Human Sentiment Analysis On Social Media

Bachelor Thesis By

Shruti Sharma(CSE/16037)
Shikha Sinha (CSE/16035)

Project thesis submitted to

Indian Institute of Information Technology, Kalyani

For the partial fulfillment of the degree of Bachelors of Technology in Computer Science & Engineering

May, 2019

Certificate

This is to certify that the thesis entitled “Human Sentiment Analysis on Social Media” being submitted by Shruti Sharma and Shikha Sinha, undergraduate students (Regis No. 0000192 and 0000190 respectively) in the Department of Computer Science, Indian Institute of Information Technology, Kalyani, India, for the award of Bachelors of Technology in Computer Science, is an original research work carried by him under my supervision and guidance. The thesis has fulfilled all the requirements as par the regulation of IIIT Kalyani and in my opinion, has reached the standards needed for submission. The works, techniques and the results presented have not been submitted to any other university or Institute for the award of any other degree or diploma.

(Dr. Sanjoy Pratihar)
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Information Technology
Kalyani, Webel IT Park, West Bengal ,741235

Declaration

I hereby declare that the work being presented in this project report entitled, “Human Sentiment Analysis - Detecting Sarcasm”, submitted to Indian Institute of Information Technology Kalyani in partial fulfilment for the award of the degree of Bachelor of Technology in Computer Science and Engineering during the period from July, 2018 to November, 2018 under the supervision of Prof. Sanjoy Pratihar, Department of Computer Science and Engineering, Indian Institute of Information Technology Kalyani, West Bengal 741235, India, does not contain any classified information.

Shikha Sinha (CSE/16035/190)
Shruti Sharma(CSE/16037/192)
Department of Computer Science and Engineering,
Indian Institute Of Information Technology,
Kalyani, Webel IT Park, West Bengal, India
Pin code-741235

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

....
(Dr. Sanjoy Pratihar)
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Information Technology
Kalyani, Webel IT Park, West Bengal ,741235

Acknowledgement

Firstly, we would like to thank our supervisor Dr. Sanjoy Pratihar, for his support and guidance to complete this project work without whom we would not be able to make out this far. We would also like to thank our friends who supported us greatly and were always willing to help us. We are very grateful to Department of Computer Science and Engineering, Indian Institute of Information Technology Kalyani, West Bengal 741235, India, for providing us this wonderful opportunity. Lastly, we would like to thank our parents and God for their never ending grace.

Shikha Sinha (CSE/16035/190)
Shruti Sharma(CSE/16037/192)
Department of Computer Science and Engineering,
Indian Institute Of Information Technology,
Kalyani, Webel IT Park, West Bengal, India
Pin code-741235

Abstract

The goal of this project is to show how sentimental analysis can help improve the user experience over a social network. The algorithm will learn what our emotions are from the text and then help us make predictions about our mood. This knowledge will change our social activities accordingly in order to be rational with our feelings. As per our survey, people move towards social media when they are bored, in need of company so after learning from this algorithm , it can suggest appropriate options to them to lighten their mood. It can reduce the impact of those which the mood worse. The project aims to implement these in the social network community for making our lives better and our experience richer and efficient.

Introduction

Sentiment Analysis refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer services to clinical medicine. Sentiment Analysis builds systems that try to identify and extract opinions within text. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:

· *Polarity*: if the speaker express a *positive* or *negative* opinion,

· *Subject*: the thing that is being talked about,

· *Opinion holder*: the person, or entity that expresses the opinion. Currently, sentiment analysis is a topic of great interest and development since it has many practical applications. Since publicly and privately available information over Internet is constantly growing, a large number of texts expressing opinions are available in review sites, forums, blogs, and social media. With the help of sentiment analysis systems, this unstructured information could be automatically transformed into structured data of public opinions about products, services, brands, politics, or any topic that people can express opinions about. This data can be very useful for commercial applications like marketing analysis, public relations, product reviews, net promoter scoring, product feedback, and customer service.

Social Sentiment Analysis is an algorithm that is tuned to analyze the sentiment of social media content, like tweets and status updates. The algorithm takes a string, and returns the sentiment rating for the "positive," "negative," and "neutral." In addition, this algorithm provides a compound result, which is the general overall sentiment of the string.

Sarcasm is a form of speech act in which the speakers convey their message in an implicit way. The inherently ambiguous nature of sarcasm sometimes makes it hard even for humans to decide whether an utterance is sarcastic or not.

