CIS520 Report: ML Crackers

Francine Leech, Ziyin Qu, Chen Xiang

December 12, 2016

1 Introduction

The rise in web and mobile based social networking has opened a stream of continuous text data. These data reflect the sentiment of individuals and masses of people. Understanding the sentiment of users is relevant on the most basic level, understanding how people are responding to a stimulus, and extending to human machine interaction systems. The ambiguity of language and emotional expression creates an interesting machine learning problem. We try to tackle one aspect of this problem by classifying tweets into two categories, joy and sadness.

2 Preliminary Methods

When starting the project, we thought that we should first experiment with different classification methods on the image data (train_color, train_img_prob) because they were smaller datasets to observe how well they predicted our emotions of interest. Using our intuition, we hypothesized that the image data contained basic information about people's sentiments. For example, lighter and bright colors represent joy, darker colors represent sadness. We tried to reduce the image datasets by preforming Principal Component Analysis (PCA) with the idea that with the main principal components our model would fit the transformed data well and have a high testing accuracy. The PCA-ed data was an input into a Gaussian Mixture Model(GMM) with two clusters specified, where one cluster was joy and the other sadness. Similarly, we used the same apporach on the word_train data. 10-fold cross validation is time intensive with PCA, so we used a 'holdout' of 20% of the train data to use as testing because cross validation with PCA is time intensive. Our KMeans method used the word_train data, with a Spearman correlation and 499 neighbors.

The results were not great (Figure 1). The image data, train_color, train_img_prob had high error between 0.40 and 0.50. The PCA and GMM model on train_words had comparable error to the error of the same model trained on the image data. KMeans had the lowest error out of all our preliminary models. We tried training the models by changing the number of clusters in the GMM and PCs in the PCA. The results were similar to our initial results.

The PCA and GMM models did not predict well because the image data did not contain enough information for this method to get a predictive classifier, unlike other classical clustering problems like human face recognition or male and female recognition. Our simple hypothesis was not true, suggesting that dark colors were found in tweets labeled as "joy" and bright colors were found in tweets in labeled as "sadness." The KMeans method also did predict well and was time consuming. The error was still high due to the sparsity of data and the redundancy of attributes, specifically the count of different words. Overall, the accuracy of preliminary models were not high enough to beat the Baseline 1, so we tried different methods.

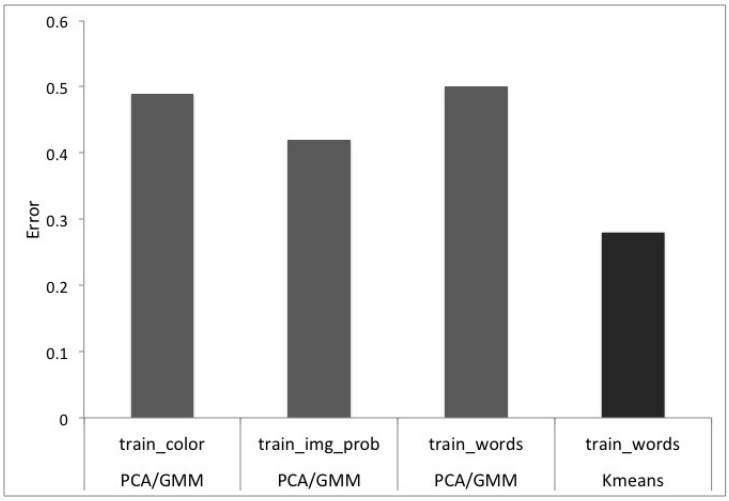

Figure 1: Error of our Preliminary Methods

3 Main Methods

Here we describe the combination of models we used in our final model. We used our Naive Bayes model to beat Baseline 1, 78%, and our final to beat Baseline 2 and the final submission.

3.1 Naive Bayes

For supervised learning, Naive Bayes classification is an effective generative method for classifying texts, such as identifying spam in email. The words_train data set uses bag of words model where counts of words matters and position of words does not matter. There are lots of advantages for Naive Bayes model, it can be a dependable baseline for text classification, it trains fast. Although the assumption of Naive Bayes which is the Conditional Independence Assumption may not be true, it may still work well on text classification problem.

