

Identifier Labeling Using Graphical Models

Brian Ferris and Stephen Friedman

Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{bdferris, sfriedma}@cs.washington.edu

Abstract

In this paper, we apply Bayesian Networks to the labeling of arbitrary string identifiers from search results over a music database. We find that our models perform with a 58% labeling accuracy, with errors primarily occurring when labeling string data not been seen during training. We also present a method for searching potential labelings which attempts to address the exponential blow up of the labeling permutation space.

Motivation

In using many search engines, one's query may have several concepts associated with it. Consider the example of searching for a song on a music search engine. A search for a single keyword could return results of songs matching the keyword in the song name, the album name, the artist's name, or combinations thereof; all of these songs could be unique. As an example, consider three bands that all covered the same song; searching for the song title would turn results across all the bands. Additionally, there may be many entities in the search space which appear to be unique do to a corrupted portion of the search string, but in fact refer to the same real-world object. As an example, consider searching for "Abbey Road" and getting back matches for "The Beatles", "The Beetles", and "Beatles," where each record is a deviation on the same underlying record. These combinations of distinct results combined with misspellings across the same result cannot easily be teased into reasonable groupings that represent the underlying categorization. Our problem is fundamentally one of identifying these groupings in an automated way.

Problem Statement

One of the first steps in this process is organizing the data into a structured way. Given some noisy text identifier of a record, we want to be able to extract relevant pieces of that identifier for use in our classification of the record. Put another way, we want to label the portions of the identifier with the type of information they provide us. The particular example we will be applying

our methods to is the task of labeling the artist, album, title and other information about a song file given its filename. Labeling the parts of the identifier in this way allows us to use that information to categorize the record appropriately.

Background

Bayesian Networks

While a complete description of Bayesian Networks is beyond the scope of this paper, we will describe the key ideas here. A graphical model is a way to describe a factored joint probability over a set of random variables. The nodes of the graph represent the random variables, and arcs between nodes specify the factorization of the joint probability over these variables. A Bayesian Network is defined as a graphical model that is directed and acyclic where a directed arc between two nodes reflects the conditional dependence of the destination node on the source node. Therefore, each node has an associated conditional probability table (CPD) which holds the probability distribution over its domain for each value combination of its parents.

Naïve Bayes Naïve Bayes is a particular type of Bayes Network where there is one root node and all other nodes are leaves of that root. Thus there is exactly one edge leading from the root node to each of the other nodes in the graph. The root node is a hidden variable, and all of the leaves are observable. This structure makes the assumption that the nodes are independent given the root. For a more intuitive idea of what this means, consider the root to be your location, and the leaves are the items you have access to. Given your location as the office, there is some probability that you have access to your computer which is independent of whether or not you have access to your chair. For example, your computer could have crashed, so you can't use it, but this has no effect on where your chair is, assuming you are a reasonable person and didn't throw it out of the room in frustration. This model is nice because it is fairly efficient and can model many typical situations. It is efficient because the root node has a domain of n values and the leaves have domains of l_i values, making the root node CPD only a 1 by n

table and the leaf-node tables only n by l_i . If we were to say that one of the leaf nodes i was dependent on both the root and leaf j , the CPD would suddenly grow to a n by l_i by l_j table.

Learning in Bayesian Networks Learning is accomplished in Bayesian Networks by taking a prior distribution and a set of observations, each consisting of a set of values for some observable nodes, and updating the CPD tables so that the joint distribution more accurately models the distribution of the set of observations. There are several methods for doing this, and the one we used is known as Expectation Maximization, or the EM algorithm. We chose this because it can handle hidden variables, thus it can be applied to Naïve Bayesian Networks.

Inference in Bayesian Networks Once we have trained our Bayesian Network, we need to have a way to make use of it by answering queries using inference. Our queries of a Bayesian network take the following form: given a set of observed values of some nodes, what is the probability that observation would occur. Thus, basic inference in a Bayesian Network is the probability distribution obtained by marginalizing the full joint distribution over the values of the unobserved variables, then finding the probability entry corresponding to the value of the observed values. This can be potentially computationally expensive, but it is exact, and is how the naïve inference engine we used worked.

