Text Sentiment Classification (I): Traditional Models

Jianlin Su

June 22, 2015

Preface: In April and May, I participated in two data mining competitions: the "Liangjian Cup" organized by the School of Physics and Electronic Engineering, and the 3rd "Teddy Cup" National Undergraduate Data Mining Challenge. Coincidentally, both competitions included a problem primarily involving Chinese text sentiment classification. When doing the "Liangjian Cup," as I was still a beginner with limited skills, I only implemented a simple sentiment classification model based on traditional ideas. In the subsequent "Teddy Cup," due to deeper study, I had already gained a basic understanding of Deep Learning and implemented a sentiment classification model using Deep Learning algorithms. Therefore, I plan to post both models on this blog for readers' reference. Beginners can compare the differences between the two and understand the relevant ideas. Experts, please feel free to ignore.

Based on Sentiment Lexicon

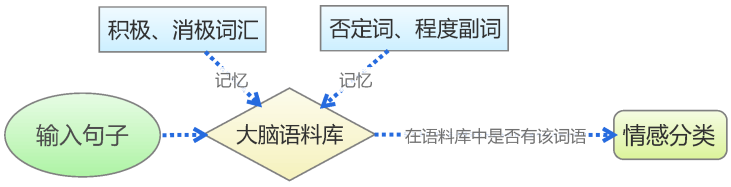

Figure 1: The simplest human judgment logic

Traditional text sentiment classification based on a sentiment lexicon is the simplest simulation of human memory and judgment logic, as shown in the figure above. First, we learn and memorize some basic vocabulary, such as negation words like "not," positive words like "like" and "love," and negative words like "dislike" and "hate," forming a basic corpus in our brain. Then, we perform a direct split of the input sentence to see if the words we memorized exist in the corresponding lexicon, and judge the sentiment based on the category of these words. For example, in "I like mathematics," the word "like" is in our memorized positive lexicon, so we judge it as having positive sentiment.

Based on this logic, we can implement lexicon-based sentiment classification through the following steps: preprocessing, word segmentation, sentiment lexicon training, and judgment. The entire process is shown in the figure below. The raw materials used to test the model include comments on Mengniu milk provided by Professor Xue Yun, as well as mobile phone review data purchased online (see attachment).

Text Preprocessing

Original corpora obtained by web crawlers usually contain information we don't need, such as extra HTML tags, so the corpus needs to be preprocessed. The Mengniu milk reviews provided by Professor Xue Yun are no exception. Our team used Python as our preprocessing tool, utilizing libraries like Numpy and Pandas, with regular expressions as the primary text tool. After preprocessing, the raw corpus is standardized as shown in the table below, where we use -1 to mark negative sentiment comments and 1 to mark positive sentiment comments.

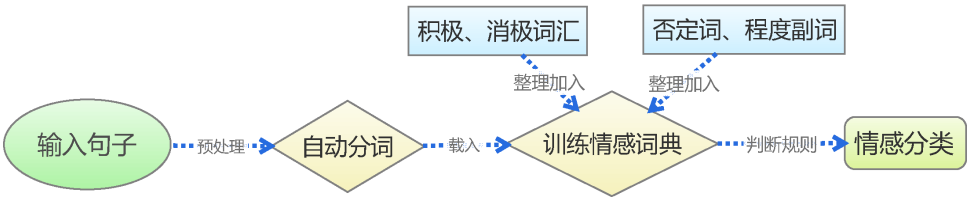

Figure 2: Text sentiment classification based on sentiment lexicon

	comment	mark
0	Mengniu is out embarrassing itself again	-1
1	Cherish life, stay away from Mengniu	-1
⋮	⋮	⋮
1171	I have always loved drinking Mengniu pure milk, always, very much	1
1172	Giving Mengniu... health is the best gift.	1
⋮	⋮	⋮

Automatic Word Segmentation

To determine whether a sentence contains words from the sentiment lexicon, we need to accurately cut the sentence into individual words, i.e., automatic word segmentation. We compared existing segmentation tools and, considering both accuracy and ease of use on the Python platform, finally chose "Jieba" as our segmentation tool.