For supervised learning, Naive Bayes method is a very good generative method on classifying texts, like spam classification problem. Actually the words_train data set uses bag of words model where counts of words matters and position of words does not matter. There are lots of advantages for Naive Bayes model, it can be a dependable baseline for text classification, it trains fast. Although the assumption of Naive Bayes which is the Conditional Independence Assumption may not be true, it may still work well on text classification problem.

To use the Naive Bayes model on words_train dataset, we use the built-in matlab function fitNaiveBayes. For training data, we sue 9-fold cross validation error to estimate the test error for Naive Bayes model, which means we randomly choose 4000 data to train and 500 data to test. For the built-in function fitNaiveBayes, there are some parameters for us to choose, for the distribution parameter, because we are using the bag-of-words model, we use the multinomial distribution in the fitNaiveBayes function.

The Naive Bayes model actually works really well. We got around 0.8 cross validation accuracy on training data. And we got 0.7962 accuracy for the test data. We successfully beat the Baseline 1 using a simple Naive Bayes model with multinomial distribution.

The Naive Bayes model actually works really well. We got around 0.8 cross validation accuracy on training data. And we got 0.7962 accuracy for the test data. We successfully beat the baseline1 using a simple Naive Bayes model with multinomial distribution.

¿¿¿¿¿¿¿ origin/master

But there are still problems with the simple Naive Bayes model. For example, the dataset matrix for words is very sparse, and for each observation there are many words did not show up. Naive Bayes model for text assumes that there is no information in words that are not observed and this may cause over-fitting. We can solve this by smoothing the Naive Bayes model.

3.2 GentleBoost

The second main method we utilized was an ensemble method. We used GentleBoost, a weak learning that was built by MATLAB under the fitensemble function. The method combines many weak learners into one high quality ensemble predictor. We chose this ensemble methods over the others offered by MATLAB, because it is preforms well with binary classification trees with many predictors (ensemble citation).

The input of the model was the word_train data. We used a 10-fold cross validation method to observed how the model preformed, specified the use 300 learners, and the type of learner as 'tree'. The average cross validation error was 0.21. The algorithm classified joy and sadness well.

The method could have improved if we increased the number of learners, however it would have taken a very long time to train because the data is large. Initially we tried the method with the default number of learners, 100 trees, and found that the cross validation accuracy only improved slightly. This slight improvement with triple number of learners reveals that the data has some intricacies or patterns that the ensemble method cannot learn.

3.3 Support Vector Machine

Support vector machines (SVMs) proved to the most promising method to classify the data. We used the MATLAB function, fitcsvm, to train an SVM model for binary classification on the on the word_train data.

We tried a simple SVM by specifying a linear kernel, and had a cross validation error was 0.2180. With a Gaussian or RBF kernel we had an error of 0.4373.

After experimenting with a variety of kernels, we found that the linear preformed the best. fitsvm allows you to make an assumption about the fraction of outliers in the data. While we could have gone through the raw tweets and looked through the data, we decided to experiment with 10%, 20%, and 30% and observed cross validation errors 0.2121, 0.2282, and 0.2131 respectively. Specifying the outliers percentage did not have an effect on our cross validation error, so we decided not to specify in our SVM final model.

Lastly we optimized our SVM by using MATLABs built in method to optimize a cross-validated SVM using Bayes Optimization (citation). The method originates from The Elements of Statistical Learning, Hastie, Tibshirani, and Friedman (2009). Paraphrasing from the MATLAB documentation, "the model begins with generating 10 base points for a "green" class, distributed as 2D independent normals with mean (1,0) and unite variance. It then generates 10 base points for a "class" that is also distributed as 2-D independent normals with mean (0,1) and unit variance. For each of the classes, it generate 100 random points by choosing a base point, b, of the respective color uniformly at random. It then generates an independent random point with 2-D normal distribution with mean b and variance I/5, where I is the 2-by-2 identity matrix. After 100 points for each of the colors has been generated, the point are classified using fitcsvm. The function bayesopt is used to optimize the parameters of the final SVM model with respect to cross validation." We submitted the method to the autograder, and it had an accuracy of 0.7991. The method was accurate enough to beat Baseline 1, but not Baseline 2. Similar to the other methods above, the

optimized SVM may not preform well because the data was sparse and high dimensional, so the hyperplane could not separate data well.