There are other approximate algorithms that work much faster, and they are based on the idea of sampling from the probability distributions. For each node with no parents, a sample value is generated based on the probability distribution for that node. These samples are used to reduce the CPD's of children, and once they have been reduced to a single distribution, a sample can be generated from that distribution. Any nodes whose values are given in the query are simply set to those values. This is repeated for a given number of samples, and the probability distribution is approximated by the distribution of values obtained by the sampling. Thus, the more samples you take, the closer your approximation becomes to the actual probability entry in the full joint distribution. For a more in depth treatment of Bayesian Networks, the reader is directed to AI: A Modern Approach, 2nd Ed. (?)

Bigram

Because we are dealing with noisy data, we wanted a fairly robust way to deal with misspellings and other small mistakes. A brief survey of the methods used in spell checking indicates that computing the character-based bigram across two words is a suitable method for dealing with misspellings. The similarity is computed by counting the number of times an ordered number of characters from one string occurs in a second string. This count is then normalized based on the length of the longest string. This prevents the similarity mea-

sure of a short string to a long string from dominating the similarity between two short strings. For a more complete description, including a comparison of bigram performance with unigram and trigram performance, the reader is directed to Word Discrimination Based on Bigram Co-occurrences (?).

Approach

Bayesian Networks

In our investigation, we compared three different Bayes Network topographies. The first was a simple network which modeled the heirarchical organization of Artist, Album and Title. This reflects the fact that there is a set of albums that an artist has, and given an album, there is a set of song titles one expects to see on that album. This model is depicted in Figure 1 and will be referred to as the Simple Model.



Figure 1: Simple Model Bayesian Network

Looking at the sample data we had, we noticed that there was very little album data contained in the filename identifier. There was, however, a lot of entries that included track numbers. The filenames would also contain some commentary useful for distinguishing similar songs, such as remixes or covers. Thus in our second model we attempted to optimize the structure for the data we had, and that model can be seen in Figure 2. We will refer to this model as our Optimized Model. Here you can see we removed the node for Album, directly tied the Title to the Artist, and added the auxiliary information of Track Number and Other. Other is a catch all label for commentary data contained in the identifier such as "live," "remix," and other extra info.

The final model we used was a Naïve Bayes model. Here, the hidden variable is called ID, which is a unique number assigned to each unique recording. Each of the possible fields is treated as a feature of the recording ID. This model can be seen in Figure 3 and will be referred to as the Naïve Model.

Labeling Inference

Once we had trained the appropriate Bayes Net, we needed to use it to perform our labeling task. First, we split the identifier field on non-alphanumeric characters into tokens. We can then assign a labeling to these tokens. For each possible labeling permutation, we can look up the probability of this labeling in the joint probability table for the model. The permutation with the highest probability is deemed to be the appropriate labeling.

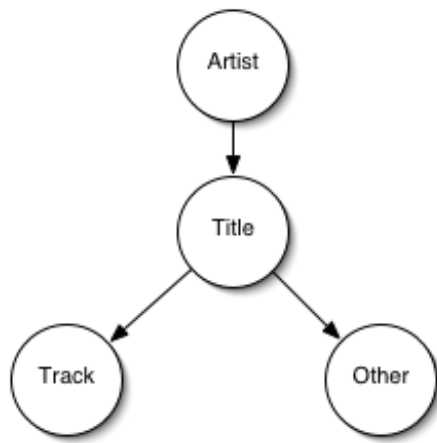


Figure 2: Optimized Model Bayesian Network

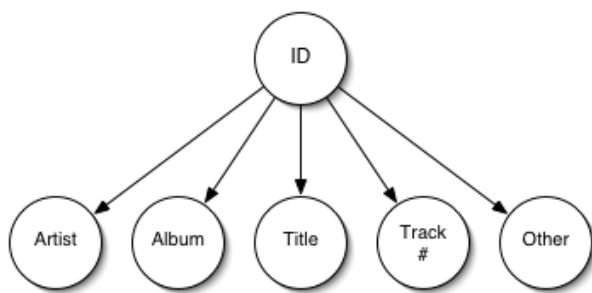


Figure 3: Naïve Model Bayesian Network

BEAM Tree Search When looking for the best labeling among all permutations, a very large search tree is generated. In this search, there are many low probability sub-trees that are explored, where we only want to find the highest probability assignment. In an attempt to optimize this search, we applied a BEAM search. In BEAM search, the top n nodes at are kept and further explored. Because we had an inherent ranking in the probability of a partial labeling, we were able to easily apply BEAM search to our search for the best labeling sequence. Inherent in our use of BEAM is the assumption that the partial labeling of the most probable labeling is also highly probable.

