any $\epsilon > 0$, $\delta > 0$ and assume that $n_1, n_2 > \frac{16}{\epsilon} d \ln(\frac{16e}{\epsilon} \ln \frac{16e}{\epsilon}) + \ln(\frac{24}{\delta})$. Let F' be the subset of F such that all members of F' achieve perfect recall on P. If $f' = \text{argmin}_{f \in F'} \sum_{i=1}^{n2} f(Z_i)$ then with probability of at least $1 - \delta$, it holds that:*
$ER(f') > 1 - \epsilon$ *and* $EP(f') > \frac{(1-\epsilon)\nu}{(1+9\epsilon)\nu+4\epsilon}$ *and* $\mathrm{E[f'(X) \neq f_t(X)]} < (10\nu + 4)\epsilon$.

Proof. To prove this theorem, start from the following one:

Theorem 1.1(Haussler, 1992) *Let $F : Z \to \{0, 1\}$ be a permissible[1] class of functions with VC dimension d. Let $(X_1, Y_1) \ldots (X_n, Y_n)$ be independent identically distributed random variables. For any $\epsilon > 0$, if Φ denotes*

$$\Pr\left[\exists f \in F : \left|\frac{1}{n}\sum_{i=1}^{n} f(X_i, Y_i) - E[f(X, Y)]\right| > \alpha\left(\frac{1}{n}\sum_{i=1}^{n} f(X_i, Y_i) + E[f(X, Y)] + \nu\right)\right]$$

$$\textit{then } \Phi \leq 8\left(\frac{16e}{\alpha\nu}\ln\frac{16e}{\alpha\nu}\right)^{d} e^{-\alpha^2\nu n/8}.$$

Therefore, the probabilistic upper bound on the error decreases with the size of the training set n. Thus, if we set the parameters $\alpha = \frac{1}{2}$ and $\nu = 2\epsilon$, and try to substitute the upper bound on the error with any $\delta > 0$, Haussler's theorem specialises to the following claim:

[1] A function class measurability condition not relevant in practice [11].

Corollary 1 *Let F be a permissible class of functions of VC dimension d and let $T = X_1 \ldots X_n$ be a set of independently and identically distributed random variables. By Haussler's theorem, if $n > \frac{16}{\epsilon}\left(d\ln(\frac{16e}{\epsilon}\ln\frac{16e}{\epsilon}) + \ln\frac{8}{\delta}\right)$, then:*

$$\Pr\Big[\ \exists f \in F : (E[f(X) \neq f_t(X)] > 3\Pr_T[f(X) \neq f_t(X)] + \epsilon) \vee \\ (\Pr_T[f(X) \neq f_t(X)] > 3E[f(X) \neq f_t(X)] + \epsilon) \quad \Big] \ \leq\ \delta.$$

The *two separate events in this expression* distinguish between large training errors and overfitting, thus providing a unifying expression for achieving *Structural Risk Minimisation*. Algebraically, the two events result from removing the absolute values present in Haussler's theorem.

Finally, we are in a position to prove Theorem 1:

From Corollary 1, we know that if $n_1 > \frac{16}{\epsilon}\left(d\ln(\frac{16e}{\epsilon}\ln\frac{16e}{\epsilon}) + \ln\frac{24}{\delta}\right)$, then $\boxed{ER(f') \geq 1-\epsilon}$ with probability of at least $1-\delta/3$, as required in the theorem statement. Consequently, $\Pr[f'(X) = 0 \wedge Y = 1] \leq \epsilon\nu$.

From $f' = \text{argmin}_{f\in F'}\sum_{i=1}^{n2} f(Z_i)$ it follows that $\Pr_U[f'(X) = 1] \leq \Pr_U[Y = 1]$:

$$\begin{aligned}
&\Leftrightarrow\ \Pr_U[f'(X) = 1 \wedge Y = 0] + \Pr_U[f'(X) = 1 \wedge Y = 1] \leq \Pr_U[Y = 1] \\
&\Leftrightarrow\ \Pr_U[f'(X) = 1 \wedge Y = 0] \leq \Pr_U[Y = 1] - \Pr_U[f'(X) = 1 \wedge Y = 1] \\
&\Leftrightarrow\ \Pr_U[f'(X) = 1 \wedge Y = 0] \leq \Pr_U[f'(X) = 0 \wedge Y = 1].
\end{aligned}$$

Similarly (again via Corollary 1), if $n_2 > \frac{16}{\epsilon}\left(d\ln(\frac{16e}{\epsilon}\ln\frac{16e}{\epsilon}) + \ln\frac{24}{\delta}\right)$ then with probability of at least $1-\delta/3$, it holds that:

$$\Pr_U[f'(X) = 0 \cap Y = 1] \leq 3\Pr[f'(X) = 0 \cap Y = 1] + \epsilon \ \textit{and}$$
$$\Pr[f'(X) = 1 \cap Y = 0] \leq 3\Pr_U[f'(X) = 1 \cap Y = 0] + \epsilon.$$

Combining the expressions obtained thus far:

$$\Pr[f'(X) = 1 \wedge Y = 0] \leq 3(\ 3\Pr[f'(X) = 0 \wedge Y = 1] + \epsilon\ ) + \epsilon \leq 9\epsilon\nu + 4\epsilon$$

Hence, with probability of at least $1-\delta$, ($1-\delta/3$, even) the *probability of error* can be bounded by:

$$\Pr[f'(X) \neq Y] = \Pr[f'(X) = 0 \wedge Y = 1] + \Pr[f'(X) = 1 \wedge Y = 0] < \\ < \nu\epsilon + 9\epsilon\nu + 4\epsilon = \boxed{10\epsilon\nu + 4\epsilon}, \text{ as required.}$$