The following table shows the segmentation results of various common tools on a typical test sentence:

> **Test Sentence:** The female secretary of the Industry and Information Technology Department personally handed over the installation of 24-port switches and other technical devices after passing through the subordinate departments every month.

Tool	Result
Jieba	Industry and Information Technology Dept/ Female/ Secretary/ Monthly/ Passing through/ Subordinate/ Department/ All/ Must/ Personally/ Hand over/ 24/ Port/ Switch/ Etc./ Technical/ Device/ 's/ Installation/ Work
ICTCLAS	Industry/n Information/n Dept/n Female/n Secretary/n Monthly/r Passing through/p Subordinate/v Department/n All/d Must/v Personally/d Hand over/v 24/m Port/q Switch/n Etc./udeng Technical/n Device/n 's/ude1 Installation/vn Work/vn
smallseg	Industry Info/ Info Dept/ Female Secretary/ Monthly/ Passing through/ Subordinate/ Department/ Must all/ Personally/ Hand over/ 24/ Port/ Switch/ Etc./ Technical/ Device/ 's/ Installation/ Work
Yaha	Industry and Information Technology Dept / Female / Secretary / Monthly / Passing through / Subordinate / Department / All / Must / Personally / Hand over / 24 / Port / Switch / Etc. / Technical / Device / 's / Installation / Work

Loading the Sentiment Lexicon

Generally speaking, the lexicon is the core part of text mining, and sentiment classification is no exception. The sentiment lexicon is divided into four parts: positive sentiment lexicon, negative sentiment lexicon, negation lexicon, and degree adverb lexicon. To obtain a more complete lexicon, we collected several sentiment lexicons from the internet, integrated and de-duplicated them, and adjusted some words to achieve the highest possible accuracy.

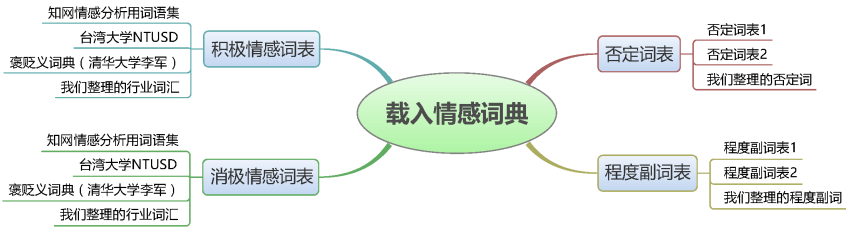

Figure 3: Building the sentiment lexicon

Our team did not simply integrate lexicons collected online; we also cleaned and updated them purposefully. Specifically, we added certain industry-specific terms to increase the hit rate. Word frequencies for certain terms vary significantly across industries, and these words might be key to sentiment classification. For example, the Mengniu milk data is from the food and beverage industry; in this industry, words like "eat" and "drink" appear frequently and usually imply positive evaluations, while "don't eat" or "don't drink" usually imply negative evaluations. In other fields, these words may have no obvious sentiment. Another example is the mobile phone industry, where "drop-resistant" or "waterproof" are positive terms. Therefore, it is necessary to consider these factors in the model.

Text Sentiment Classification

The rules for lexicon-based sentiment classification are quite mechanical. For simplicity, we assign a weight of 1 to each positive word and -1 to each negative word, assuming that sentiment values satisfy the principle of linear superposition. We segment the sentence; if the resulting word vector contains corresponding words, we add the weight. Negation words and degree adverbs have special rules: negation words flip the sign of the weight, and degree adverbs double the weight. Finally, the sentiment is judged based on the sign of the total weight. The basic algorithm is shown in the flowchart.