Naïve Inference One method of determining the probability of a labeling is Naïve Inference. Here, we compute the full joint probability distribution marginalized over the nodes that were not observed. We then look up the probability of the assignment of values to the remaining variables in this table to ascertain the probability of a particular labeling. While this is the complete and correct way to perform inference, it quickly becomes computationally expensive as the full table has entries for the full cross product of the random variable domains.

Gibbs Sampling Anticipating wanting to scale our solution up to larger databases in the future, we ran some experiments using a sampling based approximate inference method. The method we chose to try was the Gibbs Sampling method. The choice of this method is primarily because it was the method implemented in the toolkit we were using.

Robust Input-Handling

The astute reader will notice that ours is a very fragile system. If we have not seen a particular misspelling of a name in our training data, we will not be able to identify it, since it can not be paired with a value in our domain. In the Bayesian network, since it is a generative model, to properly capture all possible misspellings, we would have had to have nodes with a domain of all possible misspellings, or possibly all possible text strings. Clearly this is unrealistic, and thus we need a better solution.

Instead of having an exact match, we wanted to have an unknown string assume the value of the closest known domain value. To do this, we needed to have a measure of the similarity between two strings. One popular solution in spell checking is a bigram model, which we wish to apply here. Thus, when taking an observation to perform inference, if we don't get an exact match, we compute the bigrams across all given values in the node's domain and pick the closest match.

Learning

We take a supervised approach to learning the parameters of our Bayes Net: the conditional probability tables specifically. To do this, we take hand labeled

training data and use it as evidence in an expectation-maximization (EM) learning algorithm. The resulting domain of each node in network is composed of values seen during training.

Toolkit

In order to allow us the latitude of experimenting with several techniques in the short time allotted, we turned to pre-existing toolkits for working with graphical models. After looking at several toolkits freely available on the Internet, we chose to work with the Intel's Open Source Probabilistic Networks Library (?) which is a C++ library based on Kevin Murphy's Bayes Net Toolkit for Matlab (?).

Overview

These experiments will give us the ability to evaluate several models, a couple of different inference algorithms, and response to strict and more forgiving input methods.

Experimental Results

We obtained our training / test data set by instrumenting a P2P search client to save its search results. Specifically, we modified the open-source Limewire client (REF), which operates on the Gnutella P2P network. Our primary data set was a collection of 463 records returned for the query "Beatles" on Dec. 10, 2004 over the course of a five minute search. We then hand-labeled the components of the identifier string for each record. The tokens of the identifier were given the following labels: Artist, Album, Title, Track Number, and Other. The "other" label was used for labeling extra information such as "live" or "rare" that occasionally appeared in the id. Once labeled, the records in total uniquely identified 70 distinguishable songs.

Label	Percentage
Title	100
Artist	92.0
Album	2.6
Track #	16.0
Other	22.4
Track # or Other	30.2

Table 1: Statistical frequencies of labels over identifier set

The statistics captured in Table 1 should the relative frequencies of various labels across the search results. For example, every identifier mentioned a song title, and a large portion also mentioned an artist. The album, track number, and other labels were less frequently represented. We have included the number of occurrences of "Track #" OR "Other" for use in a subsequent discussion.

From the relatively frequencies of labels, the motivation for our migration from the Simple model to the

Optimized model is clear. There is little point including the "Album" field since it occurs so infrequently in the label data.

Model Comparison

To compare our models, we ran 100 experiment iterations on record data split randomly into $\frac{3}{4}$ training data and $\frac{1}{4}$ test data. For each iteration, we trained the model and then tested the accuracy of the results. Accuracy is measured as the percentage of correctly labeled records out of the total number of records. We performed the iterations for each of the three models. Note that our choice of a 3:1 ratio will be further rationalized in the next section.

Model	Simple	Optimized	Naïve
Mean	54.1%	58.4%	58.3%
StdDev	4.5	3.8	4.28
ConfInt	1.3	1.1	1.3

Table 2: Statistical comparison of three models at 3:1 train-test ratio

Table 2 shows the accuracy statistics for each of the three models. The most important observation to be made is that the three models perform almost identically. The Optimized and Naïve models are statistically equivalent (Student T=2.92 for 95% confidence and DF=2). The Simple model does perform worse but not dramatically so.