4 Final Method

Our final method utilizes sentiment analysis, the classification of text into categories of emotions. We used the vaderSentiment 2.4.1 package in Python. VADER is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. The advantage of this analysis is that we can use this preliminary method to discern extremely positive and extremely negative tweets by assuming that there are no extremely positive words shown in negative sentences and vise versa. We will describe the process of experimenting with two analyzes before selecting one to include in our final model.

4.1 Sentiment Analysis 1

In Python, we ran a sentimental analysis on each word present in the topwordsċsv. The input was each individual word in the list, and the output was the probability the word expresses a negative, positive, and neutral emotion. The output often looks like this, when type "funny" we can see {'compound': 0.4404, 'neg': 0.0, 'neu': 0.0, 'pos': 1.0}, which means it is an extremely positive word.

4.2 Sentiment Analysis 2

An issue we came across is from our first sentiment analysis is that we did not consider the raw tweets containing words that began with #. These hash tags may represent the topic this sentence belongs to, some specific topics always express similar emotions, like #family usually expresses a positive emotion. So instead of analyzing the sentiment of each word, we used sentence as the input and received the average emotion scores of each word.

We ran the sentimental analysis on each raw tweet using VADER package. For all the words that appeared in this sentence, we attached the resulting score to those words. For every word, an average emotion score is then based on all the raw tweets.

4.3 Final Method

Our final method beat Baseline 2 and received an accuracy of 81.82%.

From the testing on training dataset, we found that Sentiment Analysis had a high accuracy predicting "sadness". So the intuition was really straight forward, if the test data calculated by Sentiment Analysis 1 has negative words and don't have positive words, we will label it as negative. Then how we deal with the tweets the Sentiment Analysis cannot predict? W Naive Bayes and SVM and GentleBoost all have a quite good accuracy on test data, so we use the combine of the three methods through majority vote to predict them.

Figure 1 shows the structure of our final method. The majority vote goes in this way. We have three models predicting the test labels, if three of them all predict a tweet as sadness, we label this tweet as sad. If three of them all predict a tweet as joy, we will label them as joy. If any two of them predict a tweet as sadness, we label it as sadness. Finally if two of them predict a tweet as joy, we just use Naive Bayes results. Why in this situation we just use Naive Bayes instead of majority vote? Because the trainning accuracy tends to be higher for just use Naive Bayes in this situation than use majority vote.

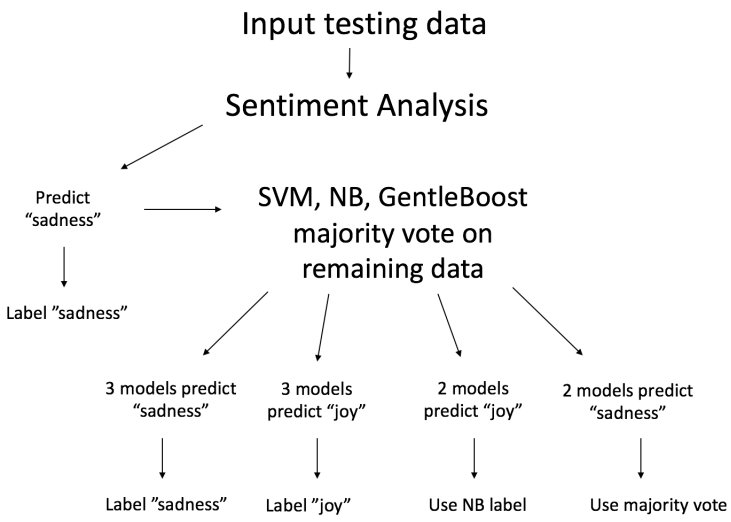

Figure 2: System Classification

Actually we also tried using Sentimental Analysis 2 with the combination of the three methods. But the accuracy did not improve. It may because the new Sentiment Analysis did not have that strong predictive words like Sentiment Analysis 1 has. Or maybe the result of Sentiment Analysis 1 is more suitable to the testdata.

5 Discussion

Sentiment analysis is a difficult problem because human emotion is multifaceted and varies in intensity. In person, understanding the emotion a person is based on several parameters, the context of what they're saying, their words, tone, body language, and facial expression. Our classification problem is more difficult because we are given short tweets with a corresponding image.

One issue Issues with Lexicon Developing the content of the lexicon is subjective. The use of language changes - how old is the lexicon being used? We don't know if the lexicon is comprehensive enough. Lexicon with emojis.

