

RICE UNIVERSITY

**Statistical Machine Learning for Text Mining
with Markov Chain Monte Carlo Inference**

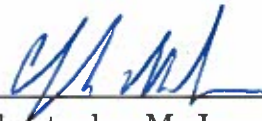
by

Anna Drummond

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:



Christopher M. Jermaine, Chair
Associate Professor of Computer Science



Luay K. Nakhleh
Associate Professor of Computer Science



Swarat Chaudhuri
Assistant Professor of Computer Science



Genevera Allen
Assistant Professor of Statistics

Houston, Texas

April, 2014

RICE UNIVERSITY

**Statistical Machine Learning for Text Mining
with Markov Chain Monte Carlo Inference**

by

Anna Drummond

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Christopher M. Jermaine, Chair
Associate Professor of Computer Science

Luay K. Nakhleh
Associate Professor of Computer Science

Swarat Chaudhuri
Assistant Professor of Computer Science

Genevera Allen
Assistant Professor of Statistics

Houston, Texas

April, 2014

ABSTRACT

Statistical Machine Learning for Text Mining
with Markov Chain Monte Carlo Inference

by

Anna Drummond

This work concentrates on mining textual data. In particular, I apply Statistical Machine Learning to document clustering, predictive modeling, and document classification tasks undertaken in three different application domains. I have designed novel statistical Bayesian models for each application domain, as well as derived Markov Chain Monte Carlo (MCMC) algorithms for the model inference.

First, I investigate the usefulness of using topic models, such as the popular Latent Dirichlet Allocation (LDA) and its extensions, as a pre-processing feature selection step for unsupervised document clustering. Documents are clustered using the proportion of the various topics that are present in each document; the topic proportion vectors are then used as an input to an unsupervised clustering algorithm. I analyze two approaches to topic model design utilized in the pre-processing step: (1) A traditional topic model, such as LDA (2) A novel topic model integrating a discrete mixture to simultaneously learn the clustering structure and the topic model that is conducive to the learned structure. I propose two variants of the second approach, one of which is experimentally found to be the best option. Given that clustering is one of the most common data mining tasks, it seems like an obvious application for topic modeling.

Second, I focus on automatically evaluating the quality of programming assignments produced by students in a Massive Open Online Course (MOOC), specifically an interactive game programming course, where automated test-based grading is not applicable due to the character of the assignments (i.e., interactive computer games). Automatically evaluating interactive computer games is not easy because such programs lack any sort of well-defined logical specification, so it is difficult to devise a testing platform that can play a student-coded game to determine whether it is correct. I propose a stochastic model that given a set of user-defined metrics and graded example programs, can learn, without running the programs and without a grading rubric, to assign scores that are predictive of what a human (i.e., peer-grader) would give to ungraded assignments.

The main goal of the third problem I consider is email/document classification. I concentrate on incorporating the information about senders/receivers/authors of a document to solve a supervised classification problem. I propose a novel vectorized representation for people associated with a document. People are placed in the latent space of a chosen dimensionality and have a set of weights specific to the roles they can play (e.g., in the email case, the categories would be TO, FROM, CC, and BCC). The latent space positions together with the weights are used to map a set of people to a vector by taking a weighted average. In particular, a multi-labeled email classification problem is considered, where an email can be relevant to all/some/none of the desired categories. I develop three stochastic models that can be used to learn to predict multiple labels, taking into account correlations.

Contents

Abstract	ii
List of Illustrations	viii
List of Tables	x
1 Introduction	1
1.1 Topic Models for Feature Selection in Document Clustering	5
1.1.1 Problem Definition	6
1.1.2 Statistical Models	7
1.1.3 Related Work	10
1.1.4 Experimental Study	11
1.2 Learning to Evaluate Student Programming Assignments in a Massive Open Online Course	14
1.2.1 Problem Definition	14
1.2.2 Statistical Model	16
1.2.3 Related Work	17
1.2.4 Experimental Study	18
1.3 Senders, Receivers and Authors in Document Classification	20
1.3.1 Problem Definition	20
1.3.2 Statistical Models	22
1.3.3 Related Work	24
1.3.4 Experimental Study	24
2 Literature Review	27
2.1 Topic Models for Feature Selection in Document Clustering	27

2.2	Learning to Evaluate Student Programming Assignments in a Massive Open Online Course	32
2.3	Senders, Receivers and Authors in Document Classification	37

3	Topic Models for Feature Selection in Document Clus- tering	46
3.1	Problem Definition	46
3.2	Mixture-of-Priors Variant	49
3.2.1	The Model	49
3.2.2	Inference	52
3.3	Directional Topic MM	54
3.3.1	Motivation	55
3.3.2	The Model	57
3.3.3	Inference	60
3.4	Experimental Study	63
3.4.1	Methodology	64
3.4.2	Results	67

4	Learning to Evaluate Student Programming Assignments in a Massive Open Online Course	76
4.1	Problem Definition	76
4.2	Distance Measures	81
4.2.1	Programs and Event Handlers	81
4.2.2	Distance Measures	82
4.2.3	Example	86
4.3	A kNN Regression Model	88
4.3.1	The Lift-Then-Compute Distance	89
4.3.2	The Compute-Then-Lift Distance	90

4.4	A Prototype-Based Model	91
4.4.1	Overview	91
4.4.2	Generative Process in Detail	91
4.5	Some Additional Wrinkles	93
4.5.1	Weighting the Distance Measures	93
4.5.2	Weighting the Prototype Fragments	93
4.5.3	A Background Prototype Fragment	94
4.5.4	Final PDF	95
4.6	Learning the Prototype-Based Model	96
4.6.1	MCMC Basics	96
4.6.2	Updating the Indicators	97
4.6.3	Updating the Scores	98
4.6.4	Updating Other Parameters	100
4.7	Experimental Study	103
4.7.1	Experimental Setup	103
4.7.2	Results	106
4.7.3	Discussion	107

5 Senders, Receivers and Authors in Document Classification 111

5.1	Problem Definition	111
5.1.1	The Motivation	113
5.2	The Novel Vectorized Representation of Senders, Receivers and Authors	116
5.3	Built-On Model	121
5.3.1	The Generative Process	121
5.3.2	Inference	123
5.4	Hybrid Bayesian Lasso	126
5.4.1	Generative Process	126

5.4.2	Inference	128
5.5	The Social Network LDA Model.	129
5.5.1	The Generative Process	130
5.5.2	Inference	134
5.6	Experiments	137
5.6.1	Experimental Setup	137
5.6.2	Results	139
5.6.3	Discussion	144
6	Conclusions	146
	Bibliography	148

Illustrations

3.1	Plate diagram for the MoP model.	52
3.2	Plate diagram for the DTMM model	60
3.3	Two point correlation functions for the three topic models.	72
4.1	Screen shot of the Asteroids game implemented as part of the interactive programming MOOC.	77
4.2	Program # 57239, Project <i>Asteroids</i>	83
4.3	Handler for key press (<code>keydown</code>) in Program # 56080, project <i>Asteroids</i>	87
4.4	Handlers for key press (<code>keydown_handler</code>) in Program # 56601 in project <i>Asteroids</i>	88
4.5	Relative importance of the various distance measures, as learned by the prototype-based model. For each project, the bars represent the weight of d_1 through d_6	107
4.6	Mean absolute error of the predictions made by the various methods as a function of the cutoff. For example, if a particular method has a MAE of 1.5 at a cutoff of 6.0, this means that the method achieved an MAE of 1.5 for all submitted programs where the true average score was 6.0.	108

4.7	AUC ROC for the predictions as a function of the “good program”/“bad program” cutoff. For example, is a particular method has a AUC ROC of 0.95 at a cutoff of 6.0, this means that it achieved an AUC ROC of 0.95 when differentiating between programs which received a 6.0 or less, and those that received greater than a 6.0. . . .	109
5.1	An example of a phishing email.	114
5.2	Representation of an email form Figure 5.1 and regression coefficient vector	115
5.3	Latent space positions κ for people associated with an email above (on the left); Vectorized representations α for four categories of people (on the right)	117
5.4	A graphical model representation of Social Network LDA. The dotted arrows represent the deterministic relationship between variables. . .	133

Tables

3.1	Accuracy (as a %) obtained via the various clustering methods	69
3.2	Bootstrapped probability (as a %) that the method associated with the row would outperform on a randomly-selected data set.	70
3.3	Average appearance probability for the top five topics in each of the clusters for the 5 Newsgroups data set.	73
3.4	Most frequent words in the most important topic for each of the learned clusters in the 5 Newsgroups data set. Words that are seemingly irrelevant are given in bold	75
4.1	Details of test data used.	104
5.1	AUC ROC for Bayesian Lasso on two subsets of Technology dataset; each row corresponds to a single label with the last row showing the average AUC across all the labels (in bold)	116
5.2	Details of test data used.	139
5.3	AUC ROC for Ridge Regression based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in bold)	140
5.4	AUC ROC for the SVM based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in bold)	141

5.5	AUC ROC for Bayesian Lasso-based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in bold)	142
5.6	AUC ROC for topic-based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in bold)	143

Chapter 1

Introduction

With the increased amount of electronically stored data, such as email, digital libraries and the World Wide Web, the need for efficient and effective information management has become critical. Therefore, designing intelligent computerized algorithms to mine information from data has become very popular in recent research in computer and information sciences (e.g., [1–5]).

Text in particular is a very common media for the formal exchange of information, and thus vast amounts of stored data are in the form of natural language text, which, compared with numerical data, is unstructured and more difficult to deal with algorithmically. My work lies in the area of text mining, the purpose of which is to process the unstructured natural language text to discover useful patterns, structures and other meaningful information. Text mining has received a lot of attention in the research community and has found its applications in many areas of business and government activities, e.g., social media monitoring [6], automatic classification of documents [7], recommendation systems [8], summarizing documents [9], crawling web sites [10]. Unsupervised document clustering, automated grading of programming assignments based on their source code, and multi-label email classification are the focus of my work.

Many machine learning based methods can be used to derive high quality information from text. In particular, my work utilizes *probabilistic modeling*, which defines a model for an analyzed problem with the use of random variables, and describes the

relationships between them. Before the natural language text can be passed as an input to machine learning algorithms, it has to be properly structured. This usually includes parsing and indexing the words to form a feature vector representation for each document. This approach, also applied in my work, is referred to as “bag-of-words” [11] and constitutes a traditional model to represent textual data. The basic text structuring process can be further refined by combining different grammatical forms of the same word (e.g., by stemming [12] or lemmatizing [13]) and often by removing certain common “stop words”. In my work I remove stop words and lemmatize text in the pre-processing stage.

All the probabilistic models which I discuss in this work are *Bayesian* [14], i.e., expressed by Bayesian (or evidential) probabilities, which together with frequency (or physical) probabilities represent two broad categories of probability interpretations. Bayesian models, unlike the other statistical techniques, incorporate for any unknown parameters the formulation of *prior distributions*, which express one’s uncertainty or belief about those parameter values. The prior distribution in combination with the likelihood function yields a *posterior distribution*, which is the probability of the model parameters given the evidence. Unlike the frequentist approach in which parameter values are considered fixed and are determined by some estimator (e.g., Maximum Likelihood [15]), the Bayesian approach offers posterior probability distribution over the parameters.

Furthermore, the designed models are *generative*, based on a *stochastic generative process*, which describes how all the observed and hidden variables were generated (in contrast to discriminative models [16] that provide a model only for the target variables conditional on observed data). Given the generative process as well as the visible data, the goal is then to infer the posterior distribution, that is, to reverse the

generative process and probabilistically “guess” how the data was actually generated.

For most probabilistic models, exact inference is intractable, and so some form of approximation has to be used (in the Bayesian framework there is a need to integrate over the model, and continuous variables requiring integration often do not have closed form; that computation cannot be performed exactly). Different approximation techniques, used for evaluating the posterior distribution of the latent variables, can be grouped into deterministic and *stochastic*. Deterministic methods are based on analytical approximation to the posterior probability distribution of the unobserved variables and provide a locally-optimal, exact solution to an approximation of the posterior (e.g., Variational Inference [17], Expectation Propagation [18]). Stochastic techniques, on the other hand, can generate exact results if given infinite computational resources; the approximation to the exact posterior arises from the use of a finite computational resource, i.e., processor time. In my work I utilize the latter approach. In particular I chose *Markov Chain Monte Carlo* (MCMC) methods [19], as basic sampling techniques suffer from severe limitations in space of high dimensionality. MCMC is a random walk on a graph; the idea is to simulate a Markov chain, and thus in MCMC the probability of jumping from one vertex to another depends only on the current vertex and not on the entire history. The MCMC algorithm which I use in my work is Gibbs sampler [20]. Briefly, applying a Gibbs sampler requires repeatedly re-sampling the value of each unseen variable from its posterior distribution, conditioned upon the current values of all of the other variables.

This thesis is comprised of three main projects. The presented projects are as follows:

1. Topic Models for Feature Selection in Document Clustering

The study investigates the usefulness of using topic models, such as the popular

Latent Dirichlet Allocation (LDA) [21] and its extensions, as a pre-processing feature selection step for unsupervised document clustering. The topic mixing proportion vectors obtained from various topic models are used as an input to an unsupervised clustering algorithm.

The work has been published as:

A. Drummond, Z. Vagena, and C. Jermaine, “Topic models for feature selection in document clustering,” in *SIAM International Conference on Data Mining*, 2013.

2. Learning to Evaluate Student Programming Assignments in a Massive Open Online Course (MOOC)

The work focuses on grading automatically the programming assignments produced by students in a Massive Open Online Course (MOOC) that covers interactive game programming course. A novel stochastic model is proposed, which based on the source code of some already graded programs. Applying “user-defined” metrics, the model learns to assign scores predictive of a grade that a human (peer-grader) would give to ungraded programs; the model does not require running the programs and does not rely on a grading rubric.

The work has been submitted for publication (under review):

A. Drummond, Y. Lu, S. Chaudhuri, C. Jermaine, S. Rixner, and J. Warren, “Learning to grade student programs in a Massive Open Online Course,” in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2014.

3. Senders, Receivers and Authors in Document Classification

The project concentrates on incorporating the information about senders/ receivers/authors of a document to solve a supervised document classification

problem. I propose a novel vectorized representation for the people associated with a document. In this solution, people are placed in a latent space of a chosen dimensionality and have a set of weights specific to the roles they can take (e.g., in the email case the categories would be TO, FROM, CC, and BCC). The latent space positions together with the weights are used to map a set of people to a vector by taking a weighted average. In particular, I consider a multi-labeled email classification problem, where an email can be relevant to all/some/none of the desired categories. I develop three stochastic models that applying the novel vectorized representation of authors can learn single or multiple labels with correlations.

The work is to be submitted for publication in *IEEE International Conference on Data Mining*, 2014.

The succeeding sections present the overview of the undertaken work organized in those three projects, For each project, I discuss the problem definition, I describe the proposed statistical model, I outline the related work, and give an overview of an experimental study.

1.1 Topic Models for Feature Selection in Document Clustering

This work investigates the use of topic models [22] as a possible future selection step for unsupervised document clustering. The next sections introduce this part of my research by describing the following: the problem definition, the proposed statistical models, the related work, the experimental study with results.

1.1.1 Problem Definition

Topic models [22] such as Latent Dirichlet Allocation (LDA) [21] and Probabilistic Latent Semantic Indexing [23] are statistical models most often used for analyzing document corpora. A *topic* in a topic model is a set of words that tend to appear together in a corpus, and the documents in the corpus are viewed as being produced by some mixture of topics. There are many reasons to build a topic model for a corpus. The modeling process typically results in each document being represented as a vector of topic proportions, where each entry of this vector describes the importance of a particular topic in the document (I subsequently refer to this vector as a document’s “topic mixing proportion” vector). Thus, topic modeling is useful as a pre-processing step for many mining and information retrieval tasks. One can first build a topic model, and then use this vector as a list of features describing the document, in addition to (or in lieu of) other features that might be collected. The resulting feature vector can then be used along with a favorite algorithm or model to perform clustering, classification, outlier detection, similarity search, etc.

My research examines the utility of topic modeling as a pre-processing step for performing unsupervised clustering of the documents in a corpus. Given that clustering is one of the most common data mining tasks, it seems like an obvious application for topic modeling.

In this work I consider the utility of the following process:

1. First, build a topic model for the corpus.
2. Next, use the model to obtain for each document a topic mixing proportion vector θ that represents the document’s location in “topic space”. In most topic models, the entries sum to one and the i th value is the prevalence of topic i in the

document.

3. Finally, use all of the θ vectors as input to an unsupervised clustering algorithm.

My work investigates different topic models that can be used in Step 1 of the process presented above. Rather than proposing and advocating a particular technique to topic modeling for document clustering, I instead seek to develop and rigorously evaluate a few alternatives that represent two approaches: (1) Learning a traditional topic model for a corpus; here “vanilla” LDA is used (2) Building a novel topic model that integrates a discrete mixture to simultaneously learn the clustering structure and the topic model that is conducive to the learned structure. I propose two variants of the second approach, i.e., Mixture of Priors, and Directional Topic Mixture Model. All three methods are introduced in the following part of this chapter.

1.1.2 Statistical Models

This section introduces three alternatives to topic modeling for document clustering that are considered in this study, i.e., “vanilla” LDA, Mixture of Priors, and Directional Topic Mixture Model.

“Vanilla” LDA

The first alternative I consider is the simplest and most obvious use of “vanilla” LDA [21] to produce the θ vectors that are used as input to the clustering. In LDA, each word in a document is produced by one of T topics; the probability of an arbitrary word in document j being produced by topic t is given by the t th entry in the vector θ_j , which I refer to as the document’s *topic mixing proportion* vector.

LDA seems to be an obvious choice because it is now one of the single most widely used algorithms for processing natural language text. However, I have reason

to expect that “vanilla” LDA is not going to be the very best option for pre-processing text as an input to a clustering algorithm. Classic LDA makes use of a Dirichlet prior on the θ vectors, which govern the mixture of topics that are found in each document. This prior has a parameter α that controls the mean of the various θ vectors, as well as how widely the vectors are dispersed from that mean. α is typically taken to be a symmetric, constant vector (in practice, $\alpha = \langle 0.1, 0.1, \dots, 0.1 \rangle$ is often used). The problem is that the Dirichlet prior is quite well-behaved: it does not encourage a “bumpy” distribution of θ vectors, which is what one would desire as input to a clustering algorithm.

Therefore, I consider two alternatives to “vanilla” LDA that can be used to pre-process a corpus as an input to a clustering algorithm.

Mixture of Priors

Mixture of Priors (MoP), the first alternative to “vanilla” LDA, is a simple modification to the LDA model. Rather than each document sharing the same prior on its topic mixing proportions, in the MoP alternative, a document chooses its prior from a mixture of K different priors. This naturally allows for a “bumpy” prior on the distribution of the θ vectors, and my hypothesis is that this would result in data that are more amenable to clustering.

In the MoP, the topic mixing proportion vector θ associated with document j is sampled from a mixture model of non-symmetric Dirichlet distributions (unlike in LDA where it is sampled from a single symmetric Dirichlet distribution); the identity of the mixture component producing this θ vector is then the cluster identity for document j , and the documents in the same mixture component tend to have similar topic proportion vectors.

Directional Topic Mixture Model

A second alternative that I propose is called the *Directional Topic Mixture Model* (DTMM). Unlike LDA, the DTMM does not rely on a Dirichlet distribution to govern to what extent the various topics influence production of a document. Instead, it relies on a spherical distribution to place documents on a unit sphere; the position of a document on the sphere controls how topics are mixed within the document. More precisely, the topic mixing proportions are produced by a mixture of T -dimensional von Mises Fisher (vMF) distributions [24]. The vMF distribution is a continuous, multivariate distribution over the unit sphere. The inspiration is taken from some existing work on using spherical models for clustering and analysis [25, 26] including a recent spherical take on the task of topic modeling [27].

The vMF distribution takes two parameters: a mean position on the unit sphere μ , and a “concentration” parameter κ that determines the spread of the distribution across the surface of the sphere—larger κ values reduce the spread. To facilitate a clustering of the documents, I use a mixture of vMF distributions to place the j th document in a latent position θ_j on the unit sphere. The identities of the various mixture components used to produce the document positions can be used to segment the documents into clusters.

In order to use the document’s latent position on the sphere to choose the topics that are used to produce the document, the document’s position in topic space is mapped to a position on the $(T - 1)$ -simplex by a transformation similar to the one used by the logistic normal distribution [28] to map the output of a normal random variate in R^d to the simplex.

By utilizing K mixture components in the spherical distribution, I obtain a “bumpy” prior on the topic mixing proportions. The hoped-for benefit of this ap-

proach is that the spherical distribution allows for a document’s cluster membership to have a more direct influence on the topic mixing proportions than in the MoP alternative, in which cluster membership supplies only a prior to a document’s topic mixing proportion.

1.1.3 Related Work

In contrast to the problem of supervised classification using topic models [29–31], there has surprisingly been very little investigation into the utility of topic models for clustering of documents [1, 3, 32–35]. Document clustering based on latent topics has recently been discussed in [2]. In that work a non-probabilistic approach, based on matrix decompositions, was employed to identify the latent topics, which were then used for clustering through k-means. My work is complementary to that, as I investigate the applicability of statistical topic models for document clustering. Also, in [1] a model-then-cluster approach was used for clustering web-blogs and several variations of the scheme were compared. Again my work is complementary to that one as it focuses on the comparison of the different ways topics can be employed for document clustering.

The mixture of priors model is reminiscent to the LDA extension for clustering that was employed in [34] to summarize activities (modeled as distributions over low-level visual features) and interactions (modeled as distributions over activities) in complicated scenes. In that work a non-Bayesian version of the model was used to represent different types of interactions. A similar model has also been employed in [35] to produce better quality, low representations of documents. There, each document is partitioned into segments (e.g. physical paragraphs) and the mixture of Dirichlet distributions is used to define the distribution of topics in each segment. In

my work I discuss a Bayesian treatment of the model and investigate its performance for document clustering.

The first generative model that employed a mixture of vMF distributions for document clustering was discussed in [26]. In that work, a k-means inspired algorithm (which was called spherical k-means) was devised to perform the clustering, based on the derived data likelihood. In [33] an EM solution for the constrained case where each vMF component has the same concentration parameter (i.e. k) was presented and in [25] the authors extended the EM solution for the general case of different concentration parameters. In all the aforementioned approaches the vMF distribution is used to directly generate a normalized *tf* or *tf-idf* vector for each document. In contrast, my work focuses on employing topic models for clustering and uses the vMF distribution to position each document in the (latent) topic space. The spherical topic models [27] are loosely related with the proposed directional topic mixture model in that both models employ the vMF distribution to derive an admixture topical model for documents.

1.1.4 Experimental Study

In this section I present an overview of an experimental study of the three methods. My goal is to discover whether there is any qualitative reason to prefer one of the above discussed methods to the others.

My basic methodology is to use each of the three methods on a number of data sets with known clusterings (CMU Newsgroups *: two subsets of the original data set

*<http://kdd.ics.uci.edu/databases/20newsgroups/>

are used: 5 Newsgroups and 10 Newsgroups, Classic 3[†], Reuters 8[‡], WebKB 4[§], and Stock) (1) to produce input to k-means clustering algorithm (2) to compare how well the resulting clusterings match the expected results.

I tested four different options:

1. Spherical k-means, using cosine distance on *term-frequency* vectors.
2. First learn an MoP model, then run k-means on the resulting model.
3. First learn a DTMM model, then run k-means on the resulting model.
4. First learn an LDA model, then run k-means on the resulting model.

For each option, I performed two experiments; each one was repeated five different times. In the first experiment, exactly the correct number of clusters was used to perform the clustering. In the second experiment, twice the correct number of clusters was used to perform the clustering.

I found that the MoP method was the best of the methods that I tested for clustering text documents. This method was followed by spherical k-means and the DTMM model, which performed similarly when the correct number of clusters was used. I also found that if the correct number of clusters is used, LDA was clearly the poorest choice for document clustering. It did outperform the spherical k-means algorithm on two of the data sets, but clearly outperformed the DTMM model only once and was never close to outperforming the MoP model. Interestingly, LDA performed much better when twice the correct number of clusters were used. In that scenario, LDA

[†]<ftp://ftp.cs.cornell.edu/pub/smart/>

[‡]<http://web.ist.utl.pt/~acardoso/datasets/>

[§]<http://www.cs.cmu.edu/~webkb/>

was competitive with spherical k-means, however, it was still clearly outperformed by the MoP model.

To investigate why the three topic models performed as they did, I carried out further analysis. The first thing that I checked was the quality of the topics themselves. It was clear that a much higher percentage of the top words associated with the LDA-based topic model were words that had little to do with the cluster in question. Second, for each cluster, and for each model, I determined the top five topics (that is, the five topics most closely associated with the cluster). I then computed the average prevalence of those five topics in the documents that belonged to the cluster. As expected, I found that the MoP model tended to produce topic proportion vectors that spread their probability across more topics than LDA did. Third, I computed the two-point correlation function over all of the topic proportion vectors for the three models. The two-point correlation function is essentially the PDF for the distance between two points randomly selected from the data set. In a data set having a good clustering structure, one would expect at least two modes in the correlation function: a small one close to zero (for pairs of points in the same cluster) and a larger mode far from zero (for pairs of points in different clusters). This is exactly what I found in the MoP correlation function and in the DTMM function over that data. The LDA correlation function, however, is not nearly as high quality, showing a lack of clustering structure.

To summarize: the MoP model seems to clearly produce the best clustering structure when used as a pre-processing step for k-means. This would seem to be the preferred approach. The DTMM model can perform well on certain data sets, but it is sometimes outperformed by a simple spherical k-means algorithm. I found that a simple LDA model is possibly not a good option for feature selection for unsupervised

learning.

1.2 Learning to Evaluate Student Programming Assignments in a Massive Open Online Course

This part of my work addresses the problem of automatically evaluating the quality of programming assignments produced by students in a Massive Open Online Course (MOOC). In particular, the focus is on an interactive game programming course, where the assignment outcomes are interactive visual computer games. The project overview is organized in the succeeding sections as follows: the problem definition, the proposed statistical model, the related work, and the experimental study with results.

1.2.1 Problem Definition

In a MOOC providing accurate feedback to students is a significant challenge. When thousands of students can enroll in a single course, there is no possible way for an instructor (even with a team of teaching assistants) to look at the work performed by each student. This problem in MOOC is addressed via automated grading and peer evaluation. Automatically evaluating interactive computer games is not an easy task because such programs lack any sort of well-defined logical specification, so it is difficult to devise a testing platform that can play a student-coded game to determine whether it is correct. Moreover, due to the character of the programming assignments, automated test-based grading is not applicable.

Automated scoring could be used to aide the peer grading process. This could be done in two ways:

1. **As an automatic gating function.** A problem observed in Rice MOOC's experience is that since the "price" of being able to grade other students assignments is turning in one's own assignment, some students will turn in an assignment that they know is incomplete, simply to be able to view and grade other student's assignments. While the motivation for doing this is a bit unclear (perhaps it is curiosity, or perhaps the motivation is to harvest a set of solutions) it is clear that a student should not be able to view others' work without first completing (or at least making a reasonable attempt to complete) the assignment. If one could automatically recognize with high accuracy those submission that are very low quality, it would make possible to automatically bar such students from taking part in the grading.
2. **Allocation of graders.** Peer graders need to be allocated carefully. There is a negative correlation between the quality of a student-produced program and the accuracy of a peer-assigned grade, implying that poor submissions need more graders to receive an accurate grade. There are also qualitative reasons to avoid assigning the same number of peer graders to every assignment. Rather than assigning many graders to give a high-quality submission a perfect score, it would be better to move some of those graders to a poor assignment to help an under-performing student with constructive comments and criticism. Also, to goal is to make sure that the strongest graders grade the poorest assignments (so they can be of help), and for the weakest students to grade the best assignments, so they can see examples of what an excellent submission looks like.

In this work I propose a stochastic model that uses a set of already peer-graded training data consisting of the written source code, to learn to assign scores that a

human (i.e., peer-grader) would give to ungraded assignments. In the predictions of the grades, the model applies “user-defined” distance measures and does not rely on execution of the program nor on a grading rubric. The goal is to utilize the fact that a MOOC is a data-rich environment. After an online course has been run one time, for each assignment, one will have an access to thousands of programs with observed scores.

1.2.2 Statistical Model

I propose a Bayesian model that views each program as a set of *prototype fragments* that can be procedures, blocks, or class definitions. The specific data set that is used here consists of event driven programs written in Python, and thus the natural fragments to compare are *event handlers* that define actions to be undertaken when a user interacts with the game. The proposed model makes use of the *prototype fragments* to predict the score of a test program. At a high level, I define a stochastic, generative process that assumes that a peer grader grades a program by seeing how similar the fragments in the program are to a special set of prototype fragments. Some of the prototype fragments are known to be good, and some are known to be bad. Based upon this goodness (or badness) each prototype fragment is assigned a score. These scores are used to score the individual fragments in the test program: a fragment in the test program is assigned a score that is the weighted average of the scores of the similar prototype fragments (the closest fragments have the highest weight). The score of the program as a whole is then the average of the scores of the individual fragments within it.

The proposed method fundamentally relies upon the definition of a set of measures that can be used to decide whether two program fragments are similar. The set of

measures utilized here combines semantic and syntactic distance measures including the type of code fragment, the bag of library calls, the bag of guarded updates, the abstract syntax tree, and the dependence graph over library functions and variables. I also consider an additional measure which views the source code as a natural language text; each program fragment is tokenized and a topic model is built for a corpus of code fragments. The distance measure reflects the Euclidean distance between the topic mixing proportion vectors of the fragments.

1.2.3 Related Work

Automated grading has received a lot of attention in the research community. Other work [36–41] also apply a static analysis approach, where some representation of the source code is analyzed without relying on executing the program. However, this work significantly differs from mine in two main areas. First, the programming assignments in the related studies are graded deterministically based on given similarity measures, while my work defines a stochastic process to predict grades. Second, in the other work, grades are calculated based on the similarity comparison to a model program or solution usually provided by the instructor. Here, already-graded programs create a training set; the various proposed distance measures are then weighted to minimize the prediction error on the training data to infer the grades for the ungraded programs.

On the statistical side, the proposed fragment-based learning method, where the various metrics are weighted to minimize the prediction error on the training data, most closely resembles the work on learning distance measures for k NN classification [42], and the larger body of work on learning of distance metrics [43, 44].

1.2.4 Experimental Study

To evaluate the proposed method, I used six projects completed by students as they participated in a “Programming 101” MOOC. The goal in each project is to design and implement an interactive video game in Python. I used a randomly selected subset of programs submitted for each assignment to predict the scores of the other programs. The average score given by all peer graders is taken as the ground truth.