Finally, using these results, one can show that the *expected precision* converges to the optimal one at the rate suggested by Theorem 1:

$$
\begin{aligned}
EP(f') &= \Pr[Y = 1 \mid f'(X) = 1] = \frac{\Pr[Y = 1 \wedge f'(X) = 1]}{\Pr[f'(X) = 1]} \\
&= \frac{\Pr[Y = 1 \wedge f'(X) = 1]}{\Pr[f'(X) = 1 \wedge Y = 0] + \Pr[f'(X) = 1 \wedge Y = 1]} \\
&= \frac{\Pr[Y = 1 \wedge f'(X) = 1]}{\Pr[f'(X) = 1 \wedge Y = 0] + \Pr[Y = 1] - \Pr[f'(X) = 0 \wedge Y = 1]} \\
&\geq \frac{ER(f')\Pr[Y = 1]}{\Pr[f'(X) = 1 \wedge Y = 0] + \Pr[Y = 1]} > \boxed{\frac{\nu - \nu\epsilon}{(1 + 9\epsilon)\nu + 4\epsilon}}
\end{aligned}
$$

□

A.2 Theoretical Foundation of Data Augmentation

In this section, a probabilistic framework for describing the classifiers and observations (training/testing data) [12] will be used to give a theoretical grounding for data augmentation:

A *mixture distribution* is the probability distribution of a random variable whose probability density function is a linear combination of density functions of other random variables. Given a finite set of probability density functions $p_1(x)...p_n(x)$ and weights $w_1...w_n$ such that $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$, the probability density function of the mixture distribution is given by: $f(x) = \sum_{i=1}^{n} w_i p_i(x)$. The cumulative distribution function of the mixture distribution is defined analogously.

A *mixture model* is a mixture distribution that represents the probability distributions of observations in the overall population, without requiring that the set of observations identifies the sub-population that an observation belongs to. Unlike mixture distributions, which are used to derive properties of the overall population from the properties of sub-populations, mixture models are used to perform statistical inference about properties of specific sub-populations given only observations on the entire population, without the sub-population membership information revealed.

We shall assume that all of our data was produced by a mixture model, and that there exists a one to one correspondence between its generative components (underlying random variables) and the classes (corresponding to labels).

Under these assumptions, every observation d_i is generated according to a probability distribution given by a mixture model parametrised by θ. The mixture model consists of generative components $c_j \in C$, where $C = \{c_1, ..., c_{|C|}\}$, and each of these components is parametrised by a disjoint subset of θ. Therefore, an observation is obtained by first selecting a component according to the prior probabilities $P(c_j|\theta)$ and then having that component generate an observation according to its conditional distribution $P(d_i|c_j;\theta)$. Therefore, the probability of any observation d_i can be expressed as a weighted sum over all generative components:

$$P(d_i|\theta) = \sum_{j=1}^{|C|} P(c_j|\theta)P(d_i|c_j;\theta) \tag{A.1}$$

By our assumption of one to one correspondence between classes and generative components, c_j denotes both the j-th generative component and the j-th class. Let y_i denote the class label for the i-th observation. If observation d_i was generated by component c_j, we enforce the correspondence: $c_j = c_{y_i}$. These class labels are known only for labelled data.

In this environment, the problem of learning concept classes is equivalent to estimating the parameters of the mixture model which produced the given set of observations. Therefore, to prove that unlabelled data is useful for learning concept classes, it suffices to show that it carries information about the parameters θ. This will hold if and only if the learning task is not *degenerate*:

$$\exists d_i, c_j, \theta', \theta''.P(d_i|c_j;\theta') \neq P(d_i|c_j;\theta'') \wedge P(c_j|\theta') \neq P(c_j|\theta'') \tag{A.2}$$

The set of tasks which fail this condition (those tasks for which any parametrisation yields the same outcomes) corresponds to the set of tasks for which learning is impossible. High-dimensional mixture models (such as Gaussian mixture models) always satisfy this condition.

To show that the unlabelled data carries information about the parameters θ, we need to show that θ and D are conditionally dependent, D being the random variable representing the probability distribution of the unlabelled observations:

We need to show $P(\theta|D) \neq P(\theta)$. Assume the opposite, that $P(\theta|D) = P(\theta)$. Hence, $\forall\theta'.P(\theta'|D) = P(\theta')$. Therefore, any two parametrisations θ' and θ'' provide the same *class probabilities* for any observation.

Applying Bayes' rule to our assumption, we find that $\forall\theta', \theta'', P(D|\theta') = P(D|\theta'')$.

Substituting this into equation 1.1, we obtain:

$$\forall \theta', \theta''. \sum_{j=1}^{|C|} P(c_j|\theta')P(d_i|c_j;\theta') = \sum_{j=1}^{|C|} P(c_j|\theta'')P(d_i|c_j;\theta'') \tag{A.3}$$

Our non-degeneracy condition(1.2) requires that there must exist an observation d_i such that some of its individual terms in equation 1.3 differ for some parametrisations θ' and θ''. If this is the case, the total probability of the observation must be different with these different parametrisations as well. This contradicts our conditional independence assumption. Therefore, it follows that $P(\theta|D) \neq P(\theta)$. The parametrisations are conditionally dependent on the observations. This means that unlabelled data *does* carry information about the parameters of the generative model.

Appendix B

Matlab code

This section contains the Matlab code samples for some of the algorithms and the data augmentation and evaluation schemes I implemented. This code constitutes around 20% of the code written. The data augmentation evaluation framework, multi-class SVMs, Bayesian Sets, Iterative Bayesian Sets, Rocchio classification, as well as the many supporting functions used for testing, evaluation and data set manipulation have been ommited from this section.

