THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

WORD CLUSTERING THROUGH COMMON SENTENCE ANALYSIS

KELVIN SHI
SPRING 2013

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Computer Science

Reviewed and approved* by the following:

Daniel Kifer
Assistant Professor
Thesis Supervisor

Jesse Barlow
Professor
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

To develop a program that could interpret the semantic meaning of text, we decided to first develop a word clustering program to determine the general topic of words. Word clustering could provide us with the context information of words, which could then be used as a resource to develop further applications. Understanding the general topic of a word has applications in several problems. For example, we could perform definition disambiguation. Some words have multiple definitions. The word “scale”, for example, has different meanings in each of the contexts: “scale a wall,” “fish’s scale,” and “weighing on a scale.” Understanding the topical context of the word would help classify the definition of each instance of the word. Furthermore, word clustering could also allow the computer program to better generate fluid sentences. By determining which words tend to belong together, the program can then place these contextually similar words together to create more natural-sounding sentences. To perform clustering, we found the frequency of each word relative to every other word, and we used these frequencies to find the characteristic relationships between certain words. We then used these relationships to cluster the words by grouping the words with the strongest relationships. To evaluate our algorithm, we decided to compare the results of our algorithm to the commonly used Latent Dirichlet Allocation (LDA) algorithm. After applying a double-blind test, we found that our algorithm was comparable to existing LDA techniques and is thus suitable for our purposes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Daniel Kifer for guiding me through the entire thesis process. The processing of performing research and writing a thesis was entirely new to me, and at the beginning, I was confused and lost. Thanks to Professor Kifer's insightful advice and patient guidance, I was able to make it this far. Through this journey, I have learned invaluable skills and engaged in some of the most interesting work in my life.

I would also like to thank Professor Kifer's machine learning class and the Computer Science and Engineering department at Penn State for participating in the evaluation of my algorithm. Without their help, I would not have been able to properly test my algorithm and compare it to an existing technique.

Chapter 1

Introduction

Background

Humans use and listen to natural language every day, yet actually implementing algorithms for computers to understand natural language (i.e. writing and speech) is still a challenge that researchers are trying to solve every day. Part of the challenge involves the tremendous amounts of ambiguity, implication, and subtext in everything we say. Anything we say is encoded in symbols that can even be difficult for humans, let alone deterministic computers. Natural Language Processing is a field that involves studying how to parse and use the information from natural speech. Having computer programs that can interface with humans using natural language has many applications, including machine translation, virtual assistants, and information storage and retrieval. Controlling programs using plain speech would allow software interfaces to be more responsive to non-programmers. Imagine being able to simply tell Microsoft Word to format the page a certain way, without needing to click through menus to find what you need. Some of the notable problems within the field of natural language processing include automatic summarization, coreference resolution, discourse analysis, machine translation, and named entity recognition. For this project, we decided to focus on keyword clustering. Given a collection of documents, we want to determine the principal groups of keywords that compose the documents. Each group of keywords consists of words that occur under similar contexts.

Motivation

Automated essay writing is a project that we are currently working on. We hope to make a computer program that would be able to write papers automatically after receiving instructions. Such a program would be an invaluable tool for those who have difficulties writing English fluently. After working with many graduate students, we have discovered that many of them, especially not native English speakers, find writing research papers difficult. Sometimes even the most brilliant scientists and engineers, with many great ideas and results, can have a difficult time writing up their work. A program that could help them write would allow them to increase their productivity and put their time to better use.

One component in automated essay writing involves understanding the contexts of different words. After all, writing is a complicated exercise that requires deep semantic learning of texts. For sentences to sound natural, vocabulary needs to match together flawlessly and convey meaning. To meet this requirement, we need to determine which words tend to go together in similar contexts. Thus, for the purposes of this thesis, we decided to focus on keyword clustering. The purpose of our project is to develop a program can take a sentence, a paragraph, or a document and cluster its corresponding keywords and thus determine its topics.

Chapter 2

Literature Review

In the literature, other research groups have tackled similar word clustering challenges. Many of these approaches are supervised, meaning that they require a certain amount of existing training data. Because most large bodies of text, called corpora, have not been previously hand-labeled, unsupervised techniques are much more popular. Current the most popular technique is the Topic Models technique using the Latent Dirichlet Allocation (LDA) algorithm.

Prior to LDA and Topic Models, other techniques have been used to cluster words. Duda et al used classical Bayesian classifiers to perform pattern classification [Duda et al. 2000]. Yang used supervised learning techniques for automated categorization into classes and topics [Yang et al. 1999]. Lagus et al used nonlinear dimensionality reduction via self-organizing maps to represent term vectors in a two-dimensional layout [Lagus et al. 1999]. The Salton et al's latent semantic indexing if-idf scheme reduced documents of arbitrary length to fixed length lists of numbers [Salton et al 1983].

