On Word Prediction Methods

Darren Kuo

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-147
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-147.html

December 16, 2011

Acknowledgement

I would like to express my deepest thanks to Professor Satish Rao for all the guidance for the past 2 years. Both the framework and our methods are joint work with Chenyu Zhao, Di Wang, James
Cook and Satish Rao.

On Word Prediction Methods

Darren Kuo
Department of Computer Science
University of California, Berkeley
Berkeley, CA 94720

December 16, 2011

Abstract

This paper evaluates prediction and topic modelling methods through the task of word prediction. In our word prediction experiment, we compare some existing and two novel methods, including a version of Cooccurrence, two versions of K-Nearest-Neighbor method and Latent semantic indexing[5], against a baseline algorithm. Furthermore, we explore the effects of using different similarity functions on the accuracies of our prediction methods. Finally, without much modifications to the framework, we were also able to perform tag classification on StackOverflow posts.

1 Introduction

Our paper presents a novel simple task of word prediction given a piece of text. The usual experiments including clustering of documents[4], querying for related document[8], similarity of documents[9], and capturing the topics given a set of documents[5][6]. Our experiments differ from these conventional experiments. We were interested in the relationship of the language of documents at the word level and the topics of the documents.

Initially, we were interested in methods for clustering documents. While exploring, it became clear that we need a simple way to measure our how useful modelling methods were in performing a certain task. We posit that performance on a simple task would yield insights into the power of each method. Our approach was to measure the effectiveness of our prediction methods by the accuracy of word prediction given a piece of text. We note that the task itself translates into problems of inherent interest; Specifically, we evaluate the methods on the problem of automatically tagging forum posts.

2 Motivation

We can view each word as a sample from different topics. The word "cluster" might be related to the topic "cloud-computing" and might also be related to the topic "theory". The word "cluster" would have conditional probabilities $P[cluster \mid cloud\ computing]$ and $P[cluster \mid cloud\ computing]$, both in the form of $P[word \mid topic]$. We can then see that the probability of the word 'cluster' can be calculated from:

$$P[cluster] = \sum_{i\ \epsilon\ topic} P[cluster \cap i] = \sum_{i\ \epsilon\ topic} P[cluster|i] * P[i] \tag{1}$$

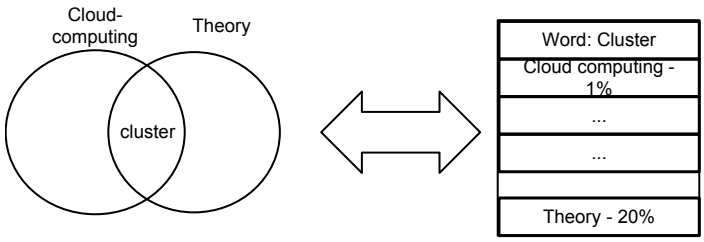

Figure 1: We may view the relationship between words and topics differently. Topics are made up of different words, while we may also view words to being some mixtures of topics.

The process of generating a document is often viewed, for example, in LSI, LDA, and pLSI models, as choosing words from the topics that describe the document. See [17] for a more detailed discussion. The ability to predict a next word for a document by a topic model indicates that it has captured the salient features of the document. This is the main motivation for using this task as a way to measure the efficacy of these topic modelling methods.

2.1 Related Work

The pLSI and LDA methods both produce a parameterised generative model for their corpus. One measure that they use is the perplexity of the generated model for the data in the corpus. This measure computes the probabilities over a subset of the documents, and the higher the computed probability, the better. This measure is interesting and a reasonable way, perhaps, to compare methods that produce models, it remains difficult to reason about. In the extreme, one must trust that the models do not simply generate a probability of 1.0 for each document.[1] Second, it is difficult to compare the understanding gained by these methods over, for example, nearest neighbor methods since the latter does not produce probabilities. A simple prediction task allows such comparisons on a common unimpeachable scale.

We note that both pLSI and LDA have been evaluated in prediction tasks previously through the use as a preprocessing method to produce a feature vector for other prediction methods. They seem effective and useful in these contexts both in terms of speed and effectiveness, but again, it becomes difficult to reason about the success of the model with complicated interaction of these methods. If the methods do indeed produce a good model, the model should be able to be used more directly to produce predictions. The task we define allows one to do so in a simple direct manner. Indeed, the task was very much motivated by the world view of these methods; the next word is chosen for a document according to the mixture of topics in the document. Thus, a good topic model should do well on this natural task.

We believe fully understand the power of these methods with respect to their own raison detre, could help improve them to be eventually useful in common in traditional information retrieval include document retrieval [**?**] and sorting document according to relevance [13].

[1] They don't do this, but the calcutions and methods are complicated enough for one to wonder about whether the semantics of the numbers produced correspond to probabilities across the different methods.

3 The Problem

We started out using words to help categorize a given piece of text. However, our definition of categorizing text here is complicated, since predicting a word in efforts to categorize a piece of text does not exactly categorizes the text under a specific topic. In other words, we are not given a set of topics and assigning a topic from that set to a given piece of text; we are assign a combination of topics, represented by a word, to a document.

The task involves predicting an extra word given a block of text. We were intereseted in this task as a way to measure the effectiveness of our prediction methods (i.e. how well we captured the topic model). Although this task seems useless, unlike other algorithms which has a clear application, the task later opened up an opportunity to attempt tag prediction with our framework. We will discuss about this more useful task of tag prediction on Stackoverflow in a later section 7.

