|  |  |
| --- | --- |
| Name Of The Student | Aman Rai |
| Internship Project Topic | Automate Detection of different emotions from textual comments and feedback |
| Name of the Organization | TCS iON |
| Name of the Industry Mentor | Mr. Debashis Roy |
| Name of the Institute | Institute of Engineering & Management Kolkata |

|  |  |  |
| --- | --- | --- |
| Date | Day # | Hours Spent |
| 23 August 2020 | 4 | 4 |
| Activities done during the day: Sentiment Analysis versus Emotional Analysis: Same or Different? Today I have read some research papers and go through some online videos and articles to know about the emotional analysis and sentiment analysis and the similarities and differences between them.  Also take a brief discussion about The Twitter Sentiment Analysis Program in Python with Naive Bayes Classification.  **Sentiment Analysis versus Emotional Analysis: Same or Different?**  As much as you may want to plan for emotional analysis taking over as the superior big brother to sentiment analysis, I believe we are a long way off from that happening. Sentiment analysis may not be glamorous and shiny, but it does what it is supposed to and doesn’t require nearly as much effort and involvement that emotional analysis does. Even if NLP matures and emotional analysis requires fewer resources to perform, there is still a risk of alienating your customers and replacing the human element behind these sentiments with numbers and clockwork. Sentiment analysis and emotional analysis are both similar and different. There will be a time when we have a more clear idea of when emotional analysis is beneficial and when it is easier and better to stick with a less in-depth, sentiment analysis.  Sentiment and emotional analytics are often used in place of one another, when we talk about tracking and measuring consumer behaviors and sentiments. Yet, many regard emotional analysis as the successor to sentiment analysis; a more sophisticated solution that digs far past the simple negative and positive division and instead looks to analyze specific emotions (sad, happy, angry, and content) and how they lead to future behaviors. This seemingly upgraded version of sentiment analysis, however, has its drawbacks. It is a lot more involved and still an evolving form of analytics. Thus, it requires a greater investment of time and resources and the return on investment of “digging deeper” into customer emotions is still vastly untested.  Twitter Sentiment Analysis Program in Python with Naive Bayes Classification.  **Abstract— Twitter1 is a micro-blogging website which provides platform for people to share and express their views about topics, happenings, products and other services. Tweets can be classified into different classes based on their relevance with the topic searched. Various Machine Learning algorithms are currently employed in classification of tweets into positive and negative classes based on their sentiments, such as Baseline, Naive Bayes Classifier, Support Vector Machine etc. This paper contains implementation of Naive Bayes using sentiment140 training data using twitter database and propose a method to improve classification . Use of SentiWordNet along with Naive Bayes can improve accuracy of classification of tweets, by providing positivity, negativity and objectivity score of words present in tweets. For actual implementation of this system python with NLTK and python-twitter APIs are used**  Sentiment Analysis is a term that you must have heard if you have been in the Tech field long enough. It is the process of predicting whether a piece of information (i.e. text, most commonly) indicates a positive, negative or neutral sentiment on the topic. In this article, we will go through making a Python program that analyzes the sentiment of tweets on a particular topic. The user will be able to input a keyword and get the sentiment on it based on the latest 100 tweets that contain the input keyword  **Introducing Sentiment Analysis**  Also known as “Opinion Mining”, *Sentiment Analysis* refers to the use of Natural Language Processing to determine the attitude, opinions and emotions of a speaker, writer, or other subject within an online mention.  Essentially, it is the process of determining whether a piece of writing is positive or negative. This is also called the Polarity of the content.  As humans, we are able to classify text into positive/negative subconsciously. For example, the sentence “The kid had a gorgeous smile on his face”, will most likely give us a positive sentiment. In layman’s terms, we kind of arrive to such conclusion by examining the words and averaging out the positives and the negatives. For instance, the words “gorgeous” and “smile” are more likely to be positive, while words like “the”, “kid” and “face” are really neutral. Therefore, the overall sentiment of the sentence is likely to be positive.  A common use for this technology comes from its deployment in the social media space to discover how people feel about certain topics, particularly through users’ word-of-mouth in textual posts, or in the context of Twitter, their *tweets*.  **System Model**  Basic architecture of the sentiment model is explained using the block diagram in fig, which shows various phases for the sentiment analysis of real time tweets. Various steps involved in this system are as explained as follows.   Process Description  * Preparing the Test Set * Preparing the Training Set * Pre-Processing Tweets in Data Set * Naive Bayes Classifier * Testing the Model   **Preparing the Test Set**  As our task of Sentiment Analysis is one that focuses heavily on textual data, one would expect there to be a lot of text processing. This is definitely correct. In fact, both our Test and Training data will merely comprise of text.  I chose to start with the Test set in order to get you all warmed up for the Training set extraction part, as it will rely more on the API. Here is a bit of an overview of what we are about to do:   * Register Twitter Application to get your own credentials. * Authenticate our python script with the API using the credentials. * Create Functions to download tweets based on a search keyword.   Registering an application with Twitter is critical, as it is the only way to get authentication credentials. As soon as we get our credentials, we will start writing code. Step 3 is where the Test set lies. We will be downloading tweets based on the term that we are trying to analyze the sentiment on.  **Create Functions to download tweets based on a search keyword**  Now we can start on making a function that downloads the Test set that we talked about. Basically, this is going to be a function that takes a search keyword (i.e. string) as an input, searches for tweets that include this keyword and returns them as twitter.Status objects that we can iterate through.  *The caveat here, though, is that Twitter limits the number of requests you can make through the API for security purposes. This limit is 180 requests per 15-minute window.*  This means, we can only get up to 180 tweets using our search function every 15 minutes, which should not be a problem, as our Training set is not going to be that large anyway. For the sake of simplicity, we will limit the search to 100 tweets for now, not exceeding the allowed number of requests. Our function for searching for the tweets (i.e. Test set) will be  def buildTestSet(search\_keyword):  try:  tweets\_fetched = twitter\_api.GetSearch(search\_keyword, count = 100)    print("Fetched " + str(len(tweets\_fetched)) + " tweets for the term " + search\_keyword)    return [{"text":status.text, "label":None} for status in tweets\_fetched]  except:  print("Unfortunately, something went wrong..")  return None  As you might have expected, this function will return a list of tweets that contain our search keyword.  *Note that we coupled — into a JSON object — every tweet’s text with a label that is NULL for now. This is merely because we are going to classify each tweet as Positive or Negative later on, in order to determine whether the sentiment on the search term is positive or negative, based on the majority count. This is how Sentiment Analysis pragmatically works.*  **Preparing the Training Set**  In this section, we will also be using our Twitter API instance from the last section. However, we need to get some things out the way first. We will be using a downloadable Training set. The tweets of which were all labeled as positive or negative, depending on the content. This exactly what a Training set is for.  *A Training set is critical to the success of the model. Data is which needs to be labeled properly with no inconsistencies or incompleteness, as training will rely heavily on the accuracy of such data and the manner of acquisition*  **Pre-processing Tweets in the Data-Set**  Before we move on to the actual classification section, there is some cleaning up to do. As a matter of fact, this step is critical and usually takes a long time when building Machine Learning models. However, this will not be a problem in our task, as the data we have is relatively consistent. In other words, we know exactly what we need from it.  Let’s talk about what matters and what doesn’t matter in Sentiment Analysis. Words are the most important part (to an extent that we will talk about in the upcoming section). However, when it comes to things like punctuation, you cannot get the sentiment from punctuation. Therefore, punctuation does not matter to Sentiment Analysis. Moreover, tweet components like images, videos, URLs, usernames, emojis, etc. do not contribute to the polarity (whether it is positive or negative) of the tweet. However, this is only true for this application.  So we know what we need to keep in the tweets we have and what we need to take out. This applies to both Training and Test sets. So let’s make a our pre-processor class:   |  | | --- | | import re | |  | from nltk.tokenize import word\_tokenize | |  | from string import punctuation | |  | from nltk.corpus import stopwords | |  |  | |  | class PreProcessTweets: | |  | def \_\_init\_\_(self): | |  | self.\_stopwords = set(stopwords.words('english') + list(punctuation) + ['AT\_USER','URL']) | |  |  | |  | def processTweets(self, list\_of\_tweets): | |  | processedTweets=[] | |  | for tweet in list\_of\_tweets: | |  | processedTweets.append((self.\_processTweet(tweet["text"]),tweet["label"])) | |  | return processedTweets | |  |  | |  | def \_processTweet(self, tweet): | |  | tweet = tweet.lower() # convert text to lower-case | |  | tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL', tweet) # remove URLs | |  | tweet = re.sub('@[^\s]+', 'AT\_USER', tweet) # remove usernames | |  | tweet = re.sub(r'#([^\s]+)', r'\1', tweet) # remove the # in #hashtag | |  | tweet = word\_tokenize(tweet) # remove repeated characters (helloooooooo into hello) | |  | return [word for word in tweet if word not in self.\_stopwords] |   Note that our code removed duplicate characters in words as we metioned earlier (i.e. “delllllicious” became “delicious”). However, it did not remove duplicate words (i.e. “corn”) from the text, but rather kept them. This is because duplicate word play a role in determining the polarity of the text  **Naive Bayes Classifier**  Naive Bayes Classifier is a classification algorithm that relies on Bayes’ Theorem. This theorem provides a way of calculating a type or probability called posterior probability, in which the probability of an event A occurring is reliant on probabilistic known background (e.g. event B evidence). For example, if Person\_X only plays tennis when it is not raining outside, then, according to Bayesian statistics, the probability of Person\_X playing tennis when it is not raining can be given as:  P(X plays | no rain) = P(no rain | X plays)\*P(x plays)/P(no rain)  All you need to know for our task is that a Naive Bayes Classifier depends on the ever-famous Bayes’ theorem. Before we move on, let’s give a quick overview of the steps we will be taking   * **Build a vocabulary (list of words) of all the words resident in our training data set.** * **Match tweet content against our vocabulary — word-by-word.** * **Build our word feature vector.** * **Plug our feature vector into the Naive Bayes Classifier.**   **Testing the Model**  This corresponds to the final evaluation that the model goes through after the training phase (utilizing training and validation sets) has been completed. This step is critical to test the generalizability of the model (Step 3). By using this set, we can get the working accuracy of our model. It is worth mentioning that we need to be subjective — and honest — by **not** exposing the model to the test set until the training phase is over. This way, we can consider the final accuracy measure to be reliable.  Training a model involves looking at training examples and learning from how off the model is by frequently evaluating it on the validation set. However, the last — and most valuable — pointer on the accuracy of a model is a result of running the model on the testing set when the training is complete.  **Conclusion**  Sentiment Analysis is an interesting way to think about the applicability of Natural Language Processing in making automated conclusions about text. It is being utilized in social media trend analysis and, sometimes, for marketing purposes. Making a Sentiment Analysis program in Python is not a difficult task, thanks to modern-day, ready-for-use libraries. This program is a simple explanation to how this kind of application works. | | |