The seven different options for predicting the ground truth score were as follows:

1. The proposed prototype-based model. The prior used on the s_k values was a $\text{Normal}(\mu, 0.5)$ distribution, where μ was the average score over the training data. Prototype fragments of 100 randomly chosen programs were used. Since the learning algorithm converges quickly, we used only 20 complete Gibbs sampling cycles to learn the model. After the 20th update cycle, the current model was used to perform all predictions.
2. k NN with Lift-Then-Compute distance. A small portion of the training set was used to determine an appropriate value for k . The median score out of the k closest matches is used as the prediction.
3. k NN with Compute-Then-Lift distance. Except for the distance measure, this is identical to the above.
4. An ensemble method where a small portion of the training set is held back, and then each of the above models are learned on the remainder of the training data. Then, an l_2 -regularized regression is used to weight the above three models in such a way that the error of the weighted prediction on the held back training data is minimized. An additional intercept term is used in the learning to unbiased

the model. The three models are then used together as an ensemble to perform the prediction.

5. An almost identical ensemble method, where the only difference is that an intercept term is not used.
6. The simple method of giving every program the average score. This is an illustrative strawman since the project scores are generally high with low variance, so it will achieve low error.
7. A “random human”. One of the humans who graded a program was chosen at random, and his/her score was used to perform the prediction. Although there is nothing preventing a statistical method from doing better than a random participant in the class, it is difficult to imagine this happening. Thus, the “random human” should be seen as something of an upper bound on the performance of any statistical method.

Based on the experiments, a few findings stand out when the mean absolute error is considered. First, the Compute-Then-Lift k NN-based regression model and the prototype-based model are (in my opinion) startlingly accurate, considering that a simple, statistical approach is used to recognizing poor programs. The results are particularly striking considering the worst programs.

Another interesting finding is that the two ensemble methods seem to do a relatively poor job compared to the prototype-based model and k NN with the Compute-Then-Lift distance. The reason for this is that for several of the programs, Lift-Then-Compute seems to be quite biased; the ensemble-based methods have a difficult time taking this into account, and suffer accordingly.

But a different picture emerges when examining the AUC ROC values. Here, bias in predicting the final score is not a problem when performing a classification task, because one can simply adjust the “good program”/“bad program” threshold to obtain more or less positive cases as desired, and correct for the bias. This is exactly the trade-off that the ROC curve explores. In this case, the Lift-Then-Compute distance does much better. Not only that, but the two ensemble-based methods that take into account all three models are clearly the best choice in terms of AUC, with the ensemble-with-intercept being the most universally applicable. Which of the three non-ensemble methods was best depended upon the program, but the prototype-based statistical model had the best performance of the three in more situations.

None of the automated methods approaches the accuracy of the random human for the higher-quality submissions, but human-like accuracy is not needed for performing tasks such as filtering out submissions from students who did not put in a serious effort on the assignment.

1.3 Senders, Receivers and Authors in Document Classification

The main focus of this project is using the information about senders, receivers or authors of a document in document classification. The next section defines the problem followed by the overview the proposed three statistical models and the related work. The experiments and the results are discussed in Section 1.3.4.

1.3.1 Problem Definition

With the increased number of electronically stored information (ESI), in 2006, amendments to the US Federal Rules of Civil Procedure, introduced the requirement to

provide ESI in the discovery phase of litigation, i.e, the initial phase where the parties in a dispute are required to provide each other with relevant information and records [45]. These changes brought the term *eDiscovery*, Electronic Discovery, to common use. eDiscovery relates to all electronic information including word documents, spreadsheets, emails, audio and video. eDiscovery has now become central to litigation, particularly when involving review of terabytes of digital data. The use of automation to manipulate ESI during any stage of eDiscovery is referred to as *Predictive coding*. The technology is also known as automated document review, automated document classification, automatic categorization, predictive categorization, or predictive ranking.

Predictive coding starts with a set of data, derived or grouped in a variety of ways, e.g., by keyword or concept searching. A sample of the data is manually reviewed and labeled by experts to create a training set. A machine learning software is then employed to categorize similar documents in the larger data set in terms of responsiveness, privilege and/or issue-relation based on the labeled examples.

My research considers how to use the information about senders/receivers/authors of a document in addition to its text content to solve a supervised document classification problem. In case of multiple authors, the simple approach of mapping people to the dimensions (e.g., binary representation) does not perform well in classification models. Hence, I propose a new method to map a set of people associated with a document to a vector. In my solution, people are placed in a latent space of a chosen dimensionality and have a set of weights specific to the roles they can play (e.g., in the email scenario the categories would be TO, FROM, CC, and BCC). The latent space positions together with the weights are used to create the vectorized representation by taking the weighted average. I use the vectorized representation of authors together

with the text content of a document in three stochastic models that I developed to classify documents.

In particular, I consider an email classification problem, as emails including attachments constitute around 75% of ESI in litigation. Furthermore, the considered problem has a multi-label character, where an email can be relevant to all/some/none of the desired categories.

The models I propose can also be applied to several other applications, including filtering messages based on their priority, assigning messages to user-created folders, or to email routing (e.g. routing email for a help desk, so that the email reaches the correct person/people). Furthermore, the models can be used to classify any documents which have some author properties.

1.3.2 Statistical Models

This section presents an overview of the three developed document classification models that apply the proposed vectorized representation for people associated with a document.

BuiltOn Model

The main idea behind *BuiltOn* model is to take a standard classifier and add to it in a meaningful way. It is done by concatenating the output of the classifier on the text content to the novel vectorized representation of people. A Bayesian Lasso model is then trained on so concatenated vectors. In this way, a new regression is performed on the residuals which are left over from the text-based classifier in order to boost the classification accuracy. In my work, I used ridge regression and SVM as the base classifiers I wanted to improve on.

Hybrid BL Model

The Hybrid Bayesian Lasso model is a full Bayesian treatment of the *BuiltOn* model. It has an iterative character where two Bayesian Lasso models run interchangeably: (1) Bayesian Lasso on the TF representation of the text of the documents (2) Bayesian Lasso on the novel vectorized representation of people in the social network.

Social Network LDA Model

The existing author-topic model [46] considers a scenario where the representation of a document's content is enriched by including the information of its authors. The model uses the topic-based representation to model both the content of the document and their authors. Topic models [22] such as LDA [21] and PLSI [23] are statistical models most often used for analyzing document corpora. A *topic* in a topic model is a set of words that tend to appear together in a corpus, and the documents in the corpus are viewed as being produced by some mixture of topics. In LDA, each word in a document is produced by one of T topics; the probability of an arbitrary word in document j being produced by topic t is given by the t th entry in the vector θ_j , which we refer to as the document's *topic mixing proportion* vector.

In the author-topic model, for each word in the document an author is chosen and then a topic is chosen from a distribution of topics specific to that author; the word is generated from the chosen topic.

In the *Social Network LDA*, I also use topic models to represent the corpus. The main difference is that in the presented model, the information about the authors are used to generate a prior for the topic mixing proportion vector of a document (email body). This is motivated by our previous study [47] which showed that a meaningful prior on the document topic mixing proportion vectors can yield a better

topic modeling structure. Furthermore, the author-topic model uses the topic-based approach only to characterize emails (topic-based content representation); I consider the problem of document classification where the content representation is by default its by-product.

1.3.3 Related Work

Email classification, in particular, has received a lot of attention in the research community [4]. The majority of work, however, addresses the single-label classification issue, either binary classification (e.g. spam filtering [48,49]) or multi-class one (e.g., email foldering [50,51]). The multi-label email classification problem, on the other hand, is tackled in the literature using problem transformation methods [52], which transform the multi-label classification problem into multiple, independent binary problems; this approach significantly differs from mine. The algorithm adaptation techniques [52], relevant to my study, are the subject of limited work. The related examples [53,54], are in the general text classification field, and as a result they naturally use only the body of documents to make the classification predictions.

1.3.4 Experimental Study

In the experimental study I used three real data sets of emails coming from eDiscovery process of actual corporations in finance, construction and technology industries. I evaluated the application of the vectorized representation of people by incorporating it in the three statistical model that I developed and testing its performance in comparison to other modeling options. In total, I learned and evaluated thirteen classification/regression models:

1. Ridge regression using the TF representation only of the text content

2. Ridge regression extending the TF vector from the above model by concatenating a binary representation of people for each category
3. BuiltOn model taking the output from the ridge regression text-based method (1) and concatenating the novel vectorized presentation of people
4. Another version of BuiltOn ridge regression (3) with added correlation between labels
5. SVM using the TF representation only of the text context
6. SVM extending the TF vector from model (5) by concatenating a binary representation of people for each category
7. BuiltOn model taking the output from the SVM text-based method (5) and concatenating the novel vectorized presentation of people
8. Another version of BuiltOn SVM (7) with added correlation between labels
9. Bayesian Lasso using the TF representation only of the text content
10. Bayesian Lasso extending the TF vector from model (9) by concatenating a binary representation of people for each category
- item Hybrid model which iterates between Bayesian Lasso performed on text and Bayesian Lasso on the novel vectorized representation of people
11. Topic-based classification model [29] where the text dictionary is extended by people for each category
12. An improved variation of the topic-based model (12), where the novel vectorized representation of people is used as a prior for topic mixing proportion vectors of documents represented by a standard text-based dictionary.

Based on the experiments and using the AUC ROC to present the results, a few findings stand out. First, across all three data sets the *BuiltOn* model has better performance than both of the standard classification methods that I tried to improve on, i.e., ridge regression and SVM. Moreover, including correlation in the *BuiltOn* model improves the results even further.

Second, across all the datasets, extending the TF text-representing vector by binary representation of people did not improve the classification power of tested ridge regression and SVM models.

Hybrid BL method shows to be dataset dependent. In different cases it performed better, the same or worse when compared to a *BuiltOn* model. The topic-based *Social Network LDA* model outperformed for each dataset a simple topic-based classification model which uses the text dictionary extended with senders and receivers. What is also important to notice is that in all the cases *Social Network LDA* did better than SVM classifier using text and people information. *Social Network LDA*'s performance is the closest to the one of *Hybrid BL*.

Chapter 2

Literature Review

This chapter covers the related work. The discussion is divided into separate subsections upon the relevance to the three projects, i.e., (1) Topic Models for Feature Selection in Document Clustering, (2) Learning to Evaluate Student Programming Assignments in a Massive Open Online Course, and (3) Senders, Receivers and Authors in Document Classification.

2.1 Topic Models for Feature Selection in Document Clustering

This work investigates the use of topic models [22] as a possible future selection step for document clustering. The full vector space model (VSM) [55] representing text documents, in which each dimension corresponds to a separate term, can be high dimensional due to a great number of words in a vocabulary. Using this document representation in clustering may make this task computationally very expensive. Furthermore, many of these dimensions may have little or no discriminatory value. Because of the above, it is necessary to reduce the feature space. Several possibilities of dimensionality reduction techniques exist to accomplish this task, including random projection, principal component analysis, or discrete cosine transform. The information retrieval community proposed Latent Semantic Indexing [56] based on the Singular Value Decomposition (SVD), followed by its probabilistic alternative:

probabilistic LSI (PLSI) [23]. PLSI introduced the concept of latent “topics” viewed as mixture components of a model. In the PLSI model, each document consists of multiple topics and each word is modeled as a sample from one of those. In this way, dimensionality of each document is reduced to a probability distribution of a constant number of topics. However, PLSI is based on the discriminative model, which leads to a problem of representing the documents from outside the training set. To overcome this issue, Blei et al. [21] came up with Latent Dirichlet Allocation (LDA), which is a generative probabilistic model of a corpus of documents. Since then, many modifications of LDA have been proposed, e.g., Hierarchical Topic Models [57], Dynamic Topic Models [58], Correlated Topic Models, [59], or Supervised Topic Models [29]. In this work I use the original LDA (referred to as “vanilla” LDA) together with two novel topic models developed as a part of this work, i.e., a mixture of priors, and the directional topic mixture model (relying on a spherical distribution on a unit sphere), to evaluate the usefulness of topic models in feature selection for a document clustering task. In particular, I investigate two approaches of using topic models in clustering the documents: (1) Learning a traditional topic model for a corpus as a first step and then applying a standard clustering algorithm, such as k-means, to cluster the documents in topic space (referred to as “model-then-cluster”) (2) Integrating a discrete mixture into the topic model to simultaneously learn the clustering structure and the topic model that is conducive to the learned structure, and then applying a standard clustering algorithm (referred to as “model-and-cluster”). The performance of “vanilla” LDA is analyzed a part of the first approach, while the two novel variants of LDA fall into the second category. The remaining part of this section of Literature Review is organized in relation to the presented three models. The section discusses the work related to model-then-cluster approach with the focus on application of tra-

ditional LDA, followed by work related to the two models of the model-and-cluster group, i.e., mixture of priors and directional topic mixture model.

Traditional LDA in a model-then-cluster approach is used in [1] to cluster web-blogs and, as the final goal, to produce the cluster description in terms of words. The authors compare different standard clustering algorithms (k-means, k-means++, affinity propagation, and Markov cluster algorithm (MCL)) used on the outcome of LDA. They find MCL performing better in a cluster description task than other algorithms. I, on the other hand, instead of comparing different clustering algorithms, investigate different ways of topic modeling that would produce a better input to those algorithms, which would then lead to higher quality clustering overall. Document clustering based on latent topics has also been recently discussed in [2]. In that work a non-probabilistic approach, based on matrix decompositions, is employed to identify the latent topics, which are then used for clustering through k-means. My work is complementary to that, as it focuses on the comparison of the different ways topics can be employed for document clustering. A model-then-cluster approach with traditional LDA used as a feature selection step for clustering was also investigated in [60]. The study, however, concentrates on visualization aspect of clustering, and thus the clustering algorithm applied on the outcome of LDA is a Self Organizing Map (SOM) [61]. The k-means algorithm is used there to cluster the prototypes of the learned SOM to improve the visualization effect.

A significantly different approach to clustering by using LDA-based models, where the topics themselves are treated as clusters and the cluster assignment is established based on the maximum likelihood of the observed data with respect to the model parameters, has been presented in many research papers. Ramage et al. [3] apply a modified LDA model as well as k-means algorithm on VSM to investigate the use of

user-generated tags from large scale bookmarking websites for improving clustering of web pages. Banerjee et al. [32] propose an online variant of Bayesian LDA for clustering text streams and compare it to other batch and online topic models. A modified LDA model has also been employed in [35] to improve performance of document clustering by producing better quality, low representations of documents. There, each document is partitioned into segments (e.g., physical paragraphs) and each topic is a distribution of those segments. Viewing topics as clusters is significantly different to the representation of clusters in my work, where the clusters are learned based on the topic mixing proportions of the topic models. Furthermore, in two presented variants of the model-and-cluster approach, the discrete mixture of clusters is integrated into the topic model itself.

The authors of [3], already mentioned above, answer similar research questions to those of my work: (1) Can one do better than traditional LDA by creating a modified version of it? (2) Do LDA-based models perform better than standard clustering algorithms based on VSM? Although the main focus of that work differs from mine, the reported overall results are very similar. Both studies show that modified LDA outperformed traditional LDA, which on the other hand was better than k-means on VSM. In my work, however, I go a step further by investigating why certain models perform better than other.

The first of the novel models presented in my work, mixture of priors model (MoP), is similar to the hierarchical LDA extension for clustering that is employed in [34]. The authors apply the model to understand activities (modeled as distributions over low-level visual features) and interactions (modeled as distributions over activities) in complicated scenes from visual data. In that work, however, a non-Bayesian version of the model is used to represent different types of interactions, while my MoP is fully

Bayesian. As far as I am aware, this is the only model presented in the literature that is comparable to the MoP.

The directional topic mixture model (DTMM), the other novel model introduced in this study, includes von Mises-Fisher distribution (vMF) components. The first generative model that employed a mixture of vMF distributions for document clustering was discussed in [26]. In that work, a k-means inspired algorithm, called spherical k-means, was devised to perform the clustering based on the derived data likelihood. Authors of [33] introduce an Expectation Maximization (EM) solution for the constrained case where each vMF component has the same concentration parameter (i.e. k). Banerjee et al. [25] extends the EM solution to the general case of different values of the concentration parameters. A mixture of vMF is also used in [32] for clustering over text streams in a batch, online and hybrid version. However, in all the aforementioned approaches the vMF distribution is used to directly generate a normalized *tf* or *tf-idf* vector for each document. In contrast, my work focuses on employing topic models for clustering and uses the vMF distribution to position each document on a unit sphere in the latent topic space. The position of a document on the sphere controls how topics are mixed within the document. Using vMF to model the topic mixing proportions is an alternative to a Dirichlet distribution used in my MoP and other existing LDA-extensions.

Another model related to my DTMM is the Spherical Admixture Model introduced in [27]. Both models employ the vMF distribution to derive an admixture topical model for documents. However, the role of the vMF distribution is substantially different in the two models. In the other model a topic is defined as an arbitrary vector on the unit hypersphere S^{M-1} and the vMF is used to (a) generate the topics and (b) generate the L_2 normalized vector for each document. In contrast, in my

DTMM, the vMF is employed to find the position of each document in the latent topic space. Thus, the modeling aspects of the models are different: the former can represent both presence and absence of words in a topic while mine focuses on the within-cluster correlation of topics.

All the vMF-related models discussed above apply Expectation-Maximization (EM), a maximum likelihood approach for inference, to infer the model parameters. In contrast, inference in my work is done by Markov Chain Monte Carlo, in particular Gibbs sampler algorithm.

2.2 Learning to Evaluate Student Programming Assignments in a Massive Open Online Course

This part of my work concentrates on automatically evaluating the quality of programming assignments produced by students in a Massive Open Online Course (“MOOC”). In particular, the focus is on an interactive game programming course, where the assignment outcomes are interactive visual computer games. This work discusses a stochastic model that uses a set of already peer-graded training data consisting of the written code, to learn to assign scores that a human (i.e., peer-grader) would give to ungraded assignments. In my model each program is viewed as a set of program fragments; semantic and syntactic distance measures are used to compare two program fragments to determine how similar they are. The comparison outcome is then used to establish the similarity of two student-submitted programs. Besides learning the scores themselves, the motivation of my work is to optimize the allocation of peer graders based on the quality of the assignment work. On the statistical side, the fragment-based learning method presented in this thesis, where the various metrics

are weighted to minimize the prediction error on the training data, most closely resembles the work on learning distance measures for k NN classification [42], and the larger body of work on learning of distance metrics [43, 44].

In terms of online education, there are two standard approaches to grading in large-scale online courses: peer evaluation and automated testing. My techniques are complementary to peer grading, as they can be used for more efficient allocation of human graders.

Automated programming assessment is conducted by variety of methods that can be grouped into two major approaches: dynamic analysis and static analysis. Dynamic analysis tests a program by executing it on provided test cases which are either manually or automatically generated. Thus, the correctness of a program under such an assessment is determined by comparing its output with that of the model program (usually generated by the instructor). This can be done, for instance, by using simple string matching, as in Online Judge [62], or pattern matching, as in ASSYST [63] which uses Unix Lex (Lexical Analyzer Generator) and Yacc (Yet Another Compiler-Compiler) to check if the student's output conforms with the tutor's specifications. CourseMaker [64], an improved version of Ceilidh, one of the courseware systems providing for the computer based assessment coursework, as well as HoGG, the Homework Generation and Grading project [65], check the output by matching it to the regular expression of the desired outcome format. Apart from correctness, under the dynamic assessment scheme, the executing efficiency is also tested. Online Judge [62] determines whether the program is able to produce its output within the time limit and memory resources supplied, while ASSYST [63] additionally applies the metric of the statement execution count. The main shortcoming of dynamic assessment is its lack of applicability to a programming assignment that does not compile and run

to produce an output. The goals and mechanics of the dynamic assessment methods are very different from those in my work, as they relate to assessing data generated by executing a program, and do not involve comparison of code fragments.

Static analysis, on the other hand, does not rely on executing programs. Instead, it analyzes some representation of their source code which can vary from textual form to a graph representation. Here, the correctness of the student’s program is usually tested by similarity-based comparison to the desired program. The methods employed include methods based on character comparison, syntax analysis and semantic analysis. The first set of methods is applied in [36], automated C programming assessment, where the program source codes are converted into pseudo-codes in order to map the source code into a more uniform text. In that study, variable declarations are compared based on the number of variables in each data type category and the similarity of the created pseudo-codes is determined by string comparison.

Methods based on syntax analysis can use Abstract Syntax Trees (ASTs) generated from the source codes to perform structure similarity analysis. In Environment for Learning Programming (ELP) [37], an online interactive and constructive environment for learning to program, the AST of a program is represented using XML. The main weakness of the system is that the comparison process produces only “yes” (for matching structure) or “no” (for un-matching structure), thus it does not identify to which degree the student program matches the model solution. Furthermore, the assessment is carried out on a “fill in the gap” type of an exercise, and therefore only the gap code supplied by the student is analyzed. ASTs generated from C parser are also used in [38] to analyze and visualize differences in software metrics such as control metrics, call graph metrics, and data structure metrics. A novel graph similarity measure, AssignSim, to quantify the similarity between programs is introduced

in [39].

The semantic level analysis can be performed on the System Dependence Graph (SDG), Control Dependence Graph (CDG) or Control Flow Graph (CFG), Method Call Dependencies (MCD), and Program Dependence Graph (PDG). SDG incorporates control dependence and data dependence into a single structure. Standardized SDG, an outcome of semantic-preserving transformation performed on SDG, are used in [40], an on-line examination system for the programming language C. eGrader [41], a graph based grading system for Java, conducts its assessment on CDG and MCD, which are constructed from the AST of the source code. PDGs are applied in AnalyseC [38] to assess the semantic level similarities of the programs.

My work is similar to the discussed above studies that apply the static analysis in the sense that it uses a combination of similarity measures which also represent static analysis approach. Some of the metrics are graph-based, e.g., those relying on AST or dependence graph over library functions and variables. Another one applies a topic model that attributes the terms of the source code, represented by tokens, to topics.

Other methods have been also used to test program correctness under the static analysis approach. More recently, [66] have used logic-based program synthesis techniques to automatically grade programming assignments, and [67] have used logic-based software verification in combination with CFG similarity to automatically grade assignments. Such logical analysis techniques are difficult to implement for my chosen domain of visual game programming, where problems do not come with crisp logical specifications (for example, our grading rubrics consist exclusively of statements like “Give 1 point if the program correctly determines whether a missile and a rock collide”).

Besides the correctness, program style is also assessed under the static analysis approach, as it is considered to be an important indicator of program quality. For instance, ASSYST [63] takes into consideration module length, number of component lines, and use of indentation. [68] assesses programming style by checking if programs follow a gathered set of coding rules falling into main categories such as modularity, typography, clarity and simplicity, independence, effectiveness, and reliability. Although, my work does not cover metrics for programming style assessment, new metrics can be very easily incorporated into the presented statistical model.

All the models discussed above significantly differ from mine in two areas of the automated grading process. First, the programming assignments in the above studies are graded deterministically based on given similarity measures. My work, however, defines a stochastic process to predict grades. Second, in presented related work, grades are calculated based on the similarity comparison to a model program or solution usually provided by the instructor. Here, already-graded programs create a training set; the various distance measures are then weighted to minimize the prediction error on the training data to infer the grades for the ungraded programs. A more similar approach to mine is presented in [39] where the grade is inferred from representative programs with known grades from three differing grade value clusters. A program score, however, is still deterministically calculated based on the similarity as a percentage.

Automatic program grading is not the only application of the program analysis methods. The methods can also be found in literature on software plagiarism detection [69–71], as well as in the clone detection [72], a topic in software engineering where the goal is to identify redundant copies of code in a code base. That work is similar to this one as it also uses distance metrics between program fragments. How-

ever, to the best of my knowledge, statistical approaches that learn from a database of labeled programs have not been explored in that literature.

The statistical analysis of the code has also been studied in domains different from automatic grading: [73–75] apply statistical methods in the area of statistical debugging where the goal is to infer likely predictors of software failures; the inference of invariants of programs from data has also been explored [76–78]; there is also a rich literature on the mining of program specifications [79]. The goals and mechanics of these approaches are very different from those in my work, as they learn from data generated by executing a program and do not involve comparison of code fragments.

2.3 Senders, Receivers and Authors in Document Classification

The main goal of this part of my thesis is document classification. In particular, my main focus lies on emails. The main characteristic that differentiates an email from a general text document is the existence of additional information in the email headers that can be exploited for various mining tasks. In this work, besides the text from the email Body, “FROM”, “TO”, “CC”, and “BCC” fields of a header are also used (in the remaining part of this thesis also referred to as participant fields).

The email classification problem, and the classification problem in general, can be a single- or multi-label assignment. In single-label classification an example can be mapped to only one label, either of two possible ones, i.e., binary classification, or to one of many, i.e., multi-class classification. In multi-label classification, on the other hand, the labels in the label set are not mutually exclusive and thus, an example can be mapped to none/some/all the labels. My research applies to the

latter classification problem, namely multi-label classification.

As the result of the lack of public email data due to privacy issues, the amount of work published in the field of email classification is small compared to the one in the area of traditional text classification. The situation, however, changed in the recent years, when the Enron email data set [80] has been made available. In the experimental study, I use email messages coming from actual corporations in the finance and construction industries.

The very recent survey of research efforts in email mining [4] categorizes all the email mining efforts into five major tasks: spam detection, email foldering, contact analysis, email network property analysis, and email visualization. The last three tasks, however, are not closely related to the subject of my work and thus will not be discussed any further.

Spam filtering was one of the first uses of email classification . The main goal is to identify and sort out unsolicited commercial emails, i.e., spam, from the users's mail stream. Spam filtering belongs to the class of binary classification problems; i.e., emails are classified into two groups: spam or not-spam (also known as ham). Many state-of-the art classifiers have been used in spam filtering, e.g., naive Bayes and SVM. Naive Bayes was initially proposed in [81] in the context of spam detection. Since then, various studies focusing on that classification method were conducted; e.g., in [48] naive Bayes is trained either on messages from all the users or single users, [49] applies it to word n-grams, using only some of the first words to reduce the classification time. SVM was initially used in [82] where its performance using binary feature representation is found outstanding comparing to boosting with decision tress. In [83] SVM also outperforms naive Bayes and shows comparable performance to random forest. Although my research problem does not fall under the spam flittering

umbrella, it still can be tackled by the state-of-the-art algorithms mentioned above. In my work I use SVM as a baseline algorithm; moreover, I use the SVM classification output as an input to one of my models in order to improve on the SVM’s classification power.

Some studies in the literature use topic models [22] to represent the text content in spam detection. That work, however, is applied to web spam, which has recently attracted more attention in the research community in comparison to email spam. Although the subject of the methods is not an email, the discussed topic-based techniques can be easily applied to email spam detection. Originally, topic models were used in document classification in [29]. There, a response variable associated with each document was added to Traditional Latent Dirichlet Allocation (LDA) [21]. LDA is also used in [84] to represent spam and ham as distributions of topics. The probability vectors are then used as classification features. [5] also applies LDA to extract sentence-level topic information for spam and not-spam examples. The authors propose two models: (1) A single topic is assigned to each sentence by majority-voting heuristic of word-topic assignments; the topic transitions between adjacent sentences is modeled (2) A topic distribution for each sentence is used to calculate the difference in topic distribution between adjacent sentences. One of my methods is similar to those studies in that it also applies topic modeling to represent content of the email. However, it is not a traditional LDA but a novel topic-based approach also incorporating the information about senders and recipients (as a prior to topic mixing proportion vectors). Furthermore, the discussed models (except [29]) first learn an LDA model and then classify the documents using the topic proportion vectors as a separate step. My model classifies and learns a topic model at the same time. The reasoning behind it is that the quality of the resulting classification benefits from

simultaneously learning label assignments and a topic model that is conducive to the learned classification.

Automatic email “foldering”, the most common application of email classification, consists of assigning email messages to predefined folders. This area of research falls under the umbrella of multi-class classification problem. Different methods have been applied to categorizing emails into folders. Among the first studies in this area, [85] applies k Nearest Neighbors (k-NN) and a rule-based method. Another algorithm from the rule induction family is introduced in [86] in a “one vs. all” classification approach. Naive Bayes is applied in [87], a freely available e-mail foldering system, and SVM and a TF-IDF similarity-based classifier in [88]. The superior abilities of SVM over naive Bayes were shown in [89] which was recently reprinted as [90], being the most influential paper of 2001. In that work, a co-training technique is used to classify emails where two independent classifiers are trained on two separate sets of features: words from the Subject and Body of the email. Then, the unlabeled data is added into training in the loop fashion; both classifiers label new examples and add most confidently predicted into the set of labeled examples. The authors of [51] extend the previous work by comparing email classification results on four classifiers: Maximum Entropy, naive Bayes, SVM and a variant of Winnow, where all the experiments are based on words frequencies. It is noted that sophisticated methods should be applied to email foldering due to low accuracies obtained by the baseline algorithms tested.

The more recent research in email foldering includes [50] where the authors improve the Naive Bayesian Multinomial classifier by addressing classes/folders imbalance with a distribution-based dataset balancing approach. The emails are represented by word frequencies as well as by TF-IDFs. They also find SVM being very robust under

imbalanced conditions resulting in no performance gain after balancing. None of the studies, however, has shown a particular method being superior to the others for a large variety of datasets. All the studies discussed above address the problem of single-label classification. My work concentrates on multi-label classification. This aspect differentiates it from all the related work presented above.

There are three main types of methods for solving the multi-label classification problem, as identified in [52]: problem transformation methods, algorithm adaptation methods and ensemble methods. Problem transformation methods transform the multi-label learning problem into one or more binary classification problems. The most common solution is to decompose the problem into multiple, independent binary classification problems and to aggregate the classification results from all the binary classifiers. Although this approach enables the use of state-of-the-art classifiers, it does not retain the correlations between existing labels. The other possible approach that takes the label dependencies into account transforms the multi-label problem into a multi-class problem where the classes are all the possible combinations of the original categories. This approach suffers from the data sparseness problem where some classes may be represented only by a few examples.

The algorithm adaptation methods, the second category of methods applied to multi-label classification, try to tackle the multi-label problem directly often by extending existing algorithms. The last group, ensemble methods, includes ensembles of classifiers that use the first two categories as base classifiers. The two algorithms, SVM and the Bayesian Lasso, which I consider a baseline in my work as well as use their output as an input to one of my models, act as binary classifiers in the one-vs-all way; hence they represent the problem transformation category. My models, however, are designed to handle the multi-label classification problem in its full form and thus can

be considered an algorithm adaptation method.

