Masterarbeit

Master Annotator: Enhancing Distant Supervision through Visual Interfaces

Zheren Luo

Course of Study: Softwaretechnik

Examiner: Prof. Dr. Thomas Ertl
Supervisor: Dr. Dennis Thom, Dipl.-Phys. Qi Han

Commenced: October 10, 2017
Completed: April 10, 2018
Abstract

The rise of social networking sites and applications such as Twitter, Facebook, Instagram, Weibo in couple years provides new ways of information sharing on the Internet. Twitter is one of the most popular micro-blogging services. Recently, tweet classification has received much attention. In tweet classification, there is a demand for massive labeled training examples. However, the labeling cost can be high since the resources regarding available time, domain expert annotators are often limited. Distant supervision is a feasible solution but has some shortages such as noisy data, skewed annotations, and the challenge to generate negative training examples. This thesis tries to overcome these difficulties and shortages by combining visual analytics. Based on the distant supervision theorem, information such as keywords, hashtags, user_mentions, and URLs existing in tweets can be used as annotators to generate training examples. Through an interactive visual interface, the generated training examples can be checked and modified, which enhances the distant supervision. An iterative analytical loop is established using the interactive visual interface and the enhanced distant supervision. It allows users to make deep exploration of tweet data. Users can get tweets according to their needs, find potential topics, label or relabel tweets, train classifiers. A software is implemented to achieve the analytical loop. A case study is conducted to demonstrate the performance and usage of the approach and the software.
# Contents

1 Introduction 13  
1.1 Motivation .......................... 13  
1.2 Goal ................................ 14  
1.3 Overview ................................ 14  

2 Background 15  
2.1 Basic ................................ 15  
2.2 Related Works .......................... 22  

3 Approach 25  
3.1 Requirements .......................... 25  
3.2 Specification ................................ 26  
3.3 Corpus Selection & Preprocessing .......................... 26  
3.4 Corpus Query .......................... 29  
3.5 Tree Generation .......................... 30  
3.6 Data Visualization .......................... 31  
3.7 Data Exploration and Labeling .......................... 36  
3.8 Classifier Training .......................... 37  

4 Implementation 41  
4.1 Data Index and Query .......................... 42  
4.2 User Interface .......................... 43  
4.3 Basic Exploration Tools .......................... 48  
4.4 Libraries .......................... 49  

5 Case Study 53  
5.1 Session Report .......................... 53  
5.2 Discussion .......................... 59  

6 Conclusion and Outlook 61  

Bibliography 63
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Process model of visual analytics</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of an ordinary tree</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of a specific tree</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of slice and dice treemap</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Analytical loop of the approach</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Search view in the user interface</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>UML diagram of the generated tree structure</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>Main view in the user interface</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>Example of a complete binary tree</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Treemap Generation Process</td>
<td>33</td>
</tr>
<tr>
<td>3.7</td>
<td>Example of a treemap that has been split many times</td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Example of a treemap with tweet points</td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>Example of Tag Cloud and Content Lens</td>
<td>36</td>
</tr>
<tr>
<td>3.10</td>
<td>Example of the treemap after classification</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>Search view of the software UI</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>Main view of the software UI</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Example of t-SNE</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of using post date coordinate</td>
<td>47</td>
</tr>
<tr>
<td>4.5</td>
<td>Example of tweet nodes after classification</td>
<td>47</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of Tag Cloud and Content Lens</td>
<td>49</td>
</tr>
<tr>
<td>4.7</td>
<td>Example of Node Detail Report</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>Lucene Index</td>
<td>54</td>
</tr>
<tr>
<td>5.2</td>
<td>Initial query result of the case study</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>Result of using post date coordinate</td>
<td>56</td>
</tr>
<tr>
<td>5.4</td>
<td>Result of classification</td>
<td>57</td>
</tr>
<tr>
<td>5.5</td>
<td>Screenshot of the main view after pressing Explore</td>
<td>57</td>
</tr>
<tr>
<td>5.6</td>
<td>Screenshot of the main view after using t-SNE coordinates</td>
<td>58</td>
</tr>
<tr>
<td>5.7</td>
<td>Result of “building OR house” query and classification</td>
<td>58</td>
</tr>
<tr>
<td>5.8</td>
<td>Screenshot of the user interface after making Geographic Radius Query</td>
<td>59</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Original Tweet Structure .............................................. 26
3.2 Preprocessed Tweet Structure ........................................ 29
3.3 A Tweet Instance ....................................................... 38
4.1 Queries and Fields ..................................................... 42
List of Algorithms

2.1 PCA algorithm ............................................................. 20
1 Introduction

1.1 Motivation

The rise of social networking sites and applications such as Twitter, Facebook, Instagram, Weibo in couple years provides new ways of information sharing on the Internet. Among those social networking services, Twitter\(^1\) is one of the most popular micro-blogging services. According to the statistics\(^2\), until January 2018, Twitter has 330 million active users. Twitter users post status messages which are called “tweets” to discuss current hot events and news, make comments on products, and express their thoughts and feelings. Hundreds of millions of tweets are posted on Twitter every day, and a wide range of topics are discussed. However, it is not easy to extract tweets on specific topics from the massive volume of tweets.

Tweet classification which is a subcategory of text classification has received much attention recently. It is a process of classifying tweets into one or more predefined categories. Same as other text classification problems, a high number of labeled training documents are always needed to properly cover the feature space. However, the labeling cost can be high since the resources in terms of available time, domain expert annotators, or historic ground-truth data are often limited.

Distant supervision\(^{[MBSJ09]}\) is a feasible method to overcome the limitation. The core idea of distant supervision is to facilitate an already existing data source that exists in the public Internet, such as social networks, traffic sensors, or microblog platforms, as a basis to generate large volumes of necessary training data. To this end, the contained information in tweets such as hashtags, user_mentions, URLs, geographic locations and post dates can be used to automatically derive most likely annotations in larger numbers. However, previous approaches in automated distant supervision have often suffered from noisy data, skewed annotations, and the challenge to generate negative training examples.

This work tries to overcome the difficulty of generating a large number of training examples and the shortages of distance supervision. It looks into the feasibility of combining the idea of visual analytics.

Visual analytics\(^{[KMS+08]}\) is an effective method that combines automated analysis techniques with interactive visualizations. Visual analytics methods adopt machines to facilitate methods like knowledge discovery in databases, mathematics, and statistics to do automatic analysis while human decision makers can receive, understand data, define data relationship, and make conclusions or decisions.

\(^1\)https://twitter.com/
\(^2\)https://www.omicoreagency.com/twitter-statistics/
1 Introduction

1.2 Goal

This work investigates how the distant supervision can be enhanced by combining the idea of visual analytics and helps users to make in-depth explorations of tweet data. For this purpose, an iterative analytical loop of extracting tweet data, visualization, labeling tweets, doing tweet classification, and evaluating labeling and classification performance is established. This approach contains a data preprocessing function for users to organize collected tweet data, power filtering functions for users to get tweets according to their specific needs, and a classifier training function for users to train classifiers. Most importantly, this approach provides a visual interface which employ a highly interactive treemap. The treemap visualizes results of user operations including making queries, labeling or relabeling tweets, and training classifiers. This approach also has several tools that help users understand and operate tweets. Users can train classifiers with labeled tweets. A software application is implemented to evaluate this approach, and a case study is conducted to demonstrate the performance and usage of the approach and the software.

1.3 Overview

This thesis consists of six chapters and is organized in following way:

1. Chapter 2 - Background This chapter introduces required fundamentals, cited works and recently published related works.

2. Chapter 3 - Approach This chapter presents the approach introduced in this thesis, including its design requirements and specification.

3. Chapter 4 - Implementation This chapter describes implementation details of the software which is implemented to achieve the approach.

4. Chapter 5 - Case Study A case study is conducted to evaluate the performance of the approach and the software. This chapter introduces the process and results of the case study.

5. Chapter 6 - Conclusion and Outlook This chapter concludes this work and discusses possible future works.
2 Background

This chapter introduces the background of this work. It has two sections: basics and related works. The basics section introduces the theoretical knowledge used in this work, which is also the preconditions to understand this thesis. In the second section, research that has been referred or has similar research directions as this work are presented.

2.1 Basic

2.1.1 Visual Analytics

The rapid development of information techniques brings human the ability to collect vast amount data. Meanwhile, large data size is usually accompanied by high complexity and variety. Dealing with such large size data is a challenge. Visual analytics is a possible and efficient solution to this problem. Its main idea is to improve analytical reasoning through interactive visual interfaces, in other words, combine the strength of human and machine. In 2004, Pak Chung Wong and J. Thomas[WT09] introduced the concept of visual analytics. Visual analytics is an effective method that combines the art of human intuition and the science of mathematical deduction. Another more detailed definition is a provided by D.A.Keim et al.[KMS+08], visual analytics can be seen as an iterative process that relates to information collection, data preprocessing, knowledge representation, interaction and decision making. Visual analytics methods adopt machines to facilitate methods like knowledge discovery in databases, mathematics, and statistics to do automatic analysis while human decision makers can receive, understand data, define data relationship, and make conclusions or decisions.

Figure 2.1 shows the process model of visual analytics. Visual analytics is different from information visualization. Information visualization aims at change how analysts work with data, and it helps them respond to problems faster. It recognizes trends, patterns, and contexts that are usually difficult to recognize in text-based data and provides interaction techniques to help users answer their questions about data. There is a certain amount of overlapping between information visualization and visual analytics as they both represent data in an easily understandable way through visual interfaces. The difference is that visual analytics combines automated analysis techniques such as data mining with interactive visualizations.

2.1.2 Distant Supervision

Distant supervision is a relation extraction method first proposed by Mintz[MBSJ09]. Distant supervision holds the view that any sentence that has two terms that appear in a known Freebase relation is likely to express that relation somehow. Distant supervision is widely used for generating
2 Background

Figure 2.1: Process model of visual analytics

training sets for machine learning. The traditional method of producing training sets is to use human annotators to label the given corpus. When the size of the corpus is large, the labeling process is costly and time-consuming. The usage of distant supervision in this work slightly different from the usage of Mintz. In this work, distant supervision is defined as a method to facilitate an already existing data source that exists in the public Internet, such as social networks, traffic sensors, or microblog platforms, as a basis to generate large volumes of necessary training data. To this end, associated metadata, such as folksonomies, user comments, or network relationships, can be used to automatically derive most likely annotations in larger numbers. In 2009, Go, Bhayani, and Huang[GBH09] used the Twitter messages with emoticons as the training data, the emoticons occurring in tweets indicate the messages are positive or negative. This work uses a similar method with theirs.