Unlike a simple negation, a sarcastic sentence conveys a negative opinion using only positive words or intensified positive words. The detection of sarcasm is therefore important, for the development and refinement of Sentiment Analysis. Sarcasm is "a form of ironic speech commonly used to convey implicit criticism with a particular victim as its target". "Irony" and "sarcasm" are both ways of saying one thing and meaning another but they go about it in different ways.

Related Work

The automatic classification of communicative constructs in short texts has become a widely researched subject in recent years. Large amounts of opinions, status updates and personal expressions are posted on social media platforms such as Twitter. The automatic labeling of their polarity (to what extent a text is positive or negative) can reveal, when aggregated or tracked over time, how the public in general thinks about certain things.

A major obstacle for automatically determining the polarity of a (short) text are constructs in which the literal meaning of the text is not the intended meaning of the sender. Previous works describe the classification of irony (Reyes et al., 2012b), sarcasm (Tsur et al., 2010), satire (Burfoot and Baldwin, 2009), and humor (Reyes et al., 2012a).

Kreuz and Caucci (2007) studied the role that different lexical factors play, such as interjections (e.g., "gee" or "gosh") and punctuation symbols (e.g., '?') in recognizing sarcasm in narratives.

Tsur et al. (2010) focus on product reviews on the World Wide Web, and try to identify sarcastic sentences from these in a semi-supervised fashion. Training data is collected by manually annotating sarcastic sentences, and retrieving additional training data based on the annotated sentences as queries. Sarcasm is annotated on a scale from 1 to 5. Tsur et al. [2010] design pattern-based features that indicate presence of discriminative patterns as extracted from a large sarcasm-labeled corpus. To allow generalized patterns to be spotted by the classifiers, these pattern-based features take real values based on three situations: exact match, partial overlap and no match.

Gonzalez-Ibanez et al. [2011] use sentiment lexicon-based features. In addition, pragmatic features like emoticons and user mentions are also used. They use SVM with SMO and logistic regression.

Reyes et al. (2012b) collect a training corpus of irony based on tweets that consist of the hashtag #irony in order to train classifiers on different types of features and try to distinguish #irony-tweets from tweets containing the hashtags #education, #humour, or #politics. Reyes et al. [2012] introduce features related to ambiguity, unexpectedness, emotional scenario, etc. Ambiguity features cover structural, morpho-syntactic, semantic ambiguity, while unexpectedness features measure semantic relatedness.

Lukin and Walker (2013) explored the potential of a bootstrapping method for sarcasm classification in social dialogue to learn lexical N-gram cues associated with sarcasm (e.g., “oh really”, “I get it”, “no way”, etc.) as well as lexico-syntactic patterns.

Liebrecht et al. (2013) explored N-gram features from 1 to 3-grams to build a classifier to recognize sarcasm in Dutch tweets. They made an interesting observation from their most effective N-gram features that people tend to be more sarcastic towards specific topics such as school, homework, weather, returning from vacation, public transport, the church, the dentist, etc. This observation has some overlap with our observation that stereotypically negative situations often occur in sarcasm.

Maynard and Greenwood [2014] include seven sets of features and study sarcastic tweets and their impact to sarcasm classification. They experiment with around 600 tweets which are marked for subjectivity, sentiment and sarcasm. Some of these are maximum/minimum/gap of intensity of adjectives and adverbs, max/min/average number of synonyms and synsets for words in the target text, etc.

Rajadesingan et al.[2015] use extensions of words,number of flips,readability features in addition to others. Hernandez-Farıasetal.[2015] present features that measure semantic relatedness between words using Wordnet-based similarity.

Abhijit Mishra and Bhattacharyya [2016] conduct additional experiments with human annotators where they record their eye movements. Based on these eye movements, they design a set of gaze based features such as average fixation duration, regression count, skip count, etc. In addition, they also use complex gaze-based features based on saliency graphs which connect words in a sentence with edges representing saccade between the words.

There are two ways for sarcasm detection and the most used way is the Machine learning based approach.The other is Lexicon based approach.

Methodology

This project of analyzing sentiments of tweets comes under the domain of "Pattern Classification" and "Data Mining". To implement sentiment analysis in our project we took a step forward by not classifying the tweets simply positive or negative but also associating different emotions to it like happy, anger, anticipation, surprise etc. Everytime the algorithm runs, we make the collected data appropriate for our use by removing those data-fields which do not serve our purpose. The data provided comes with emoticons, usernames, URL's, references and hashtags which are required to be processed and converted into a standard form. The words are also a mixture of misspelled words/incorrect, extra punctuations, and words with many repeated letters. Therefore, tweets must be preprocessed to standardize the dataset. This completes text mining. We then extract the replies to a particular tweet from our corpus and use the sentiment of the replies to deduct the sentiment of that tweet. It is mainly a content-based classification problem used in Natural Language Processing and Machine Learning.