Latent Semantic Indexing (LSI), a technique using singular value decomposition (SVD) was first introduced by Deerwester et al. in 1990 [Deerwester et al. 1990]. While the method worked well, the resulting representation was hard to interpret, and the model assumed a Gaussian noise model for the word-count data.

Probabilistic latent semantic indexing (pLSI) is a probabilistic alternative to projection and clustering methods like LSI [Hoffman et al. 1999]. pLSI involves a multinomial probability distribution over words, where each word is modeled as a sample from a mixture model. Each document is a probability distribution on a fixed set of topics. While the technique works fairly

well, it is susceptible to over-fitting.

For this project, the primary precedent has been Topic Models using the Latent Dirichlet Allocation (LDA) algorithm. Latent Dirichlet Allocation (LDA) was first introduced in the paper by Blei et al. in 2003 [Blei et al. 2003b]. LDA is a generative machine learning technique where the document is assumed to be a probability distribution over topics, and each topic is a probability distribution over words. The technique uses a three-level hierarchical Bayesian model and approximate inference techniques based on variational methods and expectation maximization algorithms for empirical Bayes parameter estimation.

The priors in each relationship are assumed to follow a Dirichlet distribution. In 2004, Rosen-Zvi et al. used an expanded version of LDA for clustering authors of text documents [Rosen-Zvi et al. 2004]. In the paper by Rosen-Zvi et al., they used a generative algorithm using a "bag of words" model similar to topic models. In this manner, topic-word, and author-topic distributions were learned in an unsupervised manner using a Markov chain Monte Carlo algorithm. Through this model, they were able to represent the high-dimensional vectors associated with the words in a document to a lower-dimensional space. Their generative model involved a two-stage stochastic process. They applied their algorithm to 150,000 abstracts from CiteSeer, 1740 papers from the Neural Information Processing Systems Conferences, and 121,000 emails from the Enron corporation [Rosen-Zvi et al. 2004].

Some groups have extended or applied variations of LDA and applied it to different problems. For example, Kataria et al. used the Wikipedia-based Pachinko Allocation Model (WPAM) a semi-supervised hierarchal model to disambiguate entity references in text [Kataria et al. 2011]. Griffiths and Steyers 2004 used Gibbs sampling and the Markov Chain Monte Carlo technique to estimate the parameters of LDA when applied to large data sets [Griffiths et al. 2004]. Erosheva et al. in 2004 extended the LDA model to model both text and citations

[Erosheva et al. 2004]. Blei and Lafferty in 2006 and Li and McCallum in 2006 examined the correlation between topics and looked at time-dependent topics [Blei et al. 2006] [Li et al. 2006]. Mei and Zhai in 2007 incorporated context analysis in the LDA model [Mei et al. 2007]. Blei et al. further looked at using a Bayesian hierarchical variant of LDA called the nested Chinese restaurant process to learn topic hierarchies from data [Blei et al. 2003a]. Ha-Thuc et al. utilized a one-scan topic model to improve the scalability of LDA for corpora over 2 GB in size [Ha-Thuc et al. 2008]. LDA topic models have been applied to a few applications that are not even text-based. Some of these applications include images [Monay et al. 2007] [Quelhas et al. 2007], videos [Niebles et al. 2006], genetics [Pritchard et al. 2000], and wearable sensor data [Newman et al. 2007] [Huynh et al. 2008].

Chapter 3

Design

The ultimate goal of our project is to create a program in the field of artificial intelligence capable of analyzing text and writing papers. Our general approach to this project has been not to create one monolithic program, but rather use a distributed architecture, where each component can run independently and generate its own resources. Thus, each component can be changed, improved, and developed iteratively without any impact on the rest of the program. For this thesis, we decided to focus on one particularly important component, the keyword categorizer and relationship determiner. By categorizing and determining the relationships between different words, it becomes possible to fulfill other tasks such as word sense disambiguation and named entity recognition.

Before taking on this task, we prepared by fetching 140,000 articles from Wikipedia, gathered from recursively fetching articles from subcategories starting with the top level category “Computing”. We used the Python Natural Language Toolkit's word tokenizer and sentence tokenizer to tokenize each article into lists of words, where each list represents a sentence. We then counted the frequency of each word to generate a frequency hash-table. We separated the words into different 'tiers' depending on their frequency (how often they appear in the text). The Table 3-1 shows the number of words that appeared in each frequency range.

Table 3-1. Word Frequencies. Number of words in each frequency range.