2.1.3 Machine Learning

Machine learning is a field of computer science that involves probability theory, statistics, convex analysis, approximation theory, and many other disciplines. It specializes in how computers simulate or realize human learning behaviors to acquire new knowledge or skills and to reorganize existing knowledge structures to continuously improve their performance. Machine learning is the core of artificial intelligence, which is to make the computer have the fundamental way of intelligence. Its applications are throughout all areas of artificial intelligence. Traditionally, computers accept a

\[1\text{http://users.jyu.fi/~pejowart/lectures/ties445/va_lecture_pw.pdf} \]
basic bunch of instructions and execute the instructions line by line. In contrast to the traditional way, machine learning does not accept the instructions. Instead, it accepts only the entered data. That is, it has a sense of human ability to handle things.

According to the different training methods, machine learning can be divided into different categories:

1. **Supervised learning:** It does training with the known examples of correct answers, that is, with labeled data. It can be further separated into regression and classification problems. This work deals with the classification problem.

2. **Unsupervised learning:** It uses unlabeled data as input, which means a machine does not know how the output data corresponds to the input data. Unsupervised machine learning reads the data and finds their own data models and rules. Typical supervised machine learning problems are clustering(similar data grouped together) and anomaly detection.

There is also semi-supervised learning. It is not related to this work and is not discussed here.

### 2.1.4 TF-IDF

TF-IDF (term frequency-inverse document frequency) is a commonly used weighting technique in information retrieval and data mining. It measures the importance of a word to a document in a collection. The importance increases proportionally to the number of times a word appears in the document and decrease proportionally to the number of documents contain the word in the collection. In this work, TF-IDF is used as the term weight for training classifier and generating t-SNE coordinates.

Obviously, TF-IDF consists of two parts. The first part Term Frequency(TF) is the number of times a word appears in a document, divided by the total number of words in the document. The second part Inverse Document Frequency(IDF) is the logarithm of the number of documents in the collection divided by the number of documents contain the word. There are different weighting schemes, the one used in this work is given in the following:

\[
tfidf(t, d, D) = \frac{tf(t, d) \cdot idf(t, D)}{\sum_{t' \in d} tf(t', d)} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \log \frac{N}{n_t}
\]

- **Term Frequency(TF):** measures how often a term appears in a document. A term may appear much more time in a longer document than a shorter document. In order to get rid of possible effect, adjustments are often used. The number of a term in a document is divided by the document length(the total number of terms in the document).

\[
TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}
\]

- **Inverse document frequency(IDF):** measures the importance of a term. Terms such as "the", "I" may frequently appear in any documents. They are assigned with high term frequencies although they contain little information. The inverse document frequency decreases the weights of terms that appear very frequently in the documents while increasing the weights of terms that appear hardly.

\[
IDF(t) = \log_e \frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}
\]
2.1.5 TreeMap

A tree is a collection of nodes. This collection can be empty, when it is not empty, it contains a root node \( r \) and zero or more non-empty subtrees \( T_1, T_2, ..., T_k \). The root of each subtree is connected by a directional edge from the root \( r \).

![Diagram](image1)

**Figure 2.2:** Example of an ordinary tree

The root of each subtree is the child of the root \( r \) and \( r \) is the parent of all subtree roots. A tree has \( N \) nodes and \( N - 1 \) edges, one of the nodes is the root. Each edge connects a node to its parent. Apart from the root, every node has a parent.

![Diagram](image2)

**Figure 2.3:** Example of a specific tree

The tree structure used in this work is the complete binary tree, it is a particular kind of tree that each node has and only two children. The tree structure can be visually represented in different ways such as node-link (which is used in Figure 2.2 and Figure 2.3), stacking, nesting, and indentation. In 1991, Johnson and Schneiderman proposed a visualization technique called Treemap[JS91]. It is a method which makes full use of available display space, mapping the full hierarchy onto a rectangle in a space-filling manner. Several different types of treemap are proposed in their work. This work uses the simple *slice and dice* layout mentioned in it.

A slice and dice treemap is generated as follows:

1. use parallel lines to divided a rectangle representing an item into smaller rectangles representing its children.

2. each child is allocated a space whose area size is proportional to some property and can be additionally encoded by color.

3. at each level of the hierarchy, switch the direction of the lines vertically or horizontally.
2.1 Basic

Figure 2.4: Example of slice and dice treemap - Generation of a slice and dice treemap for the tree shown in Figure 2.3. (a) A rectangle represents the tree root. (b) Horizontal splitting for representing tree nodes with depth one. (c) Vertical splitting for representing tree nodes with depth two.

2.1.6 Dimension Reduction

Dimension reduction belongs to unsupervised machine learning. Users may wish to reduce the number of dimensions to decrease the size of data-held memory, speed up algorithm operations, visualize data, and more. In many fields of research and applications, it is often necessary to conduct a large number of observations on many variables that reflect things and collect large amounts of data for analysis to find potential rules. A large number of multivariate samples undoubtedly provides abundant information for research and applications, but also increases the workload of data collection to a certain degree. More importantly, in most cases, there may be some correlation between many variables, which increases the complexity and inconvenience of analysis. If users analyze each variable separately, the analysis is isolated rather than synthetic. Blindly reducing the variables loses a lot of information and leads to wrong conclusions.

Therefore, there is a need to find a reasonable way to reduce the loss of information contained in the original variables while reducing the number of variables so as to achieve the purpose of conducting a comprehensive analysis of the data collected. Due to the existence of certain correlations among the variables, it is possible to synthesize various types of information that exist in the variables with a small number of synthetic variables. Principal component analysis belongs to this kind of dimension reduction methods.

PCA

Principal Component Analysis (PCA) [KPF01] is a statistical method. Through orthogonal transformation, a set of variables that may have correlations are transformed into a new set of linearly uncorrelated variables. The resulted variables are called principal components.

The principal component analysis tries to recombine the original variables into a new set of independent synthetic variables, and take out several synthetic variables to reflect as much information of the original variables as possible according to the actual needs. It tries to replace a large number of related variables (such as P variables) with a set of new unrelated synthetic variables. The usual mathematical treatment is to linearly combine the original P variables as several
new synthetic variables. The most classic way is to use $F_1$ (the first selected linear combination which is also the first synthetic variable) variance, that is, $\text{Var}(F_1)$. The greater $\text{Var}(F_1)$ is, the more information $F_1$ contains. In all selected linear combinations, $F_1$ should have the largest variance, and it is called the first principal component. If the first principal component is not enough to represent the information of the original $P$ variables, select a second linear combination $F_2$. The existed information in $F_1$ should not appear again in $F_2$. In mathematics, the expression is $\text{Cov}(F_1, F_2) = 0$, $F_2$ is called the second principal component, and so on can construct the third, fourth, ..., the $P$th principal component. The algorithm is given in the following:

**Algorithm 2.1 PCA algorithm**

**Input:** Dataset $D$ with dimension $K$: $D = \{x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(m)}\}$

1. Normalize all samples: $x^{(i)}_j \leftarrow \frac{x^{(i)}_j - \mu_j}{\sigma_j}$

2. Calculate covariance: $\sum = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})^T \rightarrow \sigma = \frac{1}{m} X' \cdot X$

3. Decompose the singular value of covariance: $[U, S, V] = \text{svd}($Sigma$)$

4. Take the $k$ eigenvectors corresponding to first $k$ eigenvalues: $u^{(1)}, u^{(2)}, u^{(3)}, ..., u^{(k)} \rightarrow U_{reduce} = U(:, 1 : k)$

5. $Z = U_{reduce} \cdot x$, where $x$ is a $n$-dimensional column vector, excluding $x_0 = 1$.

**Output:** Projection matrix: $U = \{u^{(1)}, u^{(2)}, u^{(3)}, ..., u^{(k)}\}$

**t-SNE**

t-SNE($t$-Distributed Stochastic Neighbor Embedding)[MH08] is an algorithm derived from SNE($Stochastic Neighbor Embedding$)[HR03] which was first proposed in 2002. SNE tries to map a list high dimensional samples into a list of lower dimensional samples while keeping the similarity of samples. In other algorithms, the similarity is usually measured by Euclidean distance. SNE transforms the similarity into conditional probability. It considers the distribution of samples in both high dimensional and low dimensional spaces as Gaussian distribution.

Given high dimensional data $X$ and low dimensional data $Y$, $x_i$ is the $i$-th sample. The distribution probability matrix $P$ in high dimensional space can be defined as follows:

$$p_{ji} = \frac{\text{exp}\left(- \frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \text{exp}\left(- \frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}$$

$p_{ji}$ denotes the conditional probability of $x_i$ selects $x_j$ as its neighbor. $\sigma_i$ represents the variance of Gaussian distribution centered on $x_i$. Since only the similarity of different pair of samples are calculated, $p_{ii} = 0$. 

20
2.1 Basic

$y_i$ and $y_j$ are mapping samples of $x_i$ and $x_j$. The variances of Gaussian distribution in both dimension spaces are set to $1/\sqrt{2}$. The distribution probability matrix $Q$ in low dimensional space is calculated as follows:

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}$$

$q_{ij}$ denotes the conditional probability of $y_i$ selects $y_j$ as its neighbor. $q_{ij} = 0$. If $y_i$ and $y_j$ truly reflect the relationship between high dimensional $x_i$ and $x_j$, the conditional probabilities $p_{ij}$ and $q_{ij}$ should be exactly equal. Up to now, only the conditional probability between $x_i$ and $x_j$ are considered. If the conditional probabilities between $x_i$ and all other samples are taken into consideration, a conditional probability distribution $P_i$ in high dimension space can be formed. Similarly, there exists a conditional probability distribution $Q_i$ in low dimensional space and should be consistent with $P_i$. Then, KL-distance (Kullback-Leibler Divergence) is used to measure the similarity between $P_i$ and $Q_i$, which is also the objective (cost) function:

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j P_{ij} \log \frac{P_{ij}}{Q_{ij}}$$

The final objective of SNE is to minimize the KL-distance for all samples, which can be solved by gradient descent algorithm:

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{ij} - q_{ij}) (y_i - y_j)$$

In SNE, $p_{ij}$ is not equal to $p_{ji}$ in high dimensional space, and $q_{ij}$ is not equal to $q_{ji}$ in low dimensional space. Symmetric SNE uses a more general and reasonable joint probability distribution. It constructs joint probability distribution matrices $P$ and $Q$ in high and low dimensional spaces respectively, the matrices are symmetric and can be easily operated. For any $i$ and $j$, there are $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$. Thus, the definition of $p_{ij}$, $q_{ij}$ and cost function can be changed into:

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2/2\sigma^2)}$$

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

Compared to the formula of SNE, this gradient is more simplified and computationally efficient.

