Classifying Personality by Applying Naive-Bayes Classification to Twitter

Patrick Maedgen
Texas A&M University
College Station, TX USA
pmaedgen@tamu.edu

Christoper Springstead
Texas A&M University
College Station, TX USA
chris1a.m@tamu.edu

Abstract

Social media platforms such as Twitter are one of the largest resources for mining personal information of users. Twitter allows nearly anyone to express their thoughts, interests, or commentary in up-to 280 character posts called "tweets". Meaning, these tweets are lenses into an individuals personality. We believe that using the hashtags from tweets that we can create a Naive-Bayes classifier based on six different labels: sad, happy, shy, social, introvert, extrovert. With these six labels we will be able to construct a training set to train our model. Then given a sample tweet we will be able to classify given tweet into one of the six labels.

1. Introduction

With social media having a growing impact on the future of society, more people than ever are turning to platforms such as Twitter and Facebook to express their opinions and beliefs. From this we can see that a concrete way to determine where a person stands on a topic or issue is also very important. This is the main goal of our system, to understand people based on their social media presence.

Initially we thought that based on the collection of tweets that we would be able to accurately rank tweets based on the Five-Factor model. However, we found this task to be difficult without a known data-set to train our model. To course correct we decided to modify how we collected data in order to create a data set with known labels for personality. Now, we have created data set of tweets with the six following labels: sad, happy, shy, social, introvert, extrovert. To verify that these labels are accurate for each tweet we separated each class of tweets into their own file which are then loaded all together in the system.

Before we can process these tweets in our Naive-Bayes Classification system we are going to need to collect the data (tweets) and pre-process it. In this paper, we propose a system that mines tweets from Twitter based on a given hashtag and then labels them along with other metrics that come along with tweets (likes, retweets, etc.). The labeling of the tweets is done at the same time the tweets are saved to their respective class .csv files. The data collection will be further discussed in section 3.

Once tweets have been collected for each of the six classes they will be loaded and formatted by a tweet loading script. This script takes in all of the .csv files and creates a large dictionary of tweets. This will be discussed in more detail later (section 3). After being loaded into the system we construct a training data-set and a testing data-set from the larger set of tweets. To do this we took every 10th tweet and added it to the testing data-set while every other tweet is being added to the training set. With this separation, we are able to train our classification system on the training data and then test the system with the testing set. Based on the information generated from the testing set we calculated the accuracy, precision, recall, and F1 values for the predicted classes made by our system. The specifics will be discussed later.

2. Literature Survey

It has long been understood that there exists a correlation between a person's online behavior and their personality. As there is a significant amount of prior research into this relationship. For example, it is possible to predict certain traits and attributes of a person based on what they "like" on Facebook [1]. One approach that is often used to classify a person's personality is the Five-Factor Model (Big5) [2]. This is because it is able to encompass many different personality constructs into just six traits that describe a person's personality [3].

One possible learning model is a support vector machine (SVM) [2]. As per [2], SVM produced more accurate results than other techniques such as linear regression and least absolute shrinkage and selection operator (LASSO).

[5] uses binomial logistic regression to calculate the probability sentiment behavior of Facebook users. [5] then analyzes the linearity of the five traits concerning the logit transformation of the sentiment analysis. The author found that all big5 personality traits to be linearly correlated to the

logarithm of the probability sentiment.

Just within the past decade researchers have begun to apply deep-learning algorithms. Kalghatgi et al. [4] has applied neural networks to make predictions about social media users' personality according to the Big5.

We chose to use to a supervised learning technique such as SVM due to the complexity that comes with building an unsupervised model for this task. Also, there already exists trained models in literature that use supervised learning techniques.

Based on [8] we know traditionally that SVM was used to do binary classification. However, there are modern day approaches as discussed in [8] such as one vs. all and multi-class SVM which can be used for non-binary classification.

Since our final model is based upon six different class labels and not the Five-Factor model we knew we needed a model that is capable of doing multi-class classification. We decided to go with Naive-Bayes classification system because it would allow for easy multi-class classification without much change to Bayes Theorem and algorithm.

Traditionally a Naive-Bayes classifier wouldn't produce the best results but based on [9] we know that it's possible to make improvements. To make positive changes to the Naive-Bayes classifier we will conduct document normalization which will allow for more consistent results; this will be discussed in more detail later in the paper.

3. Technical Plan

3.1. Data Collection

To generate the tweet data set we plan to mine Twitter to extract tweets from users' timelines. Doing this requires us to use the Twitter API and the python library Tweepy. One can gain access to Twitter API authorization keys with an approved Twitter developer account. Based on the parameters given to the mining function will determine the type of data collected. For example, we can choose to collect re-tweets that the specified user re-tweets or not. To get the required tweets we specify the hashtag of which tweets are tagged with and then retrieve those tweets into a list. Depending on the hashtag scraped that will determine the label in which that tweets gets. For our system we have six labels: sad, happy, shy, social, introvert, and extrovert. The fields that we scrape from the tweets are as follows: tweet text, tweet id, length, source, number of likes, number of retweets, and label. The format for the data-set files will be discussed more in section 3.2.

3.2. Data Pre-processing

Before we can begin classifying the data the data-set csv files must have a consistent layout so the sections scraped go to their corresponding tweet object field. To do this we used the Pandas DataFrame object to construct a table like structure within our csv files. This allows the data being scraped to be consisted across all the files. In addition to the files needed a consistent structure the data-sets must be labeled. To do this depending on the hashtag scraped we appended a enumerated value to each of the tweets within the data-set which will tell us which class a particular tweet belongs to. This is crucial for our systems training. Without the labels the system will not have any idea on what to look for when doing the Naive-Bayes classification on the testing data-set. Though the tweets don't come labeled directly from twitter we know how to assign them a label because at one time we are only scraping one specific hashtag (class).

3.3. Web-scraper System Flow

Before we could start to even think about classifying tweets, we needed to generate a data-set that had all the relevant information about the tweet. To do this we implemented a web-scraper. The main library that we used to make this possible is Tweepy. Tweepy allows us easy access to all of Twitter's RESTful API methods. However, twitter requires all requests to use OAuth for authentication, so we had to create a Twitter developer account to get API keys and access tokens. Once these initial steps were followed we could begin the actual tweet scraping.

We used "API.search" which returns tweets that correspond to a specific query, in our case a hashtag. With "API.search" we had to iterate through the returned tweets and return them in a list. Once we had the tweets in a list we had to create a consistent format for storing the tweets to be used later in our Naive-Bayes classifier.

To create the data-sets of tweets we knew we wanted the files to be in csv format to make parsing trivial. However, just writing to these csv files using the basic csv import in Python caused a few major bugs. One where if tweets spanned multiple lines and the other where the tweet text was being converted to byte strings due to the build in Python .encode() method. These byte strings contained odd characters since we were encoding to utf-8. To fix the problem where the tweets spanned multiple lines when loading the tweets via the tweet loader we declared a delimiter which separated the data only on commas and not white space or newlines. To fix the tweet text being written to the csv file as a byte string we used a Python library called pandas which allowed us to use a data structure called a Data Frame which is a 2D table like structure. Now before writing to the csv file the list of tweets returned from the scraping functions we created a Data Frame with the following properties: tweet text, tweet id, length of the tweet, date, source, the amount of likes, and the number of retweets. One additional item was added for tweets that were scraped based on a hashtag, their corresponding label. Now with these tweets in the Data Frame we can use the built in Data Frame.toCsv() function found in the pandas library to make

sure all the data is correct when written and not a byte string. After writing the data-set is complete and ready to be loaded into other instances of our project.

3.4. Classification System Flow

Once we have the data loaded from the tweet loader we call the train function on the train dataset. This function takes in a dictionary of tweets which maps a unique tweet id to a tweet object defined in the tweet loader class. We then need to populate the data structures within our Naive-Bayes classification class. To start we take the length of the supplied tweets dictionary to get the total number of tweets processed. We then begin iterating through the tweets dictionary checking each tweets label and updating the dictionary which maps each label to the number of tweets that correspond to that label. Next we need to populate our term frequency vectors, to do this we need to iterate through the list of words found in each of the tweets. For each new appearance of a word we set its value to one, and if the word is already found in the term frequency vector we simply get the value and update it by one. After updating this term frequency vector, we add each word to our set that contains unique words.

Now that our data structures have been made we can move onto the predict function. This function takes in a query which is a tweet. To start we need to make this string of text into a list of words. We then declare a dictionary which is going to map labels to values calculated by the Naive-Bayes formula. Next we iterate through the labels and calculate values using the Naive-Bayes formula and add them to our dictionary. To get the predicted label for a tweet we take the biggest value found in our dictionary and return its label.

3.5. Classification

Originally we planned to use multi-class SVM to classify a persons personality, but decided to use a Naive-Bayes approach as the modification to the algorithm was much simpler compared to SVM. We still need multi-class classification because of our six labels. The following are the equations we used to write our classification system.

$$P(w|c) = \frac{COUNT(w,c)+1}{(\sum_{w' \in V} COUNT(w',c)) + |V+1|}$$

The above equation is for calculating the probability of a word w for a candidate class c. In addition we had to calculate the prior probabilities for each class. The $COUNT(w,c)$ function in the above equation is the term frequency for a specific word w in the current class c. V in the above equation is the unique set of words found in the data-set. With this function we can get prediction values generated from the training data and apply those to the given test tweet.

Now that we have these probabilities for the test-data set. We used maximum posteriori (MAP) estimation to get the predicted class. We followed the below formula,

$$c_{NB} = argmax_{c_j \in C}[\log(P(c_j) + \sum_{i \in doclength} \log(P(w_i|c_j))]$$

This equation tells us that we will pick the class label that has largest sum of the logarithm of prior and likelihood probabilities. Since this value will be a number we simply pass that number to our enumerated class labels and it will return the corresponding string label.

In the code there are two main functions for this classification, a train function and a predict function. In the train function we are taking the training data-set and processing it into four sub-fields: `n_tweet_total`, `n_tweet`, `vocab`, and `s`.

`n_tweet_total` is an integer value which keeps track of the total number of tweets processed. `n_tweet` is a dictionary which maps a label to the total number of tweets for the corresponding label. `vocab` is our term frequency vector which maps a label to another dictionary which maps terms to their number of occurrences in a given label. `s` is a set of unique words for all tweets in the data set.

3.6. Evaluation

The main goal of our classification system is to determine a persons behavior based on their Twitter presence (tweets). We do this in our system by using a multi-class Naive-Bayes classification algorithm. In order to judge how well our system is working we have decided to calculate a few metrics based on the predicted label for the tweet and the known label. The values we calculated are: accuracy, precision, recall, and F1. Accuracy is calculated for total system and the precision, recall, and F1 values are calculated for each label. These values are calculated in an evaluation function which calls predict for every tweet in the test data-set.

Accuracy we know refers to the closeness of a value to a specific measurement. In our system this is the most important measure and is calculated for the total system. We used the following equation,

$$Accuracy = \frac{total \quad number \quad correct}{total \quad tested}.$$

This tells us how well our system did on the test data-set, which is comprised of every 10th tweet in the overall data-set. In our implementation of Naive-Bayes multi-class classification we got the following for our calculation.

$$Accuracy = 0.8034188034188035$$

Precision we know refers to the closeness of the measurements to each other, meaning for our system the number

of times a predicted label is the same even though the tweet is different. To calculate this in our system we followed the below equation.

$$Precision = \frac{TP}{(TP + FP)}$$

Where TP refers to true positive and FP refers to false positive. This was calculated for each label, below are the results.

Label	Precision
Sad	0.45038167938931295
Happy	0.4791666666666667
Shy	0.5000000000000000
Social	0.4818181818181818
Introvert	0.3937007874015748
Extrovert	0.5000000000000000

From these results we can see that our precision isn't high but also isn't low. This means that it's often that we see labels being predicted the same and are grouped together even if they aren't the correct label for the tweet.

The next value that we calculated to evaluate our Naive-Bayes classification system is recall. Recall for our system is the fraction of relevant tweets for a specific label that were successfully predicted. We use the below equation to calculate this value.

$$Recall = \frac{TP}{(TP + FN)}$$

Where TP stands for true positive and FN stands for false negative. This once again was calculated for each label, below are the results.

Label	Recall
Sad	0.8194444444444444
Happy	0.9200000000000000
Shy	1.0000000000000000
Social	0.9298245614035088
Introvert	0.6493506493506493
Extrovert	1.0000000000000000

Compared to the precision we can see that our recall is high. This means that our system returned more relevant labels than irrelevant labels. One reason for this is that for a few our of hashtags there may have not been a lot of tweets so if it returned a small amount of relevant ones, it's possible that it was actually a high percent of the data-set for that hashtag.

The last metric that was calculated was an F-measure (F1). This measure is a weighted harmonic mean of precision and recall, in our calculation we have them at an equal weight. This value is a measure of a systems accuracy (for each label in this case). The formula below was followed for the calculation.

$$F1 = 2 * \frac{precision * recall}{(precision + recall)}$$

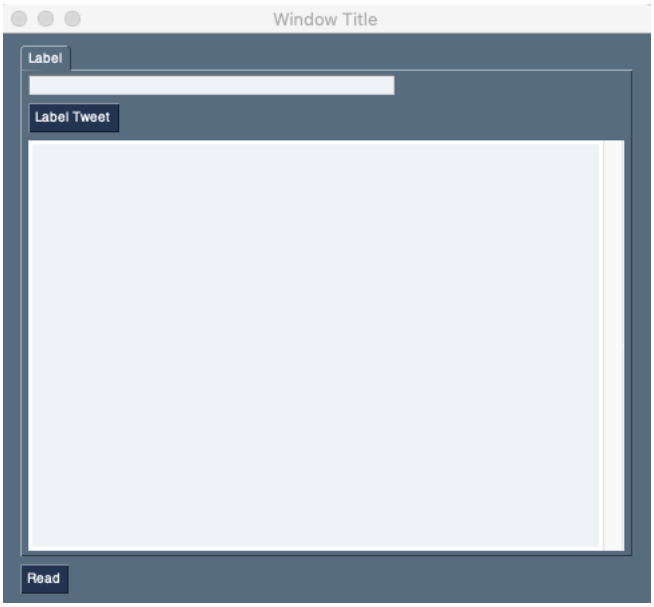
This was calculated for each label using their respective precision and recall, below are the results.