4 Our Approaches

Our approaches to this word prediction task are simple clustering methods that mainly look at the words in the text corpus. In this section, we will start by introducing some terminologies and data structures. Then, we will explaining our similarity functions and ranking functions and describe our prediction methods in detail. We will present our experiments and results in later sections.

4.1 Data Structures and Terminologies

4.1.1 Word Vector

Since our experiments works exclusively with individual words in some text, it is reasonable to have a represenation for a piece of text. The most basic and essential data structure for our experiments is the word vector. A word vector D_i is an one-dimensional array with length T, where T is the total number of words. The word vector D_i represents the document i. $D_i[j]$ denotes the term frequency for the word j in document i. In this paper, we use the terms *Document* and *Word vector* inexchangable, but they are represented the same.

The word vector is mainly used for similarity calculations, which is cruical for some of our prediction methods. Therefore, for optimization purposes, we also store the word vector in a more spare represention as a Map. Further optimizations are discussed in 4.1.3.

4.1.2 Paper

Usually, academic papers come with more than just the lines of words, but also citations. A Paper consist of a word vector along with citation references, which are currently only used for the K-Nearest-Neighbor with Citation method. As we find more data source, we expect to add more properties to this Paper class to include all aspects out the data.

4.1.3 Cache

To optimize the calculation of document similarities, we created a cache, implemented a HashMap, to cache the results. In small experiments, we maybe assume that the number of documents is

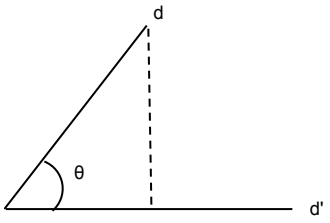

Figure 2: Cosine distance between the two documents.

less than $2^{16} = 65536$. We can fit the id of two documents into an `integer` of 32-bits as the key and the similarity score described in 4.2.1 as the value. For larger experiments, we maybe use a `long` of 64-bits. The number of total usable keys for any experiment is larger than the current implementation if we take into account the fact that only 0.8 the total paper indices are training document, then we can address $0.8 * 0.2 * D$ instead of D^2, allowing us to use a bigger D. With this optimization, we were able to speed up the runtime any of our prediction methods that uses one of the similarity function.

We also implemented a cache for document distance based on the citation network using the similar Scheme described above.

4.2 Functions

Our methods heavily depends on the similarity function, which determines similiarity of two documents. Several methods also relies on a ranking function to rank the importance of each word. In this subsection, we will presented the functions that are included in our framework.

4.2.1 Similarity Function

• **Cosine Similarity** $sim_{Cosine}(D_1, D_2)$

The simplest form of a similarity function between two word vector, which is defined as vector of term freqencies, is the dot product of the vectors, which is the cosine distance of the documents.

$$sim_{Cosine}(D_1, D_2) = \sum_{i=0}^{T} D_1[i] * D_2[i] = D_1 \cdot D_2 \tag{2}$$

We could also add in information about the individual word itself, such as the idf of a word. The inverse document frequency (idf) measures the number of occurence of a word in the entire corpus[**?**] and represents the general importance of the word in the corpus.

$$idf(w) = \log \frac{|Docs|}{\{d : w \epsilon Docs\}} \tag{3}$$

$|Docs|$ represent the total number of documents and $\{d : w \epsilon Docs\}$ measures the occurence of the term w in text corpus. Using the idf, we can redefine the similarity function above as:

$$sim_{idf}(D_1, D_2) = \sum_{i=0}^{T} D_1[i] * D_2[i] * idf_i \tag{4}$$

• **Euclidean Distance** $sim_{Euclidean}(D_1, D_2)$

The next simple function is the Euclidean Distance. The Euclidean Distance is standard for many Geometry problems. Given two document vectors, this similarity function is defined as:

$$sim_{Euclidean}(D_1, D_2) = \sqrt[2]{(\sum_{i=0}^{T}(D_1[i] - D_2[i])^2)} \tag{5}$$

In our experiments 6, we found the simple Cosine similarity works better than the euclidean distance. Initutively, we can see that it is true. Consider the following example: If we have two documents on the topic "biology" that are similar and the word "cells" appears a few more times than in the other document, this different is amplified by the square operation. Therefore, making the documents not as similarity, since a smaller value here means the documents are more similar.

• **Jaccard Coefficient[9]** $sim_{Jaccard}(D_1, D_2)$

The Jaccard coefficient measures the similarity between two documents by taking the ratio of the intersection of the words and the union of the words. The function is defined as:

$$sim_{Jaccard}(D_1, D_2) = \frac{\vec{D_1} \cdot \vec{D_2}}{|D_1|^2 + |D_2|^2 - \vec{D_1} \cdot \vec{D_2}} \tag{6}$$

For all pairs of word vectors, $0 <= sim_j(D_1, D_2) <= 1$ is true. A high Jaccard coefficent implies the documents are very similar, while a lower coefficent means it's completely different. The two extreme cases (0 and 1) means that the two documents are the same or the documents shares no words in common.

• **Pearson Correlation[9]** $sim_{Pearson}(D_1, D_2)$

The last similarity function that we will explore is the Pearson Correlation Coefficient. This coefficient is a measure of the correlation of two document, as the name suggested.

$$sim_{Pearson}(D_1, D_2) = \frac{T * \sum_{t=1}^{T} D_{1,t} * D_{2,t} - (\sum_{t=1}^{T} D_{1,t} * \sum t = 1^T D_{1,t})}{\sqrt{[m\sum_{t=1}^{T} D_{1,t}^2 - (\sum_{t=1}^{T} D_{1,t})^2][m\sum_{t=1}^{T} D_{2,t}^2 - (\sum_{t=1}^{T} D_{2,t})^2]}} \tag{7}$$

The equation above and the correlation formula look very alike and put it in terms of expectation:

$$corr(D_1, D_2) = \frac{E[(D_1 - \overline{D})(D_2 - \overline{D})]}{\sqrt{E[(D_1 - \overline{D})^2]E[(D_2 - \overline{D})^2]}} \tag{8}$$

We will later see in 6 that some of these similarity function are better than others on our datasets.