SNE has a set of limitations. One of them is the crowding problem. In the visualization of SNE, different types of clusters crowd together and can not be distinguished, which is due to the difference between the distribution of high and low dimensional spatial distances. Symmetric SNE solves the asymmetry problem in SNE and generates a simpler gradient formula, but it can not solve the crowding problem.
2 Background

t-SNE uses a symmetric SNE for the distributions in high dimensional. The distribution in low dimensional space uses a more general t-distribution which is also symmetric. The benefit of using t-distribution is to make the distance between the clusters wider, thus solving the congestion problem. The $q_{ij}$ and gradient of t-SNE are given in the following:

$$q_{ij} = \frac{(1 + \| y_i - y_j \|^2)^{-1}}{\sum_k (1 + \| y_i - y_k \|^2)^{-1}}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \| y_i - y_j \|^2)^{-1}$$

2.2 Related Works

2.2.1 Distant Supervision

In 2009, Go, Bhayani and Huang[GBH09] first adopted distant supervision in tweet sentiment classification. They used the Twitter messages with emoticons as the training data. The emoticons occurring in tweets indicate the messages are positive or negative. Their approach reminds us that other information contained in tweets can also be used as annotators, such as hashtags, URLs, user_mentions, and geolocation.

Magdy et al.[MSES15] introduced an approach of using the labels of YouTube videos as the labels of tweets that link to these videos. A Zubiaga and H Ji[ZJ13] used human-edited web page directories to deduce categories of URLs contained in tweets. The categories of URLs can be used as annotators to label a large number of tweets. Both of this two approaches achieved good performances. In this work, the URLs in the tweets are also used as annotators for labeling.

The above research shows that it is feasible to obtain training samples through the distance supervision and good results can be obtained. However, some research shows the shortages of distant supervision. Hoffmann et al.[HZW10] conducted a study of generating training data using distant supervision. They pointed out that the generated data of distant supervision is noisy and sparse, but they did not provide solutions for this problem. Takamatsu et al. [TSN12] proposed a method to reduce the number of wrong labels in distant supervision. A generative model which models the heuristic labeling process of distant supervision directly was presented and a negative pattern list was built to remove wrong labels. Since distant supervision is a feasible way to generate training data, it is meaningful to study how to overcome the drawbacks.

2.2.2 Visual Analytics

Heimerl et al.[HKBE12] designed software called Visual Classifier Building Tool that enhanced information retrieval by the interactive creation of binary classifiers. In their approach, documents were visualized in a visual interface. Users can explore the document contents, label documents, and train classifiers. The approach discussed in this work has a similar classifier training process with them.
2.2 Related Works

H. Bosch and D. Thom[BTTH+13] proposed an approach ScatterBlogs2. It was a system for microblog analysis with an interactive visual interface. Using ScatterBlogs2, an analyst can monitor and analyze microblog message by building task-tailored message filters. In their approach, tweets were visualized on a world map, LDA topic extraction was implemented to find potential events and monitor the ongoing events. They also combined classifier training into their method.

In 2010, Diakopoulos et al.[DNK10] designed a visual analytic tool called Vox Civitas to help journalists, and media experts extract news from large-scale aggregations of social media content around broadcast events. Kraft et al.[KWD+13] designed a tool called GTAC(Geo and Temporal Association Creator) that created a near real-time analysis environment for identifying event structures. Their tool extracted structured representations of events from the Twitter stream and supported event-level investigative analysis of social media data by interactively visualizing the event indicators.

2.2.3 Tweet Classification

Thom Dennis and Ertl Thomas[TE15] proposed an approach to tackle the shortages of searching engine for microblog entries such as Tweet Search and the challenge of explorative query optimization. They designed an analysis loop for microdocument retrieval. Their method employed hierarchical clustering and a highly interactive treemap to represent summarized clusters. This work is developed based their approach.

Another approach that provided interactive discovery of emergent theme in tweets was proposed by Hoeber et al.[HHE+16]. They combined automatic data processing and machine learning methods with an interactive software called Vista(Visual Twitter Analytics). Their software visualized the temporally changing sentiment and showing the top words, hashtags, user_mentions. This work has a similar idea with their approach.

There are massive studies related to distant supervision, visual analytics, and tweet classification. However, few studies that involve enhancing distant supervision with visual analytics are found. This work tries to verify the feasibility of enhancing distant supervision with visual analytics.
3 Approach

The objective of this work is to enhance distant supervision with visual interfaces and help users to make in-depth explorations of tweet data. For this purpose, an approach which combines the ideas of distant supervision and visual analytics is built. Human ability is included in the approach, a visual interface and a set of back-end data processing methods are provided for users to analyze and operate tweet data. This chapter first outlines the requirements of the approach, then introduces the approach in detail.

3.1 Requirements

To achieve the objective, a set of requirements are defined for the approach:

1. Users can process a large number of tweets and build local tweet database.
2. Users can get tweets using the information existing in tweets from the local tweet database.
3. It provides visual interfaces to show the effect of each user operation.
4. Users can observe and label the acquired tweets.
5. Users can discover the potential topics in tweets and do further exploration.
6. Users can watch the content of each single tweet.
7. Users can label one or more tweets at a time.
8. If there are mislabeled tweets, users can relabel them.
9. It can show the relationship or similarity between tweets.
10. Users can do tweet classification.
11. Users can train classifiers with labeled tweets.
12. Users can evaluate the performance of classifiers.
13. Users can save the trained classifiers locally for future use.
14. Users can use the trained classifier to get more required tweets.
3 Approach

3.2 Specification

According to above requirements, the approach is designed. It is composed of five primary functions: Corpus Selection & Preprocessing, Corpus Query, Tree Generation, Data Exploration & Labeling, and Classifier Training. These five functions form an analytical loop that allows users to explore and process tweet data iteratively. The flow chart in Figure 3.1 indicates the workflow of this analytical loop. The following sections introduce these functions in detail.

3.3 Corpus Selection & Preprocessing

3.3.1 Corpus Selection

Same as other text classification problems, the first step is corpus collection. It is necessary to collect a large number of tweets in order to cover all possible features. This work selects about two million tweets. The original structure of the selected tweets is given in Table 3.1.

<table>
<thead>
<tr>
<th>Field</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>tweet_id</td>
<td>123456789123456789</td>
</tr>
<tr>
<td>user_id</td>
<td>123456</td>
</tr>
<tr>
<td>created_at</td>
<td>201108231100</td>
</tr>
<tr>
<td>latitude</td>
<td>40.7306100000</td>
</tr>
<tr>
<td>longitude</td>
<td>-73.9352420000</td>
</tr>
<tr>
<td>place</td>
<td>New York, NY</td>
</tr>
<tr>
<td>text</td>
<td>@somebody #something, hello, http://...</td>
</tr>
</tbody>
</table>

Table 3.1: Original Tweet Structure - The structure of the tweets collected by Institut für Visualisierung und Interaktive Systeme (VIS) of Universität Stuttgart and were stored as CSV files. Each tweet has seven fields: tweet_id, user_id, created_at, latitude, longitude, place and text.

3.3.2 Preprocessing

The original structure of the collected tweets has to be transformed into a proper structure for further processing. Tweets are preprocessed for three reasons. Firstly, to facilitate distant supervision, different information existing in tweets including keywords, hashtags, user_mentions, and URLs and so on is used as annotators. Thus, this information needs to be stored separately, in order to be queried and got easily. Secondly, the original tweet texts cannot be used to train classifiers directly. The input of classifier training is feature vectors that are a set of words weighted by frequency
### 3.3 Corpus Selection & Preprocessing

**Figure 3.1:** Analytical loop of the approach - Interactive elements are in green, data extraction and processing elements are in blue. After selecting, preprocessing, and storing a certain amount of tweets, users can query the tweets hierarchically. The query structure and results are visualized as a treemap in the visual interface. Based on the visualization, users can explore potential topics and subevents. Then, they can label training examples for a specific topic or subevent. The accuracy of labels can be checked through the visual interface. When users feel satisfied with the labeling accuracy, they can train a classifier. If not, they can relabel the tweets or make further queries. The effect of the classifier is also visualized in the visual interface. Users can save the classifier locally or make new queries based on the classification result. If the effect is dissatisfying, users can relabel the tweets and train a new classifier. The consequent of new queries or classifiers can then again be visualized and evaluated.
information. Therefore, the tweet texts have to be split into single words. Besides, these features are also used in PCA and t-SNE. At last, this approach implements several exploration tools that need separately stored keywords, URLs, hashtags, and user_mentions.

Above all, it is important to notice that tweets have some characteristics that are different from other texts. First, Twitter messages are text-based and can be downloaded from Twitter API. Thus, it is straightforward to collect millions of tweets. Second, Twitter restricts the maximum length of tweets. In September 2017, Twitter began a test of doubling the maximum tweet length to 280 characters. Before that, the maximum length was 140. Third, tweets vary in length. Some tweets even only have one sentence or short phase. Fourth, Twitter users post tweets from different media, such as cellphones and web pages. Therefore, the tweet texts are more colloquial and contain more abbreviations, misspelling, slang and Internet languages, which is very different from formal literature. Lastly, there is always a variety of topics discussed by Twitter users simultaneously rather than a specific one.

Several steps are taken for tweet preprocessing. To start with, hashtags, user_mentions, and URLs are extracted from the tweet texts. For example, there is a tweet text: "@somebody, did you watch the new #movie? http://....", "@somebody", "#movie" and " http://... " are extracted from the tweet text and put into separate fields. Next, the user_mentions and the URL are removed from the tweet text. The tweet text is converted to "did you watch the new #movie?". Notice that the hashtag is kept. The preprocessed tweet texts are the foundation for later processing such as training classifiers and performing t-SNE. Thus, it has to be further cleaned. The cleaning rules introduced in the paper[VGC17] is adopted in this work.