Tier	Frequency Range	Number of Words
1	X >= 2500	3082
2	2500 > X >= 500	6486
3	500 > X >= 50	27557
4	50 > X >= 10	60894
5	X < 10	376330

From this point on, we used only the Tier 1 words, as the "working set". First we wanted to find the frequency that each word appears in the same sentences as each other. Thus, we created an N x N table, known as the contextDict, where N is 3082 (the number of words in the working set). From here on, we will refer an entry of the table as contextDict[x][y] where x is the row and y is the column. Whenever encountering one word from the working set (word1), we would go through the same sentence, looking for any other word from the working set (word2). For every word2 that we encounter, we would increment contextDict[word1][word2]. Thus, we ended up with a table with the frequency of each word in the working set in the context of every other word in the working set. The Python code for this portion is in Appendix A.0.

Next, we created an N x 1000 table (topContextWords), where the rows represent the words in the working set. We created a list of the values of the columns in each row of the contextDict and retained only the top 1000 most frequent words that occurred in the context of the corresponding row word. These 1000 words will now be known as the "characteristic words" of each row word. The Python code for this portion is in Appendix A.1.

Next, we created an N x 1 table (niFreqDict), where the rows represent the words in the working set, and each entry represents the number of words for which the corresponding row word is a characteristic word. From the niFreqDict, we created a table of weights, where each entry is simply 1 divided by corresponding entry in the niFreqDict. This table represents the uniqueness of each word in the working set. Thus, a word that appears in the characteristic words of every word in the working set would be weighted much less than a word that appears only in a few words' characteristic words. When calculating the relationship between two words, these less frequently appearing words would be a much better indicator of closeness than ones that appear in every words' characteristic words. Thus, the weights of the words ranged from 1 to 1/3082. The Python code for this portion is in Appendix A.2.

We then found a relationship value between each word in the working set with each other word in the working set, by looking at the characteristic words they had in common, and adding together the weights of those characteristic words. We put these relationship values in a new N x N table called distVectors. Below in Table 3-2, in sorted order, are the top 10 most related words for a few randomly chosen words from the working set. The Python code for this portion is in Appendix A.3.

Table 3-2. Word Relationships. The top related words for four randomly chosen words in our top vocabulary.

Word	Related Word	Relationship		Word	Related Word	Relationship
Center	Institute	0.81		consist	consists	0.6
	National	0.79			consisting	0.54
	University	0.78			contain	0.52
	College	0.77			contains	0.51
	Centre	0.77			array	0.5
	California	0.74			containing	0.5
	Society	0.73			blocks	0.5
	Sciences	0.73			represented	0.49
	Department	0.73			arranged	0.49
	Founded	0.72			divided	0.48
David	John	0.9		pure	equations	0.5
	Peter	0.88			quantum	0.48
	Michael	0.88			equation	0.48
	Paul	0.86			natural	0.48
	James	0.86			classical	0.48
	Robert	0.85			interpretation	0.48
	George	0.84			fundamental	0.48
	William	0.83			theory	0.47
	Martin	0.81			numerical	0.47
	Brian	0.81			molecular	0.47

As one can clearly see from the results, our program fairly accurately found the words most closely related to each word in the working set. For example, for the word “David”, our program found many other common male names such as “John”, “Peter”, and “Michael” with a very high relationship value. For the word “pure”, the program found words related to math and science. For the word, “Center”, the program was able to find words related to institutions. For the word “consist”, the program found words related to organization. While it is difficult to truly have an objective evaluation of the results, just from observation, the results appear to be fairly high quality. Eventually, we hope to eliminate words that are approximately the same word, such as “equation” and “equations”, as well as perform some additional tuning. We believe the quality

of the results can be partly attributed to the fact that we performed the analysis on only the most common words, which are relatively noiseless. Had we used sparser data, we suspect that we would have had more noise and more outliers in the results.

Now that we had a table with the relationships between each pair of words, we were ready to perform clustering. The first step would be sorting all the relationship pairs in order from high to low, using only the top 150,000 word pairs. We then began developing the clusters, using each pair in order. If neither of the words in the pair was in an existing cluster, we created a new cluster containing only those two words. If one of the words was in an existing cluster and the other was not, we added the unclustered word to the same cluster as the word in the existing cluster. If both words were already in different clusters, we merged the two clusters if the size of neither cluster was greater than the average cluster size. If one of the clusters were greater than average size, we would not perform the merger. The code for this portion is in Appendix A.4. Twenty randomly chosen items in the five largest clusters are shown in Table 3-3.

Table 3-3. Clusters. Twenty randomly chosen words in the five largest clusters.