It should be noted that for the feasibility of programming and testing, we made several assumptions (simplifications). Assumption 1: We assume all positive and negative words have equal weights. This only holds in simple cases; in more precise classification, "hate" is clearly more severe than "dislike." A fix is to assign different weights to each word, which we will explore in the second part. Assumption 2: We assume weights are linearly additive. This holds in most cases, but we will explore non-linearity in the second part to enhance accuracy. Assumption 3: For negation and degree adverbs, we only perform simple inversion and doubling. In reality, different adverbs have different intensities (e.g., "very much like" is stronger than "quite like"), but we did not distinguish them.

We chose Python as the implementation platform. Thanks to Python's rich extension support, we implemented all the above steps in less than a hundred lines of code, resulting in an effective sentiment classification algorithm, demonstrating Python's conciseness. Next, we will verify the effectiveness of our algorithm.

Model Result Verification

As a basic test, we first applied our model to the Mengniu milk reviews. The results were satisfactory, reaching an accuracy of 82.02%. The detailed report is in the table below:

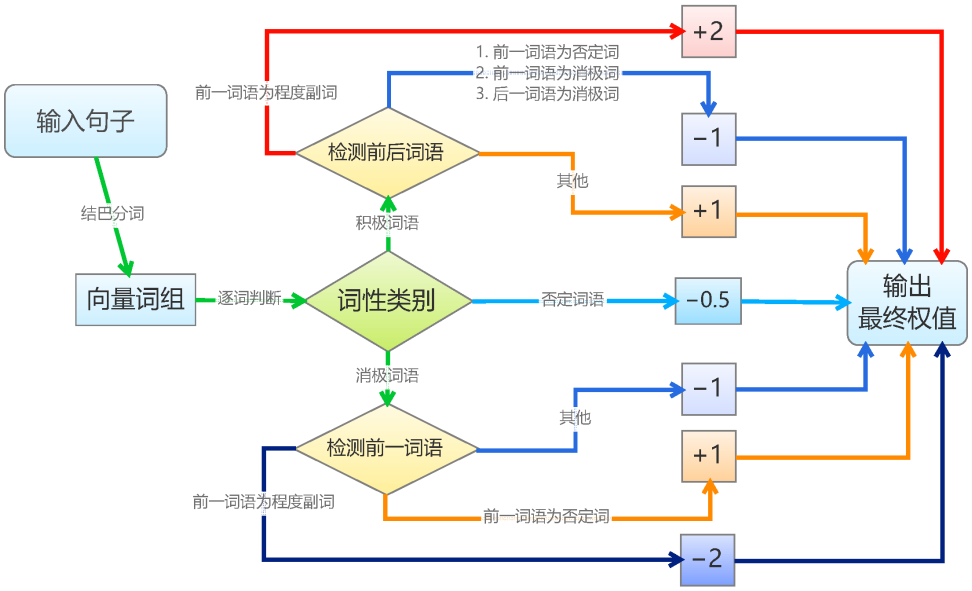

Figure 4: Flowchart of lexicon-based text classification

Data Content	Pos. Samples	Neg. Samples	Accuracy	TPR	TNR
Milk Reviews	1005	1170	0.8202	0.8209	0.8197

(Where positive samples are positive sentiment reviews, negative samples are negative sentiment data,

$$\text{Accuracy} = \frac{\text{Correctly judged samples}}{\text{Total samples}}$$

$$\text{True Positive Rate (TPR)} = \frac{\text{Positive samples judged as positive}}{\text{Total positive samples}}$$

$$\text{True Negative Rate (TNR)} = \frac{\text{Negative samples judged as negative}}{\text{Total negative samples}}$$

)

To our surprise, applying the model adjusted for milk reviews directly to mobile phone reviews also yielded an 81.96% accuracy! This indicates that our model has good robustness and performs well across different industries.

