Project report on

Sentiment Analysis on Twitter Data

Pratik Shrestha
Computer Engineering
Roll No 8

February 8, 2016

Contents

1 Introduction

Sentiment analysis is the field of study that analyzes people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes.[1] Twitter is a very popular microblogging site where millions of users post their sentiments and opinions in a daily basis. The purpose of this project is to extract the data people post on twitter and classify it as either positive or negative. To do so, we implement Bayessian Classification to the dataset.

1.1 Bayesian Classification

In this algorithm, we consider each Document d as Bag of words i.e. d= (w1,w2,.....wn) where wn is the nth word in the document and then for classification we calculate the posterior probability of the word of the document being annotated to a particular class.[2] Bayessian classification uses Bayes Theorem, which sayes,

$$p(c|d) = \frac{p(d|c)p(c)}{p(d)} \tag{1}$$

where p(d—c) is the probability of generating instance d given class c, p(c) is the probability of occurrence of class c, and p(d) is the probability of instance d occurring. Of these, p(d) can be ignored since it is the same for all classes. p(c) is simply the fraction of training instances that belong to class c.

2 Methodolody

We take a pretagged dataset of 1600000 tagged tweets. We seperate 10000 from them as test set and use the remaining as training set. These manipulations are done via Python and the data are stored in SQLite3. After training the system, it is tested using the testing set for accuracy. Accuracy is estimated using the following measures.

- **True positive:** A case where the actual result was positive and so was the prediction.
- **False positive:** A case where the prediction was positive but actual result was negative.
- **True negative:** A case where the actual result was negative and so was the prediction.
- **False negative:** A case where the prediction was negative but actual result was positive.

Accuracy The fraction of the time when the classifier gives the correct classification.

$$Accuracy = \frac{true\_positive + true\_negative}{true\_positive + true\_negative + false\_positive + false\_negative} \tag{2}$$

Recall (also known as sensitivity) How many of the actual positive cases are classified as positive.

$$Recall = \frac{true\_positive}{true\_positive + false\_negative} \tag{3}$$

Precision How often the positive prediction is correct.

$$Precision = \frac{true\_positive}{true\_positive + false\_positive} \tag{4}$$

3 Source Code

```
import nltk, sqlite3
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')


class Database:

    def __init__(self, x):

        self.conn = sqlite3.connect(x)
        self.conn.text_factory = str

    def instance(self):
        return self.conn.cursor()

    def commit(self):
        self.conn.commit()

class Bayessian_Classifier:

    def __init__(self, database = 'Data.db'):
        self.database = Database(database)
        self.stopwords = open('stopwords.txt').read().split('\n')

    def filter_words(self, text):

        word_cloud = {}
        words = tokenizer.tokenize(text.lower())
        for word in list(set(words)):

            if len(word) > 1 and not word.isdigit() and word not in self.stopwords:
                word_cloud[word] = words.count(word)

        return word_cloud
```

```
def train(self, text, tag):

    db = self.database.instance()

    word_cloud = self.filter_words(text)
    print 'filtered'
    for word in word_cloud:
        db.execute('select count from BAG_OF_WORDS where word=? and tag=?',
        (word, tag))
        existing_count = db.fetchone()
        if existing_count:
            db.execute('update BAG_OF_WORDS set count=? where word=? and tag=?',
             (existing_count[0] + word_cloud[word], word, tag))
        else:
            db.execute('insert into BAG_OF_WORDS (word, tag, count) values
            (?,?,?)', (word, tag, 1))

    self.database.commit()

def classify(self, text):

    db = self.database.instance()
    positive_count = 0.0
    negative_count = 0.0

    for word in self.filter_words(text):
        db.execute('select count from BAG_OF_WORDS where word=? and tag=?',
         (word,'P'))
        existing_positive = db.fetchone()
        if existing_positive:
            positive_count += existing_positive[0]

        db.execute('select count from BAG_OF_WORDS where word=? and tag=?',
         (word,'N'))
        existing_negative = db.fetchone()
        if existing_negative:
            negative_count += existing_negative[0]


    if positive_count + negative_count < 3:
        return 0.5
    elif positive_count == 0:
        return 0.05
    elif negative_count == 0:
```

```
            return 0.95
        else:
            return float(positive_count/(positive_count+negative_count))


if __name__ == '__main__':

    print 'Sentiment Analysis Accuracy Calculation\n'

    bayes = Bayessian_Classifier()

    #bayes.train(open('positive_training_set.py'), 'P')
    #bayes.train(open('negative_training_set.py'), 'N')

    positive_testing_set = open('positive_testing_set.txt').readlines()
    negative_testing_set = open('negative_testing_set.txt').readlines()

    true_positive = 0.00
    true_negative = 0.00
    false_positive = 0.00
    false_negative = 0.00

    for counter, positive_sentence in enumerate(positive_testing_set):
        if bayes.classify(positive_sentence) > 0.5:
            true_positive += 1
        else:
            false_negative += 1

    for i, negative_sentence in enumerate(negative_testing_set):
        if bayes.classify(negative_sentence) < 0.5:
            true_negative += 1
        else:
            false_positive += 1


    print 'True positive', true_positive
    print 'True negative', true_negative
    print 'False positive', false_positive
    print 'False negative', false_negative

    print 'Accuracy: ', (true_positive + true_negative) /
    (true_positive + true_negative + false_positive + false_negative)
    print 'Recall: ', true_positive / (true_positive + false_negative)
    print 'Precision: ', true_positive / (true_positive + false_positive)
```

4 Result

The followong results were obtained.

	True	**False**
Positive	3066	1077
Negative	3923	1934

Accuracy = 0.7003006012024048
Recall = 0.6156626506024097
Precision = 0.7436332767402377

5 Conclusion

Hence, we applied naive bayes appication and implemented sentiment analysis on twitter data. The accuracy was found to be 70.03%.

6 Reference

1. Bing Liu, Sentiment Analysis and Opinion Mining, Morgan & Claypool Publishers, May 2012.

2. Chandan Prasad Gupta, Detecting Sentiment in Nepali Texts, March 2015