Cluster	1	2	3	4	5
Twenty Random Words from Cluster	['computing', 'formula', 'differ', 'R', 'may', 'd', 'trees', 'uniform', 'unique', 'value', 'interval', 'identity', 'tree', 'every', 'O', 'column', 'finite', 'element', 'double', 'partially']	['ships', 'everything', 'revealed', 'Earth', 'follow', 'becomes', 'back', 'sending', 'sent', 'forced', 'destroyed', 'tried', 'pilot', 'begin', 'wants', 'Empire', 'enough', 'return', 'takes', 'sees']	['lives', 'tactical', 'defeating', 'flag', 'alien', 'airport', 'defeated', 'true', 'dog', 'dark', 'buildings', 'acting', 'chance', 'around', 'party', 'century', 'Doctor', 'saved', 'sea', 'construction']	['causes', 'monster', 'advance', 'worlds', 'potentially', 'heads', 'moves', 'stages', 'drop', 'quick', 'loss', 'move', 'competitive', 'failure', 'causing', 'next', 'completing', 'shoot', 'moving', 'enemies']	['Sciences', 'Association', 'funding', 'serving', 'year', 'Centre', 'Professor', 'Club', 'graduate', 'College', 'National', 'hired', 'former', 'Laboratory', 'department', 'invented', 'budget', 'MIT', 'Society', 'Congress']

As one can clearly see from the results above, the first cluster has a distinct “graphs” theme, the second cluster has a “war” theme, the third cluster has an “adventure” theme, the fourth cluster has a “game” theme, and the fifth cluster has an “institutions” theme. The results are clearly distinct and fairly high quality. However, the 2nd and 3rd clusters do have some theme overlap.

While most of our results look promising, it is very difficult to perform any objective metrics to evaluate the results of the clustering. Without a gold standard for comparison, we can only qualitatively observe the results. One possible evaluation we could use to obtain some quantitative metric would be to devise a survey where volunteers may compare the results of the

clusters generated our program to the clusters generated from the same data using other algorithms such as LDA [Blei et al 2003]. Thus, we developed a simple web interface to dynamically display two corresponding clusters and ask the volunteers to determine which one was more coherent and less noisy.

Chapter 4

Evaluation

To evaluate our algorithm, we compare our clustering algorithm to the Latent Dirichlet Allocation. To do this, we first download the articles that Blei et al. used in their demo as well as their resulting clusters [Blei et al. 2003b]. For each article, we apply our own algorithm and compare the resulting clusters to those obtained from the demo. To perform the comparison, we find the cluster from our set that best matches a given article, as well as the cluster from the LDA set that best matches the same article. To determine the best matching cluster, we enumerate the number of instances that a word from each cluster occurred in the article. The cluster that has the most instances is considered the winner. Thusly, each article is grouped with its best-matching cluster from our set and its best-matching LDA cluster.

We then create a website, where a randomly generated article appears along with its top clusters. The article is chosen at random among the X total number of articles. The left/right order of the two clusters is also randomly generated and saved. The website user can now pick the cluster that better represents the article at hand. The result is then saved. Thus, we essentially have a double-blind test where neither the user nor the tester knows the order of the clusters because the order is randomly generated and recorded. A screenshot of the website is shown in Figure 4-1.

