Universiteit Leiden
Opleiding Informatica

Sentiment Mining On Chinese Product Reviews

Name:	Jaco Tetteroo
Studentnr:	1284819
Date:	25/09/2017
1st supervisor:	Prof.dr. J.N. Kok
2nd supervisor:	Dr. A.-W. de Leeuw

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Sentiment Mining On Chinese Product Reviews

Jaco Tetteroo

Abstract

In this thesis we present a case study on sentiment mining on Chinese product reviews. We investigate to what extend we can estimate the rating belonging to the reviews by analysing the text. The problem we face is a classification problem with 5 classes on a linear scale. We find features using a bag of words approach. We compare the words of the reviews with a sentiment dictionary. These features are used to train a multilayer perceptron. This results in a small improvement of the success rate, similar to that of related work.

Contents

Abstract **i**

1 Introduction **2**

1.1 Case study 2

1.2 Problem statement 2

1.3 Methodology 3

1.3.1 Data acquisition 3

1.3.2 Experiments 4

1.4 Thesis Overview 4

2 Related Work **5**

3 Data acquisition **6**

3.1 Data overview 6

3.2 Web crawler 8

3.3 String separation 9

3.4 Word comparison 10

4 Experiments **12**

4.1 Preprocessing 12

4.2 Experiment 1 13

4.3 Experiment 2 13

4.4 Experiment 3 16

5 Conclusions **18**

6 Appendix **19**

6.1 Amazon Webcrawler 19

6.2 JD Webcrawler 22

6.3 Word comparison 26
6.4 Define center 27
6.5 Word comparison taking center into account 27

Bibliography **28**

List of Figures

3.1 Amazon webshop starting page . . . 7
3.2 JD webshop starting page . . . 7
3.3 Amazon review . . . 8
3.4 JD review . . . 8
3.5 Data retrieved from the JD webshop . . . 9
3.6 Our example separated according to the pku standard. . . . 10
3.7 Our example separated according to the ctb standard. . . . 10
3.8 Our example separated according to the pku standard transformed to a feature. . . . 10
3.9 Our example separated according to the ctb standard transformed to a feature. . . . 11

4.1 Confusion matrix experiment 1 . . . 14
4.2 Confusion matrix experiment 1 after tweaking parameters . . . 14
4.3 Confusion matrix experiment 2 with whole reviews. . . . 15
4.4 Confusion matrix experiment 2 with left part of reviews. . . . 15
4.5 Confusion matrix experiment 2 with right part of reviews. . . . 16
4.6 Confusion matrix experiment 3 . . . 17
4.7 Confusion matrix experiment 3 after tweaking parameters . . . 17

Chapter 1

Introduction

The Internet is becoming more and more accessable in the last few years. This allows people to post increasing amounts of information on the Web. Many companies and organisations are keen on understanding online user-generated texts. The field of sentiment mining focuses on the emotion hidden in a certain text. Texts can vary in many ways, one of these being language. Different languages have different rules and grammars. A lot of research has been done on analysing English texts. It is interesting to see to what extend we can use certain technique in a different language.

1.1 Case study

Our case study explores predicting the sentiment of reviews on two Chinese webshops. These reviews are written on the subject of diapers, due to the abundance of data in this area. The first webshop we look into is `www.jd.com`. This webshop is very popular in China and covers a lot of data. The other webshop is `www.amazon.cn`. This is the localized version of the American webshop `www.amazon.com`. This webshop contains less data, but the reviews are generally longer than the ones at JD. The reviews have the same structure on both webshops. Each consist of a rating compartment and a text compartment.

1.2 Problem statement

The problem we want to solve is a classification problem with 5 classes on a linear scale. The main question we will answer in this thesis is:

Research question 1: *Can we estimate the rating of a product review by looking at the sentiment of the text without having any contextual knowledge?*

To expand on this question we ask the following questions:

Research question 2: *Do we need a full review to estimate the assessment or is a smaller amount of text sufficient?*

Research question 3: *How accurately can we estimate the polarity of a review?*

To answer these questions we want to use the *bag of words* approach by using a lexicon to attach sentiment to Chinese characters. By combining the sentiment of these characters we might find the polarity of the whole text.

1.3 Methodology

Our research can be split in 2 phases: the acquisition of data and the experiments. In the data acquistion phase we retrieve the reviews from the webshops and transform the data so it can be used with the bag of words method. In the experiment phase we use a data processing tool to get our estimations.

1.3.1 Data acquisition

To acquire the data used for this research we make a web crawler which can navigate through the webshops like normal users and save any necessary information. Our crawler found 20240 reviews on Amazon and 41575 reviews on JD. Each of these reviews consists of a rating between 1 and 5 stars, and a text consisting of a string of Chinese characters.

We want to use machine learning to classify the reviews ourselves. However, machine learning algorithms don't accept texts. According to [Sebo2] a typical way to find features which such an algorithm understands, is to use a *bag of words* approach. Although this approach is quite basic, it does not perform significantly worse than more complicated methods. According to the bag of words approach we attach weights to certain words by comparing them to a sentiment dictionary.

An important difference between Chinese and Western languages is the use of spaces. In Western languages a space is used to separate two words from each other. The Chinese language has no need for spaces. Chinese words consist of one or more characters. Depending of the context certain characters may change its meaning and become an other word altogether.

In order to extract words from the strings of characters we use a string separator. The tool we use is the Stanford Word Segmenter [TCA+17] provided by the Stanford Natural Language Processing Group. This

tool uses strings of Chinese characters as an input and returns strings of the same characters with spaces inserted between the found words.

We compare the separated words with a Chinese sentiment dictionary [WK16]. This way we attach weights to the reviews, which can be used by the machine learning algorithm.

1.3.2 Experiments

We conduct three experiments to find out if we can estimate the rating of reviews by looking at the sentiment of the words existing in said reviews. To do this we use a data processing application called Weka. Weka is a tool that can modify and map data in a big amount of ways. We use the MultilayerPerceptron to classify our data.

In our first experiment we see how effective our approach is. We run the perceptron over all data with our target set between 1 and 5 stars. We see that on an evenly spread dataset this yields a success rate of 36%.

In our second experiment we want to find out if we can estimate the rating by using only half of the text of our reviews. When omiting half of the texts we get success rates of 31%, independent of which part we use.

In our third experiment we want to see if our method can be put to better use if we use it on a binary problem, instead of a classification problem with 5 classes. When running the perceptron over the data using the same parameters, we achieve a success rate of 78%.