1. Transform all Unicode symbols into ASCII. Part of accents and brackets were removed.
2. Remove all whitespace characters, apart from spaces, so that newline, tabs, etc. are excluded.
3. Remove retweets that contain the keyword "RT".
4. Transform the tweet text into lowercase.
5. Remove all non-english characters other than alphanumerics and spaces.
6. Transform appearance of multiple spaces into a single space.
7. Replace the appearance of two or more characters in succession with the character itself. For instance, “I am so huuuugry...” is converted to “i am hungry.”
8. Remove whitespace at the start and end of tweets.
9. Remove stopwords.
11. Tokenize the rest text.

After these steps, a relatively clean tweet set was generated. An original tweet is separated into 12 fields. The cleaned tweet text is stored in field text. The original tweet text is stored in the field origin_text, the extracted hashtags, user_mentions, and URLs are stored in corresponding fields. At last, the preprocessed tweets are stored for later use. Table 3.2 gives the preprocessed tweet structure.
3.4 Corpus Query

<table>
<thead>
<tr>
<th>Field</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>tweet_id</td>
<td>123456789123456789</td>
</tr>
<tr>
<td>user_id</td>
<td>123456</td>
</tr>
<tr>
<td>created_at</td>
<td>201108231100</td>
</tr>
<tr>
<td>latitude</td>
<td>40.7306100000</td>
</tr>
<tr>
<td>longitude</td>
<td>-73.9352420000</td>
</tr>
<tr>
<td>place</td>
<td>New York, NY</td>
</tr>
<tr>
<td>location</td>
<td>-73.9352420000, 40.7306100000</td>
</tr>
<tr>
<td>origin_text</td>
<td>@somebody #something, hello, http://...</td>
</tr>
<tr>
<td>text</td>
<td>something, hello</td>
</tr>
<tr>
<td>#</td>
<td>something</td>
</tr>
<tr>
<td>@</td>
<td>somebody</td>
</tr>
<tr>
<td>URL</td>
<td>http://...</td>
</tr>
</tbody>
</table>

Table 3.2: Preprocessed Tweet Structure - After preprocessing, each tweet has twelve fields: tweet_id, user_id, created_at, latitude, longitude, place, location, origin_text, text, #, @, and URL.

3.4 Corpus Query

After the preprocessing stage, it is possible to make queries for different information in different fields. This approach provides six kinds of basic queries: Keyword Query, Geographic Query, Time Range Query, URL Query, Hashtag Query, and User_mention Query. In addition, several basic queries can be combined to form a multi-condition query. Each query returns two results, relevant and irrelevant results. The relevant one contains tweets those match the current query condition, and irrelevant one is the non-matching part. Users can make queries hierarchically. The hierarchical query detail is discussed in next section. Users can define the automated annotation rules by building their own set of queries. A search view in the user interface (Figure 3.2) is provided for users to make queries.

1. **Keyword Query**: The keyword query works on the text field. Users can query the tweet set with one or more keywords. Since the stopwords are removed from tweet texts, the input keywords should not be stopwords. Boolean operations are possible in keyword query, which means users can input multiple keywords at a time and define their relationships. For instance, the input keywords are "Earthquake AND Shake OR quake NOT 5.8 ", the returned relevant tweets must contain "earthquake" and "shake", may contain quake, and do not contain 5.8.
3. **Geographic Query**: The geographic query works on the *location* field. There are two types of geographic query: rectangular query and radius query. The rectangular query has four input parameters, the minimum and maximum latitude and longitude. These four parameters form a geographic rectangle, tweets whose latitudes and longitudes are inside this rectangle are returned as relevant result. For radius query, three parameters are required: the latitude and longitude of a center point and a radius. Tweets have distances from the center point shorter than the radius are returned as relevant result. Generally speaking, rectangle queries are more precise.

3. **Time Range Query**: The time range query works on the *created_at* field. It looks for tweets whose *created_at* field values are within the input start and end time.

4. **URL Query**: The URL query works on the *URL* field. The tweets contain the input URL are returned as relevant result.

5. **Hashtag Query**: The hashtag query works on the *#* field. This query returns tweets contain the input hashtag. As is mentioned before, hashtags are not removed from tweet texts. Users can use the keyword query in replace of the hashtag query. Twitter suggests users to use only one hashtag and one user_mention in a single tweet. Thus, hashtag query does not support multiple inputs as the keyword query.

6. **User_mention Query**: The user_mention query works on the *@* field. It gives back tweets that mention the specified user. It does not support multiple inputs for the same reason as the hashtag query.

If users enter several query conditions at a time in the search view, these query conditions are performed using logical AND operations. Out of the first query, all queries are made based on the previous results. In an iterative process, users are helped to continuously improve their understanding of the tweet set, find relevant discussions and subevents, and build textual queries that fit their specific information needs.

### 3.5 Tree Generation

The previous section mentions that queries are made in a hierarchical way. To be more precise, they can be made in a tree structure. There is an initial query which is executed first to fetch an initial tweet set and generate a tree root node. Usually, the initial query is a *Time Range Query* or a *Geographic Query* or a combination of them. Except for the initial query, each query is executed on the tweet set in a specified node and returns relevant and irrelevant results. The relevant result is the tweets inside the specified node that match the current query condition, and the irrelevant result is the non-matching tweets. Then the returned results are inserted into the tree as child nodes of the specified node. A complete binary tree is suitable for describing this kind of structure. Figure 3.3 shows the structure of the tree. A generated tree contains a set of nodes, and each node contains a set of tweets. The following steps are taken to generate a tree:

1. Execute an initial query. If the result is not empty, set a tree root and insert the returned tweets into the root. Then set the root as the selected node.
3.6 Data Visualization

**Figure 3.2:** Search view in the user interface - It allows users to enter different query conditions A) Keyword Query, B) Geographic Rectangular Query, C) Geographic Radius Query, D) Time Range Query, E) URL Query, F) Hashtag Query, G) User_mention Query.

2. Execute the second query. When any of the two results is not empty, insert the relevant result as the right child node and the irrelevant result as the left child node of the root into the tree. The right child node is set as the selected node by default.

3. Select a node as the selected node or use the default one.

4. Make a new query, insert the results as the child nodes of the selected node when there is any non-empty result.

5. Repeat step 3-4.

### 3.6 Data Visualization

Data visualization helps human to extract implication of data easily and quickly. The tree generation process is introduced in last section. The generated tree is visualized using a treemap in the visual interfaces given in Figure 3.4. The data visualization of this work consists of 7 parts: (a) Treemap (b) Tweet Table (c) Progress Bar (d) Node Information Report (e) Minimap (f)Node Detail Report.
3 Approach

(g) Operation Buttons. The visual interface is interactive. Through the visual interface, users can check the content of each tweet, find the potential topics in the tweets, label or relabel tweets, which enhances the distant supervision and helps users to understand the tweet data.

Figure 3.3: UML diagram of the generated tree structure

Figure 3.4: Main view in the visual interface - It contains 7 parts: A) Treemap B) Tweet Table C) Progress Bar D) Node Information Report E) Minimap F) Node Detail Report G) Operation Buttons.
Treemap is a method for displaying hierarchical (tree-structured) data using nested rectangles. The detailed introduction of the treemap is given in Section 2.1.5. To generate a treemap, it needs to start from the tree root node and traverse the tree depth-first. At first, a rectangle is assigned to the root. Each time a node contains child nodes, its assigned rectangle is split by a line that alternates in a vertical or horizontal fashion to represent its children. The splitting process stops when it reaches leaf nodes. Figure 3.5 shows a tree which matches the tree structure of this approach. Figure 3.6 shows how to generate a treemap for Figure 3.5.

![Tree diagram](image)

**Figure 3.5:** Example of a complete binary tree - The tree structure used this approach. It is a complete binary tree, each node has and only has two child nodes except leaf nodes.

![Treemap diagrams](image)

**Figure 3.6:** Treemap Generation Process - Generation of a treemap for the tree given in Figure 3.5.

A problem occurs when the number of splitting times becomes large. The assigned rectangles for the nodes with greater depth become small and unrecognizable, see Figure 3.7. To solve this problem, this work defines a threshold. When the rectangle of a node has any side length shorter than the threshold, the node is treated as a leaf node, which means its rectangle will not be further split. In this work, it is called "display node".
The generated treemap has a series of rectangles of different sizes. Each rectangle represents a node in the tree. Users can select a node by clicking on its rectangle. As mentioned above, when a node is a display node, its subtrees are not shown in the treemap. Also, when the rectangle has been split for many times, the rectangle of a deep node becomes very small, operating such node is difficult. Therefore, a "zoom in" function is provided. Selecting a node and zooming in it, the selected node is set as a new tree root. The treemap is repainted based on the new tree root. A "zoom out" function is also provided correspondingly. After zooming in a node, the treemap does not show the original tree structure anymore. A minimap is given in the main view to show the original tree structure. The rectangle of the selected node is marked by a specified color in the minimap.

Each node contains a set of tweets. If a node is a display or leaf node, the tweets that it contains are represented by points in the treemap. In this work, the rectangles and points in the treemap are called tweet nodes and tweet points. Up to this point, the treemap looks like Figure 3.8. Through this treemap, the tree structure of queries and the returned amount of tweets of each query can be seen easily. However, none information about the contents of the tweets is given. Tweet contents need to be known in order to define labels of the tweets. In this approach, several methods are implemented to help users check the contents of the tweets in each node.

1. **Tweet Table:** A tweet table is implemented to show the tweet contents including post date, place, tweet text, hashtag, user mention, and url. Other information like latitude, longitude, and user_id is not provided because it is difficult to summarize useful information directly from it.

2. **Coordinates:** Proper coordinates can help users understand data and find potential information. This approach uses four types of coordinates for each tweet node: random, post date, decision value and t-SNE.
   - **Random:** In the default mode, the x(horizontal) and y(white) coordinates of the tweet points are generated by random.
3.6 Data Visualization

- **Post Date**: As discussed above, the splitting line separates the rectangles in horizontal and vertical directions alternatively. It is a reference to set coordinates. When the post date is used as the coordinate, the coordinate which is opposite to the splitting line is the post date. The post date coordinate sometimes gives beneficial information. For example, when users search for tweets with the keyword "earthquake" and use post date as the coordinate, there is a clear dividing line in the tweet node, whose value is very close to the time of occurrence of the earthquake. However, at other times, the effect of the post date coordinate is not obvious.