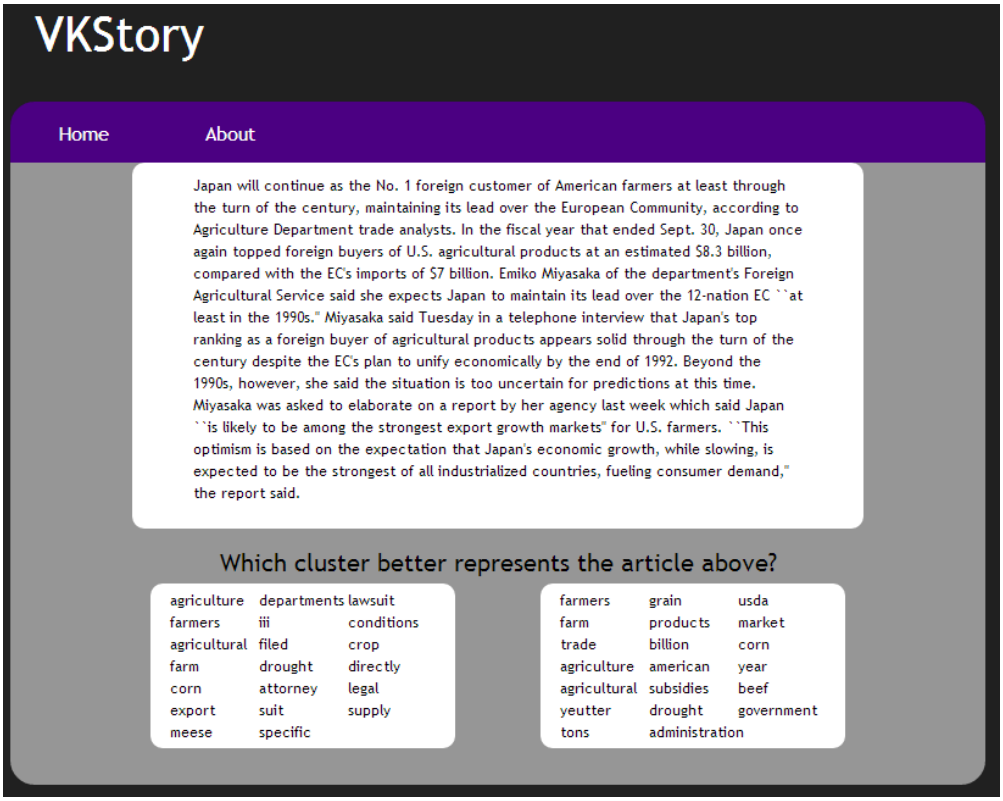

Figure 4-1. Evaluation Website. A screenshot of the website used in evaluating our keyword clustering algorithm.

After obtaining the results of the evaluation, we then calculated the p-value of the results to determine the statistical significance of our results. To calculate the p-value, we created a computer simulation where we performed 100,000 trials of N fair coin flips where N is the number of experimental data points. For each trial, we counted the number of heads obtained. We then calculated the number trials where the number of heads is greater than the number of positive quantities and divided the value by the total number of trials.

Chapter 5

Results

The results of our evaluation show that the number of people that prefer our keyword clusters is close to the number of people that prefer the topic models. The number of people that prefer the topic models is only slightly higher. Out of 73 total results, 38 results favor the topic models, while 35 results favor our keyword clusters. The results are displayed in Table 5-1.

Table 5-1. Evaluation Results. The number of trials that favored the given algorithm.

Cluster	**Number of Results Favored**
Topic Models (Reference Set)	38
Keyword Clusters (Test Set)	35

In Figure 5-1, we see the results of the simulation. The p-value for topic models being better than keyword clustering is only 0.32, which means that using a p-value threshold of 0.01, the evaluation fails to reject the null hypothesis.

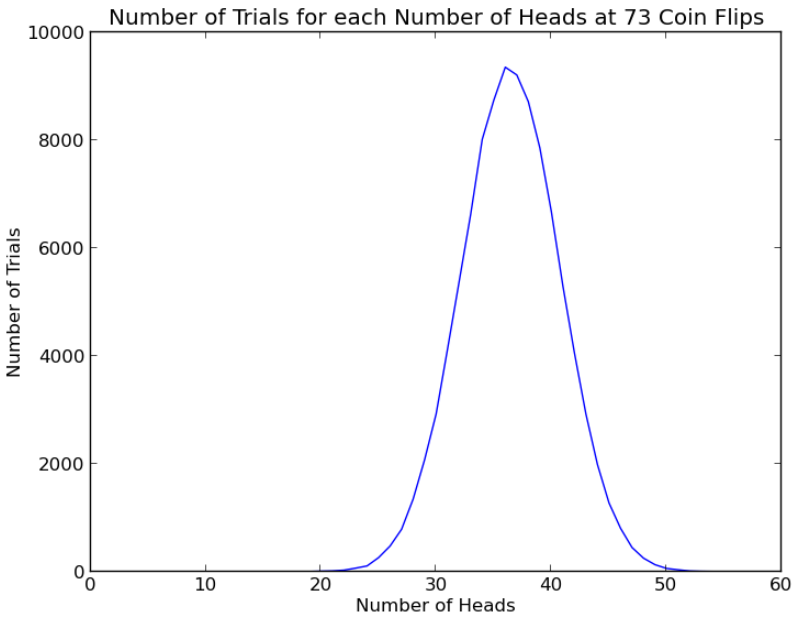

Figure 5-1. Probabilistic Simulation. The results of 100,000 trials of 73 coin flips.

Chapter 6

Discussion

The results from word clustering after 73 evaluations seem to indicate that topic models cannot be shown to be better than our custom word clustering algorithm. However, our approach has several advantages over the topic model approach:

It is simple. The topic models algorithm uses a Latent Dirichlet Allocation, a complex algorithm based on a generative model and probabilistic inference techniques. On the other hand, our keyword clustering algorithm simply uses frequency to determine the keywords and co-occurrence in the same sentence to determine the clusters.

It is customizable. Because our algorithm is simple, we can easily customize and alter our algorithm to cluster only nouns, only verbs, or even noun-verb pairs. Using LDA, we would have a much more difficult time adjusting the algorithm to suit our own particular application.

