Algorithms for Information Retrieval and Natural Language Processing tasks

Pradeep Muthukrishnan Dragomir R. Radev
School of Information and
Department of EECS
University of Michigan
Winter, 2008
{mpradeep,radev}@umich.edu

September 24, 2008

Contents

Chapter 1

Introduction

In this report, I have tried to summarize the common algorithms and approaches used for various tasks in Information Retrieval and Natural Language processing like, Text Summarization, Machine Translation, Sentiment Detection, Dependence Parsing, etc.

This report is submitted towards satisfying the requirement of the Directed Study, EECS 599 course, Computer Science division, University of Michigan, Ann Arbor under the supervision of Professor Dragomir R. Radev.

Chapter 2

Text summarization

2.1 Introduction

Text summarization is an active field of research in Information Retrieval and Machine learning. The major design goal of summarization is to convey the same amount of information available in the original source text in a much smaller document. This chapter talks about the types of summaries, different approaches and problems with respect to summarization.

2.2 Types of summaries

The summaries are broadly categorized based on purpose, input and output

1. Abstractive vs Extractive Summaries:
   Abstractive summaries or abstracts, involve identifying important parts of the source document and reformulating it in novel terms. The phrase "important parts" here refer to the parts which contain significant information content related to the source document. Extractive summaries or extracts, involve selecting sentences, words, phrases and combining them together in a cohesive and coherent manner. Abstracts are a much harder problem because the second phase in the problem of generating grammatically correct memories using a natural language generator is hard.

2. Single or Multiple document summaries:
   This categorization is based on the number of input documents to the summarizer. This parameter is often called the span parameter.

3. Informative or Indicative Summaries:
   An indicative summary just lets the reader know about the contents of the article while an informative summary can be used as a substitute for the original source document. A third category of summary is Critical summaries but no actual summarization system creates a critical summary.

2.3 Analysis of Different Approaches for Summarization

Many different approaches to summarization have been tried by various researchers over the years. One of the graph-based approaches by Radev et al [2], uses a graph-based stochastic matrix for

computing the important sentences. It was a multi-document extractive summarization system which computes a set of sentences identified as central to the topic of the documents. In this approach, an undirected graph is constructed where all sentences are the vertices and an edge corresponds to the similarity between the two vertices (sentences) being above a specified threshold. The similarity measure used is a modified version of cosine similarity as given below.

$$idf - modified - cosine(x, y) = \frac{\sum_{w \in x,y} tf_{w,x} tf_{w,y} (idf_w)^2}{\sqrt{\sum_{x_i \in x} (tf_{x_i,x} idf_{x_i})^2} \sqrt{\sum_{y_i \in y} (tf_{y_i,y} idf_{y_i})^2}} \tag{2.1}$$

where $tf_{w,x}$ is the term frequency of word w in document x, and idf_w is the inverse document frequency of word w. To compute the central or salient sentences, there are a variety of methods that could be used. For eg, the degree of a vertex can be used as a measure of centrality, where a higher degree implies more central. A more advanced method that the authors suggest is using LexRank, a modified version of PageRank. Using this method offsets the effect of outliers (noise) , that is, a set of documents unrelated to the actual topic of the input grouped together by mistake. These unrelated documents could give rise to a set of similar sentences being chosen which are totally unrelated to the actual topic, if the degree-based centrality measure is used. The eigenvector computation for computing the stationary distribution vector is done using the power method. One optimization that, I believe, could be used for computing the stationary distribution vector is to use the repeated squaring method as shown below.

input: A stochastic, irreducible and aperiodic matrix M
input : matrix size N, error tolerance ϵ
output: eigenvector p

1. $p_0 = \frac{1}{N}1$
2. repeat
3. ${M_t}^T = ({M^T}_{t-1})^2$
4. $\delta = {M_t}^T - {M^T}_{t-1}$
5. until $\delta < \epsilon$
6. return ${M_t}^T p_0$

An approach to generate more accurate summaries is to use templates for specific domains with slots for each feature related to the topic of the source documents. With this approach the more ontological rules that are added, the better is the quality of the summary. The problem with this approach is that it is highly limited to specific domains.

In the area of specific domain summarizers, Afantenos et al [3], present many different summarizers in the medical domain and the specific features of the domain which can be exploited to produce better summaries.

2.4 Evaluation

The evaluation has always been a very hard problem mainly because human summarizers, themselves tend to agree among themselves only 40% of the time, where agreeing refers to common n-grams between the summaries.
The current evaluation methods are of two types.

1. Quantifiable methods:

   This method deals with comparing the output summary with summaries written by human judges using the same source documents and computing some quantitative results. The comparison is mostly done by computing the precision and recall using metrics like the number of common 5-grams between the output summary and the human-written summary. For example, ROUGE is a recall-based fixed-length summaries which is based on 1-gram, 2-gram, 3-gram and 4-gram co-occurrence. It reports separate scores for each $n-gram$ co-occurrence but it has been shown by Lin and Hovy that unigram-based ROUGE score agrees the most with human summaries.

2. Human judged:

   This method involves human judges reading the output summary and seeing if they can relate it to the original document. For example, Pyramid is a manual method for summarization evaluation. It deals with the fundamental problem in evaluation of summarization systems which is different humans choose different sentences for the same facts. The solution to this is a gold-standard summary is created using many different human summaries and each fact's importance increases with the frequency of the fact being mentioned in the human summaries.

2.5 Conclusion

This chapter presented a broad introduction to summarization systems and a stochastic graph-based algorithm for computing an extractive summary for a multiple documents. Also a small optimization with respect to the computation of the stationary distribution vector. The evaluation methods that are generally used in tandem with summarization systems has been discussed in brief.

Chapter 3

Semi-supervised learning, random walks and electrical networks

3.1 Introduction

Semi-supervised learning (SSL) is one of the major learning techniques in Machine learning. It makes use of both labeled and unlabeled data for training. It assumes that amount of labeled data is very little compared to unlabeled data, which makes it a very practical alternative to supervised learning which uses only labeled data, which is very expensive. The most common idea of semi-supervised learning or any learning algorithm is to identify the underlying probability distribution of the input data and then using a probability model (like Bayes) to predict the output for unseen test cases. In the case of semi-supervised learning, the labeled data is used to estimate the probability distribution and the unlabeled data is used to fine-tune the distribution. Figure 2.1 shows how this is achieved.

The chapter is organized in the following way. Commonly used methods in SSL are mentioned in chapter 2 in detail and the possibility of using active learning along with SSL is covered in section 3 followed by a brief discussion on random networks and electric networks.

3.2 Common methods used in SSL

There are different approaches/models that have been tried in the past. Some of them are listed below in detail. Before looking at the different approaches, the question of which model to use for a particular problem is very important because if the model is incorrect, then it has been proved that unlabeled data can actually degrade the accuracy of the classifier. The answer depends on the nature of the problem. For example, If the different classes cluster very well, then EM with generative mixture models is a good option. If the feature set can be split into conditionally independent sets then co-training is a good option. If inputs with similar features fall into same same classes, then graph-based methods can be used. In essence, one needs to choose the approach which fits the problem structure well.

3.2.1 Generative Models

It is one of the oldest-models in SSL. It assumes a model $p(x, y) = p(y)p(x|y)$ where $p(x|y)$ is an identifiable mixture distribution, for example Gaussian mixture models. The mixture components can be identified if we have large amounts of unlabeled data, then with just one labeled input data

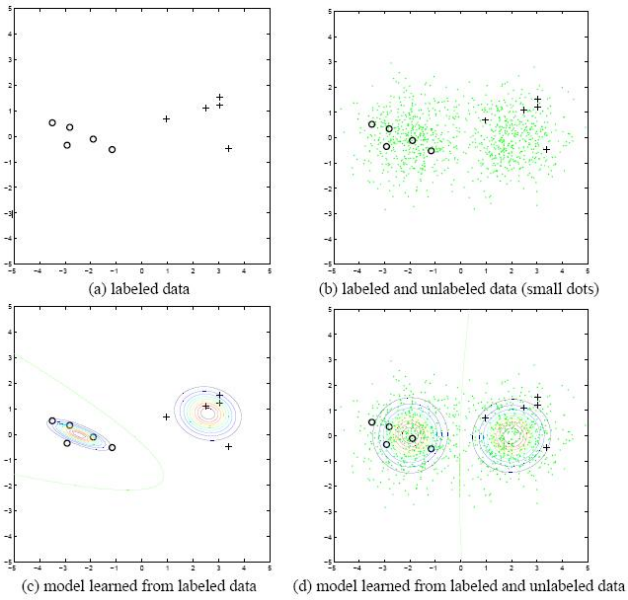


Figure 3.1: In a binary classification problem, unlabeled data can be used to help parameter estimation, Xiaojin Zhu, Semi-supervised learning literature survey. Technical chapter 1530, Computer Sciences, University of Wisconsin-Madison

for every class, we can accurately find the mixture components. Identifiability refers to the property that the distribution can be uniquely represented by a set of parameters. The reason behind the usage of an identifiable mixture is that, if identifiability is not a requirement then it is possible that we could have different distributions leading to different classifiers which output different labels for the same input.Even if the model is right, sometimes the probability distribution found using EM (Expectation maximization) could be far off from the optimal and once again, unlabeled data could degrade learning. Another approach which completely does away with probability distribution is Cluster-and-Label, which clusters the data and labels each cluster with the labeled data. The problem with this approach is that it is very hard to analyze the classification because the error almost completely depends on the clustering algorithm.

3.2.2 Self Training

In this approach, the classifier is initially trained with just the labeled data, then they are used to classify the unlabeled data, and the most reliable labelings are added to the labeled data with the predicted labels and the classifier is trained again. This happens iteratively. This procedure is also called self-teaching or bootstrapping. The problem with this method is, it is possible that a wrong labeling can manipulate itself and cause further degradation. A remedy for this is to have a mechanism integrated to unlearn some "data labelings" whenever the predicted probability of that labeling drops below a threshold.

3.2.3 Co-Training

This method could work very well if the feature set can be split into conditionally independent sets given the classification. In such a scenario, we train two different classifiers using the two different feature sets using both the labeled and unlabeled data. Then the most reliable (confident) labeling

of each classifier is added to the other classifier as labeled data and re-trained. There have also been attempts to develop two classifiers considering the whole feature set as such and using the reliable classifications of one classifier as labeled data for the other.

3.2.4 Graph-Based Methods

In all the graph-based methods, the vertices (nodes) are the labeled and unlabeled data points and the edges (may be weighted) correspond to the similarity between nodes. There is an inherent assumption with all graph-based methods of label smoothness over the graph, i.e., the labels do not change much between neighboring nodes.

All graph-based methods are very similar to each other as far as framework is concerned. There is a loss function and a regularizer. The loss function's functionality is to make sure that the predicted labeling for labeled nodes is as close as possible to the actual labeling. In other words, if this condition is not true, we have lost everything and hence loss is infinity. The regularizer is used to make sure the predicted labelings are smooth over the graph. All the graph-based methods differ only in their choice of loss functions and regularizer.

3.2.5 Mincut approach

The SSL problem can be thought of as a mincut problem in graph theory, by making all positive labels to be sources and negative labels to be sinks. Thus the SSL problem boils down to finding the minimum set of edges such that all paths from the sources to sinks are cut. In this case, the loss function can be viewed as quadratic loss with infinity weight ∞ . This constraint is required to make sure the labeled data are fixed at their labels. The regularizer is

$$\frac{1}{2}\sum_{i,j} w_{ij}|y_i - y_j| = \frac{1}{2}\sum_{i,j} w_{ij}(y_i - y_j)^2 \tag{3.1}$$

The above equality is true only in the binary labels case. Both the functions put together, the mincut problem can be thought of as minimizing the function.

$$\infty \sum_{i \in L} (y_i - y_i \mid L)^2 + \frac{1}{2}\sum_{i,j} w_{ij}(y_i - y_j)^2 \tag{3.2}$$

subject to the constraint $y_i \in \{0, 1\}, \forall i$. Classification in this approach is performed by assigning the label of the class which the input data point is closest to. There is one problem with mincut, that it gives only hard classifications, instead of marginal probabilities as other approaches do. This can be overcome by adding some random noise to the edges and creating many classifiers from the different graphs that arise from this perturbation. Now, classification can be done by taking majority vote. This approach is sometimes called as "$soft\ mincut$".

The solution for the labeling function can be viewed as local averaging, i.e, it satisfies the averaging property.

$$f(j) = \frac{1}{d_j}\sum_{i \sim j} w_{ij} f(i) \tag{3.3}$$

In other words, the solution $f(x_i)$ at an unlabeled point x_i is the weighted average of its neighbors solutions. Since the neighbors are usually unlabeled points too, so this is a self-consistent property.

3.2.6 Tree-based Bayes

In this approach, a probabilistic distribution $P(Y|T)$ is defined on discrete labelings Y over an evolutionary tree T. This is different from the graph-based methods in the sense, that the labeled and unlabeled nodes are actually leaves of the tree. And the regularizer function is abstracted by constraining nearby leaves to have similar labelings. The label is assumed to mutate over the path from the root to the leaves, and as a consequence, the tree uniquely defines the label given $P(Y|T)$.

3.3 Active Learning

Active learning is a method which complements the semi-supervised learning techniques. Active learning is used to minimize the estimated expected classification error. This is done by asking for the user annotation of data points selected by the learning algorithm. The selection algorithm should not select the data point naively based on maximum label ambiguity, or least confidence. Instead they should query points which would lead to minimization of estimated expected classification error. The general framework of active learning is captured by the following pseudocode.
input: L (set of labeled nodes) , U (set of unlabeled nodes) , weight matrix W
output: L and the classifier, F

while more labeled data required:

1. Compute labeling (classifier) function f
2. Find best query point k which minimizes the estimated expected error.
3. Query point x_k to get the label y_k
4. Add (x_k, y_k) to L, remove x_k from U.

end

Zhu et al (2003) show how active learning and semi-supervised learning could be combined using harmonic functions and gaussian distribution. The SSL method used is a graph-based method where the edge function is defined to be

$$w_{ij} = exp(-\frac{1}{\sigma^2}\sum_{d=1}^{m}(x_{id} - x_{jd})^2) \tag{3.4}$$

and the energy function is defined to be

$$E(f) = \frac{1}{2}\sum_{i,j}(w_{ij}(f(i) - f(j))^2) \tag{3.5}$$

where f is the labeling function.
The energy function is responsible for the labelings of close-by nodes to have similar labelings. Now the whole problem is to find the function f which minimize the energy function, i.e,

$$f = arg\ min_{y|L=y_L}E(y) \tag{3.6}$$

And the function which satisfies this property is known to be harmonic functions, i.e, functions which satisfy the averaging property. Because of the averaging property, the labelings of nearby

nodes are ensured to be similar. To use the active learning method efficiently we need to estimate the risk carefully. The risk function that Zhu et al have suggested is

$$R(f) = \sum_{i=1}^{n} \sum_{y_i=0,1} [sgn[f_i) \neq y_i] p^*(y_i|L) \tag{3.7}$$

Since this depends on the $p^*(y_i|L)$, which is the true label distribution at node i, given the labeled data, the above formula cannot be directly used to compute $R(f)$. Thus, as an approximation, $p^*(y_i = 1|L) \approx f_i$. Therefore, we can compute the estimated risk now as,

$$R(f) = \sum_{i=1}^{n} [Sgn(f_i) \neq 0](1 - f_i) + [Sgn[f_i) \neq 1] f_i \tag{3.8}$$

The above formula is a consequence of the fact that f_i is the probability of reaching a node 1 before reaching node 1 on a random walk. Now we can use the above formula to compute the best point to be queried which can be used to minimize the risk. That point is queried and after the labeling has been received, this point is added to the labeled data set and the whole process is repeated. Thus, by active learning one can minimize the estimated classification error by means of intelligent querying.

3.4 Random walks and electric networks

As mentioned in the previous section, let $f(i)$ be the probability of reaching a node labeled 1 before reaching a node labeled 0 on a random walk. Then it can be shown that $f(i)$ satisfies the averaging property of harmonic functions using basic probability laws. Thus $f(i)$ is a harmonic function. One of the interesting properties of harmonic functions is that if there are two harmonic functions $f(x)$ and $g(x)$ such that $f(x) = g(x) \forall x \in B$, where B is the set of boundary values, then $f(x) = g(x) \forall x$. This can be proved with the help of the maximum principle and the uniqueness principle. Both the principles are defined below.

Maximum Principle: A harmonic function f (x) defined on S takes on its maximum value M and its minimum value m on the boundary.

Uniqueness Principle: If f (x) and g (x) are harmonic functions on S such that f (x) = g (x) on B, then f (x) = g (x) for all x.

Also there exists an analogy between electric networks and random walks. The analogy is as follows. Let the entire graph be converted into a electric network, where every edge is replaced by a resistor and all nodes labeled 1 are connected to a positive voltage and all nodes labeled 0 are grounded. In this electric network, let the voltage at node x be represented by $V(x)$. Therefore, $V(x) = 0, 1$ for all labeled nodes and by basic laws of electricity, it can be shown that $V(x)$ satisfies the averaging property. Thus its a harmonic function which takes the same boundary values as our probability distribution $f(x)$. Thus by maximum principle and uniqueness principle, $f(x) = V(x)$.

3.5 Conclusion

This chapter presented an introduction to semi-supervised learning and the different approaches that are used in Semi-supervised learning. Active learning has also been discussed in conjunction with semi-supervised learning and also an example of combining active learning and semi-supervised learning has been described clearly.

Chapter 4

Semi-Supervised Learning and Active Learning

4.1 Introduction

In the last chapter, some of the most common methods used in semi-supervised learning were covered. In this chapter, we continue to discuss other methods in semi-supervised learning with respect to specific applications.

4.2 Active Learning with feature feedback

4.3 Using decision Lists

In this section, decision lists are used in the context of word sense disambiguation and lexical ambiguity resolution. There are a lot of similarities in both approaches, such as, both the approaches are completely unsupervised. Both the problem's solutions rely largely on the usage of collocation as the most important feature splitting the input into disjoint classes. Decision lists are, in simple terms, a list of feature queries that are posed about the input data points, ranked in some well-defined order to obtain the classification.

4.3.1 Word sense Disambiguation

This algorithm makes use of two core properties of the human language.

1. One sense per collocation: Nearby words provide strong and consistent clues to the sense of a target word, conditional on relative distance, order and syntactic relationship.