1.4 Thesis Overview

Chapter 2 discusses related work which we can use to evaluate our results. Chapter 3 includes the data acquisition. Chapter 4 discusses our three experiments. Chapter 5 concludes.

Chapter 2

Related Work

In this chapter we discuss similar studies to be able to evaluate our findings. The first study is a 3 class classification problem on the Twitter domain.

A case study has been conducted on the domain of Twitter [PTAV15]. Data from twitter is comparable to our data due to the fact that Twitter messages use a limit of 140 characters, producing a small amount of words per message. In this study the classification problem is one with 3 classes. Using a bag of words method, when setting tresholds by hand they achieved success rates of 33%. Using machine learning algorithms they were able to improve this to 45%.

Another case study has been conducted on reviews on multiple domains, rating them positive and negative [ZZPH08]. By using a bag of words feature selection and various machine learning algorithms they were able to achieve success rates between 70% and 80%.

Chapter 3

Data acquisition

In this chapter we give an overview of the data and describe the steps taken to acquire the data used in this research. These steps include making a web crawler, the separation of strings into words and the comparison of the data derived from said crawler with a sentiment dictionary.

3.1 Data overview

The data used in this project are reviews of diapers on Chinese webshops. These were collected on two Chinese webshops, namely `http://www.jd.com/` and `https://www.amazon.cn/`. Amazon is the localized version of the international webshop Amazon.com. JD has Chinese origins. Both shops are widely used in China. A difference between the shops is that reviews on JD tend to be shorter and include more positive ratings than those on Amazon. See Figure 3.1 for an impression of the Amazon webshop, and Figure 3.2 for JD.

The reviews of the products consist of several parts. At both webshops they have a rating, text, product ID, date and a reviewer. At Amazon it is possible for other users to rate a review, or to add a response. At JD images can be attached, and the reviewers can have a score next to their name. See Figure 3.3 for a review at Amazon, and Figure 3.4 for one at JD. We are interested in two parts available at both webshops: a rating and actual text. The rating is on an integer scale of 1 to 5. The text contains mostly simplified Chinese characters and some numbers and western words. The charachters are Mandarin and simplified. The reviews are encoded in UTF-8.

Figure 3.1: Amazon webshop starting page

Figure 3.2: JD webshop starting page

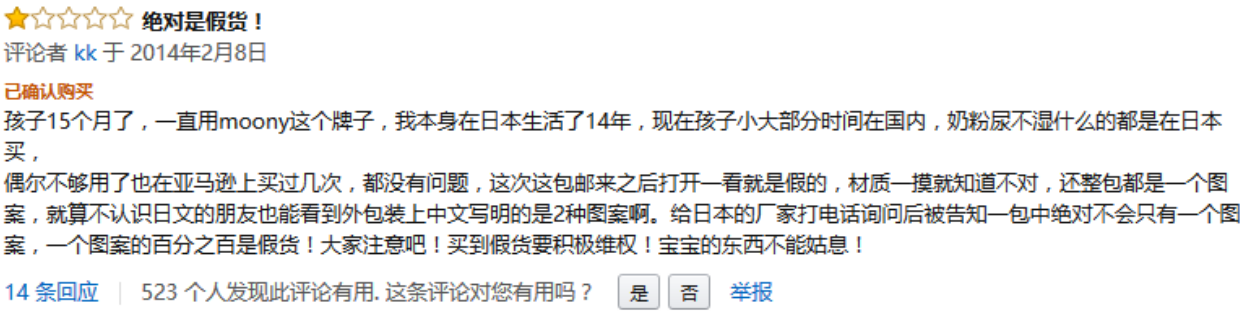

Figure 3.3: Amazon review

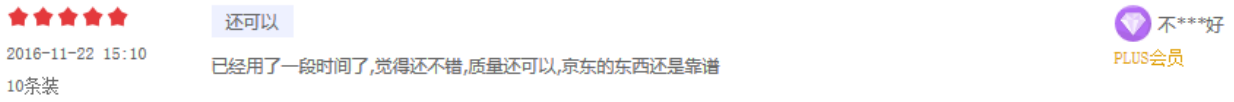

Figure 3.4: JD review

3.2 Web crawler

To get the data we made a web crawler using Java and the jsoup API [Hed]. jsoup is a Java HTML parser and can be used to extract and manipulate data using jquery-like methods. By using the jsoup *connect* method the content of a web page can be fetched and loaded in a *Document* class. The *Document* class allows us to access several HTML tags, such as *body* and *title*. *Document* is a subclass of the *Element* class. This class provides the convenient *select* method. Using this method we can quickly find HTML tags by searching for their id or class. The innerHTML of said selection can then be copied for later use.

A basic overview of the crawlers is described in Algorithm 1. The starting page, shown in Figure 3.1 and Figure 3.2, is an overview page at which the different products are shown. We use the jsoup *connect*-method to load the DOM of the page in a variable. Here it is important to seperate each product by class and to find the urls of these products. We do this by selecting the class of the product url using the jsoup *select*-method. Utilising basic *string*-methods we collect the url of the product. Then for each product the product page is parsed. On this page it is important to find the different reviews. We do this in a similar matter. When all of the reviews on this page are copied to an outputfile the next page of reviews is parsed until all reviews are collected.

Although we use this pseudocode for both web crawlers, there is still a major difference between the two. This is due to the fact that both webshops are loaded in a different way. Amazon loads a static amount of reviews on a productpage and uses a navigator to get to the next page of reviews. JD does not load products or reviews until a user scrolls down. Because our web crawlers parse the webpage on load, we don't scroll down. To solve this we load each review individualy by setting the amount of reviews per page to 1, and incrementing the page number after a review is seen. A new problem that arises is that we can not predict

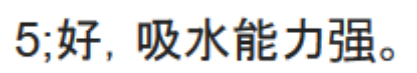

Figure 3.5: Data retrieved from the JD webshop

when all of the reviews have been collected. When this happens the program shuts down, but we get to keep the data.

The web crawlers were run in November of 2016. The Amazon dataset yields 20240 reviews. The JD dataset consists of 41720 reviews. The full code of the web crawlers can be found in Appendix 6.1 and Appendix 6.2.

Algorithm 1 Web crawler

```
loadStartingPage
for i:=0 to findAmountOfProduct do
        loadProductPage
        for j:=0 to findAmountOfReviewPages do
                loadReviewPage;
                for k:=0 to findAmountOfReviews do
                        copyText
                        copyRating
                end for
        end for
end for
```

We store the data with the rating, a semicolon and the text. Each line is a different review. An example of the data retrieved by the crawler can be seen in Figure 3.5. It has a rating of 5 stars. The text translates to *good, strong water absorption capacity*. We separate the rating and the text with a semicolon, as the Chinese language does not use these. Note that there are no spaces between Chinese characters.