It is scalable. The complexity of our algorithm is O(mn) where m is the number of words in each sentence and n is the number of sentences in the corpus. Thus, assuming that the number of words in each sentence is not incredibly high, our algorithm is very efficient. Accordingly, it was able to cluster words from almost 150,000 Wikipedia articles in only a few hours. Meanwhile, the LDA algorithm using even a small number of topics requires polynomial time to complete [Sontag et al. 2011].

As we see from the evaluation results, in the accuracy of representing the article set, the two results from the two algorithms are fairly close. LDA is a mathematically a very elegant algorithm. However, for the purpose of our application, our algorithm has a number of advantages. It is simple, customizable, and scalable.

Using our clustering algorithm, we could better determine the relationships between words. English is a language where one word can have multiple definitions; the same word in different contexts could have completely different meanings. Our word clustering algorithm would effectively allow the computer program to analyze the different contexts of words to determine the probability that the words have multiple meanings. Our algorithm could also help distinguish the meanings of the words based on context.

Another application for our program would be named entity recognition (NER). Based on the context of each proper noun, we could determine whether or not the noun is a person, a place, an object, or a concept. For example, a proper noun in the same sentence as words such as "professor," "reading," or "talking" would most likely be a person, whereas a proper noun in the same sentence as a word such as "river," "table," or "city" would most likely be a place.

A further application for our program would be in actually generating sentences. Sentences generally sound smoother when words that usually occur together are place together. For example, we use the word "who" when referring to people while we use the word "that" when referring to objects.

While our algorithm still has room for improvement, as a proof of concept, it has been shown to perform almost as effectively as the topic models technique using the Latent Dirichlet Allocation algorithm. It has a number of advantages including simplicity, customizability, and scalability. The algorithm can be used in a few natural language processing applications including definition distinction, named entity recognition, and sentence generation.

Chapter 7

Conclusion

Having a computer program that can adequately parse and encode the semantic meaning of text would be significant progress in the field of natural language processing. For this thesis, we decided to complete a component of this over-arching project, keyword clustering. Keyword clustering can be immensely useful in definition distinction, named entity recognition, and sentence generation.

For this task, we used a simple algorithm involving finding the most common words and determine the words that occurred together in similar contexts. While the topic models technique using the Latent Dirichlet Allocation is more commonly used for this word clustering, we found that our technique is more suitable for our application because it has a similar accuracy while being a simpler, more customizable, and better performing technique.

Obviously the big question is: is this good enough? How would we use this step to help achieve our next objective? For now, we believe the results are good enough to continue moving forward on our larger project. An important next step is word sense disambiguation, where we can separate the definitions of words with multiple possible definitions. To do this, we will need traits of words to distinguish between the definitions. One method would be using context clusters. After all, words with different definitions will likely be associated with different keywords, so whenever encountering the word, we can examine its surrounding words and use the clusters associated with the word to help distinguish between the definitions. Obviously we would probably need to use other factors as well, such as word tense, but we believe word categories may be an important feature in this classification problem.

Appendix A

Code

Appendix A.0. Forming the Context Table.

Below is the code for generating the table for the number of times that each word appears in the same sentence as every other word.

```
workingSet = tierSets['tier1']
contextDict = {}
for word in workingSet:
        contextDict[word] = {}
        for word2 in workingSet:
                if word2 == word: continue
                contextDict[word][word2] = 0
for sent in tokenized_sents:
        for word in sent:
                if word in workingSet:
                        for word2 in tokenized_sent:
                                if word2 not in workingSet: continue
                                if word2 == word: continue
                                contextDict[word][word2] += 1
```

Appendix A.1. Finding the Characteristic Words.

In the code below, for each keyword, we take the top 1000 most commonly seen keywords in the same sentence as that keyword.

```
topContextWords = {}
for t1word in contextDict:
        topContextWords[t1word] = sorted(dictItems, key=itemgetter(1),
reverse=True)[:1000]
```

Appendix A.2. Weights of each Context Word.

In the code below, we determine the uniqueness weight value of each context word in determining relationship strength. First, we find the number of keywords for which a given word has appeared in the set of character words. The weight is just the inverse of that number.

```
niFreqDict = defaultdict(int)
for word in contextDict:
        for word2, freq in contextDict[word]:
                niFreqDict[word2] += 1
weights = {}
for word in niFreqDict:
        weights[word] = float(1)/niFreqDict[word]
```

Appendix A.3. Word Relationship Values.

