A

PROJECT REPORT ON

**“Social Media Sentiment Analysis”**

SUBMITTED TO

University of Mumbai, Mumbai

In

Partial Fulfilment of

M.Sc. in Computer Science

BY

MR. RAMAN KUMAR

Through

University Department of Computer Science

University of Mumbai, Mumbai

**Year 2018-2019**



**University Department of Computer Science**

University of Mumbai

Ranade Bhavan, B-Wing, Ground Floor,

Vidyanagari Campus, Kalina,

Santacruz (East), Mumbai-400098.

**PROJECT IMPLEMENTATION REPORT**

A

PROJECT REPORT ON

**“Social Media Sentiment Analysis”**

SUBMITTED TO

University of Mumbai, Mumbai

In

Partial Fulfilment of

M.Sc. in Computer Science

BY

RAMAN KUMAR

Through

University Department of Computer Science

University of Mumbai, Mumbai

**Year 2018-2019**

****

**University Department of Computer Science**

University of Mumbai

Ranade Bhavan,B-Wing, Ground Floor,

Vidyanagari Campus, Kalina,

Santacruz (East), Mumbai-400098.

Certificate

This is to certify that, Mr. **Raman Kumar** **of Master of Science in Computer Science Semester-IV** Class Bearing Examination Seat No. **40538,** has satisfactorily carried out the Project on “**Social Media Sentiment Analysis**” as laid down by University of Mumbai, during the academic year 2018-2019.

Examiners

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Guide **Dr.Ambuja Salgaonkar**

Head, Department of Computer Science

**Acknowledgement**

We take this opportunity of submitting this Research Implementation to express my profound gratitude to the “University Department of Computer Science” for giving me the

Opportunity to accomplish this project work. We are very much thankful to respected HOD-Dr. Ambuja Salgaonkar.

We are also grateful to our respected Project Guide for being resourceful, helpful and also for their constant support, encouragement and Guidance, without which the successful completion of this project proposal would have been impossible.

“THANK YOU”

Mr. Raman Kumar

**Contents**

Abstract………………………………………………………………………………3

About Project…………………………………………………………………….3

Keywords………………………………………………………………………4

Design and Implementation…..………………………………………………………………5

Data Gathering…………………………………………………………………...5

Pre-processing ………………………………………………….5

Building the Wordlist………………………………………………………………..8 Classification………………………………………….8

Experimental setup and Results………………………………………………………….10

Software **/** Package Used…………………………………………………………11

Result……………………………………………………………………………...15

Analysis of the result……………………………………………………………………...17

Conculsion………………………………………………………………………………...18

Future Enhancement………………………………………………………………………19

Program Code……………………………………………………………………………..20

References…………………………………………………………………………………30

**Abstract**

The goal of this project was to predict sentiment for the given Twitter post using Python. Sentiment analysis can predict many different emotions attached to the text, but in this report only 3 major were considered: positive, negative and neutral. The training dataset was small (just over 5900 examples) and the data within it was highly skewed, which greatly impacted on the difficulty of building good classifier. After creating a lot of custom features, utilizing both bag-of-words and word2vec representations and applying the Extreme Gradient Boosting algorithm, the classification accuracy at level of 58% was achieved.

Sentiment analysis, the automated extraction of expressions of positive or negative attitudes from text has received considerable attention from researchers during the past decade. In addition, the popularity of internet users has been growing fast parallel to emerging technologies; that actively use online review sites, social networks and personal blogs to express their opinions. They harbour positive and negative attitudes about people,

Organizations, places, events, and ideas. The tools provided by natural language processing and machine learning along with other approaches to work with large volumes of text, makes it possible to begin extracting sentiments from social media. In this paper we discuss some of the challenges in sentiment extraction, some of the approaches that have been taken to address these challenges and our approach that analyses sentiments from Twitter social media which gives the output beyond just the polarity but use those polarities in product profiling, trend analysis and forecasting. Promising results has shown that the

Approach can be further developed to cater business environment needs through sentiment analysis in social media.