3.3 String separation

Our machine learning tool does not support whole texts, but needs features. We want to create features out of the texts by using the *bag of words* approach. We want to attach values to the words of the review according to a sentiment dictionary. An important difference between Chinese and most Western languages is that Chinese words are not clearly seperated. In the English language words are separated by spaces, but Chinese sentences have no need for those. Chinese words can consist of one or more characters. Each character has its own meaning, but the meaning can change when it is combined with other characters. In order to attach sentiment to the words we have to separate the reviews at logical places.

We separate these strings into words by using the Stanford Word Segmenter provided by the Stanford Natural Language Processing Group [TCA+17]. This segmenter uses strings of Chinese characters as an input and returns strings of Chinese characters with spaces between words. It provides two models with different

Figure 3.6: Our example separated according to the pku standard.

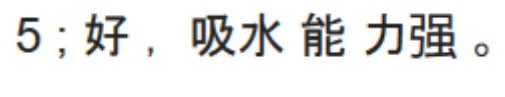

Figure 3.7: Our example separated according to the ctb standard.

segmentation standards, the Chinese Penn Treebank standard (ctb) and the Peking University standard (pku). We use both models to separate our data. Our example can be seen in Figure 3.6 and Figure 3.7.

3.4 Word comparison

To produce features which can be used by our data processing application Weka, we compare the words of each review with a sentiment dictionary. We use the Augmented NTU Sentiment Dictionary provided by the NLP Lab at Academia Sinica [WK16]. This context-free dictionary contains 27,221 simplified Mandarin words. Each word has a value for five categories, marking it positive, negative, neutral, non-opinionated or not-a-word. We use a Python script to count the values of each catergory for each word in each review. We also count how many words in each review were matched with the words in the vocabulary. This way we derive a feature vector for each review, containing a positive score, a neutral score, a negative score, a non-opinionated score, a not-a-word score and a rating. The script used to compare the words can be found in appendix 6.3.

The result of this comparison can be seen in Figure 3.8 and Figure 3.9. For our example the segmenter using pku standard found one match, giving the review 2 points in the *positive* category and 2 points in the *non opionion* category. The segmenter using ctb standard did not split the review on places so that the words matched the dictionary. We omit this feature vector from our dataset.

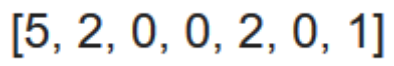

Figure 3.8: Our example separated according to the pku standard transformed to a feature.

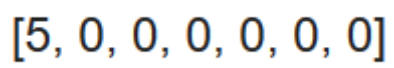

Figure 3.9: Our example separated according to the ctb standard transformed to a feature.

Chapter 4

Experiments

In this chapter we set out the preprocessing and decisions made over the data used by all experiments. Then, for each experiment we state the remaining preprocessing actions needed and the results gained.

For this thesis we conduct three experiments. In the first experiment we look to what extend we can properly estimate the rating of a given review. In the second experiment we find out if we can improve the estimation by adding alignment information to the data. In the third experiment we explore optimization by turning the rating in a binary target. The classifier used for each experiment is the MultilayerPerceptron provided by Weka.

4.1 Preprocessing

The word comparison script provided us with typical Python matrix output. These are recognized by the square brackets around instances and commas in between data points. We want to convert these files to Attribute-Relation File Format (arff) files which can be used by Weka, our data processing application. Arff files consist of a header followed by the data points. The header contains the name of the relation, the names of the attributes and their types. The instances of data is separated by newlines. The data points in each instance are seperated by commas. The target of each instance is placed in the rightmost column. To transform our data to the arff format we simply omit the brackets, and then move the column containing the rating to the right.

Another matter that needs to be addressed is the fact that the word comparison script left a lot of instances untouched. As these instances provide no useful information, we remove them from our dataset.

The third thing that is needed for every experiment is resampling our dataset for building the perceptron

model. The data we derived for a large part exists of reviews with a rating of 5. If we would not adjust the set for building the model we can get the risk of overfitting to a score of 5. Weka provides a resample filter over the data. By default, the biasToUniformClass parameter is set to 0. This parameter controls the dispersion over the target class. By setting this parameter to 1 we acquire a set with evenly spread targets.

4.2 Experiment 1

In the first experiment we explore with what precision we can estimate the rating of a review using the sentiment of the words in this review. The data we use is the resampled combination of Amazon and JD reviews, with both of the separation modes combined. This makes a big dataset consisting of 107805 instances. We use the Weka MultilayerPerceptron classifier to estimate the ratings of our reviews. The most important parameters we can change are the distribution of hidden layers, the learning rate, the momentum and the number of epochs. By default, the perceptron is made with 1 hidden layer of 5 nodes, a learning rate of 0.3, a momentum of 0.2, and 500 epochs. If we run the classifier with the default parameters, we achieve a success rate of 34% and a confusion matrix as shown in Figure 4.1. We can see that the classifier almost never classifies a review as one with 2 or 4 stars. It is possible that the model finds local maxima and minima. To reduce this chance we decrease the learning rate and momentum and increase the number of epochs. We set the learning rate and momentum both to 0,1. We set the number of epochs to 5000. This increases the success rate to 36% and yields a confusion matrix as shown in Figure 4.2.

4.3 Experiment 2

In the second experiment we want to see if we can estimate our target score by viewing only half of our reviews. We remodel our data so that we can see if the labeled words appear at the start or the end of their review. We do this by first finding the center position in each review and then comparing the reviews with the word list as done in the Word comparison section in chapter 3. This gives us vectors that are twice as large, with the labels positiveleft, neutralleft, negativeleft, nonopinionleft, nonregularleft, positiveright, neutralright, negativeright, nonopinionright, nonregularright and the target rating. As with experiment 1 we start out by using the Weka MultilayerPerceptron classifier with the default parameters. This results in a success rate of 35% and a confusion matrix as seen in Figure 4.3.

Now when we omit the right half of the text in our reviews our success rate will decrease to 31% with a confusion matrix as shown in Figure 4.4. We get a similar result when we omit the left half of 31% and a confusion matrix as shown in Figure 4.5.

