APPROXIMATING COVARIANCE MATRICES USING
LOW RANK PERTURBATIONS
WITH APPLICATIONS TO ACCENT IDENTIFICATION
AND SOCIAL NETWORK CLUSTERING

By

Jonathan Purnell

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major Subject: COMPUTER SCIENCE

Approved by the
Examinining Committee:

Malik Magdon-Ismail, Thesis Adviser

Sanmay Das, Member

Mark Embrechts, Member

Mohammed Zaki, Member

Rensselaer Polytechnic Institute
Troy, New York

August 2010
(For Graduation December 2010)
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>ACKNOWLEDGMENT</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Outline</td>
<td>3</td>
</tr>
<tr>
<td>2. Related Works and Contributions</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Math Preliminaries</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Gaussian Mixture Model</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Low Rank Gaussian Mixture Model</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Accent Identification</td>
<td>16</td>
</tr>
<tr>
<td>2.5 Clustering Social Networks</td>
<td>19</td>
</tr>
<tr>
<td>3. Low-Rank Gaussian Mixture Models</td>
<td>21</td>
</tr>
<tr>
<td>3.1 Covariance Matrix Approximations</td>
<td>22</td>
</tr>
<tr>
<td>3.1.1 Diagonal Approximation</td>
<td>23</td>
</tr>
<tr>
<td>3.1.2 Optimal Rank-k Decomposition</td>
<td>23</td>
</tr>
<tr>
<td>3.2 Low-Rank Perturbed Diagonal Decomposition (LRPDD)</td>
<td>25</td>
</tr>
<tr>
<td>3.2.1 Iterative Solution Via Conjugate Gradient Optimization</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 Experimental Results on Synthetic Data</td>
<td>31</td>
</tr>
<tr>
<td>3.2.2.1 Training Performance</td>
<td>32</td>
</tr>
<tr>
<td>3.2.2.2 Runtimes</td>
<td>34</td>
</tr>
<tr>
<td>3.2.3 Experimental Results on Real Data</td>
<td>35</td>
</tr>
<tr>
<td>3.2.3.1 Datasets</td>
<td>36</td>
</tr>
<tr>
<td>3.2.3.2 Training Performance</td>
<td>36</td>
</tr>
<tr>
<td>3.3 Discussion</td>
<td>37</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>In-sample average log probabilities for T=10k,M=6 (div. by 1000)</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>In-sample average log probabilities for T=1k,M=6 (div. by 100)</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>In-sample average log probabilities for T=10k,M=3 (div. by 1000)</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Out-sample average log probabilities T=1k,M=6 (div. by 1000)</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Runtimes using the Full matrix and the Perturbed approximation with T10k</td>
<td>35</td>
</tr>
<tr>
<td>3.6</td>
<td>Percentage accuracy of GMMs on speech samples</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Speaker Accuracies of Baselines and GMM</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Accuracies of GMM and the Extensions</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Cluster Quality</td>
<td>68</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

3.1 Alphabet data fitted by GMMs with Full and Diagonal Matrix ........................................ 24  
3.2 Alphabet data fitted by GMMs with LRPDD ................................................................. 28  
3.3 Differences in approximating a matrix with the LRPDD and Diagonal Methods .......................... 32  
3.4 Relation between runtimes and sample dimension for Low Rank Perturbed ......................... 34  
3.5 Relation between runtimes and sample dimension for Diagonal ........................................ 35  
3.6 Relation between runtimes and sample dimension for Full Covariance Matrix .......................... 36  
4.1 How the speech is pre-processed to produce MFCCs ......................................................... 43  
5.1 Average change in Log-Prob of adding a cluster on Real Data and Random Data. Shaded region shows first standard deviation of the change. ......................................................... 63  
5.2 $\alpha$ vs. overlap. ........................................................................................................... 64  
5.3 $\alpha$ vs. gain in cluster metric. ......................................................................................... 64  
5.4 Keyword Clouds for the 7 clusters ..................................................................................... 67  
5.5 Cluster notation for overlap quality .................................................................................... 68  
5.6 Conference Clouds for the 7 clusters .................................................................................. 70  
5.7 LDA Keyword Clouds for 7 clusters .................................................................................. 71  
5.8 Word/Venue Clouds for the three major intersections (word clouds on the left and venues on the right) ........................................................................................................... 72
ACKNOWLEDGMENT

My ability to produce this thesis has been dependent on the help and support of many individuals. First of all, I would like to thank my advisor, Malik Magdon-Ismail, for his mentorship over the years. Not only has he been a vital part of my work in graduate school, but he was willing to help introduce me to machine learning during my undergraduate years when the related courses were unavailable. I have benefited and been inspired by his intellect and willingness to teach and discuss concepts in both a broad and detailed sense.

I would also like to thank the other members of my doctoral committee: Sanmay Das, Mark Emberchts, and Mohammed Zaki. Their expertise and advice helped significantly to make this thesis complete and thorough.

As much as an excellent committee is necessary for academic support, so are excellent friends necessary for moral support. The many hours spent with my lab-mates, inside and outside of the lab, have helped me appreciate the value of friendship and broaden my perspective of the world. Over the years, I have had the pleasure of working along side Ali Çivril, Eli Bocek-Rivele, Mykola Hayvanovich, Andre Sarayev, and Julia Sarayev. I would also like to thank the support of my friends in the Graduate Christian Fellowship and the Third Presbyterian Church of Troy, who have been an invaluable part of my life in Troy and as a Christian PhD student. In addition, my friends from home, especially Sean Lanfear, have been tremendously dependable and helped me in more ways than I could count or remember.

Finally, I would like to thank my loved ones: my fiancée Sarah, my brother Chris, and my parents Dave and Dianne. Their endless and self-sacrificing support, tenacious encouragement, and abundant love have been a foundation from which I have found assurance and the drive to grow. I would not have made it this far, if not for them. This work is dedicated to them.
ABSTRACT

In this work, we present a new model, the Low-Rank Gaussian Mixture Model (LRGMM), for modeling data which can be extended to identifying partitions or overlapping clusters. This model is motivated by the effectiveness, yet limited scalability, of the Gaussian Mixture Model (GMM) for the problem of accent identification. The curse of dimensionality that arises in calculating the covariance matrices of the GMM is countered by using low-rank perturbed diagonal matrices. We also demonstrate the LRGMM for finding communities in social networks. We see that the efficiency of the LRGMM allows us to process larger networks than alternative approaches. Altogether, the LRGMM experiments reveal it to be an efficient and highly applicable tool for working with large high-dimensional datasets.
CHAPTER 1
Introduction

1.1 Overview

Fitting models to a dataset or clustering samples based on some metric are useful tools in better understanding internal structures (e.g. communities) or identifying the underlying distribution of the data. With the rise of Internet-based social networking, there are increasingly larger and larger datasets to process. In addition, the loss of information by dimension reduction is not always balanced by the decrease in runtime of clustering and modeling algorithms. In this work we present a new model and algorithm, called the Low-Rank Gaussian Mixture Model (LRGMM), that allows for a decrease in the runtime with respect to the dimensionality of the data. The motivation is to design a model that can be efficiently used for large datasets in high dimension.

We address one of the main difficulties in applying GMM’s to data, namely the statistical and computational complexity introduced by the quadratic dependence on $d$, the data dimension. For each mixture within the GMM, a covariance matrix must be calculated, updated, and inverted. As $d$ increases, they can quickly become cost-prohibitive. In order to address these difficulties, we introduce the LRGMM model.

Our first application was accent identification, which is known to be hard in that the systems of speech and hearing are complex and difficult to model. Often a priori knowledge, in the form of transcripts and expert evaluations, is heavily relied on for speech processing and accent identification. Recently, the Gaussian Mixture Model (GMM) has been studied as a replacement for the commonly used phoneme-based Hidden Markov Model (HMM). Both the advantage and disadvantage for the GMM over the phoneme-based HMM is that it does not use a priori knowledge. We show that when coupled with ensemble learning approaches, the GMM performs well given the lack of additional information.

In the case of speech processing, additional dimensions in the data correspond
to more precise information. When using typical methods of speech processing, we see that more dimensions means more detail and accuracy in the amplitudes of the speech at different frequencies, yielding a more comprehensive picture of the speech harmonics, a feature comparable to a fingerprint. The runtime of the GMM is quadratic in $d$, which is due to the calculation of the sample-based covariance matrix for each mixture of the model. A common heuristic to address this problem is to reduce the number of dimensions. However, common techniques to reduce dimensionality, such as principal components analysis, already have $O(Nd^2)$ complexity and so this does not address the computation issue; in addition, information would be lost. In general, the reason to use dimensionality reduction is to throw away spurious or redundant information to help curb overfitting. In the speech problem, overfitting is not as much of a problem as data is abundant. Thus, we would like to improve efficiency while maintaining the dimension. LRGMM provides one such route.

The LRGMM model preserves most of the structure of the full sample-based covariance matrix while maintaining a training time that is linear in $d$. Thus, it is computationally comparable to the diagonal approximation of the covariance matrix while yielding a more accurate model. We also demonstrate that the LRGMM does not succumb as much to overfitting the data, since the number of parameters remains linear in $d$, as opposed to quadratic for the full covariance matrix. Despite the complex theory behind this approximation, the implementation is straight-forward and relies on the conjugate gradient method, which itself has very efficient implementations.

With the LRGMM model in hand, we move towards another application: finding communities in social networks. For this, we combine the LRGMM with Sampled Spectral Distance Embedding (SSDE) to take an input weighted graph representing the relations between entities and output communities. The SSDE efficiently embeds the weighted graph into Euclidean space while minimizing the loss of information. From there, the LRGMM clusters the embedded data to find the communities. A simple heuristic allows the clusters to be overlapping and improve the accuracy of their membership. A major contribution to this task is the efficiency of both the
SSDE and LRGMM. Thus the method can work on larger and denser networks than many alternative algorithms. As a single data point, we construct meaningful overlapping communities in DBLP (1 million nodes; 30+ million edges) in under 20 minutes.

1.2 Outline

This thesis consists of three distinct topics: the LRGMM model, GMMs for accent identification, and finding communities in social networks. All are linked through the use of covariance matrices.

1. In Chapter 3, we motivate and describe the LRGMM model. The details of the model and the method of fitting it are explained. The accuracy and runtime performances of the LRGMM are then compared to using full covariance matrices and diagonal matrix approximations. This comparison is performed on both synthetic and real-world data. From the results we see that the LRGMM manages to find a better balance between accuracy and runtime than the diagonal matrix or full covariance matrix. Further, we show that the LRGMM is less inclined towards overfitting as the full covariance matrix. The technical details of this model are discussed in Chapter 3, and related to the work in [60].

2. Elaborated on in Chapter 4, we review the problem of accent identification. We discuss the speech dataset and how it is processed. We then explore the effectiveness of the GMM for this problem. The GMM is compared to alternative algorithms, such as the linear SVM and phoneme-based HMM. We also explore coupling the GMM with various ensemble learning approaches. All of these methods are compared based on how accurately they can classify Northern and Southern American English speakers. This work is elaborated in Chapter 4 and related to the works published in [56] and [59].

3. Finally, we couple the LRGMM with the SSDE algorithm and apply it to the task of finding communities in social networks. In addition to experimenting with its effectiveness, we look into the task of using the LRGMM for clustering.
We explore the heuristics of finding the optimal number of clusters and the proper level of overlap between clusters. This topic is detailed in Chapter 5. It is also related to the work in [61].

We then conclude with a summary of the experimental results and open problems that remain.
CHAPTER 2
Related Works and Contributions

2.1 Math Preliminaries

Our work uses various mathematical functions and tools that are common within the field, but are often not made explicit. For completeness, we describe some here.

*Fourier Transform*[52]: Given a continuous one-dimensional function, \( x(t) \), the Fourier transform is defined by

\[
X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt,
\]

where \( \omega \) is the angular frequency and \( j \) is the imaginary unit. The inverse of the Fourier transform is defined by

\[
x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega.
\]

Sometimes the equation is written in terms of oscillation frequency, \( k = \frac{\omega}{2\pi} \), instead of the angular frequency, \( \omega \).

The Fourier transform for discrete-time signals is defined by

\[
X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n},
\]

where the variables are the same as in the continuous equation and \( n \) is the time frame index. The inverse of this transform is

\[
x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega.
\]

*Short Term Fourier Transform (STFT)*[11]: STFT, applies a window function, \( g \), to the input signal so that the results of the Fourier transform pertain to a certain points in the time domain. The only constraint on \( g \) is that it is square integrable.
A STFT that uses a Gaussian window is called the Gabor transform.

The continuous version of the STFT is defined by

\[ G_x(\tau, \omega) = \langle x, g_{\omega}, \tau \rangle = \int x(t) g(t - \tau) e^{-j\omega t} dt, \]

where \( x(t) \) is the input signal, \( j \) is the imaginary unit, \( \tau \) is the temporal shift of the window, and the bar signifies the complex conjugate. The inverse of this transform is defined by

\[ x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} G_x(\omega) e^{-j\omega t} d\omega. \]

For the discrete time-signal, the STFT and its inverse are similarly defined by

\[ G_x(m, \omega) = \sum_{n=-\infty}^{\infty} x[n] g[n - m] e^{-j\omega n}, \quad (2.1) \]

\[ x[n] = \frac{1}{2\pi\|g\|^2} \int_{2\pi} G_x[\omega] g[n] e^{-j\omega n} d\omega, \quad (2.2) \]

where \( n \) is the time frame variable, \( m \) is the discrete temporal shift of the window and \( j \) is the imaginary unit.

Hyperplanes: A hyperplane is a geometric concept that is a generalized concept of a plane. In one-dimensional space (such as a line), a hyperplane is a point that divides a line into two rays. In 2D, the hyperplane is a line that divides the plane into half-planes. The concept can be applied in the same way to higher dimensions. Formally, the hyperplane is an affine subspace of codimension 1.

Frobenius Matrix Norm: This is also called the Euclidean norm and is defined for a matrix \( A \) by

\[ \|A\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2} \]

or

\[ \|A\|_F = \sqrt{\text{Tr}(AA^H)} \]

where for a \( n \times n \) matrix \( B \), \( \text{Tr}(B) \equiv \sum_{i=1}^{n} b_{ii} \).

Viterbi Algorithm: The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states, known as the Viterbi path, that result in a sequence of observed events in a Hidden Markov Model. The
Viterbi path then has a probability value that depends on the state transition probabilities and the observation distribution at each state. This probability of the Viterbi path represents the probability that the given Markov model generated the sequence of observed events.