**Keywords** – Sentiment Analysis, Natural Language Processing, Data Mining, Supervised Learning

**DESIGN AND IMPLEMENTATION**

**The steps in implementation are:**

* Load all Python Library
* Loading the Data
* Data Distribution
* Pre-processing Steps

1. Cleaning
2. Remove URLs
3. Remove usernames (mentions)
4. Remove tweets with \*Not Available\* text
5. Remove special characters
6. Remove numbers
7. Text processing
8. Tokenize
9. Stem

* Building the Wordlist
* Bag of words

**Classification**

* Naïve Bayes Classification
* Random Forest Classification

**Twitter DATA**

Input data consisted two CSV files: **train.csv** (5971 tweets)

**test.csv (**4000 tweets)

**Chapter 1**

**Annexure**

**1.1 Abreviation**

|  |  |
| --- | --- |
| **Symbol** | **Full form** |
| nltk | Natural Language Toolkit |
| TL | Time Line |
| **#** | Hashtag |
| RT | Retweet |
| OH | overhead |

**Fig**

|  |  |  |
| --- | --- | --- |
| SERIAL NO | FIG NO | PAGE NO |
| 1 | Figure 1 | 18 |
| 2 | Figure 2 | 19 |
| 3 | Figure 3 | 20 |
| 4 | Figure 4 | 22 |
| 5 | Figure 5 | 23 |

**Table**

|  |  |  |
| --- | --- | --- |
| Serial No | Table No | Page no |
| 1 | Table 1 | 18 |
| 2 | Table 2 | 19 |
| 3 | Table 3 | 20 |
| 4 | Table 4 | 21 |

\

**Chapter 2**

**2.1Objectives**

Sentiment analysis – otherwise known as opinion mining – is a much bandied about but often misunderstood term. In essence, it is the process of determining the emotional tone behind a series of words, used to gain an understanding of the attitudes, opinions and emotions expressed within an online mention

All in all, sentiment analysis boils down to one thing: It's the process of analyzing online pieces of writing to determine the emotional tone they carry. In simple words, sentiment analysis is used to find the author's attitude towards something.

**2.2 Scope:**

Sentiment Analysis. Sentiment analysis is the **automated** process of understanding an opinion about a given subject from written or spoken language. In a world where we generate 2.5 quintillion bytes of data every day, sentiment analysis has become a key tool for making sense of that data.

**Chapter 3**

**Theoretical Background**

Sentiment Analysis is a technique widely used in text mining. Social Media Sentiment Analysis, therefore means, using advanced text mining techniques to analyze the sentiment of the text (here, tweet) in the form of positive, negative and neutral. Social Media Sentiment Analysis, also known as Opinion Mining, is primarily for analyzing conversations, opinions, and sharing of views (all in the form of tweets) for deciding business strategy, political analysis, and also for assessing public actions.

**Application domain description**

* **Business:** In marketing field companies use it to develop their strategies, to understand customers’ feelings towards products or brand, how people respond to their campaigns or product launches and why consumers don’t buy Product
* **Politics:** In political field, it is used to keep track of political view, to detect consistency and inconsistency between statements and actions at the government level. It can be used to predict election results as well!
* **Public Actions:** Sentiment analysis also is used to monitor and analyse social phenomena, for the spotting of potentially dangerous situations and determining the general mood of the blogosphere.

**Chapter 4**

**Definition of Problem**

**Literature survey**

Sentiment Analysis can be used to observe the attitude of any Statement made by people responding or reacting to it, Today Sentiment Analysis has reached to the level where it can determine not only the positivity or negativity of a statement but also deal with the different topics and behaviours of these statements. There has been a lot of work recently in the genre of “Sentiment analysis “by multiple researchers. Actually the Evolution of this field started by early 2000’s. In the beginning it was only able to perform binary classifications that is assigning positive or negative tags to the comments. There are researches on sentiment analysis that are based on opinions of the users version of summarization system of the product.[1] There has been a lot of effort been put in this field where programmers have applied soft programming approaches, That is usually fuzzy logic and neural networks for sentiment analysis. There are algorithms construct fuzzy domain sentiment ontology tree based on the reviews that includes the extraction of sentimental words or sentences, distinct features of the products and relation amongst features thus precisely predicting the polarity of the reviews in the networking site. By designing membership functions for the process they formulated and standardized the elite process of evaluating the strength of reviewer’s opinions in the presence of an adverbial modifier on the social networks.