What would our future method look like? - Train classifier that train on emojis and hashtags since they are usually represent the topic of the tweet

Neutral tweets

- Could have used deep learning too long, overfit, dataset small ¡¡¡¡¡¡¡ HEAD - articles ======= - articles

Other methods we tried on words includes TF-IDF. TF-IDF is short for term frequencyinverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. We use this method to transform the train data and then use SVM or logistic regression methods to predict. However, the results are not satisfying. It may because the training data are too sparse and many of them didn't show up, so the result of TF-IDF can not give us more information about the train set.

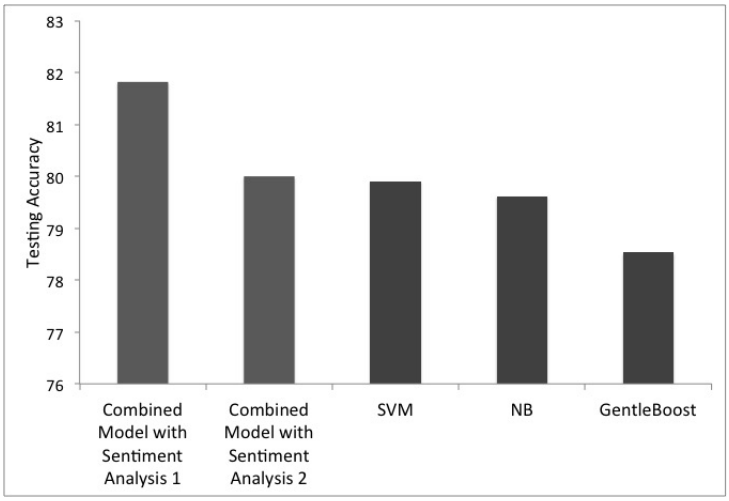

Figure 3: Test Error Final Models. The separate final models are in dark grey, and combined models with sentiment analysis are in light grey. The combined models have higher testing accuracy that the separate models. The combined model with sentiment analysis 2 preforms just as well as SVM. The model with Sentiment Analysis 1 has the highest testing accuracy.

6 Works Cited

References

7 Appendix

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  5 15:05:19 2016

@author: chen
"""
# sentimental anaylsis use vader
import pandas as pd
import re
import numpy as np
from nltk.sentiment import SentimentIntensityAnalyzer
vader_analyzer = SentimentIntensityAnalyzer()
sentimentalWords = pd.read_csv("topwords.csv", header=None)
sentimentalSentences = pd.read_csv("raw_tweets.csv", header=None)
sentimentalResult = pd.read_csv("sentimentalResult.csv", header=0)

# Ensemble 1 method
# To get sentimenral result for each word in top words list


def dealSentences(df):
    n = len(df)
    df.ix[:, 'neg'] = 0
    df.ix[:, 'neu'] = 0
    df.ix[:, 'pos'] = 0
#    res = {}.fromkeys(['neg', 'neu', 'pos'], [])
```

```
    for i in range(n):
        sentence = str(df.iloc[i].values[0])
        sentence = sentence.translate(None, '#')
        re.sub('@\w+', '', sentence)
        sentiment = vader_analyzer.polarity_scores(sentence)
        df.ix[i, 'pos'] = sentiment['pos']
        df.ix[i, 'neg'] = sentiment['neg']
        df.ix[i, 'neu'] = sentiment['neu']
    return df
# To get postitive, neutral, negtive emotions of each word
sentimentalWords = dealSentences(sentimentalData)
# To get postitive, neutral, negtive emotions of each sentence
sentimentalSentences = dealSentences(sentimentalSentences)


# Ensemble 2 method
# Make some changes to the first sentimentalData, focus on the word with #


def newSentimentalResult(df):
    n = len(df)
    sentimentalResultFinal = df
    for i in range(n):
        name = df.ix[i, '0']  # name
        name = str(name)
        if "#" in name:
            pos = df.ix[i, 'pos']
            neg = df.ix[i, 'neg']
            if pos > neg:
                sentimentalResultFinal.ix[i, 'pos'] = 1
            elif neg > pos:
                sentimentalResultFinal.ix[i, 'neg'] = 1
    return sentimentalResultFinal
# To get new postitive, neutral, negtive emotions of each word based on
# former sentimental data
sentimentalResultNew = newSentimentalResult(sentimentalResult)
```

"./sentimental analysis with python.py"