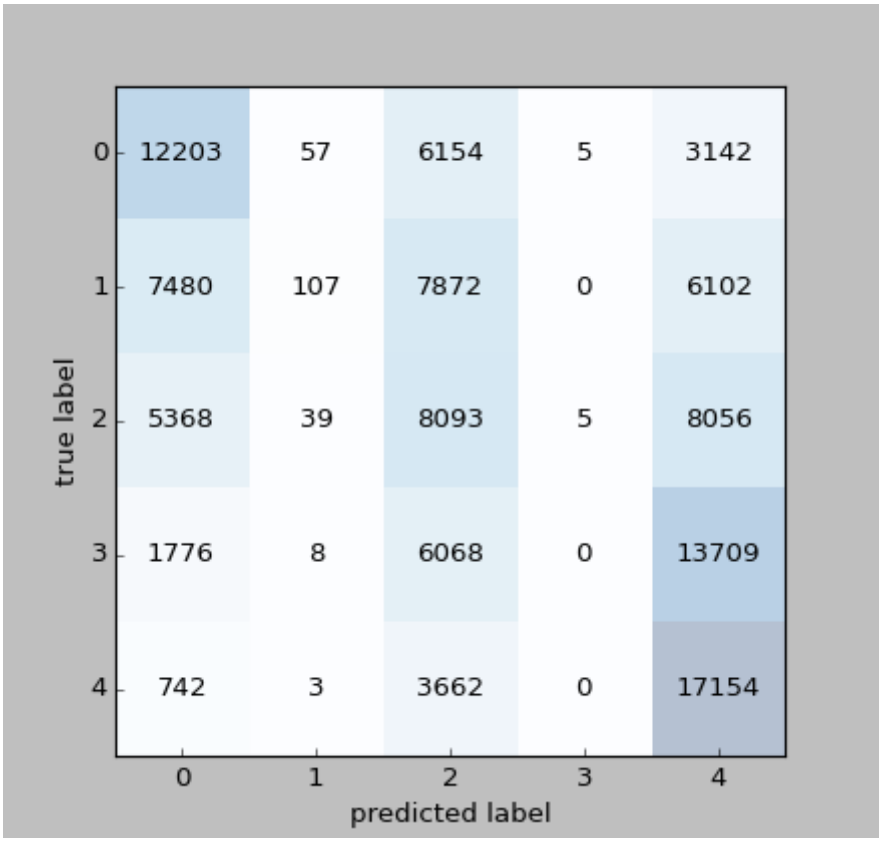

Figure 4.1: Confusion matrix experiment 1

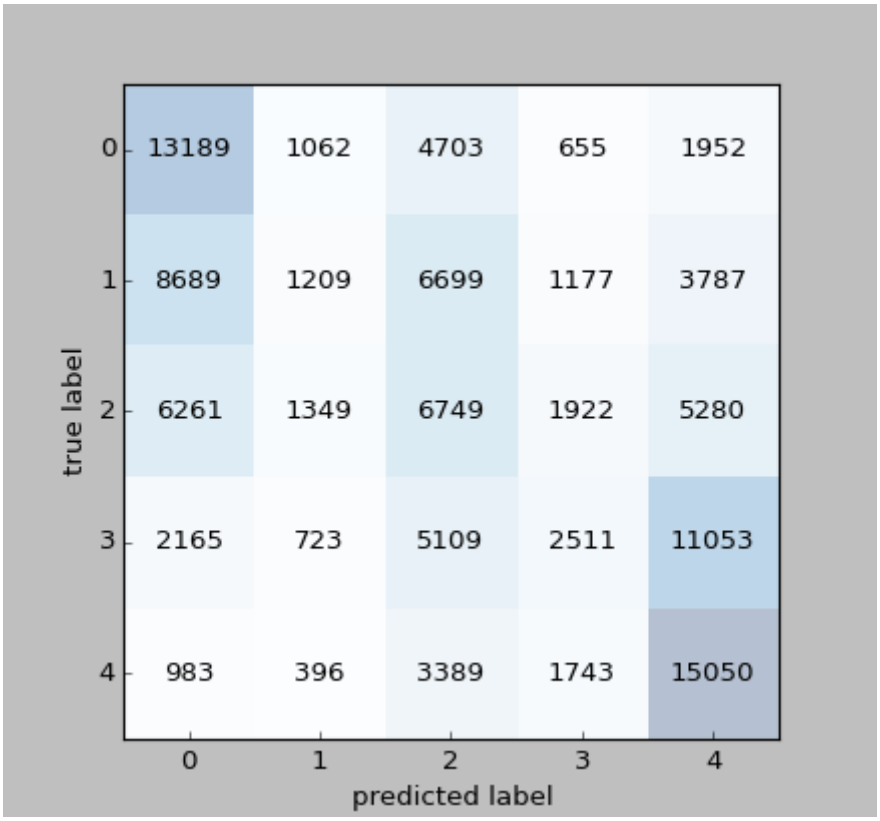

Figure 4.2: Confusion matrix experiment 1 after tweaking parameters

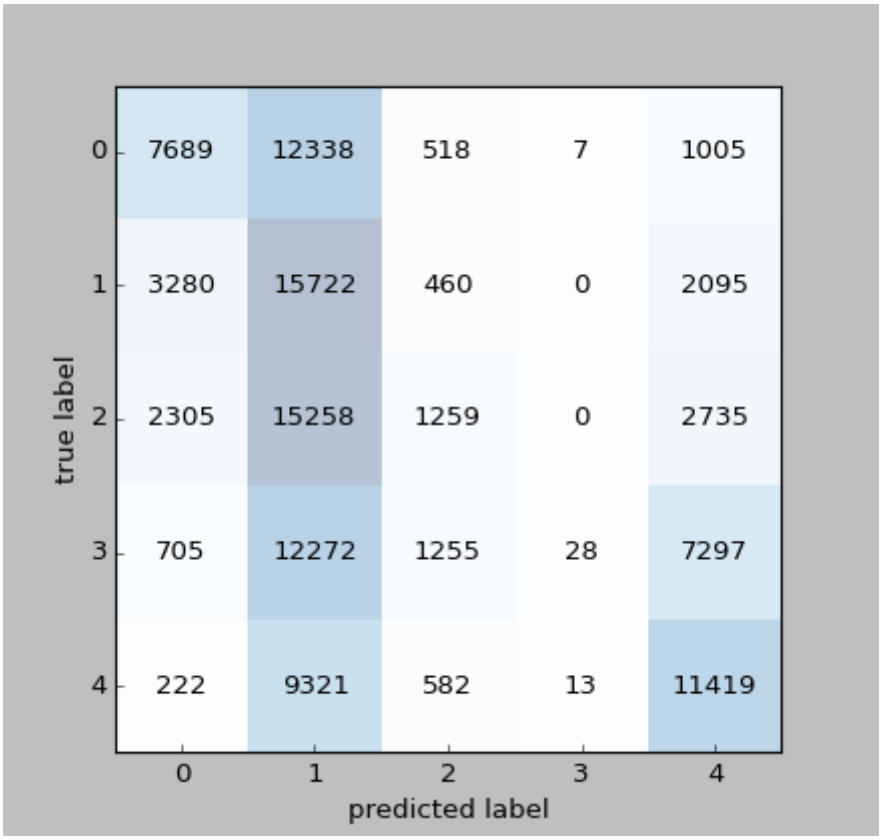

Figure 4.3: Confusion matrix experiment 2 with whole reviews.