Tools used :

1. R Sentiment Analysis tools
2. Twitter Timeline
3. Python
4. NLTK library
5. WEKA

We collected data by using by making Twitter application for accessing Twitter API and getting the required OAuth permissions. We will use this data as the training data. R-Studio is the environment developed for statistical analysis and a Graphical view of the large data sets. Using this data and the replies generated to it, we can determine the polarity of the text which further determines the polarity assigned to the person that data belongs to.

Collectively, there are over 200 million active users of Twitter and since only 11% have protected accounts, the vast majority of the resulting tweets are in the public domain.

The machine learning approach applicable to sentiment analysis mostly belongs to supervised classification in general and text classification techniques in particular. Thus, it is called "Supervised learning". In a machine learning based classification, two sets of documents are required: training and a test set. A training set is used by an automatic classifier to learn the differentiating characteristics of documents, and a test set is used to validate the performance of the automatic classifier.

Dataset

The data we will gather to do the project with will likely be live streaming tweets that we gather for a period of a few days or weeks until we have a sufficiently large amount to train our system with. To narrow the scope, using twitter data initially, we will select #sarcasm, #sarcastic, etc as well as other hashtags that allow us to tailor our system to a specific niche area to focus upon. We defined annotation guidelines that instructed human annotators to read isolated tweets and label a tweet as *sarcastic* if it contains comments judged to be sarcastic based solely on the content of that tweet. We collected over 75,000 tweets and labeled them 0 for non-sarcastic and 1 for sarcastic.
Each tuple in the dataset is of the form : (TweetNo, 0/1, Tweet)

We may write each record as a vector, x= (x_1, x_2, ..., x_n) , which we refer to as an instance. Each component x_i is known as a feature.

Features

- ❏ Use of laughter expression
- ❏ Heavy Punctuation
- ❏ Use of emoticons
- ❏ Polarity Flip : This feature tests the incongruity of tweet which might be expressed through use of sentiment words of both polarities or through phrases of implied sentiment.

- ❑ N-grams : An N-gram is an N length sequence of items, which in natural language processing tasks are usually tokens like letters or words. By pairing each N-gram with a frequency with which it appears in some document, a frequency distribution can be computed.
- ❑ Skip-Grams : A variation of Word2Vec. First, a 'history' parameter (*skip-gram value)* is chosen that represents how large the context should be. Word2Vec is a technique used to convert text into a more machine learning friendly form-- a feature vector. It takes context into account, using a maximum likelihood function to calculate the probability of a word occurring given the words surrounding it.
- ❑ Sentiment feature
- ❑ Number of repetitive sequence of characters : Looking at words with a large number of syllables has shown to be a possible method of sarcasm detection.
- ❑ Number of capitalized word : We will attempt to find patterns in capitalization and punctuation that can help us determine sarcasm classifications.
- ❑ Hashtags
- ❑ POS : Previous research has suggested that the parts of speech and the frequency, density, and patterns of those parts may be another useful tool. Using N-gram and pattern matching analysis on the POS tags of a message we hope to be able to extract useful classification information.

Feature Selection

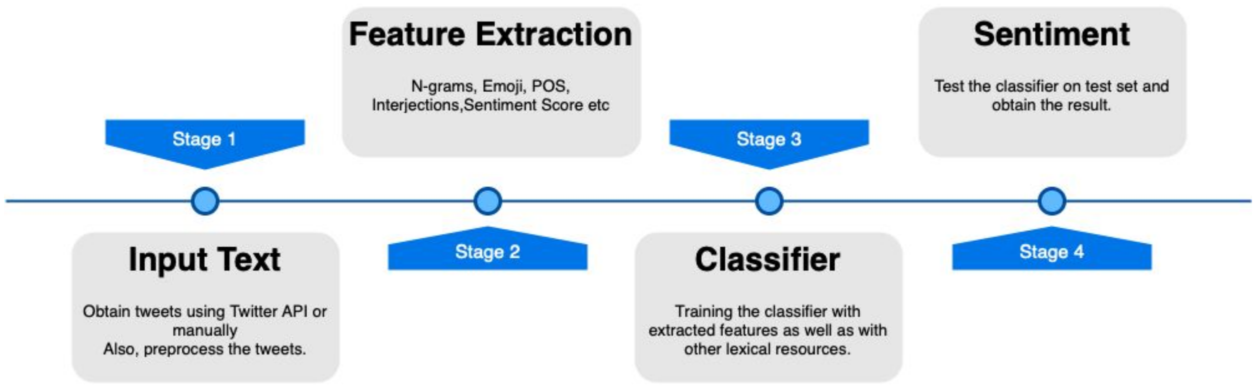
There are three main types of features for training the classifier:
•Lexical features - The lexical features are obtained from the unigram, bigram and trigram.
•Hyperbole - The hyperbole features are presence of the intensified positive words(adjectives), interjections, quotes, punctuation marks.
•Pragmatic features - The pragmatic features the presence of emoticons like frowning smileys, smiling faces etc and the mentions in the comments or the replies in case of twitter retweets.