We use the Following Machine Learning Algorithms

**Naive Bayes**

            We used Multinomial NB from sklearn.naive\_bayes package of scikit-learn for Naive Bayes classification. We used Laplace smoothed version of Naive Bayes with the smoothing parameter α set to its default value of 1. We used sparse vector representation for classification and ran experiments using both presence and frequency feature types. We found that presence features outperform frequency features because Naive Bayes is essentially built to work better on integer features rather than floats. We also observed that addition of bigram features improves the accuracy. We obtain a best validation accuracy of 60.12% using Naive Bayes with presence of unigrams and bigrams.

**Random Forest**

            We implemented random forest algorithm by using RandomForestClassifier from sklearn.ensemble provided by scikit-learn. We experimented using 10 estimators (trees) using both presence and frequency features. Presence features performed better than frequency though the improvement was not substantial.

**XGBoost**

 Xgboost is a form of gradient boosting algorithm which produces a prediction model that is an ensemble of weak prediction decision trees.

**Chapter 5**

**System Analysis & Design**

**Sentiment analysis**

In the past few years, there has been a huge growth in the use of microblogging platforms such as Twitter. Spurred by that growth, companies and media organizations are increasingly seeking ways to mine Twitter for information about what people think and feel about their products and services.

While there has been a fair amount of research on how sentiments are expressed in genres such as online reviews and news articles, how sentiments are expressed given the informal language and message-length constraints of microblogging has been much less studied. Features such as automatic part-of-speech tags and resources such as sentiment lexicons have proved useful for sentiment analysis in other domains, but will they also prove useful for sentiment analysis in Twitter? In this project, we begin to investigate this question.

Another challenge of microblogging is the incredible breadth of topic that is covered. It is not an exaggeration to say that people tweet about anything and everything. Therefore, to be able to build systems to mine Twitter sentiment about any given topic, we need a method for quickly identifying data that can be used for training.

Sentiment analysis refers to the broad area of natural language processing which deals with the computational study of opinions, sentiments and emotions expressed in text. Sentiment Analysis (SA) or Opinion Mining (OM) aims at learning people’s opinions, attitudes and emotions towards an entity.

The online medium has become a significant way for people to express their opinions and with social media, there is an abundance of opinion information available. Using sentiment analysis, the polarity of opinions can be found, such as positive, negative, or neutral by analysing the text of the opinion. Sentiment analysis has been useful for companies to get their customer's opinions on their products predicting outcomes of elections, and getting opinions from movie reviews. The information gained from sentiment analysis is useful for companies making future decisions.

**Chapter 6**

**Details of Hardware & Software Used**

Computer Processor: - Intel core i3 (minimum)

Clock Speed: - 1.2MHz Processor

Hard Disk: - 300GB

RAM: - 4GB

**Software Requirement: -**

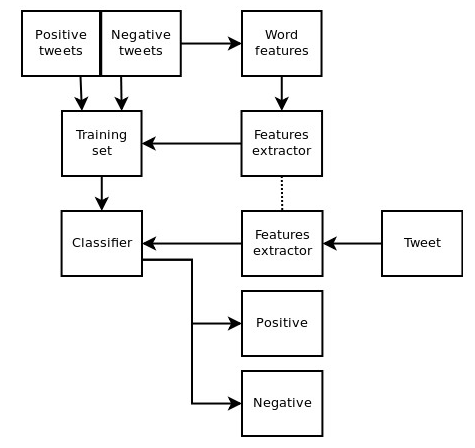
Operating System: - Linux (Ubuntu, fedora etc.), window (window 8, window 10, XP)