- **Decision Value**: The coordinate parallel to the splitting line represents the decision value. A larger decision value means the classification result of this tweet is more likely to be correct. The distance between tweet point and the splitting line is determined the decision value. Thence, users can check the tweet points that located very close to the splitting line. If most of these tweets are classified correctly, the quality of the classifier is good. Otherwise, users can relabel these tweets and start a new round of training.

- **t-SNE**: The coordinates of the tweet points are generated by the t-SNE algorithm. Tweets contain more same words are placed closer in the tweet node. This characteristic helps users to find potential relationships between tweets. However, computing t-SNE coordinates need large running memory and long running time. It is not recommended to be used when there are more than 1000 tweets.

In the decision value and post date mode, the undefined coordinate can be random or each other.
3. **Tag Cloud**: The tag cloud allows users to quickly recognize the most prominent words in tweets. These words promote potential discussed topics. An example of the tag could is given in Figure 3.9.

4. **Content Lens**: Similar to the tag cloud, the content lens also allows users to recognize potential topics. The content lens moves on the treemap and generates a smaller tag cloud around it. It can find topics that are not shown in the tag cloud. An example of content lens is given in Figure 3.9.

![Figure 3.9: Example of Tag Cloud and Content Lens](image)

*Figure 3.9*: Example of *Tag Cloud* and *Content Lens* - The figure shows how *Tag Cloud* and *Content Lens* look like after executing a *Time Range Query*. It indicates "earthquake" and "shake" appear most times, and a possible subevent is fire.

5. **Node Detail Report**: A text area that shows the most frequently appeared hashtags, user_mentions, URLs.

### 3.7 Data Exploration and Labeling

After the query results are visualized, users can explore the contents of the tweets with the methods introduced in Section 3.6. They can use *Tweet Table* to check the detailed content of each tweet. They can quickly find popular discussions using the *Tag Cloud*, *Content Lens*, and *Node Detail Report*. Some other useful information can also be got by selecting different coordinates.
According to distant supervision, the information existing in tweets can be used as annotators to label tweets as training examples. The tweets contain specific keywords, hashtags, user_mentions, or URLs can be labeled as positive training examples. Others can be labeled as negative training examples. To get more precise training examples, users can make queries to filter these labeled training examples.

In order to do tweet classification, a large number of training examples are demanded. It is time-consuming to label tweets one by one. In this work, each node contains tweets that match some query conditions. If the tweets in a node are considered as positive training examples, the tweets contained in the sibling node are considered as negative training examples. This approach allows users to label all tweets in a node at a time. The node contains positive training examples is labeled as the relevant node, and the node contains negative training examples is labeled as the irrelevant node. The tweets inside them are labeled as relevant and irrelevant tweets respectively.

The training examples generated by distant supervision suffers from noisy data, skew annotation and challenge of generating negative training examples. Labeling all tweets in a node at a time may wrongly label some tweets. To solve these problems and improve the labeling accuracy, users can check the contents of the tweets and relabel the tweets one by one or set query conditions to filter the tweets.

3.8 Classifier Training

After users label a set of tweets and are satisfied with the labeling quality, they can train a classifier. This work is doing tweet classification which belongs to text classification. Many studies have shown that linear classification has as good performance as non-linear classification in text classification problems. Thus, this work chooses the linear support vector machine (Linear SVM) as the training algorithm. After a classifier is trained, it classifies the input samples. This work focuses on binary classification. The tweets are classified into relevant and irrelevant. The classification result is visualized in the treemap. Users can check the correctness of the classification. If the correctness meets the user requirements, the classifier can be saved locally for future use. Users can make further operations based on the current classification result. If the classification result is not satisfying, users can relabel the training samples (tweets) and train a new classifier. This chapter introduces the training examples preparation, classifier selection, and classification visualization.

3.8.1 Preprocessing

An essential premise of using machine learning to do text classification is that the content of a document is necessarily related to the words contained in it. There is always a set of common words between the documents of similar subjects, and documents of different subjects contain very different words. Furthermore, it is important that not only what words are contained, but also how many times these words appear. This premise makes the vector space model (VSM) into a document representation model suitable for text classification. In this model, a text is considered as a collection of features. These features are weighted by word frequency information, and the weighted features are used to construct a vector to represent the text.
Most of the preprocessing steps have been described in Section 3.3.2, including the tweet cleaning rules and the conversion of the original tweet structure into the processed tweet structure. After the preprocessing steps, the text field of each tweet contains a set of words which are the features for classification. A tweet instance for classification can be represented as Table 3.3.

<table>
<thead>
<tr>
<th>label</th>
<th>feature 1</th>
<th>feature 2</th>
<th>feature 3</th>
<th>...</th>
<th>feature n</th>
</tr>
</thead>
</table>

Table 3.3: A Tweet Instance

Bag of words model is used to represent a tweet, which considers each word as a feature and its TF-IDF as its weight. Assuming there are two tweets TW1 and TW2, an example of converting these two tweets into feature vectors is given in the following:

TW1 = This is the first tweet.
TW2 = This is the second tweet.

TW1 and TW2 can be represented by following vectors:

V1 = (This:1, is:1, the:1, first:1, tweet:1)
V2 = (This:1, is:1, the:1, second:1, tweet:1)

The pair "word: number" indicates the number of times the word appears in a tweet. Other tweets can also be represented in this way. Through observing V1 and V2, an easier way can be found to represent them. Extracting all the words existing in all tweets to form a dictionary. The dictionary is also a vector. The dictionary D for TW1 and TW2 is given in the following:

D = (This, is, the, first, second, tweet)

Based on this dictionary, TW1 and TW2 are represented as:

V1 = (1, 1, 1, 1, 0, 1)
V2 = (1, 1, 1, 0, 1, 1)

Each number in the vector represents the number of occurrences of the word that has the same index in the dictionary vector. However, using the number of occurrences as feature weight has some shortages. In this work, TF-IDF is used as the feature weight. The detailed introduction of TF-IDF can be found in Section 2.1.4. At last, V1 and V2 are like following:

V1 = (0, 0, 0, 0.0602, 0, 0)
V2 = (0, 0, 0, 0, 0.0602, 0).

Till now, the generated feature vectors can represent the tweets.
3.8.2 Feature Selection

However, the introduced conversion process has a problem. The dictionary contains all words from all tweets in the tweet set, which means the dictionary size is huge. In this work, a dictionary usually consists of hundreds of thousands of words. A tweet used in this work includes at most 140 characters. It is wasteful to represent a tweet with a vector that has hundreds of thousands of dimensions. The solution is feature selection, which is sometimes called dimension reduction. A fact is that words like "this", "is", "the" appear very often in any tweet and have little contribution to the subject of a tweet. These kinds of words are called stopwords. Removing the stopwords helps to reduce the dimensions. The cleaning rules also help to reduce the dimensions. After calculating the TF-IDF of the words, the tweet text can be transformed to feature vectors that can be used for classifier training.

3.8.3 Classifier selection

Feature vectors are the input of the classifier training. Users have labeled a number of tweets as training examples. These training examples are first transformed into feature vectors. The computer accepts these training examples, and it considers these labels are correct. It observes the features of these training examples to guess a possible classification rule. Once this classification rule meets some requirements, this rule is considered to be approximately correct and good enough to become the classifier. When there is a new tweet that has not seen before, this classifier can determine its class.

There are some classification algorithms for doing text classification such as Naive Bayes, Decision Tree, Rocchio, and Support Vector Machine. This work employs the linear support vector machine (Linear SVM) as the training algorithm. It solves the following unconstrained optimization problem with different loss functions \( \xi(w; x_i, y_i) \):

\[
\min_w \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi(w; x_i, y_i)
\]

where \( C > 0 \) is the penalty parameter. \( w \) is the normal vector to the hyperplane. For Linear SVM, the two common loss functions are \( \max(0) \) and \( \max(0)^2 \). In the testing phase, a data point \( x \) is classified as positive if \( w^T x > 0 \), and negative otherwise. \( w^T x \) is the decision value.

When using a **Keyword Query** or a **Hashtag Query**, every returned tweet in the relevant result contains the input words. If a classifier is trained with such tweets, it may give very high weights to the input words and does not learn other features. In addition, learning and testing on the same data can cause a mistake: the trained classifier only repeats the labels of the training examples and fails to predict the data that has not been seen yet. This problem is called overfitting. To solve this problem, this approach introduces a percentage. Users can set a percentage, the set percentage of tweets keep the input words while other tweets remove the input words.
3 Approach

To check the performance of the classifier, k-fold cross-validation (CV) is employed. The k-fold cross validation randomly splits training examples into k parts. It uses k-1 parts as the training data to train the classifier and use the left one part as the test data. Each part is used as a test data once, and the other parts are used as training data at the same time.

3.8.4 Classification Visualization

The trained classifier classifies the input tweets as relevant or irrelevant to the query conditions of the selected relevant node. Users can collect the relevant tweets and put them into a selected node. After that, they can make queries on the tweets in this node.

In order to check the performance of the classification, the result is visualized in the treemap. As mentioned in Section 3.6, after doing classification, tweet nodes use decision values as coordinates, the treemap after classification is given in Figure 3.10:

![Figure 3.10](image)

**Figure 3.10:** Example of the treemap after classification - Relevant tweet points are in orange and irrelevant tweet points are in blue. The tweets that have different labels from classification results are marked with round shape. The coordinate parallel to the splitting line is determined by decision value. Tweets with smaller decision values are placed closer to splitting line.
4 Implementation

A software is developed to achieve the approach discussed in this thesis and evaluate the overall performance of the approach. It is completely developed in Java, and the used integrated development environment (IDE) is Eclipse\(^1\). It deploys Apache Lucene\(^2\) as a search engine, Java Swing\(^3\) for visualization, Stanford CoreNPL\(^4\) for NLP preprocessing, Weka\(^5\) and T-SNE-Java\(^6\) for generating tweet point coordinates and LIBLINEAR\(^7\) for classifier training. More detailed introduction of these dependencies is given in Section 4.4.