B.1 Spy-EM

```
1  function [finalClassPosteriors, P, U, N, iterationCount] = ...
       SpyEM(FeatureMatrix, PositiveSet, MixedSet, negativeThreshold, ...
       realLabels)
2
3  if( nargin < 4)
4      negativeThreshold = 0.15;
5  % entries with posterior Probability less than negativeThreshold are ...
       put in the reliable negative set - set default value to 0.15.
6  end
7
8  % Positive set contains the array indices of positive elements in the ...
       Feature Vector
9  % FeatureMatrix is the entry * features (binary) matrix
10 % MixedSet represents array indices of unlabeled elements
11 % We are implicitly assuming that P and MS cover all rows of ...
       FeatureMatrix
12 % If they are not, prepareData method can take care of that.
```

```
13
14 numPositives = length(PositiveSet);
15
16 permutationOfPositives = randperm(numPositives);
17
18 setPositives(1:numPositives) = PositiveSet(permutationOfPositives);
19
20 % use s = 10% sampling rate
21 reducedPositives = setPositives(1:round(numPositives*9/10));
22
23 spyPositives = setPositives( (round(numPositives*9/10)+1):numPositives);
24
25 mixedSet = zeros(1, length(spyPositives) + length(MixedSet) );
26
27 mixedSet(1:length(MixedSet)) = MixedSet;
28
29 mixedSet(  (length(MixedSet)+1):length(mixedSet) ) = spyPositives;
30
31 % mixedSet contains the whole mixed set + 10% of the positive set
32
33 numberOfSpies = length(spyPositives);
34 % the number of spy elements at the end of the (new) mixedSet
35
36 reorderedFeatureMatrix(1:length(reducedPositives), :) = ...
      FeatureMatrix(reducedPositives, :);
37
38 reorderedFeatureMatrix( length(reducedPositives)+1 : ...
      (length(reducedPositives) + length(mixedSet)), :) = ...
      FeatureMatrix(mixedSet, :);
39
40 initialEMLabels = InitialEM (reorderedFeatureMatrix, ...
      1:length(reducedPositives), length(reducedPositives)+1: ...
      (length(reducedPositives) + length(mixedSet)));
41 % [1,length(reducedPositives)] is the set of posteriors of positives ...
      used in training the classifier, and the rest is the mixedSet. ...
      Indexing is the same as in reducedPositives, so ...
      IEMLabelsIndexed(1) is reducedPositives(1) and ...
      IEMLabelsIndexed(length(reducedPositives)+1) is mixedSet(1)
42
43 % Place all members of initialEMClasses with posterior probability ≤ ...
      t into the Reliable Negative set, and restore the original ...
      positive set (just use the old one, PositiveSet)
44
45 initialEMLabels = initialEMLabels( (1+length(reducedPositives)) : ...
      (length(initialEMLabels) - numberOfSpies));
46
47 [valuesPosterior, valuesIndices] = sort(initialEMLabels);
```

```
48 % sort by posterior probabilities in order to easily identify the ...
       likely negatives set. No need for binary search, linear is fast ...
       enough!
49
50 bound = length(valuesPosterior);
51
52 for i=1:length(valuesPosterior)
53     if(valuesPosterior(i)≥negativeThreshold)
54         bound = i - 1;
55         break;
56     end
57 end
58
59 % since valuesIndices are indexes in the mixedSet, which is an array ...
       of indices, that means that we need to feed these into N/U with ...
       these indices:
60 negativeSet = MixedSet(sort(valuesIndices(1:bound)));
61
62 unlabeledSet = ...
       MixedSet(sort(valuesIndices((bound+1):(length(valuesIndices)))));
63
64 positiveSet = PositiveSet;
65
66
67 % Sanity check to see if all elements were divided into these three ...
       sets by checking that the sum is what it should be:
68  d = length(positiveSet) + length(negativeSet) + length(unlabeledSet);
69  isequal( sum(positiveSet) + sum(negativeSet) + sum(unlabeledSet), ...
        d*(d+1)/2)
70
71 P = positiveSet;
72 U = unlabeledSet;
73 N = negativeSet;
74
75 % what the sEM algorithm does next is train an NB classifier only ...
       using the sets P and N. This classifier is then used to classify ...
       all of U. Then, given these posteriors -> labels, we iteratively ...
       construct new NB classifiers, as in iEM, using all three sets ...
       with their labels, until they converge. We can't reuse the iEM ...
       code here, as it resets the positive set to 1 in each iteration, ...
       so elements of U labeled 1 by the initial NB won't be able to ...
       change label - which is incorrect!!!
76
77 newLabels = zeros(1, ...
       length(positiveSet)+length(negativeSet)+length(unlabeledSet));
78 newLabels(1:length(positiveSet)) = 1;
79
```

```
clear reorderedFeatureMatrix;

[reorderedFeatureMatrix, newLabels] = prepareData(FeatureMatrix, ...
    positiveSet, negativeSet, newLabels);
% obtain the feature matrix just for P and N, get newLabel to resize ...
    to get rid of its elements from unused set.

%train the classifier using only P and N
PN_NBClassifier = NaiveBayes.fit(reorderedFeatureMatrix, newLabels, ...
    'Distribution', 'mn');

% Now, we need to classify all elements of U, N according to the new ...
    classifier, and then run the EM algorithm, fixing posteriors of P ...
    to 1.
posteriors = PN_NBClassifier.posterior(reorderedFeatureMatrix);

labelsEM = zeros(1, ...
    length(positiveSet)+length(negativeSet)+length(unlabeledSet));
% this will be the list of labels for our EM algorithm

labelsEM(positiveSet) = 1;
% the elements in the positiveSet are fixed to have posterior ...
    probability 1

if(¬isempty(negativeSet) )% if the negative set is NOT empty:
    labelsEM(negativeSet) = binarizePosteriors(posteriors( ...
        length(positiveSet)+1:length(positiveSet)+length(negativeSet) ...
        ,2));
    disp('negatives not empty, all good')
end

% the negative ones are free to be relabeled, as will be those from U
posteriorsU = PN_NBClassifier.posterior(FeatureMatrix(unlabeledSet, :));

if(size(posteriorsU, 2) < 2) % in case empty or fully determined
    finalClassPosteriors = labelsEM; % if determined, U is all 0
    disp('massive bug2')
    return;
end

labelsEM(unlabeledSet) = binarizePosteriors ( posteriorsU(:,2));

oldLabels = zeros(size(labelsEM));

numIterations = 0;

while ( not(isequal(oldLabels, labelsEM)))
```

```
118
119     if(numIterations > 30)
120         str = ['Cut SEM optimization after 30 iterations - the number ...
                of different labels is:', num2str(sum(abs(oldLabels - ...
                labelsEM)))];
121         disp(str)
122         break;
123     end
124
125     EM_NBClassifier = NaiveBayes.fit(FeatureMatrix, labelsEM, ...
            'Distribution', 'mn');
126
127     posteriorProbabilities = EM_NBClassifier.posterior(FeatureMatrix);
128
129     oldLabels = labelsEM;
130
131     labelsEM = binarizePosteriors(posteriorProbabilities(:,2));
132
133     labelsEM(positiveSet) = 1;
134     % make sure to keep these fixed to 1.
135
136     numIterations = numIterations + 1;
137
138     % The next two lines, and the realLabels that might be passed in as
139     % argument, can be used to track how subsequent classifiers ...
            improve the
140     % estimation metrics! will only write this if we pass these as ...
            argument
141
142 end
143
144 iterationCount = numIterations;
145 finalClassPosteriors = posteriorProbabilities(:,2);
146
147 finalClassPosteriors(positiveSet) = 1;
```