2. One sense per discourse: The sense of a target word is highly consistent within any given document.

Considering the large amount of redundancy in the human language, the sense of the word is over-determined by the above two properties. Yarowsky, Gale and Church (1992) , have provided lots of empirical evidence to support both the above two properties. The feature vector for this problem is defined to be the set of collocating words, and they are ranked by the probability log-likelihood

ratio of the probability distribution for all collocations of a word. The log-likelihood ratio is defined below.

$$Log(\frac{Pr(Sense_A \mid Collocation_i)}{Pr(Sense_B|Collocation_i)}) \tag{4.1}$$

1. In the corpus, all examples of the given word, storing their contexts as lines in an untagged training set.

2. For each sense of the word, a few examples are tagged

4.3.2 Lexical ambiguity resolution

This algorithm is largely similar to the previous one, differing mainly in the feature vectors, which also includes the syntactic relationship between the collocating words and the word of interest. The specific problem that has been chosen is accent restoration, though the algorithm should work on any of the related problems such as, word-sense disambiguation, capitalization restoration, word choice in machine translation and homograph and homophone disambiguation. All the above problems are important because the whole semantics of the word under consideration changes dramatically with a small change in accent, context.

1. Identify Ambiguities in Accent Pattern
   Most words in Spanish/French exhibit only one accent pattern. The corpus can be analyzed to find words which have more than one accent pattern, and steps 2-5 are applied to them.

2. Collect Training Contexts
   All collocating words within a word window of $\pm k$.

3. Measure Collocational Distributions
   The core strength of this algorithm lies on the fact that there is uneven distribution of collocations with respect to the ambiguous word being classified. Certain collocations tend to indicate one accent pattern and others indicating other patterns. The types of collocations considered are

   (a) Word immediately to the right
   (b) Word immediately to the left
   (c) Word found in $\pm k$ word window.
   (d) Pair of words at offsets -2 and -1
   (e) Pair of words at offsets -1 and +1
   (f) Pair of words at offsets +1 and +2

4. Sort by Log-Likelihood ratios into decision Lists

   In this step we compute the log-likelihood ratios according to (9) and sort them in decreasing order. The collocations most strongly indicative of a particular pattern will have the largest log-likelihood. The optimal value of k depends on the type of ambiguity, while semantic or topic-based ambiguities need a large window of $k \approx 20 - 50$, while more local syntactic ambiguities need only a small window $k \approx 3 - 4$. For Spanish experiments alone, a a morphological analyzer and a basic lexicon with possible parts of speech analyzer was used along with the collocating words to generate a rich set of evidence.

4.4 Semi-supervised Learning using Co-Training

This paper considers the problem of using a large unlabeled sample to boost the performance of a learner when only a small amount of labeled data is available. Also, it is assumed that the instance can be described using two views, where one view is complete by itself for learning, provided there are enough labeled data. The goal is to use both views together to reduce the requirement of expensive labeled data, by using more inexpensive unlabeled data.

Let X = (X1, X2) be an instance space, where X1 and X2 correspond to two different views of an example. Let D be the distribution over X and let C= (C1, C2) be the combined concept class defined over X, where C1 is defined over X1 and C2 is defined over X2. One significant assumption is that D assigns probability 0 to any example x = (x1, x2) where $f1(x1) \neq f2(x2)$, where $f1 \in C1$ and $f2 \in C2$ are the target functions. This is called compatibility assumption.

4.4.1 Bipartite Graph representation

Consider a bipartite graph $G_D(X_1, X_2)$ where there is an edge (x_1, x_2) if $Pr_D(x_1, x_2) \neq 0$. The compatibility assumption implies that any connected component in G_D will have the same classification. Given a set of unlabeled examples S, one can similarly define a bipartite graph G_S having one edge (x_1, x_2) for every $(x_1, x_2) \in S$.

Consider that S contains all examples in G_D. When a new example comes in, the learner will be confident about its label, if it has previously seen a labeled example in the same connected component of G_D. Thus if the connected components in G_D are $c_1, c_2, \ldots$ and their probability masses are $p_1, p_2, \ldots$ respectively, then the probability that given m labeled examples, the label of a new example cannot be deduced is just

$$\sum_{c_j \in G_D} P_j(1 - P_j)^m \tag{4.2}$$

One can use the two views to achieve a tradeoff between the number of labeled and unlabeled examples needed. As the number of unlabeled examples increase, the number of connected components in G_S decrease until it becomes equal to the number of connected components in G_D. Since one needs only one labeled example per connected component, the number of labeled examples required to learn the target function decreases.

Let H be a connected component in G_D. Let α_H be the value of the minimum cut of H. In other words, α_H is the probability that a random example crosses this cut. If sample S were to contain all of H as one component, it must include at least one edge in that minimum cut. The expected number of unlabeled examples needed to include this edge is $\frac{1}{\alpha_H}$. Karger shows that $O(\frac{logN}{\alpha_H})$ examples are sufficient to ensure that a spanning tree is found with high probability. So, if $\alpha = min_H \alpha_H$, then $O(\frac{logN}{\alpha})$ unlabeled examples are sufficient to ensure that G_S has same number of components as G_D.

4.4.2 Learning in large input spaces

The theorem in the paper shows that given a conditional independence assumption on the distribution D, if the target class is learnable from random classification noise in the standard PAC model, then any initial weak predictor can be boosted to arbitrarily high accuracy using unlabeled examples only by co-training. A weak predictor h of a function f is defined to be a function such that

1. $Pr_D[h(x) = 1] >= \epsilon$
2. $Pr_D[f(x) = 1|h(x) = 1] >= Pr_D[f(x) = 1] + \epsilon$

This way, using a large amount of unlabeled data, we can prune away the incompatible target concepts and thereby reduce the number of labeled concepts needed to learn the concept classes.

4.5 Conclusion

This chapter briefly went over the applications of semi-supervised learning methods to the various applications in text processing.

Chapter 5

Network Analysis and Random graph Models

5.1 Introduction

This chapter is about the different network properties of real-world graphs and techniques to efficiently find them and about random graphs. The chapter is organized in the following way, section 2 introduces the different types of networks followed by Section 3, which presents the different properties of the networks and physical understanding of their implications on the network. In section 4, we go through an algorithm for finding community structure using eigenvectors of matrices. Finally in section 5, we go through the different random graph models and the distributions of real-world networks.

5.2 Types of Networks

The need for identifying the different type of networks arise from the fact that different types of networks have different properties and therefore, to study them we need to model the networks based on the type of the network. Most real-world networks can be broadly categorized into one of the following categories.

5.2.1 Social Networks

This network refers to a set of people connected together on the grounds of some underlying pattern and the interactions between them. For example, the network of friendships between people, business relationships between companies are all examples of social networks. There have been lots of experiments carried out in the past about such networks, but the main problem in such experiments is the way in which the data has been collected and the network size. For example, in the "small world" experiment by Milgram, which was designed to find the path lengths in an acquaintance network by asking participants to try to pass letters to an assigned target individual through their acquaintances, though the data was collected directly from the participants, it is subject to bias from the participants and is also very labor intensive. To get rid of such errors, the trend in this field of research has been shifting towards analyzing collaboration or affiliation networks. An example of this is the graph obtained through the Internet Movie Database, where each actor is a vertex of the graph and there is an edge between two vertices if the two actors have

acted together in a movie. Also another such graph is the communication graph, where there is an edge between two vertices if there was any communication between them.

5.2.2 Information Networks

There are three major information networks which are well-studied. One of them is the citation graph, where there is a directed edge from vertex A to vertex B, if B is cited in A, where the vertices are academic papers. The second important information network is the World Wide Web, where the vertices are pages on the WWW, and there is an edge from vertex A to vertex B if there is a hyperlink from A to B. The fundamental difference between them is that citation networks, by nature is acyclic, while the WWW is cyclic. The third type is the Preference Network, a bipartite network, where the vertices are of two types, the first being an individual and the second is an entity which they prefer, like books or movies. This network can be used for designing recommendation systems by analyzing the likes and dislikes of other people who look similar entities.

5.2.3 Technological networks

This class of networks are the man-made networks used for distribution of some commodity or resource. Common examples include the power grid, the mail routes and the Internet, the physical interconnection of systems. The problem with the internet is that it is hard to find the exact network since most systems are owned by companies and hence to get an approximate representation, traceroute programs are used, which give the path the packet traced between two points.

5.2.4 Biological Networks

The fundamental biological network is the metabolic pathways network, where vertices are metabolic substrates and products with directed edges joining them if a known metabolic reaction exists that acts on a given substrate and produces a given product. Other networks of this type include the gene regulatory network, neural network and the food web.

5.3 Network Properties

The study of network properties is important because they give insight into the possible formation mechanism and ways to exploit any structure about the graph. A few statistical properties which are generally studied are listed below.

5.3.1 Small-World Effect

As we have explained in the previous section, the experiment by Milgram resulted in the finding that it took only six steps to reach from one person to another through acquaintances. This is popularly known as the small-world effect. This measure, also known as, mean shortest (or geodesic) path length is computed using the following formula.

$$l = \frac{1}{\frac{1}{2}n(n+1)} \sum_{i \geq j} d_{ij} \tag{5.1}$$

where d_{ij} is the shortest path between i and j. This quantity can be measured by a breadth-first search which takes $\Theta(nm)$ time for a graph with n vertices and m edges. The problem with the

above definition is if the graph is disconnected the some d_{ij} values are infinity which would make l to be infinity. To get around this problem, the following formula is instead used for computation of mean shortest path length.

$$l^{-1} = \frac{1}{\frac{1}{2}n(n+1)} \sum_{i \geq j} {d_{ij}}^{-1} \tag{5.2}$$

One obvious implication is that, for graphs which exhibit the small-world effect, it would take very less time for information to spread across the network. This result is not particularly surprising when it is viewed mathematically, consider the number of vertices at a distance r from a distinguished vertex to grow exponentially, then the value of l is upper-bounded by $log\ n$. This has been proved for many different networks.

5.3.2 Transitivity or Clustering

The metric, called clustering coefficient, is the probability of vertex A being connected to vertex C, given that there exists a vertex B which is connected to both A and C. It can be computed using the following formula.

$$C = \frac{3\ \times\ number\ of\ triangles\ in\ the\ network}{number\ of\ connected\ triples\ of\ vertices} \tag{5.3}$$

An alternative definition of the clustering coefficient, given by Watts and Strogatz, is given below to compute a local clustering value at every vertex.

$$c_i = \frac{number\ of\ triangles\ connected\ to\ vertex\ i}{number\ of\ triples\ centered\ on\ vertex\ i} \tag{5.4}$$

For vertices with degree 0 or 1, $C_i = 0$, then the clustering coefficient for the whole network is average of clustering coefficient of all vertices.

5.3.3 Degree distribution

One way of representing the distribution of degree is computing the fraction of vertices, p_k, in the network that have degree k, which can be obtained by drawing the histogram of degree of vertices. For real world networks, the bin size used in the histogram should increase exponentially to get around the problem of noise in the tail, so that there are more samples in the bins in the right. Another way to get around this problem, is to use the cumulative distribution function as shown below.

$$P_k = \sum_{k'=k}^{\inf} P_{k'} \tag{5.5}$$

which is the probability that the degree is greater than or equal to k. This method is preferred over the other because the previous method loses information about the differences of points in the same bin.

5.3.4 Network Resilience

Network resilience is defined as the network's resistance to become disconnected despite removing vertices. This resistance is measured in terms of the mean shortest path distance between two vertices. This measure has been shown to vary depending on the degree of the vertices and the

order in which vertices are removed. For example, in the Internet, random removal of vertices affects the mean shortest path distance very slightly whereas when vertices of high degree are attacked, the same metric increases sharply. And this property holds true for most real world graphs.

5.3.5 Mixing Patterns

This property, referred to as assortative mixing, is about graphs where there are different "types" of vertices and we would like to define a metric to compute how much of mixing of different types of vertices exists in graphs. A matrix E is defined as e_{ij} = the number of edges between vertices of type i and vertices of type j. This matrix is normalized by dividing each entry by the sum of all entries. The conditional probability $P(j|i)$ that the neighbor of a vertex of type i is of type j can be computed to be $\frac{e_{ij}}{\sum_j e_{ij}}$. Gupta et al, have suggested this assortative mixing coefficient can be computed as,

$$Q = \frac{\sum_i P(i|i) - 1}{N - 1} \tag{5.6}$$

Some preferable properties of this formula is that it is 1 for a perfectly assortative network and 0 for completely non-assortative. There are two shortcomings of this formula, one is that since E could be asymmetric it depends on what we have along the horizontal axis. The other shortcoming is it measures all vertices with equal weights which is not very desirable because there could be too many vertices of particular type, thus, it should be given more weight. An alternative formula for assortative mixing coefficient which overcomes this shortcomings is,

$$r = \frac{Tr(e) - ||e||^2}{1 - ||e||^2} \tag{5.7}$$

5.3.6 Degree Correlations

This property is to quantify the relevance of degree of a vertex to assortative mixing, in the sense, do high degree vertices connect more often with high degree vertices? In essence, we try to answer the question of, whether vertices connect with other vertices with some preference.

5.3.7 Community structure

Most social networks have some sort of community structure within them, where community is defined to be a set of vertices with a higher than average number of edges between them and lesser edges outside this group of vertices. It is useful to discover this information, because the properties of the network at the community level may be very different from the properties at the whole network level. This information may be exploited to discover more properties of the communities.

5.4 Finding Community Structure

A new approach to finding community structures using spectral partitioning has been proposed by Newman et al, though the algorithm takes the general approach to finding community structures which is graph partitioning, the novelty lies in the computation of the partition. In graph partitioning, we want to split the vertex set into two disjoint sets such that the number of edges that run between the sets is minimized. Considering the usual adjacency matrix representation of a graph,

the number of edges between the sets is computed as

$$R = \frac{1}{2} \sum_{i,j \ in \ different \ groups} A_{ij} \tag{5.8}$$

To convert this into matrix form, we define an index vector S with n elements, such that

$$s_i = \begin{cases} +1 & \text{if vertex i belongs to group 1} \\ -1 & \text{if vertex i belongs to group 2} \end{cases}$$

Then R can be computed as,

$$R = \frac{1}{4} \sum_{ij} (1 - s_i s_j) A_{ij} \tag{5.9}$$

Let the degree of vertex i be k_i then,

$$k_i = \sum_j A_{ij} \tag{5.10}$$

Then R can be written as,

$$R = \frac{1}{4} \sum_{ij} s_i s_j (k_i \delta_{ij} - a_{ij}), \tag{5.11}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. The above equation can be written in matrix form, as

$$R = \frac{1}{4} s^T L s, \tag{5.12}$$

where L is the Laplacian matrix of the graph, with

$$L_i j = \begin{cases} k_i & \text{if i = j} \\ -1 & \text{if i and j are connected} \\ 0 & \text{otherwise} \end{cases}$$

Then if v_i's are the eigenvectors of the Laplacian matrix, then s can be represented in terms of v_i's as $s = \sum_{i=1}^{n} a_i v_i$. Then R can be written as,

$$R = \sum_i a_i {v_i}^T L \sum_j a_j v_j = \sum_{ij} a_i a_j \lambda_j \delta_{ij} = \sum_i {a_i}^2 \lambda_i, \tag{5.13}$$

where λ_i are the eigenvalues of L. Now we have a formula for computing R given an index vector S, which represents the split of the network. Since 0 is an eigenvalue of L, and the corresponding normalized eigenvector is (1, 1, ...) , therefore, the choice of S = (1, 1, 1, ...) gives the minimum possible value of $R = 0$ but the physical interpretation of this choice is to put all vertices in one group, which is not very useful. The most common fix to this problem is to fix the sizes of the two groups, n_1 and n_2, which fixes the value of a_i to,

$${a_i}^2 = ({v_1}^T s)^2 = \frac{(n_1 - n_2)^2}{n} \tag{5.14}$$

Now the best option that we are left with to minimize R is to choose the S vector to be as much parallel as possible to the second eigenvector, also called the *fiedler vector* with a_1's value being fixed, i.e, to maximize,

$$|{V_2}^T| = |\sum_i {v_i}^{(2)} s_i| \leq \sum_i |{v_i}^{(2)}|, \tag{5.15}$$

where ${v_i}^{(2)}$ is the i'th element of v_2. In other words, $|v_2^T s|$ is maximized when $v_i^{(2)} si \geq 0$. This combined with the condition that there should be exactly n_1 +1's or -1's, we maximize the above equation by assigning vertices to one of the groups in order of the elements in the Fiedler vector, from most positive to most negative, until the groups have the required sizes. For groups of different sizes, we set n_1 to be the number of +1's and calculate R and do the same with n_1 -1's and choose the choice which results in a smaller R.

But this whole approach of spectral partitioning does not perform well on most of the real-world graphs. This is mainly because the whole approach forces us to fix the sizes of the groups, which may not be known beforehand in most situations. Therefore, the remedy that Newman suggests is to modify the benefit function to,

$$Q = (number\ of\ edges\ within\ communities) - (expected\ number\ of\ such\ edges) \tag{5.16}$$

This benefit function is called modularity, and clearly maximizing this gives us tightly connected communities. The first quantity is easy to compute in the above equation, whereas the second quantity is a little harder to compute. To compute the second quantity, we assume a probability distribution for the edges and based on that we compute the expected number of edges. In the paper by Newman, he assumes that the probability of an edge incident on a vertex depends only on the expected degree of that vertex and derives the expected number of edges. A similar approach as taken in the spectral partitioning method is used again to convert the formula for Q into a matrix based formula and once again the S vector is chosen to be as much parallel as possible to the first eigenvector (when eigenvalues are arranged in non-increasing order) of the modularity matrix. This approach works very well compared to the first approach. A comparison of these two approaches is shown in fig 3.1. The graph shown in the fig 3.1 is of the interactions between dolphins, and the graph was split into two subgraphs after the departure of one of they key dolphins. The figure shows the two splits obtained by the two approaches. The second approach wrongly places only three of the 62 dolphins in the wrong group.