4.2.2 Ranking Method

The ranking method is responsible for ranking the words for prediction given a list of word vectors. Many of our methods use a similar simple ranking method, which could be outlined as follows:

1. Combine the word vectors by adding them pointwise. For each word, sum up the frequencies across the vectors.
2. Sort the combined word vector by the frequency in ascending order.
3. Prediction function will pick words from the front of the vector.

However, some clustering methods might decide to scale the frequencies for each paper before combining the word vector to incorporate some properties specific to that paper. The best example is the K-Nearest-Neighbor with Citation network method.

4.3 Prediction Methods

In this section, we will describe the various methods we use for our experiments5. Most of our methods only work with the actual text of the data, usually treating the text as a sparse word vector. The only method that uses extra information, the citation network, is the KNN with citation4.3.3.

4.3.1 Baseline Algorithm

Our baseline algorithm is a rather simple one. It only works with the words of the text corpus. The algorithm is as follows:

1. Collect frequencies for all the words in the text corpus.
2. Rank all the words in the text corpus by its global occurence.
3. When predicting new words for a given set of words, we will return the highest ranking word that has not appeared in the given set of words.

The algorithm guesses the most popular word in the text corpus that has not appeared in the given words of the testing document. Even though this algorithm very simple, it provides a reasonable baseline for comparison in our experiments.

4.3.2 K-Nearest-Neighbors

K-Nearest-Neighbors is a well known statistical approach to Automatic text categorization [16], along with other learning task. The problem of text categorization is mildly similar to our experiment. We decided to experiment with KNN to perform our task of predicting words given some text.

1. Calculate the similarity between the testing document and all training documents.
2. Keep the top K documents that are most similar to the testing document.
3. Combine the K documents into one document by summing their word vectors pointwise.
4. Predict by picking the word with the highest count in the combined word vector.

We may view KNN algorithms as a local version of the Baseline Algorithm. As the Baseline effectively only guesses the most popular of the whole training set, KNN first selects the top K documents that are similar to the testing document and then guesses the most popular words out of these top K documents. Furthermore, if we view each word as a mixture of topics, we may also see each combined frequency as a weighting for the particular mixture of topics[18], represented by a word. Nevertheless, each word would represent different mixture of topics and the notion of topics becomes less clear.

It's also clear that the performance of K-Nearest-Neighbor depends on the K value that we pick. Furthermore, the K could vary across different dataset. Enumerating through the K values, we found the best K for each of our dataset. Generally, the K-Nearest-Neighbor method performs decently well for our task, see more in 6.

4.3.3 K-Nearest-Neighbors with citation network

The citation network of the text corpus can improve the results of our experiment[14]. It's a popular belief that papers that cite each other has some overlapping topics[7]. It is common for authors of papers to read related work in their fields to provide some background for their work. Therefore, assuming authors do some paper reading before writing their own paper, if a paper is related to an author's paper, it is more likely that he/she will cite the paper over all the other papers. On the other hand, if the author didn't cite the paper, it might be that the paper was not too relevant to the author's paper.

The data from 5.1.2 has provided us with a relatively dense citation network of all the papers. In this method, we aim to use the citation information to improve the existing K-Nearest-Neighbor method. We introduce a new ranking method to help us incorporate the ideas. The new method differs only by the scaling of word vectors, metioned previously in 4.2.2.

1. Scale each vector by $\frac{1}{d}$
2. Combine the word vectors by adding them pointwise. For each word, sum up the frequencies across the vectors.
3. Sort the combined word vector by the frequency in ascending order.
4. Prediction function will pick words from the front of the vector.

d denotes the graph distance between this paper and the given testing paper in the Citation graph. This ranking function differs by the weighting of individual word vectors. In the normal K-Nearest-Neighbor algorithm, we treat all top K documents as the same. In this method, we weight documents differently by their importance suggested by the citation network graph.

Based on the thought process from above, if a paper has similar language, but is not directly cited by the testing paper, then its words should be less important. However, if the paper is cited directly by the testing paper, then its words should not be down-weighted (i.e. $d = 1$). With this heuristic, our method improves by $\sim 2\%$ on the selected few datasets5.

4.3.4 Cooccurrence (Naive Bayes)

Our Cooccurrence algorithm measures $P[newword|giventext]$ using the *naive Bayes assumption*[2]. For any testing document, we want to calculate $P[w|G]$ for each $w \epsilon W$, where G is the given set of

words in the testing document and W are all the words in the text corpus that is not in G. This calculation expands out to the following, where $\#(\cdot)$ denotes the count for all possible n word combinations.

$$P(w|G) = \frac{P(w \cap g_1 \cap g_2 \cap g_2 \cdots \cap g_n)}{P(g_1 \cap g_2 \cap g_2 \cdots \cap g_n)} \approx \frac{\#(w \cap g_1 \cap g_2 \cap g_2 \cdots \cap g_n)}{\#(\cdot)} \tag{9}$$

However, the joint probability is very hard to calculate [15]. The most obvious reason is that there usually isn't enough data to provide a reasonable count to approximate the real probability for all the given words. Those exacts words might not have appeared together in any document the training dataset. The more subtle issue is the amount of memory needed to store the count for all possible combinations of words is huge. However, we can proably employ the same integer packing scheme described in 4.1.3 to solve the memory problems.