The work on the algorithm adaptation methods for the multi-label text classification problem, particularly for email classification, is limited. The related examples are in the general text classification field and include [53] that simultaneously detects multiple categories of text using the generative Parametric Mixture Model. The authors show that the method can significantly outperform the conventional combination of binary methods when applied to multi-label text categorization using World Wide Web pages. The method, however, assumes that the multi-label text contains words that appear in single-labeled text that belongs to each category of the multi-categories, and this assumption does not always reflect the real-world data configuration. My work is complementary to the discussed one as both studies present a generative model for multi-label classification. Another study showing an algorithm adaptation method outperforming the problem transformation is [54]. There, the authors propose a multi-label classification method using the maximum entropy principle that explicitly models the correlations among data categories. Both of the methods discussed above, similarly to the one presented in this thesis, explicitly model the category correlations and do not require a threshold to determine the label for each example. They differ, however, in their application, i.e., are not applied to email classification problem, and as a result they naturally use only the Body of documents to make the classification decisions.

Besides multi-label classification, multi-label learning also includes the concept of multi-label ranking. This problem can be considered a generalization of multi-class classification as instead of predicting only the top label, it predicts the ranking of all labels. A similar approach is adopted in [91] to automatic tagging of emails with folder names. It is, however, not a classical “foldering” case as those in the studies al-

ready presented above. The authors consider the global domain of folders, i.e., many users simultaneously as opposed to separately learning foldering habits of single users. The first top tag is predicted by using modeling methods which map both tags and email messages into a joint low dimensional latent space referenced to as the latent factor representation. This representation is then learned in two alternative ways: by the maximum likelihood approach and the normalized hinge loss approach. The rest of the tags are suggested by using an empirical confusion matrix which records for each tag the tags it is most commonly confused with. This way, a single tag classification output returns multiple ranked tags. The work is similar to mine in terms of using a low dimensional latent space email representation. However, a significant aspect differentiating my work is learning multiple labels (multiple-label classification). Although multi-label ranking provides a way to handle the multi-labelled classification problem, it does not generally explicitly model the correlations between data categories, which my study does. Furthermore, it is difficult to determine into how many categories a particular example should be classified, and thresholds are usually selected heuristically.

The way the email data is represented as an input to classification methods also differentiates work available in the literature. The most commonly used features are terms extracted from the Body and the Subject fields of the email. The novel part of my approach is to learn the latent position of email network participants and to use this information, beside the email content, to support email classification. Although to the best of my knowledge, there is no similar work in the aspect of representing the senders and recipients of the message, there are still some examples in the literature that use the sender/recipient data in the classification process. These examples are discussed below.

In most cases, the values of “FROM”, “TO”, “CC”, and “BCC” fields are incorporated into the feature vector. The hybrid learning system for incoming email classification [92], combining SVM with Bernoulli naive Bayes classifier, creates a boolean feature for each sender, one boolean feature for a unique set of the union of “FROM”, “TO”, “CC”, and “BCC”, and a boolean feature for each distinct word in the Subject field of the message. Interestingly, the email body was not found useful by the authors in email classification.

As followed up work [93], the authors investigated replacing SVM with a Bernoulli naive Bayes (BNB), Multinomial naive Bayes (MNB), Transformed Weight-Normalized Complement naive Bayes, TFIDF counts, Online Passive Aggressive algorithms, and linear Confidence Weighted (CW) classifier. The results show CW and BNB performing the best for email classification with MNB and TFIDF giving the poorest results. The lack of additional predictive value of the email Body is not confirmed there as some of the algorithms improved their performance with the use of the Body field.

The authors of [94] process email addresses into a bag-of-participants which then creates a separate vector, besides the one based on the bag-of-words for the Subject and Body fields. Those vectors are then classified by three independent classifiers: for Subject, Body and participants, and the results are combined by a meta-classifier using a majority-voting technique. The results show that the combination of the three fields yields the best results. A similar approach but yielding different results was first proposed in [80], where scores from separate SVM classifiers trained on “FROM”, “TO, CC”, “Subject”, and “Body” fields are combined linearly. The weights for each are learned using ridge regression. The Body of the email shows to be the most useful feature in classification followed by the “FROM” field, with “TO” and “CC” fields being the least useful.

A very interesting approach to representing sender/recipient data is proposed in [95], an SVM classifier for personalized email prioritization. Besides using “FROM”, “TO”, and “CC”, Subject and Body fields of an email, the authors create a personal social network and use a Newman Clustering algorithm to obtain disconnected component clusters representing social groups. The feature vector includes a boolean feature for each of the cluster denoting if the user belongs to the same clusters as the sender of the message.

The clustering of social data yielding additional features for email classification is also used in [96]. Here, the information bottleneck method is used to group the addresses from “TO”, “FROM”, “CC”, and “BCC” fields into communication groups, based on the supposition that emails within the same group are more likely to be classified into the same category.

Social features based on the degree of interaction between sender and recipient, e.g., the percentage of a sender’s email that is read by the recipient, are included in [97]. The method presented there, Priority Inbox for Gmail, applies linear logistic regression to rank emails by the probability that the user will perform an action on it. According to the authors, the method is a valuable feature for Gmail.

Topic modeling was also used to represent the authors of the documents [46]. In this author-topic model, an author is chosen for each word in the document and then a topic is chosen from a distribution of topics specific to that author; the word is generated from the chosen topic. Although, the subject of the modeling are standard documents, the model could be also used to represent emails. However, the goal of that work is not a classification but characterization of the document content. As in other work discussed before, the learnt topics could be used as regressors in any classification model.

Chapter 3

Topic Models for Feature Selection in Document Clustering

3.1 Problem Definition

Topic models [22] such as LDA [21] and PLSI [23] are statistical models most often used for analyzing document corpora. A *topic* in a topic model is a set of words that tend to appear together in a corpus, and the documents in the corpus are viewed as being produced by some mixture of topics. There are many reasons to build a topic model for a corpus. The modeling process typically results in each document being represented as a vector of topic proportions, where each entry of this vector describes the importance of a particular topic in the document (I subsequently refer to this vector as a document’s “topic mixing proportion” vector). Thus, topic modeling is useful as a pre-processing step for any number of other mining or information retrieval tasks. One can first build a topic model, and then use this vector as a list of features describing the document, in addition to (or in lieu of) other features that might be collected. The resulting feature vector can then be used along with a favorite algorithm or model to perform clustering, classification, outlier detection, similarity search, etc.

In this chapter, I examine the utility of topic modeling as a pre-processing step for performing unsupervised clustering of the documents in a corpus. I consider the utility of the following process:

1. First, build a topic model for the corpus.
2. Next, use the model to obtain for each document a topic mixing proportion vector θ that represents the document’s location in “topic space”. In most topic models, the entries sum to one and the i th value is the prevalence of topic i in the document.
3. Finally, use all of the θ vectors as input to an unsupervised clustering algorithm.

Given that clustering is one of the most common data mining tasks, this seems like an obvious application for topic modeling.

My Contributions. My work investigates different topic models that can be used in Step 1 of the process presented above. Rather than proposing and advocating a particular technique to topic modeling for document clustering, I instead seek to develop and rigorously evaluate a few alternatives that represent two approaches: (1) Learning a traditional topic model for a corpus; here vanilla LDA is used (2) Building a novel topic model that integrates a discrete mixture to simultaneously learn the clustering structure and the topic model that is conducive to the learned structure. This approach allows to obtain better results because the clustering can inform the selection of topics. As an example a clustering of a corpus of data mining papers can be used. In a high-quality clustering result, one might find one cluster containing papers on frequent itemset mining, another on graph mining, another on text mining, and so on. But a topic model learned over such a corpus might easily contain a topic favoring the words: “thank”, “acknowledge”, “NSF”, “graciously”, “anonymous”, “reviewers”, and so on. This topic corresponds to the acknowledgments section of a paper, which some papers contain and others do not—but whether a paper has an acknowledgments section would appear to have no relationship to any high-

quality clustering of the corpus. If the topic modeling and clustering are performed simultaneously, then the fact that the “acknowledgments” topic is not discriminative can inform the topic modeling, and the topic can be dropped and replaced with one that is more useful in providing a meaningful clustering.

The first alternative I consider is the simplest and most obvious: use “vanilla” LDA to produce the θ vectors that are used as input to the clustering. LDA is an obvious choice as it is now one of the single most widely used algorithms for processing natural language text. I propose two variants of the second approach, i.e., Mixture of Priors, and Directional Topic Mixture Model.

However, I have reason to expect that “vanilla” LDA is not going to be the very best option for pre-processing text as input to a clustering algorithm. LDA makes use of a Dirichlet prior on the θ vectors which govern the mixture of topics that are found in each document. This prior has a parameter α that controls the mean of the various θ vectors, as well as how widely the vectors are dispersed from that mean. The problem is that the Dirichlet prior is quite well-behaved: it does not encourage a “bumpy” distribution of θ vectors, which is what one would desire as input to a clustering algorithm. Instead, if α has a large L1-norm, the prior has a single mode, which tends to force all documents to have the same proportion of each topic. Or, if the prior has a small L1-norm, it tends to push all of the probability mass to the edges of the simplex—that is, the prior favors documents that have a few topics at the exclusion of all of the others. I might expect that a prior which favors such a distribution of topic mixing proportions would not lead to the best clustering results.

Because of this, I consider two alternatives to “vanilla” LDA that can be used to pre-process a corpus as input to a clustering algorithm. The first is a simple modification to the LDA model that I call the *mixture of priors* (MoP) alternative.

Rather than each document sharing the same prior on its topic mixing proportions, in the MoP alternative, a document chooses its prior from a mixture of K different priors. This naturally allows for a bumpy prior on the distribution of the θ vectors, and my hypothesis is that this would result in data that are more amenable to clustering.

I then propose a second alternative that (unlike LDA) does not rely on a Dirichlet distribution to govern to what extent the various topics influence production of a document. This alternative, which I call the *directional topic mixture model* (DTMM), relies on a spherical distribution to place documents on a unit sphere; the position of a document on the sphere controls how topics are mixed within the document. By utilizing K mixture components in the spherical distribution, I obtain a “bumpy” prior on the topic mixing proportions. As I describe here, the hoped-for benefit of this approach is that the spherical distribution allows for a document’s cluster membership to have a more direct influence on the topic mixing proportions than in the MoP alternative, where cluster membership supplies only a prior to a document’s topic mixing proportion.

3.2 Mixture-of-Priors Variant

The first option I consider for clustering documents using a topic modeling is a simple extension to the LDA model.

3.2.1 The Model

In LDA, each word in a document is produced by one of T topics; the probability of an arbitrary word in document j being produced by topic t is given by the t th entry in the vector θ_j , which I refer to as the document’s *topic mixing proportion* vector. In “classic” LDA, θ_j is sampled from a Dirichlet distribution with parameter α ; α is

typically taken to be a symmetric, constant vector (in practice, $\alpha = \langle 0.1, 0.1, \dots, 0.1 \rangle$ is often used). In the “mixture-of-priors” variant that I propose (or MoP for short), the α vector associated with document j is instead sampled from a mixture model; the identity of the mixture component producing this alpha vector is then the cluster identity for document j , and the documents in the same mixture component tend to have similar topic proportion vectors.

Given T topics, K clusters, a dictionary with M words, and a corpus with N documents where the length of document j is n_j , the modified generative process of the MoP variant is:

1. $\phi \sim \text{Dirichlet}(\gamma)$

2. For $t \in \{1 \dots T\}$:

- (a) $\Psi_t \sim \text{Dirichlet}(\beta)$

- (b) For $k \in \{1 \dots K\}$:

$$\alpha_{k,t} \sim \text{InvGamma}(a, b)$$

3. For $j \in \{1 \dots N\}$:

- (a) $c_j \sim \text{Categorical}(\phi)$

- (b) $\theta_j \sim \text{Dirichlet}(\alpha_{c_j})$

- (c) $z_j \sim \text{Multinomial}(n_j, \theta_j)$

- (d) For $t \in \{1 \dots T\}$:

$$w_{j,t} \sim \text{Multinomial}(z_{j,t}, \Psi_t)$$

γ, β are sampled from uninformative prior $\text{InvGamma}(a, b)$, and a , and b are assumed to be known hyperparameters.

The process can be summarized as follows. In step (1) the cluster weight vector ϕ is produced, where ϕ_k is the probability of associating with cluster k . Step (2) produces two variables. First, it produces a matrix Ψ where $\Psi_{t,r}$ is the probability that topic t will produce word r . I will subsequently use $\Psi_{*,r}$ to denote the column listing the probability that each topic will produce word r . Second, a matrix α is produced, where α_k is used as the prior on the topic proportion vector for each document in cluster k .

Step (3) generates the actual documents. To produce a document, the cluster identity c_j is first generated. Next, the topic proportion vector θ_j is produced using the prior α_{c_j} common to all documents in cluster c_j . Using θ_j , the n_j words in the document are allocated to the T topics; the resulting allocation is stored in the vector z_j , where $z_{j,t}$ is the number of words in document j produced by topic t . Finally, the matrix w_j is generated having T rows and M columns, where $w_{j,t,r}$ indicates the number of times word r was produced by topic t in document j .

After this process completes, the j th document in the corpus (represented by a vector of word frequencies) is $d_j = \sum_t w_{j,t}$.

Note that aside from the use of a mixture model to supply the prior on each document's topic proportion vector, the generative process is equivalent to the process used in LDA, though I have relied on a more matrix- and vector-oriented description than is commonly used. The Bayesian network of the MoP model is depicted in Figure 3.1.

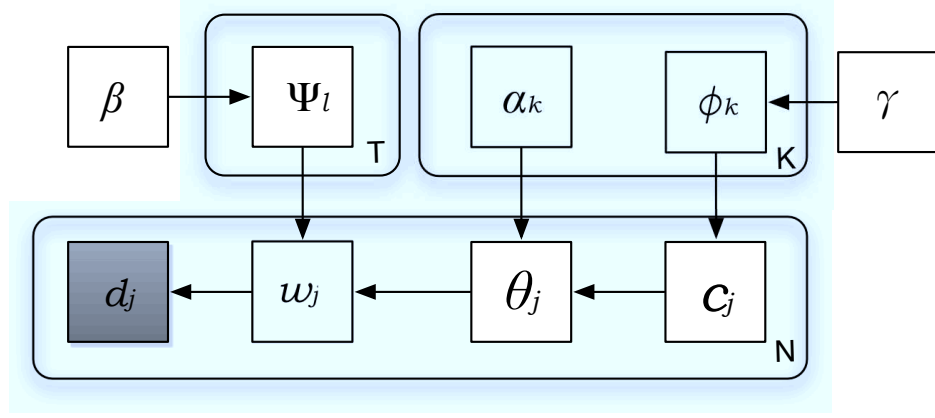


Figure 3.1 : Plate diagram for the MoP model.

3.2.2 Inference

Given the above generative process, many methods can be used to infer the posterior distribution:

$$P(\phi, \{\Psi_t\}_{t=1}^T, \{\alpha_k\}_{k=1}^K, \{c_j, \theta_j\}_{j=1}^N, \gamma, \beta, \{d_j\}_{j=1}^N | \{w_j\}_{j=1}^N).$$

Let Θ refer to the parameter set $\{\phi, \gamma, \beta\} \cup \{\Psi_t\}_{t=1}^T \cup \{\alpha_k\}_{k=1}^K \cup \{c_j, \theta_j, w_j\}_{j=1}^N \cup \{d_j\}_{j=1}^N$ and let D refer to the observed data $\{d_j\}_{j=1}^N$. From elementary probability, it is known that:

$$P(\Theta|D) = \frac{P(D|\Theta)P(\Theta)}{P(D)}$$

Based on the MoP generative process described above and the Figure 3.1, we have:

$$P(D|\Theta) = \prod_j \prod_t \text{Multinomial}(d_{j,t} | w_{j,t})$$

and

$$P(\Theta) = P(\phi)P(\Psi)P(\alpha)P(c)P(\theta)P(\gamma)P(\beta)$$

where:

$$\begin{aligned}
P(\phi) &= \text{Dirichlet}(\phi|\gamma) \\
P(\Psi_t) &= \text{Dirichlet}(\Psi_t|\beta) \\
P(\alpha_{k,t}) &= \text{InvGamma}(\alpha_{k,t}|a, b) \\
P(c_j) &= \text{Categorical}(c_j|\phi) \\
P(\theta_j) &= \text{Dirichlet}(\theta_j|\alpha_{c_j}) \\
P(w_j) &= \prod_t \text{Multinomial}(w_{j,t}|\theta_j, \Psi_t) \\
P(\gamma) &= \text{InvGamma}(\gamma|a, b) \\
P(\beta) &= \text{InvGamma}(\beta|a, b)
\end{aligned}$$

Perhaps the most straightforward method to infer the posterior distribution is to use a Gibbs sampler [20]. A Gibbs sampler is easy to derive and apply here because (with one exception) all of the variables are discrete, or else the priors used are conjugate so sampling from the posterior is easy. Briefly, applying a Gibbs sampler requires that I be able to repeatedly re-sample the value of each unseen variable from its posterior distribution, conditioned upon the current values of all of the other variables.

The update samplers for Ψ_t , θ_j , and $w_{j,*,r}$ are given as:

$$\begin{aligned}
\Psi_t &\sim \text{Dirichlet}(\beta + \sum_j w_{j,t}) \\
\theta_j &\sim \text{Dirichlet}(\alpha_{c_j} + z_j) \\
w_{j,*,r} &\sim \text{Multinomial}(d_{j,r}, \theta_j \times \Psi_{*,r})
\end{aligned}$$

The final two updates require a bit of explanation. Let us first consider the cluster membership c_j . Here, we need to compute the posterior probability for each value

explicitly, by evaluating the probability of observing θ_j given that membership. Let p_j denote a vector where $p_{j,k} = \phi_k \times \text{Dirichlet}(\theta_j|\alpha_k)$. Then, after normalizing p_j to ensure that the distribution sums to one, we have:

$$c_j \sim \text{Categorical}(p_j).$$

The most difficult update is the per-cluster prior on the topic proportion vector, because the variable is continuous and there is no applicable conjugate relationship. Hence a rejection sampler [20] must be used. Let α'_k denote the version of α_k , where the t th entry in the vector has been replaced with the candidate value α'_{k_t} . Then the posterior likelihood of α_{k_t} is computed as:

$$\alpha_{k_t} \propto \text{InvGamma}(\alpha'_{k_t}|a, b) \prod_j \begin{cases} \text{Dirichlet}(\theta_j|\alpha'_k) & \text{if } c_j = k \\ 1 & \text{if } c_j \neq k \end{cases}$$

Further, let β' and γ' be the candidate values for β and γ . Then the posterior likelihoods are given as:

$$\begin{aligned} \beta &\propto \text{InvGamma}(\beta'|a, b) \prod_t \text{Dirichlet}(\Psi_t|\beta') \\ \gamma &\propto \text{InvGamma}(\gamma'|a, b) \text{Dirichlet}(\phi|\gamma') \end{aligned}$$

3.3 Directional Topic MM

The second option I consider for clustering documents using a topic modeling is a directional topic mixture model that does not rely on a Dirichlet distribution but instead on a spherical one.

3.3.1 Motivation

One worry with the MoP model is as follows. Consider a generic Bayesian inference problem where we have a vector \mathbf{x} of observations sampled from a variable with density function $g_1(\mathbf{x}|\theta)$, where the parameter θ has a prior $g_2(\theta)$. As the number of observations in \mathbf{x} grows, the importance of $g_2(\theta)$ to the posterior distribution $f(\theta|\mathbf{x}) \propto g_1(\mathbf{x}|\theta)g_2(\theta)$ diminishes until, for all practical purposes, it disappears entirely.

This is generally seen as a feature of the Bayesian approach, in that it provides a principled mechanism for reducing the influence of prior belief as the amount of observed data increases. But due to the same phenomenon, one must take care when designing a model where one expects a prior distribution such as $g_2(\theta)$ to influence the inference process, especially if the amount of observed data will be large. Eventually, the data will swamp the prior.

One may reasonably worry that this will happen with the MoP model. The documents in a cluster share only a prior on their topic proportion vectors. If a document is of any significant length, the importance of the prior on θ_j vanishes. This is obvious from the update sampler for θ_j :

$$\theta_j \sim \text{Dirichlet}(\alpha_{c_j} + z_j)$$

Since $|z_j|_1 = n_j$, for any fixed value of α_{c_j} , as n_j increases, α_{c_j} decreases in importance. Thus, for large n_j , cluster membership will have little influence upon the inferred topic proportion vectors, and hence little influence upon the inference of the learned topics. If this is the case, one might as well first learn an LDA model over the corpus, and then cluster the documents using the topic proportion vectors as a separate step. There is nothing wrong with such an approach—as a matter of fact, this is an approach I will explore experimentally later in the paper—but the whole reason for clustering the

documents simultaneously with learning a topic model is the hope that the quality of the resulting clustering will benefit from simultaneously learning a clustering and a topic model that is conducive to the learned clustering.

Given such a concern, it is reasonable to ask how this could be addressed. An obvious idea would be to somehow make the prior on the topic mixing proportion “stronger” as the documents become larger, so that α_{c_j} is not dominated by z_j . Practically speaking, this means ensuring that $|\alpha_{c_j}|_1$ is always sufficiently large. Unfortunately, this is not easy to do. Given the $\text{Gamma}(a, b)$ prior on each $\alpha_{k,t}$, and given that the mean of a $\text{Gamma}(a, b)$ distribution is ab , I could choose to increase the shape a , or the scale b , or both. Unfortunately, neither option is very palatable:

- Increasing the shape decreases the variance of the distribution; the effect of this is that I will ensure a strong, symmetric prior on the topic mixing proportions. In other words, I will tend to ensure that every entry in θ_j is $1/T$. This would clearly result in a poor clustering.
- Increasing the scale does allow for large entries in each α_k vector, but it does not discourage small values in any meaningful way. Hence, the inference process will still be free to choose α_k vectors with small magnitude, and I have not solved the problem.

One could think of ways to engineer a way around this—such as adding an explicit constraint that each $|\alpha_j|$ be large enough—but in the end, no obvious modification seems to be entirely satisfactory. Thus, I decided to try a different approach that does not rely on a Dirichlet distribution to produce the topic proportion vector, and which allows for very explicit control of the types of topic proportion vectors that are allowed.

3.3.2 The Model

The idea I explore is simple. Rather than utilizing a mixture of Dirichlet distributions to produce the topic mixing proportions, I instead utilize a mixture of T -dimensional von Mises Fisher distributions [24] (or vMF distributions for short). The vMF distribution is a continuous, multivariate distribution over the unit sphere. I take my inspiration from some existing work on using spherical models for clustering and analysis [25, 26] including a recent spherical take on the task of topic modeling [27].

The vMF distribution takes two parameters: a mean position on the unit sphere μ , and a “concentration” parameter κ that determines the spread of the distribution across the surface of the sphere—larger κ values reduce the spread. To facilitate a clustering of the documents, I use a mixture vMF distributions to place the j th document in a latent position θ_j on the unit sphere. The identities of the various mixture components used to produce the documents’ positions can be used to segment the documents into clusters.

To be able to use the document’s latent position on the sphere to choose the topics that are used to produce the document, the document’s position is mapped to a position on the $(T - 1)$ -simplex by the following transformation:

$$\theta'_{j,k} = \frac{\rho^{\theta_{j,k}}}{\sum_i \rho^{\theta_{j,i}}}.$$

I use $f(\theta_j, \rho)$ to denote the function that takes as input a latent position θ and applies this transformation to every one of the T dimensions, and returns the resulting vector.

Note that this is similar to the transformation used by the logistic normal distribution [28] to map the output of a normal random variate in R^d to the simplex, the key difference is that the logistic normal transformation uses a fixed $\rho = e$ (where e is the base of the natural logarithm). The reason for the additional parameter ρ in my

case is that since documents are positioned on the unit sphere (rather than in R^d), without ρ the extent to which a document could suppress or promote a particular topic would be limited.

Let us consider the two-topic case. To produce the most extreme allocation of topics possible, we would have $\theta'_j = \langle \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \rangle$. If $\rho = e$, then we would have:

$$\theta'_{j,1} = \frac{e^{\frac{1}{\sqrt{2}}}}{e^{\frac{1}{\sqrt{2}}} + e^{-\frac{1}{\sqrt{2}}}} = 0.80, \quad \theta'_{j,2} = \frac{e^{-\frac{1}{\sqrt{2}}}}{e^{\frac{1}{\sqrt{2}}} + e^{-\frac{1}{\sqrt{2}}}} = 0.20$$

But by increasing ρ , we can increase the extent to which documents can promote or suppress a particular topic. If $\rho = 10$, then we have:

$$\theta'_{j,1} = \frac{10^{\frac{1}{\sqrt{2}}}}{10^{\frac{1}{\sqrt{2}}} + 10^{-\frac{1}{\sqrt{2}}}} = 0.96, \quad \theta'_{j,2} = \frac{10^{-\frac{1}{\sqrt{2}}}}{10^{\frac{1}{\sqrt{2}}} + 10^{-\frac{1}{\sqrt{2}}}} = 0.04$$

Thus, ρ can be seen as controlling the extent to which certain topics can dominate the creation of a particular document.

Given this, the generative process underlying the DTMM model is as follows:

1. $\phi \sim \text{Dirichlet}(\gamma)$
2. $\rho \sim \text{InvGamma}(1, 1)$
3. For $k \in \{1 \dots K\}$:
 - (a) $\mu_k \sim \text{vMF}(0, 0)$
 - (b) $\kappa_k \sim \text{Gamma}(1, 1)$
4. For $t \in \{1 \dots T\}$:

$$\Psi_t \sim \text{Dirichlet}(\beta)$$
5. For $j \in \{1 \dots N\}$:

- (a) $c_j \sim \text{Categorical}(\phi)$
- (b) $\theta_j \sim \text{vMF}(\mu_{c_j}, \kappa_{c_j})$
- (c) $z_j \sim \text{Multinomial}(n_j, f(\theta_j, \rho))$
- (d) For $t \in \{1 \dots T\}$:
 - $w_{j,t} \sim \text{Multinomial}(z_{j,t}, \Psi_t)$

This process resembles the LDA-based MoP generative process given in the previous section, with a few key differences. First, the parameter ρ is generated in step (2). In step (3), the mean and concentration parameters for each of the K mixture components are produced. Note that in step (3a), $\text{vMF}(0, 0)$ denotes a uniform distribution on the sphere. The other key difference compared to the MoP generative process is the use of the vMF distribution and the mapping function $f(\cdot)$ to produce the topic mixing proportion vector, as opposed to a Dirichlet distribution.

The Bayesian network of the DTMM model is depicted in Figure 3.2.

Why does this allow more control? I have asserted that a possible difficulty with the MoP alternative is the lack of control offered by the prior distribution on θ_j , and that there was no easy or obvious way to “tweak” the model to increase the importance of the prior. The DTMM model addresses this via the use of the κ_t parameter, which explicitly controls the extent to which the latent positions of the documents in a cluster can vary. With a large enough κ_t value, all of the documents in a cluster must have exactly the same latent position, meaning that they must all use exactly the same topic mixing proportion vector—which in turn means that the clustering may have more of a chance to influence the way the topics are constructed. If the latent positions of the documents in a cluster are all identical, then the prominent topics in that cluster should be built to fit those documents closely.

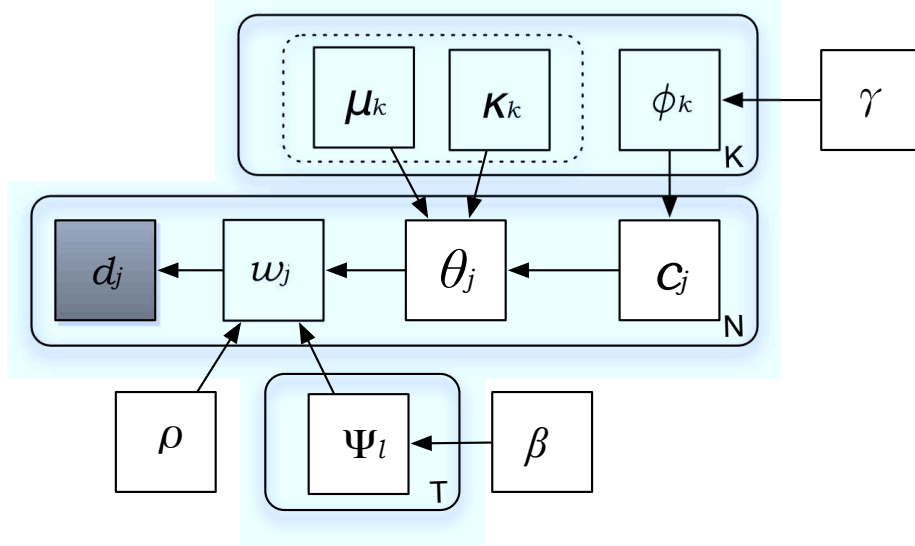


Figure 3.2 : Plate diagram for the DTMM model

In the generative process, I suggest an $\text{Gamma}(1, 1)$ prior on κ_t ; the use of such an uninformative prior will obviously to little all documents in a cluster to conform. But it is an easy matter to choose a prior that forces larger κ_t values, or even to use a large constant value for κ_t . I will explore these issues experimentally.

3.3.3 Inference

This subsection describes a Gibbs sampler that can be used to infer the posterior distribution:

$$P(\phi, \{\Psi_t\}_{t=1}^T, \{c_j, \theta_j\}_{j=1}^N, \{\mu_k, \kappa_k\}_{k=1}^K, \rho, \gamma, \beta, \{d_j\}_{j=1}^N | \{w_j\}_{j=1}^N)$$

Let Θ refer to the parameter set $\{\phi, \rho, \gamma, \beta\} \cup \{\Psi_t\}_{t=1}^T \cup \{c_j, \theta_j\}_{j=1}^N \cup \{\mu_k, \kappa_k\}_{k=1}^K \cup \{d_j\}_{j=1}^N$ and let D refer to the observed data $\{d_j\}_{j=1}^N$. From elementary probability, we know that:

$$P(\Theta|D) = \frac{P(D|\Theta)P(\Theta)}{P(D)}$$

Based on the DTMM generative process described above and the Figure 3.2, we have:

$$P(D|\Theta) = \prod_j \prod_t \text{Multinomial}(d_{j,t}|w_{j,t})$$

and

$$P(\Theta) = P(\phi)P(\Psi)P(c)P(\theta)P(w)P(\mu)P(\kappa)P(\rho)P(\gamma)P(\beta)$$

where:

$$P(\phi) = \text{Dirichlet}(\phi|\gamma)$$

$$P(\Psi_t) = \text{Dirichlet}(\Psi_t|\beta)$$

$$P(c_j) = \text{Categorical}(c_j|\phi)$$

$$P(\theta_j) = \prod_k [\text{vMF}(\theta_j|\mu_k, \kappa_k)]^{c_{jk}}$$

$$P(w_j) = \prod_t \text{Multinomial}(w_{j,t}|f(\theta_j, \rho), \Psi_t)$$

$$P(\mu_k) = \text{vMF}(\mu_k|0, 0)$$

$$P(\kappa_k) = \text{InvGamma}(\kappa_k|a, b)$$

$$P(\rho) = \text{InvGamma}(\rho|a, b)$$

$$P(\gamma) = \text{InvGamma}(\gamma|a, b)$$

$$P(\beta) = \text{InvGamma}(\beta|a, b)$$

It is a relatively easy matter to use this formula to derive a Gibbs sampler for all of the hidden variables. The update samplers for Ψ_t and ϕ are based on conjugate priors and are given as:

$$\Psi_t \sim \text{Dirichlet}(\beta + \sum_j w_{j,t})$$

$$\phi \sim \text{Dirichlet}(\gamma + \sum_j c_j)$$

Just as in the MoP model, $w_{j,*,r}$ and c_j are discrete, so we have:

$$w_{j,*,r} \sim \text{Multinomial}(d_{j,r}, f(\theta_j, \rho) \times \Psi_{*,r})$$

$$c_j \sim \text{Categorical}(p_j)$$

Here, p_j denotes a normalized vector where $p_{j,k} = \phi_k \times vMF(\mu_k, \kappa_k)$.