Data Content	Pos. Samples	Neg. Samples	Accuracy	TPR	TNR
Phone Reviews	1158	1159	0.8196	0.7539	0.8852

Conclusion: Our team initially implemented a lexicon-based text sentiment classification model. Test results show that simple judgment rules can achieve decent accuracy and robustness. Generally, a model with over 80% accuracy has production value and is suitable for industrial environments. Clearly, our model has reached this standard.

Difficulties

After two tests, we can conclude that our model's accuracy is basically above 80%. Mature commercial programs (like BosonNLP) have accuracies around 85% to 90%. This shows our simple model is effective, but also suggests that traditional lexicon-based models have limited room for improvement due to the inherent complexity of sentiment classification. We identified several difficulties:

Complexity of the Language System

Ultimately, this is because the language system in our brain is extremely complex. (1) We are doing text sen

Therefore, sentiment classification is an attempt to simulate human thinking. Our model simulates simple patterns, but real sentiment judgment is a complex network, not just simple rules.

The Brain Does More Than Just Classification

When judging sentiment, we also identify the sentence type (imperative, interrogative, or declarative), focus on every word (subject, verb, object), and consider context. These unconscious processes help form a complete understanding. Our brain is a high-speed processor doing many tasks simultaneously to achieve accurate judgment.

Learning and Prediction

Humans possess learning consciousness and ability. We acquire knowledge through teaching and our own summaries or guesses. In sentiment classification, we don't just memorize words; we infer new ones. For example, if we know "like" and "love" are positive, we can guess "fond of" is also positive. This learning ability is an optimized memory mode that links words rather than just storing them.

Optimization Ideas

Based on the analysis above, we propose the following improvements:

Introduction of Non-linear Features

Real human sentiment classification is heavily non-linear. To improve accuracy, we must introduce non-linearity. This refers to how word combinations form new meanings. Our initial model introduced simple non-linearity by treating adjacent positive and negative words as a negative block. More refined weights can be achieved via a "Lexicon Matrix":

Word	(Empty)	Like	Love	...	Hate	...
(Empty)	0	1	2	...	-1	...
Like	1	2	3	...	-2	...
Love	2	3	4	...	-2	...
⋮	⋮	⋮	⋮	⋮	⋮	...
Hate	-1	-2	-3	...	-2	...
⋮	⋮	⋮	⋮	⋮	⋮	...

Table 1: Conceptual Lexicon Matrix for combination weights

While not all combinations are valid, we can calculate combination weights. However, the number of sentiment words is large, making the matrix size (square of word count) a Big Data problem. Efficient implementation requires optimized construction and indexing schemes.

Automatic Expansion of Sentiment Lexicon

In the internet age, new words emerge constantly. Automatic expansion is necessary for timeliness. We can use unsupervised word frequency statistics on large unlabeled datasets (e.g., from Weibo or communities).

Our goal is to use the existing model for unsupervised learning to expand the lexicon, creating a positive feedback loop. We classify unlabeled data using the current model, then compare word frequencies in positive vs. negative sets. If a word (e.g., "black-hearted") appears frequently in negative reviews but rarely in positive ones, we add it to the negative lexicon with a negative weight.

Example: Suppose "black-hearted" is not in our lexicon, but "detestable" and "dislike" are.

After classification, we find "black-hearted" appears often in negative results. We add it to the negative lexicon and update:

Conclusion

Sentence	Weight
This black-hearted boss is too detestable	-2
I really dislike the practices of this black-hearted enterprise	-2
I hate this black-hearted shop	-2
This shop is really black-hearted!	0

Sentence	Weight
This black-hearted boss is too detestable	-3
I really dislike the practices of this black-hearted enterprise	-3
I hate this black-hearted shop	-3
This shop is really black-hearted!	-2

- Lexicon-based classification is easy to implement; the core is lexicon training.
- Language is complex; linear models have limited performance.
- Introducing non-linear features effectively improves accuracy.
- Unsupervised expansion mechanisms ensure robustness and timeliness.