Label	F-measure
Sad	0.5812807881773397
Happy	0.6301369863013699
Shy	0.6666666666666666
Social	0.6347305389221557
Introvert	0.49019607843137253
Extrovert	0.6666666666666666

From these results we believe that we have built an efficient system to classify tweets into six possible classes. Using this information one can infer that depending on what our system returns will give an idea of a persons personality based on a given tweet. Some oddities in the evaluation metrics can be from the six of the data-set. For some of the tweets there were few tweets meaning that our calculations could easily be influenced from small irregularities. To see these metrics you must run the classifier via command line as follows: python3 hashtagClassifier.py. This will print the accuracy, precision, recall, and F1 values.

4. GUI

The main way a user would use our system to classify a tweet into one of the six labels is through the GUI. This GUI is very simple and allows a user to input a tweet they would like to classify. This then calls our Naive-Bayes classifier to label that specific tweet based on the training data-set. The predicted class is then output to the user. We have found that it works better and is more accurate when an actual tweet is used versus a random string entered into the field. Below is an image of what the GUI looks like. This GUI is ran by executing gui.py as follows: python3 gui.py.



When using the GUI to classify tweets the evaluation metrics will not be presented only the predicted class will be returned since we are classifying one tweet at a time.

5. Conclusion and Future Work

In this system we used Naive-Bayes classification on tweets that were scraped based on hashtag which corresponded to a specific label. With the scraped data-set we were able to create a training set and a testing set. We showed that when given labeled tweets we were able to predict other tweets with around 80% accuracy to their respective class in the testing set.

To improve this system for the future we could do a number of things. To start we could use a different classification model such as k nearest neighbour based on key words that express feelings or personality. Another possible, approach to improve the current system is to get a bigger data-set. Since we had to build the data-set and label it ourselves our data-set is relatively small. With a bigger data-set our Naive-Bayes classification system would get better and better since it would have even more data to train on. This would also increase the size of the testing set since the tweets are pull from the same over arching data-set. Another improvement that could be made in the future to improve the system is to assign weights to terms in the term frequency vector. This would allow terms that related to personality to be much more influentially in the prediction of class, whereas it's currently possible to miss-classify a tweet solely based on the fact that it contained enough filler words that matched another class even though it doesn't belong to that class. Many other things are possible with the information scraped from twitter to help us classify a persons personality that were not used in our system such as data and time which could help if a system was constructed to incorporate this data.

6. Task Assignment

This project was completed by the cooperation of Christopher Springstead and Patrick Maedgen, under the instruction of Dr. Zhangyang (Atlas) Wang. Patrick took charge of the web-scraping of the tweets and the construction of the data-sets, along with the data-formatting of the .csv files. Christopher mainly worked on the tweet loader script which read the data from the data-sets, our implementation of the Naive-Bayes classification algorithm, and also the construction of the simple user GUI. The tweets were scraped via Tweepy by accessing Twitters free API, with a Twitter approved Developer account.

7. References

[1] Kosinski, Michal, David Stillwell, and Thore Graepel. Private traits and attributes are predictable from digital records of human behavior. Proceedings of the National Academy of Sciences 110.15, 2013

[2] Carducci, Giulio Rizzo, Giuseppe Monti, Diego Palumbo, Enrico Morisio, Maurizio. (2018). TwitPersonality: Computing Personality Traits from Tweets Using Word Embeddings and Supervised Learning. Information (Switzerland). 9. 10.3390/info9050127, 2018

[3] McCrae, R.R. and John, O.P.. An Introduction to the Five-Factor Model and Its Applications. Journal of Personality, 60: 175-215. doi:10.1111/j.1467-6494.1992.tb00970.x, 1992

[4] M.P. Kalghatgi, M. Ramannavar, N.S. Sidnal. A neural network approach to personality prediction based on the bigfive model. Int. J. Innov. Res. Adv. Eng, 2015

[5] M. Mostafa. 'Modelling and Analysing Behaviours and Emotions via Complex User Interactions', PhD thesis, Cardiff Metropolitan University, UK, 2019

[6] fastText English Word Vectors. Available online: https://fasttext.cc/docs/en/english-vectors.html (accessed on 8 March 2020).

[7] NLPL word embeddings repository. Available online: http://vectors.nlpl.eu/repository/ (accessed on 8 March 2020)

[8] Hsu C., Lin C., A Comparison of Methods for Multi-class Support Vector Machines, National Taiwan University, TW

[9] Rennie, Jason D. M., Improving Multi-class Text Classification with Naive Bayes, Massachusetts Institute of Technology