Train to Test Ratios

In order to evaluate the performance of our model in a variety of testing and training environments, we performed a set of experiments comparing model accuracy over various test/train ratios. Given a labeled data set, we randomly partitioned the data into two sets: one to train the model and one to test it. We varied the ratio of test to training data from 0.1 to 0.9 incrementally and evaluated the accuracy of the model with each training / test set.

Unsurprisingly, each model showed a general trend of increasing accuracy with larger training to test ratio. That is to say, with much more training than test data, the models generally performed better. Specifically, in Figure 4, we see a general trend of increasing performance. The graph demonstrates the performance of the model in a variety of training rich and training poor situations. Note that the performance of the model plateaus in the 0.5 range (LOG scale), which corresponds roughly to a 3:1 ratio of training to test data. This result was the basis of our decision to perform further training and evaluation using a 3:1 ratio in subsequent tests.

We compare both the Simple and Naïve models in Figure 5. The data set of each model is reduced to a polyfit trend-line in the graph to facilitate the differentiation of the two models. Each model exhibited a spread

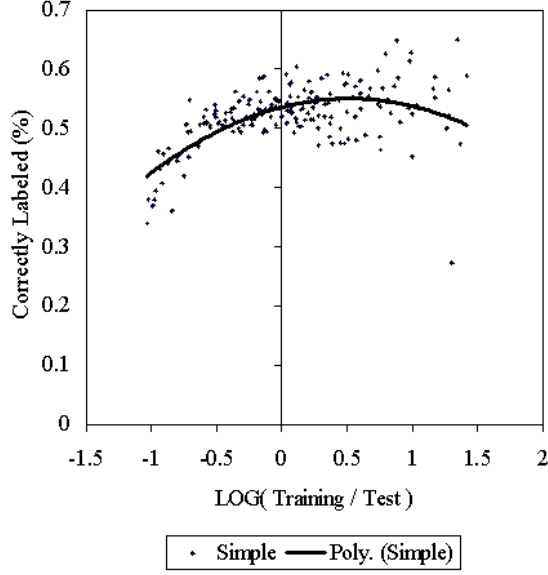


Figure 4: Simple Model and Polyfit Trend Line with Increasing Test/Training Ratio

of accuracies as the train-test ratio approached 3:1 (0.5 in logscale), so we chose a trend-line to compare the relative behavior of the two models. The trend-line is not a tight fit for either data series, but it does demonstrate the slight advantage the Naïve model holds over the Simple model. Note that we did not include the Optimized model because of the time complexity of its evaluation. Specifically for low train-test ratios, there are a large number of test records to be evaluated by the model. The poor runtime performance of the Optimized model made evaluation difficult as result.

Beam search

We ran a number of our tests using the full label space search and the beam search to evaluate both the accuracy and runtime of the beam search in comparison to the default. We evaluated the Simple model using the same incremented train-test ratio strategy evaluated in the previous section.

Figure 6 demonstrates the relative accuracy of beam search with a number of beam widths (10, 20, 30, 100) against full search. Note first that the general accuracy of beam search follows the same characteristic peak around the 3:1 train-test ratio that full search does. Also, note that the accuracy of the beam-search increases with increasing beam width. In fact, the beam search of width 100 appears to perform on average better than the full space search.

Figure 7 demonstrates the relative runtimes of beam search versus full search. We had hoped that beam search would perform faster than the full space search, but that does not appear to be the case. Notice that

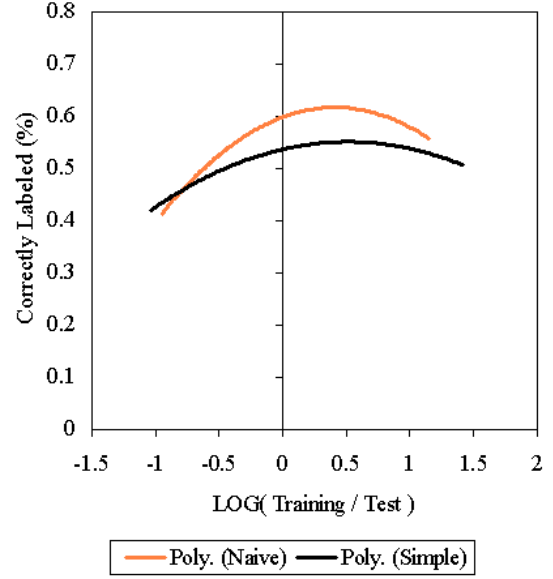


Figure 5: Comparison of Simple and Naïve Model (Polyfit) with Increasing Test/Training Ratio