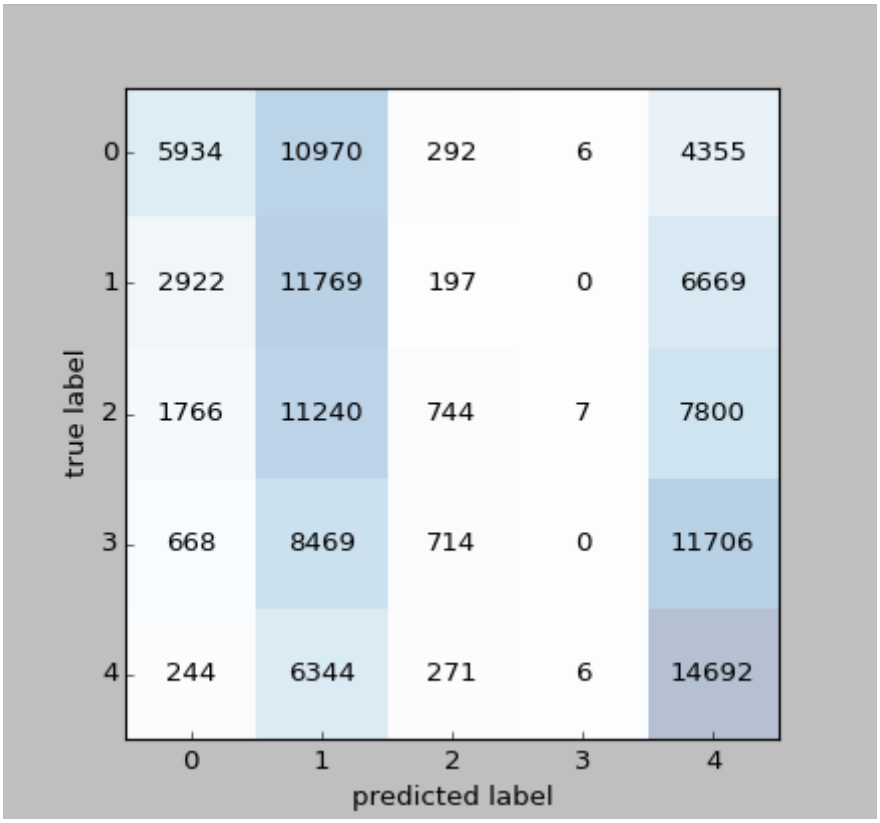

Figure 4.4: Confusion matrix experiment 2 with left part of reviews.

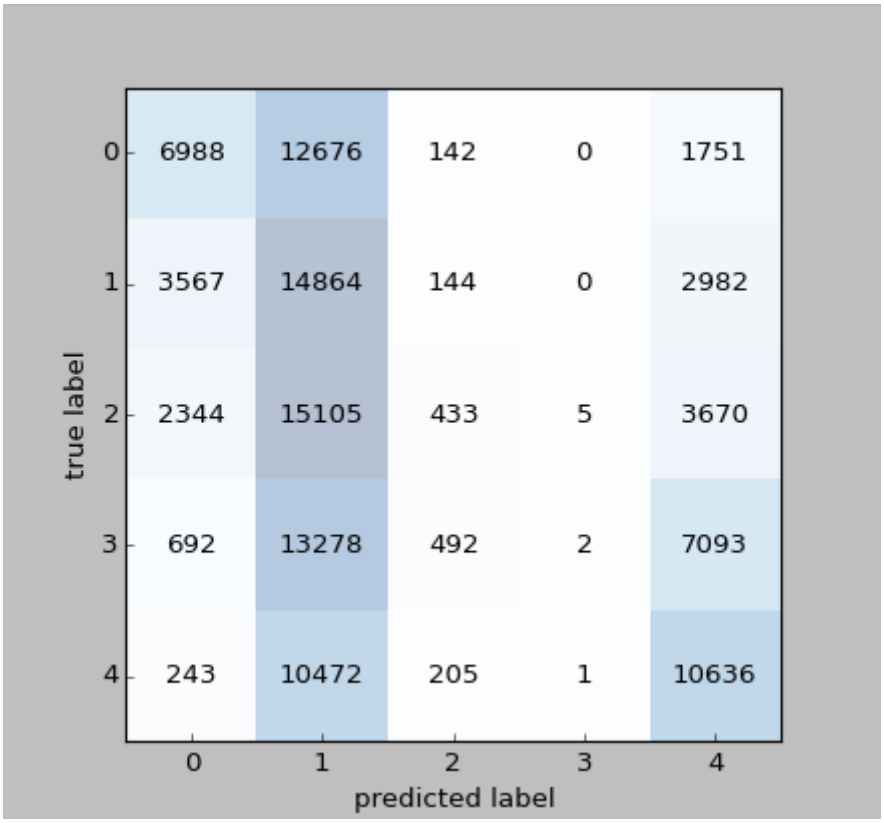

Figure 4.5: Confusion matrix experiment 2 with right part of reviews.

4.4 Experiment 3

As both of the previous experiments ignored multiple classes, we can try to improve our success rate by changing our problem in one with a binary target. Note that for transforming the original files to files with a binary target we need to choose where to draw the boundary between 0 and 1. As the original data files are tilted towards a rating of 5, we opt for a boundary between 3 and 4, making the portion of 0 a bit more in proportion to the portion of 1. After preprocessing, the dataset contains 50% instances with a target of 0, and 50% instances with a target of 1. Just like the previous experiments we run the Weka MultilayerPerceptron classifier with the default parameters. This results in a success rate of 78% and a confusion matrix as shown in Figure 4.6. We can see that the error on both sides is quite similar. Although there are less targets now it is still possible that the perceptron has landed on some local maxima or minima. To see if we can move past these we decrease the learning rate and momentum again to 0.1 and increase the number of epochs to 5000. This results in a success rate of 78% and a confusion matrix as shown in Figure 4.7.