For the Viterbi algorithm, the dynamic programming variable \( \delta_t(i) \) is used to determine the maximum probability of all sequences ending at state \( i \) at time \( t \). This variable is evaluated for all states over all time in a forward fashion. For time \( t = 1 \),

\[
\delta_1(i) = \pi(i) b_{ik_1},
\]

where \( \pi(i) \) is the probability that state \( i \) is the first state of the Viterbi path and \( b_{ik_1} \) is the probability that event \( k_1 \) is observed at state \( i \). For time \( t > 1 \), the value of \( \delta_t(i) \) is based on the probability that event \( k_t \) is observed at state \( i \), the probability of transitioning from any of the states to state \( i \) and the probability of those states at time \( t - 1 \). The recursive formula is given by

\[
\delta_t(i) = \max_j (\delta_{t-1}(j)a_{ji}b_{ik_t}),
\]

where \( a_{ji} \) is probability of transitioning from state \( j \) to state \( i \). To keep track of the best sequence of states, for each state at each time, a second function \( \phi \) is defined by

\[
\phi_t(i) = \arg \max_j (\delta_{t-1}(j)a_{ji}).
\]

When the values for \( \delta_n(i) \), where \( n \) is the maximum time, have been calculated, the last state in the Viterbi path is determined to be the state \( i \) that has the maximum value for \( \delta_n(i) \). The second to last state in the Viterbi path is then determined by \( \phi_n(i) \). The rest of the Viterbi path is constructed in a similar way, where the state \( j \) at time \( t \) is determined by the value of \( \phi_{t+1}(i) \), where \( i \) is the state of the Viterbi path at time \( t + 1 \).
2.2 Gaussian Mixture Model

The Gaussian Mixture Model (GMM) is ubiquitous both in general and within computer science. It has been applied to a variety of problems ranging across topics such as clustering, image segmentation, computational finance, speech recognition, and biometrics. It is also the foundation of our work presented in this thesis, so we discuss this model in depth.

There are several parametric and non-parametric methods for modeling data sets. While parametric models are quick to evaluate, they require a search through the parametric space for optimal values and are constrained by the underlying distributions. Non-parametric methods often yield better approximations by having a very general form but grow in complexity as the training set grows. This causes evaluation to be slow when training on larger sets. A mixture model can be viewed as a compromise between these two approaches, where the number of parameters can vary but is not directly tied to the size of the training set. Formally, a mixture model is a linear combination of component densities $p(x|i)$ and is defined by

$$p(x) = \sum_{i=1}^{M} p(x|i) \pi_i. \quad (2.3)$$

For the GMM, the component densities are Gaussian (Normal) densities, where component $i$ is specified by its center, $\mu_i$, and variance, $\sigma_i^2$,

$$p(x|i) = \frac{1}{(2\pi\sigma_i^2)^{\frac{d}{2}}} \exp \left\{ -\frac{||x - \mu_i||^2}{2\sigma_i^2} \right\}. \quad (2.4)$$

Each component is weighted by a prior probability, $\pi_i$, of a data sample having been generated by the $i^{th}$ component. They are constrained by $\sum_{i=1}^{M} \pi_i = 1$ and $0 \leq \pi_i \leq 1$. Each component is also defined by a mean vector, $\mu_i$ and a variance, $\sigma_i^2$. In our work we use multivariate Gaussian densities, but for our purposes of explaining the model we will focus on univariate densities and describe the extension for full covariance matrices later. By Bayes’ theorem, we can express the posterior probabilities

$$P(i|x) = \frac{p(x|i)\pi_i}{p(x)}, \quad (2.5)$$
which represents the probability that component \( i \) generated the data sample \( x \). The posterior probabilities satisfy \( \sum_{i=1}^{M} P(i|x) = 1 \).

Given a set of data samples, \( x_t \), the goal is to use them to calculate the parameters of our GMM. Specifically, we aim to maximize the likelihood of our parameters. The likelihood of a GMM is defined by

\[
\mathcal{L} = \prod_{t=1}^{T} p(x_t). \tag{2.6}
\]

Since we are working with Gaussian densities, we rewrite (2.6) into an error function to be minimized,

\[
E = -\ln \mathcal{L} = -\sum_{t=1}^{T} \ln p(x_t) = -\sum_{t=1}^{T} \ln \left\{ \sum_{i=1}^{M} p(x_t|i) \pi_i \right\}. \tag{2.7}
\]

By minimizing this error function, we are maximizing the likelihood of our parameters. However, it should be noted that this can be a difficult task since there are local minima for smaller groups of outlier data samples that are close together due to noise. Further, a component of the GMM can generate infinite likelihood by centering on a single data point, such as an outlier, and reducing the covariance to zero.

The first step we take in the maximum likelihood approach is to find the derivatives of the error function with respect to each of the parameters. Since the error function is a smooth differentiable function of the parameters, we aim to find an analytic solution. For the mean vectors, \( \mu_i \), and variances, \( \sigma_i^2 \), we can differentiate (2.7), after substituting in (2.4) and (2.5),

\[
\frac{\partial E}{\partial \mu_i} = \sum_{t=1}^{T} P(i|x_t) \frac{(\mu_i - x_t)}{\sigma_i^2} \tag{2.8}
\]

\[
\frac{\partial E}{\partial \sigma_i} = \sum_{t=1}^{T} P(i|x_t) \left\{ \frac{d}{\sigma_i} - \frac{||x_t - \mu_i||^2}{\sigma_i^3} \right\} \tag{2.9}
\]

Setting these derivatives to zero and moving the parameters to one side, we obtain
equations for the optimal parameter values

\[ \hat{\mu}_i = \frac{\sum_{t=1}^{T} P(i|\mathbf{x}_t) \mathbf{x}_t}{\sum_{i=1}^{T} P(i|\mathbf{x}_t)} \]  

(2.10)

\[ \hat{\sigma}^2_i = \frac{1}{d} \frac{\sum_{t=1}^{T} P(i|\mathbf{x}_t)||\mathbf{x}_t - \hat{\mu}_i||^2}{\sum_{i=1}^{T} P(i|\mathbf{x}_t)} \]  

(2.11)

\[ \hat{\pi}_i = \frac{1}{T} \sum_{i=1}^{T} P(i|\mathbf{x}_t) \]  

(2.12)

where (2.12) can be derived from (2.7) by using the softmax function and chain rule[7].

The maximum likelihood provides great insight into the properties of the optimal parameters, but is not directly applicable as a solution. Due to the posterior probabilities in (2.5), maximum likelihood provides non-linear coupled equations. One way to optimize the parameters is to use a gradient descent algorithm (such as conjugate gradients) on (2.8) and (2.9). Also, the maximum likelihood results can be used to build an iterative solution for minimizing the error function. Given a set of parameters, we can use (2.5) to calculate the posterior probabilities. Holding the posterior probabilities constant, we can then update the GMM parameters using (2.10), (2.11), and (2.12). This becomes a special case of a general procedure known as the expectation-maximization (EM) algorithm [20].

The EM approach to training the GMM begins by choosing the initial parameters of the model. This can be by randomly choosing data points for \( \mu_i \), setting \( \sigma^2_i = 1 \), and setting the priors \( \pi_i = \frac{1}{M} \). There are better alternatives, which will be discussed later. With our initial parameters, we can solve (2.5) for all \( \mathbf{x}_t \) and \( i = 1, \ldots, M \), using (2.3) and (2.4). Calculating the posterior probabilities constitutes the expectation step of the EM approach. The next step is to maximize the likelihood of our parameters with the given posterior probabilities. This is where we use (2.10), (2.11), and (2.12) to calculate our new parameters.

With a set of parameters, we can determine the error by (2.7). The change in error between two successive iterations of the EM approach can be written

\[ E^{\text{new}} - E^{\text{old}} = -\sum_{t=1}^{T} \ln \left\{ \frac{p^{\text{new}}(\mathbf{x}_t)}{p^{\text{old}}(\mathbf{x}_t)} \right\} \]  

(2.13)
where \( p_{\text{new}}(x_t) \) is the probability of \( x_t \) using the most recent parameters and \( p_{\text{new}}(x_t) \) is the probability using the next most recent parameters. Using (2.3) we can rewrite this equation as

\[
E_{\text{new}} - E_{\text{old}} = -\sum_{t=1}^{T} \ln \left( \frac{\sum_{i=1}^{M} \pi_i^{\text{new}} p_{\text{new}}(x_t|i) P_{\text{old}}(i|x_t)}{p_{\text{old}}(x_t)} \right),
\]

(2.14)

where the added term is an identity term. We can then apply Jensen’s inequality, which states that when given a set of numbers \( \lambda_i \geq 0 \) such that \( \sum_i \lambda_i = 1 \),

\[
\ln \left( \sum_i \lambda_i x_i \right) \geq \sum_i \lambda_i \ln(\lambda_i).
\]

(2.15)

Since \( \sum_{i=1}^{M} \lambda_i = 1 \), they can be used as \( \lambda_i \) in (2.15)

\[
E_{\text{new}} - E_{\text{old}} \leq -\sum_{t=1}^{T} \sum_{i=1}^{M} P_{\text{old}}(i|x_t) \ln \left( \frac{\pi_i^{\text{new}} p_{\text{new}}(x_t|i)}{P_{\text{old}}(i|x_t)} \right).
\]

(2.16)

Let us view this equation as \( E_{\text{new}} \leq E_{\text{old}} + Q \), where \( Q \) now represents the right hand side of (2.4). By ensuring \( Q \leq 0 \), we can guarantee \( E_{\text{new}} \leq E_{\text{old}} \). The proof of this can be seen in [7].

To this point we have discussed a GMM with simple variances. In higher dimensions, covariance matrices are used instead of variances. This has little impact on the derivations discussed, but changes (2.11) to

\[
\hat{\Sigma}_i = \frac{1}{d} \frac{\sum_{t=1}^{T} P(i|x_t)(x_t - \hat{\mu}_i) \otimes (x_t - \hat{\mu}_i)}{\sum_{i=1}^{T} P(i|x_t)},
\]

(2.17)

and (2.4) becomes

\[
p(x|i) = \frac{1}{(2\pi \hat{\Sigma}_i)^{d/2}} \exp \left\{ -\frac{1}{2} (x_t - \hat{\mu}_i) \hat{\Sigma}_i^{-1} (x_t - \hat{\mu}_i) \right\}.
\]

(2.18)

In practice the EM approach can get stuck in a local minima, where \( Q \approx 0 \), that is far from the true distribution of the data. This is heavily influenced by the initial parameters. With the exception of toy datasets, random initial parameters will be susceptible to these local minima and require several runs of the EM approach.
to find initial parameters that give desirable results. A more common solution for initial parameters is to use $K$-means clustering. With $K$-means clustering we can quickly find initial clusters and their centers. The data from each cluster can then be used to determine the covariance matrix and mixture weights. However, since the $K$-means clustering also requires initial centers, we still have problems with the local minima. An additional measure is to use $K$-means with $M \log M$ random initial centers and ignore the $\log M$ smallest clusters [19]. While these approaches will not eliminate the problem of local minima, which is a byproduct of our error function, in practice they will greatly reduce the chance that the local minima will be far from the true distribution of the data.

In addition to initial parameters, there has been a great deal of work on alternative versions of the EM approach. One such version that was developed early was the Incremental EM [47] approach. It was noticed that the posterior probability of a single data point contained information that could be used to improve the parameters. For Incremental EM, each expectation step only calculates the posterior probability for a single data point. Using sufficient statistics to represent the remainder of the data [8], the parameters are then improved in a modified maximization step. A single iteration of the standard EM approach is then equivalent to $T$ iterations of the Incremental EM approach, where each data point is visited exactly once (sometimes called a “pass”). While this allows an online-like approach, a “pass” ends up taking about twice as much computation time as a single iteration of standard EM.

Another approach, Sparse EM [47], took a different look at the problem. Instead of iterating over the entire dataset to update the parameters, Sparse EM was designed to update the parameters of a mixture using subsets of the data. When the posterior probabilities for a data sample are calculated, it can be seen that the data sample contributes more to certain mixtures than to others. It is therefore desirable to remove the computations of posterior probabilities that are relatively negligible to their respective mixtures. Various heuristics can be used to determine if a probability is “negligible”. Sparse EM starts off by computing all posterior probabilities, and “freezing” those that are found to be “negligible”. On each ex-
pectation step, only the “unfrozen” posterior probabilities are updated. Every few iterations, all posterior probabilities are recomputed and then those found to be negligible are frozen. This approach can also be combined with Incremental EM, where a posterior probability may be frozen for several passes, thus reducing the computation time per iteration.

The work in Incremental EM inspired another alternative EM approach called Batch EM [9], which was designed to address the problem of training on large datasets and the overhead cost of reading data into memory. Batch EM starts by loading enough data samples to fill the memory, minus space for computations. Initially, Batch EM finds dense clusters using $K$-means clustering. A large number of clusters, greater than $M$, are found. Those clusters whose covariance diagonals fall below a particular threshold (i.e. are sufficiently dense) are kept as clusters and the respective data points are replaced by sufficient statistics. On each iteration of Batch EM, more samples are read into memory until it is full. Then, those data points that are sufficiently close to the clusters previously found are used to update those clusters’ sufficient statistics and the data samples are removed from memory. The remaining data is clustered by $K$-means in order to find new clusters, as in the initialization. These two steps are repeated each iteration until all the data has been loaded into memory at least once. The approach can suffer depending on the order that points are loaded into memory, but allows for a more efficiently EM approach when dealing with large data sets and limited resources.

Another approach was designed that uses multi-resolution kd-trees [46]. This algorithm starts by recursively dividing the dataset into two parts along some “splitting” direction. This is represented by a tree, where the root node contains all the data. Each node has two children nodes that are defined by a splitting direction and a dividing point along that direction. One child node receives the points on one side of this dividing point, and the other child receives the remaining data points. The divisions continue until there a child node has only one data point or the volume containing the node’s data samples decreases past some threshold. At each node, the sufficient statistics of it’s corresponding data samples is computed. This is done for all nodes in a bottom-up manner. When a non-leaf node is processed, if its
sufficient statistics are close enough to those of its child nodes, then the child nodes are pruned and the non-leaf node becomes a new leaf node. The EM approach is then performed for each leaf node, and the non-leaf nodes are again processed to determine if pruning is possible. After several iterations, the leaf nodes are used to determine the clusters.

Another constraint on the GMM is the number of components. Using too many or too few components can degrade the performance of the GMM by combining or breaking up the true clusters of data. We have seen that Batch EM has an approach that gets around this constraint. Another approach is to overguess the number of clusters and, either during or after training, prune off smaller, unwanted clusters based on some heuristic. A recent proposed variation of the GMM is the Infinite GMM [?], which uses the infinite hierarchical Bayesian mixture model. Working with efficient parameter-free Markov chains that rely entirely on Gibbs sampling, the Infinite GMM provides a model that yields comparable results to other approaches.

2.3 Low Rank Gaussian Mixture Model

There are many methods of matrix approximation, each having their advantages and drawbacks. One common approach to matrix approximation is by decomposition. In [55], the author notes the difficulty of estimating positive definite matrices and uses decompositions to address the problem. The Cholesky decomposition is shown as a better decomposition than variance-correlation or spectral decomposition. However, the time complexity of the Cholesky decomposition is quadratic with respect to the dimension of the matrix.

Alternatively to decomposition, El Karoui uses a mixture of statistics and machine learning. In [23], El Karoui estimates the spectrum of large dimensional covariance matrices. These are covariances of large datasets in high dimension where the ratio of the two is constrained to a finite non-zero limit. Instead of estimating the eigenvalues individually, the vector of eigenvalues is treated as an observation of a probability distribution. Using random matrix theory and convex optimization, this probability distribution is estimated and used to approximate the eigenvalues. The resulting estimates of these eigenvalues can be used in common algorithms such
as PCA.

Some matrix approximations work well but are constrained to certain types of matrices. In [13], the authors proposed an iterative method of estimation using the constraint that some elements of the covariance matrix are zero. While effective, the constraint makes the approach unsuitable for the covariance matrices of a GMM.