The rest of the variables are continuous and the priors are not conjugate. Thus a rejection sampler is used. Let κ'_k , ρ' , β' , and γ' denote the candidate values of κ_k , ρ , β , and γ respectively. Then the posterior likelihoods are computed as:

$$\kappa_k \propto \text{Gamma}(\kappa'_k | a, b) \prod_t \begin{cases} \text{vMF}(\theta_j | \mu_k, \kappa'_k) & \text{if } c_j = k \\ 1 & \text{if } c_j \neq k \end{cases}$$

$$\rho \propto \text{InvGamma}(\rho' | a, b) \prod_j \text{Multinomial}(d_j | f(\theta_j, \rho'))$$

$$\beta \propto \text{InvGamma}(\beta' | a, b) \prod_t \text{Dirichlet}(\Psi_t | \beta')$$

$$\gamma \propto \text{InvGamma}(\gamma' | a, b) \text{Dirichlet}(\phi | \gamma')$$

The last two updates, i.e., for θ_j and μ_k , are the most complex. Both θ_j and μ_k are vectors of dimensionality T , i.e., the number of topics, as they denote points sampled from T -dimensional vMF distribution over the unit sphere. Both variables are sampled in a similar manner with the use of a rejection sampler. Let θ'_j and μ'_k be the version of θ_j and μ_k , where the t th entry in the vector has been replaced with the candidate value θ_{j_t} and μ_{k_t} respectively. For $T-1$ scalar values, i.e., entries in the vectors, the posterior likelihoods of θ_{j_t} and μ_{k_t} are computed as:

$$\mu_{k_t} \propto \prod_j \begin{cases} \text{vMF}(\theta_j | \mu'_k, \kappa_k) & \text{if } c_j = k \\ 1 & \text{if } c_j \neq k \end{cases}$$

and

$$\theta_{j_t} \propto \prod_j \text{Multinomial}(d_j | f(\theta'_j, \rho)) \times \begin{cases} \text{vMF}(\theta'_j | \mu_k, \kappa_k) & \text{if } c_j = k \\ 1 & \text{if } c_j \neq k \end{cases}$$

where $t \neq T$

One difficulty is that since each θ and μ must lie on the unit sphere, updating the positions on dimension at a time is not possible. Once $T - 1$ coordinates have been determined, only the sign of the final coordinate is unknown. I handle this by treating both θ_{j_T} and μ_{k_T} as slack variables. Whenever a candidate value for θ_{j_t} or μ_{k_t} for $t \neq T$ are evaluated, the value of θ_{j_T} (resp., μ_{k_T}) is automatically updated (except for the sign) to take into account the new value. This is done as follows:

$$\theta_{j_T} = \sqrt{1 - \sum_{i=1}^{T-1} \theta_{j_i}^2} ; \quad \mu_{j_T} = \sqrt{1 - \sum_{i=1}^{T-1} \mu_{j_i}^2}$$

When it comes time to actually update θ_{j_t} or μ_{k_t} , only the sign (positive or negative) is sampled. This is accomplished by evaluating the probability q_θ and q_μ of observing $|\theta_{j_t}|$ and $|\mu_{k_t}|$ based on posterior likelihoods presented above, where θ'_j and μ'_k are the versions of θ_j and μ_k , where the T th entry is replaced by its absolute value. Then, we can update the sign of the T th entry of θ_{j_t} and μ_{k_t} by sampling from $\text{Bernoulli}(q_\theta)$ and $\text{Bernoulli}(q_\mu)$ respectively.

3.4 Experimental Study

In this section, I present an experimental study of the three methods. My goal is to discover whether there is any qualitative reason to prefer one of the methods to the

others.

3.4.1 Methodology

My basic methodology is to use each of the three methods on a number of data sets with known clusterings, and to compare how well the resulting clusterings match the expected results.

To pre-process each of the text data sets, I used the the Natural Language Toolkit (NLTK) [98]. Documents were split into sentences and stop words were removed. In order to keep the meanings of the words and also reduce the size of the dictionary, I utilized lemmatizing as opposed to stemming. I created the dictionary for each corpus using n-grams as opposed to unigrams (I found that adding multi-word expressions almost uniformly improved the clustering results by a small amount). This was done by applying Turbo Topics [99] algorithm. This method finds multi-word expressions by using a back-off language model defined for arbitrary length expressions, and recursively employs the distribution-free permutation test to find significant phrases. Although the Turbo Topics model was originally developed to better visualize unigram topics models (after the model gets fit), I felt its use would be beneficial in the document pre-processing phase.

I utilized five data sets:

CMU Newsgroups*. This is the canonical text data set. I tested my algorithms on two subsets of the original data set.

1. **5 Newsgroups** is a collection of 5 different newsgroups: rec.sport.baseball, sci.crypt, sci.med, sci.space, soc.religion.christian. The data set contains 4874

*<http://kdd.ics.uci.edu/databases/20newsgroups/>

documents.

2. **10 Newsgroups** is a collection of 10 different newsgroups: comp.graphics, rec.autos, rec.sport.baseball, soc.religion.christian, sci.crypt, sci.electronics, sci.med, sci.space, talk.politics.guns, talk.politics.mideast. This dataset contains 9753 documents.

Classic 3[†]. Classic 3 is another well-known collection of documents. It contains 3893 documents from three well-separated sources; 1400 CRANFIELD documents are from aeronautical system papers, 1033 MEDLINE documents are from medical journals and 1460 CISI documents are from information retrieval papers.

Reuters 8[‡]. Reuters 8 is a subset of Reuters 21578 that were manually classified by personnel Reuters Ltd. Due to the fact that the class distribution for these documents is very skewed, two sub-collections are usually considered for text categorization tasks: R10 and R90 - the sets of the 10 and 90 classes with the highest number of positive training examples. R8 is created based on R10 by eliminating all the documents with less than or with more than one topic. After pre-processing, the collection contains 7674 documents.

WebKB 4. WebKB 4 is a subset of original WebKB[§] data set that contains web pages collected from computer science departments of various universities in January 1997. The 8,282 pages were manually classified into the following categories: student, faculty, staff, department, course, project, other. For each class, the collection contains pages from four universities: Cornell, Texas, Washington, Wisconsin, and

[†]<ftp://ftp.cs.cornell.edu/pub/smart/>

[‡]<http://web.ist.utl.pt/acardoso/datasets/>

[§]<http://www.cs.cmu.edu/webkb/>

other miscellaneous pages collected from other universities. WebKB 4 was created by discarding classes Department and Staff due to a very low number of pages from each university. The class Other was also omitted because pages were very different among this class. The resulting collection contains 2948 documents.

Stock. This is a collection of stock price information for the S&P 500. Each stock is considered to be a “document”, so the corpus size is 500. If a stock goes up on a particular day, then the “word” corresponding to that day is added to the document. There are 10 years of data in the set, and so there are approximately 10×365 unique “words” in the corpus. The ten clusters in the data correspond to the GICS sector of the company (Industrials, Financials, Health Care, etc.). Since stocks in the same sector tend to move together, one would expect that the stocks (that is, the “documents”) in the same sector would cluster with one another.

I tested four different options:

1. Spherical k-means, using cosine distance.
2. First learn an MoP model, then run k-means on the resulting model.
3. First learn a DTMM model, then run k-means on the resulting model.
4. First learn an LDA model, then run k-means on the resulting model.

All of the topic models were trained using 40 topics.

For each option, I performed two experiments. In the first experiment, exactly the correct number of clusters was used to perform the clustering. Then the accuracy was measured using the maximum matching in bipartite graph algorithm, where a matching is performed between the true and learned labels. That is, I computed the

best possible mapping of data points to “true” clusters, and the fraction of documents assigned to the correct cluster is used to measure the quality of the clustering.

In the second experiment, twice the correct number of clusters was used to perform the clustering. Again, the accuracy was measured using the maximum matching in bipartite graph algorithm permitting one-to-many assignments. In this case, the accuracy should be higher, since the underlying algorithm is allowed to group documents into smaller groups, which should thus be more “pure.”

Each clustering experiment was repeated five different times. All the models were implemented in C++ with the use of gsl-1.15 and Minuit2 libraries.

3.4.2 Results

Tables 3.1a and 3.1b show the average accuracy over the five runs for each of the four clustering methods.

To determine the significance of the results, I performed a simple, nonparametric bootstrap test where for each pair of methods a and b , I repeatedly performed the following: (1) randomly select one of the five data sets; (2) randomly select an observed clustering accuracy on that data set for both a and b ; (3) determine which method is most accurate on that data set. In Tables 3.2a and 3.2, I display the observed probability that a would perform better than b on a randomly-selected data set. I found that the MoP method is the best choice of the methods that I tested for clustering text documents. This method was followed by spherical k-means and the DTMM model, which performed similarly when the correct number of clusters was used (Table 3.2a).

Each achieved a higher average score than the other for three out of the six data sets shown in Table 3.1a. than the other for three out of the six data sets shown

in Table 3.1a. If the correct number of clusters was used, MoP was the best option in every case except for the Classic 3 data set, where the spherical k-means method achieved a near-perfect score (though both the MoP model and the spherical k-means were essentially equivalent on the Reuters 8 data set). Comparing the DTMM model and spherical k-means, it was unclear which one would be preferred. Each achieved a higher average score than the other for three out of the six data sets shown in Table 3.1a.

I also found that if the correct number of clusters is used, LDA was clearly the poorest choice for document clustering (Table 3.2a). It did outperform the spherical k-means algorithm on two of the data sets, but clearly outperformed the DTMM model only once and was never close to outperforming the MoP model. Interestingly, LDA performed much better when twice the correct number of clusters were used—see Table 3.2. Although, in this case it was still only the second poorest performance, LDA was competitive with spherical k-means, though it is still clearly outperformed by the MoP model.

I was curious as to why the three topic models performed as they did. The first thing that I checked was the quality of the topics themselves. In Table 3.4, I list the top words (by probability) for the topic that is most closely associated with each of the five clusters learned over the 5 Newsgroups data set. The words are given both for “vanilla” LDA, and for the topics learned by using the MoP model. In the table, words that are seemingly irrelevant to the newsgroup in question are bolded. It is clear that a much higher percentage of the top words associated with the LDA-based topic model were words that had little to do with the newsgroup in question. Terms such as “subject re,” “write,” and “article” were ubiquitous in the LDA topics. These are obviously important terms in the data, but have little to do with the particular

Dataset	spherical	k-means	k-means	k-means
	k-means	DTMM	MoP	LDA
5 Newsgroups	50.68	60.25	60.13	50.38
10 Newsgroups	27.57	48.77	56.76	48.63
Classic 3	96.47	69.30	89.41	59.00
Reuters 8	38.19	23.96	37.74	14.13
WebKB	38.36	40.57	52.95	49.96
Stocks	36.60	17.20	47.76	16.92

(a) Clustering methods using exactly the “correct” number of clusters

Dataset	spherical	k-means	k-means	k-means
	k-means	DTMM	MoP	LDA
5 Newsgroups	52.17	65.27	74.25	65.06
10 Newsgroups	33.10	62.40	74.79	66.58
Classic 3	87.92	80.14	95.98	77.85
Reuters 8	73.90	52.34	67.84	51.12
WebKB	53.05	48.07	53.45	60.15
Stocks	46.80	23.60	56.04	23.12

(b) Clustering methods using twice the “correct” number of clusters

Table 3.1 : Accuracy (as a %) obtained via the various clustering methods

Clustering method	spherical	k-means	k-means	k-means
	k-means	DTMM	MoP	LDA
spherical k-means	N/A	53.00	25.68	55.96
k-means DTMM	47.00	N/A	9.5	63.16
k-means MOP	74.32	90.50	N/A	92.60
k-means LDA	44.04	35.26	7.40	N/A

(a) Clustering methods using exactly the “correct” number of clusters

Dataset	spherical	k-means	k-means	k-means
	k-means	DTMM	MoP	LDA
spherical k-means	N/A	67.10	25.52	43.98
k-means DTMM	32.90	N/A	0	40.30
k-means MOP	74.48	100	N/A	83.36
k-means LDA	56.02	56.62	16.64	N/A

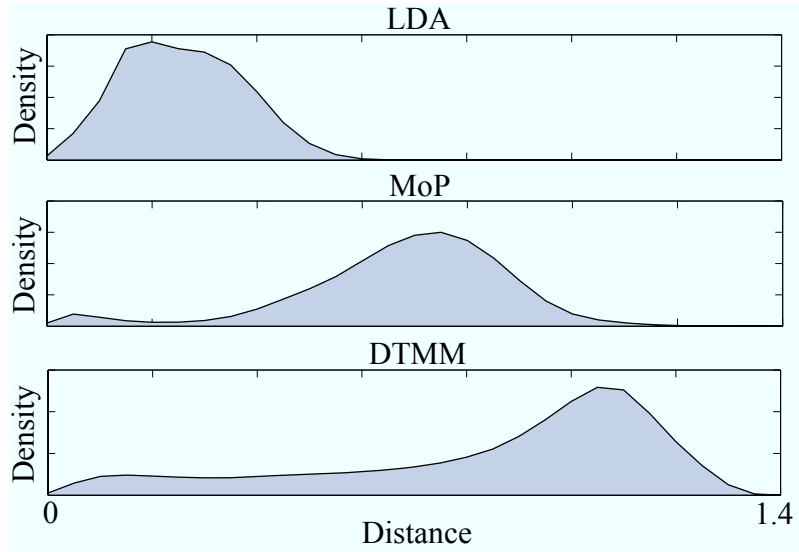
(b) Clustering methods using twice the “correct” number of clusters

Table 3.2 : Bootstrapped probability (as a %) that the method associated with the row would outperform on a randomly-selected data set.

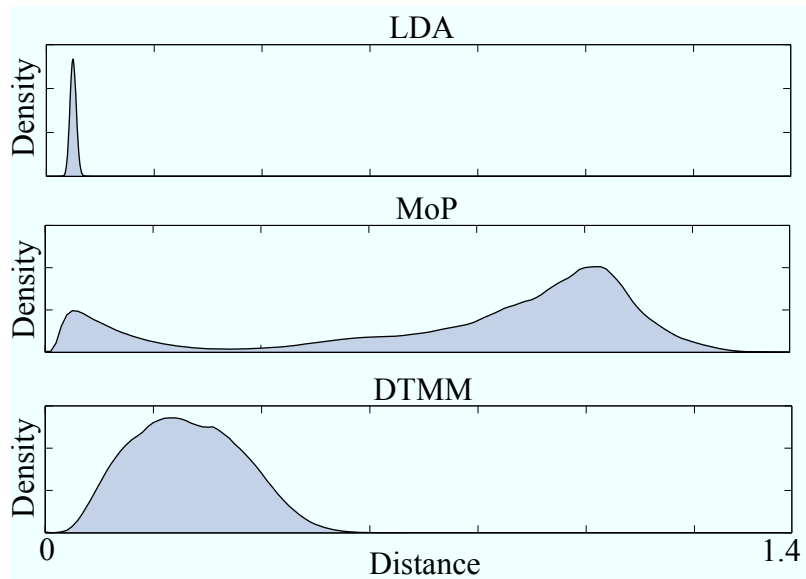
clusters.

But why does LDA underperform, and why are the topics generally poor in this sort of heterogeneous data set? At the beginning of the paper, I postulated that the Dirichlet prior could be problematic when using LDA as a pre-processing step for clustering data. In particular, I pointed out that a small-magnitude α parameter to that prior would result in topic proportion vectors that resided on the edge or corner of the simplex; that is, vectors which tended to be dominated by one or two topics. To examine whether this was a possible explanation for the poor performance, for each cluster, and for each model, I determined the top five topics (that is, the five topics most closely associated with the cluster). I then computed the average prevalence of those five topics in the documents that belonged to the cluster (see Table 3.3a and 3.3b). As expected, I found that the MoP model tended to produce topic proportion vectors that spread their probability across more topics than LDA did. For example, consider the `sci.med` data set. The fourth most common topic under the MoP model still has a 10% prevalence, whereas for LDA, the fourth most common topic had less than a 1% prevalence. One may conjecture that this forces the topics to attempt to model an entire class of documents; hence the inclusion of terms such as “subject re”, which has the effect of drastically reducing the intrinsic dimensionality of the feature space.

To further examine this, I computed (for both the 5 Newsgroups and Classic 3 data sets) the two-point correlation function over all of the topic proportion vectors for the LDA, MoP and DTMM models. The two-point correlation function is essentially the PDF for the distance between two points randomly selected from the data set. In a data set having a good clustering structure, I would expect at least two modes in the correlation function: a small one close to zero (for pairs of points in the same



(a) 5 Newsgroups dataset



(b) Classic 3 dataset

Figure 3.3 : Two point correlation functions for the three topic models.

Newsgroup	Topic appearance probability				
	#1	#2	#3	#4	#5
sci.crypt	0.295	0.128	0.122	0.119	0.117
sci.space	0.313	0.444	0.042	0.036	0.036
sci.med	0.346	0.218	0.157	0.102	0.012
religion.christian	0.555	0.322	0.025	0.015	0.007
sport.baseball	0.668	0.239	0.008	0.003	0.003

(a) MoP clusters

Newsgroup	Topic appearance probability				
	#1	#2	#3	#4	#5
sci.crypt	0.727	0.126	0.011	0.005	0.005
sci.space	0.486	0.241	0.044	0.037	0.031
sci.med	0.694	0.124	0.011	0.008	0.005
religion.christian	0.776	0.07	0.010	0.009	0.004
sport.baseball	0.741	0.085	0.013	0.012	0.006

(b) LDA clusters

Table 3.3 : Average appearance probability for the top five topics in each of the clusters for the 5 Newsgroups data set.

cluster) and a larger mode far from zero (for pairs of points in different clusters). Consider Figure 3.3a. This is exactly what I find in the MoP correlation function over the 5 Newsgroups data and in the DTMM function over that data. But the LDA correlation function is not nearly as high quality, showing a lack of clustering structure. The results are even more striking over the Classic 3 data (Figure 3.3b). This plot shows that the LDA points are almost all equi-distant from one another. However, the MoP model places the documents into a very nice clustering structure. The DTMM model over this data set does not have a clear clustering structure, but (unlike the LDA model) the documents are not all equi-distant. In fact, these results closely track what I find in Table 1: over the 5 Newsgroups data, the MoP and DTMM models have similar results, and the LDA model's results are not as good, but not poor. Over the Classic 3 data, the MoP results are excellent, and relatively speaking, the LDA results are poor, with the DTMM results in the middle.

To summarize: the MoP model seems to clearly produce the best clustering structure when used as a pre-processing step for k-means. This would seem to be the preferred approach. The DTMM model can perform well on certain data sets, but it is sometimes outperformed by a simple spherical k-means algorithm. I found that a simple LDA model is possibly not a good option for feature selection for unsupervised learning.

cluster	MoP	LDA
sci.crypt	phone, criminal, warrant, police, government, encryption, system, modem, bit, message, algorithm, program, security, serial number, nsa, data, secret, public key, attack, session key	subject re , key, write , government, clipper, encryption, chip, article , get , com , nsa, clipper chip, good , know , just , make , phone, people , need , algorithm
sci.space	earth, planet, orbit, moon, probe, atmosphere, surface, comet, mission, spacecraft, venus, object, image, satellite, space, mars, sun, solar system, first , gamma ray	subject re , edu , space, write , just , get , subject , article , know , launch, nasa, moon, go, thing , shuttle, orbit, think , earth, re , henry spencer
sci.med	doctor, cause, pain, drug, diet, vitamin, disease, day, body, patient, eat, treatment, level, help, physician, food, get , water, study, migraine	subject re , write , article , subject , get , edu , know , msg , pitt , edu , cause, doctor, just , c , com , food, disease, think , say , patient, pain
soc. religion. christian	god, say , jesus, christ, christian, believe, life, christians, love, people, bible, man, come, faith, sin, hell, truth, christianity, church, lord	god, say , people, write , know , believe, think, subject re , jesus, see , church, christian, make , come , just , christ, bible, time, christians life
rec. sport. baseball	game, team, player, win, hit, year, get , baseball, subject re , play, good, pitch, run, pitcher, edu , go , write , think , first, come	subject re , game, write , get , player, team, baseball, edu , hit, think , article , say , win, good , play, pitch, go , year, pitcher, subject

Table 3.4 : Most frequent words in the most important topic for each of the learned clusters in the 5 Newsgroups data set. Words that are seemingly irrelevant are given in **bold**.

Chapter 4

Learning to Evaluate Student Programming Assignments in a Massive Open Online Course

4.1 Problem Definition

This part of my work considers the problem of developing a scalable infrastructure capable of assigning grades to student assignments in an “Interactive Game Programming 101” MOOC.* Since the interactive programming MOOC motivating my work was first offered, more than 350,000 students have signed up, and nearly 30,000 have completed the course. This scale means that no human (or team of teaching assistants) can grade student submissions. Scale is a problem faced by many MOOCs, but a course covering interactive game programming presents a unique set of grading challenges.

Why is automated scoring of interactive games difficult? In most programming classes, grading of student programs is accomplished via automated execution of a test suite. This is not applicable for student-developed games, since a tester must generally “play” the game to determine correctness, and this is not easy for a test program to do automatically. Let’s consider a variant of the classical Asteroids game, where the player moves a space ship through a field of asteroids. The player dies when the space ship collides with an asteroid. The goal of the game place is to shoot

*“MOOC” stands for massive open online course [100]; an Internet-based MOOC may have tens of thousands of participants.

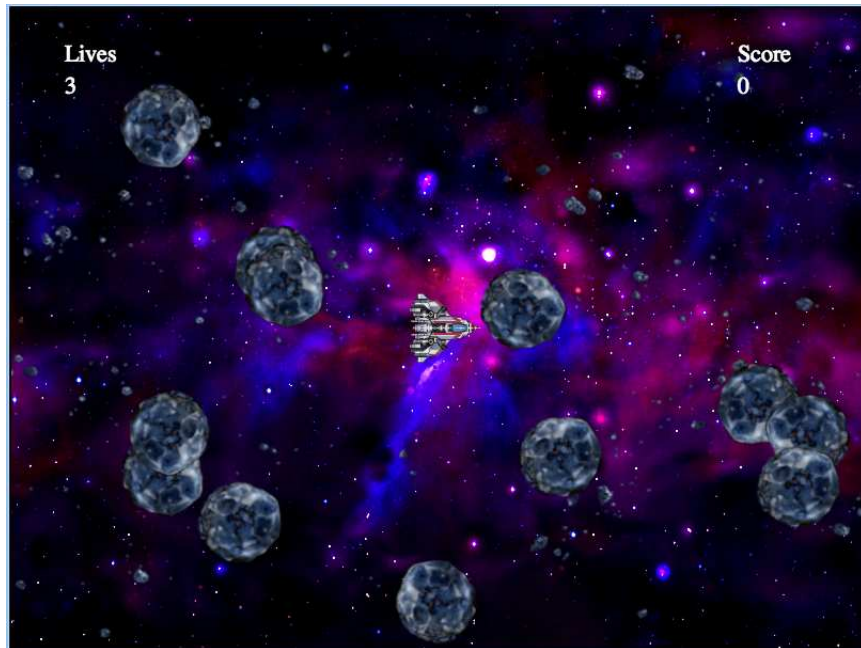


Figure 4.1 : Screen shot of the Asteroids game implemented as part of the interactive programming MOOC.

missiles at the asteroids, which breaks them up into smaller and smaller pieces, until all of the asteroids are destroyed. Hitting an asteroid with a missile earns points. A screen shot of a student-produced Asteroids game is shown in Figure 4.1.

A student's score on his or her program is based off of a number of rubric items. There are thirteen in all, including:

1. 1 point: The program spawns multiple rocks.
2. 1 point: The number of lives decreases by one when the ship collides with a rock.
3. 1 point: The program correctly determines whether a missile and a rock collide.
4. 1 point: The score is updated appropriately after missile/rock collisions.

Let's imagine that we wanted to develop a scalable testing infrastructure that attempted to automatically score these items. This would be exceedingly difficult. One difficulty would be the fact that correctness is defined in terms of the graphical and audio output. We might be able to handle this by instrumenting the code, collecting calls to the screen-drawing API at runtime, analyzing those. But any automated scoring program is somehow going to have to “play” the game. It will have to steer the space ship towards a rock to make sure that it is destroyed. It will have to aim the ship's missiles at a rock and shoot the rock in order to see the effect. Further, it would have to do all of this by (at least in a simulated fashion) manipulating the keyboard, just like a game player would. Developing a testing code that could accomplish all of this is not impossible, but it would require a huge engineering effort.

Peer grading. The standard way to deal with such difficulties in the online education world is eschew an automated approach, and to crowd-source the grading task via a mechanism called *peer grading* [101]. The MOOC software platform collects the submitted assignments, and distributed them to students in the course, so that each student is tasked not only with completing the assignment, but also grading other students' submitted assignments according to the instructor-supplied rubric.

With a high-quality peer grading infrastructure in place, there is no need to rely on an automated software to assign the actual grades to assignments; the peers can be used to do that. However, peer grading is not a panacea, and we have found that it does not obviate the need for automated scoring entirely—in fact, automated scoring could be used to make the peer grading process work better, which motivates the work described in this chapter.

For example, two ways in which we could use automated scoring to aide the peer grading process are:

1. **As an automatic gating function.** A problem observed in Rice MOOC's experience is that since the “price” of being able to grade other students assignments is turning in one's own assignment, some students will turn in an assignment that they know is incomplete, simply to be able to view and grade other student's assignments. While the motivation for doing this is a bit unclear (perhaps it is curiosity, or perhaps the motivation is to harvest a set of solutions) it is clear that a student should not be able to view others' work without first completing (or at least making a reasonable attempt to complete) the assignment. If one could automatically recognize with high accuracy those submission that are very low quality, it would make possible to automatically bar such students from taking part in the grading.

2. **Allocation of graders.** Peer graders need to be allocated carefully. There is a negative correlation between the quality of a student-produced program and the accuracy of a peer-assigned grade, implying that poor submissions need more graders to receive an accurate grade.[†] There are also qualitative reasons to avoid assigning the same number of peer graders to every assignment. Rather than assigning many graders to give a high-quality submission a perfect score, it would be better to move some of those graders to a poor assignment to help an under-performing student with constructive comments and criticism. Also, to goal is to make sure that the strongest graders grade the poorest assignments (so they can be of help), and for the weakest students to grade the best assignments, so they can see examples of what an excellent submission looks like.

[†]I have found that, in practice, the correlation between the average score on an interactive programming assignment and the standard deviation of the scores assigned by peer graders is -0.4 .

It is worth noting that unlike the scores that are used to determine whether the students actually pass the class, automated grading here is used to make the peer grading process work more smoothly. This means that we can tolerate a lot more error in the automated scores than if we used automated scoring by itself, without peer grading. Simply being able to categorize programs into “good” and “bad” submissions with reasonable accuracy would be very useful. This suggests that a statistical approach might be appropriate.

Proposed Approach. In this paper, I consider a statistical approach to recognizing good and bad interactive programs. Our goal is to utilize the fact that a MOOC is a data-rich environment. After an online course has been run one time, for each assignment, one will typically have access to thousands of (program, set of observed scores) pairs. It seems that it should be possible to learn how to automatically score an assignment from such a data set.

To that end, the approach that I propose is to view each program as being composed of a set of program fragments. A set of semantic and syntactic distance measures have been developed that can compare two program fragments to determine how similar they are. The resulting similarity measure can be used to compare the similarity of two student-submitted programs. Intuitively, if the various metrics are carefully chosen, then one might expect that two programs that are judged to be similar by the set of metrics will tend to have the same overall score. If this is the case, then the similarity measure can be used in conjunction with a historical database of graded, student-submitted programs to score new programs.

My method fundamentally relies upon the definition of a set of metrics that can be used to decide whether two program fragments are similar. The set of metrics that I utilize are defined in the next section. Then, in Section 4.3 of this chapter, I

describe how a couple of very simple k NN-based regression model can be constructed that utilizes the set of metrics. Section 4.4 describes a more sophisticated, Bayesian model that also utilizes the set of distance metrics. The method is based on the idea of choosing a set of prototype program fragments from the training set, where each prototype fragment is assigned a score that marks the fragment as being an exemplar of either high or low quality code. The level of similarity observed between the fragments in a test program and those prototype program fragments can then be used to score the program. In Section 4.7 of the chapter, I experimentally evaluate these various scoring methods (including an ensemble method that combines multiple individual scores) using a set of four programming assignments generated from running a popular interactive programming MOOC.

4.2 Distance Measures

In this section, I present the set of distance measures D on which my statistical techniques rely. Each $d_i \in D$ is a symmetric function accepting two program fragments, returning a non-negative value that measures the similarity between the two program fragments. The distance between a fragment and itself should be zero, and the distance should not decrease when the two fragments diverge.

4.2.1 Programs and Event Handlers

In my model, distances are defined between the *program fragments* that constitute a student-written program. In general, these fragments can be procedures, blocks, or class definitions. The specific dataset I use consists of event-driven game programs written in Python; here, the natural fragments to compare are *event handlers* that define the actions to be undertaken when a user interacts with the game. The event

handler may react to an event such as a mouse move, mouse click, timer expiration, etc.

A skeleton of a sample program from our dataset is shown in Figure 4.2. This program, created for the 7th programming project in the course, implements the *Asteroids* game described in Section 4.1. The program has several event handlers, for example `keydown`, which handles the pressing of a key, and `keyup`, which handles the release of a key. The program also has a `draw` handler that renders the objects on the game on every clock tick, and a timer `rock_spawner` that periodically creates new rocks. The program calls routines in a GUI programming library (`simplegui`) that the course asks the students to use.

It is worth noting that any reasonable implementation of *Asteroids* would contain event handlers to move the spaceship and fire missiles at rocks. Moreover, in this class, all programs are developed from a template that the instructors provide to the students; therefore, we would expect some level of syntactic similarity between programs. However, different student submissions can and do vary in implementation details. For instance, the color and sound scheme and even equations of motion for the spaceship, missiles and rocks may differ in different implementations. On the other hand, two programs may use the same dynamical equations and the same logic of program design — and thus be the *same* program for a reasonable definition of “same” — while differing in superficial details such as names of variables or the order in which logically independent program statements appear in the code.