5.5 Random Graph Models

To analyze the real-world graphs, it is very useful to model the graphs artificially. In order to be able to do that, we need to identify some properties of the real-world graphs. One of the major properties that we need to analyze is to identify the probability distribution of edges. Most of the distributions in real-world graphs have been found to be right-skewed, i.e, a lot of samples occur for the smaller data points. Another important property about such distributions is they follow a distribution called power law as described below.

$$P(x) = Cx^{(-\alpha)} \tag{5.17}$$

The constant C is called the exponent of the power law, which is not very useful in the sense it is introduced in the formula only for the sake that the probabilities should sum to 1 for all x. In the paper, Newman, gives a lot of real-world examples which have been found to follow power law. To figure out if a given data set follows power law or not, and if it does how do we find out the exponent of the power law. One of the common strategies is to plot the given data set along logarithmic horizontal and vertical axes and if it happens to be a straight line, then it is said to follow a power law. But this turns out to be a poor approach. This is because when we plot the data, we bin the given data points into bins of exponentially increasing bin sizes, the noise in the

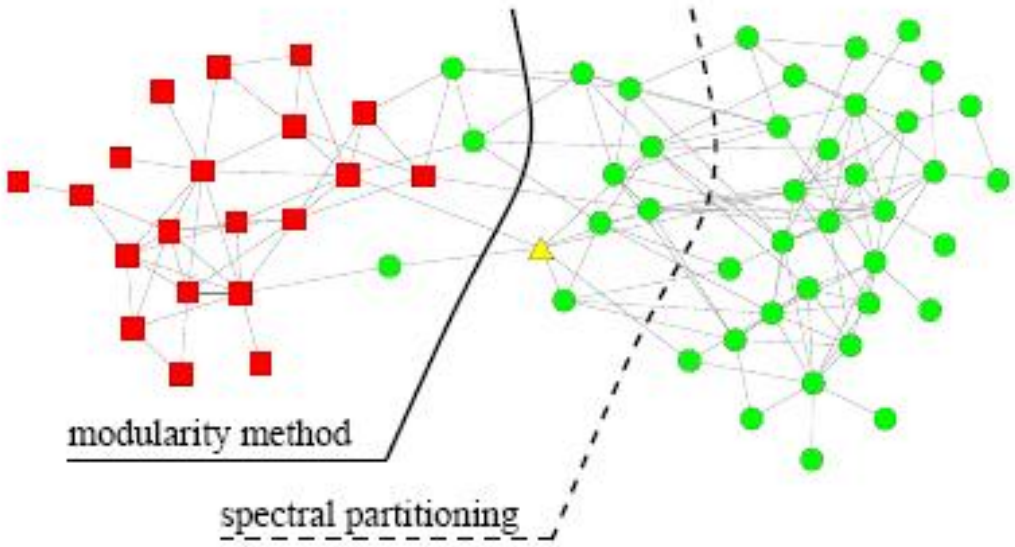


Figure 5.1: Comparison of the two approaches to graph partitioning, M.E.J Newman, Finding community structure in networks using the networks of matrices, The European Physics Journal B, 38:321 - 330, 2004

tail increases and therefore we can not identify the straight line form of the power law. Instead we can plot the cumulative distribution function,

$$P(x) = \int_{x}^{\inf} p(x')dx' \tag{5.18}$$

Thus if the original distribution follows a power law with exponent α, then the cumulative distribution function can be shown to be following a power law with exponent $\alpha - 1$. An alternative method to extract the exponent is to use the formula below,

$$\alpha = 1 + n([\sum_{i=1}^{n} ln\frac{x_i}{x_{min}}])^{-1} \tag{5.19}$$

where x_i are the measure values and x_{min} is the minimum value of the x's.

5.6 Conclusion

This chapter went over the basics of the different types of networks and the different properties of real-world graphs. Also a novel method for finding the community structures using spectral partitioning was presented and discussed.

Chapter 6

Prepositional Phrase Attachment Ambiguity

6.1 Introduction

This chapter discusses the various approaches that have been taken to solve the problem of prepositional phrase attachment ambiguity. The chapter is organized in the following way. Section 2 contains a brief description of the problem and Section 3 goes over the different approaches and section 4 summarizes the conclusions of different approaches.

6.2 Problem

The problem of prepositional phrase attachment problem (PP-Attachment) is a structural ambiguity problem which mainly arises as a subproblem of natural language parsing. The problem can be viewed as a binary classification task of deciding which site to attach a preposition to, the options being the verb or the noun. The PP-Attachment problem can be best explained with an example.

"I saw the man with the telescope"

In the above example, we need to decide if the prepositional phrase "with the telescope" attaches to the verb "saw" or to the noun "man". In this case, the correct attachment is mostly "saw" but definitely depends on context in general. The general setting of this problem is converts the input from a sentence to a 4-Tuple (V, N1, P, N2) where V is the sentence verb, N1 is the sentences object, P is the preposition, and N2 is the prepositional object. Then the problem can be restated as finding a function f (V, N1, P, N2) such that the output domain of f is $\{va, na\}$ where va means verb attachment and na means noun attachment This problem of finding the correct attachment given just the 4-tuples is generally considered to be very hard since even humans have only an accuracy rate of 88.2%.

6.3 Methods for PP-Attachment

The earliest simple methods which were used to solve this problem are

1. Right Association: The preposition tends to attach with the noun. This surprisingly has an accuracy rate of 67%

2. Minimal Attachment: a constituent tends to attach so as to involve the fewest additional syntactic nodes, which in our problem suggests verb attachment.

6.3.1 Corpus-based approach using t-scores

The first corpus-based approach to PP-attachment was done by Hindle and Rooth in 1993. Their approach dwells on the lexical co-occurrences of nouns and verbs with the preposition. They use the 13 million word sample of AP news articles as their data set. They compute the frequency of the (verb, noun, preposition) triples in this data set, and using this data set they come up with a bigram table where the first entry is a noun or a verb, suggesting which constituent should the preposition, the second entry, be attached to. To compute this bigram table they take a 3 step approach as given below.

1. Associate all verbs or nouns which are sure verb-attachment or noun-attachment with prepositions. This is identified with a set of hard-coded rules that they have given.

2. Other triples which are not attached are attached using the help of a t-score, if the t-score is above 2.1 or less than -2.1 then we assign the preposition according to the t-score. The remaining ambiguous triples are split between the noun and the verb.

3. The remaining ambiguous are attached to the noun.

Using this bigram table, they come up with a simple procedure for correct attachment based on the t-score as given below.

$$t = \frac{P(Prep|noun) - P(Prep|verb)}{\sqrt{\sigma^2(P(prep|noun)) + \sigma^2(P(Prep|verb))}} \tag{6.1}$$

where the probabilities are estimated based on the frequency counts in the corpus. If the t-score is greater than 2.1 then its assigned to the noun and if it is lesser than -2.1 its attached to the verb. Based on these new attachments the whole process is iterated over again, and further attachments are made. This method has an accuracy of 78.3%, and their confidence-based approach which is to guess only when the confidence is greater than 95% achieved a precision of 84.5%.

6.3.2 Rule-Based Approach

This approach is based on a set of rules that are learned in a particular order according to a greedy strategy to reduce training error. The first step is to run the unannotated text through an initial annotator, which could attach the preposition with the verb or noun according to simple rules. And then it follows a common supervised machine learning algorithm, which is transformation-based Error-Driven learning. This approach is intended to reduce the training error by finding transformations which reduce the error by the maximum amount. It can clearly be seen that this approach is greedy. The transformations that the procedure searches for are a simple set of transformations. The procedure then chooses the transformation which best reduces the error. The rules used are of the following form.

1. Change the attachment from X to Y if T is W.

2. Change the attachment from X to Y if T_1 is W_1, T_2 is W_2.

One specific rule mentioned in the paper is, "Change the attachment location from nl to v if p is "until". Initial accuracy after passing through the initial annotator was found to be 64%, which was to associate always with the noun. And after the transformations, the accuracy was found to be 80.8% and it is shown that the accuracy increases with the amount of training data, which is intuitive for most supervised machine learning algorithms. An optimization to this approach was suggested using word classes. Using WordNet, words were binned into different classes, and these word classes were chosen during training. For example, all words relating to time, like "in an hour", "month", "week" were replaced with the word class "Time". This way, they were able to reduce the number of transformations learnt and an improved accuracy of 81.8%

6.3.3 Back-Off model

This approach introduce by Collins et al, is a supervised machine learning algorithm which is used when there are data sparsity problems. The notation used here is

$$P'(1|v, n_1, p, n_2) = \frac{P(1, v, n_1, p, n_2)}{P(v, n_1, p, n_2)} \tag{6.2}$$

where $P'(1|v, n_1, p, n_2)$ is the maximum likelihood estimation (MLE) that the attachment should be noun attachment, and $P(v, n_1, p, n_2)$ is the frequency of the verb v, nouns n_1, n_2 and preposition p are found in one sentence, and the $P(1, v, n_1, p, n_2)$ is the frequency of noun-attachment in such cases. If this $P'(1|v, n_1, p, n_2) \geq 0.5$, then it is decided to be noun-attachment, else verb attachment. The only problem with the equation above is that it is possible that we would have never come across the 4-tuple (v, n1, p, n2) and hence the denominator would be 0 in that case and the above formula becomes useless. Infact the authors have measured this and stated that this happens for 95% of the 4-tuples found in the test set. In such cases, we back-off to 3 tuples to compute the MLE and even if that frequency count happens to be too little then we further back-off to 2 tuples and compute the maximum likelihood estimation. The only constraint that is imposed when backing off is to use make sure the 3 tuples or 2 tuples contains the preposition. The description of the algorithm has been shown in figure 5.1. The accuracy chaptered in this paper is 84.1%.

An optimization that has been suggested is to use morphological analysis which is kind of similar to Word classes used in the previous approach and this leads to a 0.4% improvement.

6.3.4 Nearest-Neighbor Method

This is another approach that has been used to deal with the problem of data sparsity. In this approach the assumption is that similar 4-tuples will have the same attachment site, where there is some notion of similarity between tuples using the word similarity between corresponding nouns, verbs and prepositions. This assumption is justified by Distributional Hypothesis in linguistics, which states that words that tend to appear in the same contexts tend to have similar meanings. Therefore, this leads us into the problem of finding similar words. There is a structure defined to all words in the corpus and this structure is parsed from the corpus. A dependency relationship is an asymmetric binary relationship between a word called head, and another word called modifier. Then the whole sentence forms a dependency tree and the root of the sentence does not modify any sentence and is called the head of the sentence. For finding the nearest-neighbor, we should have some kind of feature set defined. The feature set for a word w, is defined to be (r, w') where r represents the dependency relationship between w and w'. Since most dependency relationships involve words that are closely situated to one another, such relationships can be approximated by the co-occurrence relationship within a small window. In this case the window is restricted to be

The algorithm is then as follows:

1. **If**[3] $f(v, n1, p, n2) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, n1, p, n2)}{f(v, n1, p, n2)}$$

2. **Else if** $f(v, n1, p) + f(v, p, n2) + f(n1, p, n2) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, n1, p) + f(1, v, p, n2) + f(1, n1, p, n2)}{f(v, n1, p) + f(v, p, n2) + f(n1, p, n2)}$$

3. **Else if** $f(v, p) + f(n1, p) + f(p, n2) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, p) + f(1, n1, p) + f(1, p, n2)}{f(v, p) + f(n1, p) + f(p, n2)}$$

4. **Else if** $f(p) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, p)}{f(p)}$$

5. **Else** $\hat{p}(1|v, n1, p, n2) = 1.0$ (default is noun attachment).

The decision is then:

If $\hat{p}(1|v, n1, p, n2) >= 0.5$ choose noun attachment.

Otherwise choose verb attachment

Figure 6.1: Description of the algorithm for PP-Attachment using a back-off model, Collins et al, Prepositional Phrase Attachment through a Backed-Off Model, Third Workshop on large corpora

of size 1. Then using this as a feature vector for a word, similarity can be computed according to different similarity metrics defined previously such as Cosine Measure, Cosine of Pointwise Mutual information and so on. Let the two feature vectors be $(u_1, u_2, .., u_n)$ and $(v_1, v_2, .., v_n)$, then the different similarity metric formulae are given below.

$$sim_{cos} = \frac{\sum_{i=1}^{n} u_i \times v_i}{\sum_{i=1}^{n} {u_i}^2 \sum_{i=1}^{n} {v_i}^2} \tag{6.3}$$

For cosine of Pointwise Mutual Information, the pointwise mutual information between a feature f_i and a word u measures the strength association between them. It is defined as follows:

$$pmi(f_i, u) = log(\frac{P(f_i, u)}{P(f_i) \times P(u)}) \tag{6.4}$$

where $P(f_i, u)$, is the probability of f_i co-occurring with u. $P(f_i)$ is the probability of f_i co-occurring with any word, and P (u) is the probability of any feature co-occurring with u. The Cosine of Pointwise Mutual Information is defined as:

$$sim_{cosPMI}(u, v) = \frac{\sum_{i=1}^{n} pmi(f_i, u) \times pmi(f_i, v)}{\sqrt{\sum_{i=1}^{n} pmi(f_i, u)^2} \times \sqrt{\sum_{i=1}^{n} pmi(f_i, v)^2}} \tag{6.5}$$

Now coming to the actual algorithm used to find the attachment for a preposition, the algorithm is executed in steps and each step is taken only if the previous step fails to result in a decision. The decision is taken at each step by computing a weighted majority vote form the nearest k neighbors of the input node, where the weights are the similarity between the different neighbors and the input node. The algorithm is given below.

1. The nearest neighbors are the nodes which are identical to the input node and hence similarity is 1.

2. The nearest k neighbors are found where the neighbors also contain the same preposition, then similarity between two 4-tuples t1, t2 is computed as follows.

   $$sim(t_1, t_2) = ab + bc + ca \tag{6.6}$$

   where a, b, and c are the distributional word similarities between the verb, the nouns N_1 and N_2.

3. This step is the same as the previous function, with the only difference being the similarity function, which is,

   $$sim(t_1, t_2) = a + b + c \tag{6.7}$$

4. The set of nearest neighbors is all the training data examples with the same preposition and the similarity between the neighbors and the input node is constant and if the decision is still tied between N and V, then a noun-attachment is chosen.

This algorithm chapters different accuracy rates based on the similarity function used, and it turns out that sim_{cosPMI} is the best similarity function giving an accuracy rate of 86.5%.

6.3.5 Random Walks

The approach taken by Toutanova et al, makes use of Markov Chains and Random walks. The whole problem of computing $P(Att|V, N_1, P, N_2)$ is solved by using the joint probability distributions of $P(V, N_1, P, N_2, Att)$. For doing this, they make a few assumptions so that they could build a probability model. The assumptions are given below.

1. Given a verbal attachment, the second noun is independent of the first noun
2. Given a noun attachment, the second noun is independent of the verb.

This helps us derive formulas for computing the joint distributions as given below.

$$P(V, N_1, P, N_2, va) = P(P, va)P(V|P, va)P(N_1|P, va, V)P(N_2|P, va, V)$$
$$P(V, N_1, P, N_2, na) = P(P, na)P(V|P, na)P(N_1|P, na, V)P(N_2|P, n_1, na)$$

All factors except P (P, Att) are estimated using random walks. As an example, they show how to compute $P(N_2|P, V, va)$, they do so by constructing a Markov chain M, whose transition probabilities depend on p, v, and the fact that Att = va so that its stationary distribution π is a good approximation of $P(N_2|P, V, va)$. The set of vertices in the Markov chain are defined to be the set $W \times \{0, 1\}$, where W is the set of all words, and the bit denotes whether it is reached as the head or as a dependent. For $P(N_2|P, V, va)$, V is a head, and N_2 is a dependent. For the sake of brevity, we represent $(w, 1)$ as d_w. The initial distribution of the Markov chain gives a probability of 1 to h_V because thats where we start at. Consider an example of the sentence as "hang painting with the nail", then we are trying to estimate the probability $P(N_2 = nail|V = hang, P = with, Att = va)$, if the event of seeing V = hang, P = with, Att = va and N_2 = nail already occurred in the training set then we could use the empirical distribution itself as a good approximation. If the word "nail" occurs a lot in the context of P="with", V = "hang" and att = va, then $P(N_2 = nail|V = hang, P = with, Att = va)$ should receive high probability. And once again we use some word classes where the classes are not limited to similarity in meanings and the contexts in which they appear, but also stemming. If we did not see many occurrences of the word nail in the context we described earlier but indeed saw many occurrences of the word "nails" then we would still like $P(N_2 = nail|V = hang, P = with, Att = va)$ to be assigned a large probability, and this is achieved by assigning an edge between d_{nails} and d_{nail} with large probability. And since d_{nails} is surely going to get a high probability as a result of the large number of occurrences of the word "nails", d_{nail} is also going to get a high probability. In general, we assign edges between (w_1, b_1) and (w_2, b_2) if w_1 and w_2 belong to the same word class and $b_1 = b_2$. The edge that we just added is called as a link representing a particular inference rule.

6.3.6 Formal Model

In the formal model, the graph remains as described above. The only addition that we do to them, are the addition of inference rule links, where each link type leads from one state with memory bit b_1 to another state with memory bit b_2. Then the final stationary distribution that we compute will be a result of mixture of the basic transition distributions, where the mixture weights are learned automatically. Let the links $l_1, l_2, \ldots l_k$ be given by transition matrices $T^1, T^2, \ldots T^k$. Each matrix T^i has rows for states with memory bits startBit (i) and its rows are distributions over successor states with memory bit endBit (i) . Then the probability of going from (w_1, b_1) to (w_2, b_2) is given by

$$P(w_2, b_2|w_1, b_1) = \sum_{i:startBit(i)=b_1, endBit(i)=b_2} \lambda(w_1, b_1, i) T^i(w_2, b_2|w_1, b_1) \tag{6.8}$$

where $\lambda(w_1, b_1, i)$ is the weight of link type i for the state (w_1, b_1) The probabilities $\lambda(w, b, i)$ sum to 1 over all links l_i having startBit (i) = b. Also the various link types or inference rules that were used are given. They chapter an accuracy of 87.54%.

6.4 Conclusion

A brief overview of the algorithms that were developed in the past for PP-Attachment were explained in detail. It can be seen that lots of algorithms have almost reached the human accuracy rate when given just the 4-tuple. But it has been found that the human accuracy rate is 93.2% given the whole sentence instead of just the 4-tuple. This suggests that there is much more information underlying in the sentence which none of the current algorithms have exploited which is a very promising avenue for extending this research.