In our experiment, we viewed each word in the testing document as evidence for a word w, that is $P(w|g)$. We then posit that more evidence increases the chance that w is in the document. If we use the *naive Bayes view*, in a fashion, we get

$$P(\neg w|G) = \prod_{g \,\epsilon\, G} (1 - P(w|g)) \tag{10}$$

For small values of $P(w|g)$, we get that

$$P(w|G) = \sum_{g \,\epsilon\, G} P(w|g) tf(g) \tag{11}$$

where we use the fact $\prod_i (1 - x_i) \approx \sum_i x_i$ when the x_i's and $\sum_i x_i$ are small.

The $tf(g)$ appears under the view that extra appearances of g introduces an extra evidence for w. Moreover, the tf term improved performance in preliminary experiments.

We found that this did not work so well by itself, but has a base for then scaling by idf was our best method overall. That is, we used:

$$s(w) = idf(w) \sum_{g \,\epsilon\, G} P(w|g) tf(g) \tag{12}$$

as the scoring function for a word. We posit that this evidence based score is more informative for rare words than common words as one might expect from the approximations used above.

Lastly, we used another heuristic to keep our scores from fluating too much. We limited our guesses by our similarity-based prediction methods to only guess words that appeared in more than 4 documents. This made our similarity score a lot more stable.

4.3.5 LSI

Our LSI [5] implementation uses an approximated Single-Value Discomposition to do dimension reduction. We are given a D by T matrix, which is the Document by Term matrix, denoted M_C. Each row of the matrix is a 1 by T word vector, representing a paper. Through the process of Single-Value Decomposition, we can reduce the document-by-term matrix to document-by-k matrix.

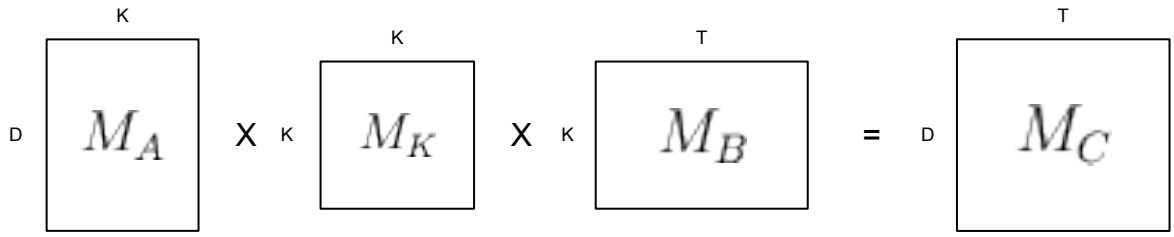

Figure 3: The document by term matrix can be approximated through an approximated Single-value decomposition. Keeping only the top k eigenvectors.

After performing SVD on the document term matrix, we have a D by K matrix M_A, a K by K matrix M_K and a K by T matrix M_B. We may view M_A as the reduced document term matrix; instead of T terms, we know have K "terms". Matrix M_K is the diagonal matrix with the eigenvalues on the diagonal. Now, we can do dimension reduction on any document word vector V through some matrix mulitplication operations (where V_k denotes the vector after dimension reduction and V_T denotes the original document as a 1 by T word vector):

$$V_k = V_T * M_B^{-1} * M_K \tag{13}$$

After reducing a word vector's dimension down to K (resulting vector denoted by V_K), we can use the 1 by K vector to predict some words.

$$P_T = V_k * M_B \tag{14}$$

From here, we proceed similarly to a ranking method described in 4.2.2 and sort the vector P_T and return the word with the highest value. The values in the P_T vector are estimated term frequency for the respective term.

Again, similar to KNN, this method performs reasonably well for some K. We did some walking on the K value to determine the best K. The best K depends on the dataset that we are using (I.e. How many total unique words, etc).

5 Experiment setup: Next word prediction

In this experiment, we are given some words in the testing document and our methods will attempt to predict more words that belong in that testing document. In this section, we will present the setup of our framework and the datasets that we experimented on.

5.1 Data

We picked a dataset that was used in other papers. The MED, CACM, CRAM and CISI dataset are well-known dataset in the text mining field. We also started working with a big dataset from arnetminer that contains citation network information.

5.1.1 MED, CACM, CRAN and CISI

These four datasets (Classic4) are heavily used in text mining as benchmarks. These datasets are original used for a querying task, which involves returning a set of related documents given a

Dataset	Documents	Citations
MED	1033	N/A
CRAN	1398	N/A
CACM	3204	46566
CISI	1460	109678-
ArnetMiner	1572277	2084019

Figure 4: Document count and Citation relationships for various datasets

query. The results of these query are produced via survey and has been studied extensively[5]. They are average good size text collection for our experiments. CACM and CISI both has citation information that could be used to experiment with K-Nearest-Neighbor with Citation, while MED and CRAN still serve as a good benchmark for our methods. In this paper, we will only present experiment results of the MED dataset. Future work should be done on the remaining three datasets.

5.1.2 ArnetMiner

Arnetminer[1] is a website that aims at data mining academic professional networks[12]. They build citation graphs from academic papers, extract researcher's profile from the web and provides searching services on academic papers. Users can view the authors that publish together or authors that cites other authors. The data provided by ArnetMiner only had the words in the abstract in each paper. However, it provided citation network information for us to experiment with KNN. The results might be even better if we were to experiment on the full papers.