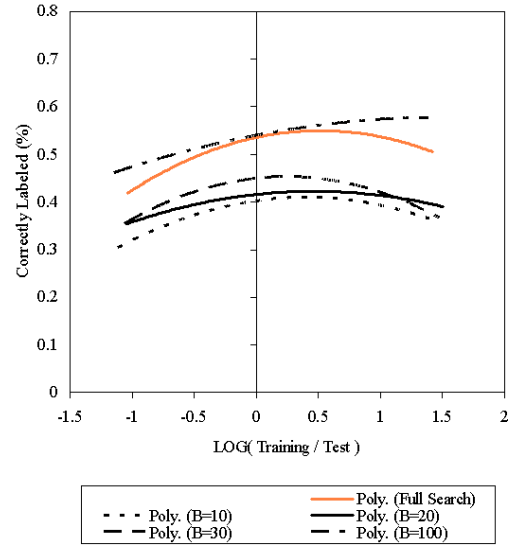


Figure 6: Comparison of **accuracy** versus train-test ratio using both full search and beam search with variable beam width (b)

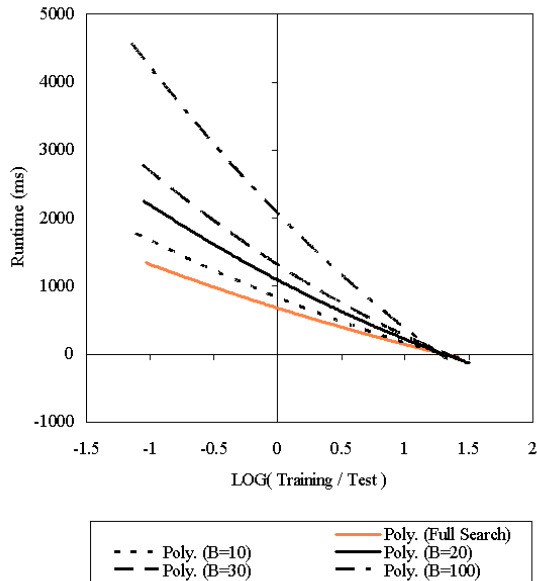


Figure 7: Comparison of **runtime** versus train-test ratio using both full search and beam search with variable beam width (b)

the full space search has a lower average runtime than an equivalent beam search of any width. Also notice that increasing beam width results in longer runtime, with the $b = 100$ beam being the longest running.

Bigram Usage

Our use of Bigram matching for allowing label matching for novel misspellings reduced the accuracy of our labeler to zero in all cases. We were surprised by this result, since we assumed Bigram would allow us to work past the inflexibility of our model to match new and novel misspellings. The reality is somewhat disappointing. Examination of our labeler traces revealed that the flaw was in our decision to take the best bigram match for a string not in our training domain, regardless of how low the probability for the match.

Consider the string "The Beatles - Help!" The correct labeling is clearly "Artist: The Beatles" and "Title: Help!" Indeed, our labeler would consider such a combination and score it with some non-zero probability. However, it would also consider the labeling "Artist: The Beatles Help!" Such an artist domain value clearly does not exist, so a Bigram match would be performed, yielding "The Beatles" as a best replacement. Now our labeler is simply calculating the $P(\text{Artist} = \text{"The Beatles"})$, which is very high considering our training data is composed almost entirely of songs by the Beatles. As such, the labeler finds that "Artist: The Beatles Help!" is the best labeling, which is incorrect.

Inference method - Gibbs vs. Naïve

In an effort to reduce the runtime complexity of our labeling algorithm, we attempted to move away from the complete naïve inference algorithm and make use of a stochastic inference algorithm. Our toolkit specifically implemented Gibbs Sampling. We evaluated our labeling algorithm using both naïve inference and Gibbs Sampling inference and found an order of magnitude decrease in accuracy when using the Gibbs Sampling method. We feel that increasing the number of samples used by the Gibbs method should improve its accuracy, but we had trouble adjusting the sample size using the toolkit API.