Chapter 7

Dependency Parsing

7.1 Introduction

The problem of dependency parsing refers to parsing in the context of grammars in which the structure is determined by the relation between a word (head) and its dependents, such a grammar is called dependency grammar. Dependency grammars are very well suited for linguistic parsing. The general phrase structure grammar does not conform well to the requirements of linguistic parsing of free word order languages like Czech. Dependency parsing is used for finding the syntactic structure of a sentence determined by the relation between a word (head) and its dependents. In dependency syntax, the words are called lexical items and the binary asymmetrical relations are called dependencies. The complete structure showing all the dependencies is called a dependency tree. An example of a dependency tree is shown in Fig 6.1. Before constructing the dependency tree, we need to understand what are the criteria for a dependency. The criteria given by Zwicky [1985] and Hudson [1990] are given below.

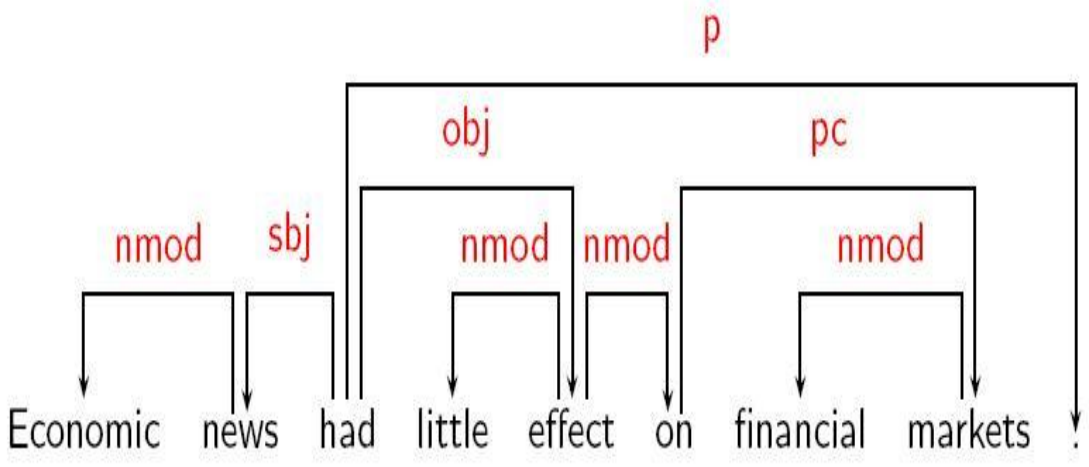


Figure 7.1: An example showing the syntactic structure (dependencies) of a sentence

1. H determines the syntactic category of C; H can replace C.

2. H determines the semantic category of C; D specifies H.
3. H is obligatory; D may be optional
4. H selects D and determines whether D is obligatory.
5. The form of D depends on H (agreement or government)
6. The linear position of D is specified with reference to H,

where H is the head, D is dependent and C is the construction. Similarly there are also conditions on the dependency graph. They are given below along with the intuition behind it. The dependency graph is:

1. Weakly Connected: Syntactic structure is complete
2. Acyclic: Syntactic structure is hierarchical
3. Single head: Every word has at most one head
4. Projectivity, that is, being able to draw the dependencies without criss-crossing. In general this is only an assumption we make to relax the constraints, because otherwise the problem becomes very hard, though non-projectivity is needed for free word order languages or long distance word dependencies.

7.2 Formal definition of the problem

The problem in dependency parsing can now be formally defined as follows,
Input: Sentence x = $w_0, w_1, w_2, \ldots, w_n$ with $w_0 = root$
Output: Dependency tree for G = (V, A) where
V = 0, 1, ..., n is the vertex set,
A is the arc set, i.e, $(i, j, k) \in A$ represents a dependency from w_i to w_j with
label $l_k \in L$.
There are two main approaches used for this problem. The first approach is Grammar-Based parsing which is using a lexicalized context free grammar and general CFG parsing algorithms are used. The second one is the approach that has been discussed in detail in this chapter, which is Data-driven parsing. This approach is further sub-divided into Transition-based methods and Graph-Based methods which are the topics of discussion of the next two sections.

7.3 Machine Learning based methods

A linear classifier is a linear function on the feature vector which computes a score or probability of a classification. Let $w \in \Re^m$ be a high-dimensional weight vector, then the linear function is of the form,
For binary classifier,

$$y = sign(w.f(x)) \tag{7.1}$$

For a multiclass classifier,

$$y = arg_y max(w.f(x, y)) \tag{7.2}$$

A set of points is said to be separable, if there exists a w such that classification is perfect. For a supervised learning algorithm, we need to model some objective function and find the weight vector which optimizes the objective. The perceptron is one of the most basic supervised learning algorithms, and the objective function is the training error function as given below.

$$w = argmin_w \sum_t 1 - y_t == sign(w.f(X)) \tag{7.3}$$

For multiclass classification,

$$w = argmin_w \sum_t 1 - (y_t == argmax_y w.f(X_t, y)) \tag{7.4}$$

The perceptron algorithm is shown in Fig 6.2. It iteratively alters the weight vector such that the number of training errors are reduced, and it can be proven that the training error is bounded, and hence the algorithm is bound to terminate after a finite number of iterations. The margin of a classifier is defined to be the minimum distance between the separating hyperplane and any of the data points. Formally, a training set T is separable with the margin $\gamma \geq 0$ if there exists a vector u with $||u|| = 1$ such that:

$$u.f(x_t, y_t) - u.f(x_t, y') \geq \gamma \tag{7.5}$$

for all $y' \in y_t{}' = y - y_t$ and $||u||$ is the norm of u. It can be seen that to maximize the margin, we can alternatively minimize the norm of w, which is the basis for support vector machines.

7.3.1 Non-Linear Classifiers

Lots of real-life data sets are not linearly separable, in such cases we need to build a non-linear classifier. One of the most common non-linear classifiers is the K-nearest neighbors. In the simplest form, the problem is simplified into finding the K-nearest neighbors in the training set and each of those nodes vote for classification. Thus a training set T, a distance function d, and K define a non-linear classification boundary. The most commonly used distance function is the Euclidean distance.

7.3.2 Kernels

A kernel is a similarity function between two points which is symmetric and positive semi-definite generally denoted as $\phi(x_t, x_r) \in R$. Mercer's theorem states that any kernel can be written as product of some function on the two points. Noting that in the perceptron algorithm, $f(x_t, y_t)$ is added $\alpha_{y,t}$ times and $f(x_t, y_t))$ is subtracted $\alpha_{y,t}$ times to the W vector, then we can write W as,

$$W = \sum_{t,y} \alpha_{y,t}[f(x_t, y_t) - f(x_t, y_t)] \tag{7.6}$$

Using this, the arg max function is written as,

$$y* = argmax_{y*} \sum_{t,y} \alpha_{y,t}[\phi((x_t, y_t), (x_t, y*)) - \phi((x_t, y), (x_t, y*))] \tag{7.7}$$

Using the above equation the perceptron algorithm is rewritten using only kernels as shown in Fig 6.3. It still seems unclear as to how this whole new transformation helps. The computation of the non-linear kernel is much more efficient than computing the dot-product in a higher dimensional feature space, which leads to the whole algorithm being extremely efficient.

7.4 Transition system

Another way to model the construction of the dependency tree is using finite state machines or transition systems. A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$ where

1. C is a set of configurations, each of which contains a buffer of (remaining) nodes and a set A of dependency arcs,
2. T is a set of transitions, each of which is partial function T:C -¿ C
3. c_s is an initialization function, mapping a sentence x = $w_0, w_1, \ldots w_n$ to a configuration with $\beta = [1, \ldots, n]$,
4. $C_t \subseteq C$ is a set of terminal configurations.

A configuration represents a parser state and a transition sequence represents a parsing action. The transitions that the transition system takes determines the set of arcs added to A. Therefore, the only thing that is left is to know which transition to take at every state. A system which tells us which transition to take at every point is called the Oracle and oracles are approximated by linear classifiers. There are some specific models of transition systems using stacks and lists. The stack-based and list-based transition system are shown in Fig 6.3. The stack based transition systems is subdivided into two more transition systems for the purpose of dependency parsing, shift-reduce dependency parsing and arc-eager dependency parsing. They differ only in the way the transitions are modeled. The two are shown in Fig 6.4 and Fig 6.5. It can be clearly seen from the way the transitions are modeled that the space complexity for both transition systems is $\Theta(n)$ whereas the time complexity for stack based transition system is $\theta(n)$ whereas its $\theta(n^2)$ for list-based system.

Now we are concerned with the problem of how to build the oracle given that the systems are in place. As mentioned before the oracles are approximated using classifiers and they are trained using treebank data. The classifier takes a configuration in the form of feature vector of the transition system and outputs a classification or a transition based on the training transition sequences. The feature vectors are usually defined in terms of the target nodes, neighboring words, and the structural context. Most supervised learning algorithms that we discussed above can be used to solve this problem.

7.5 Graph-Based Methods

It can be clearly seen that the dependency graph is a spanning tree since it covers all the vertices (words) , is connected, and is a tree. We define G_x to be the complete multi-digraph for the sentence $x = (w_0, w_1, \ldots, w_n)$ with $V = w_0, w_1, \ldots, w_n$ and $(i, j, k) \in A$ for all $(i, j) \in V \times V$ such that $i \neq j$ and $k \in L$ where L is the set of all labels or dependency relations. Every edge (i, j, k) in the graph has a weight equal to ${w_{ij}}^k$. Now the problem of finding a dependency tree is modeled as finding the maximum spanning tree of G_x. The maximum spanning tree problem is mathematically defined as,

$$G' = argmax_{G' \in T(G)} w(G') = argmax_{G' \in T(G)} \prod_{(i,j,k) \in G'} {w_{ij}}^k \tag{7.8}$$

where T (G) is the set of all spanning trees. From the above formula, it can be seen that we are making the assumption that choosing an edge is dependent from other choices. A simple reduction that is applied to this graph is to replace the $|L|$ edges between every pair of vertices

A stack-based configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is a triple $c = (\sigma, \beta, A)$, where

1. σ is a stack of tokens $i \leq m$ (for some $m \leq n$),
2. β is a buffer of tokens $j > m$,
3. A is a set of dependency arcs such that $G = (\{0, 1, \ldots, n\}, A)$ is a dependency graph for x.

A stack-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where

1. C is the set of all stack-based configurations,
2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [1, \ldots, n], \emptyset)$,
3. T is a set of transitions, each of which is a function $t : C \rightarrow C$,
4. $C_t = \{c \in C | c = (\sigma, [\,], A)\}$.

Notation:

- $\sigma|i$ = stack with top i ($|$ left-associative)
- $i|\beta$ = buffer with next token i ($|$ right-associative)

Figure 7.2: Stack Based transition System

Transitions:

- Left-Arc$_k$:
$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$
- Right-Arc$_k$:
$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$$
- Shift:
$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$$

Preconditions:

- Left-Arc$_k$:
$$\neg[i = 0]$$
$$\neg\exists i' \exists k' [(i', i, k') \in A]$$
- Right-Arc$_k$:
$$\neg\exists i' \exists k' [(i', j, k') \in A]$$

Figure 7.3: Shift-Reduce Dependency parsing

Transitions:

- Left-Arc$_k$:
  $(\sigma|i,j|\beta,A) \Rightarrow (\sigma,j|\beta,A\cup\{(j,i,k)\})$
- Right-Arc$_k$:
  $(\sigma|i,j|\beta,A) \Rightarrow (\sigma|i|j,\beta,A\cup\{(i,j,k)\})$
- Reduce:
  $(\sigma|i,\beta,A) \Rightarrow (\sigma,\beta,A)$
- Shift:
  $(\sigma,i|\beta,A) \Rightarrow (\sigma|i,\beta,A)$

Preconditions:

- Left-Arc$_k$:
  $\neg[i=0]$
  $\neg\exists i'\exists k'[(i',i,k')\in A]$
- Right-Arc$_k$:
  $\neg\exists i'\exists k'[(i',j,k')\in A]$
- Reduce:
  $\exists i'\exists k'[(i',i,k')\in A]$

Figure 7.4: Arc-Eager Dependency parsing

A list-based configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is a quadruple $c = (\lambda_1, \lambda_2, \beta, A)$, where

1. λ_1 is a list of tokens $i_1 \leq m_1$ (for some $m_1 \leq n$),
2. λ_2 is a list of tokens $i_2 \leq m_2$ (for some m_2, $m_1 < m_2 \leq n$),
3. β is a buffer of tokens $j > m_2$,
4. A is a set of dependency arcs such that $G = (\{0,1,\ldots,n), A)$ is a dependency graph for x.

A list-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where

1. C is the set of all list-based configurations,
2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [\,], [1,\ldots,n], \emptyset)$,
3. T is a set of transitions, each of which is a function $t : C \rightarrow C$,
4. $C_t = \{c \in C | c = (\lambda_1, \lambda_2, [\,], A)\}$.

Notation:

- $\lambda_1|i$ = list with head i and tail λ_1 (| left-associative)
- $i|\lambda_2$ = i and tail λ_2 (| right-associative)

Figure 7.5: List-based Transition System

Transitions:

- Left-Arc$_k$:
  $(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, \lambda_2, j|\beta, A \cup \{(j, i, k)\})$
- Right-Arc$_k$:
  $(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1|i|j, [\,], \beta, A \cup \{(i, k, j)\})$
- No-Arc:
  $(\lambda_1|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i|\lambda_2, \beta, A)$
- Shift:
  $(\lambda_1, \lambda_2, i|\beta, A) \Rightarrow (\lambda_1.\lambda_2|i, [\,], \beta, A)$

Preconditions:

- Left-Arc:
  $\neg[i = 0]$
  $\neg\exists i' \exists k' [(i', k', i) \in A]$
- Right-Arc:
  $\neg\exists i' \exists k' [(i', k', j) \in A]$
- No-Arc:
  $\exists i' \exists k [(i', k, i) \in A]$

Figure 7.6: Projective Parsing for list-based transition systems

by ${w_{ij}}^k = argmax_k {w_{ij}}^k$. Now the complete multi-digraph has been reduced to a simple digraph. McDonald et al, use the Chu-Liu Edmonds algorithm to compute the maximum spanning tree. But a small problem is that the Chu-Liu-Edmonds algorithm assumes the weight of the spanning tree to be the sum of weights of edges. This can be easily rectified by taking logarithms.

$$G = argmax_{G' \in T(G)} \prod_{(i,j,k) \in G'} {w_{ij}}^k \tag{7.9}$$

$$= argmax_{G' \in T(G)} log \prod_{(i,j,k) \in G'} {w_{ij}}^k \tag{7.10}$$

$$= \prod_{(i,j,k) \in G'} log {w_{ij}}^k \tag{7.11}$$

Therefore, we can set ${w_{ij}}^k$ to $log\ {w_{ij}}^k$ and apply the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965;. Edmonds, 1967) which is given in Fig 6.6. The algorithm first finds the maximum incoming arc for each vertex and if this happens to be a tree then it is the maximum spanning tree. Else we contract the cycle and consider that as a single vertex and recalculate arc weights into and out of cycle. Now we have a smaller graph compared to the original graph and we recursively apply the algorithm on the reduced graph. The way we compute the arc weights after contraction is

1. Outgoing arc weight = Equal to the max of outgoing arc over all vertices in cycle
2. Incoming arc weight = Equal to the weight of the best spanning tree that includes the head of the incoming arc and all nodes in cycle

Chu-Liu-Edmonds(G_x, w)

1. Let $M = \{(i^*, j) : j \in V_x, i^* = \arg\max_{i'} w_{ij}\}$
2. Let $G_M = (V_x, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $< G_C, c, ma >= \text{contract}(G, C, w)$
6. Let $G = \text{Chu-Liu-Edmonds}(G_C, w)$
7. Find vertex $i \in C$ such that $(i', c) \in G$ and $ma(i', c) = i$
8. Find arc $(i'', i) \in C$
9. Find all arc $(c, i''') \in G$
10. $G = G \cup \{(ma(c, i'''), i''')\}_{\forall (c,i''') \in G} \cup C \cup \{(i', i)\} - \{(i'', i)\}$
11. Remove all vertices and arcs in G containing c
12. return G

contract$(G = (V, A), C, w)$

1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $i \in V - C : \exists_{i' \in C}(i', i) \in A$
   Add arc (c, i) to G_C with
   $ma(c, i) = \arg\max_{i' \in C} score(i', i)$
   $i' = ma(c, i)$
   $score(c, i) = score(i', i)$
4. For $i \in V - C : \exists_{i' \in C}(i, i') \in A$
   Add edge (i, c) to G_C with
   $ma(i, c) = \arg\max_{i' \in C} [score(i, i') - score(a(i'), i')]$
   $i' = ma(i, c)$
   $score(i, c) = [score(i, i') - score(a(i'), i') + score(C)]$
   where $a(v)$ is the predecessor of v in C
   and $score(C) = \sum_{v \in C} score(a(v), v)$
5. return $< G_C, c, ma >$

Figure 7.7: Chu-Liu-Edmonds Algorithm, Y. J. Chu and T. H. Liu, On the shortest arborescence of a directed graph, Science Sinica, v.14, 1965, pp.1396-1400 and J. Edmonds, Optimum branchings, J. Research of the National Bureau of Standards, 71B, 1967, pp.233-240.

Now that we have a method for solving the problem given a weighted digraph, the only problem is how to find the weights. As before, we turn to linear classifiers for this. The weights are computed as,

$$w_{ij}{}^{k} = e^{w.f(i,j,k)} \tag{7.12}$$

The features that could be used for this purpose were discussed in McDonald et al [2005]. Some of them are listed below.

1. Part-of-speech tags of the words w_i and w_j and the label l_k
2. Part-of-speech of words surrounding and between w_i and w_j
3. Number of words between w_i and w_j, and their orientation
4. Combinations of the above

Thus we can calculate the weights using the machine learning algorithms as discussed above.

7.6 Conclusion

This chapter clearly explained about the problem of dependency parsing and showed how it can be expressed as a classification problem. All the required learning algorithms for classification were also discussed. The graph-based models and transition system based models were also explained in the context of dependency parsing.

Chapter 8

Statistical Machine Translation

8.1 Introduction

Automatic translation from one human language to another using computers is known as machine translation. There have been many different approaches to machine translation like Rule-based machine learning, where algorithms are developed to understand and generate the syntactic structure of sentences. The most common approach to machine translation has been the statistical approach which is mainly because of the growing availability of bilingual machine readable-text. The statistical approach refers to probabilistic rules learned from various statistics observed in the bilingual corpora available.

8.2 A simple statistical machine translator

One of the earliest and simple statistical machine translators was given by Brown et al [1]. As with most modern approaches too, the underlying model is based on Bayes theorem, i.e, given a source sentence S seeking a translated sentence T, the error is minimized by choosing the sentence S which is most probable given T, which can be mathematically represented as given below.