5.2 Framework

Our framework takes in a text corpus, which is a collection of related documents and runs the specified methods on the data. The document collection is randomly splitted into two groups, training set and testing set, depending on the input parameters. The training documents are then passed into our prediction methods for training. The testing documents are again randomly splitted into two sets of words, the given words and the testing words. The given words are words that are used for prediction, while the testing words are effectively viewed as answer keys to our experiments. It is expected that a higher testing percent increases the performance of our methods.

There is a clear seperation between training and testing in our framework. When the prediction methods are training, they would have access to the testing data. Furthermore, when the prediction methods are predicting new words, they won't have access to the answer key.

Each prediction method does training in the construction, where it is given the training data. All the methods implement the procedure `predict`, which takes in the testing data and a k value that specifies how many words to predict. The framework is responsible for evaluating the predictions returned by each method. For each test document, the framework will check the number of predictions in the testing words.

6 Results: Next word prediction

In our experiments, we train on 0.8 of the collection of documents and test on 0.7 of the words in each testing document.

6.0.1 MED

In this section, we will present some results for the experiment we described in 5. We will first compare the results of using different similarity functions for KNN. Then, we will discuss about the effects of different K values for LSI and KNN respectively. Finally, we will end with the results of predicting more than one word using various methods.

• Different similarity functions affects our methods

We discover that Jaccard works the best out of the 4 similarity function for our KNN methods. The Jaccard coefficent offers a slight increase over Dot product and Pearson, and a big increase over Euclidean distance. From this point on, we will use the Jaccard Coefficent as our primary similarity function for our experiments. See results on 5.

It is currently still unclear to us why Jaccard does better than Pearson and why euclidean does significantly worse. Our theory is that it has to do with the different characteristics of the dataset. More future work should be done on these similarity measure to understand clearly when to use each of these.

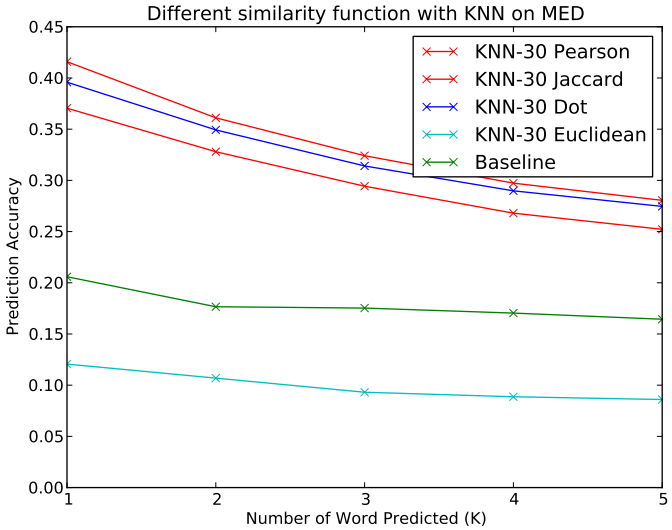

Figure 5: Different similarity functions with KNN on the MED dataset.

• The effect of different K parameters for KNN and LSI

KNN depends highly on the number of neighbors we use for prediction. We found that the optimal K value differs from datasets to datasets. The general trend is that K increase as the num-

ber of training documents increases.

On the other hand, LSI seems to be dependent on the number of unique terms, since we are performing dimension reduction on each word vector. The more unique words generally correlates to the need for more dimensions to capture the necessary important information in the text.

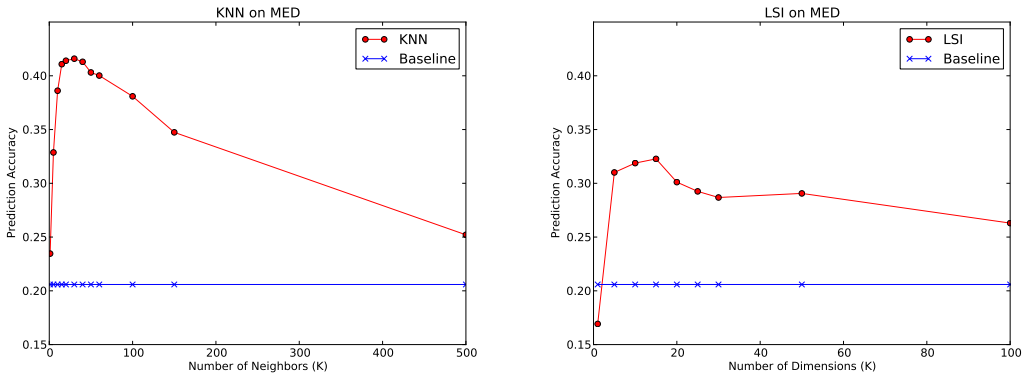

Figure 6: Experiment results on differnt parameters for KNN and LSI. The K-Nearest-Neighbor and Latent Semantic Indexing methods both reply on the K parameter, which are k neighbors and k dimensions respectively. Plotting the accuracy of each method over the k values allows us to find and use the best K parameter to predict words.

• **Predicting more than one word**

The first word usually has the best quality when compared to the other words predicted by our methods. As part of the discussion for which K to pick for KNN, the best K for predicting one word generally reminds the best for predicting additional words. Same for LSI, the best K dimension seems to continue to outperform LSI with other different dimensions as we increase the number of words we predict.

6.0.2 ArnetMiner

We also run our experiments on the ArnetMiner dataset. The dataset was too large for our machines, so we randomly sample 5000 documents from the dataset. However, note that the above sample method of the dataset destroys the citation network structure of the data. This effect decrease the amount of information we gain with our KNN-with-Citation method. To address the issue of random sampling destroying citation network, we came up with the following scheme for sampling.