In the code below, we sum together the weight values of the common context words of each pair of keywords to determine their relationship. We perform this operation for each pair of keywords.

```
distVectors = defaultdict(lambda: defaultdict(int))
for word1 in contextDict2:
        for word2 in contextDict2:
                if word1 == word2: continue
                if word2 in distVectors and word1 in distVectors[word2]: continue
                distVector = getDistVector(word1, word2)
                distVectors[word1][word2] = distVector
                distVectors[word2][word1] = distVector
sortedDistVectors = {}
for item in distVectors:
        sortedDistVectors[item] = sorted(distVectors[item].items(), key=itemgetter(1),
reverse=True)
```

Appendix A.4. Clustering.

In the code below, we cluster the keywords by sorting all the relationships from high to low and systematically adding the word pairs into clusters together. If two words are already in different clusters, we merge their clusters if the clusters are below average size.

```
class ClusterDict(dict):
	def __init__(self, *args, **kwargs):
		super(ClusterDict, self).__init__(*args, **kwargs)
		keys = self.keys()
		assert all([type(item) == int for item in keys])
		if len(keys) > 0:
			self.id_counter = max(self.keys()) + 1
			self.allWordsClustered = multUnion(self.values())
		else: self.id_counter = 0; self.allWordsClustered = set([])
	def addNewCluster(self, cluster):
		assert isinstance(cluster, set)
		self[self.id_counter] = cluster
		self.id_counter += 1
		for word in cluster: self.allWordsClustered.add(word)
	def addNewClusters(self, clusters):
		for cluster in clusters:
			self.addNewCluster(cluster)
	def addClusterFromList(self, itemList):
		newCluster = set(itemList)
		self.addNewCluster(newCluster)
	def deleteCluster(self, cluster_id):
		del self[cluster_id]
	def combineClusters(self, cluster1_id, cluster2_id):
		if cluster1_id == cluster2_id: raise
		self.addNewCluster(self[cluster1_id].union(self[cluster2_id]))
		self.deleteCluster(cluster1_id)
		self.deleteCluster(cluster2_id)
	def addToCluster(self, cluster_id, new_item):
		self[cluster_id].add(new_item)
		self.allWordsClustered.add(new_item)
	def getCluster(self, cluster_id):
		return self[cluster_id]
	def findClusterWithWord(self, word):
		for cluster_id in self:
			if word in self[cluster_id]: return cluster_id
		return -1
	def getNumClusters(self):
		return len(self.items())
	def getAveClusterLen(self):
```

```
                numClusters = self.getNumClusters()
                if numClusters == 0: return 0
                sumLen = sum([len(cluster) for cluster_id, cluster in self.items()])
                return float(sumLen)/numClusters
        def gtAveLen(self, cluster_id):
                aveLen = self.getAveClusterLen()
                if len(self[cluster_id]) > aveLen: return True
                else: return False

distanceSet = {}
def getSortedTuple(word1, word2):
        return tuple(sorted([word1, word2]))
for word1 in sortedDistVectors:
        for word2, dist in sortedDistVectors[word1]:
                distanceSet[getSortedTuple(word1, word2)] = dist
sortedDists = sorted(distanceSet.items(), key=itemgetter(1), reverse=True)

clusterDict = ClusterDict()
for wordpair, relstr in sortedDists[:150000]:
        w0cluster = clusterDict.findClusterWithWord(wordpair[0])
        w1cluster = clusterDict.findClusterWithWord(wordpair[1])
        if w0cluster == -1 and w1cluster == -1:
                clusterDict.addNewCluster(set([wordpair[0], wordpair[1]]))
        elif w0cluster == -1: clusterDict.addToCluster(w1cluster, wordpair[0])
        elif w1cluster == -1: clusterDict.addToCluster(w0cluster, wordpair[1])
        elif w0cluster == w1cluster: pass
        elif clusterDict.gtAveLen(w0cluster) or clusterDict.gtAveLen(w1cluster): pass
        else: clusterDict.combineClusters(w0cluster, w1cluster)
```

REFERENCES

[Blei et al. 2003a] Blei, D. M., Jordan, M. I., Griffiths, T. L., Tenenbaum, J. B. 2003. Hierarchical Topical Models and the Nested Chinese Restaurant Process. Neural Information Processing Systems.

[Blei et al. 2006] Blei, D. M. , Lafferty, J. D. 2006. Correlated Topic Models. Advances in Neural Information Processing Systems.

[Blei et al. 2003b] Blei, D. M., Ng, A. Y., Jordan, M. I. 2003. Latent Dirichlet Allocation. Journal of Machine Learning Research.

[Deerwester et al. 1990] Deerwester, S. C., Dumais, S. T., Landauer, T.K., Furnas, G. W., Harshman, R. A. 1990. Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science.

[Duda et al. 2000] Duda, R. O., Hart, P. E., Stork, D. G. Pattern Classification (2nd Edition). Wiley-Interscience. October 2000.