4.2.2 Distance Measures

The developed distance measures over program fragments (event handlers) aim to abstract out the low-level, syntactic aspects of code, in particular the variable and

```

import simplegui                                keys += 1
...                                              my_ship.turn("left")
class Ship:                                     if key == simplegui.KEY_MAP["right"]:
    ...                                         keys += 1
    def thrusty(self, direction):              my_ship.turn("right")
        if direction == "stop":                if key == simplegui.KEY_MAP["up"]:
            my_ship.thrusters(False)           keys += 1
        elif direction == "go":                my_ship.thrusty("go")
            my_ship.thrusters(True)
    ...
def draw(canvas):
    ...
    time += 1
    ...
    canvas.draw_image(nebula_image, ...)
    canvas.draw_image(debris_image, ...)
    canvas.draw_image(debris_image, ...)
    ...
    def rock_spawner(): ...
    timer = simplegui.create_timer(...)
def keydown(key):
    global keys
    if key == simplegui.KEY_MAP["left"]:
        timer.start()
        frame.start()

```

Figure 4.2 : Program # 57239, Project *Asteroids*

function names that can be changed without altering the module's semantics. The measures distinguish between code fragments on basis of their use of the names whose

semantics are invariant across programs — namely, library functions and constants — as well as patterns of control and data flow within them.

Before the distances are computed, some preprocessing of the event handlers needs to be performed, where the code of auxiliary functions called from the event handlers are inlined into the handler. For instance, in 4.2, the code for the function `my_ship.thrusty` is inlined into the `keydown` handler. The distance measures now operate as follows:

d_1 : Type of code fragment The first of our distance measures (called d_1 from now on) abstracts each event handler by its *type*. In this setting, it is natural to define the type of an event handler is the set of events that it listens to. For instance, the `keydown` handler in 4.2 listens to keyboard presses, and the `keyup` handler listens for releases to pressed keys. Now, let S_1 and S_2 be the sets of events to which the event handlers H_1 and H_2 listen. Then $d_1(H_1, H_2)$ is 1 if $S_1 = S_2$ and 0 otherwise.

d_2 : Bag of library calls The distance measure d_2 abstracts an event handler by the set of library functions that it calls. For example, the fragment of the `draw` handler shown in 4.2 would be abstracted by the set `{draw_image}`. The d_2 -distance between two handlers is the set edit distance between their abstractions defined in this way.

d_3 : Bag of guarded updates In the distance measure d_3 , an event handler is translated to a set of guarded statements of the form (g, s) , where g is a boolean expression and s is a sequence of assignments and function calls. Each s represents a sequence of imperative variable updates happening along a control flow path; g is the condition under which this path is executed. For instance, the code fragment “`if B : S_1 else S_2` ”, where S_1 and S_2 are assignments, is rewritten into (B, S_1) and $(\neg B, S_2)$. Next, the program-specific variable names are “erased” by replacing each

variable with a placeholder $\#$. The distance between two guarded updates is defined as the sum of the string edit distance between the guards and the string edit distance between the updates. The set edit distance between two sets of guarded updates is now well-defined. The d_3 -distance between two event handlers is the Hausdorff distance between the sets of guarded updates extracted from them.

d_4 : Abstract syntax tree The distance measure d_4 abstracts each event handler by an abstract syntax tree. Nodes in this tree represent statements that appear in the handler; each node is labeled with the following information: (a) The type of statement that it represents — i.e., whether it is an assignment or an if-then-else; (b) if the node is a library call, the name of the function called. There is an edge from node u to node v if the scope of v is nested within that of u . The d_4 -distance between two event handlers is now defined as the tree edit distance between their tree abstractions.

d_5 : Dependence graph over library functions In the next distance measure, called d_5 , an event handler is abstracted by a *dependence graph* [102]. To construct this dependence graph, first an intermediate graph G is considered where there is a node for each distinct variable used by the event handler, and each library function called by the handler. There is an edge from a node u to a node v if there is a data or one of the following holds: (a) v is a variable, and the program has an assignment $v := e$, where e is an expression in which u (or a call to u in case u is a library function) occurs; (b) v is a library function, and the program has a call to v where one of the arguments is an expression that references u ; (c) u is a variable that appears in a boolean expression b , such that b guards a block inside v is either updated (in case it is a variable) or called (in case it is a library function). It is noted that there is a path from a node u to node v if there is a causal dependency

between u and v — i.e., if data from u flows into v in some execution, or if the value of u determines, either directly or transitively, whether a call or update to v takes place. The graph G is constructed using a simple program analysis. The dependence graph is now constructed by deleting from G all variable nodes, and adding an edge between remaining (library function) nodes u and v if there is a path from u to v in G consisting exclusively of variable nodes.

The d_5 -distance between two event handlers is defined as the graph edit distance between their dependence graphs defined this way. An issue with the graph edit distance measure is that it is NP-hard to compute. Therefore, in practice, an approximation of graph distance defined in prior work [103] is used in place of exact graph edit distance.

d_6 : Dependence graph over library functions and variables The measure d_6 is just like d_5 , except the intermediate graph G used in the construction of d_5 is now used as the graph abstraction of the event handler.

I also considered an additional measure which views the source code as a natural language text; each program fragment is tokenized and a topic model is learnt for a corpus of code fragments. The distance measure reflects the Euclidean distance between the topic mixing proportion vectors of the fragments. This measure, however, does not improve the classification power of any of the developed models and is not discussed any further in the remaining part of this Chapter.

4.2.3 Example

Let’s consider the two programs in Figures 4.3 and 4.4. The handlers for the “key press” event have different names in the two programs (`keydown` in the former, `keydown_handler` in the latter); they also differ in the amount by which they change

```

class Ship:
    ...
    def decrement_ang_val(self):
        self.angle_vel -= 0.02
    def increment_ang_val(self):
        self.angle_vel += 0.02

def keydown(key):
    if key == simplegui.KEY_MAP["left"]:
        my_ship.decrement_ang_val()
    if key == simplegui.KEY_MAP["right"]:
        my_ship.increment_ang_val()
    if key == simplegui.KEY_MAP["up"]:
        my_ship.setThrust(True)
    if key == simplegui.KEY_MAP["space"]:
        my_ship.shoot()

```

Figure 4.3 : Handler for key press (**keydown**) in Program # 56080, project *Asteroids*

the velocity of the ship on a key press. However, aside from this, they are very similar. On the other hand, both handlers are quite different from the **keydown** handler in 4.2.

The developed distance measures recognize this distinction. The vector of distances between the two “key press” handlers according to metrics d_1 - d_6 is $\langle 0, 0, 40, 3, 5, 15 \rangle$. However, the distance between the key press handlers in the programs in 4.3 and 4.2 is the (larger) $\langle 0, 0, 66, 60, 1, 45 \rangle$.

```

class Ship:
    ...
    def increment_angle_vel(self):
        self.angle_vel += 0.1
    def decrement_angle_vel(self):
        self.angle_vel -= 0.1

def keydown_handler(key):
    if key == simplegui.KEY_MAP["left"]:
        my_ship.decrement_angle_vel()
    if key == simplegui.KEY_MAP["right"]:
        my_ship.increment_angle_vel()
    if key == simplegui.KEY_MAP["up"]:
        my_ship.set_thrust(True)
    if key == simplegui.KEY_MAP["space"]:
        my_ship.shoot()

```

Figure 4.4 : Handlers for key press (`keydown_handler`) in Program # 56601 in project *Asteroids*

4.3 A k NN Regression Model

For classification/regression problems where pairwise distances between data are meaningful and easy to compute, k NN-based methods often show excellent performance. Considering the simplicity of k NN classification/regression, this makes k NN an attractive option for this application.

Utilizing the bag of distance functions D from the previous section to build k NN model does require a bit of thought, since the metrics are most naturally computed

between fragments such as procedures or event handlers, whereas to perform k NN regression in our scenario, we need to be able to compute the distance between entire *programs*. It is the programs in the training set that are labeled with a score, and not the individual fragments.

It appears that there are two obvious ways to use the various metrics to compute a distance between programs, which I discuss now.

4.3.1 The Lift-Then-Compute Distance

In the first method, called the Lift-Then-Compute distance, the fragment-to-fragment distances are first “lifted” independently to the program level, giving one program-to-program distance for each metric in the set D . Then the l_2 norm of the resulting vector is computed to give the distance between two programs.

To “lift” a fragment-to-fragment distance $d \in D$ to the program level and compute the distance between a test program x_1 and a program x_2 in the training set, a maximal bipartite matching between the fragments in the test program and the labeled program will be used. To do this, first a bipartite graph is created, whose vertices on the left side correspond to the fragments in x_1 , and whose vertices on the right side correspond to the fragments in x_2 . Let x_{1j} denote the j th fragment in x_1 and let x_{2k} denote the k th fragment in x_2 . An edge from fragment x_{1j} to fragment x_{2k} is labeled with the value

$$max_d - d(x_{1j}, x_{2k}).$$

Here, max_d is the maximal value of $d(\cdot)$, computed by comparing the fragments in the test program with all of the fragments in the entire labeled training data set. In this way, edges between two highly similar program fragments are labeled with a large value, and those that are quite different labeled with a small value.

Let E denote the set of edges in the maximal matching computed over the resulting graph. Then the “lifted” value of d computed over the two programs is defined as:

$$lift(d, x_1, x_2) = \sum_{(x_{1j}, x_{2k}) \in E} d(x_{1j}, x_{2k}).$$

To compute the actual distance between x_1 and x_2 , the following is computed:

$$dist(x_1, x_2) = \left(\sum_{d \in D} lift(d, x_1, x_2)^2 \right)^{\frac{1}{2}}$$

This distance can then be used as the basis of a k NN regression model.

One natural concern is that the various distance metrics in D could operate on very different scales. Thus, in practice the lifted distances are normalized so that they all fall in the range zero to one.

4.3.2 The Compute-Then-Lift Distance

In the Lift-Then-Compute distance described above, first each individual metric is lifted via a bipartite matching so that it can be used as a program-to-program metric. Then the distance was computed. As an alternative, all of the metrics can be used at once to compute the distance between each fragment-fragment pair, and then perform a single bipartite matching between the two programs to compute the overall distance. Here, the method is called the Compute-Then-Lift distance.

Specifically, we first a bipartite graph is built where the edge (x_{1j}, x_{2k}) is labeled with the value

$$\left(\sum_{d \in D} (max_d - d(x_{1j}, x_{2k}))^2 \right)^{\frac{1}{2}}.$$

Here, max_d is defined as before.

Let E denote the set of edges in the maximal matching computed over the resulting graph. Then the Compute-Then-Lift distance distance from the test program to the

labeled training program is simply defined as

$$dist(x_1, x_2) = \frac{1}{|E|} \sum_{(x_{1j}, x_{2k}) \in E} ed(x_{1j}, x_{2k})$$

where $dist(x, x')$ is the Euclidean distance between two program fragments:

$$ed(x_{1j}, x_{2k}) = \left(\sum_{d \in D} d(x_{1j}, x_{2k})^2 \right)^{\frac{1}{2}}$$

4.4 A Prototype-Based Model

4.4.1 Overview

In this section, I describe a somewhat more sophisticated, Bayesian model that provides a simple alternative to k NN regression. The model makes use of the set of *prototype fragments* to predict the score of a test program. At a high level, I define a stochastic, generative process that assumes that a peer grader grades a program by seeing how similar the fragments in the program are to a special set of prototype fragments. Some of the prototype fragments are known to be good, and some are known to be bad. Based upon this goodness (or badness) each prototype fragment is assigned a score. These scores are used to score the individual fragments in the test program: a fragment in the test program is assigned a score that is the weighted average of the scores of the similar prototype fragments (the closest fragments have the highest weight). The score of the program as a whole is then the average of the scores of the individual fragments within it.

4.4.2 Generative Process in Detail

The stochastic, generative process underlying the proposed model is as follows. Given a set of programs denoted by X , one can imagine that a peer grader uses the following process to generate the set of scores Y for those programs:

1. First, the programs are deterministically broken into a set of N program fragments, according to the program structure. I use x_{ij} to denote the j th fragment of the i th program in X .
2. Second, the peer grader randomly selects K of the fragments (without replacement) to serve as *prototype fragments*. Let p_k denote the k th prototype fragment.
3. Next, each of the K prototype fragments is issued a score $s_k \sim \text{Normal}(\mu_s, \sigma_s^2)$. μ_s and σ_s^2 are known hyperparameters.
4. A grading variance σ_g^2 is then generated with a $\text{InvGamma}(1, 1)$ prior.

5. Finally, I stochastically generate the set of scores Y for the programs in X .
 To grade program i , one can imagine that the grader compares each of the fragments in the program with each of the prototype fragments, and determines how similar the fragments are to the prototypes. A prototype fragment that is similar to a fragment in program i is going to contribute strongly to the grade for program i . Specifically, let w_{ijk} denote the extent to which prototype fragment k is found to be similar to program fragment x_{ij} . The weight w_{ijk} is then computed as:

$$w_{ijk} = \frac{\exp(\text{dist}(x_{ij}, p_k))}{\sum_{k'} \exp(\text{dist}(x_{ij}, p_{k'}))} \quad (4.1)$$

Here, $\text{dist}(x, p)$ is the Euclidean distance between the two program fragments, defined exactly as in the previous section. Then, assuming that there are n_i fragments in program i , it is given the numerical grade:

$$y_i \sim \text{Normal}\left(\frac{1}{n_i} \sum_{jk} w_{ijk} s_k, \sigma_g^2\right)$$

4.5 Some Additional Wrinkles

The process can be made more realistic by adding a few additional twists, as I describe now.

4.5.1 Weighting the Distance Measures

There are two key problems with the generative process as described, with respect to the distance computation induced by the set of distance measures D .

First, I would like the user to be free to add as many distance functions to D as he or she desires, safe with the knowledge that an additional distance function cannot hurt—even if it is worthless—it can only help.

Second, even if the metrics in D are all useful, they may function at different scales.

Both problems can be easily addressed by associating a weight with each $d \in D$. Specifically, let α_d denote the weight given to distance measure d . Then we modify $dist(x, p)$ so that it is computed as:

$$dist(x, p) = \left(\sum_{d \in D} \alpha_d d(x, p)^2 \right)^{\frac{1}{2}}$$

An appropriate prior for α_d is a $\text{InvGamma}(1, 1)$ distribution.

4.5.2 Weighting the Prototype Fragments

Another potential drawback with the original generative process is that it assumes that all prototype fragments are of the same importance in scoring a student's program. In fact, it stands to reason that certain fragments are of greater importance. For example, it may be that a certain bug can render a program unusable. If a prototype fragment contains a canonical example of this bug, then any program fragment

that is close to that fragment should receive a poor score, regardless of the other prototype fragments that the program fragment is close to.

It can be allowed for this by also providing for a simple weighting scheme for the prototype fragments. Specifically, I modify the definition of w_{ijk} in Equation 4.1 so that it include a prototype-specific weight β_k :

$$w_{ijk} = \frac{\beta_k \exp(\text{dist}(x_{ij}, p_k))}{\sum_{k'} \beta'_k \exp(\text{dist}(x_{ij}, p_{k'}))}$$

Here, an appropriate prior distribution on β_k is a Laplace(1) distribution, folded so that it produces only positive values. I choose the Laplace distribution since it is the standard regularizing prior, used to perform Lasso-type feature selection [104].

4.5.3 A Background Prototype Fragment

A final, obvious drawback with the simple model described thus far is that since the approach learns only from the training data included in X and Y , it might have problems grading a program that contains a fragment that is unlike any x_{ij} in the training data.

One can address this problem as well by designating prototype fragment 1 to be a special, “background” prototype fragment. Its distance from all other fragments is defined to be zero, so that Equation 4.1 becomes:

$$w_{ijk} = \frac{\begin{cases} \beta_1 & \text{if } k = 1 \\ \beta_k \exp(\text{dist}(x_{ij}, p_k)) & \text{otherwise} \end{cases}}{\beta_1 + \sum_{k' > 1} \beta_{k'} \exp(\text{dist}(x_{ij}, p_{k'}))}$$

This effectively means that as a fragment x moves further from any of the prototype fragments, the score s_1 becomes more and more important when computing x ’s

contribution to the program that is being graded. In this way, we are somewhat protected from outliers, in the sense that a fragment that is totally dis-similar from all of the non-background prototype fragments will be assigned an appropriate background score that is learned from the training data, rather than being associated with the arbitrary prototype fragment that happens to be closest to it.

4.5.4 Final PDF

With these updates, the prior on the set of model parameters Θ (which consists of the various α_d , β_k , s_k , and p_k values) is:

$$F(\Theta) = \frac{1}{Z} \text{InvGamma}(\sigma_g^2 | 1, 1) \prod_k \text{Normal}(s_k | \mu_s, \sigma_s^2) \times \\ \prod_{d \in D} \text{InvGamma}(\alpha_d | 1, 1) \times \\ \prod_k \text{FoldedLaplace}(\beta_k | 1)$$

Here, Z is a constant normalization term, assigning a uniform probability of $(K! \binom{N}{K})^{-1}$ to each possible set of prototypes chosen. The likelihood of the observed data is:

$$F(Y|X, \Theta) = \prod_i \text{Normal}(y_i | \frac{1}{n_i} \sum_{jk} w_{ijk} s_k, \sigma_g^2)$$

Given this model, there are many potential ways to learn the parameters in Θ . I utilize a fairly simple Gibbs sampler for this purpose which is described in the section below.

4.6 Learning the Prototype-Based Model

4.6.1 MCMC Basics

Now that the generative process and associated PDF are defined, samples need to be drawn from $F(Y, Y' | \Theta, X)F(\Theta)$. I will do this using a Gibbs sampler [105]. Gibbs sampling is a technique for generating samples from a high-dimensional probability distribution function. A Gibbs sampler simulates a Markov chain whose stationary distribution is the desired target distribution. The procedure works as follows. Assume that $X = (X_1, X_2, \dots, X_d)$ is a d -dimensional vector of continuous and/or discrete random variables. A key requirement of the Gibbs sampler is that we can efficiently generate samples from the conditional distributions $f(X_j | X_{-j})$ for $j \in \{1..d\}$, where X_{-j} is the vector comprising all variables except X_j . To generate k samples the Gibbs sampler proceeds as follows:

- (1) Select an initial value $X^{(0)} = (X_1^{(0)}, X_2^{(0)}, \dots, X_d^{(0)})$.
- (2) For each sample $i = \{1 \dots k\}$, sample each variable $X_j^{(i)}$ from $f(X_j^{(i)} | X_1^{(i)}, \dots, X_{j-1}^{(i)}, X_{j+1}^{(i-1)}, \dots, X_d^{(i-1)})$.

After running the simulation for k steps of a “burn in” period, the current state of the chain can be taken as a sample from the target distribution.

Running a Gibbs sampler then requires to derive the conditional, posterior distribution for each of the values of interest. In my case, the values of interest contain (1) all of the y'_{ij} values, indicating whether the j th fragment in program i is chosen as a prototype fragment; (2) the s_k values associated scores with those prototypes; (3) the $\alpha(d)$ values weighting the various distance measures; and (4) the β_k values weighting the various prototype fragments.

4.6.2 Updating the Indicators

Let's begin by considering the update of each y'_{ij} . As mentioned above, y'_{ij} takes the value k if the j th fragment in program i is chosen as the k th prototype fragment; a value of 0 indicates that this fragment has not been chosen as a prototype, and the value 1 is not allowed, since prototype 1 is the special background fragment. Using Equation ??, it can be written:

$$F(y'_{ij} | \{y'_{i'j'} | i \neq i' \vee j \neq j'\}, \{s_k\}, \{\alpha(d)\}, \{\beta_k\}, \{y'_{i'j'}\}) \propto \prod_{i'} \text{Normal}(y_{i'} | \frac{1}{n_{i'}} \sum_{j'k} w_{i'j'k}(Y') s_k, \sigma_g^2)$$

On the surface of it, our task then seems easy. I simply compute $F(y'_{ij} | \cdot)$ for all possible values of y'_{ij} , normalize those values, and then choose one of the k values at random, with probability proportional to the normalized value.

There is, however, a significant problem with this simple method for updating y'_{ij} . Specifically, it ignores the fact that $F(y'_{ij} | \cdot)$ takes the value zero if the various y'_{ij} are inconsistent—in a correct assignment, only one y'_{ij} can take the value k , for $k \in \{2 \dots K\}$; otherwise, if two y'_{ij} 's take the value k , then the PDF cannot be evaluated. This is problematic because it means that if we choose to update a single y'_{ij} at a time, the MCMC algorithm can never progress, because there will be only one consistent assignment for y'_{ij} (specifically, the value that y'_{ij} took before the update).

What all of this means is that only one y'_{ij} value at a time cannot be updated during MCMC sampling. Instead, at each iteration of the Gibbs sampler, the following has to be done many times. I randomly choose two program fragments, referred to by (i, j) and (i', j') , respectively, subject to the constraint that either (a) $y'_{i,j}$ must be non-zero, or (b) $y'_{i',j'}$ must be non-zero, or (c) both must be non-zero—in other words, I make sure that at least one of the program fragments that we selected is currently

being used as a prototype fragment. Let $y'_{ij} = k_1$ and $y'_{i'j'} = k_2$ before the update.

Then let:

$$p_1 = \prod_a \text{Normal}(y_a | \frac{1}{n_a} \sum_{bk} w_{abk}(Y') s_k, \sigma_g^2)$$

and let $p_2 =$

$$\prod_a \text{Normal}(y_a | \frac{1}{n_a} \sum_{bk} \left(Y' \text{ with } \begin{cases} y'_{ij} = k_2 \text{ and} \\ y'_{i'k'} = k_1 \end{cases} \right) s_k, \sigma_g^2)$$

p_1 is intuitively the “goodness” of the current assignment, and p_2 is the “goodness” of the assignment that swaps the values of y'_{ij} and $y'_{i'j'}$. To evaluate p_2 , we simply re-evaluate $F(Y, Y' | X, \Theta)$ with an updated version of Y' where we have swapped the values of y'_{ij} and $y'_{i'j'}$. Then, with probability of $\frac{p_2}{p_1 + p_2}$ we decide to accept this swap; otherwise, we reject it. In this way, the identities of the prototype fragments are updated.

4.6.3 Updating the Scores

The next update is the update of various score values, where:

$$\begin{aligned} F(s_k | \{y'_{ij}\}, \{s_{k'} | k' \neq k\}, \{\alpha(d)\}, \{\beta_{k'}\}, \{y_{ij}\}) \propto \\ \text{Normal}(s_k | \mu_s, \sigma_s^2) \times \\ \prod_i \text{Normal}(y_i | \frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'}, \sigma_g^2) \end{aligned}$$

While the above equation looks similar to the equation used for updating the indicators telling us which program fragments were in fact prototype fragments, there is a key difference: s_j is not discrete. Fortunately, with a bit of math it can be shown that $F(s_k | \cdot)$ is in fact a normal distribution, which is very easy to sample from. To

begin, the equation can be re-written as:

$$\begin{aligned}
F(s_k|.) &\propto \frac{1}{\sqrt{2\pi\sigma_s^2}} e^{-\frac{(\mu_s - s_k)^2}{2\sigma_s^2}} \times \\
&\prod_i \frac{1}{\sqrt{2\pi\sigma_g^2}} e^{-\frac{(y_i - \frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'})^2}{2\sigma_g^2}} \\
&\propto e^{-\frac{(\mu_s - s_k)^2}{2\sigma_s^2}} \prod_i e^{-\frac{(y_i - \frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'})^2}{2\sigma_g^2}} \\
&= e^{-\frac{(\mu_s - s_k)^2}{2\sigma_s^2} - \sum_i \frac{(y_i - \frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'})^2}{2\sigma_g^2}}
\end{aligned}$$

Since every normal distribution function is itself an exponentiated quadratic function (and vice versa: every exponentiated quadratic function is proportional to some normal distribution function) it follows that if the exponent in the above can be written as a quadratic function of s_k , then the normal, marginal posterior for s_k will be derived. Working on the exponent, it can be re-written as:

$$\begin{aligned}
&-\frac{(\mu_s - s_k)^2}{2\sigma_s^2} - \sum_i \frac{(y_i - \frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'})^2}{2\sigma_g^2} \\
&= \frac{2s_k\mu_s - s_k^2}{2\sigma_s^2} + \\
&\sum_i \frac{\frac{2y_i}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'} - (\frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'})^2}{2\sigma_g^2} \\
&= \frac{2s_k\mu_s - s_k^2}{2\sigma_s^2} + \sum_{ij} \frac{\frac{2y_i}{n_i} w_{ijk}(Y') s_k}{2\sigma_g^2} - \\
&\sum_i \frac{\left(\frac{1}{n_i} \sum_{jk'} w_{ijk'}(Y') s_{k'}\right)^2}{2\sigma_g^2} + c_1
\end{aligned}$$

Here, c_1 is a constant that does not depend upon s_k :

$$\begin{aligned}
&= \frac{2s_k\mu_s - s_k^2}{2\sigma_s^2} + \sum_{ij} \frac{\frac{2y_i}{n_i} w_{ijk}(Y') s_k}{2\sigma_g^2} - \\
&\quad \sum_i \frac{\frac{1}{n_i^2} \sum_{jk'} w_{ijk}(Y') w_{ijk'}(Y') s_k s_{k'}}{2\sigma_g^2} + c_2 \\
&= \frac{2s_k\mu_s - s_k^2}{2\sigma_s^2} + \sum_{ij} \frac{\frac{2y_i}{n_i} w_{ijk}(Y') s_k}{2\sigma_g^2} - \\
&\quad \sum_{ij} \frac{\sum_{k'} \frac{1}{n_i^2} w_{ijk}(Y') w_{ijk'}(Y') s_k s_{k'}}{2\sigma_g^2} + c_2 \\
&= as_k + bs_k^2 + c_2
\end{aligned}$$

where:

$$\begin{aligned}
a &= \frac{\mu_s}{\sigma_s^2} + \sum_{ij} \frac{\frac{2y_i}{n_i} w_{ijk}(Y') - \sum_{k' \neq k} \frac{1}{n_i^2} w_{ijk}(Y') w_{ijk'}(Y') s_{k'}}{2\sigma_g^2} \\
b &= \frac{-1}{\sigma_s^2} + \sum_{ij} \frac{w_{ijk}^2(Y')}{2n_i^2 \sigma_g^2}
\end{aligned}$$

Then, since $F(s_k|\cdot)$ is an exponentiated quadratic function of the given form, we have:

$$F(s_k|\cdot) = \text{Normal}\left(\frac{-b}{a}, \frac{-1}{2a}\right)$$

where a and b are as defined above. Thus, updating s_k is as simple as computing a and b and then sampling from the resulting normal distribution.

4.6.4 Updating Other Parameters

There are two remaining parameter types that must be updated during sampling: (1) the $\alpha(d)$ weights for the distance measures; and (2) the β_k weights for the prototype fragments.

Unfortunately, each of those classes of variables must be updated using a *rejection sampler* [106] rather than sampling from the desired marginal posterior distributions

directly, because those marginal posteriors do not come from a standard family that we can readily sample from (such as the normal distribution in the case of the marginal posterior for the various score values).

Briefly, rejection sampling is a general-purpose technique used to sample $x \sim g(x)$, where g is an arbitrary function. Rejection sampling works by computing an “envelope” function f that fully encloses the function to be sampled from (so $g(x) > f(x)$ for all x in the domain of f). Then, random points are chosen at uniformly from the portion of the (x, y) plane that is under the function f (that is, we choose uniformly from (x, y) where x is in the domain of f and $y \leq f(x)$). The first time that $f(x) > y$ for a randomly-selected (x, y) pair, x is accepted and returned as $x \sim g(\cdot)$.

Rejection sampling is typically slower than using techniques that sample from a known distribution (such as the Gamma or Inverse Gamma distributions), since rejection sampling can require evaluating $g(x)$ many times to both compute the envelope function $f(\cdot)$, and to test whether $f(x) > y$ at a particular (x, y) . But its strength is that it is a general-purpose mechanism that does not require that the function to be sampled from come from a known distribution family.

The update distribution of β_k is:

$$\begin{aligned}
F(\beta_k|\cdot) &\propto \prod_i \text{Normal}(y_i | \frac{1}{n_i} \sum_{jk} w_{ijk'}(Y') s_{k'}, \sigma_g^2) \times \\
&\quad \text{FoldedLaplace}(\beta_k | 1) \\
&= \prod_i \text{Normal}(y_i | \frac{s_k}{n_i} \sum_{jk} w_{ijk'}(Y') s_{k'}, \sigma_g^2) \times \\
&\quad \text{FoldedLaplace}(\beta_k | 1) \\
&\propto \exp \left(-\beta_k - \sum_{ijk'} \frac{(y_i - \frac{s_k}{n_i} w_{ijk'}(Y'))^2}{2\sigma_g^2} \right) \\
&= \exp \left(\sum_{ijk'} \frac{y_i s_k w_{ijk'}(Y')}{n_i \sigma_g^2} - \sum_{ijk'} \frac{s_k^2}{n_i^2 \sigma_g^2} w_{ijk'}^2(Y') - \beta_k \right)
\end{aligned}$$

This function can be evaluated easily during rejection sampling, though it may be a bit expensive to evaluate repeatedly. Since for each program fragment and for each prototype fragment, $w_{ijk'}(Y')$ must be evaluated, which itself requires $O(K)$ time, we have an $O(K^2N)$ overall evaluation time for $F(\beta_k|\cdot)$ at a particular value of β_k . However, this can be alleviated somewhat by a careful implementation. For a given β_k , the first time $w_{ijk'}(Y')$ is evaluated for a given β_k , we can compute $a_{ijk'}$ and $b_{ijk'}$ so that:

$$w_{ijk'}(Y') = \frac{\beta_k a_{ijk'}}{b_{ijk'} + \beta_k a_{ijk'}}$$

where both $a_{ijk'}$ and $b_{ijk'}$ have constant values with respect to β_k . These values can be re-used when evaluating $F(\beta_k|\cdot)$ at a different value of β_k . This has the benefit of reducing the complexity of subsequent evaluations of $F(\beta_k|\cdot)$ during rejection sampling to $O(KN)$.