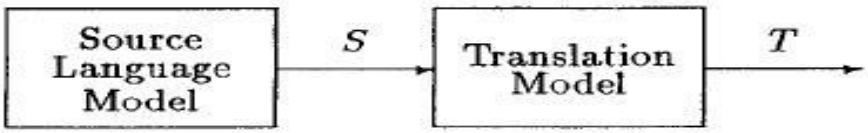
$$Pr(S|T) = \frac{Pr(S)Pr(T|S)}{PR(T)} \tag{8.1}$$

Since the denominator does not depend on T, we can drop it when choosing S to maximize $Pr(S|T)$. The above equation has two factors $Pr(S)$, referred to as the language model, and $Pr(T|S)$ known as the translation model. Also we need a method for searching over all sentences S which maximizes the product of the two factors, this method is often referred to as Decoder. These three methods (models) together constitute a statistical translation system and is shown in Fig 7.1. This is the model that is considered in all the approaches described in this chapter.

$Pr(S)$ can not be computed as such from the corpora statistics, since it is very possible, almost for sure, that we would not have seen every possible sentence in the source language. For example, to compute $Pr(S = s_1, s_2, \ldots, s_n)$ we can use the following formula,

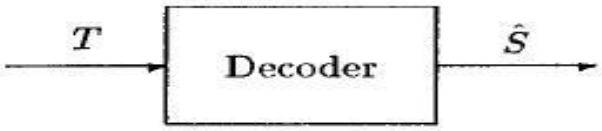
$$Pr(s_1, s_2, \ldots, s_n) = Pr(s_1)Pr(s_2|s_1)\ldots Pr(s_n|s_{n-1}, \ldots, s_1) \tag{8.2}$$

The problem with the above equation is that there are too many probabilities to be computed. Because of this problem they settle for an approximate language model, the N-gram model, which



$$\Pr(S) \times \Pr(T \mid S) = \Pr(S,T)$$

A *Source Language Model* and a *Translation Model* furnish a probability distribution over source-target sentence pairs (S,T). The joint probability $\Pr(S,T)$ of the pair (S,T) is the product of the probability $\Pr(S)$ computed by the language model and the conditional probability $\Pr(T \mid S)$ computed by the translation model. The parameters of these models are estimated automatically from a large database of source-target sentence pairs using a statistical algorithm which optimizes, in an appropriate sense, the fit between the models and the data.



$$\hat{S} = \underset{S}{\operatorname{argmax}} \Pr(S \mid T) = \underset{S}{\operatorname{argmax}} \Pr(S,T)$$

A *Decoder* performs the actual translation. Given a sentence T in the target language, the decoder chooses a viable translation by selecting that sentence $\hat{S}$ in the source language for which the probability $\Pr(S \mid T)$ is maximum.

Figure 8.1: Statistical Machine Translation System, Brown et al, Computational Linguistics Volume 16, Number 2, June 1990

is based on how many N-grams (consecutive set of N words) have been seen before. For example, the formula for calculating $Pr(S = s_1, s_2, \ldots, s_n)$ using a bigram model (N = 2) is,

$$Pr(S = s_1, s_2, \ldots, s_n) = Pr(s_1 \; as \; start \; of \; sentence) Pr(s_2|s_1) \ldots Pr(s_n \; as \; end \; of \; sentence) \tag{8.3}$$

The translation model is a little more complex. There are three different factors in this model. They are $Pr(t|s)$ which denotes the probability that the word t is the translation of s. Before talking about the other two factors we need to introduce the notion of distortion and fertility. For example, consider the French string "Le chien est battu par Jean" whose corresponding English translation is "John does beat the dog". The word "Le" corresponds to "the" in English and "battu" corresponds to "beat" and it can be seen that the French words are not aligned with the corresponding English words, and this is referred to as distortion. The distortions are also modeled using a probability distribution and their model for distribution is $Pr(i|j, l)$ where i is a target position, j is a source position, and l the target sentence length. It can also be seen from the example translation the word "does" does not have any corresponding equivalent in the French sentence, whereas the word "beat" produces two words, "est" and "battu". Fertility is defined as the number of French words that an English word produces in a given alignment. An example alignment is shown in Fig 7.2. Thus the translation model is modeled using three probability distributions, namely, fertility probabilities $Pr(n|e)$ for each English word e and some moderate value of n, translation probabilities $Pr(f|e)$ for each French word f and English word e, and the distortion probability distribution.

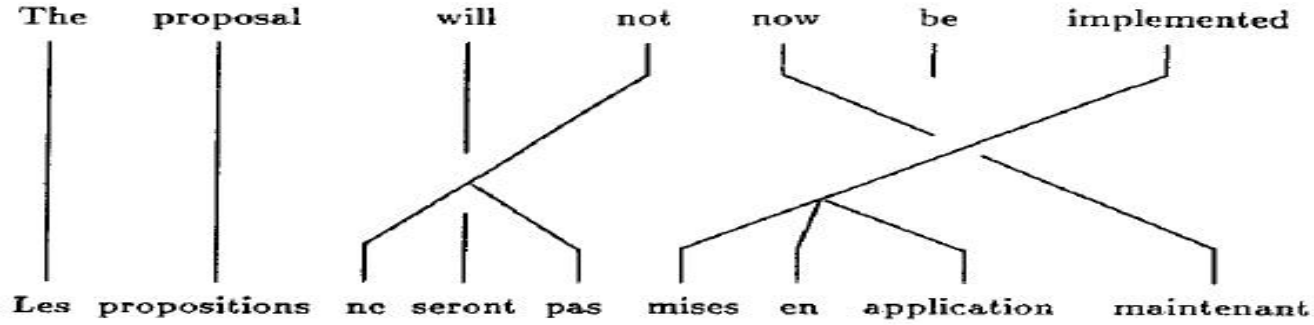


Figure 8.2: An example of Alignment, Brown et al, Computational Linguistics Volume 16, Number 2, June 1990

The decoder is based on stack search, in which initially there is a hypothesis that the target sentence arose in some sense from a sequence of source words that we do not know. The hypothesis is then extended through a series of iterations. The most promising entries are extended in every iteration. For example if the initial state in the stack is $(Jean \; aime \; Marie|*)$ then it could be extended in the following ways.

- (Jean aime Marie | John (1) *) ,
- (Jean aime Marie | *loves (2) *) ,
- (Jean aime Marie | *Mary (3)) ,
- (Jean aime Marie | Jeans (1) *)

The search ends when there is a complete alignment on the list that is more promising than other alignments.

8.3 IBM's Statistical Machine Translation System

The IBM's statistical machine translation was described in The mathematics of Statistical Machine Translation by Brown et al[1993] [2]. The modeling of the system takes place in an iterative fashion. The base model remains the same as what I described in the previous section. This model too has three components, the language model, the translation model and the decoder. The language model uses the trigram model $b(z|xy) = \frac{number\ of\ occurrences("xyz")}{number\ of\ occurrences("xy")}$. But it is possible that even in the above model that we can get a value of zero for the number of occurrences count. Hence instead of the above formula they use a smoothed version of it as given below.

$$\begin{aligned} b(z|xy) = 0.95 \times \frac{number\ of\ occurrences("xyz")}{number\ of\ occurrences("xy")} + \\ 0.04 \times \frac{number\ of\ occurrences("yz")}{number\ of\ occurrences("z")} + \\ 0.008 \times \frac{number\ of\ occurrences("xyz")}{total\ words\ seen} + \\ 0.002 \end{aligned}$$

Since we have a 0.002 added to the fractional counts, we will never get a probability of zero. It also makes sense to have different coefficients chosen for different trigrams that we are calculating the probability for. Before moving on to the translation model we need to devise a method to compare two different language models. For this reason, they introduce the notion of perplexity.

$$perplexity = 2^{\frac{-log(P(e))}{N}} \tag{8.4}$$

where N is the number of words in the test data. A good model will have relatively high P (e) values and hence low perplexity. Thus lower the perplexity, the better the model.

Now the translation model is initially a set of three probability distributions which are, the fertility distribution, the translation distribution and the distortion distribution. The difference between this translation model and the translation model described in the previous section lies in the slight change in the distortion distribution. The distortion distribution of this model is slightly simpler. Here the distribution is $Pr(t_{pos}|s_{pos})$ which is the probability that a source word at s_{pos} will generate a target word at t_{pos}. The next model is built on top of this model. The distortion probability distribution for this model is not based on the position of the source word alone but instead is based on the lengths of the source and target sentences. There is another addition to this model which is a mechanism to introduce some French words for which there are no corresponding English words. One can think of this as the source sentence having an invisible NULL word at the start of the sentence and that NULL word generates some words in the target language. This introduces another factor which is the probability, p_0, that the NULL word generates a target word. Now the whole model is shown in Fig 7.3.

This leads to the problem of how to estimate these probability distributions automatically from large corpora. Assuming we have aligned sentence pairs then computing the distribution is fairly easy. Since we can just count the number of times the event occurs and divide by the size of the sample space. But since we do not have aligned sentence pairs we choose to generate a distribution called the alignment probability distribution. Suppose we have computed the alignment probabilities $Pr(a|e, f)$, now consider a particular sentence pair and compute the counts required for the distortion distribution, the fertility distribution and the translation distribution considering all possible alignments. Then the counts are weighted as follows

$$fractional - count = \sum_{all\ possible\ alignments} Pr(a \mid e, f) \times count \tag{8.5}$$

1. For each English word ei indexed by i = 1, 2, ..., l, choose fertility phi-i with probability n (phi-i | ei).
2. Choose the number phi-0 of "spurious" French words to be generated from e0 = NULL, using probability p1 and the sum of fertilities from step 1.
3. Let m be the sum of fertilities for all words, including NULL.
4. For each i = 0, 1, 2, ..., l, and each k = 1, 2, ..., phi-i, choose a French word tau-ik with probability t (tau-ik | ei).
5. For each i = 1, 2, ..., l, and each k = 1, 2, ..., phi-i, choose target French position pi-ik with probability d(pi-ik | i,l,m).
6. For each k = 1, 2, ..., phi-0, choose a position pi-0k from the phi-0 - k + 1 remaining vacant positions in 1, 2, ... m, for a total probability of 1/phi-0!.
7. Output the French sentence with words tau-ik in positions pi-ik (0 <= i <= l, 1 <= k <= phi-i).

Figure 8.3: Statistical Machine Translation System, Kevin Knight, Statistical machine translation tutorial workbook, JHU summer workshop [1999]

We can compute $Pr(a \mid e, f)$ as $\frac{Pr(a,f|e)}{Pr(f|e)}$. The numerator of this fraction can be computed using the model itself. And the denominator can be computed as

$$Pr(f \mid e) = \sum_{all\ possible\ alignments} Pr(a, f \mid e). \tag{8.6}$$

Therefore the whole problem is reduced to computing $Pr(a, f \mid e)$. The formula for computing this probability is given below.

$$Pr(a, f \mid e) = \binom{m - phi_0}{phi_0} \times p_0 \times (m - 2phi_0) \times p_1^{phi_0} \times$$

$$\sum_{i=1}^{n} n(phi_i \mid e_i) \times \sum_{j=1}^{m} t(f_j \mid e_{aj}) \times$$

$$\sum_{j:a_j \neq 0}^{m} d(j \mid a_j, l, m) \times \sum_{i=0}^{l} phi_i! \times \frac{1}{phi_0!}$$

where,

1. e = English sentence
2. f = French sentence
3. e_i = the i^{th} English word
4. f_j = the j^{th} French word
5. l = number of words in English sentence
6. m = number of words in French sentence
7. a = alignment (vector of integers $a_1, \ldots, a_m$, where each a_j ranges from 0 to 1)
8. a_j = the English position connected to by the j^{th} French word in alignment a
9. e_{aj} = the actual English word connected to by the j^{th} French word in alignment a
10. phi_i = fertility of English word i (where i ranges from 0 to 1) , given the alignment a.

There is one problem with the above methodology for computation of the parameters. Initially we said we can use the parameter values to be able to compute $P(a, f \mid e)$ and for getting parameter values we need to use $P(a, f \mid e)$. The way we solve this problem is using the standard Expectation minimization algorithm. The idea for this algorithm is to use uniform probability values and an uniformly random value for p_1 and now we can compute the alignments probabilities and using these probabilities we can compute the counts for the parameter values. We repeat this procedure again and again and they will converge on good parameter values. There is still one problem with this approach. It computes the probability distributions by enumerating all alignments, the number of alignments are too many to compute even for a single pair of sentences. Fortunately there is an efficient way of computing the probability distributions without enumerating all the alignments which is explained below.

$$Pr(a \mid e, f) = \frac{Pr(a, f \mid e)}{\sum_{all\ possible\ alignments} P(a, f \mid e)} \tag{8.7}$$

The denominator can be computed as

$$\sum_{all\ possible\ alignments} P(a, f \mid e) = \sum_{all\ possible\ alignments} \sum_{j=1}^{m} t(f_j \mid e_{aj}) \tag{8.8}$$

When the above equation's right hand side is expanded and factorized by taking common factors out we get the following equation.

$$\sum_{all\ possible\ alignments} \sum_{j=1}^{m} t(f_j \mid e_{aj}) = \sum_{j=1}^{m} \sum_{i=0}^{l} t(f_j \mid e_i) \tag{8.9}$$

More methods to estimate parameter values are given in the Mathematics of Statistical translation by Brown et al. All of these methods rely on using a simple model to estimate the parameter values and these estimates are passed to the more complex model as initial parameter values and then these estimates are used as the initial values of the EM algorithm.

8.4 Conclusion

This chapter covered the fundamental algorithms in statistical machine translation (SMT) . The source-channel approach, also called the Bayesian approach is predominantly the standard approach in SMT and all the basics of this approach has been discussed in this chapter.

Chapter 9

Decoders and non-source channel approaches for Machine Translation

9.1 Introduction

The predominant approach taken towards Statistical Machine Translation (SMT) is the source channel approach, which was covered in the last chapter. The IBM models were also based on this approach. However, there are a few shortcomings of this approach as given below.

1. The IBM model translation operations are movement (distortion) , duplication, translation of individual words. Thus it does not model the structural and syntactic aspects of the source language.

2. The combination of the language model and the translation model can be proved to be the optimal translation models if the probability distributions of the language model and the translation model are accurate. But it is often the case that the approximations are poor.

3. There is no way to extend the model to incorporate other features and dependencies.

4. The model is trained using the maximum-likelihood approach but the final evaluation criterion is totally different. We would rather prefer to train a model such that the end-to-end performance is optimal.

In the remaining sections, we will discuss approaches that overcome the above shortcomings.

9.2 Direct Modeling of the Posterior probability

This approach tries to directly model the posterior probability $Pr(e \mid f)$ instead of using the Bayes rule. This is the approach taken in Och et al[2002][2003]. The framework for this model is based on the maximum entropy model (Berger et al, 1996) . In this framework there are M feature functions $h_i(e, f)$ for all $i = 1, 2, \ldots, M$ each having a model parameter λ_i. In this model the translation probability is given by,

$$Pr(e \mid f) = \frac{exp[\sum_{i=1}^{M} \lambda_i h_i(e, f)]}{\sum_{e'} exp[\sum_{i=1}^{M} \lambda_i h_i(e', f)]} \tag{9.1}$$

The decision is then based on the following equation,

$$\hat{e} = argmax_e\{\sum_{i=1}^{M} \lambda_i h_i(e, f)\} \tag{9.2}$$

Now the problem is to train the model, in other words, finding parameter values. The parameter values are set using the equation

$$\hat{\lambda_i} = argmax_{\lambda_i}\{\sum_{s=1}^{S} logp_{\lambda_i}(e_s \mid f_s)\} \tag{9.3}$$

This corresponds to maximizing the likelihood of the direct translation model. Some of the features which could be used are,

1. Sentence Length feature:

$$h(f, e) = length(e) \tag{9.4}$$

   This tries to pull down the sentence length of the generated sentence.

2. Other language models could be used in the following way,

$$h(f, e) = h(e) \tag{9.5}$$

3. Lexical features which fire if a certain lexical relationship (f, e) occurs

$$h(f, e) = (\sum_{j=1}^{J} \delta(f, f_j))(\sum_{i=1}^{I} \delta(e, e_i)) \tag{9.6}$$

4. We could use certain grammatical dependencies between the source and target language, suppose k (.) counts the number of verb groups that exists in source or target language, then one possible feature function could be,

$$h(f, e) = \delta(k(f), k(e)) \tag{9.7}$$

The training is done using the Generalized Iterative Scaling algorithm. There are two problems with this model right now. The first problem is that the renormalization step in equation 8.1 will take a long time since it considers all possible sentences. This problem is overcome using not all the sentences but considering only the best n probable sentences. The second problem is that in equation 8.3, there is not one single reference translation but R_s reference translations. Hence that equation is modified to accommodate this,

$$\hat{\lambda_i} = argmax_{\lambda_i}\{\sum_{s=1}^{S} \frac{1}{R_s} \sum_{r=1}^{R_s} logp_{\lambda_i}(e_s, r \mid f_s)\} \tag{9.8}$$

For the purpose of evaluation, they do not use just a single evaluation criterion mainly because there is no one single accepted criterion. A few of the evaluation criterion they have used are,

1. SER (Sentence Error Rate) : This is the number of sentences which match exactly with one of the reference translations.

2. WER (Word Error Rate) : This is the minimum number of substitution, insertion and deletion operations that need to be performed to transform the generated sentence into the reference translation.

3. mWER (multi-reference word error rate) : Since there are multiple reference translations, it is better to use the minimum edit distance between the generated translation and reference translations

4. BLEU: This commonly used measure is the geometric mean of the precision of $n-grams$ for all values of $n \leq N$ between the generated translation and the reference translation multiplied by a factor BP (.) that penalizes short sentences

$$BLEU = BP(.).exp(\sum_{n=1}^{N} \frac{log(p_n)}{N}), \tag{9.9}$$

where p_n denotes the precision of $n-grams$ in the generated translation. Clearly larger number of common $n-grams$ is better, hence larger BLEU scores are better.