B.2 Roc-SVM

```
1  function [finalClass, reliableNegatives, svmClassifierFinal, ...
       svmFirstResults, SVMFirstClassifier] = RocSVM(FeatureMatrix, ...
       PositiveSet, MixedSet, kkdThreshold, numIter, numClusters, alpha, ...
       beta)
2
3  % This code implements the Rocchio SVM (with k-clustering) PU ...
       learning algorithm given in (Learning to Classify Texts using PU ...
       data, Li & Liu)
4
5  % kkdThreshold is the proportion of points allowed to be on the wrong ...
       side of the separating hyperplane. The lower it is, the better ...
       result the algorithm should produce. Furthermore, the lower it ...
       is, the longer it takes to converge ( exponentially increases the ...
       running time). A value that is too low leads to overfitting.
6
7  % numClusters - parameter used for k-means clustering after Rocchio, ...
       default 10
8
9  % reliableNegatives - the set of RN identified in step 1 of the algorithm
10
11 if(nargin < 6)
12      numClusters = 10; alpha = 16; beta = 4;
13 end
14
15 if(nargin < 5)
16     numIter = 1000000;
17 end
18
19 if(nargin < 4)
20     kkdThreshold = 0;
21 end
22
23 numEntries = size(FeatureMatrix, 1);
24
25 reliableNegatives = Rocchio(FeatureMatrix, PositiveSet, MixedSet, ...
       alpha, beta);
26
27 if(length(reliableNegatives) == 0)
28     disp('degenerate svm - no RNs')
29     finalClass = zeros(1, size(FeatureMatrix, 1));
30     return;
31 end
32
```

```
33 KMeansCentroids = kmeans(FeatureMatrix(reliableNegatives, :), ...
       numClusters, 'start', 'sample', 'emptyaction', 'singleton');
34
35 temporaryLabels = zeros(numClusters, length(KMeansCentroids));
36
37 for i = 1:length(KMeansCentroids)
38     temporaryLabels(KMeansCentroids(i), reliableNegatives(i)) = 1;
39 end
40
41 NegativeCluster = cell(numClusters);
42
43 for i = 1:numClusters
44   NegativeCluster{i} = LabelToArray(temporaryLabels(i, :));
45 end
46
47 % Given the cluster of each entry, produce centroids for each cluster:
48
49 positiveCentroids = zeros(numClusters, size(FeatureMatrix, 2), 'single');
50 negativeCentroids = zeros(numClusters, size(FeatureMatrix, 2), 'single');
51
52 for i = 1:numClusters
53    [positiveCentroids(i, :), negativeCentroids(i, :)] = ...
          calculateCentroids(FeatureMatrix, PositiveSet, ...
          NegativeCluster{i}, alpha, beta);
54 end
55
56
57 % similarityMatrix[i,j] = cosine_similarity(centroid i, entry j)
58 similarityMatrixPositive = zeros(numClusters, numEntries, 'single');
59 similarityMatrixNegative = zeros(numClusters, numEntries, 'single');
60
61 % the list of norms of all entries in the FeatureMatrix, needed for ...
       calculation
62 featureNorms = sqrt(sum(FeatureMatrix.^2,2))';
63
64 for i = 1:numClusters
65
66     % calculate the dot product of the i-th centroid with all entries
67     similarityMatrixPositive(i, :) = (sum(FeatureMatrix .* ...
           repmat(positiveCentroids(i,:), size(FeatureMatrix, 1), 1), 2));
68     similarityMatrixNegative(i, :) = (sum(FeatureMatrix .* ...
           repmat(negativeCentroids(i,:) , size(FeatureMatrix, 1), 1), 2));
69
70     % now, divide the dot product by the norm of the respective document:
71     similarityMatrixPositive(i, :) = similarityMatrixPositive(i, :) ...
           ./  featureNorms;
```

```
72      similarityMatrixNegative(i, :) = similarityMatrixNegative(i, :) ...
            ./ featureNorms;
73
74      % in order to complete the cosine similarity calculation, we need ...
            to divide by norm of centroid:
75      similarityMatrixPositive(i, :)  = similarityMatrixPositive(i, :) ...
            / norm(positiveCentroids(i,:));
76      similarityMatrixNegative(i, :) = similarityMatrixNegative(i, :) / ...
            norm(negativeCentroids(i,:));
77
78  end
79
80  % Left to determine which elements of RN stay in the negative set. ...
        For each entry of the unlabeled set, check if the maximal of its ...
        negative similarities is greater than the maximum of its positive ...
        similarities.
81
82  % Precompute these maxima/minima:
83  maxSimilarityPositive(1:numEntries) = max( ...
        similarityMatrixPositive(:, 1:numEntries));
84  maxSimilarityNegative(1:numEntries) = max( ...
        similarityMatrixNegative(:, 1:numEntries));
85
86  similarityDifference = maxSimilarityNegative - maxSimilarityPositive;
87  % entries which have a positive value belong in the negative set
88
89  % As in Rocchio, we need to make sure to remove positives from RN:
90  similarityDifference(PositiveSet) = -1;
91
92  reliableNegatives = sort(LabelToArray(similarityDifference));
93
94  % Now, construct the set Q, which is the set of the remaining ...
        unlabeled entries (the set of their indices):
95
96  Q_Label = zeros(1, numEntries);
97
98  Q_Label(MixedSet) = 1;
99
100 Q_Label(reliableNegatives) = 0;
101
102 Q = LabelToArray(Q_Label);
103
104 % The RocSVM Method returns either the first classifier produced (if ...
        the SVM construction step failed), or the last classfier. Hence, ...
        save the classifications of the first classifier, and then ...
        iterate onwards:
105
```

```
svmLabels = zeros(1, length(PositiveSet) + length(reliableNegatives));
svmLabels(1:length(PositiveSet)) = 1;
svmLabels( ( length(PositiveSet)+1 ) : ( ...
    length(PositiveSet)+length(reliableNegatives)) ) = -1;

SVMTrainingSet = prepareData(FeatureMatrix, PositiveSet, ...
    reliableNegatives, zeros(1, numEntries) );

SVMFirstClassifier = svmtrain( full(SVMTrainingSet), svmLabels, ...
    'kktviolationlevel', kkdThreshold, 'options', statset('MaxIter', ...
    numIter )); %'Display', 'iter'
SVMClassifier = SVMFirstClassifier;

svmResults = svmclassify(SVMClassifier, FeatureMatrix);
svmFirstResults = binarizePosteriors( svmResults, 1 );

svmResults(PositiveSet) = 0;
svmResults(reliableNegatives) = 0;

W = LabelToArray(-svmResults);

% As long as there remain negative elements extracted from Q in the ...
    SVM step: designate the set of these as W, remove them from Q and ...
    repeat the step.

while(not(isempty(W)))

    % First, remove W from Q:
    tempLabels = zeros(1, numEntries);
    tempLabels(Q) = 1;
    tempLabels(W) = 0; % W is a subset of Q, so just remove them from it.
    Q = LabelToArray(tempLabels); % gets the new value of Q

    reliableN = zeros(1, length(reliableNegatives) + length(W)); % ...
        the new set of reliable negatives
    reliableN(1:length(reliableNegatives)) = reliableNegatives;
    reliableN( (length(reliableNegatives)+1):(length(reliableN)) ) = W;
    reliableNegatives = reliableN;

    % svmLabels will contain the training labels for the SVM
    svmLabels = zeros(1, length(PositiveSet) + ...
        length(reliableNegatives));
    svmLabels(1:length(PositiveSet)) = 1;
    svmLabels( ( length(PositiveSet)+1 ) : ( ...
        length(PositiveSet)+length(reliableNegatives)) ) = -1;