Because $\alpha(d)$ (the weighting on distance measure d) is also used to compute the various $w_{ijk'}(Y')$ terms, its marginal posterior looks almost identical to that for

$F(\beta_k|.)$, the only difference coming from the prior:

$$F(\alpha(d)|.) \propto \exp\left(\sum_{ijk'} \frac{y_i s_k w_{ijk'}(Y')}{n_i \sigma_g^2} - \sum_{ijk'} \frac{s_k^2}{n_i^2 \sigma_g^2} w_{ijk'}^2(Y')\right) \times \\ \text{InvGamma}(\alpha(d)|1, 1)$$

Unfortunately, however, the same trick cannot be used to reduce the evaluation time of $F(\alpha(d))$ down to $O(KN)$, because $\alpha(d)$ is used in each and every evaluation of the function $\text{dist}(.)$. Hence, none of the terms are constant with respect to $\alpha(d)$. Fortunately, the number of distinct distance measures is typically going to be much smaller than K (the number of prototype fragments) and so the costly evaluation is not as problematic.

4.7 Experimental Study

4.7.1 Experimental Setup

To evaluate the proposed scoring methods, I chose to study the last six projects completed by students as they participated in a popular interactive programming MOOC.

Programming assignments considered. The goal in each project is to design an interactive video game in Python. In the first (easiest) project that I consider, the goal is to implement Stopwatch, a simple game where players attempt to stop a stopwatch exactly on the second. In the second project, students implement Pong, a classic simple tennis-like game. In the next, students implement Memory, a card game where players aim to turn pairs of matching cards. Successful programs here render images and allow mouse inputs from users, and typically, use dictionaries and lists. The next project is concerned with implementing a game of computer

Project	Training Size	Test Size	Avg. # Event Handlers	Avg. # Lines of Source	Avg. # Peer Graders
Stopwatch	3727	1599	2.74	87.07	5.06
Pong	3976	1709	3.10	154.28	5.09
Memory	4442	1917	3.058	95.33	5.18
Blackjack	3120	1347	4.11	226.8	5.23
Asteroids	2901	1255	3.94	263.93	5.50
Adv. Asteroids	2080	900	5.01	365.16	4.80

Table 4.1 : Details of test data used.

Blackjack; skills tested here include the ability to use tiled images and classes. The next project (referenced earlier in Sec. 4.1 and Sec. 4.2) is Asteroids, a classic arcade game where the player controls a space ship. Here, students are asked to correctly model acceleration and friction, use sounds, and use classes for spaceships and sprites. The last project is a more advanced version of the Asteroids game. It is the most difficult project in the course. Successful programs here use sets, groups of sprites, collisions, and sprite animation.

Some statistics regarding the six different programs are given in Figure 4.1.

Methodology. I used a randomly selected subset of the programs submitted for each assignment as a training set. All peer grades are visible during training, and the goal during testing is to predict the score of the other programs. The average score given by all peer graders for a test program is taken as the ground truth.

The seven different options for predicting the ground truth score were as follows:

1. k NN with Lift-Then-Compute distance. A small portion of the training set was

used to determine an appropriate value for k . The median score out of the k closest matches is used as the prediction.

2. k NN with Compute-Then-Lift distance. Except for the distance measure, this is identical to the above.
3. The prototype-based model. The prior used on the s_k values was a Normal($\mu, 0.5$) distribution, where μ was the average score over the training data. Prototype fragments of 100 randomly chosen programs were used. Since the learning algorithm converges quickly, I used only 20 complete Gibbs sampling cycles to learn the model. After the 20th update cycle, the current model was used to perform all predictions.
4. An ensemble method where a small portion of the training set is held back, and then each of the above models are learned on the remainder of the training data. Then, an l_2 -regularized regression is used to weight the above three models in such a way that the error of the weighted prediction on the held back training data is minimized. An additional intercept term is used in the learning to unbiased the model. The three models are then used together as an ensemble to perform the prediction.
5. An almost identical ensemble method, where the only difference is that an intercept term is not used.
6. The simple method of giving every program the average score. This is an illustrative strawman since the project scores are generally high with low variance, so it will achieve low error.
7. A “random human”. One of the humans who graded a program was chosen

at random, and his/her score was used to perform the prediction. Although there is nothing preventing a statistical method from doing better than a random participant in the class, it is difficult to imagine this happening. Thus, the “random human” should be seen as something of an upper bound on the performance of any statistical method.

4.7.2 Results

Results are given in Figure 4.6 and Figure 4.7.

Figure 4.6 gives the mean absolute error (MAE) achieved by each of the methods over all programs whose “true score” was below the specified cutoff. In general, there are relatively few programs that score poorly, so it is not difficult to achieve a low MAE by simply guessing a high score every time—this is why “always mean” does better and better as more and more programs are considered at the right of the plot. It is a much more difficult task to achieve a low MAE at all possible cutoffs.

The second figure gives the AUC ROC for each of the methods at all possible “good program”/“bad program” cutoffs. That is, it tests each method’s ability to determine whether a program was above or below a particular cutoff. Note that the task gets more and more difficult as the cutoff increases, because the distinctions between good programs and bad programs become more and more subtle with increasing cutoff.

Also Figure 4.5 shows the weights learned over each of the distance metrics by the prototype-based statistical model. All of the weights are normalized so that the most important metric is arbitrarily given a weight of one. This gives some idea of the relative importance of the six metrics for scoring each program, as determined by the fragment-based model.

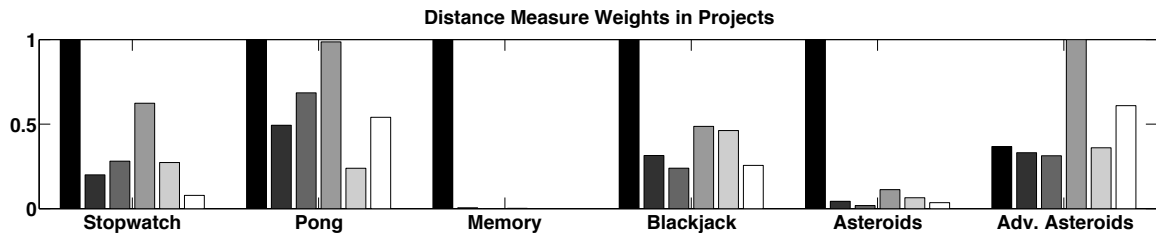


Figure 4.5 : Relative importance of the various distance measures, as learned by the prototype-based model. For each project, the bars represent the weight of d_1 through d_6 .

4.7.3 Discussion

A few findings stand out when examining the MAE plots.

First, the Compute-Then-Lift k NN-based regression model and the prototype-based model are (in my opinion) startlingly accurate, considering that a simple, statistical approach was used to recognizing poor programs. The results are particularly striking considering the worst programs. For example, for the Asteroids assignment, the worst 108 assignments received an average score of around 8 out of 20. Compute-Then-Lift and the prototype-based model had an average error of slightly greater than ± 3 when scoring such programs.

Another interesting finding is that the two ensemble methods seem to do a relatively poor job compared to the prototype-based model and k NN with the Compute-Then-Lift distance. The reason for this is that for several of the programs, Lift-Then-Compute seems to be quite biased; the ensemble-based methods have a difficult time taking this into account, and suffer accordingly.

But a different picture emerges when examining the AUC ROC plots. Here bias in predicting the final score is not a problem when performing a classification task,

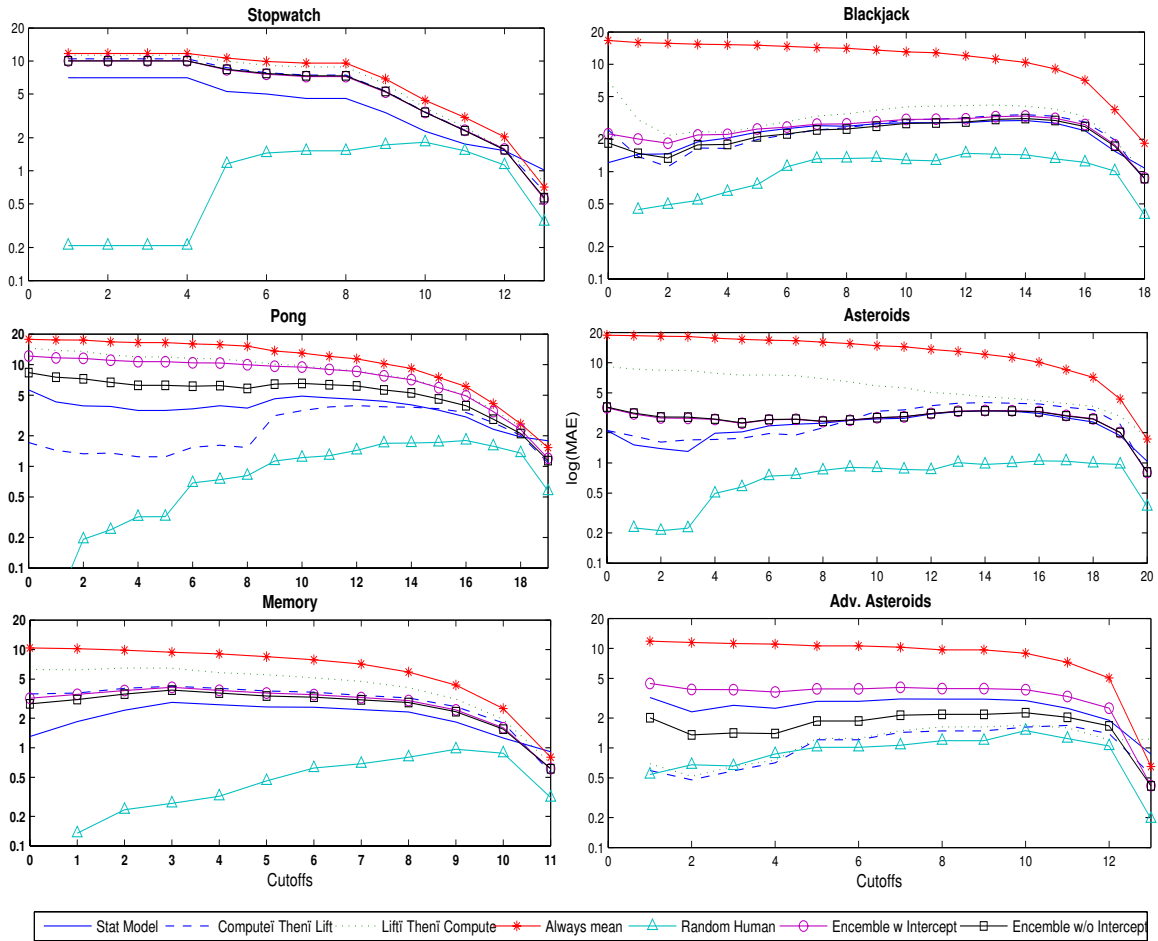


Figure 4.6 : Mean absolute error of the predictions made by the various methods as a function of the cutoff. For example, if a particular method has a MAE of 1.5 at a cutoff of 6.0, this means that the method achieved an MAE of 1.5 for all submitted programs where the true average score was 6.0.

because one can simply adjust the “good program”/“bad program” threshold to obtain more or less positive cases as desired, and correct for the bias. This is exactly the trade-off that the ROC curve explores. In this case, the Lift-Then-Compute distance does much better. Not only that, but the two ensemble-based methods that take

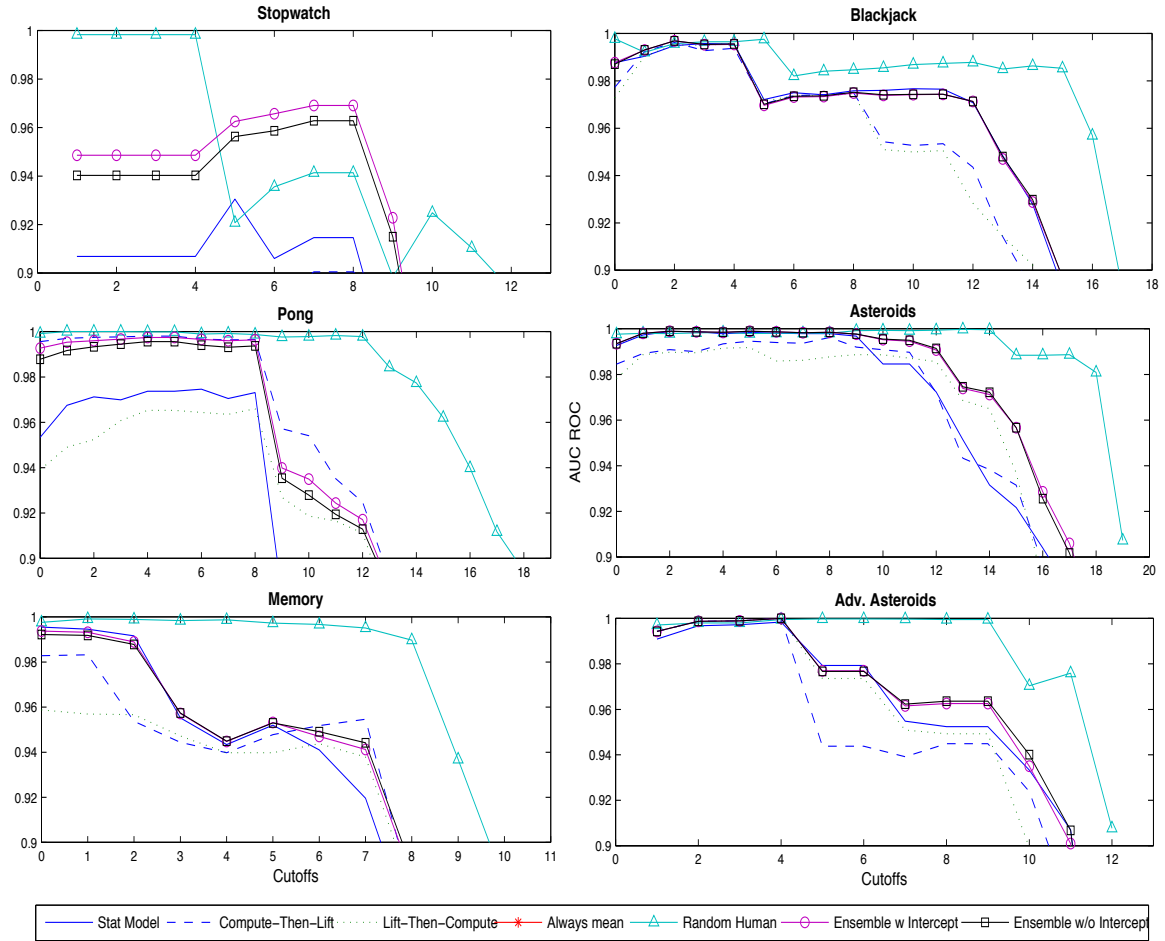


Figure 4.7 : AUC ROC for the predictions as a function of the “good program”/“bad program” cutoff. For example, is a particular method has a AUC ROC of 0.95 at a cutoff of 6.0, this means that it achieved an AUC ROC of 0.95 when differentiating between programs which received a 6.0 or less, and those that received greater than a 6.0.

into account all three models are clearly the best choice in terms of AUC, with the ensemble-with-intercept being the most universally applicable. Which of the three non-ensemble methods was best depended upon the program, but the prototype-based

statistical model had the best performance of the three in more situations.

While the ensemble-based methods might be the best, the question is: How well did they do? Are they useful? The answer seems to be that they did very well. They have an AUC of above 0.9 throughout the majority of the cutoff values considered, which indicates that they are very accurate for differentiating good from bad submissions. It is true that none of the automated methods approaches the accuracy of the random human for the higher-quality submissions, but human-like accuracy is not needed for performing tasks such as filtering out submissions from students who did not put in a serious effort on the assignment.

Finally, I close by discussing the relative importance of the various measures, as learned by the prototype-based model. This is plotted in Figure 4.5. It is interesting—and expected—that for the first three assignments, the type of event handler (`keypress`, for example) is of highest importance when comparing two fragments. This makes sense, because we generally want to ensure that two event handlers are of the same type when using one to score another. Only after it is clear that the two event handler types match, other metrics are considered. After that, the bag-of-guarded-updates is most important. Interestingly, the type of event handler drops in weight for the last (and most complex) project, and the more sophisticated measures are more important.

Chapter 5

Senders, Receivers and Authors in Document Classification

5.1 Problem Definition

In 2006, amendments to the US Federal Rules of Civil Procedure introduced the requirement to provide all relevant electronically stored information (ESI) in the discovery phase of litigation [45] (the discovery phase is the initial phase where the parties in a dispute are required to provide each other with relevant information and records). These changes brought the term *eDiscovery* or Electronic Discovery, to common use. eDiscovery relates to all electronic information including word documents, spreadsheets, emails, audio and video. eDiscovery has now become central to litigation, particularly when involving a review of terabytes of digital data. Clearly, automation can help with this task. The use of automation to manipulate ESI during any stage of eDiscovery is referred to as *Predictive coding*.

Predictive coding starts with a set of data grouped in a variety of ways, e.g., by keyword or concept searching. A sample of the data is manually reviewed and labeled by experts to create a training set. A machine learning software is then employed to categorize similar documents in the larger data set in terms of responsiveness, privilege and/or issue-relation based on the labeled examples. In this project I concentrate on case issue classification. Case issues are not mutually exclusive. Thus, this is an example of a multi-labeled classification problem.

In litigation, emails including attachments constitute around 75% of ESI. Unfortunately, exploiting the knowledge embedded in emails with automatic tools is challenging, due to the unstructured, noisy, and mixed language nature of this communication medium. The main characteristic that differentiates an email from a general text document is the existence of additional information in the email headers that can be exploited for various mining tasks. This work shows how in a meaningful way to use such information about senders/receivers/authors of a document to solve a supervised classification problem. I propose a novel vectorized representation of an arbitrary dimensionality of people associated with a document. In this solution, social network participants are placed in a latent space of a chosen dimensionality and have a set of weights specific to the roles they can play (e.g., in the email scenario the categories would be TO, FROM, CC, and BCC). The latent space positions together with the weights are used to map a set of people to a vector by taking a weighted average. The hypothesis is, that in case of multiple authors, the simple approach of mapping people to the dimensions (e.g., binary representation) does not perform well in classification models. In my solution, I use averaging instead of summation; I treat multiple authors as a set and calculate the weighted average.

Although, I specifically consider emails here, my method would apply to any set of documents that have some authors, senders or receivers.

Contributions I apply the proposed vectorized representation of authors in multiple modeling scenarios. As an example, I develop three Bayesian stochastic models, *Built-On*, *Hybrid Bayesian Lasso*, and *Social Network LDA*, that applying the novel solutions can learn single or multiple labels also with their correlations.

5.1.1 The Motivation

A very common approach to represent an email in a computational task is to have the subject, text and attachments in a TF format with the dimensionality of a size of the dictionary [82,88]. In many cases this is where it ends – senders/recipients are not even taken into account [50,81,83]. If people are considered though, the authors are usually presented in the binary format where the dimensionality is of the size of the social network [94] (or that multiplied by four if one also wants to represent the type of the role the person plays in the email; then we would have the binary vector for FROM, TO, CC, BCC [92]).

To show how an email can be used in a computational task and what shortcomings the standard approach may bring, I will use a simple example. Imagine a situation where the computational task is to predict if an incoming email is a phishing* email or not using historical dataset of emails and a regression model. The incoming email I am going to use as an example is presented in Figure 5.1. Imagine that this email was sent to hundred of people who do not have any history of any phishing activity. Looking at the short content, there is a lot of words, in the highlighted passages, which one would expect to be very indicative of a phishing label. A representation of this email could look like the TF vector shown in Figure 5.2, with other remaining words from the dictionary on the left (not shown to simplify the example), and the binary representation of the rest of the social network on the right (including hundred of the email recipients). Based on the historical data a regression model was trained which learnt the regression coefficients presented in vector β in Figure 5.2. The β values would suggest that the words from the text are very indicative of a phishing label.

*Phishing is the act of attempting to acquire sensitive information such as usernames, passwords, or credit card details by pretending to be a trustworthy entity.

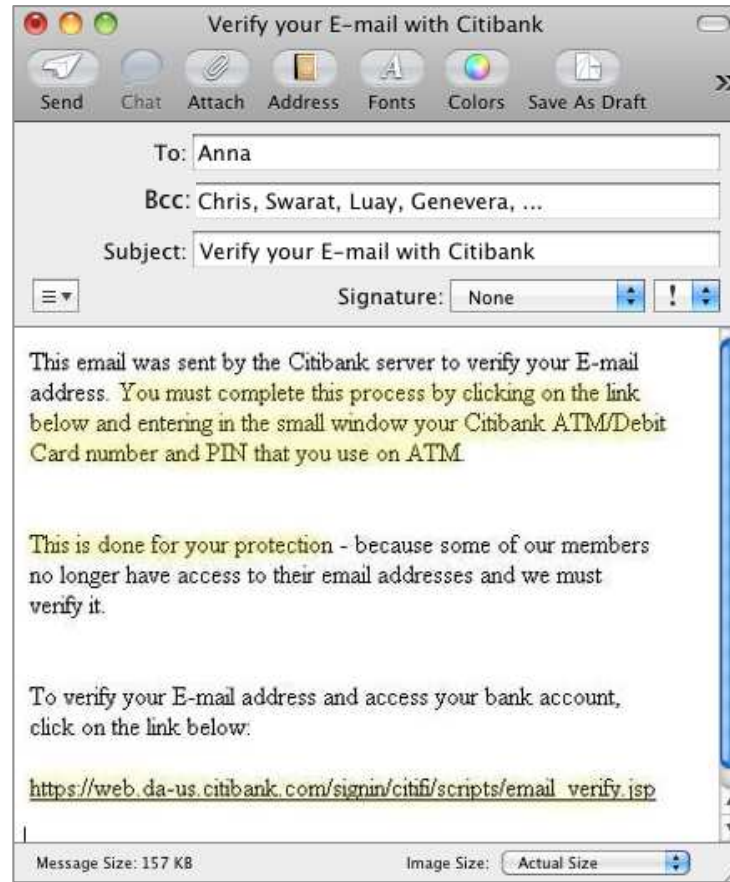


Figure 5.1 : An example of a phishing email.

When it comes to people, they are mostly not indicative of any phishing activities based on the history. Now, we would want to use those parameters β to predict if our email is a phishing email or not. As a linear regression model is to be used, we would have a dot product of the TF and β vectors. The hypothetical results for this scenario are shown in Figure 5.2. Although the content of the email would give a positive value of 3.3, the hundred of recipients not indicative of a phishing label would drag the overall value down to -36.7. In this simple case, the small negative value would strongly suggest that the email is not a phishing email – which would not be correct. The reason behind it is that people got simply mapped to the dimensions

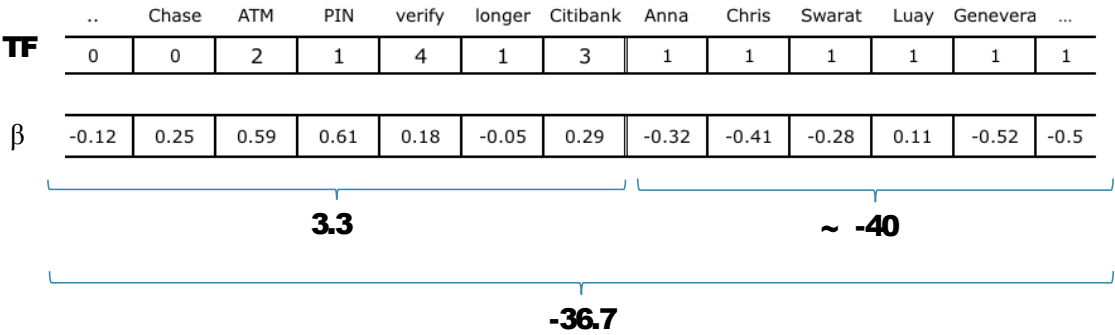


Figure 5.2 : Representation of an email form Figure 5.1 and regression coefficient vector

which makes the solution additive in nature.

To see if that intuition holds in reality, I carried out a small experiment using two subsets of 500 emails of one of the real datasets I use in the experimental part of this work. One of the subsets included emails with single people in the different header parts. In the other one, each email was associated with 17-19 people. I applied Bayesian Lasso on both subsets using the the email content as TF vectors extended by the binary representation for people associated with the emails. The task was to predict the true labels (case issues) of the emails in the label by label manner (i.e., one vs. all). I calculated AUC ROC on the Bayesian Lasso output (see Table 5.1). The average AUC for the second subset is lower than for the first one and close to 0.5 which supports the hypothesis that the simple mapping people to the dimensions does not work well in a multiple-author scenario.

Single-author	Multiple-author
Scenario	Scenario
0.6533	0.5596
0.6352	0.887
0.6964	0.5406
0.6899	0.3116
0.6598	0.7544
0.7915	0.4209
0.5193	0.6469
0.6602	0.2554
0.6632	0.5471

Table 5.1 : AUC ROC for Bayesian Lasso on two subsets of Technology dataset; each row corresponds to a single label with the last row showing the average AUC across all the labels (in **bold**)

5.2 The Novel Vectorized Representation of Senders, Receivers and Authors

On a high level, in my solution, I can take any or all of the categories of authors of a document and use them to get a vector of arbitrary dimensionality to represent people. To do that, I use a latent space of a chosen dimensionality and represent each participant of the social network as a point in that latent space. To explain the idea I will use a simple example. Figure 5.3 shows an email header with seven people associated with it. In the email case, we have a sender and three types of

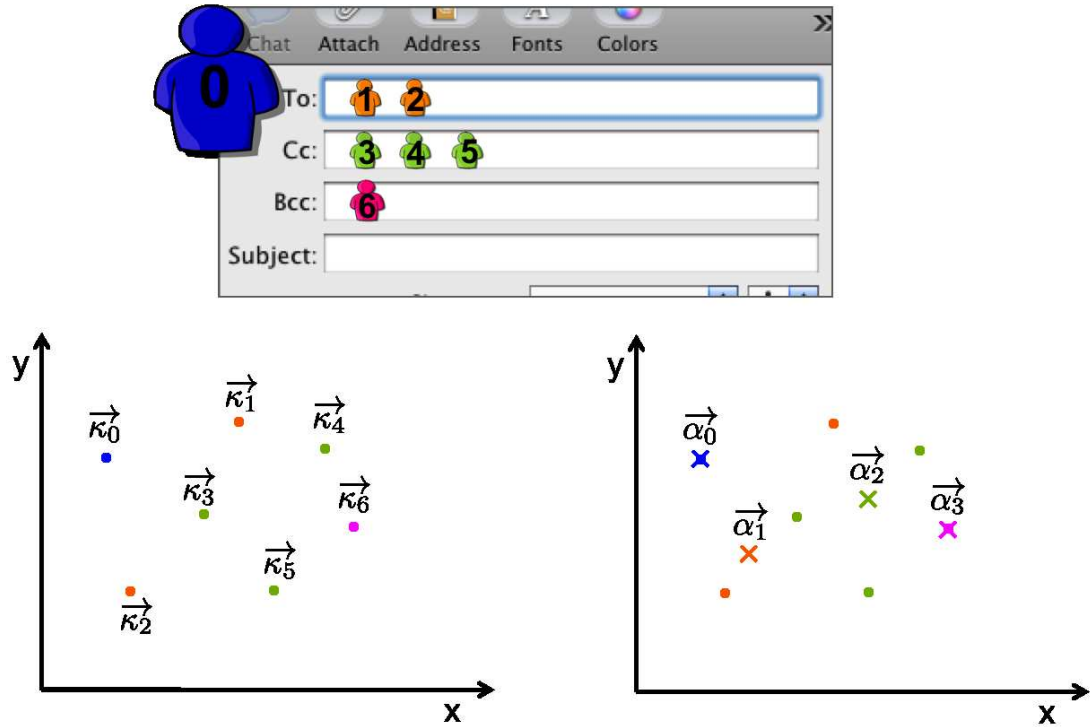


Figure 5.3 : Latent space positions κ for people associated with an email above (on the left); Vectorized representations α for four categories of people (on the right)

recipients which gives us in total four categories of people. In the discussed example, we have a sender, two main recipients, two recipients receiving the Carbon Copy and one getting the Blind Carbon Copy of the email. Each person has their position κ in the latent space which is shown in Figure 5.3 on the bottom left. In this example, the latent space is of dimensionality equals to 2. The way the latent space positions are generated depends on the statistical model I use this solution in. This is described in more detail in the later part of this Chapter.

As mentioned before, the method can take into account different categories of people associated with a document. The way the categories are taken into equation is by introducing category specific weights. Each person has such a set of weights. It

does seems reasonable to have those weights, as someone can be very indicative of a label as a sender of the email but not as much as a recipient. The main idea is that instead of mapping each person to a dimension in, e.g., a TF vector, I treat them as a set and calculate the weighted average of that set (or category):

$$\overrightarrow{\alpha_{d,c}} = \frac{\sum_{p(d,c)} \overrightarrow{\kappa_p} w_{p,c}}{\sum_{p(d,c)} w_{p,c}} \quad (5.1)$$

For each person p in the category c of email d , I have their latent space position κ_p and their category specific weight $w_{p,c}$. As we can see, the function becomes nonlinear. In the example in Figure 5.3 the weighted average values α for each of the four categories of people are shown on the bottom right marked with a cross. As we only have one person for category 0, i.e., FROM and category 3, i.e., BCC, α_0 and α_3 are in the same location as the latent space position of people in those categories, i.e., κ_0 and κ_3 . This is because the weights in the nominator and the denominator would cancel themselves. For the categories with multiple people, here TO and CC, their α representations, i.e., α_1 and α_2 , would be somewhere in between the positions of people in those categories. Specifically, to calculate the vectorized representation for the category TO for the email in the discussed example, we would have:

$$\overrightarrow{\alpha_1} = \frac{\overrightarrow{\kappa_1} w_{1,1} + \overrightarrow{\kappa_2} w_{2,1}}{w_{1,1} + w_{2,2}}$$

Using the formula 5.1, I can choose to group people in multiple ways. I use two different options in my models. In one way, I group people to four categories: TO, FROM, CC and BCC and calculate for email d the weighted average $\overrightarrow{\alpha_{d,c}}$ for each category c . The final vectorized representation for authors of a document d is a concatenated vector $\overrightarrow{\alpha_d}$ of the results for each category:

$$\overrightarrow{\alpha_d} = [\overrightarrow{\alpha_{d,0}}, \overrightarrow{\alpha_{d,1}}, \overrightarrow{\alpha_{d,2}}, \overrightarrow{\alpha_{d,3}}] \quad (5.2)$$

Hence, the dimensionality of such a vectorized representation in this case is four times the dimensionality of the latent space in which people are positioned. I use this approach in *BuiltOn*, and *Hybrid BL* models.

In another model, *Social Network LDA*, I need an option of having any number of dimensions without a restriction that it is dividable by the number of categories used (here by four). This is because the vectorized representation is used as a prior in that model. To be able to provide that flexibility, I slightly modify the formula and I pull all the categories together:

$$\vec{\alpha}_d = \frac{\sum_c \sum_{p(d,c)} \vec{\kappa}_p w_{p,c}}{\sum_c \sum_{p(d,c)} w_{p,c}} \quad (5.3)$$

It is easy to see the only difference is that I loop through through all the categories as opposed to keeping them separately. This way I can fully control the dimensionality of the resulting vector – which in here is the dimensionality of the latent space.