Sentiment based features can be extracted from number of positive/negative words. Positive/Negative word is said to be highly emotional if it's POS tag is one amongst : 'JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'.
Emotion features are extracted from emoticons.
Punctuation features from number of exclamations, question-mark, ellipsis, capital letter words.

Main Methodology

1. Know and Research about the objective which is human sentiment analysis and humour detection.
2. Collect Related Data - We are collecting features/column and tokenizing them whereby subsets of a text (paragraphs, sentences, words, punctuation) are identified.
3. Integrate Data – Form a dataset in csv/excel or any other format
4. Data Exploration- Load Data, Remove Duplicates etc.
5. Data Preprocessing-This includes dealing with missing values, categorical data, adding and removing columns.
6. Data Modelling-Make a good data model by applying appropriate ML algorithms.
7. Data Validation-Test your model by using different validation method like k-fold cross validation.
8. Model Optimization-Build an optimized model by checking it with different algorithm and their accuracy.
9. Data Analysis and Visualization-Analyze and then explain your findings with proper charts and skills.

Proposed Architecture


Stage 1
Input Text
Obtain tweets using Twitter API or manually
Also, preprocess the tweets.
Feature Extraction
N-grams, Emoji, POS, Interjections,Sentiment Score etc
Stage 2
Stage 3
Classifier
Training the classifier with extracted features as well as with other lexical resources.
Sentiment
Test the classifier on test set and obtain the result.
Stage 4

Classifiers:

1.Naive Bayes

A Naive Bayes classifier is a probabilistic machine learning model that’s used classification task. The crux of the classifier is based on the Bayes theorem.
Bayes theorem: P(A/B)=(P(B/A)P(A))/P(B)

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

2.Decision Tree

The classification technique is a systematic approach to build classification models from an input dat set. For example, decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and naive Bayes classifiers are different technique to solve a classification problem. Each technique adopts a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data. Therefore, a key objective of the learning algorithm is to build predictive model that accurately predict the class labels of previously unknown records.

Decision Tree Classifier is a simple and widely used classification technique. It applies a straightforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test record. Each time time it receive an answer, a follow-up question is asked until a conclusion about the class label of the record is reached.

3.Logistic Regression

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.
Sometimes logistic regressions are difficult to interpret; the Intellectus Statistics tool easily allows you to conduct the analysis, then in plain English interprets the output.

4.Random Forest

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty

good indicator of the feature importance.
Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

5.Support Vector Machines

A SVM is a discriminative classifier formally defined by a separating hyperplane.In other words,given labelled training data,the algorithm outputs an optimal hyperplane which categorizes new examples.In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Proposed Algorithm:

1) Importing libraries and dataset(imported by API connection of Twitter)
2) Reading CSV file
3) Clean data by removing stopwords, lemmatize and tokenize
4) Feature extraction by

- Counting the user mentions in a tweet
- Counting the punctuations in a tweet
- Calculating the polarity of the hashtag
- Calculating the emoji sentiment from the emojis present in the tweet
- Counting the interjections in the tweet
- Counting the capital words in the tweet
- Counting the repeated letter in a word (ex. "Whaaat")
- Calculating Sentiment score of the tweet
- Finding the polarity flip in a tweet i.e positive to negative or negative to positive change
- Finding the number of Nouns and Verbs in a tweet
- Counting the intensifiers in a tweet
- Finding the most common unigrams
- Finding the most common bigrams and skipgrams(skip 1/2 grams in a tweet)
- Finding the total number of passive aggressive statements in a tweet

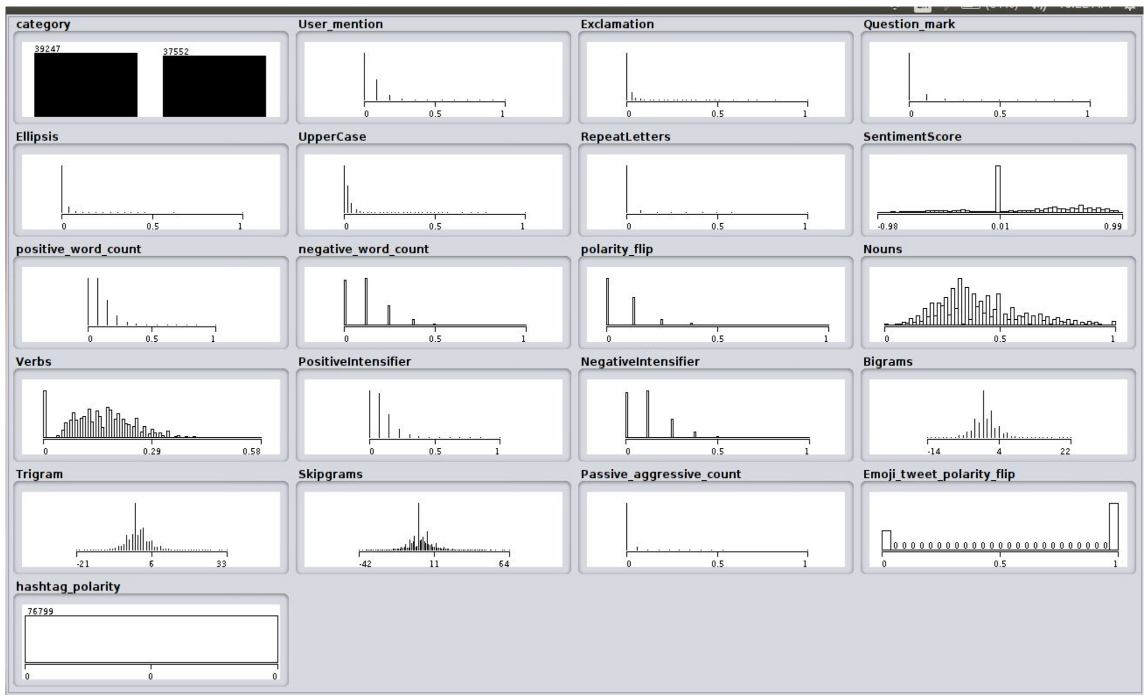
5) Making a feature list
6) Calculating the feature score by normalizing all the features in the feature list
7) Checking accuracy of the sarcasm detection on the basis of feature list using various classifiers
   -SVM Model
   -Decision Tree

- Naïve Bayes
-Logistic Regression
-Random forest

Result

The tweets are saved in a .csv file after running the code. It then gives following results. The tweets are classified as positive/negative sentiment and further into emotions like anger, disgust, anticipation,trust, joy, surprise etc. The accuracy of result may vary depending on the bias of dataset. A dataset having more of sarcastic tweets will show more sarcastic result.
Most work in sarcasm detection relies on SVM. However we have obtained the best results on Random Forest with an accuracy of 94.5%

Visualization on different features (after normalization) :



Classification result using different classifiers

1) Random Forest

Correctly Classified Instances	94.5351%

Incorrectly Classified Instances	5.4649%

2) Decision Tree

Correctly Classified Instances	91.9179%
Incorrectly Classified Instances	8.0821%

3) SVM

Correctly Classified Instances	92.1861%
Incorrectly Classified Instances	7.8139%

4) Naive Bayes

Correctly Classified Instances	81.0727%
Incorrectly Classified Instances	18.9273%

5) Logistic Regression

Correctly Classified Instances	91.552%
Incorrectly Classified Instances	8.448%

Conclusion

Microblogging nowadays became one of the major types of the communication. A recent research has identified it as online word-of-mouth branding (Jansen et al., 2009). The large amount of information contained in microblogging web-sites makes them an attractive source of data for opinion mining and sentiment analysis. In our research, we have presented a method for an automatic collection of a corpus that can be used to train a sentiment classifier. We used sentiment analysis tools and observed the difference in distributions among positive, negative and neutral sets. From the observations we conclude that authors use syntactic structures to describe emotions or state facts. Some POS-tags may be strong indicators of emotional text. We used the collected corpus to train a sentiment classifier. Our classifier is able to determine positive, negative and neutral sentiments of documents. The classifier is based on the multinomial Naıve Bayes classifier that uses N-gram and POS-tags as features.

Statistical approaches use features like sentiment changes. To incorporate context, additional features specific to the author, the conversation and the topic have been explored in the past.

Future Work

The project on sentiment analysis can be extended further to deeper levels. We can come up with a methodology to improve user experience like bringing in Unfriend option on Facebook for people which have negative impact on the user's frame of mind. The project is mainly designed for Twitter. We can have a similar application for Facebook wherein the intended features could be designed and incorporated. Another future task is to have a hate-email classifier in addition to spam.