```

```
143     % SVMTraining set will contain the part of FeatureMatrix with P ...
            and RN
144     SVMTrainingSet = prepareData(FeatureMatrix, PositiveSet, ...
            reliableNegatives, zeros(1, numEntries) );
145
146     % Train i-th classifier:
147     SVMClassifier = svmtrain( full( SVMTrainingSet) , svmLabels, ...
            'kktviolationlevel', kkdThreshold, 'options', ...
            statset('MaxIter', numIter));
148     size(W)
149     % Use S_i to classify all the entries:
150     svmResults = svmclassify(SVMClassifier, FeatureMatrix);
151
152     svmResultsW = -svmResults;
153     svmResultsW(PositiveSet) = 0;
154     svmResultsW(reliableNegatives) = 0;
155     svmResults = binarizePosteriors(svmResults, 1); % -1 for N needs ...
            to be 0
156     W = LabelToArray(svmResultsW);
157
158 end
159
160 % Left to check whether svmResults( PositiveSet ) has more than 5% of ...
        negatives: if so, the SVM classifier malfunctioned -> revert to S1.
161
162 svmR = sum(svmResults(PositiveSet)) / length(PositiveSet); % the ...
        count of positives classified as such by the last classifier
163
164 if(svmR < 0.95)
165
166     finalClass = svmFirstResults; % positives labeled with 1, ...
            negatives with -1.
167     svmClassifierFinal = SVMFirstClassifier;
168     str = ['Reverted to original classifier - SVM iterations ...
            converged with ', num2str(svmR)];
169     disp(str);
170
171 else
172
173     finalClass = svmResults; % will labeled with 1, negatives with -1.
174     svmClassifierFinal = SVMClassifier;
175     str = ['Using the final classifier. SVM iterations converged with ...
            ', num2str(svmR)];
176     disp(str);
177
178 end
```