The software has two views: the main view and the search view. It can be used to preprocesses selected tweets and creates local tweet database. It offers robust search operations to obtain required tweets. A tweet contains information including text content, post date, geographic location, hashtag, user_mention, URL, etc. According to the distant supervision theorem, this information can be used as annotator to generate labels. For example, a tweet that contains the keyword "earthquake" has very high possibility that its content is discussing an earthquake. This software has different types of queries to search for different information. Users can define a set of queries to get tweets according to their requirements. The queries can be done in a hierarchical way, which means a new query can be made based on the results of previous queries. Users can enter the query conditions in the search view, and the query results are shown in the main view. Through designing a set of queries, users can define their own rules for generating positive or negative training examples. The generated training examples might suffer from noisy data, skewed annotations, and the challenge to generate negative training examples. Thus, users can use the main view to check the contents and labels of the tweets. If a tweet is wrongly labeled, it can be modified. Through the main view, a user can also get a sense of the accuracy of the labels. This work focuses on binary classification. Therefore, the generated labels are only relevant and irrelevant. When users are satisfied with the labeled tweets, they can use the labeled tweets to train a classifier. The generated classifier classifies the tweets existing in the relevant and irrelevant tweet nodes. The result is shown in the main view. Users can check its performance. If the performance is not satisfying, they can relabel the tweets and train a new classifier. If satisfying, they can save the generated classifier locally or use it for further training examples generation.

The following sections give an overview the user interface, the basic exploration tools, and the used libraries. Local tweet database generation, query creation, treemap generation, and classifier training techniques are introduced as well.

\(^1\)https://www.eclipse.org/
\(^2\)http://lucene.apache.org/core/
\(^3\)https://docs.oracle.com/javase/9/docs/api/javax/swing/
\(^4\)https://stanfordnlp.github.io/CoreNLP/
\(^5\)https://www.cs.waikato.ac.nz/~ml/weka/
\(^6\)https://github.com/lejon/T-SNE-Java
\(^7\)https://docs.oracle.com/javase/9/docs/api/java/swing/
4 Implementation

4.1 Data Index and Query

Tweet classification needs a large number of labeled training examples. In order to generate these training examples, an even larger tweet data source is needed as well as powerful query operations. In particular, fast query response is essential for this approach. The information contained in tweets are used as annotators for generating training examples, including tweet_id, user_id, post date, text, geolocation, user_mention, URL, and hashtag. This information has different data types and needs to be stored separately in order to be easier queried. Apache Lucene can achieve fast search response and offers storage and query functions for various types of data. Thus, it is adopted as the search engine. As mentioned in Section 3.3.2, the collected tweets are firstly preprocessed and the information is extracted. The preprocessed tweets are stored as Lucene index. Table 4.1 shows the mapping between the designed and the Lucene fields and queries. Besides, Lucene also calculates word frequency information, which can be used to generate TF-IDF. A more detailed introduction of Lucene is given in Section 4.4.1.

<table>
<thead>
<tr>
<th>Design Field</th>
<th>Lucene Field</th>
<th>Design Query Type</th>
<th>Lucene Query Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tweet_id</td>
<td>LongPoint</td>
<td></td>
<td>Set Query</td>
</tr>
<tr>
<td>user_id</td>
<td>LongPoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>LongPoint</td>
<td>Time Range Query</td>
<td>Range Query</td>
</tr>
<tr>
<td>latitude</td>
<td>DoublePoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>longitude</td>
<td>DoublePoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>place</td>
<td>TextField</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>TextField</td>
<td>Hashtag Query</td>
<td>Term Query</td>
</tr>
<tr>
<td>@</td>
<td>TextField</td>
<td>User_Mention Query</td>
<td>Term Query</td>
</tr>
<tr>
<td>origin_text</td>
<td>StringField</td>
<td></td>
<td></td>
</tr>
<tr>
<td>url</td>
<td>StringField</td>
<td>URL Query</td>
<td>Term Query</td>
</tr>
<tr>
<td>location</td>
<td>StoredField</td>
<td>Geographic Query</td>
<td>Strategy Query</td>
</tr>
<tr>
<td>text</td>
<td>Field</td>
<td>Keyword Query</td>
<td>Query</td>
</tr>
</tbody>
</table>

Table 4.1: Queries and Fields - The correspondence between the queries and fields designed by this approach and the Lucene query syntax.
4.2 User Interface

The look of the user interface is given in Figure 4.1 and Figure 4.2. It consists of a main view and a search view. The search view is provided for users to enter query conditions and the main view is used for visualizing and operating the tweets.

4.2.1 Search View

The search view can be seen in Figure 4.1. It consists of a set of text input boxes(Figure 4.1.A), a Search button(Figure 4.1.B) and a slider(Figure 4.1.C). The search view is opened and closed by the button Search in the main view.

Users can input the query conditions through the text input boxes. A single condition or multiple-condition query is supported. When there are more than one query conditions at a time, these queries are performed in an AND manner. This software supports six query types: keyword, hashtag, URL, user_mention, geolocation, and time range query. Each type of the queries has one text input box except for Geographic Query and Time Range Query. Geographic Query has two subtypes, Rectangular Query and Radius Query. Rectangular Query has two input text boxes, users need to input (minimum latitude, maximum latitude) in Lat Range and (minimum longitude, maximum longitude) in Lon Range. Radius Query has three text input boxes, the latitude and longitude of the center point have to be input in Lat and Lon respectively, and the radius has to be input in Distance. For Time Range Query, the start time and end time has to be input in Start Time and End Time using the format yyyy-MM-dd HH:mm.

The slider(Figure 4.1.C) is designed for solving the overfitting problem that exists in classifier training. Users can set a percentage by the slider, the set percentage of return tweets keeps the words that are same as the input words of Keyword Query and Hashtag Query, other tweets remove these words. The default percentage is 0%.

After inputting the query conditions and setting the slider, users can press the button Search to start the query process.

4.2.2 Main View

The main view of the software can be seen in Figure 4.2. It is a highly interactive visual interface that shows the results of user operations and helps users to achieve a better understanding of tweet data and manipulate it. The main view consists of 7 parts: treemap, tweet table, progress bar, minimap, node information report, node detail report and operation buttons.

The treemap(Figure 4.2.A) is the core of the main view. It shows the query tree structure based on binary space partitioning. Every rectangle in the treemap represents a node in the query tree. The points inside the rectangle represent the tweets inside the node. A node can contain at most 100,000 tweets even when there are more tweets meeting the query conditions, in order to make sure the points in the rectangle are not crowd.
4 Implementation

Figure 4.1: Search view of the software UI - It consists of A) text input boxes for query conditions B) a search button C) a keyword retention ratio

Figure 4.2: Main view of the software UI - It contains 7 parts: A) Treemap B) Tweet Table C) Progress Bar D) Node Information Report E) Minimap F) Node Detail Report G) Operation Buttons.
4.2 User Interface

Users can select tweets in the treemap using mouse or Content Lens in three ways. First, they can click on a tweet node, the tweets inside the corresponding tree node are selected at the same time. Secondly, they can click a single tweet point, the selected point is marked by red and the corresponding tweet is selected. Lastly, they can use the Content Lens to select several tweets at a time. Tweets corresponding to tweet points covered by Content Lens are selected.

The default look of a tweet point is a dark gray square point. After classification, tweets that are classified as relevant and irrelevant are marked by orange and blue respectively. Tweets have different labels and classification results are marked by round shape. If users train a classifier with two nodes, the corresponding splitting line of the relevant tweet node is red. Otherwise, it is black.

The tweet table (Figure 4.2.B) shows the selected tweets. When a node is selected, the tweet table shows all tweets inside it. When a single tweet is selected, the tweet table only shows this tweet. When a Content Lens selects a set of tweets, the tweet table shows this set of tweets. The tweet table has six columns including Date, Place, Text, @, #, and url. The number of rows is determined by the number of selected tweets.

The minimap (Figure 4.2.E) shows the original tree structure. When users do not zoom in any node, it looks same as the treemap. When users zoom in a node, the treemap is repainted using the node as the tree root. It does not show the original tree structure anymore. In such case, the minimap displays the structure of the original tree, and it marks the selected node by a defined color.

A progress bar (Figure 4.2.C) shows the real-time information of fetching query results including query start, the number of fetched tweets, query end. The node information report (Figure 4.2.D) displays some useful information of the selected node, including the depth of the node in the tree and the number of tweets, hashtags, user mentions, and URLs.

When the rectangle of a node is too small, or the density of tweet points in a tweet node is high, it becomes difficult to process the tweet points. Users can zoom in the node by pressing the button Zoom In. The zoomed in node are set as a new tree root, and its rectangle expands to the entire treemap. The treemap is repaint according to the subtree of the new tree root. Pressing the button Zoom Out, the treemap returns to the state before last zoom in.

Using a proper coordinate system helps users to get a better understanding of the tweet data. The main view offers two buttons for users to choose a coordinate system: t-SNE Map and Use Post Date. Figure 4.3.a and Figure 4.3.b show how a tweet node looks like before and after deploying t-SNE. It can be seen from the Figure 4.3.b, some tweet points gather together. The distance between two tweet points represents their similarity. Generally speaking, tweets that contain more same words are put closer. The t-SNE algorithm is not recommended to execute on too high dimension data. The input data dimension of t-SNE should be less than 50. Thus, PCA is performed first to reduce the data dimension. The input data of PCA and t-SNE is feature vectors similar to the Linear SVM. The difference is that the feature weight (TF-IDF) of Linear SVM is calculated based on all tweets in local index but the TF-IDF used in PCA and t-SNE is calculated based on the tweets in a node, because t-SNE in this work is designed for finding potential topics and relationships in a node. After input the features vectors, PCA and t-SNE calculate their two dimension coordinates. Then the tweet points are painted using the generated coordinates. Besides, the input feature vectors usually have very high dimensions, operating such data often needs long running time and large running memory. In order to ensure the performance, PCA and t-SNE in this approach is not recommended to process more than 1000 tweets.
4 Implementation

Figure 4.3: Example of t-SNE - The appearance of a tweet node before and after employing t-SNE are given in (a) and (b). (a) The tweet points are placed randomly. (b) Some tweet points gather together.