1. Randomly sample k documents in the dataset.
2. Perform depth first search on the citation network from each of those documents.
3. Stop when you have n documents.

KNN with citation did not improve over the KNN method with the randomly sampled ArnetMiner data. See Figure 9. However, when we sampled using breadth first search, picking $k = 200$

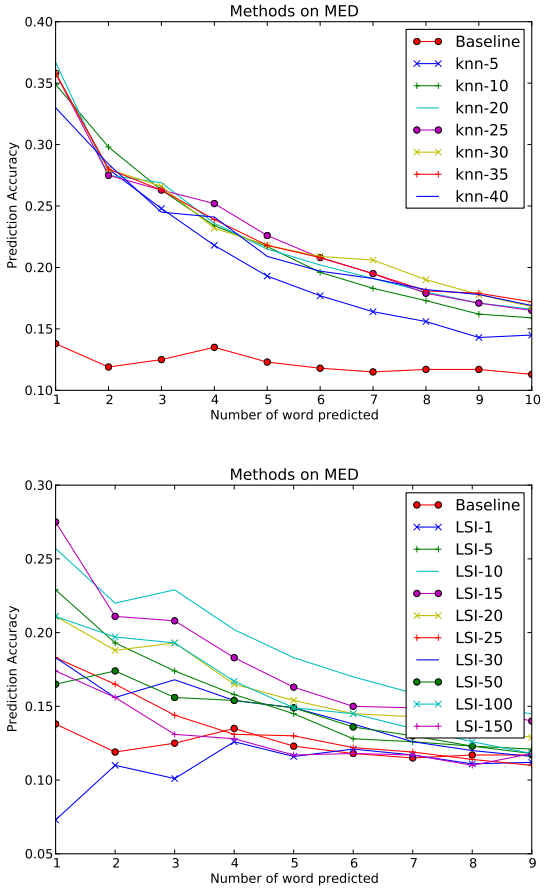

Figure 7: Predicting more words decreases the accuracy of the words

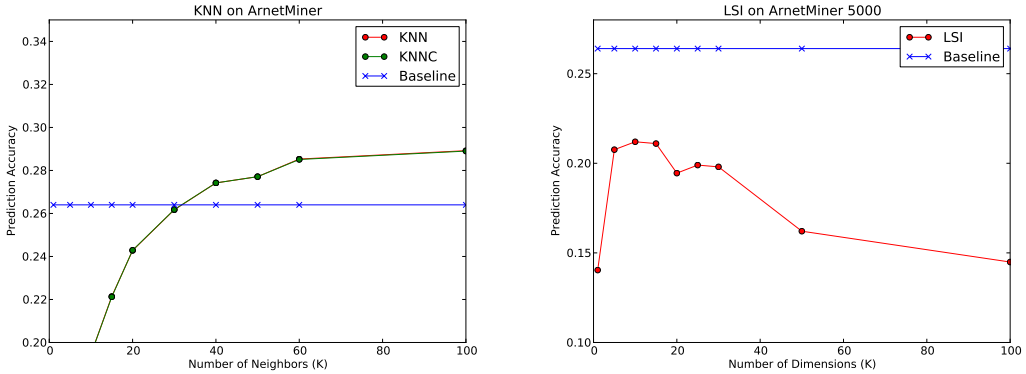

Figure 8: Word Prediction experiment with KNN, KNN-C and LSI on Arnetminer 5000 dataset over different k (neighbors and dimensions)

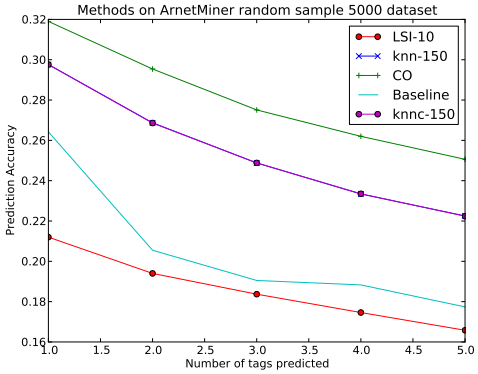

Figure 9: Word Prediction experiment with KNN, KNNC LSI, CO and Baseline methods on Arnetminer 5000 dataset

and $n = 20000$, KNN with Citation improves over the plain KNN. See figure 10.

It is also surprising the LSI does not do well on this dataset. The simple Baseline method seems to beat LSI by a big margin. However, we should keep in mind that ArnetMiner is a dataset with a lot of different topics. It is possible that Baseline is doing well because of some common word that should be designated as a stopword for this dataset. On the other hand, our other methods do not perform as well, since it's hard to selectively capture the topic model given so many topics.

6.0.3 Stackoverflow

We got **StackOverflow** data mainly for the tag classification task. However, it would also be interesting to see the results of our methods on this data for this word prediction task. We run all our methods on the StackOverflow for the sake of comparison. The results are displayed in table 12.