B.3 Multi-class augmentation

```
function [augmentedResults, originalResults] = ...
    assessMultiClass(TrainFM, TrainLabels, TestFM, TestLabels, ...
    UnlabeledFM, kkdThreshold)

numClasses = length(unique(TestLabels));

originalLabeling = MultiClassSVM(TrainFM, TrainLabels, TestFM, 0);

originalResults = quality(originalLabeling, TestLabels);

% We need to extract new positives of every class!

newPositives = cell(numClasses);

for i = 1 : numClasses

    idxAccess = LabelToArray((TrainLabels == i));

    augFM = zeros(length(idxAccess) + size(UnlabeledFM, 1), ...
        size(TrainFM, 2));

    augFM(1:size(UnlabeledFM, 1), :) = ...
        UnlabeledFM(1:size(UnlabeledFM, 1), :);

    augFM( size(UnlabeledFM, 1) + 1 : size(UnlabeledFM, 1) + ...
        length(idxAccess), :) = TrainFM(idxAccess, :);

    Labels = zeros(size((augFM), 1), 1);

    Labels( size(UnlabeledFM, 1) + 1 : size(UnlabeledFM, 1) + ...
        length(idxAccess) ) = 1;

    % as of now, we can only use RocSVM:
    [¬, ¬, newPositives{i}] = augmentData(augFM, Labels, [], 0, 0, 1, ...
        kkdThreshold, 10000000);

    newPositives{i} =  newPositives{i} (LabelToArray( newPositives{i} ...
        ≤ size(UnlabeledFM, 1) ))  ;

end

newPos = cell(numClasses);

```

```
for i = 1 : numClasses
    newPos{i} = newPositives{i};
end

for i = 1 : numClasses

    numMemb = size(newPositives{i})

    for j = 1 : numClasses

        if i ≠ j

            newPositives{i} = setdiff( newPositives{i}, newPos{j});

        end
    end
end

for i = 1 : numClasses

    if ( ¬isempty( newPositives{i} ) )

        str = ['Number of new positives of class', num2str(i), 'is:', ...
            num2str(size(newPositives{i}, 2))];
        disp(str)

        TrFM = zeros( size(TrainFM, 1) + size(newPositives{i}, 2), ...
            size(TrainFM, 2));
        TrFM(1:size(TrainFM, 1), :) = TrainFM( 1 : size(TrainFM, 1), :);
        TrFM( size(TrainFM, 1) + 1 : size(TrFM, 1), :) = UnlabeledFM( ...
            newPositives{i}, : );

        TrainL = zeros( size(TrainFM, 1) + size(newPositives{i}, 2), 1);
        TrainL(1:size(TrainFM, 1), :) = TrainLabels( 1 : ...
            size(TrainFM, 1));
        TrainL(size(TrainFM, 1) + 1 : size(TrFM, 1)) = i;

        TrainFM = TrFM;
        TrainLabels = TrainL;
    end

end

newLabelling = MultiClassSVM(TrainFM, TrainLabels, TestFM, 0);

augmentedResults = quality(newLabelling, TestLabels);
```

Appendix C

Project Proposal

Introduction and Description of the Work

Fundamentally, there exist two different types of tasks in machine learning, based on the desired outcome of the machine learning algorithm: *unsupervised* and *supervised* learning.

Unsupervised learning algorithms are presented with the list of inputs to the function, $x_i \in X$, and the goal of the algorithm is to infer interesting structure from this data. It is typically assumed that the points are drawn i.i.d.(independently and identically distributed) from a common distribution on X. Essentially, the task of unsupervised learning is to estimate the probability density function of that distribution (assuming one exists).

Supervised learning algorithms generate a function $f : X \to Y$ that maps potential inputs to desired outputs, also known as labels. The desired function maps any $x_i \in X$ to a specific label $y_i \in Y$. The algorithm is trained using a list of input-output pairs (x_i, y_i). In the case where the set of labels Y is finite, the problem is known as classification; otherwise, the problem is known as regression.

The basic assumption of supervised learning is that the (x_i, y_i) pairs are independent and identically distributed random variables drawn from an underlying probability distribution on $X \times Y$. The function f obtained by learning from the training data tries to capture the probability density function of this distribution. The main condition required for supervised learning to work (i.e. to be able to generalise from a finite training set to potentially infinitely many "unseen" test

cases) is that the *smoothness assumption of supervised learning* holds. Informally, the condition is that if the points x_1 and x_2 are *close*, their corresponding outputs y_1 and y_2 should be *close* as well.

After estimating $p(x|y)$ using an unsupervised learning procedure, the predictive density can then be inferred using *Bayes theorem*:

$$p(y|x) = p(y)\frac{p(x|y)}{\int_y p(x|y)p(y)\mathrm{d}y} \tag{C.1}$$

This equations forms the basis of *Bayesian inference*, a method in which Bayes' rule is repeatedly used to update the probability estimate for a hypothesis as additional evidence is obtained. It rests on the *Bayesian interpretation* of the concept of probability, which regards it as an abstract quantity either theoretically assigned for representing a certain state of knowledge, or calculated from previously assigned probabilities. This contrasts the traditional *frequentist* view which defines the probability of an event as the limit of its relative frequency in a large number of trials.

Semi-supervised learning algorithms combine these two approaches. The data set X consists of $X_l = (x_1, ..., x_l)$, the set for which the set of labels $Y_l = (y_1, ..., y_l)$ is known, and the set of unlabeled data $X_u = (x_{l+1}, ..., x_{l+u})$. Given this data sets as input, the algorithm tries to create the same prediction function as in the case of supervised learning. However, in addition to using the labeled data for training, it attempts to use the unlabeled data to increase the quality of the prediction. The unlabeled data can increase the quality of the prediction only if the distribution of $p(x)$ observed from the unlabeled data carries information that is useful in the inference of $p(y|x)$.

To formalize this notion, we generalise the supervised learning case to obtain the *semi-supervised smoothness assumption*: If x_1 and x_2 are points that are *close* to each other in a *high density region*, then their corresponding outputs y_1 and y_2 should be *close* as well. By transitivity, it follows that all points in the same *cluster* are likely to have their respective outputs clustered as well.