Figure 4.4.a and Figure 4.4.b show how a tweet node looks like before and after using post date as a coordinate. If the number of tweets increases suddenly at a certain time, there is usually an event happened at this time. The post date of these tweets is usually occurrence time of the event. It also helps users to find sub-events caused by the event. To calculate the post date coordinate, the created_at field is used. Several steps are taken to generate post date coordinates. First, iterate through all the tweets in the node to find the maximum and minimum dates. Next, normalize the post dates of all tweets. Then, use the normalized dates to generate the coordinates. At last, paint the tweet points using the generated coordinates.

Users can label the tweets in the treemap in two ways. First, they can label all tweets in the selected node as positive or negative training examples using buttons Relevant Node and Irrelevant Node. Secondly, they can label a tweet as a positive or negative training example by selecting a tweet point and using buttons Relevant Point and Irrelevant Point. After generating enough training examples, users can train a classifier by pressing the button Train Classifier.

Figure 4.5 shows the results of training a classifier. The generated classifier classifies the input training examples. The orange tweet points represent the tweets that are classified as relevant while the blue tweet points represent irrelevant tweets. The distance between a tweet point and the splitting line is proportional to the decision value of this tweet. The points that are closer to the splitting are more likely to be wrongly classified. Users can get a sense of classifier quality by checking these points, and they can relabel wrongly classified tweets using buttons Relevant Point and Irrelevant Point. If users are not satisfied with the classification result, they can train a new classifier with the relabeled tweets. If they are satisfied, they can save the classifier locally by pressing the button Save Current Model. If users want to further explore the tweets classified as relevant, they can use the button Explore. All relevant tweets are collected and put into the currently selected node. The color of the splitting line is changed to red indicating a classifier is trained.
4.2 User Interface

Figure 4.4: Example of using post date coordinate - The appearance of a tweet node before and after using post date coordinate are given in (a) and (b). (a) The tweet points are placed randomly. (b) The y-positions of the tweet points are determined by their post dates.

Figure 4.5: Example of tweet nodes after classification - After classification, tweets classified as relevant and irrelevant are in orange and blue respectively. Tweets that have different classification results and labels are marked by round shape.
4 Implementation

4.3 Basic Exploration Tools

Users can check tweet contents in the *Tweet Table*. However, it is very time-consuming to read many tweets and difficult to draw conclusions, e.g., find potential topics. In addition to the *Treemap* and *Tweet Table*, three more tools are provided: tag cloud, content lens, and node detail report. They are used to investigate selected tweets in a summarized fashion. These three tools convey the most frequently appeared contents of selected tweets to users.

4.3.1 Tag Cloud

The tag cloud is opened and closed by the button *Show Tag Cloud*. When the density of *tweet points* is high, it affects the visual performance of the tag cloud. The appearance of the tag cloud can also hinder operating *tweet points*. Thus, users can decide whether to open it or not.

Previous chapters have mentioned the usage of the tag cloud. It helps users assess the tweet contents in the node. Figure 4.6 shows the tag cloud. The tag cloud gives the most frequently appeared words in the node. The words stored in *text* fields are used for generating tags. All words of all tweets in the node are weighted, and words with the highest weights are used as tags. There are different kinds of weighting schemas. This work first tries TF-IDF. However, TF-IDF gives high weights to the words that seldom appear, which means few documents contain these words. This approach needs to get as much returned tweets for each query as possible. As a result, TF-IDF is not adopted. Then this work tries the term count. Term count is the number of occurrences of a term in all tweets in a node. Since the stopwords are removed from the tweet text, the generated tag cloud does not contain common but meaningless words and shows potential topics and discussions. Thus, this work adopts term count as the tag weight. After getting the weighted words, the words with highest term count are used as tags. The number of tags shown in each *tweet node* depends on the area size of the *tweet node*. The maximum number is 50. The font size of the tags is determined by term count. The higher the term count is, the larger the font size is. The generated tags are displayed in a circular shape in the *tweet node*.

4.3.2 Content Lens

The content lens is opened and closed by the button *Content Lens*. It displays the most frequent words of the tweets inside it. Its size can be adjusted by rolling the mouse wheel. The content lens can move over the treemap freely. It can find words that are not shown in the tag cloud, which may indicate potential topics. Thus, it is particularly useful when moving in the *tweet node* which uses t-SNE coordinates. In such node, tweets contain same words gather together. Users can discover these same words using the content lens. Additional, the content lens can be used as a selector to select multiple tweets at a time. An example of content lens is given in Figure 4.6. It shares the same tag generation algorithm with the tag cloud. The tag cloud generates the tags based on all tweets in a node while content lens generates tags based on the tweets inside the content lens. The generated tags are placed around the content lens.
4.4 Libraries

As mentioned at the begin of this chapter, some Java libraries are used to implement this software. The necessary libraries are introduced in this section.

![Figure 4.6: Example of Tag Cloud and Content Lens](image)

The screenshot shows how Tag Cloud and Content Lens look like after executing a time range query. It indicates "earthquake" and "shake" appears most times, and a possible subevent is fire.

4.3.3 Node Detail Report

The node detail report(Figure 4.7) is a text area that shows the most frequent hashtags, URLs and user_mentions in the selected node and their number of occurrence. To generate the node detail report, the #, @ and url fields of the tweets are used. All the hashtags, URLs, and user_mentions of the tweets in the node are collected, and their number of occurrences is calculated. Then, these hashtags, URLs, and user_mentions are sorted by their number of occurrences, and the most frequent ones are shown in the node detail report. The maximum shown number is 50. Users can make hashtag query, URL query, or user_mention query according to the node detail report.
Figure 4.7: Example of the Node Detail Report - Node Detail Report shows the most frequent hashtags, user_mentions, and URLs.

4.4.1 Lucene

Lucene is a Java-based full-text search library which provides indexing and searching capabilities for applications. It can index text-typed data, as long as the format of data can be converted to text, Lucene can index and search the data. The preprocessing and query functions of this software are implemented with the help of Lucene. Some basic concepts of Lucene are listed in the following:

1. **Indexing and Searching**: Lucene can achieve very fast search responses, which is benefit from indexing. The process of building an index is to convert the source data into an index that is convenient for querying. Lucene uses a mechanism called inverted index. Inverted index means there is a maintained word or phrase table, for each word or phrase in the table, there is a linked list saving which documents contain the word or phrase. This allows users to get search results very fast after they enter the search condition. After indexing the source data, users can search on the index. The search engine first parses the search keywords and then search on the established index. The returned results are ranked by either their relevance to the query or sorted by an arbitrary field.

2. **Document**: A unit of searching and indexing. An index consists of one or more documents.

3. **Field**: One of the properties of a document.
   - **LongPoint**: Index long type field.
   - **DoublePoint**: Index double type field.
   - **TextField**: Index and tokenize. The input strings are broken into lexical units or tokens.
   - **StringField**: Index but not tokenize. The input strings are stored as a whole.
4. **Query**: An abstract class for searching.

5. **Analyzer**: The analyzer is used when a document is indexed and queried. It examines the text of fields and generates a token stream. Analyzers may be a single class, or they may be composed of a set of tokenizers and filters. It is necessary to choose the right analyzer according to the language and application needs.

6. **Tokenizer**: Break field data into lexical units, or tokens.

7. **Filters**: Examine a stream of tokens and keep, transform or discard them.

### 4.4.2 Standford CoreNPL

Standford CoreNPL is a well-known suite for Natural Language Processing (NLP). It provides a set of human language technology tools. In this work, it is used to perform lemmatization. For grammatical reasons, words appear in several inflected forms, e.g., have, has, had. In this approach, lemmatization is important for reducing the feature dimension. It also helps improve query efficiency.

### 4.4.3 LIBLINEAR

This work adopts LIBLINEAR as the training library. It is an open source library for large-scale linear classification. It supports logistic regression and linear support vector machine. LIBLINEAR is very efficient for large-scale classification problems such as text classification. For text classification, the nonlinear mapping does not improve performances. Without using kernels, one can fast train a much larger set through a linear classifier. Chapter 3 has introduced the generation of feature vectors, the input data of classification is the feature vectors.

Using LIBLINEAR needs to select a solver and some parameters. According to the recommendation of LIBLINEAR guide paper [FCH+08], the default dual-based solver is selected, and the only parameter is C which is the cost of constraints violation.

LIBLINEAR provides an easy solution to find a suitable C. If a primal-based classification solver is used, LIBLINEAR provides a -C option to find best C automatically through conducting cross-validation several times. If a dual-based classification solvers is used, users can first use a primal-based solver to find the best C and use it in the dual-based solvers.

Besides, LIBLINEAR has several important methods:

1. `Linear.train()`: accept training examples and training options then start training.
2. `Linear.predict()`: accept a test example and output a predicted class.
4.4.4 Weka and T-SNE-Java

To generate the t-SNE coordinates, the Weka library and T-SNE-Java are used. The t-SNE algorithm is not recommended to execute on very high dimension data directly. The input data dimension should be reduced to less than 50 first. This dimension reduction process is done by PCA which is implemented using the library Weka. Weka is a data mining software in Java. It supports a collection of machine learning algorithms. To do PCA, the package `weka.attributeSelection.PrincipalComponents` is used.

After the number of dimensions is reduced to less than 50, t-SNE can be performed using the library T-SNE-Java. It is a pure Java implementation of Van Der Maaten and Hinton’s t-SNE clustering algorithm. It supports several different versions of t-SNE, this work uses the recommended Barnes Hut version which is proved faster than other versions.
5 Case Study

This chapter presents the results of a case study that was conducted on March 09, 2018, to evaluate the overall performance of this approach as well as present the workflow and functionalities of the implemented software. In this case study, an analyst tried to investigate the influence caused by a certain event. He had to collect related tweets, discover potential events. When he found events that he was interested in, he trained classifiers for classifying tweets that were related to the events or not. Then he used the trained classifiers to get more tweets related to the events. This chapter first introduces the process of the case study, then discusses the results.

5.1 Session Report

The 2011 Virginia earthquake occurred on August 23 at 17:51:04 UTC in the Piedmont region of the US state of Virginia. It was a magnitude 5.8 earthquake and the epicenter is 37°56′10″N,77°55′59″W. The earthquake affected more than a dozen U.S states and was felt by more people than ever before in the US history. A large number of earthquake-related tweets were posted after it happened. However, an analyst could not notice these earthquake-related tweets unless he searches the tweets with keywords because these tweets were obscured by daily random chatters. For the same sake, discovering tweets that show the potential effect of the earthquake was even more difficult.