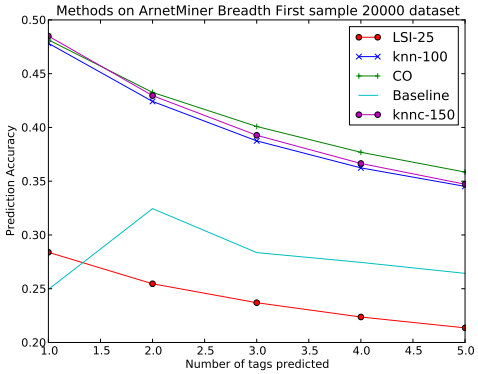

Figure 10: Word Prediction experiment with KNN, KNNC LSI, CO and Baseline methods on Arnetminer 5000 dataset

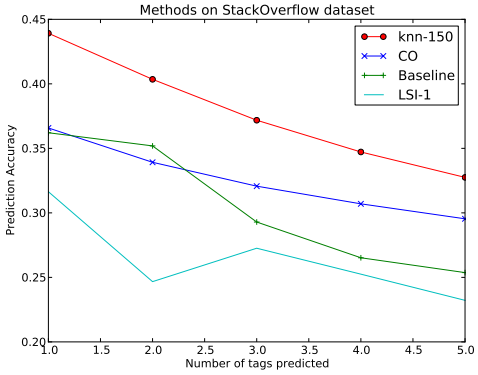

Figure 11: Word Prediction experiment with KNN, LSI, CO and Baseline methods on StackOverflow dataset

There was a surprising result that requires more investigation. Cooccurrence does not outperform other methods on this dataset. One theory that we had is the type of language this dataset is very different from the other datasets we have. StackOverflow datasets are all Question-Answer type causal language, while the Arnetminer and Classic 4 dataset are written in a more formal fashion. The details of why Cooccurrence does not continue to do well with this dataset need to be evaluated in more depth.

7 Experiment setup: Tag classification

In this section, we will describe the tag classification experiment, one useful application that could be implemented easily using our word prediction system. The increase amount of categorized information makes tag classification a real world application that could be applied to forums, blogs and even academic paper site [10].

Methods	MED	ArnetMiner 5000	ArnetMiner BF	StackOverflow
Baseline	18.6%	26.42%	24.9%	36.2%
KNN*	37.6%	29.8%	46.4%	43.9%
KNNC*	N/A	29.8%	47.5 %	N/A
LSI*	29.3%	21.2%	28.4%	34.6%
CO	42.5%	31.9%	48.1%	36.6%

Figure 12: Best results predicting one word on the Med data with training percent of 0.8, testing percent of 0.7 KNN* and LSI* denotes the best accuracy by all the K values respectively.

7.1 Motiviation

The motiviation behind doing tag classification is very simple: important words that are used through out a post or an answer might as well be a tag. We can treat tags as just another word and restrict the predictions to just tags.

We know our prediction methods does well compare to our baseline on the word prediction task. However, we also wanted to try out our system on this tag classification task to see if the outperforming trend is common across different task.

7.2 Data: StackOverflow

We found all of **StackOverflow** forum post up till September 07, 2011 avaliable through the **Creative Commons Data Dump**[3], which is an effort to make all Stack Exchange's site publicly available to encourage work on the data. All the post were formatted in XML, which makes it relatively easy to parse and extract information. The various metadata, which have not been studied by us, are also included in the dump.

The StackOverflow dataset contains about 2 million complete post, including question and answers. We ran our experiments on datasets of 10000 posts, due to memory issues. As mentioned, we have not fully look into the metadata to create a citation-network-like data to enable the use of KNN with Citation Network.

Unfortunately, we currently only have one source of data for this experiment. In the future, we hope we can find more datasets that already has tags available with the text for more comparisons.

7.3 Framework

We share the same framework as the word prediction experiment. We only had to make several minor additions to our code base.

1. Changed the framework to handle a more compact data format to prepare for the large amount of data from StackOverflow.
2. Allow the prediction methods that use all the words in the testing documents as given words (i.e testing percent = 1.0)
3. Restrict prediction methods to only predict tags. Each method will gain another procedure that reflects this restriction.

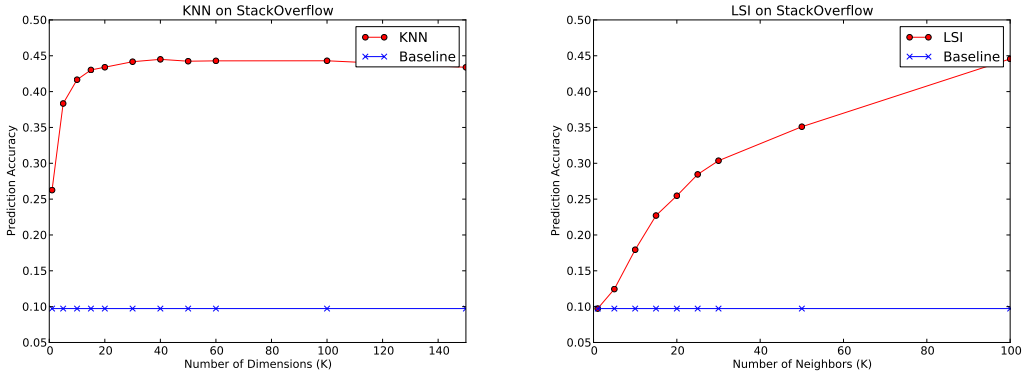

Figure 13: KNN and LSI on StackOverflow dataset

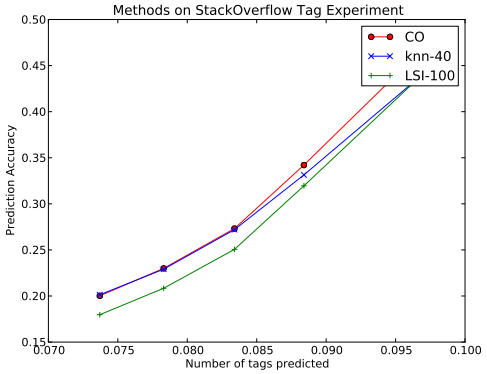

Figure 14: KNN, LSI, CO and Baseline predicting multiple tags on StackOverflow dataset

8 Results: Tag prediction

Unlike the other experiment, the Baseline method performs fairly poor on this tag classification task. Since the given text in the testing documents contains no tags and the Baseline method always guess the most popular tag, the 9% performance of Baseline tells us that approximately 9% of the tags are the most common tag. Due to the low performance of Baseline, the experiment is an interesting one. The experiment shows that we cannot do well simply by guessing the most popular tag, which is the best move if we know nothing about the data. With our more sophisticated methods, we outperform Baseline by a huge margin.

All our methods seems to outperform the baseline method. In particular, Coccurrence does the best on the tag prediction task. This is not an entirely surprising results. We may view tags as meaningful words that are not too popular except among the post that are strongly related to that tag. This claim that Cooccurrence is expected to do decent is based on the fact that our Cooccurrence method take the extra step of using the idf of a word when predicting. This causes the method to downweight tags that are popular but non-specific.

Methos	Accuracy
Baseline	9.7%
CO	47.3%
KNN*	44.5%
LSI*	44.6%

Figure 15: Performance of each prediction method for predicting one tag on the tag experiment

9 Future work

As mentioned in the paper, future work should be done on using the citation information on other dataset to experiment with the performance of the K-Nearest-Neighbor with Citation method. Furthermore, the ranking function of the KNNC should be studied more in depth. The current function is a heuristic that provided a decent increase in accuracy over the normal KNN. However, we do not believe it is the best use of the information provided in the citation network. Other source tagged data should be used to test the tag classification aspect of our framework. Perhaps, a SVM implementation of a tag classification system will be useful.

The performance of the Cooccurrence method dropped drastically when using the StackOverflow data to perform the basic word prediction experiment. Furthermore, the LSI method was outperformed by the Baseline method on the ArnetMiner 5000 dataset. Investigating what properties of the dataset made these two methods' performance fluates across these datasets an interesting and important task.

The Coocurrence method could be further extended by using tri-grams and 4-grams instead of just bi-gram cooccurrence. Although this extension will lead to a slower runtime, we believe it will make use of information hidden in variable length phrases instead of just word. For instance, the word "uncertainty" might appear with the phrase "Quantum Mechanics" often. However, with just bi-grams, we would only gather counts for the pairs `(quantum, uncertainty)` and `(mechanics, uncertainty)`, while the counts for `(quantum mechanics, uncertainty)` is more reliable.

Drawing directly from our experiments, we could also use Boosting on our existing methods and view it as a linear programming problem as attempt to improve up on our current methods.

In retrospect, the random sample of ArnetMiner could have been done better. The sample size of 5000 seems too small for a collection that contains 1.5 million documents. Perhaps, smarter way to sample the dataset will yield better and more accurate results.

Lastly, we had numerous memory and runtime limitations that could be improved with a distributed version of each of our algorithms. There is also a fast approximation of KNN graph construction for high dimensional data [11].

10 Conclusion

Our simple novel experiment provides a way of measure our prediction methods. The experiment emphasize on predicting words, which are mixtures of topics. Although it is hard to compare with other existing method due to the difference in task, we believe our task provides a new way to

evaluate these commonly used methods. Furthermore, our tag classification system implemented on top of our word prediction framework provides a way to perform an useful task of classifying text under some given set of tags.

11 Acknowledgement

I would like to express my deepest thanks to Professor Satish Rao for all the guidance for the past 2 years. Both the framework and our methods are joint work with Chenyu Zhao, Di Wang, James Cook and Satish Rao.

References

[1] A citation network dataset. http://www.arnetminer.org/citation, January 2011.

[2] K. N. Andrew McCallum. A comparison of event models for naive bayes text classification, 1998.

[3] J. Atwood. Stack overflow creative commons data dump. http://blog.stackoverflow.com/category/cc-wiki-dump/, September 2011.

[4] M. E. Benjamin C.M. Fung, Ke Wang. Hierarchical document clustering using frequent itemsets, 2003.

[5] H. T. S. V. Christos H. Papdimitriou, Prabhakar Raghavan. Latent semantic indexing: A probabilistic analysis, 1998.

[6] M. I. J. David Blei, Andrew Y. Ng. Latent dirichlet allocation, 2002.

[7] T. H. David Cohn. The missing link - a probabilistic model of document content and hypertext connectivity, 2001.

[8] T. Hofmann. Probabilistic latent semantic indexing, 1999.

[9] A. Huang. Similarity measures for text document clustering, 2008.

[10] I. V. Ioannis Katakis, Grigorios Tsoumakas. Multilabel text classification for automated tag suggestion, 2008.

[11] Y. S. Jie Chen, Haw-ren Fang. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection, 2009.

[12] L. Y. J. L. L. Z. Z. S. Jie Tang, Jing Zhang. Arnetminer: extraction and mining of academic social networks, 2008.

[13] R. M. T. W. Lawrence Page, Sergey Brin. The pagerank citation ranking: Bringing over to the web., 1998.

[14] X. G. Minh Duc Cao. Combining contents and citations for scientific document classification, 2005.

[15] M. L. Paul Dagum. Approximating probabilistic inference in bayesian belief networks is np-hard, 2003.

[16] S. Tan. Neighbor-weighted k-nearest-neighbor for unbalanced text corpus, 2005.

[17] Y. Yang. An evaluation of statistical approaches to text categorization, 1998.

[18] X. L. Yiming Yang. A re-examination of text categorization methods, 1999.