The experiments were conducted on the VERBMOBIL task which is a speech translation task in the domain of hotel reservation, appointment scheduling etc. The results were such that addition of each new feature results in a systematic improvement in almost all the evaluation criterion. In Och et al[2003], they show how to use the same evaluation criterion when training and hence obtain optimal end-to-end performance. In this paper they train the model considering different evaluation criterion like BLEU, mWER, etc. Without loss of generality, assume the function $E(r, e)$ gives the error between r and e. Then the total error for s pairs of sentences is given by $total\ error = \sum_{s=1}^{S} E(r_s, e_s)$. Then in such a setting the training is performed using the equation below.

$$\hat{\lambda_i} = argmin_{\lambda_i}\{\sum_{s=1}^{S} E(r_s, \hat{e}(f_s; \lambda_i))\} = argmin_{\lambda_i}\{\sum_{s=1}^{S}\sum_{k=1}^{K} E(r_s, e_{s,k})\delta(\hat{e}(f_s; \lambda_i), e_{s,k})\} \tag{9.10}$$

with

$$\hat{e}(f_s; \lambda_i) = argmax_{e \in C_s}\{\sum_{m=1}^{M} \lambda_m h_m(e \mid f_s)\} \tag{9.11}$$

The above method has two problems, of which the first problem is the argmax operation because of which one can not use the gradient descent method. The second problem is that there are many local optima. Both these problems are overcome by using a slight approximation (smoothed) version of the above equation as given below.

$$\hat{\lambda_i} = argmin_{\lambda_i}\{\sum_{s,k} E(e_{s,k}) \frac{p(e_{s,k} \mid f)^{\alpha}}{\sum_k p(e_{s,k} \mid f)^{\alpha}}\} \tag{9.12}$$

To see how the above equation has helped solve the two problems mentioned, consider Figure 7.1, the unsmoothed error count has clearly many optima compared to the smoothed error count, which makes it easier for the optimization algorithm. They also propose a new optimization algorithm for optimizing the unsmoothed error count given in Eq 10. Their experiments were run on the 2002 TIDES Chinese-English small data track task. Their results indicate that it is indeed the case that training based on the final evaluation criterion gives better results at the end. This suggests the possibility of using an evaluation criterion which correlates well with human evaluation and improve the quality of translation systems and even other NLP-related tasks.

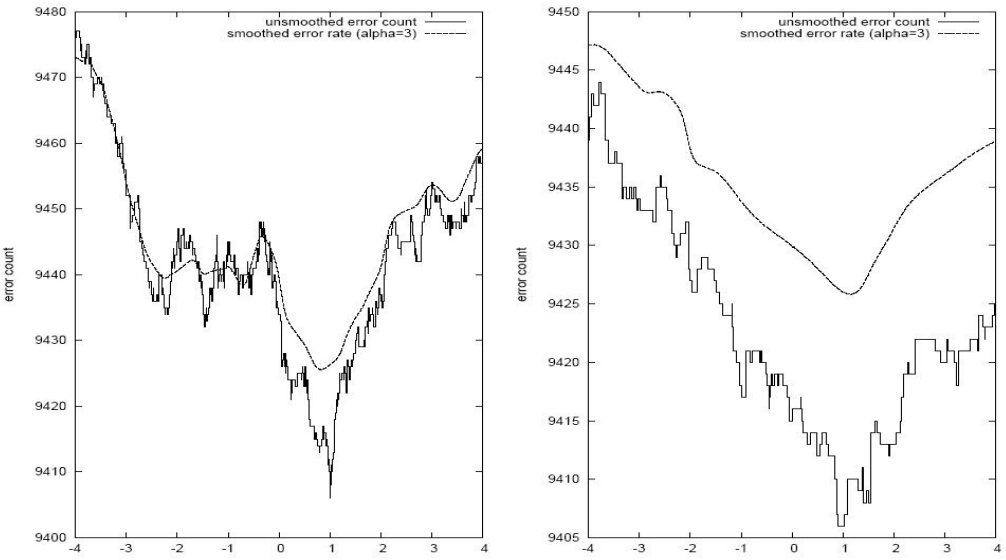


Figure 9.1: The graph on the left is the error count and the the graph on the right is the smoothed error count, Och et al, Proceedings of the 41st Association for Computational Linguistics, 2003

9.3 Syntactical Machine translation

One of the major criticisms of the IBM models for SMT is that they do not incorporate syntax or the structural dependencies of the languages they model. It is suspected that the model may not work well for a pair of languages with different word order like English and Japanese. This problem was approached using translating a source parse tree instead of translating a source sentence. This approach was suggested in Yamada et al[2001] and Charniak et al[2003]. Since the input is a source parse tree, the operations required should be different from those required for the IBM model. The operations they have suggested are reordering child nodes, inserting extra words at each node, and translating leaf words. The reordering operation is to bridge the word order differences in SVO-languages and SOV-languages. The word-insertion operation is intended to capture linguistic differences to specify syntactic cases. Figure 7.2 shows how these operations are used to translate English sentences into Japanese.

Each of the operations are modeled using a probability distribution. The reordering probability distribution is modeled using the r-table which gives the probability of reordering the child nodes of a non-terminal node. The insertion operation is modeled using two probability distributions, the first of which gives the probability of inserting at a particular position and the second distribution gives the probability of the word to be introduced. The translation probability distribution, the t-table specifies the probability $t(w_i, w_j)$, of translating word w_i into word w_j. Formally speaking, there are three random variables N, R, and T which model the channel operations. Let the English sentence be $(\epsilon_1, \epsilon_2, \ldots \epsilon_n)$ and the target sentence be $(f_1, f_2, \ldots, f_n)$. The notation $\theta = (v, p, t)$ be the values associated with the random variables N, R and T. The probability of the translated sentence given the English parse tree is,

$$p(f \mid \epsilon) = \sum_{\theta: str(\theta(\epsilon))=f)} p(\theta \mid \epsilon) \tag{9.13}$$

where $str(\theta(\epsilon))$ is the sequence of leaf words of a tree transformed by θ from ϵ. Assuming all the

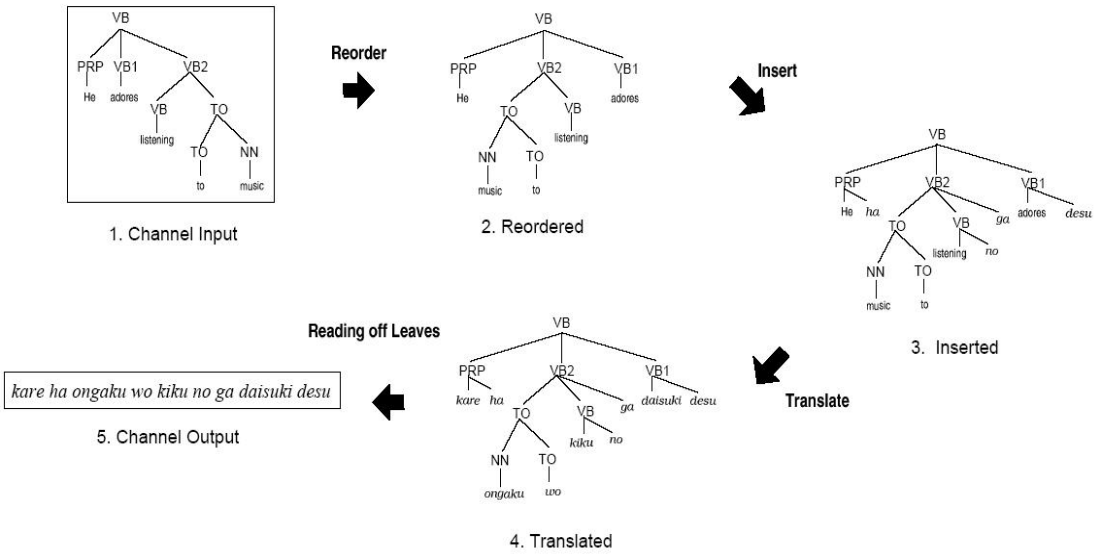


Figure 9.2: Channel Operations: Reorder, Insert and Translate, Yamada et al, Proceedings of the 39th Association for Computational Linguistics, 2001

transform operations are independent and the random variables are determined only by the node itself. Therefore,

$$p(f \mid \epsilon) = \sum_{\theta: str(\theta(\epsilon))=f} \prod i = 1^n n(v_i \mid N(\epsilon_i)) r(p_i \mid R(\epsilon_i)) t(t_i \mid T(\epsilon_i)) \tag{9.14}$$

The model parameters are estimated automatically from the corpus using the algorithm given in Figure 7.3. Since there are too many combinations possible in the algorithm, they have given an EM algorithm which computes the parameters in polynomial time.

The evaluation was done by human judgement of the alignments produced by the decoder. These alignments were graded on a scale of 1.0 with three quantization levels of 1.0 being Correct, 0.5 being Not sure, and 0.0 Wrong. The average alignment score was 0.582 whereas IBM model 5 scored 0.431 justifying the need to model the syntactical aspects of the languages.

9.4 Fast Decoder

In the previous section we saw the use of syntax-based machine translation system. The performance of such a system is dependent on the decoder used. One such decoder for the syntax-based translation system discussed above was proposed in Yamada and Knight [2002]. Since the system works on the noisy channel model, the decoder works in the reverse direction of the channel, i.e, given a Chinese sentence they try to find the most probable English parse tree. To parse in such a way, they obtain a Context Free Grammar (CFG) for English using the training corpus (pairs of English parse trees and Chinese sentences) . This grammar needs to be extended to include the channel operations like reordering, insertion, and translation. These rules are added along with the corresponding probabilities from the r-table, n-table and t-table respectively. Using this information we can build a decoded tree in the foreign language order. It is very simple to convert this tree to an English parse tree. We just need to apply the reorder operations in reverse and translating the

1. Initialize all probability tables: $n(\nu|\mathcal{N})$, $r(\rho|\mathcal{R})$, and $t(\tau|\mathcal{T})$.
2. Reset all counters: $c(\nu, \mathcal{N})$, $c(\rho, \mathcal{R})$, and $c(\tau, \mathcal{T})$.
3. For each pair $\langle \mathcal{E}, \mathbf{f} \rangle$ in the training corpus,
   For all $\boldsymbol{\theta}$, such that $\mathbf{f} = \mathrm{Str}(\boldsymbol{\theta}(\mathcal{E}))$,
   - Let cnt = $\mathrm{P}(\boldsymbol{\theta}|\mathcal{E}) / \sum_{\boldsymbol{\theta}:\mathrm{Str}(\boldsymbol{\theta}(\mathcal{E}))=\mathbf{f}} \mathrm{P}(\boldsymbol{\theta}|\mathcal{E})$
   - For $i = 1 \ldots n$,
     $c(\nu_i, \mathcal{N}(\varepsilon_i))$ += cnt
     $c(\rho_i, \mathcal{R}(\varepsilon_i))$ += cnt
     $c(\tau_i, \mathcal{T}(\varepsilon_i))$ += cnt
4. For each $\langle \nu, \mathcal{N} \rangle$, $\langle \rho, \mathcal{R} \rangle$, and $\langle \tau, \mathcal{T} \rangle$,
   $n(\nu|\mathcal{N}) = c(\nu, \mathcal{N}) / \sum_{\nu} c(\nu, \mathcal{N})$
   $r(\rho|\mathcal{R}) = c(\rho, \mathcal{R}) / \sum_{\rho} c(\rho, \mathcal{R})$
   $t(\tau|\mathcal{T}) = c(\tau, \mathcal{T}) / \sum_{\tau} c(\tau, \mathcal{T})$
5. Repeat steps 2-4 for several iterations.

Figure 9.3: Training Algorithm: Yamada et al, Proceedings of the 39th Association for Computational Linguistics, 2001

Chinese words into English using the t-table. Then the probability can be calculated as the product of the probability of the parse tree and the probability of translating. The probability of the parse tree is the prior probability of the sentence, which is computed using n-gram language model. The probability of translating is the product of probabilities associated with each rule that we applied during converting the decoded tree. Thus we can choose the tree with the highest probability. The only problem that we are left with right now is that the space of decoded trees is very high. This is easy to imagine since there are many sequences of channel operations that can be performed which give the same tree. For example in their experiment in converting Chinese News into English, there are about 4M non-zero entries in the t-table and about 10k CFG rules for parsing English and about 120k rules added to that for accommodating the channel operations. Thus this algorithm is not going to work because it would take too much time to translate even small sentences. The solution to this is forego optimality and build a less time-consuming approximate decoder. The idea is to use a dynamic programming based beam search algorithm. The dynamic programming part is to build the parse tree bottom up. The recurrence is on the tuple ¡non-terminal, input substring¿. If the parser cost can be computed only based on the features of the subtree, then we choose to keep only the best subtree, otherwise we choose to keep the beam-width best subtrees. For decoding faster, they have suggested pruning based on the different probability distributions. For example, for a given Chinese sentence, we only consider those English words e such that $P(e \mid f)$ is high for the Chinese words f. Pruning based on the r-table is done by choosing the top N rules such that the $\sum_{i=1}^{N} prob(rule) \times prob(reord) > 0.95$. This limits the number of rules to be used. In their experiment, of the total of 138, 862 rules, only 875 rules contribute towards the 95% of the probability mass. Similarly other pruning rules are used to further limit the number of choices available at every stage in the algorithm.

9.5 Conclusion

In this chapter, the developments regarding decoders and non-source channel approaches are covered in the field of statistical machine translation.

Chapter 10

Sentiment Analysis

10.1 Introduction

Sentiment analysis refers to the process of determining the attitude, emotion of a speaker with respect to some topic. This relatively new field of research is a very interesting and challenging field. The attraction and motivation for this field is largely based on the potential applications. For example, most search engines currently in use are keyword-based but there are other features of the textual data based on which we could search, for example we could search based on the sentiment of the page, or also known as the polarity of the page. As another use of sentiment analyzer, consider a company X which has a product P and there are many feedback mails about it. In this case the sentiment analyzer could be used to find what features of the product P are appreciated and what features have to be improved.

Broadly speaking, there are two tasks associated with sentiment analysis. One of the tasks is to find the polarity of the sentence and the second task is to assign the sentiment to the target of the sentiment. For example, the sentence "The Canon Powershot camera is great" conveys a positive sentiment towards the Canon Powershot camera, which is the target.

This chapter is organized in the following way. Section 2 discusses some of the graph based approaches for sentiment analysis and section 3 reviews other approaches followed by the conclusion in Section 5.

10.2 Graph Based Methods

This section would cover two graph based methods for sentiment analysis. One of the approaches models the graph using only the information from the sentences in the document, whereas another approach uses the results of another classifier.

10.2.1 Using minimum cuts for sentiment analysis

This paper by Pang et al, deals with the problem of identifying which sentences are subjective in a given document. The idea is that if we reduce the number of objective sentences in a document then the accuracy of document polarity classifiers would improve. For this purpose they use existing classifiers for predicting polarity of a document which are based on Support Vector Machines (SVM) and Naive Bayes (NB) . The architecture of their system is shown in Fig 9.1. The contribution of this paper is stage 2 in the diagram which is the subjectivity detector. They model the problem in terms of the min-cut graph problem. They define a weighted undirected graph for a document

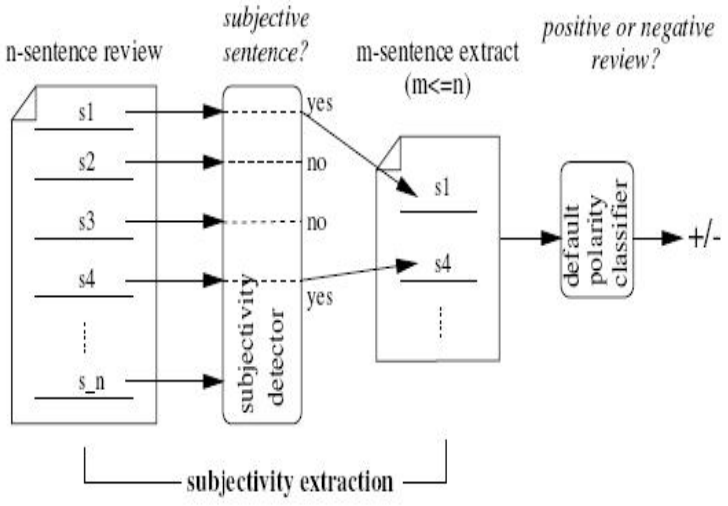


Figure 10.1: Architecture of the Sentiment Analyzer, B Pang and L Lee, A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, Proceedings of ACL, pp. 271–278, 2004.

in the following way. Graph G = (V, E) where the set of vertices, V, is the set of sentences in the document and 2 extra vertices, source and sink. The source and sink are the two different classifications, in our case, the subjective node, and the objective node. The edges between the two nodes x_i and x_j have a weight $assoc(x_i, x_j)$, proportional to the importance of both the nodes being in the same class. Each edge between x and the class node C_i, has a weight $ind_i(x)$ corresponding to the importance of x being in class C_i. A cut in the graph G is defined as splitting the vertex set V into two disjoint sets S and S' such that $S \bigcup S' = V$. The cost of the cut is defined to be

$$Cost = \sum_{x \in C_1} ind_1(x) + \sum_{y \in C_2} ind_2(y) + \sum_{x \in C_1, y \in C_2} assoc(x, y) \tag{10.1}$$

The min-cut can be computed efficiently using any polynomial-time algorithm designed for this purpose. The biggest strength of this algorithm is that we could use knowledge about the language to get the costs for the edges between a sentence node and the class node and simultaneously use knowledge-lean methods to get the assoc (x, y) weights. Now the only problem that remains is how to compute the weights. The individual weights are computed by running a Naive Bayes algorithm or SVM on the sentences of the document. The feature vector, f, they use is the bag of words vector, i.e, f (x) =1 if the word x is present in the document. Then the weight of the edge between a sentence and C_1, where C_1 is the subjective class, is set to the probability that the sentence is subjective according to the Naive Bayes algorithm. The association weights are set as a function of the proximity between the two sentences.

$$assoc(s_i, s_j) = f(j - i) \; if \; (j - i) \; \leq \; T \; and \; 0 \; otherwise \tag{10.2}$$

The different functions tried for $f(d) = 1, e^(1-d), \; and \; \frac{1}{d^2}$. Thus using this algorithm we obtain all the subjective sentences which are sent to a Naive Bayes document polarity classifier. This improves accuracy of the document polarity classifier to 86.4% when compared to using the whole

document. The number of words in the subjective sentences obtained are only 60% of total words in the document. The data was taken from the movie reviews from rottentomatoes.com. The size of the subjectivity extracts can be controlled by just using the N most subjective sentences, which are the sentences with the highest probability according to the Naive Bayes detector. It is seen that using just the top 15 sentences gives the same accuracy as using the whole review. The accuracies using N-sentence extracts using the Naive bayes and the SVM as default polarity classifiers is shown in Fig 9.2. This paper shows that when more objective sentences are removed the accuracy of the

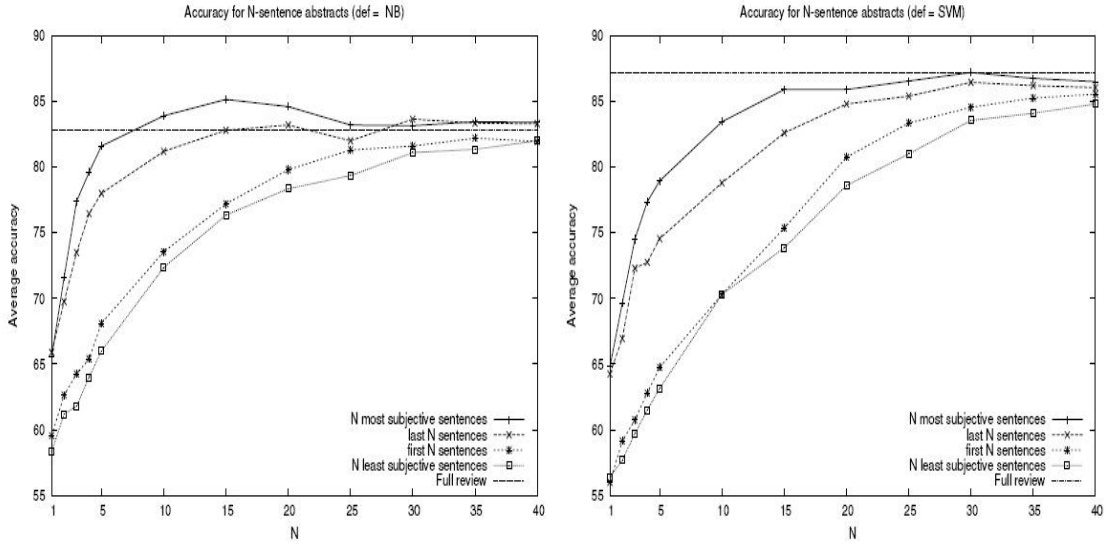


Figure 10.2: Accuracy using N-sentence extracts using the Naive Bayes and SVM, Pang et al, A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts, Proceedings of ACL, pp. 271–278, 2004.

document polarity classifier can be improved.