In this case study, the analyst wanted to analyze the impact of the Virginia earthquake in the United States. Thus, tweets posted between August 22nd, 2011 and August 25th, 2011 were selected. These tweets were collected by Institut für Visualisierung und Interaktive Systeme (VIS) of Universität Stuttgart and were stored as CSV files. These files were preprocessed by the software, and a local Lucene index was built. At last, 1,986,707 tweets were indexed. A screenshot of the Lucene index is given in Figure 5.1.

The analyst first employed Geographic Query and Time Range Query to get tweets posted between August 23rd, 2011 and August 26th, 2011 and within the US geographical range: latitude from 19.50139 to 64.85694 and longitude from -161.75583 to -68.01197. Here, Rectangular Query was used in order to specify a more precise geographic range. 94324 tweets were acquired and immediately visualized (Figure 5.2). The analyst recognized the hottest topics through Tag Cloud and Node Detail Report. As shown in Tag Cloud, the term "earthquake" appeared most times. In Node Detail Report, the hashtags "earthquake" and "Earthquake" appeared 874 and 106 times respectively.

According to the distant supervision theory, the term "earthquake" is a perfect annotator that indicating a tweet is earthquake related. The analyst wanted to get all tweets that were earthquake related. He tried to do it in two ways. Firstly, he employed Hashtag Query with the word "earthquake". Secondly, he employed Keyword Query using the same word "earthquake". Keyword Query returned 16331 tweets while Hashtag Query returned 3094 tweets. In order to get more earthquake-related tweets for classifier training, he decided to continue with Keyword Query.

The returned tweets were distributed between 23 August and 26 August. The analyst pressed button Use Post Date to observe the post dates of the tweets. A screenshot of the result is given in Figure 5.3. He found out that after 17:51, there was a sudden increase in the number of tweets, and the tweets posted before 17:51 were discussing another earthquake(Colorado Earthquake). Since it was slow to read the tweets one by one, the analyst used Content Lens to explore these tweets. In order to get rid of these irrelevant tweets, he adopted a Time Range Query with the query condition [2011-08-23 17:51, 2011-08-26 23:59].

16252 tweets were returned and visualized in the right tweet node while other incompatible tweets were visualized in the left tweet node. The analyst decided to train a classifier to find more earthquake-related tweets. He set the lower right tweet node as the relevant node and the left tweet node as irrelevant node using the buttons Relevant Node and Irrelevant Node then pressed the
Figure 5.2: Initial query result of the case study - The screenshot of the user interface after executing the initial Geographic Query and Time Range Query. The word "earthquake" appeared most times in both Tag Cloud and Node detail Report.

button *Train Classifier*. After the classifier was generated, it classified the tweets in the relevant and irrelevant node. The tweets that were classified as relevant were marked by orange and irrelevant were marked by blue in the Treemap. The analyst evaluated the classification accuracy by checking the contents of tweets in Tweet Table. When he found a tweet was wrongly classified, he modified it with the buttons Relevant Point and Irrelevant Point. After checked a set of tweet points, the analyst was satisfied with the quality of the classifier, he trained a new classifier with the relabeled tweets and saved the newly generated classifier locally for future use. Figure 5.4 shows the results of classification.

To further explore the earthquake-related tweets, the analyst pressed the button *Explore*, all tweets that were classified as earthquake-related were collected and placed in the selected node. The effect is shown in the Figure 5.5. From Tag Cloud and Node Detail Reporter, the analyst got some useful information. The tags such as Virginia, east, coast, Washington, Toronto, 5.9, 5.8 showed epicenter, affected cities, and magnitude of the earthquake. It could be seen in the Node Detail Reporter, #USGS\(^2\) was mentioned 25 times, which is a scientific agency of the United States government. Hashtags such as #NYC, #DC, #Washington indicated the cities affected by the earthquake. The Twitter media accounts such as cnnbrk, CNN, BreakingNews, USGS were mentioned dozens of times. The analyst also saw terms "hurricane" and "#hurricanirene" appeared in Tag Cloud and Node Detail Reporter. He found that some of the earthquake-related tweets mentioned Hurrican Irene\(^3\) that was going to land in Virginia.

\(^2\)https://www.usgs.gov/

\(^3\)https://en.wikipedia.org/wiki/Hurricane_Irene
The analyst decided to look into damage caused by the earthquake. He executed a new **Keyword Query** using "damage". 197 tweets were returned. The analyst then used t-SNE coordinates and **Content Lens** to explore the returned tweets. Figure 5.6 shows the result. He found out that some tweets contained URLs regarding the earthquake damage. No one died but someone got injured in the earthquake. He also noticed most of the damage-related tweets were relevant to buildings. Thus he decided to look into the tweets that report the damage of buildings. He created a new **Keyword Query** using "building OR house" and trained a new classifier. The result is shown in the Figure 5.7.

After the above processing, the area size of the newly generated *tweet nodes* became smaller, the analyst pressed *Explore* button and used the *Zoom In* function. In order to increase the probability that the collected tweets were describing the buildings affected by the Virginia earthquake, the analyst adopted a **Geographic Radius Query**. The input query condition is lat: 37.5610, lon: 77.5559, distance: 600km. A magnitude 5.5 eastern U.S. earthquake can usually be felt as far as 483 km from its origin. Thus, the analyst chose 600km as the distance. A screenshot of the result is given in Figure 5.8. After zooming in a certain node, the complete query tree structure cannot be seen from *Treemap* anymore. The analyst used *Minimap* to determine the position of the current node in the query tree. Through *Tweet Table* and *Tag Cloud*, he found that many buildings were shaking during the earthquake, and most people evacuated from the buildings in a short time.
Figure 5.4: Result of classification - relevant and irrelevant tweets were marked by orange and blue respectively. Tweets with different training labels and classification results were marked with round points.

Figure 5.5: Screenshot of the main view after pressing Explore - All tweets classified as relevant were collected and put into the selected node.
5 Case Study

Figure 5.6: Screenshot of the main view after using t-SNE coordinates - The lower right rectangle showed the returned tweets of the Keyword Query using "damage", Content Lens showed some tweets were discussing the injury.

Figure 5.7: Result of "building OR house" query and classification - Words such as "shake" and "evacuate" may be related to potential events. Besides, the newly generated tweet nodes became smaller.
Figure 5.8: Screenshot of the user interface after making Geographic Radius Query - The treemap is repainted based on the zoomed in node. After Geographic Radius Query, tweets reported building damage in the earthquake-affected area were displayed in the right rectangle.

This case study terminated here. It was still possible to further explore the returned tweets and find potential discussions.

5.2 Discussion

In conclusion, the software allows the analyst to discover the potential aspects of an event and define training examples for a topic he is interested in. He was supported with different tools to find the potential information contained in a set of tweets. The trained classifiers can help him find relevant tweets.

However, after several queries, the amount of returned tweets drops quickly. This problem can be mitigated by using multiple keywords but cannot be solved. Besides, relevant information could have been missed, it may not be discovered by the analyst or not contained in the returned tweets of a certain query.

Although the User_mention Query and URL Query were implemented, they did not have significant help for obtaining useful tweets in this case study. This problem could be caused by the limited number of collected tweets, and many URLs were expired since the tweets were posted seven years ago. This problem may be solved by using larger tweet set or using newer posted tweets.
6 Conclusion and Outlook

In this work, an approach of enhancing distance supervision through visual interfaces is introduced. The approach adopts techniques from different fields including visual analytics, machine learning, information retrieval and so on. In this work, tweet preprocessing function is designed to convert the original tweet structure to a more suitable structure for further processing. Local tweet data index is built to store the preprocessed tweets for later queries. Powerful query functions are implemented to allow users to query tweet data hierarchically. Treemap is adopted for data visualization. Users can observe the query structure and results, operation results, and classification results through the treemap and a set of exploration tools. According to the concept of distant supervision, users can label tweets that are queried with specific information in the tweets such as geolocation, post date, URL, hashtag, and user_mention as training examples. This method solves the difficulty of generating training samples. Users can check and modify the labels of the tweets in the user interface, which solves the shortages of distant supervision. Classifiers can be trained with the training examples, and the trained classifiers are used for finding more required tweets or are saved for future use.

A software is implemented to achieve the approach. The software consists of back-end data processing methods and a user interface. A set of tools are applied to help users explore and get a better understanding of the tweet data. The tools includes Tag Cloud, Content Lens, Tweet Table and Node Detail Report and so on. Users can operate the tweet data through the user interface. Besides, a case study is conducted to evaluate the performance of the approach.

Through this method, the distant supervision is enhanced through visual interfaces. Users can generate a large number of training examples using the information contained in tweets as annotators. They can use the labeled tweets for training classifiers and use the classifiers to obtain more tweets. The effect of labeling and classification can be observed through the visual interface, wrongly labeled tweets can be modified. Meanwhile, during this process, users understand the tweet data better through the visual interface and the exploration tools. They can discover potential topics and continuously explore the tweets.

There are still some limitations in this approach. First, the number of tweets decreases after each query. Although more tweets can be obtained using classifiers, the reduction of tweet amount is not changed. Second, users may need to restart the query process several times if they do not know enough about the tweets. Third, the tweets are stored as Lucene index. There are nearly two million tweets. A tweet node can read at most 100,000 tweets in order to ensure visualization effect. These read tweets may be affected by the Lucene scoring mechanism. If they can be randomly collected, the results may be better.
Outlook

There are different directions in which further works can be done. Two directions are considered in this approach. On the one hand, the limitations existing in this approach should be overcome. On the other hand, the unused information in tweets can be utilized.

Firstly, to solve the tweet amount reduction problem, it might be feasible to add a function that loads tweets into the selected node at real-time and train a classifier that distinguishes tweets are relevant to the selected node or not. Secondly, to reduce the times of returning to the beginning, a function that shows the effect of a operation before actually executing it may be helpful. Thirdly, to get rid of the impact caused by the Lucene scoring mechanism, a different type of database may be used.

Apart from the information that has been utilized in this approach, there is some ignored information. A growing number of Twitter users add images to tweets in order to express opinions or feelings. These pictures might also be utilized for doing tweet classification. It is also possible to add sentiment classification to this approach, which can help users to get a sense of how the public react to a particular event. In the future, this software can be implemented as a real-time version, which allows users to discover real-time events.
Bibliography


All links were last followed on April 9, 2018.
Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature