ERG3020 Course Project - Twitter
CUHK-SZ, Apr 22, 2018

Course Project

Student Name: Piao Chuxin, Ye Xiaoxing
Student ID: 115010058, 115010270

Contents

1 Abstract

Twitter is a great firehouse of real-time information. Using the data from twitter, people can gain information such as political leaning and other tendencies. Through the classification process of the data before the election day, the more competitive candidate with a higher approval rating can be obtained.

2 Introduction

American President Election is one of the catchiest affairs all over the world. To predict the candidate with higher approval rate beforehand, social networks can be effective in obtaining the political leaning of citizens in the United States.

Twitter is one of the influential social networks that can be used in predicting the political leaning of people who use twitter. Having downloaded the data of one day from the twitter website, the emotion scores have been computed, represented by a number from -1 to 1, in which 1 represents the positive emotion while -1 represents the negative emotions. After obtaining the score of emotion, the keywords which could represent different candidate have been searched. For example, Trump or Donald could represent Donald Trump, and Hillary or Clinton could represent Hillary Clinton. Compared the twitter score which contains different key words to the average emotional score of all the twitter score, the rough political leaning of different candidates can be obtained.

To tell the political leaning of people, we use sentiment analysis. Sentiment Analysis, sometimes known as opinion mining, is a field of study combining the use of natural language processing, text analysis, and linguistics, to identify and study affective and subjective information. Tipically, it analyses peoples opinions towards entities, such as products, companies, and parties. Thanks to the rapid-developing social media, this area has now been a hot topic with a constant source of textual data.

3 Model

3.1 Sentiment Analysis

To evaluate the tweets' sentiment, we first trained a series of models. They are...

1. Naive Bayes
   (a) Simple Naive Bayes
   (b) Multinomial Naive Bayes
   (c) Bernoulli Naive Bayes
2. Linear Model
   (a) Logistic Regression
   (b) Stochastic Gradient Descent
3. Support Vector Machine
   (a) Linear Support Vector Classification

In this part, we would like to introduce the models one by one, and then our implementation.

3.1.1 Naive Bayes

Naive Bayes classifier is a simple probabilistic classifier based on Bayes' Theorem, while assuming the independence between features. This technique has been studied for over 60 years, and it is now hot since it is simple but effective. It is still a popular and the baseline method for text classification. With proper pre-processing, ti is evn competitive with advanced methods like SVM.

Naive Bayes is a simple technique for constructing classifiers. Class labels are assigned to instances, where the labels is a finite set. It is a family of algorithems based the principle that all assume the independence of feature, given the class variable. Despite the oversimplifiatied assumptions, the Navie Bayes classifiers works well in many real-world cases. And, one of the advantage is that, only a small training set is required, to guess the essential parameters. [Wik18]

Naive Bayes is a conditional probability model. Given an instance to be classified, its features are represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$, and the classification is assigned to the probabilities, $p(C_k \mid x_1, \ldots, x_n)$, for each of K possible classes C_k. Using Bayes' Theorem, the conditional probability can be transformed to

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

which, in plain English, can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

If we break down the equation [Let15],

- $p(C_k)$ (Prior) = frequency of class C_k / total number of samples
- $p(\mathbf{x}|C_k)$ (Likelihood) = (frequency of $\mathbf{x}$ / number of samples) where class = C_k
- $p(\mathbf{x})$ (Evidence) = frequency of $\mathbf{x}$ / total number of samples

We only care about the numerator of that fraction, since the denominator does not depend on C, and it is effectively constant since the values x_i are already given. By the joint probability model, the numerator is equivalent to $p(C_k, x_1, \ldots, x_n)$, and by the chain rule for repeated applications,

$$\begin{aligned}
p(C_k, x_1, \ldots, x_n) &= p(x_1, \ldots, x_n, C_k) \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2, \ldots, x_n, C_k) \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) p(x_3, \ldots, x_n, C_k) \\
&= \ldots \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) \ldots p(x_{n-1} \mid x_n, C_k) p(x_n \mid C_k) p(C_k)
\end{aligned}$$

THen the naive conditional independence assumptions (assuming that each feature is conditionally independent with each other, given the category C_k) come, $p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$. Combine all equations,

$$\begin{aligned}
p(C_k \mid x_1, \ldots, x_n) \propto p(C_k, x_1, \ldots, x_n) &= \\
&= p(C_k)\ p(x_1 \mid C_k)\ p(x_2 \mid C_k)\ p(x_3 \mid C_k)\ \cdots \\
&= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,.
\end{aligned}$$

Then, the conditional distribution over the C becomes,

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

where the evidence $Z = p(\mathbf{x}) = \sum_k p(C_k)\ p(\mathbf{x} \mid C_k)$ is a scaling factor only depending on $x_1, \ldots, x_n$, and now we have the naive Bayes probability model.

The naive Bayes classifier combines the naive Bayes probability model with a decision rule. One common rule is MAP decision rule, which is the equation of

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{argmax}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

Different models of Naive Bayes classifier The prior of class shall be calculated by assuming all classes have equal probability, or by calculating an estimate from the training set. Before we can estimate the parameters, we shall assume a distribution from the training set. This assumption is often called the event model of the Naive Bayes classifier. Since we are classifying document data, which have discrete features, we use multinomial and Bernoulli distributions.

Multinomial distribution This multinomial navie bayes models the word counts With such a multinomial event model, the feature vectors represent the frequencies, where certain events have been generated by a multinomial distribution $(p_1, \ldots, p_n)$ where p_i is the probability that event **i** occurs. If we take x_i as the numvber of times event **i** in a particular instance, then the feature vector $\mathbf{x} = (x_1, \ldots, x_n)$ is now a histogram. Then, the likelihood function is given by

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

This model is suitable for text classification. The features are the words, and the values is the frequencies. If we break the previous equation, it becomes

$$p(x_i \mid C_k) = \frac{N_{C_k x_i} + \alpha}{N_{C_k} + \alpha n}$$

,

where $N_{C_k x_i}$ is the occurances of x_i in Class C_k; N_{C_k} is the total occurances of features in Class C_K; n is the number of features and α is the smoothing parameters ranged in $[0, 1]$, and in most cases it is 1. The reason why we need a smoothing parameter is that if a given feature does not exist in the training data, the estimate will become zero, and then all probabilities under the Naive Bayes classifier will be zero. Adding the smoothing parameters is a way to regularizing naive Bayes, which is called Lidstone smoothing or Laplace smoothing (when $\alpha = 1$).

Bernoulli distribution In the Bernoulli event model, we use binary values for features. In document classification, the binary value determines the existence of features, which is, different from the multinomial model which uses the frequencies. The likelihood can be given by

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} p_{ki}^{x_i} (1 - p_{ki})^{(1-x_i)}$$

It also explicitly models the absent terms, and the absence is now also a feature.

3.1.2 Linear Model

Linear model, or linear regression, is a linear modeling approach to find the relationship between a scalar dependent variable y and explanatory variables X. It is a technique for predicting a real value using a straight line in 2-D model or plane or hyperplane in higher dimensions.

Each input variable $\mathbf{x}$ is weighted using a coefficient ($\mathbf{b}$). The goal of the algorithems is to discover the best-fit set of coefficients.

$$y = b_0 + b_1 \times x_1 + ...$$

Logistic Regression Logistic regression is a regression model where the dependent variable is categorical. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function [lr-].

The cost function for the L2 logistic regression is:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Stochastic Gradient Descent Stochastic gradient descent is a simple but efficient approach to fit linear models. coefficients are evaluated and updated every iteration, to minimize the error of the model. In this way, the model makes a prediction for an instance, and updated for the next prediction in order to reduce the error. This requires two parameters,

- Learning Rate: Limit the amount each update;
- Epochs: Iterations to run through the training data.

Then, the coefficients ($\mathbf{b}$) are updated using the equation:

$$\mathbf{b} = \mathbf{b} - \text{learning\_rate} \times \text{error} \times x$$

where error is the prediction error attributed to the weight.

3.1.3 Support Vector Machine

Support Vector Machine is a set of supervised machine larning algorithems, which can be used for classification and regression. In this algorithem, we first plot each data as a point in space, with the value being the value of particular coordinate. Classification is then performed by finding the hyperplane which differentiate the two classes well.

Support Vector Classification is one application of the SVM.

Given training vectors $x_i \in \mathbb{R}^p, i = 1, \ldots, n$, in two classes, and a vector $y \in \{1, -1\}^n$, SVC solves the following primal problem:

$$\min_{w,b,\zeta} \left\{ \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \right\}$$
$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, ..., n$$

Its dual is

$$\min_{\alpha} \left\{ \frac{1}{2} \alpha^T Q \alpha - \mathbf{1}^T \alpha \right\}$$
$$\text{subject to } y^T \alpha = 0$$
$$0 \leq \alpha_i \leq C, i = 1, ..., n$$

where $\mathbf{1}$ is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is $\text{sgn}(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho)$ [PVG+11].

3.2 Implementation

All fils of the code we mention here are available online: https://github.com/Yexiaoxing/tweet-sentiment-analysis . To implement the project, we first need tweets data. The tweets ID data from GitHub Repo (chrisalbon/election˙day˙2016˙twitter) was then borrowed. The original dataset contains 6,546,824 tweets posted on election day, which includes one of the following keywords,

1. hillary
2. hillary
3. trump
4. #yourefired
5. election
6. #election2016
7. #electionday
8. #uselections2016
9. gop
10. democrat
11. #ivoted vote voted
12. #senate
13. #uselection
14. #house
15. congress
16. #madampresident

Due to the Twitter's Term of Service, the dataset only contains the IDs [twi18]. So, we need to hydrate them. Hydrate means to get the details from a collection of Tweet IDs. The tweets˙fetch.py file is designed to do so. It accepts various parameters, and two of them are input of a csv files containing IDs, and output of the details.

```python
"""This module fetches full tweets (also called hydrate tweets) from given csv file."""
import pandas as pd
import tweepy
import csv
from tqdm import trange
from optparse import OptionParser
from typing import List

# Insert your Twitter API key here
consumer_key = ''
consumer_secret = ''

access_token = ''
access_secret = ''


def retrieve_tweets(tweet_ids: List[str],
                    output_file_name: str,
                    proxy: str = ""):
    """
    Retrieve tweets from list of tweet ids.

    @param tweet_ids: List of tweet id.
    @param output_file_name: The file to write.
    """
    print("Output:", output_file_name)

    # Authorization with Twitter
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_secret)
    api = tweepy.API(
        auth,
        wait_on_rate_limit=True,
        wait_on_rate_limit_notify=True,
        retry_count=3,
        retry_delay=5,
        retry_errors=set([401, 404, 500, 503]),
        proxy=proxy)

    # Create output file
    csvFile = open(output_file_name, 'w', encoding='utf-8')
    csvWriter = csv.writer(csvFile)
    csvWriter.writerow([
        "text", "created_at", "geo", "lang", "place", "coordinates",
        "user.favourites_count", "user.statuses_count", "user.description",
        "user.location", "user.id", "user.created_at", "user.verified",
        "user.following", "user.url", "user.listed_count",
        "user.followers_count", "user.default_profile_image",
```

```
        "user.utc_offset", "user.friends_count", "user.default_profile",
        "user.name", "user.lang", "user.screen_name", "user.geo_enabled",
        "user.profile_background_color", "user.profile_image_url",
        "user.time_zone", "id", "favorite_count", "retweeted", "source",
        "favorited", "retweet_count"
    ])

    print("Total IDs", len(tweet_ids))
    # Append tweets to output file

    # Twitter allows a batch of 100 tweets
    for tweetid_batch in trange(len(tweet_ids) // 100):
        try:
            status_es = api.statuses_lookup(
                tweet_ids[tweetid_batch * 100:tweetid_batch * 100 + 100])
            for status in status_es:
                csvWriter.writerow([
                    status.text, status.created_at, status.geo, status.lang,
                    status.place, status.coordinates,
                    status.user.favourites_count, status.user.statuses_count,
                    status.user.description, status.user.location,
                    status.user.id, status.user.created_at,
                    status.user.verified, status.user.following,
                    status.user.url, status.user.listed_count,
                    status.user.followers_count,
                    status.user.default_profile_image, status.user.utc_offset,
                    status.user.friends_count, status.user.default_profile,
                    status.user.name, status.user.lang,
                    status.user.screen_name, status.user.geo_enabled,
                    status.user.profile_background_color,
                    status.user.profile_image_url, status.user.time_zone,
                    status.id, status.favorite_count, status.retweeted,
                    status.source, status.favorited, status.retweet_count
                ])
        except Exception as e:
            print(str(e))


def main(options, args):
    """Initialize the board, solver object and call the solve() function."""
    df = pd.read_csv(options.infile)
    retrieve_tweets(df.iloc[:, 0], options.out, proxy=options.proxy)


if __name__ == '__main__':
    parser = OptionParser(usage="Usage: %prog -i input_file" +
                          " -o output_file -p proxy_address")
    parser.add_option("-p", "--proxy", dest="proxy",
                      metavar="str", help="Proxy address", default="")
    parser.add_option("-i", "--in", dest="infile",
```

```
                      help="Input CSV file", metavar="FILE")
    parser.add_option("-o", "--out", dest="out",
                      help="Output CSV file", metavar="FILE")
    (options, args) = parser.parse_args()
    if not options.infile:
        parser.error('Input CSV filename not given')

    if not options.outfile:
        parser.error('Output CSV filename not given')
    main(*parser.parse_args())
```

It is required to obtain twitter API key from https://apps.twitter.com/. Since there is a rate limit for the API endpoints, the whole hydrating will take days if multi-process is used. The fetched details include

> "text", "created_at", "geo", "lang", "place", "coordinates", "user.favourites_count", "user.statuses_count", "user.description", "user.location", "user.id", "user.created_at", "user.verified", "user.following", "user.url", "user.listed_count", "user.followers_count", "user.defaul˙profile_image", "user.utc_offset", "user.friends_count", "user.default_profile", "user.name", "user.lang", "user.screen_name", "user.geo_enabled", "user.profile˙background˙color", "user.profile_image_url", "user.time_zone", "id", "favorite_count", "retweeted", "source", "favorited", "retweet_count"

almost all information of a tweet and a user.

Before we can analyze the sentiment, we need a model. Here we implement a voting algorithem, which queries from all models we have mentioned before, and gives the confidence (score) of the sentiment. The voting algorithem is implemented in vote˙classifier.py file.

```
from nltk.classify import ClassifierI
from statistics import mode


class VoteClassifier(ClassifierI):
    def __init__(self, *classifiers):
        """Voting Classifier

        Arguments:
            classifiers {ClassifierI} -- List of classifiers
        """

        self._classifiers = classifiers

    def sentiment(self, features: list):
        """Classify the sentiment and get the confidence.

        Arguments:
            features {list} -- List of features

        Returns:
```

```
            str -- positive or negative
            int -- confidence
        """

        votes = [c.classify(features) for c in self._classifiers]
        choice_votes = votes.count(mode(votes))
        conf = choice_votes / len(votes)
        return mode(votes), conf

    def classify(self, features: list):
        """Classify the sentiment

        Arguments:
            features {list} -- List of features

        Returns:
            str -- positive or negative
        """

        votes = [c.classify(features) for c in self._classifiers]
        try:
            return mode(votes)
        except:
            return "pos"

    def confidence(self, features: list):
        """Get the confidence by talling the votes for and against the
        ↪  winning vote

        Arguments:
            features {list} -- List of features

        Returns:
            int -- confidence
        """
        votes = [c.classify(features) for c in self._classifiers]
        choice_votes = votes.count(mode(votes))
        conf = choice_votes / len(votes)
        return conf
```

Then, we train all the models we mentioned before. The positive and negative training sets are from GitHub Repo (aalind0/NLP-Sentiment-Analysis-Twitter). In the naive Bayes part, we utilized both NLTK and sklearn's classifiers. When training, the dataset is separated into two parts, the first 10,000 features as training set and the remaining as the testing set.

```
#!/usr/bin/env python3

# Training the classifiers and then pickling.
# Executing it sucks time. :P
```

```
5
6  import pickle
7  import random
8  from datetime import datetime
9
10 import nltk
11 from nltk.classify.scikitlearn import SklearnClassifier
12 from nltk.tokenize import word_tokenize
13 from sklearn.linear_model import LogisticRegression, SGDClassifier
14 from sklearn.naive_bayes import BernoulliNB, MultinomialNB
15 from sklearn.svm import SVC, LinearSVC
16
17 from utils import info, pickling
18 from vote_classifier import VoteClassifier
19
20 allowed_word_types = ["J", "R", "V"]
21
22
23 def read_corporas(positive: str="data/corporas/positive.txt", negative:
   ↪  str="data/corporas/negative.txt"):
24     """Read corporas
25
26     Keyword Arguments:
27         positive {str} -- Path to positive text (default:
   ↪  {"data/corporas/positive.txt"})
28         negative {str} -- Path to negative text (default:
   ↪  {"data/corporas/negative.txt"})
29
30     Returns:
31         list -- documents
32         list -- all words
33     """
34
35     # Defining and Accessing the corporas.
36     # In total, approx 10,000 feeds to be trained and tested on.
37     all_words: list = []
38     documents: list = []
39
40     info("Accessing the corporas...")
41
42     for p in open(positive, "r"):
43         p = p.strip()
44         documents.append((p, "pos"))
45         words = word_tokenize(p)
46         pos = nltk.pos_tag(words)
47         for w in pos:
48             if w[1][0] in allowed_word_types:
49                 all_words.append(w[0].lower())
50
51     for p in open(negative, "r"):
```

```
52          documents.append((p, "neg"))
53          words = word_tokenize(p)
54          pos = nltk.pos_tag(words)
55          for w in pos:
56              if w[1][0] in allowed_word_types:
57                  all_words.append(w[0].lower())
58
59      return documents, all_words
60
61
62  def get_features(all_words: list, length: int=5000):
63      """Calculate the most frequent words as features
64
65      Arguments:
66          all_words {list} -- All words of the string.
67
68      Keyword Arguments:
69          length {int} -- Length of features (default: {5000})
70
71      Returns:
72          list -- features
73      """
74
75      return list(nltk.FreqDist(all_words).keys())[:length]
76
77
78  def find_features(document: str, features: list):
79      """The feature finding function, using tokenizing by word in the
        ↪ document.
80
81      Arguments:
82          document {str} -- Document
83          features {list} -- List of features
84
85      Returns:
86          [type] -- [description]
87      """
88
89      words = word_tokenize(document)
90      _features = {w: (w in words) for w in features}
91      return _features
92
93
94  if __name__ == '__main__':
95      info("Training classifiers. This may take few minutes to finish.")
96      documents, all_words = read_corporas()
97
98      info("Getting top 5000 words as features...")
99      word_features = get_features(all_words)
100     pickling("data/pickles/word_features5k.pickle", word_features)
```

```
101
102     info("Tokenizing and finding features for training...")
103     featuresets = [(find_features(rev, word_features), category)
104                    for (rev, category) in documents]
105
106     # Shuffling
107     random.shuffle(featuresets)
108     info("Length of the feature sets: " + str(len(featuresets)))
109
110     # Partitioning the training and the testing sets.
111     testing_set = featuresets[10000:]
112     training_set = featuresets[:10000]
113
114     print()
115     info("Training and successive pickling of the classifiers...")
116     info("This will take much time. Be patient.")
117
118     print()
119     info("Current Algorithm: " + "NLTK Original Naive Bayes")
120     nb_classifier = nltk.NaiveBayesClassifier.train(training_set)
121     info("Accuracy Percent:", str((nltk.classify.accuracy(
122         nb_classifier, testing_set)) * 100))
123     pickling("data/pickles/original_naive_bayes.pickle", nb_classifier)
124
125     print()
126     info("Current Algorithm: " + "Sklearn Multinomial Naive Bayes")
127     mnb_classifier = SklearnClassifier(MultinomialNB())
128     mnb_classifier.train(training_set)
129     info("Accuracy Percent:", str(
130         (nltk.classify.accuracy(mnb_classifier, testing_set)) * 100))
131     pickling("data/pickles/multinomial_naive_bayes.pickle", mnb_classifier)
132
133     print()
134     info("Current Algorithm: " + "Sklearn Bernoulli Naive Bayes")
135     bnb_classifier = SklearnClassifier(BernoulliNB())
136     bnb_classifier.train(training_set)
137     info("Accuracy Percent:", str(
138         (nltk.classify.accuracy(bnb_classifier, testing_set)) * 100))
139     pickling("data/pickles/bernoulli_naive_bayes.pickle", bnb_classifier)
140
141     print()
142     info("Current Algorithm: " + "Sklearn Logistic Regression")
143     lr_classifier = SklearnClassifier(LogisticRegression())
144     lr_classifier.train(training_set)
145     info("Accuracy Percent:", str(
146         (nltk.classify.accuracy(lr_classifier, testing_set)) * 100))
147     pickling("data/pickles/logistic_regression.pickle", lr_classifier)
148
149     print()
150     info("Current Algorithm: " + "Sklearn SGD classifier")
```

```
151     SGD_classifier = SklearnClassifier(SGDClassifier())
152     SGD_classifier.train(training_set)
153     info("Accuracy Percent:", str(
154         (nltk.classify.accuracy(SGD_classifier, testing_set)) * 100))
155     pickling("data/pickles/sgd.pickle", SGD_classifier)
156 
157     print()
158     info("Current Algorithm: " + "Sklearn Linear SVC")
159     linearSVC_classifier = SklearnClassifier(LinearSVC())
160     linearSVC_classifier.train(training_set)
161     info("Accuracy Percent:", str(
162         (nltk.classify.accuracy(linearSVC_classifier, testing_set)) * 100))
163     pickling("data/pickles/linear_svc.pickle", linearSVC_classifier)
164 
165     print()
166     info("Current Algorithm: " + "Sklearn SVC")
167     SVC_classifier = SklearnClassifier(SVC())
168     SVC_classifier.train(training_set)
169     info("Accuracy Percent:", str(
170         (nltk.classify.accuracy(SVC_classifier, testing_set)) * 100))
171     pickling("data/pickles/svc.pickle", SVC_classifier)
172 
173     print()
174     # Voting classifier.
175     info("All classifiers are trained. Evaluating the voted classifier...")
176     voted_classifier = VoteClassifier(
177         nb_classifier, mnb_classifier, bnb_classifier, lr_classifier,
          ↪  linearSVC_classifier, SGD_classifier, SVC_classifier)
178 
179     info("Accuracy percent:",
180           str((nltk.classify.accuracy(voted_classifier, testing_set)) * 100))
```

The training output is like this,

```
# xiaoxing @ bogon in ~/Projects/tweet-sentiment-analysis on git:master x [15:49:34]
$ python train_classifiers.py
[INFO] (02:37:16) Training classifiers. This may take few minutes to finish.
[INFO] (02:37:16) Accessing the corporas...
[INFO] (02:37:31) Getting top 5000 words as features...
[INFO] (02:37:31) Tokenizing and finding features for training...
[INFO] (02:37:54) Length of the feature sets: 10662

[INFO] (02:37:54) Training and successive pickling of the classifiers...
[INFO] (02:37:54) This will take much time. Be patient.

[INFO] (02:37:54) Current Algorithm: NLTK Original Naive Bayes
[INFO] (02:39:00) Accuracy Percent: 71.6012084592145

[INFO] (02:39:00) Current Algorithm: Sklearn Multinomial Naive Bayes
[INFO] (02:39:39) Accuracy Percent: 71.45015105740181

[INFO] (02:39:39) Current Algorithm: Sklearn Bernoulli Naive Bayes
[INFO] (02:40:14) Accuracy Percent: 73.26283987915407

[INFO] (02:40:14) Current Algorithm: Sklearn Logistic Regression
[INFO] (02:40:47) Accuracy Percent: 74.77341389728097

[INFO] (02:40:47) Current Algorithm: Sklearn SGD classifier
/Users/xiaoxing/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarni
: max_iter and tol parameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDClassifier'>
 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to m
_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
[INFO] (02:41:26) Accuracy Percent: 72.35649546827794

[INFO] (02:41:26) Current Algorithm: Sklearn Linear SVC
[INFO] (02:42:03) Accuracy Percent: 71.45015105740181

[INFO] (02:42:03) Current Algorithm: Sklearn SVC
[INFO] (02:43:28) Accuracy Percent: 48.338368580060425

[INFO] (02:43:28) All classifiers are trained. Evaluating the voted classifier...
[INFO] (02:44:34) Accuracy percent: 72.9607250755287
```

After we have the trained model, we shall apply the analysis to all tweets we have. Before that, a sentiment analysis function is implemented.

```python
import pickle
import random
from statistics import mode

import nltk
from nltk.classify import ClassifierI
from nltk.classify.scikitlearn import SklearnClassifier
from nltk.tokenize import word_tokenize
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.svm import SVC, LinearSVC

from vote_classifier import VoteClassifier


classifiers = {
    "NLTK Naive Bayes": "data/pickles/original_naive_bayes.pickle",
    "Multinomial Naive Bayes": "data/pickles/multinomial_naive_bayes.pickle",
    "Bernoulli Naive Bayes": "data/pickles/bernoulli_naive_bayes.pickle",
    "Logistic Regression": "data/pickles/logistic_regression.pickle",
    "LinearSVC": "data/pickles/linear_svc.pickle",
```

```
    "SVC": "data/pickles/svc.pickle",
    "SGDClassifier": "data/pickles/sgd.pickle"
}

trained_classifiers = []
for classifier in classifiers.values():
    with open(classifier, "rb") as fh:
        trained_classifiers.append(pickle.load(fh))

voted_classifier = VoteClassifier(*trained_classifiers)

word_features5k_f = open("pickles/word_features5k.pickle", "rb")
word_features = pickle.load(word_features5k_f)
word_features5k_f.close()


def find_features(document: str, features: list):
    """The feature finding function, using tokenizing by word in the
    ↪  document.

    Arguments:
        document {str} -- Document
        features {list} -- List of features

    Returns:
        [type] -- [description]
    """

    words = word_tokenize(document)
    _features = {w: (w in words) for w in features}
    return _features


def sentiment(text):
    """Sentiment function.

    Arguments:
        text {str} -- Tweet string.

    Returns:
        str -- sentiment mode (pos or neg)
        int -- confidence
    """

    feats = find_features(text, word_features)
    return voted_classifier.sentiment(feats)
```

Then, use a multi-processing technique sentiment˙calculation˙multithread.py to apply to all rows.

```
print("Loading datasets... It may take a longer time.")
from sentiment import sentiment
import pickle
import pandas as pd
import numpy as np
from multiprocessing import cpu_count, Pool


print("The following result should be neg and 1.0")
print(sentiment("He is an incapable person. His projects are totally
 ↪  senseless."))

cores = cpu_count()  # Number of CPU cores on your system
partitions = cores // 4 or 1  # Define as many partitions as you want


def parallelize(data, func):
    data_split = np.array_split(data, partitions)
    pool = Pool(cores)
    data = pd.concat(pool.map(func, data_split))
    pool.close()
    pool.join()
    return data


def par_func(cs):
    print("Processing batch of", len(cs.index))
    cs["sentiment"] = cs["text"].apply(sentiment)
    return cs


def main(i):
    print("Reading tweet file", i)
    cs = pd.read_csv("full_tweets_" + str(i) + ".csv")
    print("Total tweets", len(cs.index))
    print("Detecting sentiment in parallel...")
    cs = parallelize(cs, par_func)
    cs.to_csv("full_tweets_with_sentiment_" + str(i) + ".csv")
    print("---------")


if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser(description='Calculate tweets sentiment')
    parser.add_argument("index")
    args = parser.parse_args()

    main(args.index)
```

3.3 Accuracy

The accuracy varies because we randomly our training sets. But it should be stable at around $[65, 75]$. This is a demo run:

- NLTK Multinomial Naive Bayes: 72.9607250755287
- Sklearn Multinomial Naive Bayes: 70.2416918429003
- Sklearn Bernoulli Naive Bayes: 72.35649546827794
- Sklearn Logistic Regression: 70.69486404833837
- Sklearn Linear SVC: 67.97583081570997
- Sklearn SGD classifier: 67.06948640483384
- Voted Classifier: 71.75226586102718

3.4 Analysis

After we have the dataset and the sentiment data, we can now do analysis. Since we have limited memory, we need to optimize the usage. Here we mainly convert categorical data into binary numbers, and choose the correct data types (csv˙optimize˙to˙pickle.py).

```
# coding: utf-8

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from tqdm import tqdm, tqdm_notebook
tqdm.pandas(tqdm_notebook)

df = pd.DataFrame()

# If failed to import, run
# `sed -e 's/\r/ /g' full_tweets_with_sentiment_7.csv > 7.csv` on all
for i in range(1, 8):
    print("Reading index", i)
    d = pd.read_csv(open("data/" + str(i) + ".csv", 'r'), encoding='utf-8',
                    engine='c', low_memory=False)
    df = pd.concat([d, df])
    print("File rows", len(d.index), "Total rows", len(df.index))


df['user.lang'] = df['user.lang'].astype('category')
df['user.time_zone'] = df['user.time_zone'].astype('category')
df['lang'] = df['lang'].astype('category')
df['source'] = df['source'].astype('category')
df['user.profile_background_color'] =
  ↪  df['user.profile_background_color'].astype(
```

```
    'category')

df['user.created_at'] = pd.to_datetime(df['user.created_at'])
df['created_at'] = pd.to_datetime(df['created_at'])

df['favorite_count'] = df['favorite_count'].apply(
    pd.to_numeric, downcast='unsigned')
df['user.listed_count'] = df['user.listed_count'].apply(
    pd.to_numeric, downcast='unsigned')

df.select_dtypes(include=['int64']).describe()  # Pick columns for
↪ improvement

df.to_pickle('data/all.pickle')  # Save the optimized object
```

And, run the score˙calculation.py. This will do two parts, one convert the sentiment data into a score ranged from -1 to 1. If the score is between -1 and 0, it means negative, otherwise, positive. Second is to predict what the tweet is talking about, Trump or Hillary. Here we use a naive method – we have two preset keywords, one for Trump and another for Hillary. If the whole tweet is talking about Trump (no Hillary keyword appear), then the score will be multiplied by -1, while about Hillary by 1. We assume that negative tweets about Trump indicate the positive mind on Hillary, and vice versa. If a tweet is talking about two things, we will then split it into sentenses, and find the most object. After this transformation, the final score will still be ranged from -1 to 1, but now the $[-1, 0)$ part is about Trump, lower then more supportive, and the $(0, 1]$ part is about Hillary, higher then more supportive.

```
# coding: utf-8

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

print("Loading all models for sentiment calculation...")
from sentiment import sentiment as s

from tqdm import tqdm, tqdm_notebook
tqdm.pandas(tqdm_notebook)

df = pd.read_pickle("data/all.pickle")

import ast


def score(senti):
    # Since the previous sentiment calulation will save the score and
    ↪ category into a string, we need to eval it.
    senti = ast.literal_eval(senti)
    if senti[0] == 'neg':
        return -1 * float(senti[1])
```

```
    return float(senti[1])


# Make the score range from [-1, 1]
df['sentiment_score'] = df['sentiment'].progress_apply(score)


def category(text):
    if any(map(lambda i: i in text, trump_keywords)) and any(map(lambda i: i
    ↪  in text, hillary_keywords)):
        return 0
    if any(map(lambda i: i in text, trump_keywords)):
        return -1
    elif any(map(lambda i: i in text, hillary_keywords)):
        return 1
    else:
        return 0


def sentiment(text):
    sen = s(text)
    if sen[0] == 'neg':
        return -1 * float(sen[1])
    return float(sen[1])


def category_and_score(entry):  # -1 trump, 1 hillary
    '''
    Find the category of tweet string.
    '''
    text = entry['text']
    if any(map(lambda i: i in text, trump_keywords)) and any(map(lambda i: i
    ↪  in text, hillary_keywords)):
        if "." in text:
            split_sentenses = text.split('.')
            s = {s: category(s) * sentiment(s) for s in split_sentenses}
            score = sum(s.values())

            if not score:
                return 0
            return score / len(s)
        else:
            return 0
    else:
        return float(category(text)) * float(entry['sentiment_score'])


trump_keywords = ['trump', 'yourefired', 'republi', 'gop']
hillary_keywords = ['hillary', 'madampresident', 'democrat']

```

```
71
72 df['score'] = df.progress_apply(category_and_score, axis=1)
73
74 df['score'].describe()
75
76 trump_scores = df[df['score'] < 0.0]['score']  # trump
77 hillary_scores = df[df['score'] > 0.0]['score']  # hillsry
78 non_scores = df[df['score'] == 0]['score']  # undeciable or no preference
79
80
81 trump_scores.describe()
82 hillary_scores.describe()
83 non_scores.describe()
84
85
86 results, edges = np.histogram(trump_scores, normed=True)
87 binWidth = edges[1] - edges[0]
88 plt.bar(edges[:-1], results * binWidth, binWidth)
89
90 results, edges = np.histogram(hillary_scores, normed=True)
91 binWidth = edges[1] - edges[0]
92 plt.bar(edges[:-1], results * binWidth, binWidth)
93 plt.show()
```

```
In [8]: df['score'] = df.progress_apply(category_and_score, axis=1)
100%|██████████| 4695447/4695447 [36:54<00:00, 2120.42it/s]
```

Here is some descriptive information on the score distribution.

- For Trump:

  count : 106121.000000

  mean : -0.915455

  std : 0.151850

  min : -1.000000

  25% : -1.000000

  50% : -1.000000

  75% : -0.800000

  max : -0.033333

- For Hillary:

  count : 196243.000000

  mean : 0.959576

  std : 0.111571

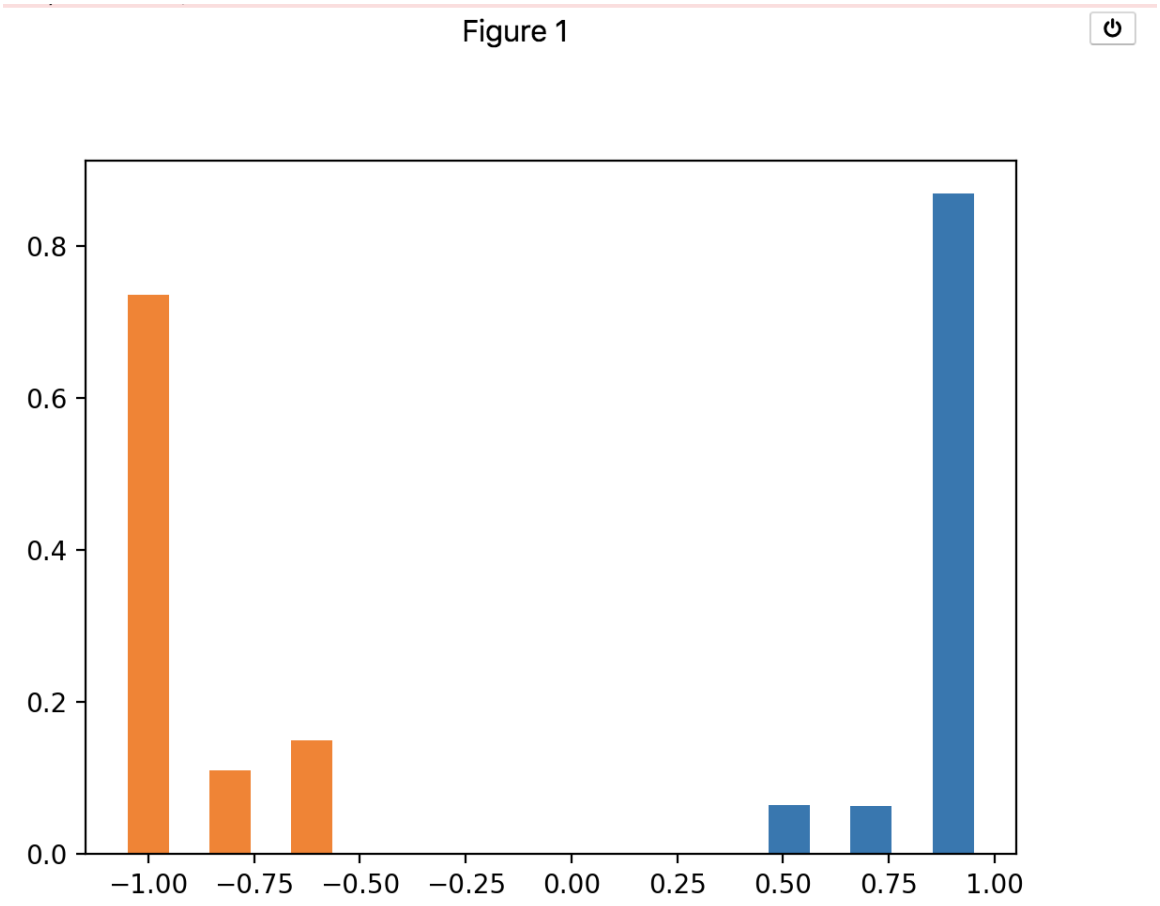
min : 0.028571

25% : 1.000000

50% : 1.000000

75% : 1.000000

max : 1.000000



4 Conclusion and Findings

References

[Let15] Letian: *Three common models of Naive Bayes: Gaussian, Multinomial, Bernoulli.* http://www.letiantian.me/2014-10-12-three-models-of-naive-nayes/. Version: Mar 2015

[lr-] *1.1.11. Logistic regression.* http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

[PVG+11] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[twi18] *Twitter Terms of Service*. https://twitter.com/en/tos. Version: Apr 2018

[Wik18] Wikipedia contributors: *Naive Bayes classifier — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Naive_Bayes_classifier&oldid=838385521. Version: 2018. – [Online; accessed 29-April-2018]