The Generative Process

The vectorized representation for people associated with a document exists in an email-people, or in general document-author, network. The network can be presented as a bipartite graph, where one disjoint set of nodes (“user nodes”) represents people and the other set (“email nodes”) the emails that have been exchanged. Each edge represents a link between an email and some individual. There are four types of links based on the type of communication, i.e., *FROM*, *TO*, *CC* and *BCC*. For instance, person *A* sends an email *e* to person *B*, and puts *C* and *D* into CC, as well as *E* into BCC. Based on that, there will be 5 links related to the particular email, i.e., a *FROM* link between node *A* and *e*, a *TO* link between *B* and *e*, two *CC* links between nodes *e* and *C* as well as *e* and *D*, and one link of the *BCC* type between

nodes e and E . The type of the link will dictate the weight applied to an edge. The four types of weights are learnt for each user. For instance, person A may be more predictive as a sender for a particular issue code, i.e., label, but not as much as a receiver of the message. In this case person's A *weight FROM* will be greater than his *weight TO*. The weights are used in calculating the vectorized representation of people explained in Section 5.2.

Given L labels, M users, a corpus with D documents, a latent space of dimensionality T , and the adjacency matrix P for the *TO*, *FROM*, *CC* and *BCC* connections between the “user nodes” and the “email nodes”, where p_d stores the connections for document d , the generative process of the vectorized representation for people associated with a document is as follows:

1. For each label $l \in L$:

(a) For each user $m \in M$:

i. Generate the set of weights

$$\Omega_{m,l} = \{\omega_{m,l,c}\}_{c=0}^C$$

$$\omega_{m,l,c} \mid a, b \sim \text{InvGamma}(a, b)$$

ii. For each dimension $t \in T$:

Generate the position for each person:

$$\kappa_{m,l,t} \mid \nu \sim \text{Normal}(0, \nu)$$

(b) For each document $d \in D$:

$$\text{Let } \alpha_{d,l} = f(\kappa_{*(d),l}, \Omega_{*,l}, p_d)$$

The whole process takes place for each label l and can be summarized as follows. In step (1a) I generate for each user m the set $\Omega_{m,l}$ containing weights $\omega_{m,l,c}$ for each category c . Next, I produce $\kappa_{m,l,t}$ which is the position of person m for label

l in T dimensional Euclidean space. In step (1b) for each document, I calculate the vectorized representation of authors for each label l according to equations 5.2 or 5.3, using the formula in equation 5.1.

Application

How could we use this vectorized representation of people to support classification? First, it can be used in a dot product with a coefficient matrix in a linear classification model, like regression. I do that in my *BuiltOn*, and *Hybrid BL* models. Second, it can be used as a prior in a topic-based classification model. This is what I do in my *Social Network LDA* model. The three models are described in the proceeding sections.

5.3 Built-On Model

The main idea behind *BuiltOn* model is to take a standard classifier and add to it in a meaningful way. It is done by concatenating the output of the classifier on the text content the to the novel vectorized representation of people. A Bayesian Lasso model is then trained on so concatenated vectors. In this way, a new regression is performed on the residuals which are left over from the text-based classifier in order to boost the classification accuracy. In my work, I use ridge regression and SVM as the base classifiers I want to improve on.

5.3.1 The Generative Process

In the subsequent part, I will refer to individual emails also as documents. Let the size of the vocabulary be $A = |Dict|$. Let $w_{d,i} \in Dict$ be the i th observation in d th document. Let there be D such documents, of which D_L are labeled and the rest, i.e.,

D_U have unobserved labels. Let the set of labels be $L = \{l_1, l_2 \dots l_L\}$. Each document has a variable $y_{d,l} \in \{-1, 1\}$ for every label, which indicates whether the label is applied to document d or not. In many cases $y_{d,l}$ will be unobserved.

Given the generative process in Section 5.2 for the latent space positions κ and the set of weights Ω as well as L labels, M users, a corpus with D documents, the adjacency matrix P for the *TO*, *FROM*, *CC* and *BCC* connections between the “user nodes” and the “email nodes”, where p_d stores the connections for document d , and $\beta_{d,l}$ being the output of a binary classifier for document d for label l , the generative process of the *BuiltOn* model is:

1. Draw $\sigma \mid a, b \sim \text{InvGamma}(a, b)$
2. Draw $\Sigma \mid \Psi, df \sim \text{IW}(\Psi, df)$
3. For each dimension $t \in T + 1$:
 - (a) For each label $l \in L$:

$$\text{Draw } \Lambda_{l,t} \mid \sigma \sim \text{Laplace}(0, \sigma)$$
4. For each document $d \in D$:
 - (a) Draw $y_d \mid \alpha_d, \beta_d, \Lambda \sim$

$$\text{Normal}(\bigvee_l \text{conc}(\beta_{d,l}, \alpha_{d,l}) \Lambda_l^\top, \Sigma)$$

The process can be summarized as follows. In step (1) we generate the Laplace distribution’s scale parameter σ and the covariance matrix Σ in step (2). In step (3) a regression matrix Λ is generated. In step (4) for each document, we create the set of labels y_d for document d . Using Multivariate Normal for generating labels accommodates for the correlations between the labels. The covariates in the model

are the vectors resulting from the concatenating a binary classifier outputs $\beta_{d,l}$ and vectorized representation of people $\alpha_{d,l}$, Λ are the regression coefficients on those vectors.

The choice of placing a zero-mean Laplace prior distribution on the regression matrix Λ is equivalent to a LASSO method in a frequentist context. This will cause more regression parameters to be driven to zero.

5.3.2 Inference

To employ a Bayesian approach, given X the posterior distribution $P(\Theta|X)$ over the parameter set Θ has to be determined.

In this case, the posterior distribution is:

$$P(\{\{\kappa_{m,l}, \Omega_{m,l}\}_{l=1}^L\}_{m=1}^M, \{\{\alpha_{d,l}\}_{l=1}^L\}_{d=1}^D, \Sigma, \Lambda, \sigma, \nu, \{y_{du}\}_{du=1}^{D_U} | \{p_d, \beta_d\}_{d=1}^D, \{y_{dl}\}_{dl=1}^{D_L}).$$

From elementary probability, we know that:

$$P(\Theta|X) = \frac{P(X|\Theta)P(\Theta)}{P(X)}$$

Furthermore, from the generative process above and Section 5.2, it follows that:

$$\begin{aligned} P(X|\Theta) &= \prod_{dl} \prod_l P(y_{dl,l} | f(\beta_d, \kappa_{dl,l}, \Omega_{*(dl),l}), \Lambda, \Sigma) \\ &= \prod_{dl} \text{Normal}_L(\bigvee_l \text{conc}(\beta_{dl}, \alpha_{dl,l}) \Lambda_l^\top, \Sigma) \end{aligned}$$

Also, we know that:

$$\begin{aligned}
P(\Theta) = & \prod_m \prod_l \prod_t P(\kappa_{m,l,t} | \nu) \times \prod_m \prod_l \prod_c P(\omega_{m,l,c} | a, b) \times \\
& \times \prod_l \prod_t P(\Lambda_{l,t} | \sigma) \\
& \times \prod_{du} P(y_{du} | \beta_{du}, \kappa_{*(du)}, \Omega_*, \Lambda, \Sigma) \\
& \times P(\Sigma | \Psi, df) \times P(\nu) \times P(\sigma)
\end{aligned}$$

where:

$$\begin{aligned}
P(\kappa_{m,l,t}) &= \text{Normal}(\kappa_{m,l}, 0, \nu) \\
P(\omega_{m,l,c}) &= \text{InvGamma}(\omega_{m,l,c}, 1, 1) \\
P(\Lambda_{l,t}) &= \text{Laplace}(\Lambda_{l,t}, 0, \sigma) \\
P(\Sigma) &= \text{IW}(\Sigma, \Psi, df) \\
P(\nu) &= \text{InvGamma}(\nu, 1, 1) \\
P(\sigma) &= \text{InvGamma}(\sigma, 1, 1) \\
P(y_{du}) &= \text{Normal}_L(\bigvee_l \text{conc}(\beta_{du}, \alpha_{du,l}) \Lambda_l^\top, \Sigma)
\end{aligned}$$

These give a formula for all of the components of $P(\Theta|X)$, except from $P(X)$. Obtaining an expression for $P(X)$ is very difficult as it involves integrating out all of the variables in Θ from $P(X, \Theta)$. A common solution to the problem of inferring a complex posterior distribution, like mine, is to apply an MCMC algorithm, such as Gibbs sampler [20]. Some key advantages of using Gibbs sampler are that (1) $P(X)$ becomes irrelevant, and (2) the Gibbs sampler obtains samples from $P(\Theta|X)$, which may be of more use than a closed form of the distribution itself. Those samples can be used to estimate the mean and other descriptive statistics of the components of Θ as well as joint statistics, e.g., covariance between variables.

A Gibbs sampler can be derived and applied here; with a few exceptions, all of the variables are discrete, or else the priors used are conjugate so sampling from the posterior is easy. Briefly, applying a Gibbs sampler requires that we be able to repeatedly re-sample the value of each unseen variable from its posterior distribution, conditioned upon the current values of all of the other variables.

For brevity, I do not give a derivation of the update samplers, but shortly describe them. The update samplers for Σ , and ν are based on conjugate priors and are given as:

$$\Sigma \sim \text{IW}(D + \Psi, \sum_{d=1}^D (y_d - \mu)(y_d - \mu)^\top)$$

$$\nu \sim \text{InvGamma}(a + M * T * L/2, b + \sum_m \sum_l (\kappa_{m,l})(\kappa_{m,l})^\top / 2)$$

The labels y_d are discrete. Here, I evaluate the probability of observing y_d given that (1) label $y_{d,l}$ is present, i.e., $y_{d,l} = 1$, and (2) label $y_{d,l}$ is absent, i.e., $y_{d,l} = -1$. Let p_d denote a vector where $p_{d,0} = \text{Normal}(y_d : y_{d,l} = -1) | \bigvee_l \text{conc}(\beta_{du}, \alpha_{du,l}) \Lambda_l^\top, \Sigma)$, and $p_{d,1} = \text{Normal}(y_d : y_{d,l} = 1 | \bigvee_l \text{conc}(\beta_{du}, \alpha_{du,l}) \Lambda_l^\top, \Sigma)$. After normalizing p_d , we have:

$$y'_{d,l} \sim \text{Binomial}(p_{d,1})$$

Then, $y_{d,l} = 1$ if $y'_{d,l} = 1$, and $y_{d,l} = -1$ otherwise.

The rest of the variables are continuous and the priors are not conjugate. Thus, a rejection sampler [20] is used. Let $\kappa'_{m,l}$ denote the version of $\kappa_{m,l}$ where the t th entry in the vector has been replaced with the candidate value $\kappa'_{m,l,t}$. Then the posterior likelihood of $\kappa_{m,l,t}$ is computed as:

$$\kappa_{m,l,t} \propto \text{Normal}(\kappa'_{m,l,t} | 0, \nu) \prod_{d=1}^{D(m)} \text{Normal}(\bigvee_l \text{conc}(\beta_{d,l}, \alpha_{d,l}) \Lambda_l, \Sigma)$$

where $d = 1$ to $D(m)$ denote all the “email nodes” connected to the “user node” m , and $\alpha'_{d,l}$ indicates an updated version of $\alpha_{d,l}$ taking into account the candidate value $\kappa'_{m,l,t}$.

Similarly, having $\omega'_{m,l,c}$ to indicate the candidate value for $\omega_{m,l,c}$, the posterior likelihood of $\omega_{m,l,c}$ looks as follows:

$$\omega_{m,l,c} \propto \text{InvGamma}(\omega'_{m,l,c}|a, b) \prod_{d=1}^{D(m)} \text{Normal}\left(\bigvee_l \text{conc}(\beta_d, \alpha_{d,l})\Lambda_l, \Sigma\right)$$

where $d = 1$ to $D(m)$ denote all the “email nodes” connected to the “user node” m and $\alpha'_{d,l}$ indicates an updated version of $\alpha_{d,l}$ taking into account the candidate value $\omega'_{m,l,c}$.

Also, let σ' be the candidate values for σ . Then, the posterior likelihood is given as:

$$\sigma \propto \text{InvGamma}(\sigma'|a, b) \prod_{l=1}^L \prod_{t=1}^T \text{Laplace}(\Lambda_{l,t}|0, \sigma')$$

5.4 Hybrid Bayesian Lasso

The Hybrid Bayesian Lasso model is a full Bayesian treatment of the *BuiltOn* model. It has an iterative character where two Bayesian Lasso models run interchangeably: (1) Bayesian Lasso on the TF representation of the text of the documents (2) Bayesian Lasso on the novel vectorized representation of people in the social network.

5.4.1 Generative Process

The Bayesian Lasso on the TF vectors of the text is a well-known regularized Bayesian linear regression formulation as presented in [107]. Let the data set \mathbf{X} be an $D \times A$ matrix of standardized regressors and \tilde{y}_d a vector of size L of centered responses for document d . Given the above, the generative process from Section ?? as well as L

labels, a dictionary with A words, M users, and a corpus with D documents, the generative process of the *Hybrid BL* model is:

1. Draw $\sigma^2 \mid a, b \sim \text{InvGamma}(r, \rho)$
2. Draw $\beta \sim \text{Laplace}(0, \varsigma)$
3. $\varsigma^2 \sim \text{Gamma}(a, b)$
4. For each document $d \in D$:
 - (a) For each label $l \in L$:

$$\text{Draw } \tilde{y}_d \sim \text{Normal}(\beta \cdot \mathbf{x}, \sigma^2)$$
5. Draw $\sigma \mid a, b \sim \text{InvGamma}(a, b)$
6. Draw $\Sigma \mid \Psi, df \sim \text{IW}(\Psi, df)$
7. Draw $\nu \mid a, b \sim \text{InvGamma}(a, b)$
8. For each dimension $t \in T$:
 - (a) For each label $l \in L$:

$$\text{Draw } \Lambda_{l,t} \mid \sigma \sim \text{Laplace}(0, \sigma)$$
 - (b) Draw $\hat{y}_d \mid \alpha_d, \beta_d, \Lambda \sim$

$$\text{Normal}(\bigvee_l \text{conc}(\beta_{d,l}, \alpha_{d,l}) \Lambda_l^\top, \Sigma)$$
 - (c) Let $y_d = \tilde{y}_d + \hat{y}_d$

Steps (1-4) cover the generative process for the first part of the model, i.e., text-based Bayesian Lasso, and steps (5-8) describe the generative process for Bayesian on the vectorized representation for people associated with a document

Steps(5-10) are almost the same as the generative process for *BuiltOn* model presented in the previous section. The two differences are that first, the dimensionality in step (9) is T as opposed to $T+1$; second, in step (9b) $\text{Normal}(\bigvee_l \text{conc}(\beta_d, \alpha_{d,l})\Lambda_l, \Sigma)$ is replaced by $\text{Normal}(\bigvee_l \alpha_{d,l}\Lambda_l, \Sigma)$.

5.4.2 Inference

The MCMC simulation for learning the model is presented below. For the textual part we used the Gibbs sampler suggested in [107] which looks as follows:

$$\begin{aligned} 1/(\tau)^2 &\sim \text{InvGamma} \left(\sqrt{\frac{\zeta^2 \sigma^2}{\beta^2}}, \zeta^2 \right) \\ \beta &\sim \text{Normal} (\mathbf{A}^{-1} \mathbf{X}^\top, \sigma^2 \mathbf{A}^{-1}) \\ \zeta^2 &\sim \text{Gamma} \left(A + r, \frac{1}{\sum_p \tau^2 / 2 + \rho} \right) \\ \sigma^2 &\sim \text{InvGamma} \left(\frac{D + A + a}{2}, \frac{b + 1 + \sum_d (\tilde{y}_d - \beta \cdot \mathbf{x})^2}{2} \right) \end{aligned}$$

where

$$\begin{aligned} \mathbf{A} &= \mathbf{X}^\top \mathbf{X} + \mathbf{D}_\tau^{-1} \\ \mathbf{D}_\tau &= \text{diag} (\tau_1^2, \tau_2^2, \dots) \end{aligned}$$

The Gibbs sampler for the Bayesian Lasso on the vectorized representation of people part resembles the one of *BuiltOn* in Section 5.3.2. All the updates look the same except the ones for κ , ω and labels y where $\text{Normal}(\bigvee_l \text{conc}(\beta_d, \alpha_{d,l})\Lambda_l, \Sigma)$ is replaced by $\text{Normal}(\bigvee_l \alpha_{d,l}\Lambda_l, \Sigma)$.

5.5 The Social Network LDA Model.

The existing author-topic model [46] considers a scenario where the representation of a document's content is enriched by including the information of its authors. The model uses the topic-based representation to model both the content of the document and their authors. Topic models [22] such as LDA [21] and PLSI [23] are statistical models most often used for analyzing document corpora. A *topic* in a topic model is a set of words that tend to appear together in a corpus, and the documents in the corpus are viewed as being produced by some mixture of topics. In LDA, each word in a document is produced by one of T topics; the probability of an arbitrary word in document j being produced by topic t is given by the t th entry in the vector θ_j , which I refer to as the document's *topic mixing proportion* vector.

In the author-topic model, for each word in the document an author is chosen and then a topic is chosen from a distribution of topics specific to that author; the word is generated from the chosen topic.

In the *Social Network LDA*, I also use topic models to represent the corpus. The main difference is that in the presented model, the information about the authors are used to generate a prior for the topic mixing proportion vector of a document (email body). This is motivated by my previous study [47] which showed that the meaningful prior on the document topic mixing proportion vectors can yield a better topic modeling structure for clustering/classification problems. Furthermore, the author-topic model uses the topic-based approach only to characterize emails (topic-based content representation); I consider the problem of multi-label document classification where the content representation is by default its by-product. In that respect, my model resembles the supervised topic model [29]. Although, I do the regression in the same way (described in more detail in the generative process below), the significant

difference is the prior on the topic mixing proportion vectors where I use the novel vectorized representation of authors. As a matter of fact, I use a supervised topic model as a baseline model in my experimental study.

In Social Network LDA I have chosen the topic modeling approach to represent the documents. However, other methods can be used here to reduce the dimensionality of the data.

5.5.1 The Generative Process

In LDA [21], each word in a document is produced by one of T topics; the probability of an arbitrary word in document d being produced by topic t is given by the t th entry in the vector θ_d , which I refer to as the document’s *topic mixing proportion* vector. In “classic” LDA, θ_d is sampled from a Dirichlet distribution with parameter α ; α is typically taken to be a symmetric, constant vector (in practice, $\alpha = \langle 0.1, 0.1, \dots, 0.1 \rangle$ is often used). In the presented model, the vector α_d is the vectorized representation of people calculated as in Equation 5.3.

Given T topics, L labels, a dictionary with A words, M users, and a corpus with D documents where the length of document d is N_d , the generative process of the *Social Network LDA* is presented below. Here, I also include the steps for generating the positions for people in the latent space and the set of weights, as they slightly differ from the generative process shown in Section 5.2.

1. Draw $\sigma \mid a, b \sim \text{InvGamma}(a, b)$
2. Draw $\Sigma \mid \Psi, df \sim \text{IW}(\Psi, df)$
3. For each user $m \in M$:

- (a) Generate the set of weights

$$\Omega_m = \{\omega_{m,c}\}_{c=0}^C$$

$$\omega_{m,c} \mid a, b \sim \text{InvGamma}(a, b)$$

- (b) For each topic $t \in T$:

$$\text{Generate the position of the "user node": } \kappa_{m,t} \mid a, b \sim \text{InvGamma}(a, b)$$

- 4. For each topic $t \in T$:

- (a) Draw $\phi_t \mid \beta \sim \text{Dirichlet}_A(\beta)$

- (b) For each label $l \in L$:

$$\text{Draw } \Lambda_{l,t} \mid \sigma \sim \text{Laplace}(0, \sigma)$$

- 5. For each document $d \in D$:

- (a) Let $\alpha_d = f(\kappa_{*(d)}, \Omega_*, p_d)$

- (b) Draw $\theta_d \mid \kappa_{1:M}, \Omega_{1..M} \sim \text{Dirichlet}_T(\alpha_d)$

- (c) For each word i :

$$\text{Draw } z_{d,i} \mid \theta_d \sim \text{Mult}_T(\theta_d)$$

$$\text{Draw } w_{d,i} \mid z_{d,i}, \phi_{1:T} \sim \text{Mult}_A(\phi_{z_{d,i}})$$

- (d) Draw $y_d \mid z_{d,1:N_d}, \Lambda \sim \text{Normal}(\bar{z}_d \Lambda^\top, \Sigma)$

The process can be summarized as follows. In step (1) we generate the Laplace distribution's scale parameter σ . In step (2) we produce the covariance matrix Σ . In step (3), for each user m , the set Ω_m containing weights $\omega_{m,c}$ for each category c is generated. Further, we produce the positions of the "user nodes" κ in T dimensional Euclidean space. In step (4) we generate two variables. First, we produce a matrix ϕ where $\phi_{t,r}$ is the probability that topic t will produce word r . Second, we produce a regression matrix Λ .

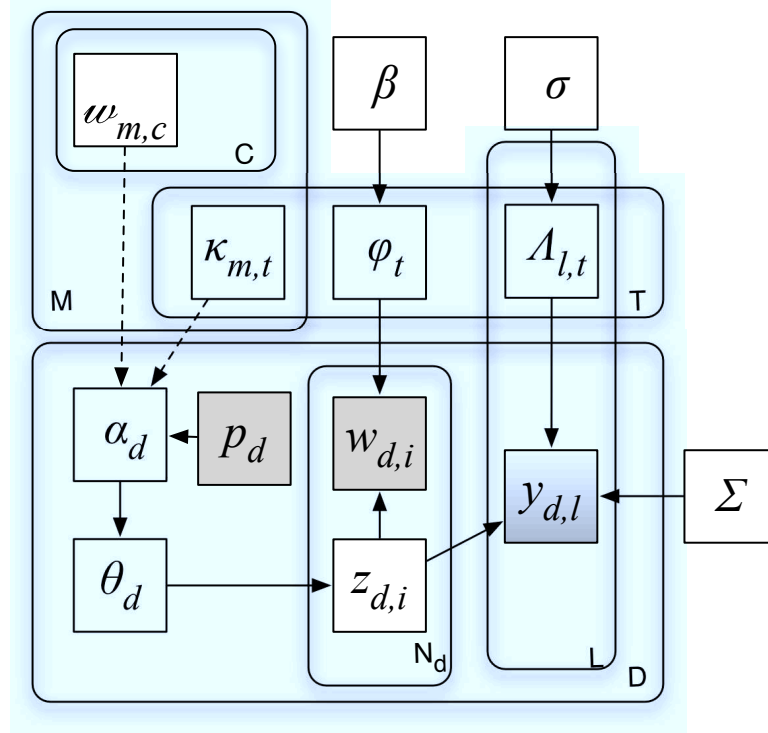
In step (5) we produce the actual documents. To produce a document d , the α_d vector is first calculated as a weighted average of the positions $\kappa_{*(d)}$ of all the “user nodes” connected to the “document node” d as shown in Equation 5.3. Next, the topic proportion vector θ_d is generated using the prior α_d calculated above. Using θ_d , the N_d words in the document are allocated to the T topics; the resulting allocation is stored in the vectors z_d , where $z_{d,i}$ is the topic t which produced word i . Here, we also calculate matrix C^D having D rows and T columns, where $C_{d,t}^D$ indicates the number of words produced by topic t in document d . Finally, we produce word $w_{d,i}$ by sampling from the topic vector θ_t . Based on that we calculate the matrix C^A having T rows and A columns, where $C_{t,r}^A$ indicates the number of times word r was produced by topic t across the whole corpus.

The last step is to create the set of labels y_d for document d . Using Multivariate Normal for generating labels accommodates for the correlations between the labels. The covariates in the model are $\bar{z}_{1:D}$, where $\bar{z}_d = [\bar{z}_{d,1}, \dots, \bar{z}_{d,t}, \dots, \bar{z}_{d,T}]$ is the empirical topic distribution for document d , and $\bar{z}_{d,t} = (1/N_d) \sum_{i=1}^{N_d} z_{d,i,t}$. This means, that each entry of the auxiliary variable \bar{z}_d is the percentage of the words in document d that come from topic t . Λ are the regression coefficients on those frequencies. Usually, a linear model, as one used here, includes an intercept term. Here, by definition, the components of \bar{z}_d always sum to one, hence, the intercept term is omitted. By regressing the response $y_{d,l}$, on \bar{z}_d instead of θ_d , the response is made dependent on the actual topic frequencies in the document (as in [29]) , and not the mean of the distribution generating the topics.

The family of probability distributions corresponding to this generative process is depicted as a graphical model in Figure 5.4.

The choice of placing a zero-mean Laplace prior distribution on the regression

Figure 5.4 : A graphical model representation of Social Network LDA. The dotted arrows represent the deterministic relationship between variables.



matrix Λ is equivalent to a LASSO method in a frequentist context. This will cause more regression parameters to be driven to zero.

The model is made fully Bayesian by putting appropriate priors on all of the parameters. The conjugate prior over covariance matrix Σ is Inverse Wishart with degrees of freedom $df = L + 2$ and scale matrix $\Psi = I$, chosen to give a broad prior. To complete the model specification, β and σ are sampled from uninformative prior $\text{InvGamma}(a, b)$, and a and b are assumed to be known hyperparameters.

5.5.2 Inference

The posterior distribution for *Social Network LDA* is given by:

$$P(\{\phi_t\}_{t=1}^T, \{\kappa_m, \Omega_m\}_{m=1}^M, \{\theta_d, \alpha_d, z_d\}_{d=1}^D, \Sigma, \Lambda, \beta, \sigma, \{y_{du}\}_{du=1}^{D_U} | \{w_i\}_{i=1}^N, \{p_d\}_{d=1}^D, \{y_{dl}\}_{dl=1}^{D_L}).$$

From the generative process described above, it follows that:

$$\begin{aligned} P(X|\Theta) &= \prod_d \prod_i^{N_d} \prod_t P(w_{d,i} | \phi_t, z_{d,i}) \times \prod_{dl} \prod_l P(y_{dl,l} | z_{dl}, \Lambda, \Sigma) \\ &= \prod_d \prod_i^{N_d} \prod_t (\text{Mult}_A(w_{d,i}, \phi_t))^{z_{d,i,t}} \\ &\quad \times \prod_{dl} \text{Normal}_L(y_{dl}, \bar{z}_{dl} \Lambda^\top, \Sigma) \end{aligned}$$

Also, from the Bayesian network presented as a plate diagram in Figure 5.4 and the priors listed above, we know that:

$$\begin{aligned} P(\Theta) &= \prod_d \prod_{i=1}^{N_d} P(z_{d,i} | \theta_d) \times \prod_d P(\theta_d | \alpha_d(\kappa_*, w_*, p_d)) \\ &\quad \times \prod_m \prod_t P(\kappa_{m,t} | a, b) \times \prod_m \prod_c P(\omega_{m,c} | a, b) \times \\ &\quad \times \prod_t P(\phi_t | \beta) \times \prod_l \prod_t P(\Lambda_{l,t} | \sigma) \\ &\quad \times \prod_{du} P(y_{du} | z_{du}, \Lambda, \Sigma) \\ &\quad \times P(\Sigma | \Psi, df) \times P(\beta) \times P(\sigma) \end{aligned}$$

where:

$$P(z_{d,i}) = \text{Multinomial}_T(z_{d,i}, \theta_d)$$

$$P(\theta_d) = \text{Dirichlet}_T(\theta_d, \alpha_d(\kappa_*))$$

$$P(\kappa_{m,t}) = \text{InvGamma}(\kappa_{m,t}, 1, 1)$$

$$P(\omega_{m,c}) = \text{InvGamma}(\omega_{m,c}, 1, 1)$$

$$P(\phi_t) = \text{Dirichlet}_A(\phi_t, \beta)$$

$$P(\Lambda_{l,t}) = \text{Laplace}(\Lambda_{l,t}, 0, \sigma)$$

$$P(\Sigma) = \text{IW}(\Sigma, \Psi, df)$$

$$P(\beta) = \text{InvGamma}(\beta, 1, 1)$$

$$P(\sigma) = \text{InvGamma}(\sigma, 1, 1)$$

$$P(y_{du}) = \text{Normal}_L(y_{du}, \bar{z}_{du}\Lambda^\top, \Sigma)$$

As in the previous models, a Gibbs sampler can be also derived and applied here. The update sampler for Σ and σ are exactly the same as in the *BuiltOn* model.

The update samplers for ϕ_t , Θ_d , and $z_{d,i}$ are given as:

$$\phi_t \sim \text{Dirichlet}(\beta + C_{t,r}^A)$$

$$\theta_d \sim \text{Dirichlet}(\alpha_{d,t} + C_{d,t}^D)$$

$$z_{d,r} \sim \text{Categorical}[\theta_d \times \phi_{*,r} \times \text{Normal}(y_d, \bar{z}_d\Lambda^\top, \Sigma)]$$

The first two updates are based on conjugate priors. The $z_{d,r}$ update requires a bit of explanation. Here, we need to compute the posterior probability for each value explicitly, by evaluating the probability of observing $z_{d,r}$ given the topic assignment. For each topic, the vector \bar{z}_d has to be updated. Then, let p_{dr} denote a vector where $p_{dr,t} = \theta_d \times \phi_{t,r} \times \text{Normal}(y_d | \bar{z}_d\Lambda^\top, \Sigma)$. After normalizing p_{dr} to ensure that the

distribution sums to one, we have:

$$z_{d,r} \sim \text{Categorical}(p_{dr})$$

Similarly to $z_{d,r}$, the labels y_d are also discrete. Here, we evaluate the probability of observing y_d given that (1) label $y_{d,l}$ is present, i.e., $y_{d,l} = 1$, and (2) label $y_{d,l}$ is absent, i.e., $y_{d,l} = -1$. Let p_d denote a vector where $p_{d,0} = \text{Normal}(y_d : y_{d,l} = -1 | \bar{z}_d \Lambda^\top, \Sigma)$, and $p_{d,1} = \text{Normal}(y_d : y_{d,l} = 1 | \bar{z}_d \Lambda^\top, \Sigma)$. After normalizing p_d , we have:

$$y'_{d,l} \sim \text{Binomial}(p_{d,1})$$

Then, $y_{d,l} = 1$ if $y'_{d,l} = 1$, and $y_{d,l} = -1$ otherwise.

The rest of the variables are continuous and the priors are not conjugate. Thus, a rejection sampler [20] is used. Let κ'_m denote the version of κ_m where the t th entry in the vector has been replaced with the candidate value $\kappa'_{m,t}$. Then the posterior likelihood of $\kappa_{m,t}$ is computed as:

$$\kappa_{m,t} \propto \text{InvGamma}(\kappa'_{m,t} | a, b) \prod_{d=1}^{D(m)} \text{Dirichlet}(\theta_d | \alpha'_d)$$

where $d = 1$ to $D(m)$ denote all the “email nodes” connected to the “user node” m . α'_d indicates an updated version of α_d taking into account the candidate value $\kappa'_{m,t}$.

Similarly, let $\omega'_{m,c}$ indicate the candidate value for $\omega_{m,c}$. The posterior likelihood of $\omega_{m,c}$ becomes:

$$\omega_{m,c} \propto \text{InvGamma}(\omega'_{m,c} | a, b) \prod_{d=1}^{D(m)} \text{Dirichlet}(\theta_d | \alpha'_d)$$

where $d = 1$ to $D(m)$ denote all the “email nodes” connected to the “user node” m . α'_d indicates an updated version of α_d taking into account the candidate value $\omega'_{m,c}$.

Further, let $\Lambda'_{l,t}$ be the candidate value for $\Lambda_{l,t}$ and let Λ' denote the version of Λ where the entry in l th row and t th column in the matrix has been replaced by the

candidate value $\Lambda'_{l,t}$. Then, the posterior likelihood is given as:

$$\Lambda_{l,t} \propto \text{Laplace}(\Lambda'_{l,t}|0, \sigma) \prod_{d=1}^D \text{Normal}(y_d|\bar{z}_d\Lambda'^T, \Sigma)$$

Also, let β' and σ' be the candidate values for β and σ respectively. Then, the posterior likelihoods are given as:

$$\begin{aligned} \beta &\propto \text{InvGamma}(\beta'|a, b) \prod_t^T \text{Dirichlet}(\phi_t|\beta') \\ \sigma &\propto \text{InvGamma}(\sigma'|a, b) \prod_{l=1}^L \prod_{t=1}^T \text{Laplace}(\Lambda_{l,t}|0, \sigma') \end{aligned}$$

5.6 Experiments

5.6.1 Experimental Setup

To evaluate my solution, I utilize three datasets of emails with attachments coming from the real eDiscovery cases in the area of construction, finance and technology; I call the datasets Construction, Finance and Technology accordingly. The details of the datasets are presented in Figure 5.2. Each email dataset was labeled by the experts (i.e., attorneys reviewing the documents in the discovery phase of a litigation) according to some case issues. Those labels are considered the ground truth. The classification task that I performed was to predict all the case issues that an email relates to.

For training, I randomly selected about 60% of documents, making sure that all the participants are represented in the training data. In other words, in the training set there is at least one document associated with every single person in the social network.

I learn and evaluate thirteen classification/regression models:

1. Ridge regression using the TF representation only of the text content (called *RR text*)
2. Ridge regression extending the TF vector from model (1) by concatenating a binary representation of people for each category (*RR text+people*)
3. BuiltOn model taking the output from the *BuiltOn RR* text-based method (1) and concatenating the novel vectorized presentation of people (*BuiltOn RR*)
4. Another version of *BuiltOn RR* (3) with added correlation between labels (*BuiltOn RR Corr*)
5. SVM using the TF representation only of the text content (*SVM text*)
6. SVM extending the TF vector from model (5) by concatenating a binary representation of people for each category (*SVM text+people*)
7. BuiltOn model taking the output from *SVM text* (5) and concatenating the novel vectorized presentation of people (*BuiltOn SVM*)
8. Another version of *BuiltOn SVM* (7) with added correlation between labels (*BuiltOn SVM Corr*)
9. Bayesian Lasso using the TF representation only of the text content (*BL text*)
10. Bayesian Lasso extending the TF vector from model (9) by concatenating a binary representation of people for each category (*BL text+people*)
11. Hybrid model which iterates between Bayesian Lasso performed on text and Bayesian Lasso on the novel vectorized representation of people (*Hybrid BL*)

Data Set	# Labels	Avg. Prevalence	Training Size	Test Size	# Participants	Dict. Size	Max. Doc. Length
Construction	9	12.22%	8,332	6,254	2,051	26,185	356,580
Finance	8	14.02%	4,659	3,074	1,084	11,295	165,491
Technology	8	16.24%	3,569	2,253	3,835	18,930	211,184

Table 5.2 : Details of test data used.

12. Topic-based classification model [29] where the text dictionary is extended by people for each category (*Topic Model text+people*)
13. An improved variation of the topic-based model (12), where the novel vectorized representation of people is used as a prior to topic mixing proportion vectors of documents represented by a standard text dictionary (*Social Network LDA*)

5.6.2 Results

The results for the three datasets and thirteen different learning models are presented in Tables 5.3, 5.4, 5.5, and 5.6 in the form of AUC ROC. AUC ROC [108] stands for the Area Under the Curve of Receiver Operating Characteristics and plots the true positive vs. the false positive values for a binary classifier as its discrimination threshold varies. Since a random method has an AUC of 0.5, minimally, classifiers should perform better than this; the higher they score than one another (i.e., the larger the area under the ROC curve), the better their expected performance.

	RR text	RR text+people	RR BuiltOn	RR BuiltOn Corr
C	0.7804	0.5589	0.7848	0.7815
o	0.7789	0.7634	0.8037	0.8039
n	0.5856	0.5854	0.7031	0.7033
s	0.6501	0.5011	0.7061	0.7053
t	0.5044	0.6234	0.6358	0.6359
r	0.5179	0.6475	0.6591	0.6654
u	0.5219	0.5612	0.8039	0.8038
c	0.6754	0.6901	0.6652	0.6760
t	0.6049	0.6538	0.6479	0.6485
	0.6244	0.6205	0.7122	0.7137
	0.8559	0.8568	0.8663	0.8640
F	0.7226	0.7261	0.7687	0.7684
i	0.7951	0.7921	0.8185	0.8182
n	0.6905	0.7059	0.7706	0.7722
a	0.8601	0.8396	0.8654	0.8594
n	0.7395	0.7368	0.7650	0.7648
c	0.7986	0.7593	0.8073	0.8080
e	0.8296	0.8234	0.8335	0.8330
	0.7865	0.7800	0.8119	0.8110
	0.6793	0.6138	0.6795	0.6802
	0.5430	0.5820	0.5458	0.5465
T	0.6674	0.7003	0.6682	0.686
e	0.5993	0.6445	0.5960	0.5968
c	0.8647	0.8634	0.8668	0.8668
h	0.6526	0.6336	0.6526	0.6531
	0.7563	0.7227	0.7557	0.7556
	0.6528	0.6099	0.6597	0.6597
	0.6769	0.6713	0.6780	0.6806

Table 5.3 : AUC ROC for Ridge Regression based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in **bold**)

	SVM text	SVM text+people	SVM BuiltOn	SVM BuiltOn Corr
C	0.9220	0.9228	0.9254	0.9163
o	0.9489	0.9509	0.9595	0.9588
n	0.8821	0.8847	0.8878	0.8891
s	0.7917	0.8074	0.8131	0.8092
t	0.7897	0.7911	0.8524	0.8577
r	0.7874	0.7982	0.8203	0.8430
u	0.9274	0.9111	0.9419	0.9460
c	0.7357	0.7247	0.8517	0.8533
t	0.7153	0.7245	0.7889	0.8523
	0.8334	0.8350	0.8712	0.8806
	0.9644	0.9648	0.9614	0.9631
F	0.9171	0.9129	0.9037	0.9156
i	0.9419	0.9411	0.9415	0.9428
n	0.9263	0.9248	0.8810	0.9224
a	0.9660	0.9658	0.9740	0.9720
n	0.8364	0.8413	0.8575	0.8506
c	0.9518	0.9524	0.9462	0.9493
e	0.9281	0.9243	0.9274	0.9298
	0.9290	0.9284	0.9241	0.9307
	0.8748	0.8764	0.9310	0.9513
	0.6185	0.6341	0.8204	0.8598
T	0.8220	0.8186	0.9499	0.9678
e	0.7662	0.7696	0.8637	0.8935
c	0.8835	0.8743	0.9893	0.9903
h	0.6521	0.6603	0.8769	0.8810
n	0.9406	0.9443	0.9789	0.9813
	0.8308	0.8389	0.9161	0.9316
	0.7986	0.8021	0.9158	0.9321

Table 5.4 : AUC ROC for the SVM based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in **bold**)

	BL text	BL text+people	Hybrid BL
C	0.7915	0.7774	0.8227
o	0.8236	0.7976	0.8206
n	0.6884	0.7135	0.7201
s	0.7021	0.6720	0.7243
t	0.6385	0.6170	0.6347
r	0.6571	0.6751	0.6900
u	0.7858	0.7975	0.8104
c	0.6491	0.6808	0.6858
t	0.7193	0.6380	0.6649
	0.7173	0.7077	0.7304
F	0.8119	0.8161	0.8392
i	0.7192	0.7150	0.7961
n	0.7575	0.7803	0.7865
a	0.7183	0.7742	0.7910
n	0.8180	0.7142	0.8675
c	0.7364	0.7208	0.7636
e	0.8731	0.8652	0.8447
	0.8789	0.8768	0.8448
	0.7892	0.7828	0.8167
T	0.7303	0.7003	0.7764
e	0.5839	0.5633	0.5290
c	0.7144	0.5492	0.7408
h	0.6673	0.5863	0.6604
	0.8378	0.5321	0.8820
	0.6622	0.5498	0.6913
	0.7744	0.6746	0.8008
	0.7169	0.6481	0.7165
	0.7109	0.6005	0.7247

Table 5.5 : AUC ROC for Bayesian Lasso-based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in **bold**)

	Topic Model text+people	Social Net. LDA
C	0.3784	0.7593
o	0.7280	0.9314
n	0.7004	0.7964
s	0.4492	0.6028
t	0.5659	0.7867
r	0.5944	0.6374
u	0.8025	0.8827
c	0.4149	0.5171
t	0.5317	0.6174
	0.5739	0.7257
	0.8721	0.8927
F	0.7295	0.7740
i	0.7790	0.8397
n	0.6048	0.7986
a	0.8534	0.8952
n	0.6742	0.7935
c	0.8447	0.8581
e	0.8719	0.9017
	0.7787	0.8442
	0.7439	0.7278
	0.3637	0.5673
T	0.3879	0.6560
e	0.5808	0.6090
c	0.3370	0.7788
h	0.6284	0.7025
n	0.7606	0.8545
	0.6614	0.7771
	0.5580	0.7091

Table 5.6 : AUC ROC for topic-based models with each row corresponding to a label and the last row showing the average AUC across all the labels (in **bold**)

5.6.3 Discussion

Tables 5.3 and 5.4 present the performance of the *BuiltOn* model which aimed at boosting the accuracy of the base classifiers: ridge regression and SVM. Across all the datasets, the *BuiltOn* model succeeded in improving the AUC scores, and in some cases the improvement was quite significant (e.g., Construction dataset in Table 5.3, Technology dataset in Table 5.4).

Tables 5.3 and 5.4 relate to the experiments carried out with the use of the full Bayesian models. Also here, across all three datasets, applying the vectorized representation for people associated with a document improved the AUC score, i.e., the AUC for the *Hybrid BL* model is higher when compared to Bayesian Lasso performed on TF text-based vectors and the TF ones extended by the binary representation of people. Comparing it to the the *BuiltOn* models, *Hybrid BL* method shows to be dataset dependent. In different cases it performed better, the same or worse then the *BuiltOn*.

The topic-based *Social Network LDA* model outperformed a simple topic-based classification model [29] for each dataset (here, the text dictionary was extended with senders and receivers). Comparing it to the other models, it is important to notice that in all the cases *Social Network LDA* did better than SVM classifier using text and people information (SVM text+people). Furthermore, *Social Network LDA*'s performance is the closest to the one of *Hybrid BL*.

Looking at the experimental results across all the methods, a few findings stand out. First, including the information about people in the form of the proposed vectorized representation, either in the *BuiltOn* model or the *Hybrid BL*, brings better performance when compared to the classification methods using the standard approach, i.e., ridge regression, SVM, and BL. Second, including correlation in the

models mostly improves the results even further. Third, it seems to be a bad idea to use the TF text-based vectors extended with the binary representation of people in a classification task. This approach never helped and sometimes even significantly decreased the classification accuracy (e.g., Table 5.5).

Overall, the results show that the proposed vectorized representation for people associated with a document can be successfully applied in multiple ways in different models to perform a document classification task.

Chapter 6

Conclusions

This PhD Thesis covers my work organized in three projects: Topic Models for Feature Selection in Document Clustering, Learning to Evaluate Student Programming Assignments in a Massive Open Online Course, as well as Senders, Receivers and Authors in Document Classification.

In the first project, Topic Models for Feature Selection in Document Clustering, I investigated the idea of using topic models such as Latent Dirichlet Allocation and its extensions as a feature selection in a pre-processing step for unsupervised document clustering, where documents are clustered in “topic space.” The first option I investigated was learning a topic model for a corpus, and then using a standard clustering algorithm to cluster the documents in topic space. I also investigated integrating a discrete mixture into the topic model itself, thereby learning the topic model and the clustering structure simultaneously. I proposed two variants of the second approach, one of which was found experimentally to be the best option.

In the second project, Learning to Evaluate Student Programming Assignments in a Massive Open Online Course, I considered the problem of using a statistical model to predict the score that a human, peer grader will give to a student-produced program in a MOOC. A set of simple distance measures was proposed that can be used to compute the distance between programs and/or program fragments, and found that this set of measures can be used to produce a regression model that is very accurate when predicting the average score that will be assigned to a program

by a set of human graders. I found that even a very simple k NN regression model can do a startlingly good job of assigning accurate scores, especially considering that the underlying program is never actually executed. A more sophisticated method that utilizes an ensemble of predictors was the best method overall.

In the third part of my work, Senders, Receivers and Authors in Document Classification, I proposed a novel vectorized representation of author properties of documents that improves classification performance in multiple-author scenario. Further, I developed three statistical models that apply this approach and evaluated them on three real email datasets coming from the eDiscovery process.

Bibliography

- [1] W.-F. Xuan, B.-Q. Liu, C.-J. Sun, D.-Y. Zhang, and X.-L. Wang, “Finding main topics in blogosphere using document clustering based on topic model,” in *ICMLC*, 2011.
- [2] X. Wang, J. Tang, and H. Liu, “Document clustering via matrix representation,” in *ICDM*, 2011.
- [3] D. Ramage, P. Heymann, C. D. Manning, and H. Garcia-Molina, “Clustering the tagged web,” in *WSDM*, pp. 54–63, ACM, 2009.
- [4] G. Tang, J. Pei, and W.-S. Luk, “Email mining: tasks, common techniques, and tools,” *Knowledge and Information Systems*, pp. 1–31, 2013.
- [5] Y. Suhara, H. Toda, S. Nishioka, and S. Susaki, “Automatically generated spam detection based on sentence-level topic information,” in *WWW*, pp. 1157–1160, ACM, 2013.
- [6] B. Pang and L. Lee, “Opinion mining and sentiment analysis,” *Foundations and trends in information retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.
- [7] F. Sebastiani, “Machine learning in automated text categorization,” *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.
- [8] M. J. Pazzani and D. Billsus, “Content-based recommendation systems,” in *The adaptive web*, pp. 325–341, Springer, 2007.

- [9] M. Hu and B. Liu, “Mining and summarizing customer reviews,” in *SIGKDD*, pp. 168–177, ACM, 2004.
- [10] S. Chakrabarti, M. van den Berg, and B. Dom, “Focused crawling: a new approach to topic-specific web resource discovery,” *Computer Networks*, vol. 31, no. 11-16, pp. 1623–1640, 1999.
- [11] H. M. Wallach, “Topic modeling: Beyond bag-of-words,” in *ICML*, pp. 977–984, ACM, 2006.
- [12] J. B. Lovins, *Development of a stemming algorithm*. MIT Information Processing Group, Electronic Systems Laboratory, 1968.
- [13] A. G. Jivani *et al.*, “A comparative study of stemming algorithms,” *Int. J. Comp. Tech. Appl*, vol. 2, no. 6, pp. 1930–1938, 2011.
- [14] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian data analysis*. CRC press, 2013.
- [15] L. Le Cam, “Maximum likelihood: an introduction,” *International Statistical Review*, vol. 58, no. 2, pp. 153–171, 1990.
- [16] C. M. Bishop *et al.*, *Pattern recognition and machine learning*, vol. 1. springer New York, 2006.
- [17] H. Attias, “A variational bayesian framework for graphical models,” *NIPS*, vol. 12, no. 1-2, pp. 209–215, 2000.
- [18] T. P. Minka, “Expectation propagation for approximate bayesian inference,” in *Conference on Uncertainty in artificial intelligence*, pp. 362–369, Morgan Kaufmann Publishers Inc., 2001.

- [19] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, “An introduction to mcmc for machine learning,” *Machine learning*, vol. 50, no. 1-2, pp. 5–43, 2003.
- [20] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*. Springer, 2010.
- [21] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *JMLR*, vol. 3, pp. 993–1022, 2003.
- [22] D. M. Blei, “Introduction to probabilistic topic models,” *CACM*, 2011.
- [23] T. Hofmann, “Probilistic latent semantic analysis,” in *UAI*, 1999.
- [24] P. J. Kanti V. Mardia, *Directional Statistics*. Wiley, 2000.
- [25] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra, “Clustering on the unit hypersphere using von mises-fisher distributions,” *JMLR*, vol. 6, pp. 1–39, 2005.
- [26] A. Banerjee and J. Ghosh, “Frequency sensitive competitive learning for clustering on high-dimensional hyperspheres,” in *IJCNN*, 2002.
- [27] J. Reisinger, A. Waters, B. Silverthorn, and R. J. Mooney, “Spherical topic models,” in *ICML*, 2010.
- [28] J. Atchison and S. Shen, “Logistic-normal distributions: Some properties and uses,” *Biometrika*, vol. 67, no. 2, p. 261, 1980.
- [29] D. Blei and J. McAuliffe, “Supervised topic models,” in *NIPS*, 2007.
- [30] J. Zhu, A. Ahmed, and E. P. Xing, “Medlda: Maximum margin supervised topic models for regression and classification,” in *ICML*, 2009.
- [31] S. Lacoste-Julien, F. Sha, and M. I. Jordan, “Disclda: Discriminative learning for dimensionality reduction and classification,” in *NIPS*, 2008.

- [32] A. Banerjee and S. Basu, “Topic models over text streams: A study of batch and online unsupervised learning,” in *SDM*, 2007.
- [33] S. Zhong and J. Ghosh, “Generative model-based document clustering: a comparative study,” *KAIS*, vol. 8, no. 3, pp. 374–384, 2005.
- [34] X. Wang, X. Ma, and E. Grimson, “Unsupervised activity perception by hierarchical bayesian models,” in *CVPR*, 2007.
- [35] M. M. Shafiei and E. E. Milios, “Latent dirichlet co-clustering,” in *ICDM*, 2006.
- [36] K. A. Rahman, S. Ahmad, and M. J. Nordin, “The design of an automated c programming assessment using pseudo-code comparison technique,” in *National Conference on Software Engineering and Computer Systems*, 2007.
- [37] N. Truong, P. Roe, and P. Bancroft, “Static analysis of students’ java programs,” in *Australasian Conference on Computing Education*, vol. 30, pp. 317–325, Australian Computer Society, Inc., 2004.
- [38] W. Wu, G. Li, Y. Sun, J. Wang, and T. Lai, “Analysec: A framework for assessing students’ programs at structural and semantic level,” in *ICCA*, pp. 742–747, IEEE, 2007.
- [39] K. A. Naudé, J. H. Greyling, and D. Vogts, “Marking student programs using graph similarity,” *Computers & Education*, vol. 54, no. 2, pp. 545–561, 2010.
- [40] T. Wang, X. Su, Y. Wang, and P. Ma, “Semantic similarity-based grading of student programs,” *Information and Software Technology*, vol. 49, no. 2, pp. 99–107, 2007.

- [41] F. Al Shamsi and A. Elnagar, “An intelligent assessment tool for students java submissions in introductory programming courses,” *Journal of Intelligent Learning Systems and Applications*, vol. 4, no. 1, pp. 59–69, 2012.
- [42] J. Blitzer, K. Q. Weinberger, and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” in *Advances in neural information processing systems*, pp. 1473–1480, 2005.
- [43] E. P. Xing, M. I. Jordan, S. Russell, and A. Ng, “Distance metric learning with application to clustering with side-information,” in *Advances in neural information processing systems*, pp. 505–512, 2002.
- [44] M. Schultz and T. Joachims, “Learning a distance metric from relative comparisons,” *Advances in neural information processing systems*, vol. 16, p. 41, 2004.
- [45] D. S. Witte and D. A. Portinga, “E-discovery and the new federal rules of civil procedure: They apply to you,” *Michigan Bar Journal*, vol. 86, no. 3, p. 36, 2007.
- [46] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, “The author-topic model for authors and documents,” in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 487–494, AUAI Press, 2004.
- [47] A. Drummond, Z. Vagena, and C. Jermaine, “Topic models for feature selection in document clustering,” in *SDM*, pp. 521–529, SIAM, 2012.
- [48] R. Segal, “Combining global and personal anti-spam filtering,” in *CEAS*, 2007.

- [49] A. Çıltık and T. Güngör, “Time-efficient spam e-mail filtering using n_i/i_i -gram models,” *Pattern Recognition Letters*, vol. 29, no. 1, pp. 19–33, 2008.
- [50] P. Bermejo, J. A. Gámez, and J. M. Puerta, “Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets,” *Expert Systems with Applications*, vol. 38, no. 3, pp. 2072–2080, 2011.
- [51] R. Bekkerman, A. Mccallum, G. Huang, and Others, “Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora,” *Center for Intelligent Information Retrieval, Technical Report IR*, vol. 418, 2004.
- [52] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, “An extensive experimental comparison of methods for multi-label learning,” *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, 2012.
- [53] N. Ueda and K. Saito, “Parametric mixture models for multi-labeled text,” in *Advances in neural information processing systems*, pp. 721–728, 2002.
- [54] S. Zhu, X. Ji, W. Xu, and Y. Gong, “Multi-labelled classification using maximum entropy method,” in *ACM SIGIR conference on Research and development in information retrieval*, pp. 274–281, ACM, 2005.
- [55] G. Salton, A. Wong, and C.-S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.
- [56] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, “Indexing by latent semantic analysis,” *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

- [57] T. Griffiths, M. Jordan, J. Tenenbaum, and D. M. Blei, “Hierarchical topic models and the nested chinese restaurant process,” *Advances in neural information processing systems*, vol. 16, pp. 106–114, 2004.
- [58] D. M. Blei and J. D. Lafferty, “Dynamic topic models,” in *Proceedings of the 23rd international conference on Machine learning*, pp. 113–120, ACM, 2006.
- [59] D. Blei and J. Lafferty, “A correlated topic model of science,” *Annals*, vol. 1, no. 1, pp. 17–35, 2007.
- [60] J. R. Millar, G. L. Peterson, and M. J. Mendenhall, “Document clustering and visualization with latent dirichlet allocation and self-organizing maps,” in *FLAIRS Conference*, vol. 21, pp. 69–74, 2009.
- [61] T. Kohonen, “The self-organizing map,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.
- [62] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, “On automated grading of programming assignments in an academic institution,” *Computers & Education*, vol. 41, no. 2, pp. 121–131, 2003.
- [63] D. Jackson and M. Usher, “Grading student programs using assyst,” in *ACM SIGCSE Bulletin*, vol. 29, pp. 335–339, ACM, 1997.
- [64] C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas, “The coursemarker cba system: Improvements over ceilidh,” *Education and Information Technologies*, vol. 8, no. 3, pp. 287–304, 2003.
- [65] D. S. Morris, “Automatic grading of student’s programming assignments: an interactive process and suite of programs,” in *Frontiers in Education*, 2003.

- FIE 2003 33rd Annual*, vol. 3, pp. S3F–1, IEEE, 2003.
- [66] R. Singh, S. Gulwani, and A. Solar-Lezama, “Automated feedback generation for introductory programming assignments,” in *PLDI*, pp. 15–26, 2013.
 - [67] M. Vujošević-Janičić, M. Nikolić, D. Tošić, and V. Kuncak, “Software verification and graph similarity for automated evaluation of students’ assignments,” *Information and Software Technology*, 2012.
 - [68] K. Ala-Mutka, T. Uimonen, and H.-M. Jarvinen, “Supporting students in c++ programming courses with automatic program style assessment,” *Journal of Information Technology Education*, vol. 3, no. 1, pp. 245–262, 2004.
 - [69] S. Schleimer, D. Wilkerson, and A. Aiken, “Winnowing: local algorithms for document fingerprinting,” in *SIGMOD*, pp. 76–85, 2003.
 - [70] C. Liu, C. Chen, J. Han, and P. S. Yu, “GPLAG: detection of software plagiarism by program dependence graph analysis,” in *KDD*, pp. 872–881, ACM, 2006.
 - [71] L. Prechelt, G. Malpohl, and M. Philippsen, “Finding plagiarisms among a set of programs with jplag,” *J. UCS*, vol. 8, no. 11, p. 1016, 2002.
 - [72] L. Jiang, G. Mishnerghi, Z. Su, and S. Glondu, “Deckard: Scalable and accurate tree-based detection of code clones,” in *ICSE*, pp. 96–105, IEEE Computer Society, 2007.
 - [73] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, “Bug isolation via remote program sampling,” in *PLDI*, pp. 141–154, 2003.

- [74] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, “Scalable statistical bug isolation,” in *PLDI*, pp. 15–26, 2005.
- [75] A. X. Zheng, M. I. Jordan, B. Liblit, M. Naik, and A. Aiken, “Statistical debugging: simultaneous identification of multiple bugs,” in *ICML*, pp. 1105–1112, 2006.
- [76] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, “The daikon system for dynamic detection of likely invariants,” *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 35–45, 2007.
- [77] A. V. Nori and R. Sharma, “Termination proofs from tests,” in *ESEC/SIGSOFT FSE*, pp. 246–256, 2013.
- [78] R. Sharma, S. Gupta, B. Hariharan, A. Aiken, and A. V. Nori, “Verification as learning geometric concepts,” in *SAS*, pp. 388–411, 2013.
- [79] G. Ammons, R. Bodík, and J. R. Larus, “Mining specifications,” in *POPL*, pp. 4–16, 2002.
- [80] B. Klimt and Y. Yang, “The enron corpus: A new dataset for email classification research,” in *Machine learning: ECML 2004*, pp. 217–226, Springer, 2004.
- [81] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A bayesian approach to filtering junk e-mail,” in *Learning for Text Categorization: Papers from the 1998 workshop*, vol. 62, pp. 98–105, 1998.
- [82] H. Drucker, D. Wu, and V. N. Vapnik, “Support vector machines for spam categorization,” *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1048–1054, 1999.

- [83] G. Rios and H. Zha, “Exploring support vector machines and random forests for spam detection.,” in *CEAS*, 2004.
- [84] I. Bíró, J. Szabó, and A. A. Benczúr, “Latent dirichlet allocation in web spam filtering,” in *AIRWeb*, pp. 29–32, ACM, 2008.
- [85] T. R. Payne and P. Edwards, “Interface agents that learn an investigation of learning issues in a mail agent interface,” *Applied Artificial Intelligence*, vol. 11, no. 1, pp. 1–32, 1997.
- [86] W. W. Cohen, “Learning rules that classify e-mail,” in *AAAI Spring Symposium on Machine Learning in Information Access*, vol. 18, p. 25, California, 1996.
- [87] J. Rennie, “ifile: An application of machine learning to e-mail filtering,” in *KDD Workshop on Text Mining*, Citeseer, Citeseer, 2000.
- [88] J. D. Brutlag and C. Meek, “Challenges of the email domain for text classification,” in *ICML*, pp. 103–110, 2000.
- [89] S. Kiritchenko and S. Matwin, “Email classification with co-training,” in *CAS-CON*, p. 8, IBM Press, 2001.
- [90] S. Kiritchenko and S. Matwin, “Email classification with co-training,” in *CAS-CON*, pp. 301–312, IBM Corp., 2011.
- [91] Y. Koren, E. Liberty, Y. Maarek, and R. Sandler, “Automatically tagging email by leveraging other users’ folders,” in *SIGKDD*, pp. 913–921, ACM, 2011.
- [92] J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker, “A hybrid learning system for recognizing user tasks from desktop activities and email messages,” in *IUI*, pp. 86–92, ACM, 2006.

- [93] V. Keiser and T. G. Dietterich, “Evaluating online text classification algorithms for email prediction in tasktracer,” in *CEAS*, pp. 4–7, Citeseer, July 16-17 2009.
- [94] T. Tam, A. Ferreira, and A. Lourenço, “Automatic foldering of email messages: a combination approach,” in *Advances in Information Retrieval*, pp. 232–243, Springer, 2012.
- [95] Y. Yang, S. Yoo, F. Lin, and I.-C. Moon, “Personalized email prioritization based on content and social network analysis,” *IEEE Intelligent Systems*, vol. 25, no. 4, pp. 12–18, 2010.
- [96] M. Wang, Y. He, and M. Jiang, “Text categorization of enron email corpus based on information bottleneck and maximal entropy,” in *ICSP*, pp. 2472–2475, IEEE, 2010.
- [97] D. Aberdeen, O. Pacovsky, and A. Slater, “The learning behind gmail priority inbox,” in *LCCC: NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*, 2010.
- [98] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O’Reilly Media, inc., 2009.
- [99] D. M. Blei and J. D. Lafferty, “Visualizing topics with multi-word expressions,” *arXiv:0907.1013v1 [stat.ML]*, 07 2009.
- [100] L. Pappano, “The year of the mooc,” *The New York Times*, vol. 4, 2012.
- [101] R. M. Palloff and K. Pratt, *Building learning communities in cyberspace*. Jossey-Bass Publishers San Francisco, 1999.

- [102] J. Ferrante, K. J. Ottenstein, and J. D. Warren, “The program dependence graph and its use in optimization,” in *Symposium on Programming*, pp. 125–132, 1984.
- [103] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, “Comparing stars: On approximating graph edit distance,” *PVLDB*, vol. 2, no. 1, pp. 25–36, 2009.
- [104] P. M. Williams, “Bayesian regularization and pruning using a laplace prior,” *Neural Computation*, vol. 7, no. 1, pp. 117–143, 1995.
- [105] G. Casella and E. I. George, “Explaining the gibbs sampler,” *The American Statistician*, 167–174, 1992.
- [106] C. P. Robert and G. Casella, *Monte Carlo statistical methods*, vol. 128. Springer New York, 1999.
- [107] T. Park and G. Casella, “The bayesian lasso,” *JASA*, vol. 103, no. 482, pp. 681–686, 2008.
- [108] J. Davis and M. Goadrich, “The relationship between precision-recall and roc curves,” in *ICML*, pp. 233–240, ACM, 2006.