# SENTITEXT: TWEETS CLASSIFIER

## Table of Contents

Background
Motivation
Literature Review
Architecture
Implementation
  - Classifiers Used
  - Libraries Used
Initial Approach
Current Approach
  - Lexical/Dictionary-based processing using hand-coded rules
  - Classification using NLP and supervised machine learning techniques
    - Document Processor
    - Username and RT Remover
    - Special Characters Remover
    - Word Normalizer
    - Stopwords Remover
    - Parts-of-Speech Tagger
    - Pro/Noun Remover
    - String Tokenizer
    - Spell Checker
    - SynSet Generator
    - Word Sense Disambiguator
    - Word Stemmer
    - Unigrams
    - Bigrams
    - tf and tf-idf
      - tf and tf-idf Generator for Unigrams
      - tf and tf-idf Generator for Bigrams
    - Feature Vector to Classifier
Classifiers
  - Multinomial Naive Bayes Independence Classifier
  - Support Vector Machine (SVM)
Results
Conclusion
Background

Very large amounts of Twitter information is available on the World Wide Web in the form of tweets with a length of 140 characters. These messages or posts can be extracted using the Twitter API and analyzed for various applications. Some of these are topic classification, presidential electoral opinion, reviews of a product, among others. The information present within these documents are studied using its sentence structure to provide useful insights required for knowledge discovery. I apply lexical (hand-coded rules with the use of dictionaries) and natural language-based processing approaches for investigating the objective of a one or more sentences. Furthermore, machine learning algorithms such as Naive Bayes and Support Vector Machines are employed to classify preprocessed text as positive/happy or negative/sad label to detect the opinion or sentiment of the tweet based on the mood of the user posting it. Datasets of 1891 happy tweets and 1894 sad tweets as training data are used and 1896 happy tweets and 1895 sad tweets as testing data are used; along with a dataset of 800 thousand positive and 800 thousand negative tweets as training data and 498 tweets as testing data; along with 7086 positive texts and negative texts as training data and 33052 texts as testing data are used for sentiment analysis.

Motivation

My objective is to classify the sentiment of a tweet/text by applying lexical and supervised machine learning techniques by cleaning and preparing data with the help of

1 "Character Counting | Twitter Developers." 2014. 16 May. 2016 [https://dev.twitter.com/overview/api/counting-characters]


5 "For Academics - Sentiment140 - A Twitter Sentiment Analysis Tool." 2014. 16 May. 2016 [http://help.sentiment140.com/for-students/]

6 "Data - UMICHI SI650 - Sentiment Classification | Kaggle in Class." 2012. 16 May. 2016 [https://inclass.kaggle.com/c/si650winter11/data]
natural language processing. A comparative study of two algorithms - Naive Bayes and Support Vector Machines is performed. I achieve feature vectors or feature extraction from the tweet, required for the classifiers, with the help of hand-coded rules and language processing. The core of my capstone project lies in detecting the person’s mood or emotion based on the positivity or negativity of the tweet. I plan to achieve my goal with the help of different techniques for processing a tweet, which is in the form of a natural english written language. The field of sentiment analysis is very useful for determining the opinions of different people on various topics mentioned above as they post on social media websites. This area helps an analyst to classify the popularity or decline of a certain trending news or based on the input hash tag provided by an end-user for a sentiment analysis-based application.

**Literature Review**

I reviewed a course, books and research papers on [7](#) [8](#) [9](#) [10](#) [11](#) [12](#). For lexicon or dictionary based approaches I referenced to Dr. Peter Turney’s research paper on Thumb’s up or thumbs down. This was a great start towards an unsupervised machine learning approach to perform sentiment analysis. Dr. Turney’s approach is to use a dictionary of words and then find the mutual importance of a given word and its index with respect to the positive and negative words. However, I tweaked his approach. My approach is discussed under the section Initial Approach. Dr. Bing Liu does an excellent job at providing details and explaining concepts for both a lexicon and a machine

---


8 Turney, Peter D. "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews." *Proceedings of the 40th annual meeting on association for computational linguistics* 6 Jul. 2002: 417-424.


12 "Natural Language Processing - | Coursera." 2013. 16 May. 2016 [https://class.coursera.org/nlp/lecture](https://class.coursera.org/nlp/lecture)
learning based approach for NLP and sentiment analysis. Dr. Liu provides a list of positive and negative lexicons and uses semantic orientation of a word to determine the sentiment of the text. In one of his other papers on Thumbs up?: sentiment classification using machine learning techniques, Dr. uses machine learning algorithms and the concepts of natural language processing to detect the sentiment of a sentence. Dr. Liu finds the parts-of-speech of words and uses a 2-gram approach. His goal is to perform different machine learning classifiers’ analysis using three different algorithms: Naive Bayes, Maximum Entropy and Support Vector Machines. Dr. Liu also takes into consideration the position of the word and uses that as a feature for text classification. However, Dr. Liu did not generate a synonym set using WordNet. SynSet generation is important for normalizing similar word synonyms to a common word. I also reviewed Dr. Manning’s course on Natural Language Processing from where I got an in-depth understand of NLP concepts. Most of my research work is self-performed with the help of the above references.

Architecture
Text Classification

I am using Training and Classification as two important tasks of my machine learning classifier. An input file, preferably a text file, is read, preprocessed, cleaned and prepared for obtaining features out of it. I generate a vector or list of features extracting from the processed text file and feed the document matrix to my machine learning classifiers - Multinomial Naive Bayes Independence Classifier and Support Vector Machines. A label class of positive or negative class is provided to the classifier so that it learns from the training data or examples. This is how training a machine learner takes place. The next step is to test my machine learner to determine how well have I trained it. The input to the classifier will be everything given to it in the training part except for the label. The task of a machine learner is a label the text based on the positive or negative classification labels.

**Implementation**

**Classifiers Used**

Multinomial Naïve Bayes Independence, Support Vector Machines.
Libraries Used

Porter Stemmer Algorithm for Word Stemming, OpenNLP POS Tagger or StanfordNLP POS Tagger, WordNet, SentiWordNet, StanfordNLP, LIBSVM.

Initial Approach

My initial approach was to classify a person’s mood with the help of Stanford’s CoreNLP Sentiment analysis suite. I used real time data extracted from my newly created Twitter test account with handle @hash1302. The tweets were extracted using Twitter4J API using my Twitter credentials for OAuth (Open Authorization). These access tokens allow me to access Twitter data without revealing my password and thus use the API to retrieve Twitter data. Tweets were then fed to the Stanford CoreNLP Library which performed natural language or sentiment analysis for me.

Working: The Tweet Manager class performs a query to search for a given topic of interest. The results are stored in a QueryResult instance and added to a tweetList. The Stanford CoreNLP annotators - tokenize, ssplit, pos, lemma, parse and sentiment are used. Tokenize is used to perform word tokenizer; ssplit is used to perform sentence

---

14 "Porter stemmer in Java - Tartarus." 2006. 16 May. 2016 <http://tartarus.org/martin/PorterStemmer/java.txt>


18 "Java API for WordNet Searching (JAWS) - Lyle School of Engineering." 2008. 16 May. 2016 <http://lyle.smu.edu/~tspell/jaws/>


20 "Stanford CoreNLP – a suite of core NLP tools ... - GitHub Pages." 2015. 16 May. 2016 <http://stanfordnlp.github.io/CoreNLP/>

split or tokenization; pos is used for parts of speech. Lemma is used to return the base
form of a word; parse is used to derive a parse tree; sentiment to used to derive the
sentiment of a document. The Stanford CoreNLP first instantiates its object to work with
its library. I feed in all the annotators. I split the tweet in sentences using ssplit and start
analyzing each of the sentence. I use word tokenization to split the words and then draw
a parse tree to get the predicted class of the sentiment. I divide the sentiments into five
categories - very negative, negative, neutral, positive, and very positive.

This approach served as a baseline in the field of sentiment analysis using natural
language and machine learning. I needed to study these fields in depth and apply the
techniques to build my own text classification tool or an analysis using concepts learned
from NLP and machine learning.

Current Approach

**Lexical/Dictionary-based processing using hand-coded rules**

Negative Words Dictionary: Curse, Hell, Hate, Irk.
Negation Words Dictionary: Not, Neither, Nor.

Logic: Tokenize document identifying every token using dictionaries, eventually
normalizing +/- polarities. The approach is to split the words with a regular expression
delimiter containing comma or whitespace first. I perform this step to generate tokens.
Every token is checked against the dictionary of stopping words list in order to remove
them. However, there are stopwords exceptions. A classic one is the word not. Not is a

---

<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>

23 Turney, Peter D. "Thumbs up or thumbs down?: semantic orientation applied to unsupervised
classification of reviews." Proceedings of the 40th annual meeting on association for computational

<http://sentiment.christopherpotts.net/lexicons.html>
stopping word as well as a negation word and it can be used to invert the meaning of a sentence. The next step is to use the stringtokenizer to analyze the string of non-stopping words. For each word I checked, using constant time, if the word is present in the dictionary of positive or negative words. I assign a sentiment value of +5 and -5 respectively for a positive and negative word. I classified the sentiments under five classes or rules. A tweet is neutral if it does not contain any positive or negative words in it or if both positive and negative words cancel each other out. For every positive word I increment the polarity of positive word count by 5 and for every negative word I decrement the polarity of negative word count by 5. If the tweet contains more than one positive word then it will be classified as a very positive sentiment otherwise a positive one. On the other hand, if the tweet contains more than one negative word then it will be classified as a very negative sentiment otherwise a negative one. The results of this approach were not satisfactory, Therefore, a classification language model needed to be used where machine learning algorithms can be employed after the text is preprocessed (cleaning and preparation) and every token is analyzed with natural language-based processing model so as to determine the true meaning of the token in a given sentence or document required for sentiment analysis.

**Classification using NLP and supervised machine learning techniques**

**Document Processor**

The goal of a document processor is to separate the documents from the class labels. This is important because I had to write code to read documents and not labels. I

---


separated these two using the AWK programming language tool. The documents were separated from the labels using the tab delimiter ("\t"). I wrote a single line code:

For documents: awk -F "\t" '{print $1} training-dataset > training-dataset-document
For labels: awk -F "\t" '{print $2} training-dataset > training-dataset-label

For training files with positive and negative class labels
Positive documents: awk -F "\t" '{print $1} training-positive-dataset > training-positive-dataset-document
Positive labels: awk -F "\t" '{print $2} training-positive-dataset > training-positive-dataset-label

Negative documents: awk -F "\t" '{print $1} training-negative-dataset > training-negative-dataset-document
Negative labels: awk -F "\t" '{print $2} training-negative-dataset > training-negative-dataset-label

Username and RT Remover
Removing usernames (@hash1302) and special characters from the given tweet/text in document.

Logic: If a tweet contains the "@" symbol, find the index of the next blank space character. Now, I remove the username by using the substring function from the index of the blank space character to end of line. An important function is to trim for any leading or trailing whitespaces.

Special Characters Remover
Logic: If a tweet contains any special or non-printable characters other than a-z, A-Z, 0-9, blank space, !, @, #, $, %, ^, &, *, ( ), -_, =, +, [ ], { }, |, ;, ;, ":", ", comma, ., <, >, ?, /, ` or ~ characters I replace these characters with an empty string. This function can be extended for UTF-8 validation.
Word Normalizer

Normalizing speech exaggerations - more than two letters in word is normalized to an actual word.

Logic: If a tweet contains a word "nooooo" it needs to be normalized to no. The task here is to check if a consonant or a vowel in a word is repeated more than two times. Thus, I check that the character at a position i is the same as the characters in position i + 1 and i + 2 in a given string present in a document. I now needed to normalize the word by removing the exaggerations or exaggerated characters. So I used the Pattern class in Java to determine the next available word boundary. I used non-greedy regular expression matching the repeated character until a word boundary is found and then replaced this matched string with a single instance of the repetitive character; therefore, normalizing the text.

Stopwords Remover

Removing stopping words from a given text in a document so that text processing time is efficient.

Logic: If a tweet contains stopping words they need to be removed. The document is tokenized and every word or token is checked against the list of stopping words. I do this to reduce the number of features in my document matrix and also because stopping words are not relevant in the task of text classification. Furthermore, training with fewer words (with the removal of stopping words) will be faster than training with every token present in the document.

Parts-of-Speech Tagger

Parts-of-Speech tagging for tokens in given text in a document to identify/recognize named entities.

Different parts of speech and examples are provided below.²⁹

b. Noun: Identifies a person or a thing. Example: I love machines.
c. Adjectives: Identifies the quality of a noun. Example: I am a good computer scientist.
d. Determiner: Identifies the limit for a noun. Example: I have a dog named Emma.
e. Pronoun: Replaces a noun. Example: My name is AS. I study computer science.
f. Adverbs: Describes a verb. Example: I am very hungry right now.
g. Preposition: Acts as a bridge between a noun and a word. Example: I go to RIT.
h. Conjunction: Joins clauses. Example: I study computer science and I love my work.
i. Interjection: Exclamations. Example: Oh! How are you?

The goal is to use a parts of speech tagger to identify different parts of speech with labels in a given tweet. I need this for semantic orientation of words or to determine the context of the word. There are various taggers. I experimented with and used two of them - OpenNLP and Stanford NLP POS tagger. The Stanford POS tagger is trained with the help of the Maximum Entropy machine learning algorithm using the Penn TreeBank Project. An instance of the MaxentTagger class is created using the left3words model trained on the Wall Street Journal data. The output of the MaxentTagger will be tokens and its associated parts-of-speech. The form is: Token_1/POS_1, Token_2/POS_2, …, Token_N/POS_N.

Pro/Noun Remover
Remove prepositions, determiners, nouns and pronouns to reduce feature vectors for the classification task.

Logic: I believe some parts-of-speech are not required for the classification task as they will add noise to the machine learners. From the previous steps of identifying parts-of-speech for tokens in a document, I can remove the extraneous ones by splitting

---


the document first using a stringtokenizer and then using the “_” character to separate the word and its parts-of-speech tag. If the POS tag (right part of the “_” delimiter) is a preposition, determiner, noun or pronoun then the token (left part of the “_” delimiter) is removed.

String Tokenizer

An important concept in NLP is tokenizing the string. I needed to be careful while tokenizing string. While I used the StanfordNLP Tokenizer I believe I can do the same using my own algorithm and so I did.

Spell Checker

Spelling Corrector using Peter Norvig’s Spell Check Algorithm

Logic: Given an incorrectly spelled word, the task is to use the Minimum Edit Distance algorithm to correct the word. I find all the combinations of 1 or 2 edits in the token and match the edited word with that in the dictionary.

SynSet Generator

Synonym set generation for given text in a document using Wordnet to arrange like words together.


Logic: The WordNet API provides with the list of words and its synonym sets and antonyms. I use WordNet to rank words in the same synonym set together because if a token in the testing data is not present in the document matrix it can be replaced with the first word in the synonym set. The goal is to replace every word with the first word present in its synonym set. As a result, there will be fewer words and its associated presence and frequency in the feature vector. Thus, this is drastically improve the training time of classifier. However, implementation time of the tool will be less efficient.

Word Sense Disambiguator

Word sense disambiguation of given token in a document using the SentiWordNet tab separated text file.

SentiWordNet (SWN), along with, POS tagging perform an excellent task of semantic orientation of a word. SWN can be used to extract the rank of polarity score of a given token. The input to an SWN is a token and its POS and the output of an SWN is the polarity score of this token. An example word is blue. Blue in its adjective form will have a negative polarity, whereas in its noun form will have a positive polarity. The format of the SWN file is the word_POS, word_ID, word_Positive_Polarity_Score, word_Negative_Polarity_Score, SynonymSet#Rank and the description of the word. Every term in the format is tab separated. From the POS obtained for the word I can then pass the word list as an argument to the SWN program to obtain a polarity score for the word. The SynSet score is word_Positive_Polarity_Score - word_Negative_Polarity_Score.

Word Stemmer

Word stemming using the Porter Stemming algorithm in order to deal with the root of a given word.

I am using the implemented version of the Porter Stemming Algorithm. The algorithm is a 6-step process. The first step is used to remove any plurals present in the word and -ed and -ing. The second step is used to the terminal y to an i when there is another vowel present in the stem of the word. The third step is used to map suffices (double) to suffices (single); therefore, -ization (= -ize plus -ation) maps to -ize. The next or the
fourth step is used to deal with -ic-, -full, -ness etc - which is similar to the previous or the third step. The fifth step is used to remove the -ant, -ence suffices. The sixth step is used to remove any final -e to eventually give a final stemmed word.

Unigrams

Unigrams are single words formed by using a stringtokenizer with whitespace as a delimiter. Unigrams are generally not good for keeping the context of the document in mind or maintaining semantic orientation. Unigrams can be generated with the help of the Penn Bank Tree Tokenizer. The options of this tokenizer are either tokenize by sentence or by tokens. I pass the text file handle to be tokenized as a parameter to the instance of the DocumentPreprocessor class. The text is tokenized by sentences. Also, if I would like the text or document to be tokenized by tokens I pass the file handle to the PTBTokenizer class instance.

An alternative way to generate tokens as unigrams is to use the Stringtokenizer Java class and use whitespace as a delimiter. Furthermore, if a comma or an apostrophe is detected I treat these as individual tokens as these will be essential in the classification task.

Bigrams

Bigrams or pairs of words are better maintaining semantic orientation or context tracking. Bigram (word pairing) generation is required for conditional dependence in the Multinomial Naive Bayes Independence Classifier. For generating bigrams a tokens I use my word pairing algorithm. The algorithm pairs or concatenates two adjacent words and these can provide a better context for a sentence or document.

tf and tf-idf

Unigram (single word) and Bigram (word pairing) term frequency (tf) and term frequency-inverse document frequency (tf-idf) generator. tf is also called the.
tf of a word/token is the number of times a word or a token occurs in the document divided by the total number of words in the document ignoring the case of the word. tf is also called the bag-of-words\textsuperscript{40} model.

$idf$\textsuperscript{41} of a word/token is the logarithm (base 2) of the number of documents present divided by the total number of documents that contain the word or token in question ignoring the case of the word.

Therefore, $tf-idf$\textsuperscript{42} of a word $= tf \ (word) \ * idf \ (word)$

To sum it up:

bag-of-features vs. tf-idf is analogous to word counting vs. word importance.

tf and tf-idf Generator for Unigrams

Term Frequency - Inverse Document Frequency (tf-idf) and bag-of-features can be determined for the generated unigrams. The bag-of-features model specifies the count of or the occurrence of the token, whereas tf-idf specifies the frequency or the importance of a given word token. My document matrix for the bag-of-features approach will contain attributes $token1:word\_Count1 \ ... \ tokenN:word\_CountN$ and for tf-idf approach the attributes will be $token1:word\_tf\_idf1 \ ... \ tokenN:word\_CountN$.

tf and tf-idf Generator for Bigrams

Similar to the previous step, tf-idf and bag-of-features can be determined for the generated bigrams. As mentioned, the bag-of-features model counts the number of times a word or a token occurs where as tf-idf computes the the frequency or the relevance of a given word or token. My document matrix for the bag-of-features


approach will contain attributes token1:bigram_Count1 … tokenN:bigram_CountN and for tf-idf approach the attributes will be token1:bigram_tf-idf1 … tokenN:bigram_tf-idfN.

Feature Vector to Classifier
Text Classification using the generated feature vectors from the unigram and bigram bag-of-features and tf-idf approaches.

I will be using the following two algorithms for classifying my natural language processed text:

Classifiers

Multinomial Naive Bayes Independence Classifier

The Naive Bayes classifier is based on the Bayes theorem which states that the probability of event A given that event B has occurred equals to the probability of event B given that event A has occurred times the probability of event A divided by the probability of event B. For text classification, I determine the class of the text by computing individual probabilities of the tokens given its occurrence or count or frequency in both positive and negative classes. The Naive Bayes class of this tweet is determined by the maximum probability either of the positive or the negative class. The feature vector or features will be tokens such as adjectives, adverbs followed by adjectives. These tokens are obtained by converting the document into features where the attributes are possible words and values are the number of times a word occurs or the frequency of the term in a given document. I find all the documents with positive outcomes and all the documents with negative outcomes and compute the probability of documents with a positive outcome and documents with a negative outcome. I then compute the probability of the tokens given that they occurred in the positive class and all the tokens given that they occurred in the negative class. The probability of the word given that the class is positive or negative is given by: P ( wk | + ) = ( nk + 1 ) / ( n + | Vocabulary | ). When I am classifying or testing a new document I will determine the value or class of it by calculating the probabilities of all the tokens in it given its
occurrence or frequency in positive and negative classes. The maximum probability is the class that this document belongs to.

Support Vector Machine (SVM)

is good at classifying binary sets that are linearly separable. The goal of an SVM is a hyperplane design that helps to classify all the training or feature vectors in the two classes separated by this hyperplane. While designing the hyperplane I have to choose such a value for the margin such that it is maximum to separate the two classes. I decide on two classes - positive and negative such that all $g(x)$ which are $\geq 1$ will be classified in class 1 and all the other $g(x) \leq -1$ will be classified in class 2.
<table>
<thead>
<tr>
<th>Types of Features</th>
<th>bof/</th>
<th>NB (%)</th>
<th>SVM (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-idf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unigrams</td>
<td>bof</td>
<td>74</td>
<td>70</td>
</tr>
<tr>
<td>unigrams</td>
<td>tf-idf</td>
<td>75</td>
<td>71</td>
</tr>
<tr>
<td>unigrams + bigrams</td>
<td>bof</td>
<td>75</td>
<td>73</td>
</tr>
<tr>
<td>unigrams + bigrams</td>
<td>tf-tdf</td>
<td>76</td>
<td>74</td>
</tr>
<tr>
<td>unigrams + pos</td>
<td>bof</td>
<td>75</td>
<td>74</td>
</tr>
<tr>
<td>unigrams + pos</td>
<td>tf-idf</td>
<td>76</td>
<td>75</td>
</tr>
<tr>
<td>bigrams + pos</td>
<td>bof</td>
<td>76</td>
<td>77</td>
</tr>
<tr>
<td>bigrams + pos</td>
<td>tf-idf</td>
<td>76</td>
<td>77</td>
</tr>
<tr>
<td>unigrams + pos + wsd</td>
<td>bof</td>
<td>76</td>
<td>76</td>
</tr>
<tr>
<td>unigrams + pos + wsd</td>
<td>tf-idf</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>bigrams + pos + wsd</td>
<td>bof</td>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>bigrams + pos + wsd</td>
<td>tf-idf</td>
<td>75</td>
<td>76</td>
</tr>
<tr>
<td>adjectives</td>
<td>bof</td>
<td>74</td>
<td>72</td>
</tr>
<tr>
<td>adjectives</td>
<td>tf-idf</td>
<td>75</td>
<td>73</td>
</tr>
</tbody>
</table>
Conclusion

The initial approach involved classifying text into five classes - happy, very happy, neutral, sad, very sad using the Stanford Core NLP suite. Although this technique is effective, it does not take into account the negation of the tweet. “I am not sad” is classified as a negative sentence when it is clearly a positive text. I overcame this by using another technique to classify the tweet into either one of these classes involved simple math - incrementing and decrementing the polarity score of the text based on the lexicon or dictionaries of positive, negative, stopping and negation words. My technique takes into account the negation of the text and handles it correctly. However, it does not take into account context of the words in the language model and semantic orientation of the text. It is clearly not a true natural language-based processing approach towards text classification.

Machine learning techniques are better at text classification using binary classes than lexical/dictionary analysis. Although the NB is simple to implement, its accuracy is good in comparison to SVM. The use of natural language processing is essential for pos extraction required for context evaluation. WordNet API is useful for generating synset terms. SentiWordNet is useful in word sense disambiguation (wsd) for semantic orientation. Bigrams preserve some word ordering. Adjectives alone may be useful but are not very useful because they need to be use along with unigrams or bigrams to generate a high mode of accuracy.