Data augmentation refers to a family of methods that construct iterative algorithms by introducing unobserved training data. In general, constructing data augmentation schemes that result in both simple and fast algorithms is a highly non-trivial task, as successful strategies vary greatly with the type of data models used in the specific application. It is a common problem in semi-

supervised learning, and the following problem is one of its best known instances, often encountered in problems such as classification of text documents:

Entity set expansion problem: Given a set S of seed entities of a particular class S and a set D of candidate entities, we wish to determine which of the entities in D belong to S. In other words, we expand the set S based on the given seeds.

There is a class of semisupervised learning algorithms that learns from a set of *positive* examples P and a set of *unlabeled* examples $U(PU\ learning)$. The key characteristic of PU learning is that there is *no negative training example* available for learning. This is contrary to the typical classification context, in which the training data consists of a series of examples of *every possible class*.

The problem of *entity set expansion* maps directly to PU learning: the set of seed entities S is the set of positive examples P, whereas the set of candidate entities D maps directly to the set of unlabeled examples U. This means that the existing body of algorithms developed for *PU learning* is directly applicable to entity set expansion, providing a good way to assess the performance of novel approaches to this problem by comparing their performance to the already established ones.

Bayesian sets(Ghahramani et al, 2005) take an information retrieval inspired approach to the problem of entity set expansion. A *relevance criterion* for each $d \in D$ is defined, calculated, and used to sort all the unlabeled data in D in order of the likelihood that they are of the same class as the elements of the candidate set S. This relevance criterion is based on the current understanding of patterns of generalisation in human cognition:

$$score(\mathbf{x}) = \frac{p(\mathbf{x}|D_s)}{p(\mathbf{x})} \tag{C.2}$$

where $D_s \subset D$ represents the concept class of entities of class S. D represents the universe of all entities in this data model.

The input of the Bayesian sets algorithm consists of the set of items D, a probabilistic model $p(\mathbf{x}|\theta)$, $x \in D$, and a prior on the model parameters $p(\theta)$. The query given to the algorithm is the set $D_s = \{x_i\} \subset D$.

The algorithm calculates the $score(x)$ for all $x \in D$. Subsequently, it returns this list of elements as output, sorted by decreasing scores. Each of these scores represents the probability that the specific element belongs to the concept class D_s.

Clearly, the core complexity of this algorithm lies in computing the *score* function. For simple models in which $p(\mathbf{x}|\theta)$ has an independent Bernoulli distribution for all of the candidate's features ($x_i \in \mathbf{x}$) the computation of all the scores can be reduced to a single sparse matrix multiplication. Even using this simple model Bayesian sets exibit avid performance.

The mathematics involved in the computation of the scores is quite involved and will not be further examined here. However, it is important to note that the computational complexity of Bayesian sets is linear with respect to the size of D, even when $p(\mathbf{x}|\theta)$ uses a distribution from the exponential family. The Bayesian sets algorithm is very flexible; it can be used for a wide variety of data (discrete, continuous, text) and with different probabilistic models.

S-EM algorithm: An interesting classifier alternative to Bayesian sets (which are a *ranking algorithm*), is presented in (Partially Supervised Classification of Text Documents, Liu et al., 2002). The algorithm proposed is called the *S-EM algorithm*. It is based on repeated applications of the *Naive Bayesian* classifier and the *Expectation-Maximisation* algorithm, details of which will be omitted here. The algorithm first runs a reinitialisation phase which "grows" the positive set P using an NB classifier trained with the current perception of P to find other likely members of the positive set. It repeats this step until the algorithm converges, i.e. the naive classifier can't find any new members of the positive set.

Subsequently, the algorithm proceeds by using a "spy" technique to identify *reliable negatives(RN)* in the unlabeled set U. This is achieved by training another NB classifier with only a portion of the positive set $P_1 \subset P$. Then, we can track the behaviour, i.e. the subsequent classification of the members of $P \setminus P1$ by the Naive Bayesian classifier. If we assume that the hidden positives in U that we're trying to identify behave similarly to elements of $P \setminus P1$, we can identify some *reliable negatives* from the set U, which are then used to re-train the classifier until it converges to produce the final entity set. The paper contains a theoretical foundation for partially supervised classification, as well as an experimental evaluation of S-EM algortihm's F-Score $F = \frac{pr}{p+r}$, where p is the precision, and r is the recall achieved. This gives us ample justification to use this algorithm as a frame of reference for assessing the performance of Bayesian sets.

(Distributional Similarity vs. PU Learning for Entity Set Expansion, Li et al., 2010) claims that the bidirectional pull of the constructed classifier makes it

superior to ranking methods such as Bayesian sets. Furthermore, they propose an interesting, non-trivial assertion that classification methods produce superior results to ranking methods in general. Even though the S-EM algorithm is at the moment applicable only to text classification, it would be interesting to try to extend it to other data sets supported by Bayesian sets and compare their performance on those data models in an attempt to gain further insight into this proposition.

Resources Required

The most suitable language for implementing these algorithms is *Matlab*. *R* is a viable alternative. We shall aim to implement the whole project in Matlab, potentially adding R interoperability for certain aspects of the projects.

The project will be completed in cooperation with Dr Holden's collaborator, Dr Matthew Trotter of the Celgene Institute for Translational Research Europe (CITRE), Sevilla, Spain. The data set required to examine protein localization and interactions will be provided by Dr Holden's current collaborators at the Anne McLaren Laboratory for Regenerative Medicine. The data set for examining statements used with the theorem prover will be provided by Dr Holden.

Starting Point

Artificial Intelligence I taken in Part 1B.

I will have to familiarise myself with most of Part II Artificial Intelligence II course, as well as do further reading into the field of machine learning, especially the area of semi-supervised learning.

Furthermore, there is a body of approximately five to ten research papers that contain work relevant to this project that I will have to familiarise myself with.