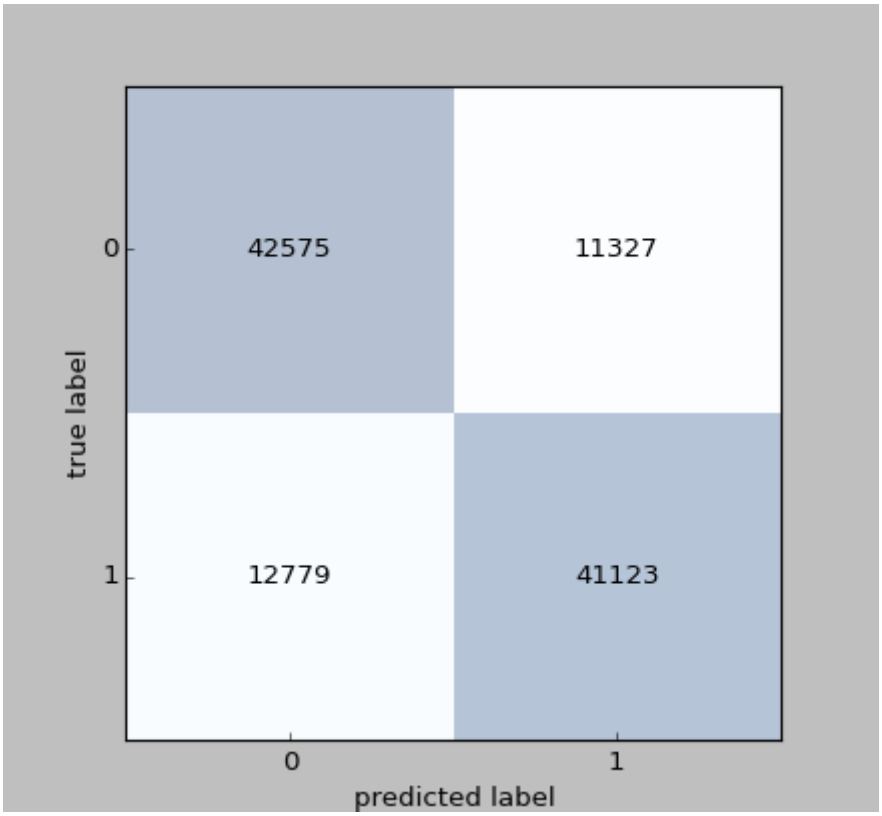

Figure 4.6: Confusion matrix experiment 3

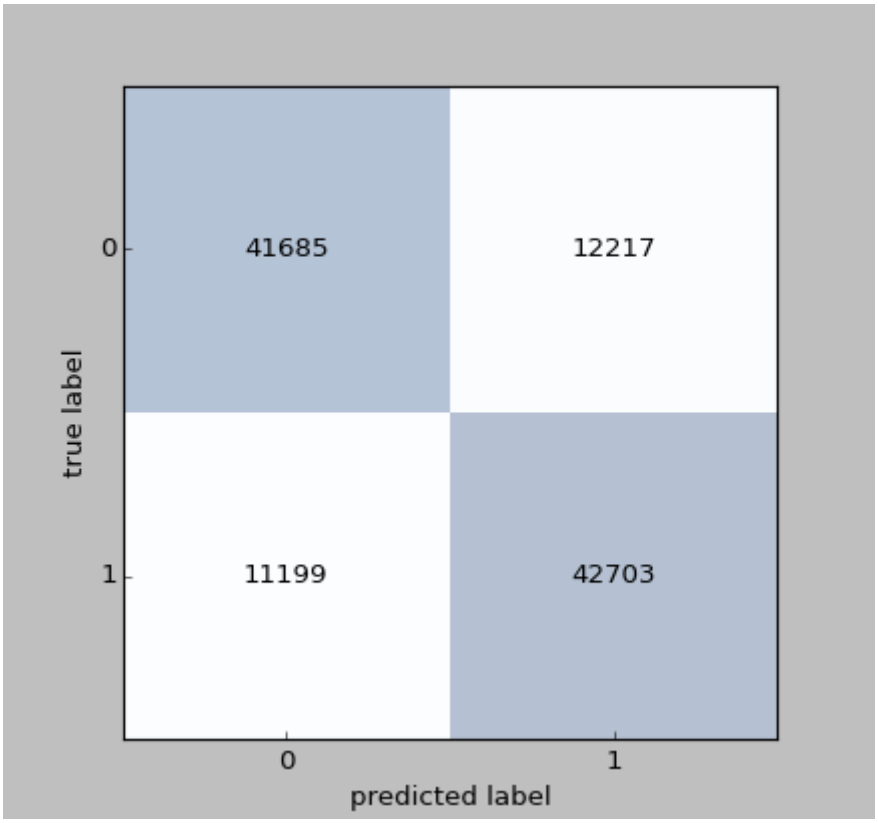

Figure 4.7: Confusion matrix experiment 3 after tweaking parameters

Chapter 5

Conclusions

In this chapter we conclude by answering our research questions and comparing our results to related work.

Research question 1: *Can we estimate the rating of a product review by looking at the sentiment of the text without having any contextual knowledge?*

Although it is not enough to accurately predict even half of the ratings, we clearly see that using a bag of words method in combination with a multilayer perceptron on evenly spread data increases the success rate to 34%. Although initial adjustments improve our results slightly, we see that the model still skips 2 of the classes most of the time. If we compare these results with [PTAV15] we can see that an improvement on a problem with less classes is similar to ours. If we want to correctly estimate the rating of a product review, we need another solution. Looking at structure of sentences might be an enrichment.

Research question 2: *Do we need a full review to estimate the assessment or is a smaller amount of text sufficient?*

When we know the position of positive or negative words we do not get an improvement, nor a decline in our success rate in comparison with when we don't look at position at all. When we omit half of our reviews though, we clearly get a decline in our success rate. In this case it does not matter if we keep the left part or the right part.

Research question 3: *How accurately can we estimate the polarity of a review?*

When we bring our problem back to a binary problem, we yield much better results. We achieve a success rate of 78%. When we compare these results to [ZZPH08] we see that ours are as can be expected.