Conclusions

The most startling result of our experiments was the lack of differentiation in any of our label models. The Optimized and Naïve models were statistically indistinguishable and the Simple model was only a few percentage points off. More importantly, none of our models performed particularly well, where well is defined as approaching 90-100%. What is the explanation for these poor performance statistics? Upon examination of our labeling approach, it becomes clear that our implementation is not all that different than a straight table lookup. Examination of the joint-probability tables for all of our models revealed that the probability matrices were quite sparse. That is to say, probabilities typically approached 1 for one labeling combination and zero for all others. In essence, our probability tables are effectively straight string to label lookup tables. Because our record set was relatively small, the domain for each label had few distinct values and there was little chance for a mismatch of any probability.

Considering this observation, the next question is why each model performed dramatically less than a straight table lookup, which we theorize would approach 100% accuracy. The answer lies in the motivation for the use of Bigram matching. Our model is very inflexible to domain values that it has not previously observed. Thus, any token from a song identifier that has not been seen in the training data cannot be labeled using our model. We later examined the number of such cases in a 75% training, 25% test data experiment, and found that a full 36% of test records fell into the category of "previously-unseen domain values". Thus, our models will fail to characterize 36% of all test records on average. We can now start to see that the combined 58% accuracy of our models plus the 36% of all test records that it cannot classify in any case accounts for almost 100% of the total test set.

The revelation concerning both the lookup-table-like performance of our models and the large quantity of records with unseen domain values was a large motivator for our exploration of the Bigram matching algorithm. Unfortunately, as elaborated in the results section, our application of Bigram failed to address the unseen domain value problem. We believe that Bigram

is still appropriate, but our failure lies primarily in using Bigram to pick a single replacement value as opposed to using Bigram to generate a probability distribution over possible values. In essence, Bigram could be considered a sensor model of sorts in the way that it assigns relative similarity measures between a given string and a dictionary set.

The failure of our Beam search to optimize the labeling runtime is now clear in hindsight. When comparing the full-search method over a variety of data models, it was not the label permutations that were computationally expensive, but instead the calculation of the joint probability distribution for a given model. With the full-search, this calculation is only performed for leaf nodes. However, in our beam search, the probability calculation is performed repeatedly for each partial-labeling depth. The benefits of reducing the factorization space are thus completely overshadowed by the increased cost of probability calculation.

Suggestions for Future Research

While we did obtain some results, they were far from ideal. There are several candidates for improvement and further research that were not explored due to time constraints. One topic open for study is the choice of the Bigram-based input reconciliation. Being able to integrate new and novel values using a Bigram similarity distribution over existing domain values still seems a novel idea. Perhaps there are better methods of choosing the closest domain value from arbitrary text. In the process of looking for ways of making medicinal names less confusing, Grzegorz Kondrak and Bonnie Dorr developed a similarity measure that may be useful here that known as BI-SIM.(?) Because this was developed to identify the perceived similarity or confusability between two names, we feel it would be useful for identifying the term that someone intended if they were confused.

It is important to note that our experiments with labeling of MP3 search results was motivated by our ultimate goal of identifying results that referred to the same underlying song, allowing for unification of the data set displayed to the end user. In addition to the actual file identifier which we have been attempting to parse and label, many MP3 files contain additional structured meta-data in the form of an ID3 tag, which potentially identifies the artist, album, title, and other attributes of a given song. While this seems an ideal source for identification data, our collection of search results indicates that ID3 were incomplete in the majority of cases. So while ID3 tag information is not a replacement for the file identifier, it is an obvious supplement for use in song identification. Additionally, unsupervised learning may be possible using this labeled ID3 tag information.

Much of our work for this project was inspired by recent research in unifying the record space of the Citeseer online citation database. Specific research by Lafferty, et. al. (?) used conditional random fields (CRFs) to

accurately label arbitrary citation strings. We feel this is an obvious choice for further exploration. Unfortunately, our existing graphical model toolkit does not directly support CRFs so more work is required for an implementation in this area.

Acknowledgements

1. The Intel Open Source Probabilistic Networks Library (?) move the complete implementation of a simple graphical model out of the realm of "exceedingly difficult in the given period of time" and into the realm of "maybe we can even try two models..."
2. Kevin Murphy's list of graphical model toolkits was useful for evaluating the available options and features of existing toolkits.
3. We had a number of technical conversations Jayant Madhavan regarding graphical models and their application to this project.

Appendix A - Contributions

Brian Ferris wrote the results and conclusion section of the paper, while Stephen Friedman wrote the background and approach section. Stephen did the initial work on learning using the toolkit and Bigram implementation, while the label matching system and the inference performed within was written by Brian. Both spent much time hashing out various model, implementation, and project issues. Much was learned by all.