Substance and Structure of the Project

The project involves performing an initial study of the extent to which the Bayesian Sets algorithm of Ghahramani and Heller might be extended and applied within the area of drug design, the identification of protein localization and

interactions, as well as classifying statements used in automated theorem proving. More specifically, we aim to address the problem of *data augmentation* (*entity set expansion*), as defined in the introduction.

As an *optional* part of the project, we will try to create a variation of the S-EM algorithm that would be applicable to problems other than text classification.

The principal goal of the project is to implement a properly tested, documented and optimized open source software library allowing the Bayesian Sets algorithm and extensions to be applied easily to new data sets.

As for the optional part of the project, in case we are successful in deriving a variation of the S-EM algorithm required for application to data models other than text classification, we shall aim to integrate it in the library as well.

The data sets provided will allow us to address problems in:

1. Drug design: Achieving data augmentation for the supervised algorithms applied in this area. The dataset will be provided by Dr Matthew Trotter of the Celgene Institute for Translational Research Europe (CITRE), Sevilla, Spain.

2. Regenerative medicine: Identifying protein localisation and interactions in the context of regenerative medicine. The dataset will be provided by Dr Holden's collaborators at the Anne McLaren Laboratory for Regenerative Medicine.

3. Classification of statements used for a supervised theorem prover: Recent work by Dr Holden applied machine learning techniques to first order theorem proving. We will try to achieve data augmentation on the set of statements attempted by the theorem prover.

Subsequently, the aim of the project is to investigate the performance of the Bayesian set algorithm (and *potentially* our variation of S-EM), producing quantitative comparisons of their performance using the three different data sets available to us.

To quantify how much our data augmentation affected the performance, we will first compute the *F-Score*, *prediction@N* and other metrics to assess the performance of the algorithms implemented. Subsequently, the original algorithms used in these different applications will be applied to our augmented data set to measure the change in their performance.

If our attempt at the extension of S-EM to tasks other than text classification is successful, an examination similar to the one taken in (Li et al., 2010) will be taken to further examine their claim that classification methods are superior to ranking ones.

The project has the following main sections:

1. Familiarisation with the programming language to be used (*Matlab*) and the surrounding software tools. Detailed examination of the Bayesian sets and the S-EM algorithm. An attempt to derive further variations of Bayesian sets, as well as an attempt to extend the S-EM algorithm to problems other than text classification.

2. Development and implementation of the Bayesian sets algorithm and its extensions derived in the first phase. Potentially, development and implementation of the modified S-EM algoritm as well. Creating, optimising, and testing an open source library which will allow direct application of these methods to new, general data sets.

3. Application of the developed software to the data sets available. Thorough examination and quantification of their performance.

4. Experimental evaluation of the performance changes in the three algorithms used with the original data sets after data augmentation.

5. Writing the Dissertation.

Success Criteria

The following should be achieved:

- Development and implementation of the original Bayesian sets algorithm, as well as its further derivations required for application to the three data sets available.

- Implementing, testing, optimising and documenting an open source library to allow application of Bayesian sets to new data sets in order to achieve data augmentation.

- Application of our implementation of Bayesian sets to the three data sets available; measuring how well data augmentation is achieved.

- Experimental investigation of the extent to which data augmentation achieved leads to an improvement in performance in the original application areas.

Extensions

- Derivation and implementation of an extension to the S-EM algorithm that will allow application to problems other than text classification.
- Inclusion of this algorithm as part of the open source library created.
- Application of this algorithm to the data sets available. Measuring the quality of data augmentation achieved.
- Comparison of S-EM algorithm's performance with the performance of Bayesian sets.

Evaluation

The success of data augmentation achieved can be measured using standard metrics such as the *F-score*(prediction, recall), *Prediction@N* and others.

The change in the *subsequent performance* of the classifiers used is harder to measure, due to the different nature of algorithms used in the three application areas considered.

We would like to obtain a *uniform* measure of success of data augmentation across these different data sets. To achieve that, we will first apply a *standard classifier implementation* such as SVM^{light} to all three data sets. Subsequently, we will measure (the change in) its performance after our attempt at data augmentation.

This should yield a result that will be representative of the impact data augmentation can have in these application areas.

Timetable and Milestones

19.10.2012 - 18.11.2012

Background reading in machine learning and semi-supervised learning. Thorough examination of the Bayesian sets algorithm, S-EM algorithm, as well as any other relevant scientific literature. Attempt at deriving further variations of Bayesian sets, and an attempt at generalising the S-EM algorithm.

Milestones: Pseudocode and theoretical background required to start implementation.

19.11.2012 - 16.12.2012

Familiarisation with Matlab and the relevant libraries. Implementation of Bayesian sets and the original S-EM algorithm.

Milestones: A first prototype of the Bayesian sets algorithm.

17.12.2012 - 31.12.2012

Further work on implementing the Bayesian sets and (*potentially*) the modified S-EM algorithm. Testing on the text classification problems available in the literature.

Milestones: A tested version of Bayesian sets. A prototype of the modified S-EM algorithm.

1.1.2013 - 31.1.2013

Final work on the S-EM algorithm. Optimising, testing and documenting the library so that it allows application of these two algorithms to text classification.

1.2.2013 - 15.2.2013

Adapting the library to allow application to the three data sets available.

Milestones: The library can now be easily applied to general data sets.

16.2.2013 - 28.2.2013

Applying the algorithm to the available data sets and measuring the success of data augmentation achieved.

Milestones: Results of evaluation now available.

1.3.2013 - 31.3.2013

Examination into how the use of the newly obtained (augmented) data sets improves the performance of the algorithms used in the three application areas.

1.4.2013 - 15.4.2013

Writing the dissertation.

Milestone: Draft of the Dissertation.

16.4.2013-1.5.2013

Final work on the Dissertation.

Milestone: Final version of the Dissertation. Final version of the library implemented submitted.