[Erosheva et al. 2004] Erosheva, E., Stephen, F., Lafferty, J. 2004. Mixed-Membership Models of Scientific Publications. Proceedings of the National Academy of Sciences.

[Farrahi et al. 2011] Farrahi, K., Gatica-Perez D. 2011. Discovering Routines from Large-Scale Human Locations using Probabilistic Topic Models. ACM Transactions on Intelligent Systems and Technology.

[Griffiths et al. 2004] Griffiths, T. L., Steyvers, M., 2004. Finding Scientific Topics. Proceedings of the Natural Academy of Sciences.

[Ha-Thuc et al. 2008] Ha-Thuc, V. Srinivasan, P. 2008. Topic Models and a Revisit of Text-Related Applications. Proceedings of the 2nd PhD Workshop on Information and Knowledge Management.

[Hoffman et al. 1999] Hoffman, T. 1999. Probabilistic Latent Semantic Indexing. Proceedings of the 22nd annual ACM SIGIR Conference on Research and Development in Information Retrieval.

[Huynh et al. 2008] Huynh, T., Fritz, M., Schiele, B., 2008. Discover of activity patterns using topic models. Proceedings of the Workshop on Ubiquitous Computing.

[Kataria et al. 2011] Kataria, S. S., Kumar, K. S., Rostogi R., Sen, P., Sengamedu, S. H., 2011. Entity Disambiguation with Hierarchical Topic Models. Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[Lagus et al. 1999] Lagus, K., Honkela, T., Kaski, s., Kohonen, T. 1999. Websom for Textual Data Mining. Journal of Artificial Intelligence Review, Volume 13 Issue 5-6.

[Mei et al. 2007] Mei, Q., Shen, X., Zhai, C. 2007. Automatic Labeling of Multinomial Topic Models. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[Monay et al. 2007] Monay, F., Gatica-perez, D. 2007. Modeling Semantic Aspects for Cross-Media Image Retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[Newman et al. 2007] Newman, D., Hagedorn, K., Chemudugunta, C., Smyth, P. 2007. Subject Metadata Enrichment using Statistical Topic Models. Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries.

[Niebles et al. 2006] Niebles, J. C., Wang, H., Fei-fei, L., 2006. Unsupervised Learning of Human Action Categories using Spatial-Temporal Words. British Machine Vision Conference.

[Pritchard et al. 2000] Pritchard, J. K., Stephens, M., Donnelly, P., 2000. Inference of Population Structure Using Multilocus Genotype Data. Genetics.

[Quelhas et al. 2007] Quelhas, P., Monay, F., Odobez, J. M., Gatica-perez, D. 2007. A Thousand Words in a Scene. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[Rosen-Zvi et al. 2004] Rosen-Zvi, M., Griffiths, T., Steyvers, M., Smyth, P., 2004. The Author-Topic Model for Authors and Documents. Proceedings of the 20th conference on Uncertainty in Artificial Intelligence.

[Salton et al. 1983] Salton, G., McGill, M. J. 1983. Introduction to Modern Information Retrieval. New York: McGraw-Hill.

[Sontag et al. 2011] Sontag, D., Roy, D. M. 2011. Complexity of Inference in Latent Dirichlet Allocation. 25 Annual Conference on Neural Information Processing Systems.

[Yang et al. 1999] Yang, Y., Liu, X. 1999. A Re-Examination of Text Categorization Methods. Proceedings of the 22nd annual ACM SIGIR conference on Research and Development in Information Retrieval.

ACADEMIC VITA

Kelvin Shi

kqs5175@outlook.com

Education

B.S., Electrical Engineering, 2013, The Pennsylvania State University, University Park, PA

Honors and Awards

Diefenderfer Scholarship, Penn State College of Engineering, 2012-2013

Schreyer Scholarship, Penn State Schreyer Honors College, 2009-2013

National Merit Scholarship, National Merit Scholarship Corporation, 2009-2013

Dean's List, 7 semesters, 2009-2013

Association Memberships/Activities

Presidential Leadership Academy

Strategic and Global Security Scholars Program

Professional Experience

Student Space Programs Labs, 2011-2012

Intel-Cornell Embedded Design Competition, 2011-2012

Electronics lab Research Assistant, 2010

Research Interests

I have research interests in several fields in computer science, including natural language processing, machine learning, computer vision, systems administration, computer forensics, reverse engineering, and network security. Specifically, I am currently working on a project to design a computer program that could programmatically generate essays.

Publications

Gangfeng Ye, Kelvin Shi, Robert Burke, Joan M. Redwing, and Suzanne E. Mohney "Ti/Al Ohmic Contacts to N-Type GaN Nanowire," Journal of Nanomaterials. Vol. 2011, Article ID 876287, 2011. doi:10.1155/2011/876287.