10.2.2 Graph-Based Semi Supervised Learning for sentiment analysis

The problem they try to solve in this paper by Goldberg et al, is the rating inference problem. The rating inference problem is given lots of reviews and the corresponding numerical rating, we should predict the rating for an unseen review. Unlike the paper by Pang et al, Goldberg et al not only use labeled data but also use unlabeled data to propose a semi-supervised algorithm. The rating inference problem is formally defined as, given n review documents $x_1, x_2, \ldots, x_n$ each represented by a feature vector. The first $l \leq n$ documents are labeled with ratings $y_1, y_2, \ldots, y_n \in C$. The remaining documents are unlabeled in the general setting but can also be optionally labeled with $\hat{(}y_{l+1}), \ldots \hat{(}y_n)$ obtained from another learner. The unlabeled documents also form the test set, this setting is called as transduction. The set of numerical ratings are $C = \{c_1, c_2, \ldots, c_n\}$ with $c_1 \leq \ldots \leq c_2 \in R$. Now the problem is to find a function F:x-¿R that gives a continuous rating. Then classification is performed by mapping f (x) to the nearest quantized value in C.

Now we define how the graph is built. The graph G = (V, E) , V consists of 2n nodes and weighted edges among some of the vertices. Each document is a node in the graph, and all labeled documents have a dongle node attached to it, which is the label of the document. This edge has

a weight of a large number M, which represents the influence of the label. Similarly all unlabeled documents are connected to the label obtained by the previous learner. The weight between the unlabeled document and the dongle node is set to 1. Each unlabeled node is connected to K nearest labeled neighbor nodes. The distance between two documents is measure using a similarity measure w. They use two different similarity measures, one of which is the positive-sentence-percentage and the other is mutual information modulated word-vector cosine similarity. Thus the edge weight between a labeled document x_j and unlabeled document x_i x_i and x_j is $a.w_{ij}$. Similarly each unlabeled document is connected to k' unlabeled documents with a weight of $b.w_{ij}$. In this model k, k', a and b are parameters. It is assumed that nodes which are connected would have similar labels with the probability of similarity being proportional to the weight of the edge between them. Thus the rating function f (x) should be smooth with respect to the graph.

The general approach to semi-supervised learning algorithm is to define an energy function over the graph and minimize the energy function by tweaking the parameters. Let L be $1 \dots l$ and U be $l+1, \dots n$ be labeled and unlabeled vertices respectively. Then the energy function is defined as
$L(f) = \sum_{i \in L} M((f(x_i) - y_i)^2 + \sum_{i \in U} (f(x_i) - \hat{y_i})^2 +$
$\sum_{i \in U} \sum_{j \in KNN_L(i)} a.w_{ij}(f(x_i) - f(x_j))^2 +$
$\sum_{i \in U} \sum_{j \in K'NN_U(i)} b.w_{ij}(f(x_i) - f(x_j))^2$

where $KNN_L(i)$ is the set of labeled neighbors of i and $k'NN_U(i)$ is the set of unlabeled neighbors of i. A loss in this function means that the unlabeled nodes' labels are close to its labeled neighbors and also its unlabeled neighbors, which is exactly what we want, a smooth function over the graph. This formula is modeled as a matrix operation and a formula for minimizing it is obtained. They tune the parameters of their algorithm using cross-validation. Then this algorithm is compared to two other algorithms, which are metric labeling and regression. They use reviews by each author as a dataset. On evaluation it is found that the positive-sentence-percentage similarity function performs better than the cosine similarity between the word vector. The SSL method performs well for small sizes of the labeled data but when there is more labeled data available, the SVM regressor achieves better accuracy which is probably because the positive-sentence-percentage similarity function is not good enough.

10.3 Other approaches

In the paper by Hurst et al, they strongly believe in the need for the fusion of sentiment detection and assignment of the sentiment to a target, referred to as topicality. In other words, they argue that it is not enough to just find the polarity of a sentence but the topicality of the sentence should also be identified and check if the sentiment expressed is towards the topicality. As an example, consider the sentiment analyzer to be used to identify the pros and cons of a product. The sentence "It has a BrightScreen LCD screen and awesome battery life" does not express anything positive towards BrightScreen LCD screen, the product in question, whereas a sentiment analyzer would evaluate this sentence to be positive.

They solve this problem by using a topicality detector and a sentiment detector independent of each other and hope that the sentiment expressed would be towards the topic of the sentence. The polarity detector is explained below.

Their polarity detector works as follows. In the first step, the sentences are tokenized and segmented into discrete chunks. The chunking is done by running a statistical tagger for achieving Part-Of-Speech tagging and tagging each word with a positive/negative orientation using a polarity lexicon. As a final step, these chunks are grouped into higher order groupings and compared with

a small set of syntactic patterns to identify the overall polarity of the sentence. For example, the sentence "This car is really great" is tokenized and POS tagged into (this DET) , (car NN) BNP, (is VB) BVP, (really RR, great JJ) BADJP. The basic chunk categories are DET, BNP, BADVP, BADJP, BVP, OTHER. This POS tagged sentence is grouped into higher order groupings and compared to syntactical patterns. The syntactic patterns used are Predicative modification (it is good) , Attributive modification (a good car) , equality (it is a good car) , Polar cause (it broke my car) . The system also captures negation, for example, sentences like "It is not good", "it is never any good" are correctly identified to be negative though there are positive words in them.

They evaluate their algorithms on 20, 000 messages obtained by crawling usenet, online message boards, etc. Of these messages they found 982 messages to be about a particular topic and used only those messages. These messages when tokenized gave rise to 16, 616 sentences. On evaluation of their polarity detector, they found that positive polarity was detected with a 82% precision and negative polarity with 80% precision.

The topical detector for sentences is very hard to design since there are very few features for a single sentence. Hence they design a topical detector for the whole message at first and then try to use the same algorithm for sentences. The document topical detector works using a winnow algorithm. Assume documents are modeled using the bag-of-words model. The winnow algorithm learns a linear classifier using the given labeled data. The linear classifier is of the following form.

$$h(x) = \sum_{w \in V} f_w c_w(x) \tag{10.3}$$

where $c_w(x)$ is 1 if the word w is present in the document x else it is 0. f_w is the weight of the feature w. If $h(x) \geq V$, then the classifier predicts topical, otherwise predicts irrelevant. The learning algorithm is given below.

1. Initialize all f_w to 1.

2. For each labeled document x in the training set:

   calculate h (x)

   If the document is topical, but winnow predicts irrelevant, update each weight f_w where $c_w = 1$ by $f_w = 2 \times f_w$

   If the document is irrelevant and the winnow predicts relevant, update each weight f_w where $c_w = 1$ by $f_w = \frac{f_w}{2}$

This algorithm is adapted to predict topicality for a sentence. If a document is predicted to be irrelevant, then all the sentences are predicted to be irrelevant. Else the topicality detection algorithm is run on each of the individual sentences considered as the whole document. The result of the algorithm is the prediction for the sentence. On evaluation it is found that the document-level topical detection has a precision of 85.4% while the sentence-level topical detection has a precision of 79%. But the problem is that there is a significant drop in the recall of sentence-level topical detection. This is because the number of features at the sentence-level is very few. Hence for a sentence to be predicted topical, the words should be very topical. Thus for truly topical sentences, the strength of the word weights may not be large enough to overcome the default feature weight leading to a loss in recall. Finally they evaluate their original claim of a sentiment expressed is indeed targeted at the topic of the sentence. A precision of 72% demonstrates that the locality assumption is true in most instances.

10.4 Evaluation of Machine learning algorithms for Sentiment Analysis

In the paper by Pang et al, they analyze the effectiveness of the different machine learning algorithms when applied to sentiment analysis. The accuracy of these algorithms are compared with the human-produced baseline performances. The three learning algorithms that they have evaluated are Naive Bayes, SVM and maximum entropy classification.

First let's consider the human-produced baseline. In this experiment, two graduate students in computer science are asked to come up with positive and negative word lists to classify movie reviews. The classification procedure is very simple. If the frequency of positive words is greater than the frequency of the negative words, the review is classified as positive, and negative otherwise. If the frequencies are equal then it is considered a tie. For both the lists the accuracy rate is quite low and the tie rate is very high. In fact they try to create a list of words by looking at the test data and investigating the frequency counts. Even this list of words has a low accuracy rate of 69% but a better tie rate of 16%. The results of this experiment is shown in Fig 9.3. The results of this experiment is that corpus-based techniques are required to classify based on sentiment. Before discussing the evaluation of the three machine learning algorithms, they explain the feature

	Proposed word lists	Accuracy	Ties
Human 1	positive: *dazzling, brilliant, phenomenal, excellent, fantastic* negative: *suck, terrible, awful, unwatchable, hideous*	58%	75%
Human 2	positive: *gripping, mesmerizing, riveting, spectacular, cool, awesome, thrilling, badass, excellent, moving, exciting* negative: *bad, cliched, sucks, boring, stupid, slow*	64%	39%
Human 3 + stats	positive: *love, wonderful, best, great, superb, still, beautiful* negative: *bad, worst, stupid, waste, boring, ?, !*	69%	16%

Figure 10.3: Results of the human-produced baseline, Pang et al, Proceedings of the ACL-02 conference on Empirical methods in natural language processing, p.79-86, July 06, 2002

vector used. They use the standard bag-of-features framework. Let $\{f_1, f_2, \ldots, f_m\}$ be the set of features for document d. Example features are unigram presence, bigram presence. Let $n_i(d)$ be the frequency of the f_i feature. The document vector is represented as $(n_1(d), n_2(d), \ldots, n_m(d))$ Naive Bayes: One common approach to classify text is to assign a document d to class $C* = argmax_c P(c \mid d)$. Using Bayes theorem

$$P(c \mid d) = \frac{P(c)P(d \mid C)}{P(d)} \tag{10.4}$$

P (d) can be safely left out since it does not affect the choice of c. For computing $P(d \mid c)$ they

assume the features are independent of each other and therefore use the following formula for computing $P(c \mid d)$

$$P_{NB}(c \mid d) = \frac{P(c)\prod_{i=1}^{m} P(f_i \mid c)^{n_i(d)}}{P(d)} \tag{10.5}$$

The training is done using relative frequency estimation of P (c) and $P(f_i \mid c)$, using add-one smoothing.

Maximum Entropy (ME) : It has been shown by Nigam et al (1999) , that maximum entropy classification outperforms Naive Bayes sometimes. It also uses conditional probability to classify. $P(c \mid d)$ is computed as follows

$$P_{ME}(c \mid d) = \frac{1}{Z(d)} exp(\sum_{i} \lambda_{i,c} F_{i,c}(d,c)) \tag{10.6}$$

where Z (d) is a normalization function. $F_{i,c}(d,c)$ is a feature function for feature f_i and class c, defined as follows

$$f_{i,c}(d,c) = 1 \; if \; n_i(d) \; \geq \; 0 \; and \; c' \; = c \; and \; 0 \; otherwise \tag{10.7}$$

One of the advantages of using maximum entropy over naive Bayes is that ME does not make any assumptions about the independence of the feature functions. $\lambda_{i,c}$ is the weight of the feature function for class c. If this value is large then the feature f_i is a strong indicator that the document is of class c. The improved iterative scaling algorithm by Della Pietra et al (1997) is used for training. During training the parameter values are tuned to maximize the entropy of the induced distribution subject to the constraint that the expected value of the feature functions with respect to the model are equal to the expectation with respect to training data.

Support Vector Machines:
In SVM the idea is to find a separating hyperplane of large margin. This hyperplane is used for classification by finding out which side of the hyperplane the input falls into. The search for the separating hyperplane, w, is carried out using the following formula

$$w = \sum_{j} \alpha_j c_j d_j, \alpha_j \geq 0, \tag{10.8}$$

where α_i's are obtained by solving a dual optimization problem. Those d_j's for which $\alpha_j \geq 0$ are called support vectors because they are the only vectors contributing to w.

For the evaluation they use the IMDB movie reviews located in the rec.arts.movies.reviews newsgroup. In their evaluation they find that the SVM is the best performing algorithm while the Naive Bayes is the worst performing one. They conclude by saying that though the algorithms beat the human-produced baseline, they are still very far away from the accuracies obtained in the topic categorization experiment. They also reason this saying it is mainly because of the mixture of positive and negative features in many documents which the bag-of-features feature vector fails to model.

10.5 Conclusion

This chapter covered fundamentals of sentiment analysis and the different approaches used for detecting the polarity of a sentence. To summarize, The problem of identifying opinions and sentiments is a lot harder than a general topic classification task. The reason for it is that the

features used concentrate on the statistics of the corpus, whereas more semantic knowledge should be integrated to have a high precision high recall classifier. It could also be the case that the statistical machine learning algorithms do not perform very well because of the features. The most commonly used feature vector is based on the bag-of-features model. Also it is important to not just classify a sentence as polar, but also put it in context. In other words, we need to bind the sentiment detected with the target of the sentiment.

Chapter 11

Blog Analysis

11.1 Introduction

A Weblog, commonly known as Blog, is a website (mostly personal) where entries are commonly displayed in reverse chronological order. The Blogs are generally used for providing commentary or news on a particular subject or mostly personal diaries. These Blogs are heavily hyperlinked forming many active micro-communities. Also technorati.com (blog search engine) announces that it has been tracking more than 100 million blogs, which means the Blogspace (the space of all Blogs) is a useful source of unstructured information. Recently there has been a lot of work in analyzing the Blogspace and retrieving information from them. This chapter would survey some of the papers about the analysis and ranking of blogs.

The chapter is organized in the following way. The analysis of the evolution of Blogpsace is covered in Section 2 followed by an algorithm for ranking Blogs in Section 3. The information diffusion analysis is covered in Section 4 followed by the conclusion.

11.2 Analysis of Evolution of Blogspace

The Blogspace can not be analyzed with the standard graph model, since a simple directed graph does not capture the link creation as point phenomenon in time. The first paper to analyze the evolution of Blogspace by Kumar et al, identified the need for modeling the Blogspace as an evolving directed graph and introduced a new tool called Time Graphs. The major contributions of this paper has been in finding the rapid growth of the Blogspace around the end of 2001 and in finding the growth of the number of active micro-communities, and going on to show that this is not a direct implication of the growth of the Blogspace itself.

The notion of communities in the web graph is generally characterized by a dense bipartite graph and almost all dense bipartite subgraphs was assumed to denote a community. But this idea does not extend well for Blogs, since they are heavily influenced by time, i.e, a given topic might start an intense debate leading to heavy hyperlinking and then fade away. Thus in Blogspace, communities are identified by not just the subgraphs but also a time interval. But there could also exist communities which exist all through because of a common interest. As an example of such temporal communities, a burst occurred when one member Firda, (http://www.wannabegirl.org) posts a series of daily poems about other bloggers in the community and includes link to their blogs. Thus burst occurs from March-April 2000.

11.2.1 Identifying Communities in Blogspace

The process of identifying communities is split into two sub-processes. The first task is to extract dense subgraphs from the blog graph and the second task is to perform burst analysis on each of the subgraphs. For this purpose, they define a novel data structure which takes the time of edge creations into account. A time graph G = (V, E) consists of a set of nodes, V, where each node $v \in V$ has an associated interval D (v) on the time axis (called the duration of v) . Each edge, $e \in E$, is a triple (u, v, t) where u and v are nodes in V and t is a point of time in the interval $D(u) \bigcap D(v)$. The prefix of G at time t, is also a time graph $G_t = (V_t, E_t)$ where $V_t = \{v \in V \mid D(v) \bigcap [0, t] \neq \phi\}$. Likewise $E_t = \{(u, v, t') \in E \mid t' \leq t\}$

As a preprocessing step in identifying the community subgraph, we consider the blog graph as an undirected graph and remove nodes that are linked-to by a large number of nodes. The intuition is that these nodes are too well-known for the type of communities they want to identify. The actual algorithm consists of two steps the pruning step and the expansion step. In the pruning step, vertices of degree zero and degree one are removed and vertices of degree two are checked if they participate in a complete graph of 3 vertices. If they do, then they are passed to the next step which is the expansion step. The expansion step is an iterative process. In each iteration, the vertex which contains the maximum number of edges to the current community is identified and added to the community if the number of edges is greater than t_k which is a threshold depending on the number of iterations. Thus, the expansion step gives a set of densely connected nodes which is the identified community.