Sarcasm detection research has grown significantly in the past few years, necessitating a look-back at the overall picture that these individual works have led to. Future work may focus on other forms of sarcasm. Sarcasm is closely related to language/culture-specific traits. Future approaches to sarcasm detection in new languages will benefit from understanding such traits, and incorporating them into their classification frameworks.

Very few approaches have explored deep learning-based architectures so far. Future work that uses these architecture may show promise.

References

- https://begriffs.com/posts/2015-02-25-text-mining-in-r.html
- https://knightlab.northwestern.edu/2014/03/15/a-beginners-guide-to-collecting-twitter-https://joparga3.github.io/Udemy_text_analysis/#example-selecting-all-text-from-page---not-selecting-contents-images-etc
- Twitter Sentiment Analysis System - Shaunak Joshi and Deepali Deshpande , International Journal of Computer Applications (0975 – 8887) Volume 180 – No.47, June 2018
- P. Basile, V. Basile, M. Nissim, N. Novielli, V. Patti."Sentiment Analysis of

Microblogging Data”. To appear in Encyclopedia of Social Network Analysis and Mining, Springer. In press.data-and-a-bit-of-web-scraping/

- Automatic Sarcasm Detection: A Survey - Aditya Joshi, Pushpak Bhattacharyya, Mark J Carman

Annexure(Implementation in Python)