References

- Deep Learning Notes: `http://blog.csdn.net/zouxy09/article/details/8775360`
- Yoshua Bengio, et al. A Neural Probabilistic Language Model, 2003
- Sentiment Analysis Dataset: `http://www.datatang.com/data/11857`
- Jieba Segmentation: `https://github.com/fxsjy/jieba`
- NLPIR/ICTCLAS: `http://ictclas.nlpir.org/`
- smallseg: `https://code.google.com/p/smallseg/`
- yaha: `https://github.com/jannson/yaha`
- HowNet Sentiment Word List: `http://www.keenage.com/html/c_bulletin_2007.htm`
- NTUSD Lexicon: `http://www.datatang.com/data/11837`
- BosonNLP: `http://bosonnlp.com/product`

Implementation Platform

Tested environment:

- **Windows 8.1** OS.
- **Python 3.4** (Better Unicode/Chinese support than 2.x).
- **Numpy** for numerical computation.
- **Pandas** for data analysis.
- **Jieba** for Chinese word segmentation.

Code List

Resource: Sentiment Lexicon.zip

Preprocessing

```
#-*- coding: utf-8 -*-
import numpy as np
import pandas as pd
import jieba

def yuchuli(s,m):
    wenjian = pd.read_csv(s, delimiter='     xxx     ', encoding='utf-8
        ', header= None, names=['comment'])
    wenjian = wenjian['comment'].str.replace('(<.*?>.*?<.*?>)','').str.
        replace('(<.*?>)','').str.replace('(@.*?[ :])',' ')
    wenjian = pd.DataFrame({'comment':wenjian[wenjian != '' ]})
    wenjian.to_csv('out_'+s, header=False, index=False)
    wenjian['mark'] = m
    return wenjian.reset_index()

neg = yuchuli('data_neg.txt',-1)
pos = yuchuli('data_pos.txt',1)
mydata = pd.concat([neg,pos],ignore_index=True)[['comment','mark']]
```

Loading Lexicon

```
negdict = []
posdict = []
nodict = []
plusdict = []
sl = pd.read_csv('dict/neg.txt', header=None, encoding='utf-8')
for i in range(len(sl[0])): negdict.append(sl[0][i])
sl = pd.read_csv('dict/pos.txt', header=None, encoding='utf-8')
for i in range(len(sl[0])): posdict.append(sl[0][i])
sl = pd.read_csv('dict/no.txt', header=None, encoding='utf-8')
for i in range(len(sl[0])): nodict.append(sl[0][i])
sl = pd.read_csv('dict/plus.txt', header=None, encoding='utf-8')
for i in range(len(sl[0])): plusdict.append(sl[0][i])
```

Prediction Function

```
def predict(s, negdict, posdict, nodict, plusdict):
    p = 0
    sd = list(jieba.cut(s))
    for i in range(len(sd)):
        if sd[i] in negdict:
            if i>0 and sd[i-1] in nodict: p = p + 1
            elif i>0 and sd[i-1] in plusdict: p = p - 2
            else: p = p - 1
        elif sd[i] in posdict:
            if i>0 and sd[i-1] in nodict: p = p - 1
            elif i>0 and sd[i-1] in plusdict: p = p + 2
            elif i>0 and sd[i-1] in negdict: p = p - 1
            elif i<len(sd)-1 and sd[i+1] in negdict: p = p - 1
            else: p = p + 1
        elif sd[i] in nodict:
            p = p - 0.5
    return p
```

Simple Test

```
tol = 0
yes = 0
mydata['result'] = 0
for i in range(len(mydata)):
    tol = tol + 1
    if predict(mydata.loc[i,'comment'], negdict, posdict, nodict,
        plusdict)*mydata.loc[i,'mark'] > 0:
         yes = yes + 1
         mydata.loc[i,'result'] = 1
print(yes/tol)
```