Chapter 6

Appendix

6.1 Amazon Webcrawler

This is the script used to retrieve the reviews from Amazon. It is made in Java. We use a Mozilla Firefox user agent to mimic a normal web browser. We load the DOM elements of the starting webpage. We select the url for each product on this page. Then for each product, we load the DOM elements of their page. Here we select the url for the review page. We load the DOM elements of the review page. For each review page each review is written to an output file.

```
package crawler;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class AmazonCrawler{

   public static String ua = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36
       (KHTML, like Gecko) Chrome/35.0.1916.47 Safari/537.36";
```

```
public static void visitProductPage(String pageURL, Writer writer){
   try {
      Document doc2 = Jsoup.connect(pageURL).timeout(50000).userAgent(ua).get();
      Element productTitleElement = doc2.select("span#productTitle").first();
      if(productTitleElement != null){
         String productTitleRaw = productTitleElement.toString();
         int beginIndex = productTitleRaw.indexOf(">") + 1;
         int endIndex = productTitleRaw.indexOf("</");
         String productTitle = productTitleRaw.substring(beginIndex, endIndex);
         Element reviewPage = doc2.select("a.a-link-emphasis.a-text-bold").first();
         if(reviewPage != null){
            String reviewPageRaw = reviewPage.toString();
            int beginIndex2 = reviewPageRaw.indexOf("href") + 6;
            int endIndex2 = reviewPageRaw.indexOf(">") - 1;
            String reviewPageURL = reviewPageRaw.substring(beginIndex2, endIndex2);
            Document doc3 = Jsoup.connect(reviewPageURL).timeout(50000).userAgent(ua).get();

            Elements reviewBlocks = doc3.select("div.a-section.review");

            Element pagination = doc3.select("ul.a-pagination").first();

            if(pagination != null){
               String paginationRaw = pagination.toString();
               int endIndex3 = paginationRaw.indexOf("a-last") - 22;
               int beginIndex3 = endIndex3 - 1;
               while(Character.isDigit(paginationRaw.charAt(beginIndex3 - 1)))
                  --beginIndex3;
               int numberOfPages = Integer.parseInt(paginationRaw.substring(beginIndex3, endIndex3));

               for(int i = 2; i <= numberOfPages; ++i){
                  String targetURL = reviewPageURL;
                  targetURL =
                      targetURL.concat("&reviewerType=all_reviews&pageNumber=").concat(Integer.toString(i));
                  doc3 = Jsoup.connect(targetURL).timeout(50000).userAgent(ua).get();
                  reviewBlocks.addAll(doc3.select("div.a-section.review"));
               }
            }
```

```
            int numberOfReviews = reviewBlocks.size();


            for(int i = 0; i < numberOfReviews; ++i){
               if(reviewBlocks.get(i) != null){
                  String reviewBlockRaw = reviewBlocks.get(i).toString();


                  int beginIndex4 = reviewBlockRaw.indexOf("span class=\"a-icon-alt") + 24;
                  int endIndex4 = beginIndex4 + 3;
                  String rating = reviewBlockRaw.substring(beginIndex4, endIndex4);


                  int beginIndex5 = reviewBlockRaw.indexOf("span class=\"a-size-base review-text") +
                      37;
                  int endIndex5 = reviewBlockRaw.indexOf("<div class=\"a-row a-spacing-top-small
                      review-comments\">") - 17;
                  String review = reviewBlockRaw.substring(beginIndex5, endIndex5);


                  writer.write(rating + ";" + review + "\r\n");
               }
            }


         }
      }
   } catch (IOException e) {
      e.printStackTrace();
   }

}


public static void fetchPage(String baseURL, Writer writer){
   Document doc = null;
   try {
      for(int i = 1; i <= 20; ++i){
         String pageURL = baseURL;
         if(i>1)
            pageURL = pageURL.concat("&page=").concat(Integer.toString(i));
         doc = Jsoup.connect(pageURL).timeout(50000).userAgent(ua).get();
         Elements productLinks = doc.select("a.a-link-normal.s-access-detail-page.a-text-normal");
         int numberOfProductLinks = productLinks.size();
```

```
            for(int j = 0; j < numberOfProductLinks; ++j){
               String thisProduct = productLinks.get(j).toString();
               int beginIndex = thisProduct.indexOf("href") + 6;
               int endIndex = thisProduct.indexOf("h2") - 3;
               String thisProductURL = thisProduct.substring(beginIndex, endIndex);
               visitProductPage(thisProductURL, writer);
            }
         }

      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   public static void main(String[] args) {
      String startURL =
          "http://www.amazon.cn/s/ref=nb_sb_noss_1?__mk_zh_CN=%E4%BA%9A%E9%A9%AC%E9%80%8A%E7%BD%91%E7%AB%99"
           + "&url=search-alias%3Daps&field-keywords=%E5%B0%BF%E5%B8%83";

      try {
         FileWriter writer = new FileWriter("amazondata.txt");
         fetchPage(startURL, writer);
         writer.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

}
```

6.2 JD Webcrawler

This is the script used to retrieve the reviews from JD. As with the Amazon crawler, it is made in Java. We use a Mozilla Firefox user agent to mimic a normal web browser. We load the DOM elements of the starting webpage. We select the url for each product on this page. Now for each product we find the url for a