Feature_extraction.py

```
# Import Packages

import csv
import os
import re
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import constants
import numpy as np
import pandas as pd
from ekphrasis.classes.segmenter import Segmenter

# Initialize variables
sid = SentimentIntensityAnalyzer()
ps = PorterStemmer()
lemm = WordNetLemmatizer()

FEATURE_LIST_CSV_FILE_PATH = os.curdir + "\\data\\feature_list.csv"
DATASET_FILE_PATH = os.curdir + "\\data\\dataset.csv"
stopwords = stopwords.words('english')

# Read Data in Dataframe
def read_data(filename):
        data = pd.read_csv(filename, header=None, encoding="utf-8", names=["Index", "Label",
"Tweet"])
    return data

# Remove stopwords, lemmatize and tokenize
```

```
def clean_data(tweet,lemmatize=True, remove_punctuations=True, remove_stop_words=False):
    stopwords = nltk.corpus.stopwords.words('english')
    lemm = nltk.stem.wordnet.WordNetLemmatizer()
    tokens = nltk.word_tokenize(tweet)
    if remove_punctuations:
        tokens = [word for word in tokens if word not in string.punctuation]
    if remove_stop_words:
        tokens = [word for word in tokens if word.lower() not in stopwords]
    if lemmatize:
        tokens = [lemm.lemmatize(word) for word in tokens]
    return tokens

# Counts the user mentions in a tweet
def user_mentions(tweet):
    return len(re.findall("@([a-zA-Z0-9]{1,15})", tweet))

# Counts the punctuations in a tweet
def punctuations_counter(tweet, punctuation_list):
    punctuation_count = {}
    for p in punctuation_list:
        punctuation_count.update({p: tweet.count(p)})
    return punctuation_count


# Calculates the polarity of the hashtag
def hashtag_sentiment(tweet):
    hash_tag = (re.findall("#([a-zA-Z0-9]{1,25})", tweet))

    hashtag_polarity = []
    for hashtag in hash_tag:
        tokens = (seg.segment(hashtag))
        ss = sid.polarity_scores(tokens)
        if 'not' not in tokens.split(' '):
            hashtag_polarity.append(ss['compound'])
        else:
            hashtag_polarity.append(- ss['compound'])
    sentiment = 0
    if len(hashtag_polarity) > 0:
        sentiment = round(float(sum(hashtag_polarity) / float(len(hashtag_polarity))), 2)
    return sentiment

# Counts the capital words in the tweet
```

```
def captitalWords_counter(tokens):
    upperCase = 0
    for words in tokens:
        if words.isupper() and words not in constants.exclude:
            upperCase += 1
    return upperCase

# Counts the repeated letter in a word (ex. "Whaaat")
def repeatLetterWords_counter(tweet):
    repeat_letter_words = 0
    matcher = re.compile(r'(.)\1*')
    repeat_letters = [match.group() for match in matcher.finditer(tweet)]
    for segments in repeat_letters:
        if len(segments) >= 3 and str(segments).isalpha():
            repeat_letter_words += 1
    return repeat_letter_words

# Sentiment score of the tweet

def getSentimentScore(tweet):
    return round(sid.polarity_scores(tweet)['compound'], 2)

# Finds the polarity flip in a tweet i.e positive to negative or negative to positive change
def polarityFlip_counter(tokens):
    positive = False
    negative = False
    positive_word_count, negative_word_count, flip_count = 0, 0, 0
    for words in tokens:
        ss = sid.polarity_scores(words)
        if ss["neg"] == 1.0:
            negative = True
            negative_word_count += 1
            if positive:
                flip_count += 1
                positive = False
        elif ss["pos"] == 1.0:
            positive = True
            positive_word_count += 1
            if negative:
                flip_count += 1
                negative = False
    return positive_word_count, negative_word_count, flip_count
```

```
# Finds the number of Nouns and Verbs in a tweet
def POS_count(tokens):
    # tokens = clean_data(tweet, lemmatize= False)
    Tagged = nltk.pos_tag(tokens)
    nouns = ['NN', 'NNS', 'NNP', 'NNPS']
    verbs = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    noun_count, verb_count = 0, 0
    no_words = len(tokens)
    for i in range(0, len(Tagged)):
        if Tagged[i][1] in nouns:
            noun_count += 1
        if Tagged[i][1] in verbs:
            verb_count += 1
        return round(float(noun_count) / float(no_words),2), round(float(verb_count) /
float(no_words),2)

# Counts the intensifiers in a tweet
def intensifier_counter(tokens):
    # tweet = clean_data(tweet, lemmatize= False)
    posC, negC = 0, 0
    for index in range(len(tokens)):
        if tokens[index] in constants.intensifier_list:
            if (index < len(tokens) - 1):
                ss_in = sid.polarity_scores(tokens[index + 1])
                if (ss_in["neg"] == 1.0):
                    negC += 1
                if (ss_in["pos"] == 1.0):
                    posC += 1
    return posC, negC

# Finds the most common bigrams and skipgrams(skip 1/2 grams in a tweet )
def skip_grams(tokens, n, k):
    skip_gram_value = 0
    # tokens = clean_data(tweet, lemmatize= False)
    a = [x for x in nltk.skipgrams(tokens, n, k)]
    for j in range(len(a)):
        for k in range(n):
            ss = sid.polarity_scores(a[j][k])
            if (ss["pos"] == 1):
                skip_gram_value += 1
            if (ss["neg"] == 1):
```

```
            skip_gram_value -= 1
    return skip_gram_value

# Finds the most common unigrams
def find_common_unigrams(data_set):
    unigram_sarcastic_dict = {}
    unigram_non_sarcastic_dict = {}
    sarcastic_unigram_list = []
    non_sarcastic_unigram_list = []
    tweets = data_set['Tweet'].values
    labels = data_set['Label'].values
    for i, tweet in enumerate(tweets):
            tokens    =    clean_data(tweet,    lemmatize=True,    remove_punctuations=True,
remove_stop_words=True)
        for words in tokens:
            if words in unigram_sarcastic_dict.keys() and int(labels[i]) == 1:
                unigram_sarcastic_dict[words] += 1
            else:
                unigram_sarcastic_dict.update({words: 1})
            if words in unigram_non_sarcastic_dict.keys() and int(labels[i]) == 0:
                unigram_non_sarcastic_dict[words] += 1
            else:
                unigram_non_sarcastic_dict.update({words: 1})

    # Creat list of high frequency unigrams
    # change value > 'x' where x is the frequency threshold

    for key, value in unigram_sarcastic_dict.items():
        if value > 1000 and key not in stopwords:
            sarcastic_unigram_list.append(key)
    for key, value in unigram_non_sarcastic_dict.items():
        if value > 1000 and key not in stopwords:
            non_sarcastic_unigram_list.append(key)
    return sarcastic_unigram_list, non_sarcastic_unigram_list

# Finds the total number of passive aggressive statements in a tweet

def passive_aggressive_counter(tweet):
   sentence_count = 0
   new_sentence = []
   i = 0
   for j in range(len(tweet)):
        if '.' != tweet[j]:
            i += 1
            new_sentence.append(tweet[j])
        else:
            makeitastring = ''.join(map(str, new_sentence))
            tokens = (nltk.word_tokenize(makeitastring))
            if len(tokens)<3 and len(tokens) >0:
                sentence_count += 1
            if i == len(tweet):
                return 0
            new_sentence = []
    return sentence_count

def unigrams_counter(tokens, common_unigrams):
    common_unigrams_count = {}
    for word in tokens:
        if word in common_unigrams:
            if word in common_unigrams_count.keys():
                common_unigrams_count[word] += 1
            else:
                common_unigrams_count.update({word: 1})
    return common_unigrams_count

# Normalize every feature
def normalize( array):
    max = np.max(array)
    min = np.min(array)
    def normalize(x):
        return round(((x-min) / (max-min)),2)
    if max != 0:
        array = [x for x in map(normalize, array)]
    return array


def main():
    # Read data and initialize feature lists
    data_set = read_data(DATASET_FILE_PATH)
    label = list(data_set['Label'].values)
    tweets = list(data_set['Tweet'].values)
    user_mention_count = []
    exclamation_count = []
    questionmark_count = []
```

```
ellipsis_count = []
emoji_sentiment = []
interjection_count = []
uppercase_count = []
repeatLetter_counts = []
sentimentscore = []
positive_word_count = []
negative_word_count = []
polarityFlip_count = []
noun_count = []
verb_count = []
positive_intensifier_count = []
negative_intensifier_count = []
skip_bigrams_sentiment = []
skip_trigrams_sentiment = []
skip_grams_sentiment = []
unigrams_count = []
passive_aggressive_count = []
emoji_tweet_flip = []
hashtag_sentiment_score = []
emoji_count_list = []


COMMON_UNIGRAMS = find_common_unigrams(data_set)
print(COMMON_UNIGRAMS)

# For every tweet, extract all the features and append to corresponding list
for t in tweets:
    hashtag_sentiment_score.append(0)
    tokens = clean_data(t)
    user_mention_count.append(user_mentions(t))
    p = punctuations_counter(t, ['!', '?', '...'])
    exclamation_count.append(p['!'])
    questionmark_count.append(p['?'])
    ellipsis_count.append(p['...'])
    interjection_count.append(interjections_counter(t))
    uppercase_count.append(captitalWords_counter(tokens))
    repeatLetter_counts.append(repeatLetterWords_counter(t))
    sentimentscore.append(getSentimentScore(t))
    x = polarityFlip_counter(tokens)
    positive_word_count.append(x[0])
    negative_word_count.append(x[1])
```

```
        polarityFlip_count.append(x[-1])
        x = POS_count(tokens)
        noun_count.append(x[0])
        verb_count.append(x[1])
        x = intensifier_counter(tokens)
        positive_intensifier_count.append(x[0])
        negative_intensifier_count.append(x[1])
        x = getEmojiSentiment(t)
        emoji_count_list.append(x[1])
        emoji_sentiment.append(x[0])
        skip_bigrams_sentiment.append(skip_grams(tokens, 2, 0))
        skip_trigrams_sentiment.append(skip_grams(tokens, 3, 0))
        skip_grams_sentiment.append(skip_grams(tokens, 2, 2))
        unigrams_count.append(unigrams_counter(tokens, COMMON_UNIGRAMS))
        passive_aggressive_count.append(passive_aggressive_counter(t))
        if (sentimentscore[-1] < 0 and emoji_sentiment[-1] > 0) or (sentimentscore[-1] > 0 and
emoji_sentiment[-1] < 0):
            emoji_tweet_flip.append(1)
        else:
            emoji_tweet_flip.append(0)

    # Creates a list of list of features
    feature_label = zip(label, normalize(user_mention_count), normalize(exclamation_count),
normalize(questionmark_count),normalize(ellipsis_count),normalize(uppercase_count),normalize(
repeatLetter_counts),sentimentscore,normalize(positive_word_count),normalize(negative_word_c
ount),normalize(polarityFlip_count),noun_count,verb_count,normalize(positive_intensifier_count)
,normalize(negative_intensifier_count),skip_bigrams_sentiment,skip_trigrams_sentiment,skip_gra
ms_sentiment, normalize(passive_aggressive_count), emoji_tweet_flip,
 hashtag_sentiment_score)

    # Headers for the new feature list
    headers = ["label", "User mention", "Exclamation", "Question mark", "Ellipsis",
"UpperCase","RepeatLetters", "SentimentScore", "positive word count", "negative word count",
"polarity flip","Nouns", "Verbs", "PositiveIntensifier", "NegativeIntensifier", "Bigrams",
"Trigram", "Skipgrams","Passive aggressive count", "Emoji_tweet_polarity flip",
"hashtag_polarity"]

    # Writing headers to the new .csv file
    with open(FEATURE_LIST_CSV_FILE_PATH, "w") as header:
        header = csv.writer(header)
        header.writerow(headers)
    # Append the feature list to the file
    with open(FEATURE_LIST_CSV_FILE_PATH, "a") as feature_csv:
        writer = csv.writer(feature_csv)
        for line in feature_label:
            writer.writerow(line)

if __name__ == "__main__":
    main()
```