The burst analysis is done by using the Kleinberg's burst analysis algorithm which is defined below. In the case of Blog graph evolution, the events are defined to be the arrival of an edge. In the Kleinberg algorithm, the generation of sequence of events is modeled using the automaton with two states, high and low. The inter-arrival time is independently distributed according to an exponential distribution where the parameter depends on the current state. There is a cost associated with a state transition and the optimal low cost state transition sequence that is likely to generate the the given stream.

The blog graph was obtained by crawling the data from seven popular blog sites: http://www.blogger.com, http://www.memepool.com, http://www.globeofblogs.com,
http://www.metafilter.com, http://blogs.salon.com, http://www.blogtree.com, and
Web_Logs subtree of Yahoo!.

Four features of the Blogspace are analyzed in this experiment. First the degree distribution is seen to be increasing along the y-axis with time but maintaining the same shape as time progresses. The in-degree and the out-degree distribution do not vary much with time. Both the distributions are shown in Fig 10.1. The next feature to be analyzed is connectivity, in particular the strongly connected component (SCC) . The graph for this is shown in Fig 10.2. The graph shows the fraction of nodes involved in the three biggest SCC's in the blog graph. The rise in the fraction of nodes involved shows that increasing number of nodes are involved in the SCC's or in other words, the whole blog graph is becoming increasingly more connected. Another interesting feature that they have analyzed is the community structure. They show that more and more nodes (blog posts or pages) are participating in at least one community. They do so by plotting the number of nodes participating in a community as time progresses. They also plot the fraction of nodes participating in some community along with time. Both these graphs are shown in Fig 10.3. Both the graphs clearly indicate that the blog graph is being structured into many communities.

The last feature that they analyze is the burstiness of the communities, they do this by plotting the number of communities in the high-state along with time. This graph is also clearly increasing

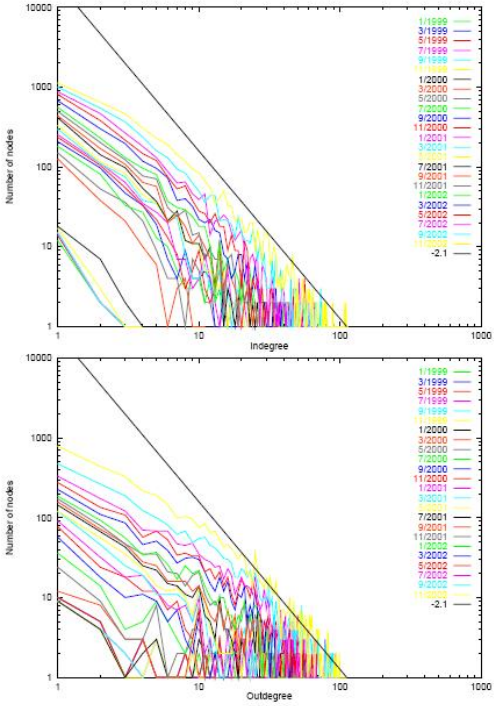


Figure 11.1: Degree distribution evolution in blogspace, Kumar et al, On the bursty evolution of blogspace, Proceedings of the 12th international conference on World Wide Web, May 20-24, 2003, Budapest, Hungary

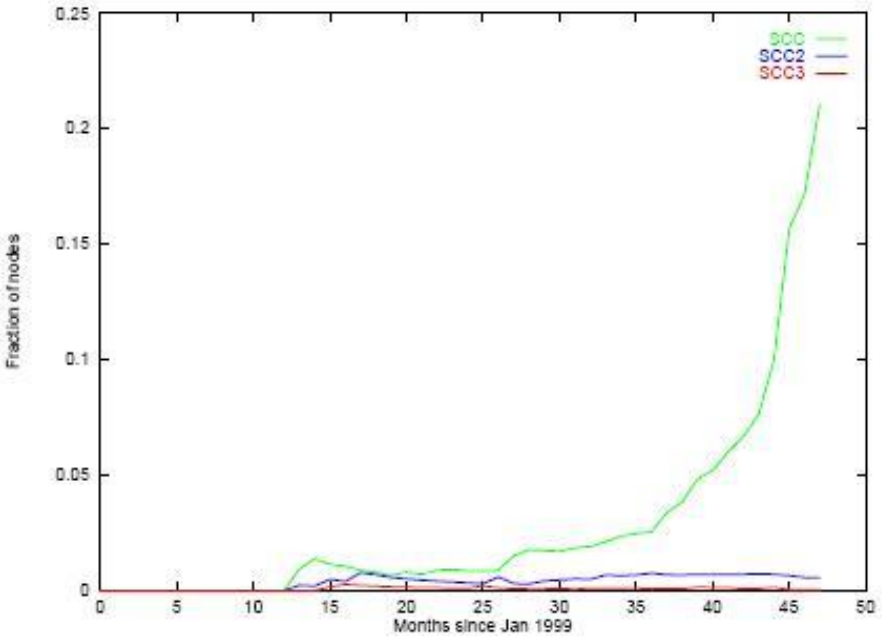


Figure 3: SCC evolution

Figure 11.2: Evolution of connectedness in Blogspace, Kumar et al, On the bursty evolution of blogspace, Proceedings of the 12th international conference on World Wide Web, May 20-24, 2003, Budapest, Hungary

indicating that more and more communities are becoming increasingly active. All the features synthesized together clearly shows that there has been a significant change in the amount of blogging during 2000-01. To see if these features are observed because of the sociology of the Blogspace, they plot the same features for a random graph generated using a time-evolving version of $Erdös - Rényi$ graphs. All the features show the same pattern for the random graph except the community structure graph, which shows that the blog graph being structured into communities is not a coincidence but indeed a result of the sociology of the Blogspace.

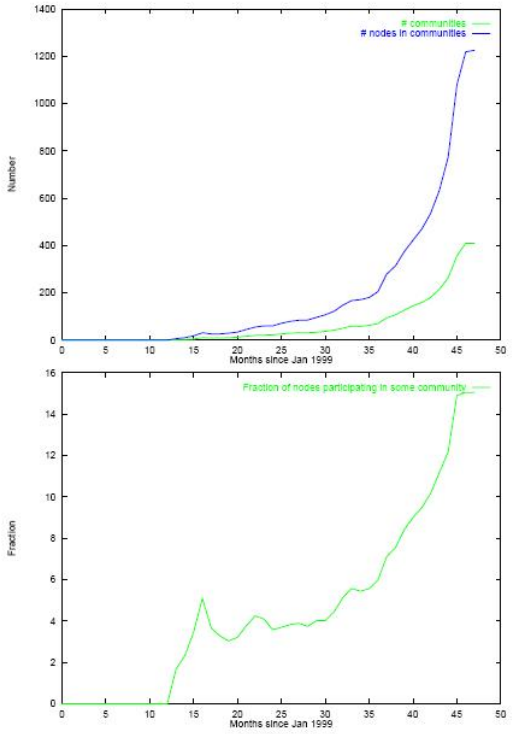


Figure 11.3: Evolution of community structure in Blogspace, Kumar et al, On the bursty evolution of blogspace, Proceedings of the 12th international conference on World Wide Web, May 20-24, 2003, Budapest, Hungary

11.3 Ranking of Blogs

The ranking algorithms used by search engines for web pages are not suited for ranking blog posts. The reason for that is the ranking algorithms used by search engines like google use the explicit link structure between the pages. Such an algorithm would fail in the case of ranking blog posts because bloggers do not post explicit links to the blogs where they read about an article, but instead post direct links to the URL, i.e, post links to the source directly. Thus what we really need is to track information flow and give credit to the blog which was the first to post about the article. This ranking algorithm would then be able to rank blog based on how important they are for for propagating information. To do this, we should infer links other than the explicit links present in the page, i.e, find out from which post another post could have possibly learnt of an URL before posting it. That does not mean we can leave out the explicit links present. Two types of explicit

link information available are the blogroll and the via links. The blogroll is the set of links that are found on the left side of the blog page. This is generally the set of links which the blogger visits often and they too visit the blog often. Via links are links to the blog which posted the article first. But unfortunately, these two types of links account for only 30% of the URLs mentioned. This further motivates the need to infer links other than those present to identify information spread in blogspace. To predict if there exists a link between two blogs through which information could have spread is predicted using a SVM classifier. The similarity function between two nodes is based on the following

1. blog_sim: The number of other blogs the two sites will have in common
2. link_sim: The number of non-blog links (URLs) shared by the two
3. text_sim: Textual similarity
4. Timing and repeating history information reflecting the frequency of one blog being affected before the other.

Both blog_sim and link_sim are calculated as vector-space cosine similarity measure that ranges between 0 (no overlap in URLs) and 1 (complete overlap in URLs) . The formula for computing the similarity is given below,

$$link\_sim(A, B) = \frac{||n_A \bigcup n_B||}{\sqrt{(}||n_A||) \times \sqrt{(}||n_B||)} \tag{11.1}$$

Textual similarity is calculated as the cosine similarity of term vectors weighted using TFIDF scheme. Timing and history information is obtained by answering the following question. Of the URLs mentioned in Blog A how many were mentioned in blog B a day earlier? Two SVM classifiers were constructed using these features. The first classifier classified the input into three classes (reciprocated links, one-way links, unlinked) The second classifier differentiated between linked and unlinked alone. The second classifier had a much better accuracy and hence was used in the final ranking algorithm.

Now we can describe the complete algorithm. The algorithm builds a graph of all the blogs as nodes and all the inferred links are weighted as follows. Suppose blog b_i cites a URL u, after blog b_j or on the same day, then the weights are assigned as ${w_{ji}}^u = w(\triangle d_{ji})$ where $d_{ji} = d_j - d_i$ where d_i is the day of which the URL u is posted in blog b_i and the weight function, w, is a decreasing function. This weight is normalized as follows,

$$\frac{{w_{ji}}^u}{n_j \sum_k {w_{jk}}^u} \tag{11.2}$$

After all the URLs are considered all the directed edges are merged together and all the weights of the same edge are summed. This graph is called the *implicit information flow graph*. Now PageRank is run on this graph to rank the blogs. The top 20 results obtained using this algorithm and PageRank alone are completely different which indicates that information flow is completely left out by Google search engine and other search engines which rely only on explicit link structure.

11.4 Conclusion

This chapter covered an extensive analysis of the blogspace and explained why using only the explicit link structure links are not good enough to rank blogs. It also explained how links between

blogs can be inferred through which information might have spread and use this information to better rank blogs.

Chapter 12

References

1. Dragomir Radev, Eduard Hovy, Kathleen McKeown, Introduction to the special issue on summarization, Computational Linguistics, December 2002, Volume 28, Issue 4, pages 399-408.

2. Gunes, Erkan, Dragomir Radev, LexRank: Graph-based Lexical Centrality as Salience in Text Summarization, JAIR, 2004.

3. Stergos D. Afantenos, Vangelis Karkaletsis, Panagiotis Stamatopoulos, Summarization from Medical Documents: A Survey, Artificial Intelligence in Medicine, Volume 33, Issue 2, February 2005, Pages 157-177

4. Dragomir Radev, Information Retrieval tutorial presented at the ACM SIGIR 2004, http://www.summarizati

5. Peter G. Doyle (Dartmouth), J. Laurie Snell (Dartmouth), Random Walks and Electric Networks, Categories: math.PR Probability Theory
http : *//front.math.ucdavis.edu/*0001.5057

6. Xiaojin Zhu, Semi-supervised learning literature survey. Technical Report 1530, Department of Computer Sciences, University of Wisconsin, Madison, 2005.
http : *//www.cs.wisc.edu/ jerryzhu/pub/ssl_survey.pdf*

7. Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani, Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. In ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining, 2003.
http : *//pages.cs.wisc.edu/ jerryzhu/pub/zglactive.pdf*

8. Gunes Erkan, Graph-based Semi-Supervised Learning Using Harmonic Functions
http : *//tangra.si.umich.edu/ radev*/767/*gunes*2*.pdf*

9. Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co- training. In Proceedings of the 11th Annual Conference on Computational Learning Theory, pages 19-26, 2001. Morgan Kaufmann.

10. Thorsten Joachims. Transductive learning via spectral graph partitioning. In Tom Fawcett and Nina Mishra, editors, Proceedings of the 20th International Conference on Machine Learning, pages 290-297, 2003. AAAI Press.

11. David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In Proceedings of the 33rd Meeting of the Association for Computational Linguistics, pages 189-196, 1995. Morgan Kaufmann.

12. David Yarowsky. Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In Proceedings of the 32nd Meeting of the Association for Computational Linguistics, pages 88-95, 1994. Morgan Kaufmann.

13. Hema Raghavan, Omid Madani and Rosie Jones, "Active Learning on Both Features and Documents" Journal of Machine Learning Research (JMLR), 7(Aug):1655- 1686, 2006

14. M. E. J. Newman. The structure and function of complex networks. SIAM Review, 45(2):167–256, 2003.

15. M. E. J. Newman. Power Laws, Pareto distributions and Zipf's law, Contemporary Physics, Vol 46, Issue 5, P.323-351

16. M. E. J. Newman. Finding community structure in networks using the eigenvalues of matrices, Physical Review E, Vol 74, 2006.

17. Donald Hindle, Mats Rooth, Structural Ambiguity and Lexical Relations, Meeting of the Association for Computational Linguistics, 1993

18. Adwait Ratnaparkhi: Statistical Models for Unsupervised Prepositional Phrase Attachment CoRR cmp-lg/9807011: (1998) Adwait Ratnaparkhi, Jeffrey C. Reynar, Salim Roukos: A Maximum Entropy Model for Prepositional Phrase Attachment. HLT 1994

19. Kristina Toutanova, Christopher D. Manning, and Andrew Y. Ng. 2004 .Learning Random Walk Models for Inducing Word Dependency Distributions. In Proceedings of ICML 2004.

20. Eric Brill, Philip Resnik: A Rule-Based Approach to Prepositional Phrase Attachment Disambiguation. COLING 1994: 1198-1204

21. Michael Collins, James Brooks, Prepositional Phrase Attachment through a Backed-Off Model, 1995, Third workshop on large corpora

22. Shaojun Zhao and Dekang Lin. 2004. A Nearest-Neighbor Method for Resolving PP-Attachment Ambiguity. In Proceedings of IJCNLP-04. for Resolving PP-Attachment Ambiguity

23. http://dp.esslli07.googlepages.com/

24. Nivre, J. Dependency Grammar and Dependency Parsing. MSI report 05133. *Växjö* University: School of Mathematics and Systems Engineerin, 2005.

25. Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kbler, S.,Marinov, S. and Marsi, E. MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, 13(2), 95-135, 2007.

26. McDonald, R., Pereira, F., Crammer, K., and Lerman, K. Global Inference and Learning Algorithms for Multi-Lingual Dependency Parsing. Unpublished manuscript, 2007.

27. McDonald, R., and Nivre, J. Characterizing the Errors of Data-Driven Dependency Parsing Models. Proceedings of the Conference on Empirical Methods in Natural Language Processing and Natural Language Learning, 2007.

28. Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, Paul S. Roossin, A statistical approach to machine translation, Computational Linguistics, v.16 n.2, p.79-85, June 1990

29. Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, Robert L. Mercer, The mathematics of statistical machine translation: parameter estimation, Computational Linguistics, v.19 n.2, June 1993

30. Yaser Al-Onaizan, Jan Curin, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz-Josef Och, David Purdy, Noah A. Smith, and David Yarowsky. Statistical machine translation: Final report. Technical report, The Center for Language and Speech Processing, John Hopkins University, Baltimore, Maryland, USA, September 1999.

31. Knight, Kevin. 1999b. A Statistical MT Tutorial Workbook.
Available at http://www.isi.edu/natural-language/ mt/wkbk.rtf.

32. Franz Josef Och. Minimum error rate training in statistical machine translation. In Proceedings of the 41st Meeting of the Association for Computational Linguistics, pages 160-167, 2003. MIT Press.

33. Franz Josef Och. GIZA++: Training of statistical translation models, 2000.

34. Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In Proceedings of the 40th Meeting of the Association for Computational Linguistics, pages 295-302, 2002. MIT Press.

35. Franz Josef Och and Hermann Ney. The alignment template approach to statistical machine translation. Computational Linguistics, 30(4):417-449, December 2004.

36. Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast decoding and optimal decoding for machine translation. In Proceedings of the 39th Meeting of the Association for Computational Linguistics, pages 228-235, 2001. MIT Press.

37. Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. A smorgasbord of features for statistical machine translation. In Proceedings of the Human Language Technology Conference.North American chapter of the Association for Computational Linguistics Annual Meeting, pages 161-168, 2004. MIT Press.

38. Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In Proceedings of the 39th Meeting of the Association for Computational Linguistics, pages 523-530, 2001. MIT Press.

39. Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics Annual Meeting, 2003. MIT Press.

40. Kenji Yamada and Kevin Knight. A decoder for syntax-based statistical machine translation. In Proceedings of the 40th Meeting of the Association for Computational Linguistics, pages 303-310, 2002. MIT Press.

41. Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based language models for statistical machine translation. In Proceedings of the 8th International Workshop on Parsing Technologies (IWPT '03), April 23-25, 2003.

42. Bo Pang , Lillian Lee, A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts, Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, p.271-es, July 21-26, 2004, Barcelona, Spain

43. Andrew Goldberg and Xiaojin Jerry Zhu, Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization, HLT-NAACL 2006 workshop - Textgraphs: Graph-based Algorithms for Natural Language Processing.

44. M. Hurst and K. Nigam. Retrieving topical sentiments from online document collections. In Document Recognition and Retrieval XI, pages 27–34, 2004.

45. Bo Pang, Lillian Lee, Shivakumar Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning Techniques, Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)

46. Adar, E., Zhang, L., Adamic, L., and Lukose, R. Implicit structure and the dynamics of blogspace. Presented at the Workshop on the Weblogging Ecosystem at the 13th International World Wide Web Conference (New York, May 18, 2004)

47. Daniel Gruhl , R. Guha, David Liben-Nowell, Andrew Tomkins, Information diffusion through blogspace, Proceedings of the 13th international conference on World Wide Web, May 17-20, 2004, New York, NY, USA

48. Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, Andrew Tomkins, On the bursty evolution of blogspace, Proceedings of the 12th international conference on World Wide Web, May 20-24, 2003, Budapest, Hungary