The sample covariance is the covariance of the samples from a particular class. The common covariance is the weighted sum of sample covariances. When working with a limited amount of data, [38] finds that a mixture of these matrices and their diagonals can be a robust approximation of the covariance matrices.

These previous approaches each suffer from one of two drawbacks: they have a time complexity that grows quadratically with the data dimension or they cannot be efficiently inverted. This makes them ill-suited for GMMs. The LRGMM model we propose uses a matrix decomposition that avoids both of these drawbacks by aiming to approximate the inverse of the covariance matrix directly. This decomposition involves using a rank-1 perturbation of a diagonal matrix. It efficiently uses the samples so that the time it takes to approximate the inverse covariance matrix is linear in \(d\), the data dimension. Thus, the rank-1 perturbed diagonal matrix has a lower time complexity and only requires that the covariance matrix is sample-based. This makes the approximation very suitable for the GMM and for other tasks that use a sample-based covariance matrix or its inverse.

**Our Contribution:** We have proposed a new matrix decomposition to approximate the covariance matrices of a Gaussian Mixture Model. Instead of using the diagonal, we use a low-rank perturbation of a diagonal matrix and approximate the inverse of the covariance matrix, rather than the matrix itself. The conjugate gradient method is used to optimize the parameters with respect to the log probability formula of a GMM. By exploiting the structure of the covariance matrix and approximating its inverse, we developed an algorithm that is bounded linearly with respect to sample dimension. This makes its computational cost comparable to the that of the diagonal approximation.

In our experimental results we compared our approximation method to the diagonal approximation and the full covariance matrix. The in-sample accuracies
show that while the perturbed matrix is not an exact approximation, it is consistently better than the diagonal as expected. Further, the out-sample accuracy shows that this improved flexibility does not result in overfitting, since the perturbed approximation also out-performs the diagonal approximation in this respect as well. A look into the runtimes of these experiments provides reinforcement that our method has a comparable computation cost to the diagonal approximation, with respect to sample dimension.

Although more complex in theory, the implementation of our method is not much more involved than implementing the diagonal matrix. Since the conjugate gradient method has very efficient implementations, this implementation is also efficient.

In the future, it may be possible to lower the absolute computational cost of the perturbed approximation by finding the optimal values and thresholds for the conjugate gradient method. Also, one could look into using various ranks of perturbation. Intuitively, it seems that all perturbations from rank 1 to full rank can be similarly derived and provide a greater flexibility in finding the balance between computational cost and accuracy.

### 2.4 Accent Identification

Speaker identification involves modeling the processed speech from a speaker to be identified. When done successfully, their speech can be distinguished from that of other speakers. Accent identification can be thought of as an extension of speaker identification, in that we are modeling the processed speech from a large group of people in order to distinguish their speech. In [62], a Gaussian Mixture Model (GMM) was successful (80.8%) in identifying individual speakers with 15 seconds of speech over the telephone. The success of the GMM was due to its ability to capture spectral characteristics that make the speaker's voice unique from other speakers. In the task of accent identification, the goal is to find the spectral characteristics that uniquely identify the voices of a particular group. The challenging aspect of this problem is that the spectral characteristics of two groups is often more alike than one would observe between two individual speakers.
There is some ambiguity in the term ‘accent’, since this can either mean the way a speaker pronounces their primary language or a secondary language. For example, we might notice that a speaker who’s primary language is German speaks English with a German accent (e.g. when they use the ’v’ sound instead of the ’w’ sound). Similarly, we may say that someone from one of the Southern states in the US has a Southern accent, when they exhibit what’s known as the ’southern drawl’. In this thesis, we use the term accent to mean how the speaker pronounces their primary language, such as a Southern accent from the previous example. While there are several possible factors that contribute to its formation, this type of accent is predominantly correlated with geographical location [42].

Nevertheless, all our work could be applied to the other case of accent (in the secondary language). Accents arising in a secondary language are due to the primary languages set of phonemes. Accents arising in a primary language are due to modifications to a common set of phonemes. In other words, the differences primary language accents have much more subtle identifying characteristics than the differences between secondary language accents. As another example, it is often quite easy to identify that someone speaking English has a different primary language. In contrast, the difference between a Carolina accent and a Southern accent or a Northern accent is much more subtle.

The most common model used in speech recognition, and also in speaker and accent identification, is the phoneme-based Hidden Markov Model (HMM). Although effective and practical for dealing with a single speaker, the HMM can become costly to train when working with groups of speakers. This is in part due to the processing of larger amounts of data needed for training. It is also due to the requirement of transcripts for training the HMM, which, among other things, identifies when a word starts and ends. These transcripts require some human supervision and thus add to the cost of gathering the necessary data. There has been recent work in using other models that do not require this type of a priori knowledge.

Zheng [75] attempts to discriminate between accents of Mandarin, using properties known a priori of Shanghai-accented Mandarin. The speakers were separated by experts into two groups: those having little accent and those having a strong
accent. The two groups are then modeled and tested separately. They report an identification accuracy of 69% on the weakly accented group and 86% on the strongly accented group. Zheng does not report results when using all the data to train accent classifiers, so it is not clear how to avoid the human expert who initially classifies the data into weak and strong groups.

In another attempt to avoid building a phonetic-based model, Chen [14] used supervised GMMs on the standard features, Mel-frequency cepstral coefficients (MFCCs). As we do in our work, each accent is modeled by a GMM. During testing, the GMMs are used to determine the probability the test data belongs to each of the accents. The test data is classified as the accent who’s GMM produced the highest probability. His work focused on the task of distinguishing four types of Mandarin. They reported a classification error 88.3% for the male speakers.

The work by Zheng et al. indicates the advantage gained in identification accuracy when the accent is treated as a matter of degree. A similar perspective exists in the machine learning community in such forms as outlier detection [37] [45], categorization by intrinsic margin [44], and data cleansing [34]. Angelova [4] applied this to the facial recognition problem by removing, or pruning, the 'noisy' samples and features from the training set. Vezhnevets [70] used this pruning technique to improve AdaBoost accuracy.

Our work in accent identification was motivated by the observation that the field of machine learning provides applicable ensemble learning extensions for GMMs. We studied extending the GMM with both pruning and boosting and also how it effects the task of accent identification. We also compared the performance of the GMM with the phoneme-based HMM and the linear Support Vector Machine (SVM). We present results that show the advantage of using ensemble learning with GMMs over the GMM alone or the linear SVM. Also, we notice the GMM performs comparably to the HMM, given that the GMM does not take advantage of the a priori knowledge that the HMM does.

Our Contribution: Applying ensemble learning methods we have demonstrated the ability to increase speaker accent accuracy without using additional information. Further, it does not require a transcript of the speech, training phoneme-based
models, or extensive supervision. Further, we have shown that both the GMM and its extensions by ensemble learning are worthwhile approaches that yield a significant improvement over simple linear classifiers and closely approach the performance of other classifiers that use apriori knowledge, like the HMM.

2.5 Clustering Social Networks

Finding communities in social networks has been rapidly researched during the past decade [27, 35, 32, 50]. Early on researchers worked with the ‘small world’, power-law, and network transitivity properties. These properties characterized communities of isolated groups where a small number of objects in a community were connected to objects in other communities. A more recently proposed property is the connectedness of communities, where objects within the community have shorter path lengths than to objects outside the community [24]; building on such definitions, one should expect communities to be densely connected subgroups of the graph [48, 49].

A drawback of the aforementioned approaches is that they are graph partitioning methods – the communities cannot overlap. In reality, communities often overlap and sometimes in non-trivial ways. To address efficiency and memory footprint issues, work has progressed by extending algorithms to work within subspaces [65], identifying overlapping groups using local optimality [5, 6], and removing the constraints on the number of communities [3]. Methods for finding overlapping communities by searching for clique-like structures also exist [53, 73, 72].

Our Contribution: In this thesis, we propose an algorithm that can handle large datasets, particularly those with a high number of objects; we also allow overlap. The work in [51, 71] is similar to ours in that the first step in our algorithm is spectral, similar to multi-dimensional scaling (MDS); their algorithm is based on spectral properties of the Laplacian of a graph which they construct from a metric embedding; in some sense, they do the reverse of what we do. However, construction of the Laplacian is Ω(n^2). While we use GMMs as our tool for “soft” metric clustering, any method for metric clustering which readily gives overlapping clusters could be applied. GMMs happen to be an area of very active research (see
for example [18, 19, 63, 69]). We use a fast approximation to MDS [17] based on an
online sampling of the squared distance matrix. The Nyström method is similar to
MDS, and has been used for spectral segmentation (eg. [28]). Sparse approximations
to matrices in machine learning, typically to approximate the Gram matrix in kernel
methods have been considered ([66, 22]); in all cases, the methods sample the whole
matrix ($\Omega(n^2)$). When it comes to graph partitioning, there is much work, and we
suggest [41] for a good overview; a comprehensive review is given in [27], focusing
mostly on partitioning, but also considering overlapping clusters.
Sample covariance matrices provide important information about the probability distribution of the samples. This information can be used in pre-processing, such as PCA and whitening, where the samples can be reduced in dimensionality or decorrelated. Also, this information is used in tightly fitting probabilistic models to a dataset, such as the Gaussian Mixture Model, which we will be looking at in this chapter. In particular, we will be looking at high dimensional models.

The problem we address is the computational cost of training a Gaussian Mixture Model (GMM) incurred when using the full covariance matrix. The covariance is the defining characteristic of the GMM over other popular clustering algorithms, such as k-means clustering. Typically the EM algorithm is used for training a GMM on a given dataset. The expectation step uses the inverse of the covariance matrix to calculate the probability of each sample. The maximization step updates the covariance matrix using these probabilities. The running time of a single step of this EM algorithm is $O(TMd^2)$, where $d$ is the data dimension, $T$ is the number of samples, and $M$ is the number of mixtures in the model. The appearance of $d^2$ can be computationally prohibitive for high dimension problems, and thus one often uses an approximation to the full covariance. In addition, having the flexibility to fit $O(d^2)$ covariance matrix parameters to the data can lead to overfitting problems. The ideal approximation for the full covariance is one that is not only accurate and calculated quickly, but also has an inverse that can be efficiently used to calculate sample probabilities quickly.

Since we are focusing on the GMM, our fitting the data metric is the log likelihood of the GMM.

$$\log L = -\frac{1}{2} \sum_{t=1}^{T} (x_t - \mu)^T \hat{\Sigma}^{-1} (x_t - \mu) + \frac{T}{2} \log |\hat{\Sigma}^{-1}| - \frac{T}{2} \log 2\pi$$ (3.1)

where $T$ is the number of samples, $d$ is the sample dimension, and $\hat{\Sigma}$ is the estimate
of $\Sigma$. We will first give our discussion for a single mixture for simplicity. All our arguments extend to multiple mixtures. After a straightforward differentiation, it is easy to obtain that the log likelihood is maximized for

$$
\mu = \frac{1}{T} \sum_{t=1}^{T} x_t;
$$

$$
\Sigma = \frac{1}{T} \sum_{t=1}^{T} (x_t - \mu)(x_t - \mu)^T.
$$

This can be computed in $O(Td^2)$. The basic analytic task we address is how to efficiently choose $\hat{\Sigma}$, under some form of sparsity constraint, so as to do so efficiently.

A major drawback of the covariance matrix is its computational cost. The cost of calculating the covariance matrix grows quadratically with respect to the sample dimension. Thus, as the dimension of our data grows, the curse of dimensionality comes into effect. Although there are methods for feature selection or dimension reduction, it is not always desirable or sufficient. Leaving out samples only results in linear speed up since the computational cost grows linearly with respect to our sample size.

Due to the computational cost, associated with the full covariance matrix, an approximation of the covariance matrix is often used instead. The problem then becomes one of balancing computational cost with accuracy. A simple approximation is to use the diagonal of the covariance matrix or, in other words, use only the dimension by dimension variances; this ignores all correlation information. As we will explore further, there are several proposed approximations which mostly utilize decompositions of the covariance matrix. We introduce a new decomposition using a low rank perturbation of a diagonal matrix decomposition (LRPDD). We show how to efficiently use such a decomposition to compute $\Sigma^{-1}$ for use in the GMM model. We denote this the low rank GMM (LRGMM).

### 3.1 Covariance Matrix Approximations

We first discuss some well known covariance matrix decompositions before introducing LRPDD.
3.1.1 Diagonal Approximation

First we will cover the diagonal approximation. We rewrite our log likelihood equation with the diagonal matrix $D$, where $\Delta x_t = x_t - \mu$ and $\hat{\Sigma} = D \approx \Sigma = \frac{1}{T} \sum_t \Delta x_t \Delta x_t^T$

$$\log L = -\frac{1}{2} \sum_{t=1}^{T} (x_t - \mu)^T D^{-1} (x_t - \mu) + \frac{T}{2} \log |D^{-1}| - \frac{T d}{2} \log 2\pi$$  \hspace{1cm} (3.2)

which is maximized when we minimize

$$\varepsilon = \frac{1}{T} \sum_{t=1}^{T} \Delta x_t^T D^{-1} \Delta x_t - \log |D^{-1}|$$ \hspace{1cm} (3.3)

$$= \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\Delta x_t(i)^2}{D_{ii}} - \sum_{i=1}^{d} \log \frac{1}{D_{ii}}$$ \hspace{1cm} (3.4)

We can now solve analytically for $D$ by taking the derivative of this function and setting it to zero

$$\frac{\partial \varepsilon}{\partial D} = \frac{1}{T} \sum_{t=1}^{T} \Delta x_t^2(i) - D_{ii} = 0$$

$$D_{ii} = \frac{1}{T} \sum_{t=1}^{T} \Delta x_t^2(i)$$ \hspace{1cm} (3.5)

The derivation is given in Chapter 2. We can see that, with the diagonal constraint, the maximizer of $\log L$ is the diagonal of $\Sigma$. This can be computed in $O(Nd)$, but loses all off-diagonal correlations. This loss can be seen in the different models resulting from using a full covariance matrix versus using a diagonal approximation in Figure 3.1.

3.1.2 Optimal Rank-k Decomposition

Another popular decomposition is the eigenvalue decomposition

$$\Sigma = \sum_{i=1}^{k} \lambda_i u_i u_i^T,$$ \hspace{1cm} (3.6)
where $\lambda_i$ and $u_i$ are the $i^{th}$ eigenvalue and eigenvector of $\Sigma$. This is the optimal rank-$k$ decomposition under any unitarily invariant norm. However, this is not an invertible decomposition and so it is not appropriate for GMMs which use $\Sigma^{-1}$. 

Figure 3.1: Alphabet data fitted by GMMs with Full and Diagonal Matrix
### 3.2 Low-Rank Perturbed Diagonal Decomposition (LRPDD)

Our approximation to $\Sigma$ is using a low-rank perturbation of a diagonal matrix

$$\Sigma^{-1} \approx D^2 + aa^T \quad (3.7)$$

where $D$ is our diagonal matrix and $a$ is a $d$-dimensional vector that defines the low-rank perturbation. We use $D^2$ is used to ensure the approximation is positive semi-definite without having to add constraints.