review. We set the amount of reviews per page to 1, and increment the pagenumber until we have collected all reviews. The reviews are written to an output file.

```
package crawler;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;

public class JDCrawler{

   public static String ua = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36
       (KHTML, like Gecko) Chrome/35.0.1916.47 Safari/537.36";

   public static boolean visitCommentPage(String commentURL, String productId, Writer writer){
      try {
         Document doc2 = Jsoup.connect(commentURL).timeout(50000).userAgent(ua).get();
         String commentInfo = doc2.toString();
         int index = commentInfo.indexOf(",\"score\":") + 9;
         int beginIndex = commentInfo.indexOf(",\"score\":", index + 1) + 9;
         int eindIndex = beginIndex + 1;
         if(beginIndex == 8)
            return false;

         String rating = commentInfo.substring(beginIndex, eindIndex);

         int beginIndex2 = commentInfo.indexOf(",\"content\":") + 12;
         int eindIndex2 = commentInfo.indexOf(",\"creationTime\":") - 1;
         String content = commentInfo.substring(beginIndex2, eindIndex2);

         writer.write(rating + ";" + content + "\r\n");


      } catch (IOException e) {
         e.printStackTrace();
```

```
    }
    return true;
}

public static void scrapeGoodComments(String productId, Writer writer){
    String commentBaseURL = "http://sclub.jd.com/comment/productPageComments.action?productId=";
    commentBaseURL = commentBaseURL.concat(productId);
    commentBaseURL = commentBaseURL.concat("&score=3&sortType=6&pageSize=1");
    boolean guard = true;
    int i = 0;
    while(guard){
        String commentURL = commentBaseURL.concat("&page=").concat(Integer.toString(i));
        guard = visitCommentPage(commentURL, productId, writer);
        ++i;
    }
}

public static void scrapeAverageComments(String productId, Writer writer) {
    String commentBaseURL = "http://sclub.jd.com/comment/productPageComments.action?productId=";
    commentBaseURL = commentBaseURL.concat(productId);
    commentBaseURL = commentBaseURL.concat("&score=2&sortType=6&pageSize=1");
    boolean guard = true;
    int i = 0;
    while(guard){
        String commentURL = commentBaseURL.concat("&page=").concat(Integer.toString(i));
        guard = visitCommentPage(commentURL, productId, writer);
        ++i;
    }
}

public static void scrapeBadComments(String productId, Writer writer) {
    String commentBaseURL = "http://sclub.jd.com/comment/productPageComments.action?productId=";
    commentBaseURL = commentBaseURL.concat(productId);
    commentBaseURL = commentBaseURL.concat("&score=1&sortType=6&pageSize=1");
    boolean guard = true;
    int i = 0;
    while(guard){
        String commentURL = commentBaseURL.concat("&page=").concat(Integer.toString(i));
```

```
            guard = visitCommentPage(commentURL, productId, writer);
            ++i;
        }
    }

    public static void fetchPage(String baseURL, Writer writer){
        Document doc = null;
        try {
            String pageURL = baseURL;
            doc = Jsoup.connect(pageURL).timeout(50000).userAgent(ua).get();
            Elements productLinks = doc.select("div.gl-i-wrap");
            int numberOfProductLinks = productLinks.size();

            for(int j = 0; j < numberOfProductLinks; ++j){
                String thisProduct = productLinks.get(j).toString();
                int beginIndex = thisProduct.indexOf("href") + 6;
                int endIndex = thisProduct.indexOf("onclick") - 2;
                String thisProductURL = thisProduct.substring(beginIndex, endIndex);

                if((!(thisProductURL.substring(0,1)).equals("h"))){
                    int beginIndex1 = thisProductURL.indexOf("com") + 4;
                    int eindIndex1 = thisProductURL.indexOf("html") - 1;
                    String thisProductId = thisProductURL.substring(beginIndex1, eindIndex1);
                    scrapeGoodComments(thisProductId, writer);
                    scrapeAverageComments(thisProductId, writer);
                    scrapeBadComments(thisProductId, writer);
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        String startURL = "http://search.jd.com/Search?keyword=%E5%B0%BF%E5%B8%83&enc=utf-8"
                + "&wq=%E5%B0%BF%E5%B8%83&pvid=ybo2nfti.a6bsdc";
```

```
      try {
         FileWriter writer = new FileWriter("jddata.txt");
         fetchPage(startURL, writer);
         writer.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

}
```

6.3 Word comparison

This Python script is used to compare every word in our data file with every word in the word list file. When a match is found, the values attached to the word in the word list are added to the values belonging to the review. The output is printed to the command line.

```
import sys

matrix = []

with open(sys.argv[1], "r") as d: #data file
  with open(sys.argv[2], "r") as w: #word list file
    for line in d:
      instance = [0, 0, 0, 0, 0, 0, 0] #rating, positive, neutral, negative, no opinion, no word,
          number of words
      for word in line.split():
        if instance[0] == 0:
          instance[0] = int(word)
        else:
          w.seek(0)
          for row in w:
            if(word == row.split(",")[0]):
              instance[1] += int(row.split(",")[2])
              instance[2] += int(row.split(",")[3])
              instance[3] += int(row.split(",")[4])
              instance[4] += int(row.split(",")[5])
```

```
            instance[5] += int(row.split(",")[6])
            instance[6] += 1
    print(instance)
```

6.4 Define center

This Python script is used to find the most centered position in each review. The string *center* is inserted so we can add information about position to the reviews.

```
import sys

with open(sys.argv[1], "r") as d: #data file
  for line in d:
    words = line.split()
    center = int(len(words)/2) + 1
    words.insert(center, "center")
    print(str(words))
```

6.5 Word comparison taking center into account

This Python script is similar to the Word comparison script, with the addition that it stores data belonging to words before the string *center* in the left of the output, and words after *center* to the right.

```
import sys

with open(sys.argv[1], "r") as d: #data file
  with open(sys.argv[2], "r") as w: #word list file
    for line in d:
      instance = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #rating, positivel, neutrall, negativel, no
          opinionl, no wordl, number of wordsl, positiver, neutralr, negativer, no opinionr, no
          wordr, number of wordsr
      left = True
      for word in line.split():
        if instance[0] == 0:
          instance[0] = int(word)
```

```
    elif word == "center":
      left = False
    else:
      w.seek(0)
      for row in w:
        if(word == row.split(",")[0] and left):
          instance[1] += int(row.split(",")[2])
          instance[2] += int(row.split(",")[3])
          instance[3] += int(row.split(",")[4])
          instance[4] += int(row.split(",")[5])
          instance[5] += int(row.split(",")[6])
          instance[6] += 1
        elif(word == row.split(",")[0]):
          instance[7] += int(row.split(",")[2])
          instance[8] += int(row.split(",")[3])
          instance[9] += int(row.split(",")[4])
          instance[10] += int(row.split(",")[5])
          instance[11] += int(row.split(",")[6])
          instance[12] += 1
  print(instance)
```

Bibliography

[Hed] Jonathan Hedley. jsoup api. `https://jsoup.org/apidocs/`.

[PTAV15] Evangelos Psomakelis, Konstantinos Tserpes, Dimosthenis Anagnostopoulos, and Theodora A. Varvarigou. Comparing methods for twitter sentiment analysis. *CoRR*, abs/1505.02973(1), 2015.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1), 2002.

[TCA+17] Huihsin Tseng, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. A conditional random field word segmenter for sighan bakeoff 2005. 2017.

[WK16] Shih-Ming Wang and Lun-Wei Ku. Antusd: A large chinese sentiment dictionary. 2016.

[ZZPH08] Changli Zhang, Wanli Zuo, Tao Peng, and Fengling He. Sentiment classification for chinese reviews using machine learning methods based on string kernel. 2008.