The log-likelihood of a GMM, using this approximation is

$$\log L = -\frac{1}{2} \sum_{t=1}^{T} (x_t - \mu)^T (D^2 + aa^T)(x_t - \mu) + \frac{T}{2} \log |(D^2 + aa^T)| - \frac{Td}{2} \log(2\pi) \quad (3.8)$$

where $T$ is the number of samples and $d$ is the sample dimension. Our goal is to maximize (3.8). Thus, our optimization problem is to minimize

$$\varepsilon = \frac{1}{T} \sum_{t=1}^{T} (x_t - \mu)^T (D^2 + aa^T)(x_t - \mu) - \log |(D^2 + aa^T)| \quad (3.9)$$

where $D$ is a diagonal matrix with diagonal elements $D_{ii}$, and $a$ is a vector $[a_1, \ldots, a_d]^T$. We may also formulate a slightly different optimization problem

$$\varepsilon = ||\Sigma^{-1} - (D^2 + aa^T)||_F^2 \quad (3.10)$$

for a given $\Sigma$. These two problems define a new matrix decomposition which preserves rank.

In finding the optimal values for $D$ and $a$, it may seem that using the diagonal of $\Sigma$ for $D$ is sufficient. However, this requires fitting $aa^T$ to $\Sigma$ with zeros along the diagonal. This pulls $a$ away from its optimal value and towards a saddle-point at $a = 0^d$.

In finding the optimal values for our parameters, $D$ and $a$, we search for the point where the gradient of our optimization equation reaches zero. First, we find
the gradient with respect to the $\alpha$ element of $D$, denoted $D_\alpha$:

\[
\frac{\partial \varepsilon}{\partial D_\alpha} = 2 \sum_{t=1}^{T} x_t^T D_\alpha x_t - \text{trace} \left( (D^2 + aa^T)^{-1} \frac{\partial (D^2 + aa^T)}{\partial D_\alpha} \right) = \sum_{t=1}^{T} x_t^T D_\alpha x_t - \text{trace} \left( \left( (D^{-2} - \frac{D^{-2}aa^T D^{-2}}{1 + a^T D^{-2}a} \right) (2D^\alpha) \right) = \sum_{t=1}^{T} x_t^T D_\alpha x_t - 2D^{-1}_\alpha + \frac{\text{trace}(2D^\alpha D^{-2}aa^T D^{-2})}{1 + a^T D^{-2}a} \tag{3.11}
\]

where the derivative of the log $| (D^2 + aa^T) |$ term is derived using Jacobi’s formula for the differential of a matrix determinant, $d \det(A) = \text{trace}(\text{adj}(A)dA)$; $D^\alpha$ is the vector $D$ with all elements set to zero except $D_\alpha$; and $\text{trace}()$ is the trace function.

This formula can be written in vector form as:

\[
\frac{\partial \varepsilon}{\partial D} = 2\text{diag}(\Sigma) D - 2D^{-1} + \frac{2(D^{-2}a)^2}{1 + a^T D^{-2}a} \tag{3.12}
\]

The gradient for $a$ can be derived in a similar manner:

\[
\frac{\partial \varepsilon}{\partial a_\alpha} = 2 \sum_{t=1}^{T} \left( \sum_{i=1}^{d} x_t a_i \right)^2 x_t a - \text{trace} \left( (D^2 + aa^T)^{-1} \frac{\partial (D^2 + aa^T)}{\partial a_\alpha} \right) = \frac{2}{T} \sum_{t=1}^{T} \left( \sum_{i=1}^{d} x_t a_i \right)^2 x_{na} - \frac{2a_\alpha}{D_\alpha} + \frac{2a_\alpha}{D_\alpha} \sum_{i=1}^{d} \frac{a_i^2}{D_i} \tag{3.13}
\]

which can be put into vector form as

\[
\frac{\partial \varepsilon}{\partial a} = 2\Sigma a - 2D^{-2}a + \frac{2(D^{-2}a)(D^{-2}a)^T a}{1 + a^T D^{-2}a} \tag{3.14}
\]

The solution of these equations is non-trivial. We first looked at setting to zero the gradient with respect to $a$. Through some simplification we obtain from (3.15) the following three equivalent conditions for $a$:

\[
\Sigma a = \frac{2D^{-2}a}{1 + a^T D^{-2}a} \tag{3.16}
\]

\[
2(1 + a^T D^{-2}a)D^2 \Sigma a = a \tag{3.17}
\]

\[
a = \frac{(D^2 \Sigma)^{-1} a}{1 + a^T D^{-2}a}. \tag{3.18}
\]
Each of these equations give us some information. Eqn. (3.16) shows that this is a generalized eigenvalue problem. Despite the symmetric positive semi-definite nature of the covariance matrix, there does not seem to be a linear-time solution to this problem. Eqn. (3.17) is a potential candidate for updating $a$ in an iterative manner (where the left-hand side values are from initial values or the previous iteration). Unfortunately, multiplying by $\Sigma$ makes this an expanding equation and actually diverges from the optimal value of $a$. Eqn. (3.18) gives us a contracting version of Eqn. (3.17), but requires multiplying by the inverse of $\Sigma$. Multiplying by $\Sigma$ can be done in $O(Td)$ because $\Sigma$ is a sum of outer products, but multiplying by $\Sigma^{-1}$ cannot.

We also looked at setting to zero the gradient with respect to $D$. From (3.12), we can simplify down to the following equations:

$$\frac{D^{-2}}{1 + a^T D^{-2} a} = \text{diag}(\Sigma)$$  \hspace{1cm} (3.19)

$$D^{-2} = \text{diag}(\Sigma)(1 + a^T D^{-2} a)$$ \hspace{1cm} (3.20)

$$D^{-2} = \text{diag}(\Sigma) + \frac{(D^{-2} a)^2}{1 + a^T D^{-2} a}.$$ \hspace{1cm} (3.21)

Again, although these equations are coupled with the value of $a$, they give us some insight into $D$. First, (3.19) supports the intuition that the optimal value of $D$ is not just the diagonal of $\Sigma$, but involves a contribution from $a$. Second, as with the gradient for $a$, (3.20) and (3.21) give us two possible equations to compliment (3.17) and (3.18) in an iterative update approach. However, since the problems of multiplying by $\Sigma$ and $\Sigma^{-1}$ still remain for $a$, the efficiency of these equations for $D$ do not lead to an overall efficient approach to maximizing the parameters.

By using an approximation based on a low-rank perturbation of a diagonal matrix, our intention is to obtain an improved accuracy over the diagonal approximation but maintain a linear bound with respect to the sample dimension. The low-rank perturbation adds another $d$ parameters over the diagonal approximations $d$ parameters. Therefore we expect the computational cost to be higher than the diagonal approximation, but still well below that of the full-rank approximation which has $d(d + 1)/2$ parameters. Further, by directly approximating the inverse
of the covariance matrix, the cost of inversion is avoided. Note that this low rank perturbation implies a similar low rank perturbation expression for $\Sigma$ itself. The difference between the LRPDD and the diagonal matrix approximation can be seen in Figure 3.2, comparing to Figure 3.1.

### 3.2.1 Iterative Solution Via Conjugate Gradient Optimization

Instead of an analytical solution, we use the conjugate gradient descent to reach an optimum[36] (See Algorithm 1 for a summary of the conjugate gradient algorithm). We initialize $a$ to be a random vector, whose elements are random samples from the uniform distribution in $[0, 1]$; $D$ is initialized as the inverse of the diagonal of $\Sigma$, which can be easily calculated in linear time. The gradient referred to as $g$ in the algorithm is just the concatenation of the gradient for $a$ and the gradient for $D$. The search direction is referred to as $v$. The threshold, $\tau_g$, is set to 0.001, our stopping criterion.

Although one generally expects to use $d$ steps of the conjugate gradient method, practice shows that only a few steps are needed.

With these formulas and algorithms, we can perform the EM algorithm for the LRGMM efficiently in $O(Td)$ time. For the expectation step, we calculate the log
Algorithm 1 Conjugate Gradient

1: **Input**: data $x$, diagonal matrix $D_0$, perturbation $a_0$, threshold $\tau_g$
2: $g_0 \leftarrow [\frac{\partial \varepsilon}{\partial a_0}; \frac{\partial \varepsilon}{\partial D_0}]$; $v_0 \leftarrow -g_0$.
3: while $|g_i| > \tau_g$ do
4: Perform a line search along $v_i$ to find optimal step size, $s$.
5: $a_{i+1} \leftarrow a_i + s \cdot v_i(1 \ldots d)$.
6: $D_{i+1} \leftarrow D_i + s \cdot v_i(d + 1 \ldots 2d)$.
7: $g_{i+1} \leftarrow [\frac{\partial \varepsilon}{\partial a_{i+1}}; \frac{\partial \varepsilon}{\partial D_{i+1}}]$.
8: $\beta \leftarrow \frac{g_i + 1}{g_i^T g_i} g_i^T g_{i+1}$.
9: Calculate new direction, $v_i \leftarrow -g_{i+1} + \beta v_i$.
10: end while

likelihoods of each data point for each component, using (3.8). First, we calculate the log-determinant of the covariance matrix

$$
\log |(D^2 + aa^T)| = \log(a^T D^{-1} a |D| + |D|) \tag{3.22}
$$

$$
= \log([a^T D^{-1} a + 1] |D|) \tag{3.23}
$$

$$
= \log(a^T D^{-1} a + 1) + \log |D|. \tag{3.24}
$$

Calculating $|D|$ and $a^T D^{-1} a$ is straightforward and only requires one pass over the parameters. We add to this term the log of the prior term, $\log \pi$. Finally, we compute the remainder of (3.8)

$$
\frac{1}{T} \sum_{t=1}^{T} x_t(D^2 + aa^T)x_t
$$

$$
= \frac{1}{T} \sum_{t=1}^{T} x_t^T D^2 x_t + x_t^T aa^T x_t
$$

$$
= \frac{1}{T} \sum_{t=1}^{T} x_t^T D^2 x_t + (x_t^T a)^2
$$

$$
= \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{d} x_t^2(i) D_i^2 + \frac{1}{T} \sum_{t=1}^{T} (\sum_{i=1}^{d} x_t^2(i) a_i)^2, \tag{3.25}
$$

where $x_t$ and $x_t(i)$ is replaced by $x_t - \mu$ and $x_t(i) - \mu(i)$ respectively. We can see that computing this term of (3.8) only requires $O(Td)$ time and therefore computing the log-likelihood of the data for the given parameters only takes $O(MTd)$ time.
The equations for the maximization step, (3.12) and (3.15), have been derived. Since $D$ is a diagonal matrix, multiplying $D$ or $D^{-1}$ by a vector takes only $O(d)$ time. For (3.12), we need to multiply by the diagonal of $\Sigma$. This can be done in efficiently, since

$$
\Sigma_{ii} = \sum_{t=1}^{T} (x_t(i) - \mu_t(i))^2.
$$

(3.26)

A more difficult problem arises when we compute (3.15). We notice here that we need to solve $\Sigma a$ efficiently. Again, by using the definition of $\Sigma$, we can solve this efficiently;

$$
\Sigma a = \sum_{t=1}^{T} (x_t - \mu)(x_t - \mu^T)a
$$

(3.27)

$$
= \sum_{t=1}^{T} (x_t - \mu)(x_t^T a - \mu^T a)
$$

(3.28)

$$
= \sum_{t=1}^{T} (x_t - \mu)\beta_t
$$

(3.29)

$$
= \sum_{t=1}^{T} x_t^T \beta_t - \mu \sum_{t=1}^{T} \beta_t,
$$

(3.30)

where $\beta_t = x_t^T a - \mu^T a$. This can be solved in four steps, each only requiring $O(Td)$ time. First, we compute $\mu^T a$. Second, we compute $\beta_t$ for $t = 1, \ldots, T$. Each value of $\beta_t$ only needs to compute $x_t^T a$, so computing all of them can be done in $O(Td)$ time. Thirdly, given $\beta_t$, we compute $\sum_{t=1}^{T} x_t^T \beta_t$ and $C = \sum_{t=1}^{T} \beta_t$ simultaneously, which again can be done in $O(Td)$ time. Finally, we finish the equation by subtracting $\mu C$ from the result of previous step. Thus, having an efficient solution for $\Sigma a$, we have an efficient solution for computing the gradients for both $D$ and $a$. Further, this means that the conjugate gradient approach can be efficiently executed. As we stated earlier, in practice the number of conjugate gradient iterations needed for accurate approximations is much less than the data dimension or number of data samples. So, it can be expected that the LRGMM can be accurately trained in nearly linear time.

So far, we have discussed LRPDD for a single mixture. In order to extend to multiple mixtures, we need to introduce mixture weights, $\pi_k$. These values determine
the contribution of each mixture to the entire model. Mixture weights can also be viewed as prior probabilities that determine the chance a data point was generated by a particular mixture. Many of the details of these parameters are reviewed in the discussion on the Gaussian Mixture Model in Chapter 2. The main concern of training multiple mixtures is calculating the posterior probabilities of the data samples with respect to each model, \( p_{tk} \). These values are calculated using (3.8) with the added term \( \log \pi_k \), to account for the mixture weight. The posterior probabilities are then normalized for each data sample over all mixtures;

\[
\sum_{k=1}^{M} M p_{tk} = 1. \tag{3.31}
\]

For the maximization step, the mixture weights are approximated by the posterior probabilities;

\[
\pi_k = \frac{1}{T} \sum_{t=1}^{T} p_{tk}. \tag{3.32}
\]

Also, in estimating the values for \( D \) and \( a \) for mixture \( k \), data samples \( x_t \) are weighted by the respective posterior probability, \( p_{tk} \). The mixture means are now updated by

\[
\mu_k = \frac{1}{T} \sum_{t=1}^{T} p_{tk} x_t. \tag{3.33}
\]

When calculating the gradients for \( D \) and \( a \), an element on the diagonal of \( \Sigma \) and \( \beta_t \) now become

\[
\Sigma_{ii} = \sum_{t=1}^{T} p_{tk} (x_t(i) - \mu_k(i))^2 \tag{3.34}
\]

\[
\beta_t = p_{tk} (x_t^T a_k - \mu_k^T a_k). \tag{3.35}
\]

In the next section, we explore how our approximation compares to the diagonal approximation and the full rank covariance matrix.

3.2.2 Experimental Results on Synthetic Data

A key property of the LRPDD is that it captures the correlation information of the covariance matrix. To help visualize the capabilities of the LRPDD, we plotted
the differences of the LRPDD and diagonal approximations from the full covariance matrix in Figure 3.3.

To qualitatively illustrate the performance of our algorithm, we synthesized GMM data. Training datasets consisted of samples generated by a GMM with six clusters, randomly generated means, and randomly generated positive semi-definite covariance matrices for each mixture. An experiment consists of training a GMM on a dataset using one of the approximation methods described above (full covariance matrix, diagonal, or low rank perturbed). The results from our experiments with these datasets are shown in Tables 3.1 - 3.4.

3.2.2.1 Training Performance

We show performance from two perspectives, the in-sample and out-sample average log probabilities. Experiments were performed over various sample dimensions, dataset sizes, and number of model mixtures. Tables 3.1 thru 3.4 show results for datasets of different sample size, $T$, and number of mixtures in the model, $M$. 100 datasets were generated for each combination of sample dimension and dataset size.
Table 3.1: In-sample average log probabilities for $T=10k, M=6$ (div. by 1000)

<table>
<thead>
<tr>
<th>Dims.</th>
<th>Diag.</th>
<th>Pert.</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-109</td>
<td>-101</td>
<td>-87</td>
</tr>
<tr>
<td>50</td>
<td>-1032</td>
<td>-1005</td>
<td>-734</td>
</tr>
<tr>
<td>100</td>
<td>-2250</td>
<td>-2219</td>
<td>-1615</td>
</tr>
</tbody>
</table>

Table 3.2: In-sample average log probabilities for $T=1k, M=6$ (div. by 100)

<table>
<thead>
<tr>
<th>Dims.</th>
<th>Diag.</th>
<th>Pert.</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-109</td>
<td>-101</td>
<td>-88</td>
</tr>
<tr>
<td>50</td>
<td>-948</td>
<td>-925</td>
<td>-868</td>
</tr>
<tr>
<td>100</td>
<td>-2243</td>
<td>-2189</td>
<td>-2050</td>
</tr>
</tbody>
</table>

The average log probabilities are the result of averaging over the log probabilities from these 100 datasets.

An average log probability, by itself, yields little information, so the values have been divided by 1000 to make comparison easier.

From these tables, we can see a clear trend that the perturbed matrices provide a better approximation over the diagonal matrices. Since we see this in both our in-sample and out-sample accuracies, we can be assured that the performance of the perturbed matrix is not due to overfitting. Further, the out-sample accuracies bolded in Table 3.4, show that, at high dimensions, there is a greater potential for the full matrix to overfit than the perturbed matrix. The LRPDD provides the user a compromise between the loss of information from using a diagonal matrix with the potential for overfitting that comes from using the full covariance matrix.

Table 3.3: In-sample average log probabilities for $T=10k, M=3$ (div. by 1000)

<table>
<thead>
<tr>
<th>Dims.</th>
<th>Diag.</th>
<th>Pert.</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-111</td>
<td>-103</td>
<td>-100</td>
</tr>
<tr>
<td>50</td>
<td>-1035</td>
<td>-1009</td>
<td>-897</td>
</tr>
<tr>
<td>100</td>
<td>-2254</td>
<td>-2199</td>
<td>-1994</td>
</tr>
</tbody>
</table>
Table 3.4: Out-sample average log probabilities $T=1k,M=6$ (div. by 1000)

<table>
<thead>
<tr>
<th>Dims.</th>
<th>Diag.</th>
<th>Pert.</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-11</td>
<td>-10.2</td>
<td>-8.95</td>
</tr>
<tr>
<td>50</td>
<td>-95.6</td>
<td>-94.5</td>
<td>-95.6</td>
</tr>
<tr>
<td>100</td>
<td>-225.9</td>
<td>-232.2</td>
<td>-249.1</td>
</tr>
</tbody>
</table>

3.2.2.2 Runtimes

As we claimed in the previous sections, our motivation is to improve approximation without incurring large computational cost. We timed the training phase of each of the experiments. Figures 3.4 to 3.6 graph these runtimes with respect to sample dimension.

![Figure 3.4: Relation between runtimes and sample dimension for Low Rank Perturbed](image)

Since the scale of time is not the same, the runtimes for each approximation are plotted separately. Each graph shows how the training time increases with dimension for models with various numbers of mixtures (in this case, 3, 6, and 9 mixtures). This makes it easier to see how the computational cost increases with respect to sample dimension. In these graphs, we can see that both the diagonal and
perturbed approximations have a cost that increases linearly with respect to sample dimension, since they are bounded by $O(d)$. Also, we see the full covariance matrix cost increases quadratically. Table 3.5 shows the ratio of runtimes between using the full matrix to the perturbed matrix clearly growing with the data dimension.

### 3.2.3 Experimental Results on Real Data

We look at the performance of our low rank perturbed approximation on dealing with real data. In particular, we look at the case of training GMMs on speech.
3.2.3.1 Datasets

The speech comes from a TIMIT corpus of American English accents. Speakers who had grown up from various places in America were recorded while speaking 10 sentences. We look particularly at those speakers who were raised in the northern and southern regions.

The data is processed so that every 25ms of speech yields a 39-dimensional sample. We then train two GMMs, one for the northern samples and another for the southern samples. Test samples are then categorized based on which GMM gives a higher log probability. Whole words are similarly classified based on which GMM gives a higher average log probability.

3.2.3.2 Training Performance

Here are the results of the accuracy of classifying speech samples using various approximations for the covariance matrix.

As we saw in the synthetic data, the low rank perturbed approximation gives a better performance over the diagonal approximation. In particular, there is a
Table 3.6: Percentage accuracy of GMMs on speech samples

<table>
<thead>
<tr>
<th></th>
<th>Full Cov.</th>
<th>Diagonal</th>
<th>Perturbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Acc.</td>
<td>61.6%</td>
<td>49.5%</td>
<td>54.3%</td>
</tr>
<tr>
<td>Word Acc.</td>
<td>76.5%</td>
<td>45.9%</td>
<td>60.0%</td>
</tr>
</tbody>
</table>

significant increase with respect to the accuracy in classifying spoken words.

3.3 Discussion

We have proposed a new method of approximating the covariance matrix for a Gaussian Mixture Model. Instead of simply using the diagonal, we use a low-rank perturbation of a diagonal matrix and approximate the inverse of the covariance matrix, rather than the matrix itself. The conjugate gradient method is used to optimize the parameters with respect to the log probability formula of a GMM. By exploiting the structure of the covariance matrix and approximating its inverse, we developed an algorithm that is bounded linearly with respect to sample dimension. This makes its computational cost comparable to the that of the diagonal approximation, while at the same time we can model correlations.

In our experimental results we compared our approximation method to the diagonal approximation and the full covariance matrix. The in-sample accuracies show that while the perturbed matrix is not an exact approximation, it is consistently better than the diagonal as expected. Further, the out-sample accuracy shows that this improved flexibility does not result in overfitting, since the perturbed approximation also out-performs the diagonal approximation in this respect as well. A look into the runtimes of these experiments provides reinforcement that our method has a comparable computation cost to the diagonal approximation, with respect to sample dimension.

Although more complex in theory, the implementation of our method is not much more involved than implementing the diagonal matrix. Since the conjugate gradient method has very efficient implementations, this step is also efficient. It is of theoretical interest to develop analytic solutions to the LRPDD problem. This matrix decomposition may also be of independent interest.
In the future, it may be possible to lower the absolute computational cost of the perturbed approximation by using adaptive step sizes and thresholds for the conjugate gradient method. Also, one could look into using more than a rank-1 perturbation. Intuitively, it seems that perturbations from rank 1 to full rank can be similarly derived and provide a greater flexibility in finding the balance between computational cost and accuracy. The code for this model can be found bundled with our social networking code, which is available online [58].

The applicability of both this covariance matrix approximation and LRGMM cannot be understated. GMMs are ubiquitous and LRPDD combined GMMs (LRGMM) makes them computationally convenient and flexible enough to capture correlations. One possible application is in dealing with the recently increasingly common problem of clustering large datasets ([39],[74],[33]) and high dimensional data ([54],[21],[26]). Not only can information on the correlations between data dimensions be quickly calculated, but LRGMM can use this information to cluster the data. This makes the LRGMM an appropriate alternative to k-means clustering. Further, using even simple heuristics, the LRGMM can be a suitable replacement for classifiers.

As we will see in Chapter 5 becomes a powerful algorithm when coupled with tools to embed graphs into Euclidean space. Often a weighted graph can represent a myriad of types of relationships within networks. By using effective embedding, this information can be minimally distorted while yielding a Euclidean representation. With LRGMM, this information can be further used effectively to cluster the nodes of the original graph based upon the relationships it represented. The result is an efficient algorithm for clustering communities in graphs, for example social networks.

Another emerging application for the GMM is accent identification, which we discuss in Chapter 4. Although the Hidden Markov Model (HMM) is the model of choice for speech recognition problems, it requires apriori knowledge of the speech spoken in the form of transcripts. While this is manageable for training on an individual, this requirement becomes a large obstacle for processing speech from several speakers. The growing popularity of the GMM is that it does not require the transcripts that HMMs do. By utilizing the LRGMM, not only can larger corpus of data be processed, but the number of features can be increased without the severe
computational penalty that would be seen in the GMM.
CHAPTER 4
Learning American English Accents using GMMs

The human voice carries a great deal of information, not only in the words spoken, but also by way of pitch, harmonics and other auditory features. With these features, the field of speech recognition has been successful at speaker recognition and gender recognition. In this paper, we study the harder problem of accent recognition. The accent may be due to the fact that the speaker is a non-native speaker of the spoken language (e.g. a native German speaking English), or that the speaker is a native speaker that is using a particular dialect of the language (e.g. an English speaker with a 'Southern' United States accent). The accurate identification of a speaker’s accent has important uses. A non-native speaker’s accent can be used to identify their native language. The system can then switch to that native language or pre-process the non-native speech to improve the accuracy of speech recognition. A speaker’s accent can also be used as a corroborative biometric feature, to (for example) determine the speakers geography of origin.

The non-native speaker’s accent is mainly characterized by the difficulties in correctly pronouncing certain words or affixes (e.g. in English, cough vs. though) or in pronouncing sounds that do not occur in their native language. The accents of native speakers on the other hand, are more subtle, being differentiated by how they pronounce certain syllables or, in some cases, if they pronounce the syllable at all. In general identifying native speaker accents is considerably harder than identifying accents for non-native speakers; i.e. distinguishing between two speakers both speaking English but one of whom is native German and the other Native Afghani is considerably easier than if both are American, one Northern and the other Southern. We look at the latter harder problem of identifying speaker accents when both speakers are from the same native language, and our experimental test bed is a data set containing Northern and Southern accents in American English. The speakers are labeled as Northerners or Southerners depending on where they were raised as children. Naturally, our techniques and conclusions could be applied
to the easier case when both speakers are of different natives.

There are two typical approaches to speech identification. The typical Hidden Markov Model (HMM) approach and, a common machine learning approach, the Gaussian Mixture Model (GMM). The former is used to model words or phonemes, the atomic speech unit, and requires transcripts or some human preprocessing. Alternatively, the latter can work with the raw speech signal not requiring human input.

We build on the GMM approach by exploring the effects of applying ensemble learning. Ensemble learning improves upon classifiers at the expense of complexity and increased training time. Since, for accent identification, training is only performed once and separately from classification, it is acceptable to increase training time for higher accuracy. Also, the improvements we implement mostly add complexity to the training step and has a small impact on the time required for the identification step. In addition to bagging and Boosting, we present a simple ensemble method for pruning data, based off of the bagging method, which outperforms all the other methods.

**Problem definition:** Given a dataset of \( T \) speech samples of speakers from \( N \) geographical regions, \( \mathbf{x}_t, \) for \( t = 1, \ldots , T, \) build a classifier that distinguishes the \( N \) groups. Our classifier consists of \( N \) GMMs, one for each group of speakers and defined by a set of parameters, \( \lambda_k = \{ \pi_k, \mu_k, \Sigma_k \}, \) \( k = 1, \ldots , N. \)

### 4.1 Feature Extraction for Accent Identification

The spectral space of speech yields very effective features for various speaker identification problems. This is because the spectrum of speech reflects the vocal tract which is the dominant physiological factor in the uniqueness of a speaker’s voice. The most popular features are the cepstral coefficients of a mel-frequency filterbank. The mel-frequency filterbank emphasizes particular ranges of frequencies in a way that is similar to how the human ear perceives sound.

Our feature set comes from the mel-frequency cepstral coefficients (MFCC) of the speech samples. The data samples were pre-emphasized with a filter defined
by $H(z) = 1 - 0.97z^{-1}$. The speech is then processed in 25-ms frames with 15-ms overlap between frames. Each frame is multiplied by the Hamming window, which is defined

$$\text{Hamming}(z) = 0.54 - 0.46 \cos\left(\frac{2\pi z}{Z - 1}\right),$$

where, in our case, $Z$ is the number of signal samples in a 25-ms frame.

To obtain the MFCCs, we use cepstrum analysis which is based off of the Fourier Transform (see Section 2.1). The cepstrum is the Fourier Transform of the logarithm of the Fourier Transform of a signal. This is also known as the Fourier Transform of the decible spectrum of a signal. To calculate MFCCs, the spectrum of the speech frame is calculated and transformed into Mel frequency bands before the Fourier Transform is applied a second time.

The mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The reference point between mel scale and normal frequency measurement is defined by equating a 1000 Hz tone, 40 dB above the listener’s threshold, with a pitch of 1000 mels. Conversion of $f$ hertz into $m$ mel is given by

$$m = 1127.01048 \ln(1 + \frac{f}{700})$$

and the inverse by

$$f = 700(e^{m/1127.01048} - 1)$$

The process results in 12 cepstral coefficients. Cepstral mean subtraction was performed in each data sample to remove effects of channels. This involves finding the average value of each of the coefficients over the local speech data (in our case, the spoken word) and subtracting that average from all the cepstral coefficients.

Our feature set uses these centered coefficients and the signal energy, which is the signals amplitude. The first and second order differences of the coefficients and energy are also included giving us a total of 39 parameters. The processing from speech to MFCCs was done with the HTK 3.0 toolkit [1]. Figure 4.1 illustrates the complete process.
Figure 4.1: How the speech is pre-processed to produce MFCCs

4.2 The Gaussian Mixture Model (GMM)

We remind the reader of the details of the Gaussian Mixture Model. A Gaussian mixture density is a weighted sum of $M$ component densities. Each component density is a $d$ dimensional Gaussian density with mean vector $\mu_i$ and covariance matrix $\Sigma_i$. It is defined as:

$$P_i(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{d/2} \det(\Sigma)_i^{1/2}} e^{-0.5(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

where $x$ is a $d$-dimensional random vector, $P_i(x|\mu_i, \Sigma_i), i = 1, \ldots, M$, are the component densities. $\pi_i > 0, i =, \ldots, M$ are the mixing weights, which satisfy the normalization constraint that $\sum_{i=1}^{M} \pi_i = 1$. The full mixture model is characterized by a parameter set consisting of the mixing weights, mean vectors, and covariance matrices for the $M$ components. They are collectively notated

$$\lambda = \{\pi_i, \mu_i, \Sigma_i\}, \quad i = 1, \ldots, M,$$
The probability of a sample, $x$, being generated by the full mixture model is

$$p(x|\lambda) = \sum_{i=1}^{M} \pi_i P_i(x|\mu_i, \Sigma_i). \quad (4.4)$$

Each of the $N$ accents are modeled by GMMs that are independently trained using speech from the corresponding accent. In this case, we trained a GMM on Northern speakers and another GMM on Southern speakers. Training started with determining the initial mean values, $\mu$; by choosing frames of speech randomly (and without replacement) from the training data. The initial mixture weights were set to be uniform. From the initial means and mixture weights, the covariance matrices were calculated.

After initializing the GMM parameters, a number of iterations of the EM algorithm are applied. In the expectation step, the probabilities of each frame of speech data are calculated by:

$$p_{ij} = \pi_i P_i(x_j). \quad (4.5)$$

During the maximization step, the parameters are updated using the values from the previous expectation step, as follows:

$$\hat{\mu}_i = \frac{\sum_j p_{ij} x_j}{\sum_j p_{ij}}; \quad (4.6)$$

$$\hat{\Sigma}_i = \frac{\sum_j p_{ij} (x_j - \hat{\mu}_i) \otimes (x_j - \hat{\mu}_i)}{\sum_j p_{ij}}; \quad (4.7)$$

$$\hat{\pi}_i = \frac{1}{T} \sum_i p_{ij}; \quad (4.8)$$

($T$ is the number of data samples in the training set).

During identification, we looked at the sum of the logarithmic probabilities over all the speech data for a given speaker. This sum is defined

$$\sum_{t=1}^{T} \log p(x_t|\lambda_k), k = 1, \ldots, M, \quad (4.9)$$
where $M$ is the number of accents modeled. The speaker was then classified according to which accent model yielded the greatest average logarithmic probability for the speaker’s speech, $\hat{k}$, as shown in

$$\hat{k} = \arg \max_{k=1}^{M} \sum_{t=1}^{T} \log p(x_t|\lambda_k)$$

(4.10)

In our work with accent identification, we also considered the option of using PCA to reduce the dimensionality of the data before modeling with a GMM. We found that we could retain about 90% of the information while reducing the data down to 25 dimensions. However, we noticed that the data sample classification accuracy using the reduced dimension data, 45.9%, was lower than for using the full data, 52.7%.

4.2.1 Baselines

To better understand our results, we will establish a baseline with two other approaches: the Hidden Markov Model (HMM) and a linear Support Vector Machine (linear SVM).

**HMM.** The HMM uses transcripts to help model the data. The transcripts consist of more than just a written form of the words spoken. Transcripts also indicate the time a word starts and ends, so that corresponding speech can be removed from other words and noise. They may also contain the start and end times of the phonemes that make up each spoken word. The HMM is therefore an optimistic baseline which has the benefit of apriori knowledge provided by human segmentation. Conversely, the linear SVM represents the simplest of classifiers and is used as a lower bound.

The HMM is designed to model a sequence of observations. Each observation is assumed to be independently generated from a particular probability density, called an observation density. It is also assumed that there is a transition probability that determines whether the next observation in the sequence will come from the same observation density or one of the other densities. These are represented by a transition matrix, $A$, where $A_{ij}$ is the probability that density $j$ will be used, given that an observation from density $i$ has just been generated. The assumption that
transitions only depend on the immediately previous observation density is why it is a Markov model. Since the transitions are not explicitly observed, it is a Hidden Markov model.

The HMM was trained and tested using the HTK package, similar to what was used by Teixeira[67]. They are phoneme-based models, where each state is trained on a particular phoneme. It is a continuous HMM, in that the observation densities of each state are continuous. A combination of the Viterbi alignment and the Baum-Welsh algorithm are used by the HTK package to train the model and classify the test speech samples.

**SVM.** The Support Vector Machine works on sets of data that are labeled as belonging to one of two classes. Typically, the SVM is used to classify data into two classes though, it can been expanded to work with multiple classes. The goal of the SVM is find a mapping of the data into a space where the separation between the classes of data is as large as possible. New data points are then mapped into this space and similarly classified. Formally, the SVM maximizes the margin to the hyperplane. Originally, the SVM was proposed as a linear classifier. Since the early 1990’s, the kernel trick has been applied which replaces the dot product with a non-linear kernel function. Thus the data can be mapped into higher dimensions and spaces that are non-linear in the original input space. In our experiments we use the linear classifier kernel, also known as the linear SVM.

The linear SVM was used by way of the LIBLINEAR implementation [25]. The linear SVM classifier used the MFCCs from each 25ms window of speech as input to output a class. These baseline classifiers yield probabilities for each 25ms frame. Speakers are classified by these in a manner similar to that shown in (4.10).

### 4.2.2 Ensemble Learning

Since the linear SVM can typically handle noise well, its poor performance (see Table 4.1) indicates that there is some property of the data that makes the task difficult. When we looked into the misclassified samples, we found that a majority of them were Southern speakers that spoke very similar to the Northern speakers. These speakers presented two problems. First, they were misclassified as Northern
Table 4.1: Speaker Accuracies of Baselines and GMM

<table>
<thead>
<tr>
<th>Method</th>
<th>Alone</th>
<th>Bag</th>
<th>Prune</th>
<th>Boost</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMM</td>
<td>60%</td>
<td>65.9%</td>
<td>69.92%</td>
<td>63.9%</td>
</tr>
<tr>
<td>GMM-Diag</td>
<td>50.4%</td>
<td>56.27%</td>
<td>62.85%</td>
<td>51.52%</td>
</tr>
<tr>
<td>LRGMM</td>
<td>57.85%</td>
<td>61.45%</td>
<td>66.36%</td>
<td>61.36%</td>
</tr>
<tr>
<td>LINSVM</td>
<td>53.4%</td>
<td>47.36%</td>
<td>51.9%</td>
<td>53.4%</td>
</tr>
<tr>
<td>HMM</td>
<td>78.2%</td>
<td>75.9%</td>
<td>73.7%</td>
<td>75.9%</td>
</tr>
</tbody>
</table>

speakers. Second, they caused the model of Southern speakers to become more like
the model of Northern speakers. Thus, some Northern speakers became misclassified
as well. These speakers had 'weak' accents, producing confusing samples that mis-
lead the training of the GMMs. In this section we look at methods from ensemble
learning which could help in dealing with these confusing samples.

4.2.2.1 Bagging

Bagging, also known as bootstrap aggregating and introduced by Breiman [10],
is our first extension of the GMM. It is straightforward to implement, given the
underlying classifier. A number of subsets, \( S_j \), \( j = 1 \ldots W \), are chosen randomly
with replacement from the training set, \( S_{\text{train}} \). Each subset has the same number
of samples as the training set. In training, the samples from a bootstrap subset are
partitioned into sets determined by the labeled accents, \( S^m_j = x \in S_j | \text{label}(x) = m \),
for \( m = 1 \ldots M \). The subset \( S_j \) is used to train a discriminant, \( h^m_j(x) \). Thus, we
have a GMM for each subset-accent pair.

During classification, the data is passed through each discriminant, \( h^m_j \). For
each accent, the discriminants are summed over the bootstrap subsets. The data is
classified to the accent that yields the greatest sum:

\[
H_{\text{bagging}}(x) = \arg \max_{m=1}^{M} \sum_{j=1}^{W} h^m_j(x). \quad (4.11)
\]

By combining the results over several classifiers, the intent is to decrease the influ-
ence of 'noisy' data on the final classification.

4.2.2.2 AdaBoost

The next extension we looked at was AdaBoost. This is a well known meta-algorithm that also works on the assumption that hard but informative samples may be present in the training set. Instead of ignoring these, AdaBoost focuses on learning to properly classify the hard samples.

AdaBoost is an iterative algorithm where several classifiers are, in turn, trained and then applied to the training set. On each iteration the training samples are weighted based on how many times they were incorrectly classified by the previous classifiers. For the first iteration, the weighting is uniform. On each subsequent iteration the weights are updated to emphasize the misclassified samples [29].

Specifically, to find the hard samples, the classifier from the previous iteration is applied to the weighted training set. Each sample that is misclassified is determined as 'hard' and has its weight increased. Conversely, the samples that are properly classified have their weights decreased. Thus, when the next classifier is trained it will put emphasis on correctly classifying samples proportional to the number of times they are misclassified over previous iterations. Since we are classifying our data as Northern or Southern speakers, we define the classifier at iteration $j$ as

$$h_j(x_t) = \begin{cases} +1, & \hat{k} = \text{Northern model} \\ -1, & \hat{k} = \text{Southern model} \end{cases} \quad (4.12)$$

In similar fashion, the labels for each training sample are one of two values, $y_t \in \{-1, +1\}$.

Initially, the weights are equal for all samples, $w_0(t) = \frac{1}{T}, t = 1 \ldots T$. After training the first classifier, $h_0(t) = \text{arg max}_{k=1}^{M} P(x_t|\lambda_k)$, we calculate the error. The error for classifier at iteration $j$ is:

$$\epsilon_j = \sum_{t=1}^{T} w_j(t)[y_t \neq h_j(x_t)]. \quad (4.13)$$

Since the samples are weighted during training, the calculation of the error must
involve a weighted sum. The error is the sum of the weights over the misclassified samples, so that successive iterations focus on these samples.

The error of the classifier is used to calculate a weighting coefficient, $\alpha_j = 0.5 \ln \frac{1 - \epsilon_j}{\epsilon_j}$. This coefficient has two purposes. The first purpose is to determine by what magnitude and direction to change the sample weights. The second purpose is to be the weight of classifier $h_j$ when it is used for classification. Intuitively, if $h_j$ has a low error rate (i.e. successfully learned the weighted data), then the coefficient is large and AdaBoost will put more emphasis on the samples that have been misclassified by $h_j$ (especially those that have also been misclassified by previous classifiers). Also, $h_j$ will have a greater part in classifying samples in the final classifier. If the error rate is very high, then the coefficient becomes negative and indicates the weighted data presents difficulty in training the underlying model (in this case, the GMM). Thus, the weight of the samples is reduced and the contribution of this classifier is inverted (since it is most likely wrong) before being added to the final classifier. Finally, if a classifier performs close to random, the weights are not changed significantly and the contribution of the classifier to the final classifier is almost nil. The sample weights are updated as follows

$$w_{j+1}(t) = \frac{w_j(t)e^{-\alpha_j y_t h_j(x_t)}}{\sum_{t=1}^{T} w_j(t)e^{-\alpha_j y_t h_j(x_t)}}.$$  \hspace{1cm} (4.14)

After several iterations are completed, a 'strong' classifier, $H$, is formed by a linear combination of the iterative classifiers, $h_j$. The iterative classifiers are weighted proportionally to their accuracy on classifying the training set, $\alpha_j$. We define the relation between the strong classifier and the iterative classifiers as

$$H(x_t) = \text{sign} \left( \sum_{j=1}^{W} \alpha_j h_j(x_t) \right).$$  \hspace{1cm} (4.15)

where $W$ is the number of iterative classifiers that were trained.

As can be seen from Chapter 2, the contribution of a data sample in the maximization step is determined by the data sample’s posterior probabilities. Therefore, the boosting weights need only to be multiplied with the posterior probability from
(2.5), which gives us the new posterior probability function

\[ P(i|x_t) = w(t) \frac{p(x_t|i)p_i}{p(x_t)} \]  \hspace{1cm} (4.16)

The boosting weight will then be propagated through the posterior probability in the maximization step.

4.2.2.3 Pruning

While bagging gains robustness to noise by averaging over several classifiers, another philosophy is to remove the data samples that are “hard” to classify. The pruning approach uses several classifiers to identify “hard” data samples and removes them from training set. The resulting pruned data set is then used to train the final classifier. The goal is to remove data that may cause the final classifier to overfit and decrease accuracy.

Pruning starts off by training classifiers on random subsets of the training set, as in the bagging approach. These classifiers are called ‘weak’ classifiers, to differentiate them from the ‘strong’ classifier that is trained at the end. After the weak classifiers are trained, they are each applied to the entire training set. For each data sample, the sum of log probabilities over all weak classifiers is calculated for each accent. A data sample is classified according to which accent yielded the greatest sum. A data sample that is classified differently from how its labeled is considered a ‘hard’ or ‘noisy’ sample. These samples are removed from the training set to create a new pruned training set. The ‘strong’ classifier is defined as a classifier trained on this pruned training set.

\[ S_{pruned} \subset \bigcup_{t: H_{pruning}(x_t) = y_t} x_t \]  \hspace{1cm} (4.17)

where \( H_{pruning} \) is defined in the same manner as \( H_{bagging} \) in Eq. 4.11. The outline of the algorithm is given in Algorithm 2.
Algorithm 2 Pruning

**Input:** data $x_i$, num. of subsets $m$, subset size $n$, accents $a_j$

**for** $k = 1$ to $m$ **do**
- Generate subset $k$ by choosing $n$ data from $x$, uniformly and with replacement.
- Train ’weak’ classifier on subset $k$.
**end for**

**for** each data $x_i$ **do**
- **for** each accent $a_j$ **do**
  - **for** each subset $k$ **do**
    - Classify $x_i$ with model for accent/subset $(a_j,k)$
    - Add log prob. of $x_i$ for $(a_j,k)$ in a sum for accent $a_j$
  **end for**
- **end for**
- Find the accent with the greatest sum, $a'$
- **if** $x_i$’s label matches $a'$ **then**
  - Add $x_i$ to the prune training set
- **else**
  - Discard $x_i$
- **end if**
**end for**

Train ’strong’ classifier on pruned training set

4.3 Experimental Results

4.3.1 TIMIT Corpus

We conducted experiments classifying real data samples, which were 25ms frames of speech. A word is a collection of samples. To classify a word, we used the sum-log-prob of all samples in the word. Similarly, the speakers were classified based on which accent yields the greatest sum-log-prob of all samples.

The complete (i.e.unpruned) training set was recorded from 98 speakers and contained 8,295 words, with 4,490 coming from the 53 Northern speakers and 3,805 coming from the 45 Southern speakers. The testing set was recorded from 35 speakers and contained 2,988 words with 1,542 coming from the 18 Northern speakers and 1,446 coming from the 17 Southern speakers. Due to the small number of speakers in the given test set, we used cross-validation as described below (see also [31]).

The speech data was taken from the TIMIT’s Acoustic-Phonetic Continuous Speech Corpus[30]. The speakers were primarily Texas Instrument personnel and selected to be representative of different geographical dialect regions. The dialect
region was defined as the geographical area of the U.S. where the speaker lived during their childhood years (ages 2 to 10). The Northern region primarily consisted of New York, Michigan, Wisconsin, Minnesota, and the Dakotas. The Southern region primarily consisted of Louisiana, Mississippi, Alabama, southern Georgia, the eastern half of the Carolinas and Florida.

4.3.1.1 Speech Processing and GMM Training

Speech processing was done with the HTK toolkit [1]. Although the HTK provides tools for training GMMs it did not provide a way of applying weights to the samples as needed in Adaboost, so a separate implementation was used in order to perform boosting on the frames of speech (code can be found [57]). In training the GMM, we used an EM approach. On each iteration of training, the parameters were initialized and then re-estimated once. After looking at using 8, 16 and 32 components in the GMM (see Table 4.2), we decided to use 8 mixtures for the GMMs used in the ensemble experiments since this allowed the experiments to run more quickly and still revealed the differences in accuracies between the methods (the performance of all 3 are similar).

4.3.1.2 Experiments

In addition to training plain vanilla GMMs, linear SVM and HMMs, we also used them with ensemble learning. The linear SVM was used to directly train on and classify the MFCCs from the series of 25ms windows of speech. For the HMM, we followed the tutorial in the HTKBook [1] to create monophone HMMs. The results of these approaches, and an 8 mixture GMM, are show in Table 4.1.

Results

These experiments reveal two important facts about the GMM. First, as compared to the linear SVM, the GMM provides a significant advantage over a linear classifier, particularly when ensemble learning is used. Second, despite lacking the transcripts used by the HMM approach, the GMM with ensemble learning is comparable given the lack of apriori knowledge. The poor performance of the linear SVM displays the limits of the linear classifier. Although we did not explore using
Table 4.2: Accuracies of GMM and the Extensions

<table>
<thead>
<tr>
<th>Method</th>
<th>Per Sample</th>
<th>Per Word</th>
<th>Per Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMM-8</td>
<td>51.5%</td>
<td>54.6%</td>
<td>60%</td>
</tr>
<tr>
<td>GMM-16</td>
<td>51.1%</td>
<td>52.6%</td>
<td>62.9%</td>
</tr>
<tr>
<td>GMM-32</td>
<td>51.7%</td>
<td>53.5%</td>
<td>62.9%</td>
</tr>
<tr>
<td>Bagging</td>
<td>52.8%</td>
<td>55.1%</td>
<td>65.9%</td>
</tr>
<tr>
<td>Boosting</td>
<td>51.9%</td>
<td>54.4%</td>
<td>63.9%</td>
</tr>
<tr>
<td>Pruning</td>
<td>51.2%</td>
<td>53.2%</td>
<td>69.92%</td>
</tr>
</tbody>
</table>

a Gaussian kernel, it would be interesting to see how it compares to the GMM. Intuitively, it would seem still too simple to outperform the GMM.

Classifiers were trained with 4 iterations of the EM approach. For the bagging and the pruning methods, 8 weak classifiers were used. For boosting, 8 iterations of boosting were performed. The methods were tested by cross-validation, leaving out one speaker for testing and training on the remaining 132 speakers. The results of the experiments are shown in Table 4.2. A comparison of all methods is shown in Table 4.1.

We observe that the increase in the number of components in the GMM seems to have little or no effect. The accuracy per sample and per word remains constant and the accuracy per speaker increases slightly. Applying the Bagging and Boosting technique shows an overall increase in accuracy, with Bagging having particularly increased the per speaker accuracy.

Pruning shows an interesting trend. While the per sample and per word accuracy are similar to the GMM accuracies, the per speaker accuracy increases significantly over the Bagging and Boosting. The reason behind this is that the correctly classified words were more evenly distributed over the speakers with pruning rather than in bagging or boosting. In other words, it is more likely that a speaker will have more than 50% of their words correctly classified with pruning than with the other techniques. This results in getting a higher per speaker accuracy despite a lower per word accuracy.
4.4 Discussion of Results

Recent research has shown what seems to be an upper bound of about 65% when it comes to correctly identifying a speaker’s accent without using apriori knowledge of the accent’s characteristics. Our contribution has been to demonstrate that using GMMs with ensemble learning technology but without apriori information, it is possible to get comparable accuracy. Ensemble learning methods significantly increase speaker accent accuracy. Not needing the transcripts is useful because we don’t need extensive human supervision or the need to train phoneme-based models. Further, we have shown that both the GMM and its extensions by ensemble learning are worthwhile approaches that yield a significant improvement over simple linear classifiers and closely approach the performance of other classifiers that use apriori knowledge, like the HMM. It would be interesting to also consider non-linear SVM approaches (eg. using Gaussian, RBF kernels).

Our experiments involved discerning a Northern American English accent from a Southern accent. From here it would be interesting to explore multi-accent identification and the ability to discern similar sounding accents (eg. Southern accent vs. Carolina accent). Also, our results show that there may be a chance to improve on identifying a speaker’s native language. In this case, training could be done on speakers using a language that is their secondary language with the goal of identifying the primary language. Since the approaches to accent identification are highly related to non-native accent identification, it is very likely that advances in one area will work well in the other area. Another path to explore is implementing other ensemble learning techniques that replicate the ability of human experts to determine the strength of an accent, as in Zheng’s work [75]. This may include combining or hybridizing the ensemble techniques we explored in this paper.

The main drawback to the Gaussian Mixture Model is that the computational cost of calculating the full covariance matrices grows quadratically with the dimensions of data. As we’ve seen, these matrices are important and allow the GMM to use critical information that increases its accuracy over other approaches like the linear SVM. In particular, while the diagonal elements of the covariance matrix are important, the correlation information in the covariance matrix off-diagonal el-
elements are very valuable. Therefore, quickly approximating the covariance matrix with a diagonal matrix is insufficient. This is exactly where LRGMM comes in. LRGMM gives us a method to approximate the covariance matrix efficiently while maintaining correlation information. We compare LRGMM in Table 4.1. In particular, we see that it does not suffer significantly compared to the full covariance matrix, yet it is significantly better than the diagonal approximation.
CHAPTER 5
Discovering Overlapping Social Groups: SSDE-Cluster

Social, collaboration and interaction networks (Twitter, Facebook, DBLP, Citeseer, Amazon, etc.) are a source of vast, multi modal information. Many communities have taken advantage of such networks in order to communicate ideas broadly and quickly. The same technology has also allowed researchers to observe the dynamic nature of communities and how individuals work within them. One of the fundamental tasks is to identify the overlapping communities in a network of interactions – nodes abstractly represent the entities of interest (books, people, etc) and edges represent the interactions between nodes (e.g. an edge between two people if they purchased the same book). Clusters represent the collections of nodes which are “similar” and the same node may be in more than one cluster (a person may have interests in science fiction and philosophy).

As networks increase in size, clustering algorithms need to be scalable. Even quadratic time algorithms will soon be unacceptable. For our experiments, we use the DBLP paper-paper network, with about 1 million papers, and over 30 million edges. Since there is no accepted definition of what a cluster in a social network is, the typical approach has been to “define” a cluster by constructing an intuitive algorithm the result of which is a cluster. We take this approach, in that we present an intuitive algorithm with a focus on efficiency. The result of the algorithm is “our definition of a cluster” and the validation is whether, in real social networks, the algorithm can produce good results.

**Problem Statement**  Given a weighted undirected graph, we seek to cluster its $n$ nodes, based upon their distances from each other, into overlapping groups. Further, we would like to achieve this in near linear time with respect to the dataset size and the number of clusters. We propose a solution to this problem, which we call SSDE-Cluster.
Overview of SSDE-Cluster

The input is a (typically sparse) social network graph $G = (V, E)$ on $n$ nodes (eg. books as nodes and (positive) edge similarities between books being the probability that a user would purchase both books conditioned on purchasing one). Typically, the definition of the node-similarities is application-specific, and is a determining factor for the quality of the results. The output is a set of clusters $C_1, \ldots, C_M$. These are overlapping clusters, which differs from other techniques which seek to create a partition. Partitioning is when a graph is divided into disjoint subsets. Thus, each node has membership with only one subset. However, in social networks, some nodes have multiple membership. Partitioning cannot capture this information and is therefore limited in correctly identifying communities and subsets of a network. While some approaches allow nodes to have multiple memberships, they carry a high computational cost. For example, the clique finding approach [53] clusters into near-cliques. Since a node may be part of multiple cliques, it may also be part of multiple clusters. However, the cost of finding all cliques of a particular size is a very computationally intensive task.

Our approach has two phases. The graph distances define a finite metric on $n$ points. The first phase efficiently embeds this metric in $\mathbb{R}^d$, so that Euclidean distances reasonably approximate the graph metric. According to the Johnson-Lindenstrauss theorem, $d = O(\log n)$ dimensions essentially suffices for this (practically, a constant number of dimensions (say $d = 100$)). In our experiments, we find that 5-10 dimensions suffices. The challenge is to construct this embedding without visiting every entry in the distance matrix (that would make the algorithm quadratic); we do this by carefully sampling $d$ rows of the distance matrix, which can be done in $O(nd)$, because the graph is sparse.

The advantage of first creating an approximate embedding is that now techniques can be used for clustering in metric spaces, for example $k$-means. We choose to use a Gaussian Mixture Model (GMM) because it can readily be adapted to give overlapping clusters ($k$-means delivers a partitioning). We choose the number of clusters by comparing to a random data set, and we determine the extent to which the clusters overlap by selecting a threshold to maximize the quality of the clusters.
5.1 The SSDE-Cluster Algorithm

The input is a weighted graph, $G = (V, E)$. The algorithm can be broken down as follows.

1. Run SSDE (spectral embedding) to approximately embed the graph in $d \ll n$ dimensions.

2. Run a GMM algorithm to compute “soft clusters”; determine $K$, the number of clusters, by comparing the marginal value of adding a cluster on the real data with random data.

3. Use the GMM probabilities to construct overlapping clusters; determine the degree of cluster overlap by locally optimizing a cluster density.

5.1.1 Sampled Spectral Distance Embedding (SSDE)

SSDE is an approximation to classical multidimensional scaling which was introduced in the context of fast graph drawing [15]. The distance matrix $D$ is the symmetric $n \times n$ matrix containing all the pair-wise distances. Suppose we position vertex $v_i$ at $x_i \in \mathbb{R}^d$. We are seeking a positioning that approximates the graph theoretical distances with the Euclidean distances, i.e.,

$$||x_i - x_j|| \approx D_{ij}, \quad i, j = 1, 2, \ldots, n. \quad (5.1)$$

We begin the derivation of the MDS equation (see [15]) by taking the square of each side where we get,

$$x_i^2 + x_j^2 - 2x_i \cdot x_j \approx D_{ij}^2. \quad (5.2)$$

Let $L$ be an $n \times n$ symmetric matrix such that $L_{ij} = D_{ij}^2$, for $i, j = 1, 2, \ldots, n$. Let $X, Q$ and $1_n$ be defined as follows:

$$X^T = [x_1, \ldots, x_n], Q^T = [||x_1||^2, \ldots, ||x_n||^2], 1_n^T = [1, \ldots, 1]. \quad (5.3)$$

Now (5.2) can be written as

$$[Q1_n^T]_{ij} + [Q1_n^T]_{ji} - 2[X X^T]_{ij} \approx L_{ij}. \quad (5.4)$$
Since $D_{ij} = D_{ji}^T$, and $(Q1_n^T)^T = 1_nQ^t$, the entire set of equations in matrix form is

$$Q1_n^T + 1_nQ^T - 2XX^T \approx L.$$  \hfill (5.5)

From this equation, since $\text{rank}(X) \leq d$, it immediately follows that the rank of $L$ is at most $d + 2$. That is $L$ has low rank (when $d \ll n$). As it stands, (5.5) is hard to solve on account of the dependence on $Q$ and $X$. We expand it into a more convenient form by using a projection matrix

$$\gamma = I_n - \frac{1}{n}1_n1_n^T,$$  \hfill (5.6)

where $I_n$ is the $n \times n$ identity matrix. Multiplying both sides of (5.5) by $\gamma$ from the left and right, we get

$$\gamma Q1_n^T \gamma + \gamma 1_nQ^T \gamma - 2\gamma XX^T \gamma \approx \gamma L \gamma.$$  \hfill (5.7)

Since $\gamma$ is a projection operator, (5.5) becomes

$$(\gamma X)(\gamma X)^T \approx -\frac{1}{2}\gamma L \gamma,$$  \hfill (5.8)

where we have used the fact that $\gamma = \gamma^T$. We may interpret this equation more easily by setting

$$Y = \gamma X = (X - \frac{1}{n}1_n1_n^TX).$$  \hfill (5.9)

$Y$ is an $n \times d$ matrix containing the same vectors of $X$ in a different coordinate system; one in which $\text{mean}(Y) = 0$; in this centered coordinate system, (5.5) is greatly simplified. So, we obtain the MDS equation,

$$YY^T \approx -\frac{1}{2}\gamma L \gamma = M.$$  \hfill (5.10)

The spectral decomposition of $M$ then gives $Y = [\sqrt{\lambda_1}u_1, \ldots, \sqrt{\lambda_d}u_d]$, where $\lambda_1, \ldots, \lambda_d$ are the top $d$ eigenvalues of $M$ and $u_1, \ldots, u_d$ are the associated eigenvectors. One useful property of MDS is that when the distance matrix is can be embedded in $d$ dimensions with minimal loss, then MDS recovers such an approximate embedding.
The drawback of MDS is that it is slow, $\Omega(n^2d)$. The next two lemmas give the insight for speeding up MDS.

**Lemma 1 ([40])** Any $n$-point finite metric can be embedded (to within $\epsilon$) in $\mathbb{R}^d$ for $d = O(\ln n/\epsilon^2)$.

This lemma says that from the point of view of distances, we can approximately treat the graph as $n$ points in a (small) $d$-dimensional space. For practical purposes, $d \leq 100$ suffices.

**Lemma 2** For any $n$ points in $d$-dimensions, let $L_{ij} = \|x_i - x_j\|^2$. Then rank($L$) $\leq d + 2$.

We can prove this lemma by recalling that $XX^T$ is an $n \times d$ matrix and thus has at most rank $d$. Also both $Q1_n^T$ and $1_nQ^T$ are rank 1 matrices. Thus, looking again at (5.5), we see that $L$ has a rank of at most $d + 2$. Note that the rank does not depend on $n$. Combining these two lemmas: any $n$-point finite metric is approximately embedable in $d = O(\ln n/\epsilon^2)$ dimensions, which means that the “numerical” rank of $L$ is roughly $d + 2 = O(\ln n/\epsilon^2)$. Practically, this means that a small number (say 100) suitably chosen rows of $L$ captures all the information in $L$. SSDE works in three steps (see [15]).

1. Sample a subset $c \approx 100$ columns to represent $L$, we denote this by $C$. These columns correspond to the nodes $v_{i_1}, \ldots, v_{i_c}$. We use the standard greedy 2-approximation to the $c$-center problem to select the nodes. This serves to “spread” out the nodes and give them large “numerical” rank. The first node is selected arbitrarily to compute the first column (SSSP); each subsequent node selected is farthest from the current set of selected nodes.

2. Construct a low rank approximation to $L$, $\hat{L} = C\phi^+C^T$, where $\phi$ is the $c \times c$ matrix of the intersection of $C$ and $C^T$. Some care may be needed in computing $\phi^+$ in a stable way (see [15]).

3. Construct the embedding $Y$ from the spectral decomposition of $-\frac{1}{2}\gamma\hat{L}\gamma$. 
We briefly comment on the running time of the SSDE phase. The first step involves \(c\) SSPP computations which can be performed in \(O(c|E|\log |V|) = O(cn \log n)\). Since \(\gamma = I_n - \frac{1}{n}1_n1_n^T\), \(\gamma C\) can be computed in \(O(nc)\) time. Let \(Q = \gamma C\), and construct the singular value decompositions

\[
Q = U_Q \Sigma_Q V_Q^T, \quad \phi = U_\phi \Sigma_\phi V_\phi^T,
\]

where \(U_Q\) is \(n \times c\), and all the other matrices are \(c \times c\); this takes \(O(nc^2 + c^3)\) time. Then,

\[
-\frac{1}{2} \hat{\gamma} \hat{L} = -\frac{1}{2} U_Q \Sigma_Q V_Q^T U_\phi \Sigma_\phi \Sigma_Q U_Q^T.
\]

The matrix product \(\Sigma_Q V_Q^T U_\phi \Sigma_\phi V_\phi^T V_Q \Sigma_Q\) and its eigen-decomposition \(U \Sigma U^T\) can be computed in \(O(c^3)\) time; hence, the spectral decomposition \(-\frac{1}{2} \hat{\gamma} \hat{L}\) can be computed in \(O(nc^2 + c^3)\). The embedding is \(Y = U_Q U(-\frac{1}{2} \Sigma)^{1/2}\). In practice, we may only need (say) the top 5 dimensions, and so a power iteration could be used in lieu of SVD (as was done in [15]).

### 5.1.2 Soft Clustering with a Gaussian Mixture Model (GMM)

After embedding the graph, our next task is to cluster the nodes. Many metric clustering methods exist, and we choose a standard GMM trained using the E-M algorithm (see Chapter 3 where GMMs and the E-M algorithm are given). One advantage of the GMM over (say) straight \(K\)-means is that the cluster probabilities will be useful in constructing overlapping clusters. Recall that a GMM is a weighted sum of \(M\) component densities \(p_1(x), \ldots, p_M(x)\); each component density is a \(d\) variate Gaussian function with mean \(\mu_k\) and covariance matrix \(\Sigma_k\):

\[
p_k(x) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)}.
\]

The GMM is \(p(x) = \sum_{k=1}^M \pi_k p_k(x)\), where the mixture weights satisfy \(\sum_{k=1}^M \pi_k = 1\). The model parameters are \(\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1, \ldots, M}\). We use a simple E-M procedure for training. We initialize the GMM by selecting \(\mu_k\) randomly without replacement, and setting \(\pi_k = 1/M\) and \(\Sigma_k = I_d\). One might also consider a better initialization of the \(\mu_k\) (e.g. using the greedy 2-approximation to the \(M\)-center problem, or more...
sophisticated methods as mentioned in the related work in Chapter 2). In the expectation step, the probabilities for node $i$ to belong to cluster $k$ are calculated:

$$p_{ki} = \pi_k p_k(x_i).$$

We then update the parameters using these probabilities during the maximization step:

$$\mu_k \leftarrow \frac{\sum_i p_{ki} x_i}{\sum_i p_{ki}}; \quad \Sigma_k \leftarrow \frac{\sum_i p_{ki}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i p_{ki}}; \quad \pi_k \leftarrow \frac{1}{n} \sum_i p_{ki}.$$

One iteration of E-M is $O(nMd^2)$; for small $M$ and $d$ this is acceptable. LRGMM (Chapter 3) gives us a method for reducing this to $O(nMd)$ using low rank perturbations to a diagonal covariance matrix. The output of the GMM phase is the set of probabilities $\{p_{ki}\}$, which will be used in computing the discrete overlapping clustering. For some applications, the probabilities themselves would suffice.

### 5.1.3 Determining the Number of Clusters

Determining the optimal number of clusters to use is an important but difficult problem. Having too few clusters will lump communities together, while having too many will break them down. This task of choosing the optimal number of clusters is called *model selection*, since we are choosing a model from a class of parametric models with varying numbers of parameters (i.e. the number of clusters is a parameter). Some commonly used criterions for model selection are the Bayesian Information Criterion (BIC) [64], Akaike Information Criterion (AIC) [2], and the Likelihood Ratio Test (LRT) [12].

We describe a simple heuristic for determining the number of clusters by comparing the real data clusters to random data. Too many clusters results in over-fitting the data and breaking up the true communities into several smaller communities.

We generated a random data set uniformly over the same space as the embedded data. Since there are no clusters in the random data, the increase in log-likelihood by adding an additional cluster is purely from over-fitting. When the
benefit from adding a cluster to the real data is not significantly more than that of adding a cluster to random data, we argue that it is time to stop adding clusters. Figure 5.1 shows the successive gains of adding a cluster to the embedded DBLP data versus to the random data; from Figure 5.1, we discern that about 7 to 10 clusters is appropriate (we used 7 in our experiments).

5.1.4 Determining the Extent of the Overlap

Every node $i$ is assigned to its most likely cluster, $\text{argmax}_k p_{ki}$. This constructs a partition of the nodes into clusters. We now extend this so that clusters may overlap. For node $i$, define $\alpha_{ki} = p_{ki}/\max_k p_{ki}$; where $\alpha_{ki}$ measures how diffuse node $i$’s membership. If $\alpha_{ki}$ is large (i.e. near one), then node $i$ is likely to belong to that cluster. By applying a threshold to the values of $\alpha_{ki}$, we allow nodes to have multiple memberships. Assume we have a cluster metric which measures the quality
**Figure 5.2:** $\alpha$ vs. overlap.

**Figure 5.3:** $\alpha$ vs. gain in cluster metric.
of a cluster. For concreteness, we use

\[ E(C) = \lambda \frac{W_{\text{in}}(C)}{W_{\text{in}}(C) + W_{\text{out}}(C)} + (1 - \lambda) \frac{W_{\text{in}}(C)}{|C|(|C| - 1)}; \]

\( W_{\text{in}} \) (resp. \( W_{\text{out}} \)) is the sum of similarities within the cluster (resp. from within to outside); \( E(C) \) combines similarity internally and to the outside with the average internal similarity. We used \( \lambda = \frac{1}{2} \).

We used \( \alpha_{ki} \) to define an ordering over node-cluster pairs, starting with high \( \alpha \). Initially, we had set \( \alpha^* = 0.3 \) which gives about 1.2 clusters per paper – this average number of clusters per paper could also be a user input to compute \( \alpha^* \). The comparison of \( \alpha \) to the average number of clusters per paper is shown in Figure 5.2.

However, we decided to try another approach to finding \( \alpha^* \). We processed the nodes according to the ordering, each time adding the node to its corresponding cluster if the metric for that cluster increases. This is similar in spirit to the local optimizations performed in [5, 6]. For comparison, we estimated the expected gain in the cluster metric by adding a node. We averaged, for the nodes outside the cluster, the degree of the node and the number of those edges that connect to a node in the cluster. We also averaged the weight of these edges. We then compared the increase observed in adding a node to the expected increase of adding a node. This is shown in Figure 5.3. We can see that \( \alpha^* = 0.7 \) is a good global threshold for our dataset.

### 5.2 The DBLP Network

In general, the social network is constructed by defining the “agents” (nodes) and the interactions or relationships between them (communications, similarities, etc.). Remember that the input to the algorithm is a graph. Exactly how that graph is constructed can have a large impact on the success, but that is not the emphasis of this research. We assume the graph is given. We apply our algorithm to the Digital Bibliography and Library Project (DBLP) data [43]; to define the graph we choose papers as nodes, and two papers are related if they have common authors. The underlying assumption is that papers having similar authors are more
likely to have similar content, which is reasonable, since most authors tend to have a focus area. We use the Jaccard index of the author sets to define similarity; for two papers $i, j$, let $A_i, A_j$ be their respective author sets. Then

$$s_{ij} = \frac{|A_i \cap A_j|}{|A_i \cup A_j|}; \quad d_{ij} = \frac{1}{s_{ij}}.$$  

($s_{ij}$ is the similarity, and $d_{ij}$ is a measure of “distance”). Most clustering algorithms for social networks work with the similarities $\{s_{ij}\}$ and try to optimize some measure of intra-cluster similarity versus inter-cluster similarity. Since our algorithms are metric based, we need a measure of difference, so we simply use the inverse-similarity; this allows us to construct our finite metric.

### 5.2.1 Validating Clusters

The recurring problem with applying a clustering algorithm to real data (where the “definition” of the cluster is the “result of the algorithm”) is to validate these clusters as good. We use human judgment based on the title and venue information of the papers. We emphasize that the titles are not available to the clustering algorithm. We just use the title words for human validation. Similarly we also use venues for validation. If there is coherence in the title words or venues, this indicates good clusters. We preprocess title texts by removing stop words and stemming [68]. For a cluster of papers $C_k$ (remember, clusters can overlap), we construct a word probability distribution $h_k(w)$. Similarly, we can construct a background distribution $h(w)$ for the entire data set of titles. The words $w$ for which $h_k(w) \gg h(w)$ are descriptive of the cluster; we can also identify the words $w$ for which $h_k(w) \ll h(w)$, which are indicative of not being in the cluster. This breaks down the papers into “topics” based on title texts. Visually, we depict the descriptive words using word clouds.

We also compare with an LDA analysis of the title data. Note that in general networks, such an LDA analysis is not possible. However for comparison purposes only, LDA on the title words is also considered. We perform a similar analysis for the paper venues, which should hopefully identify the conferences corresponding to the various topics.
5.3 Experimental Results

For our study, we constructed the DBLP network as described above and clustered the giant component, which consisted of about 900K papers (nodes) and over 30 million (weighted) edges. We chose $c = 25$ for the SSDE phase and then took the top 5 eigenvectors for the embedding. We clustered using the GMM into 7 clusters (as discussed earlier) and used a threshold $\alpha^* = 0.3$ to construct the overlapping clusters (a paper is in approximately 1.2 clusters on average). The
process of embedding and clustering took under 20 min on a modest single CPU machine.

**Cluster and Overlap Quality.** A typical pair of clusters is illustrated in Figure 5.5. With respect to the paper-paper similarity, we can measure the average similarity inside a set \((A\text{ and }B)\) as compared with the average between a set and the intersection\((A,C\text{ and }B,C)\) and the average between sets \((A,B)\). We expect the within set similarity to be larger than that between a set and an intersection in which the set participates, which in turn should be larger than the between set similarity. This is indeed found to be true for our overlapping clusters (see Table 5.1), which lends credence to their validity.

Since the clustering was done with paper-paper similarities based on authorship overlap, we may validate the clusters by looking at the text-based and conference based topics they represent (as discussed earlier). The 7 word clouds representing the clusters are shown in Figure 5.4 and the 3 largest intersections are shown in Figure 5.8. Our algorithm roughly breaks down the papers into 7 intuitive topics, with corresponding representative conferences shown in Figure 5.7.

---

**Table 5.1: Cluster Quality**

<table>
<thead>
<tr>
<th></th>
<th>SysParSim</th>
<th>ImageTh</th>
<th>SysArch</th>
<th>Learn</th>
<th>CombTh</th>
<th>WebDB</th>
<th>RoboBio</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysPS</td>
<td>\textbf{2.5}</td>
<td>2.4,1,7,1,3</td>
<td>2.4,1,7,1,3</td>
<td>2.4,1,6,1,3</td>
<td>2.5,1,4,1,0</td>
<td>2.4,1,7,1,2</td>
<td></td>
</tr>
<tr>
<td>ITh</td>
<td>\textbf{2.3}</td>
<td>2.4,1,8,1,5</td>
<td>2.4,1,7,1,3</td>
<td>2.4,1,7,1,5</td>
<td>2.4,1,6,1,3</td>
<td>2.3,1,7,1,5</td>
<td></td>
</tr>
<tr>
<td>SysA</td>
<td>\textbf{2.4}</td>
<td>2.4,1,7,1,5</td>
<td>2.4,1,8,1,5</td>
<td>2.4,1,8,1,4</td>
<td>2.4,1,8,1,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lrn</td>
<td>\textbf{2.4}</td>
<td>2.4,1,6,1,4</td>
<td>2.4,1,5,1,2</td>
<td>2.4,1,7,1,4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTh</td>
<td>\textbf{2.4}</td>
<td>2.4,1,7,1,2</td>
<td>2.4,1,8,1,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WbDB</td>
<td>\textbf{2.5}</td>
<td>2.4,1,9,1,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBio</td>
<td>\textbf{2.4}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Qualitatively, we see that the results are similar. One main difference between LDA and SSDE-cluster is that SSDE-cluster seems to have merged Web and DB into one cluster and produced a new topic which combines robotics and bioinformatics.

We illustrate the word clouds for the intersections for the three largest intersections in Figure 5.8. The robotics and bioinformatics cluster did not really have any significant intersections.

5.4 Discussion

We’ve presented an algorithm to quickly find overlapping communities in very large social networks. Our algorithm involves two fast and linear-time phases (hiding a log factor in the SSSP task for weighted graphs). Further exploration of what distance matrices are captured well by the greedy node selection algorithm in SSDE, and further study on how to convert soft cluster probabilities into a discrete overlapping clustering would be warranted. Also, our current algorithm puts every node into at least one cluster, and methods for outlier detection (i.e. allowing isolated nodes) could considerably help. In general, for efficient clustering on huge graphs, one cannot explore all the similarities between nodes $\Omega(n^2)$, and so such sampling based approximation approaches are likely to be the only feasible means of tackling such problems. The code used for these experiments is available online [58].
Figure 5.6: Conference Clouds for the 7 clusters
Figure 5.7: LDA Keyword Clouds for 7 clusters
Figure 5.8: Word/Venue Clouds for the three major intersections (word clouds on the left and venues on the right)
CHAPTER 6

Conclusion

Overall, this work presented a new approach, the Low-Rank Gaussian Mixture Model (LRGMM), for modeling data which can be extended to identifying partitions or overlapping clusters. We’ve demonstrated the effectiveness of the LRGMM on accent identification and finding communities in social networks. Not only have we shown that the LRGMM is computational more efficient, but also that it outperforms alternative methods. In addition, when dealing with problems such as accent identification, the LRGMM requires no apriori knowledge of the data.

6.1 Our Contribution

The LRGMM is the core of our contribution. It exploits the structure of the sample-based covariance matrix so that the training time is linearly bounded to the dimensionality of the data. Thus it is computational comparable to the diagonal approximation of the covariance matrix while yielding a more accurate model. We also demonstrated that the LRGMM does not succumb to over fitting the data, since the number of parameters remains limited with respect to the full covariance matrix. Despite the complex theory behind this approximation, the implementation is straight-forward and relies on the conjugate gradient method, which itself has very efficient implementations.

In addition to the LRGMM, we have presented two promising applications, one of which is to accent identification. The problem of accent identification is both difficult and hindered by human supervision. For effective identification, a model like the HMM requires a great deal of a priori knowledge such as transcripts. Further, due to the complex nature of human speech, hearing, and comprehension the dimensionality of speech corpus tend to be high (on the order of 50) in order to capture as much information as possible. The LRGMM is well suited to the task, as we have shown, in that it does not require the a priori knowledge the HMM requires and it scales linearly with the dimensionality of the data. Therefore we have a
method that can work efficiently on a larger dataset that would be time-consuming to gather if transcripts were required.

The other application demonstrated for the LRGMM is the task of finding communities within social networks. For this, we combined the LRGMM with Sampled Spectral Distance Embedding (SSDE) to take in a weighted graph representing the relations between people and return identified communities of these people. The SSDE efficiently embeds the weighted graph into Euclidean space while minimizing the loss of information. From there, the LRGMM clusters the embedded data to find the communities. A simple heuristic allows the clusters to be overlapping and improve the accuracy of their membership. A major contribution to this task is the efficiency of both the SSDE and LRGMM. Thus the method can work on larger and denser networks than many alternative algorithms.

6.2 Open Problems

While we have a working version of the LRGMM, there are a couple ways to improve upon it. Currently we use the conjugate gradient to improve the approximation. Finding appropriate stopping criteria can further improve the runtime of the LRGMM. Also, the LRGMM uses a rank 1 perturbation. It is possible to create an adaptive LRGMM that adds additional ranks of perturbations. This would give the user more direct control over the balance between accuracy in the approximation of the covariance matrix and the computational cost of training. Finally, as previously stated, the overlapping clusters are determined by a simple heuristic. There is promise in exploring alternative heuristics for assigning data points to clusters. Using different metrics can also contribute to improving the quality of the clusters.

In our application to accent identification, we found that the LRGMM may suitable for native language identification. This allows identification of a speaker’s primary language based on their pronunciation of some secondary language. Also, it would be interesting to explore ensemble methods based on the ability of human experts to determine the strength of an accent, as in Zheng’s work [75]. This may include combining or hybridizing the ensemble techniques we explored in this paper.

In our application to social networking, a drawback of the SSDE/LRGMM
method is that it requires the given weighted graph be a connected component (i.e. no isolated nodes or subset of nodes). On the other side, the method also puts every node into a cluster. Allowing for outlier detection and isolated communities could considerably improve community identification.
BIBLIOGRAPHY