Tools: - Python, Jupyter

**Chapter 7**

**Detailed Life Cycle of the Project**

**7.1 Flow diagrams**

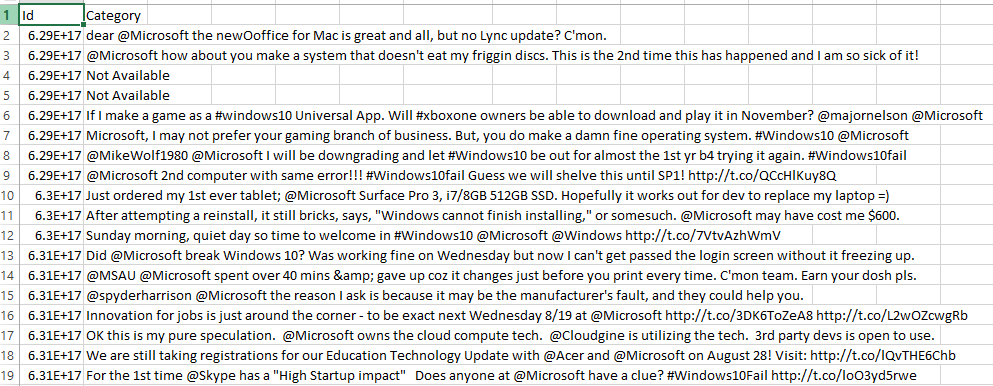


7.1 flow chart

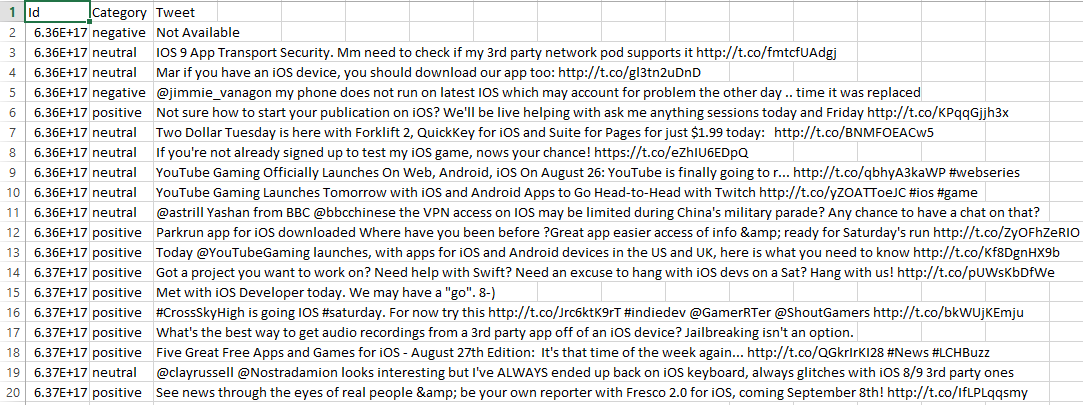
**7.2 Input and Output Screen Design**

Dataset:

Testdata.csv



Train.csv



Sentiment Analysis on training dataset

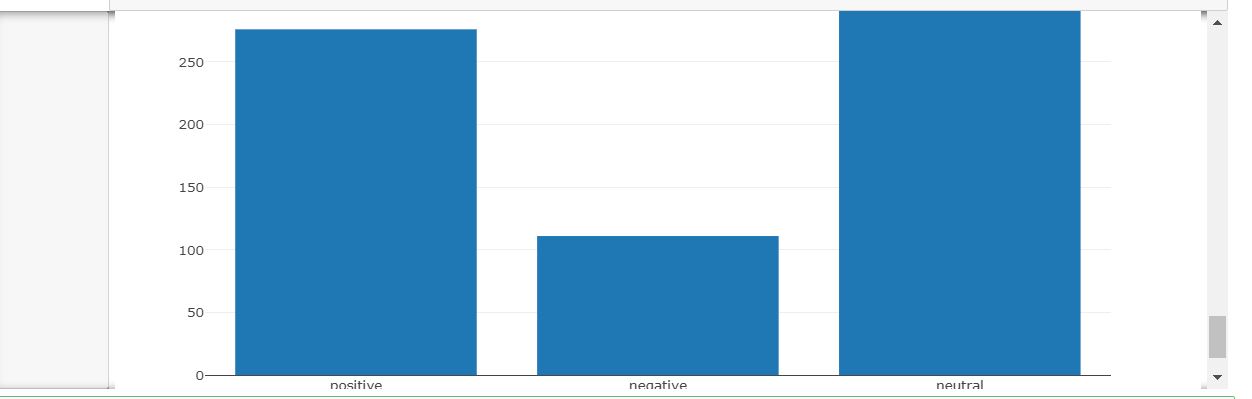
****

Fig 1

Data cleansing

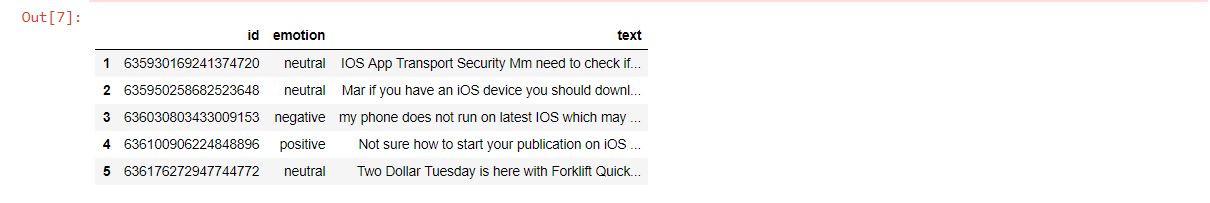
****

Table 1

Tokenization and stemming

****

Table 2

Create Wordlist

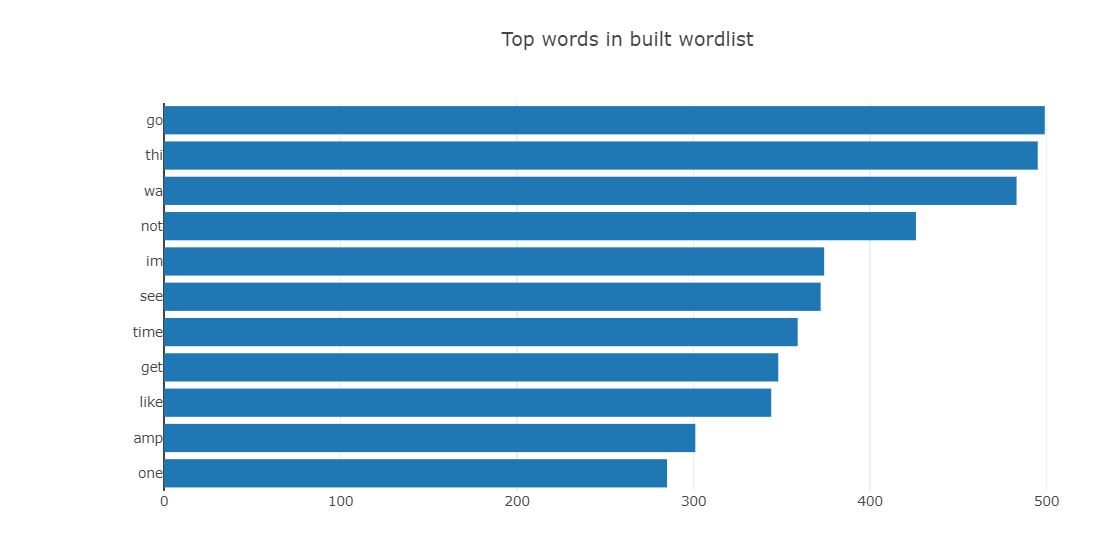


Fig 2

Bag of words

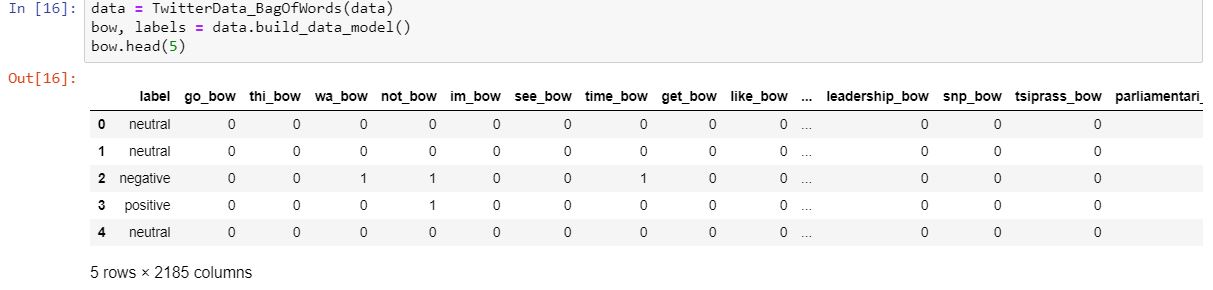


Table 3

Most common word of the sentiments

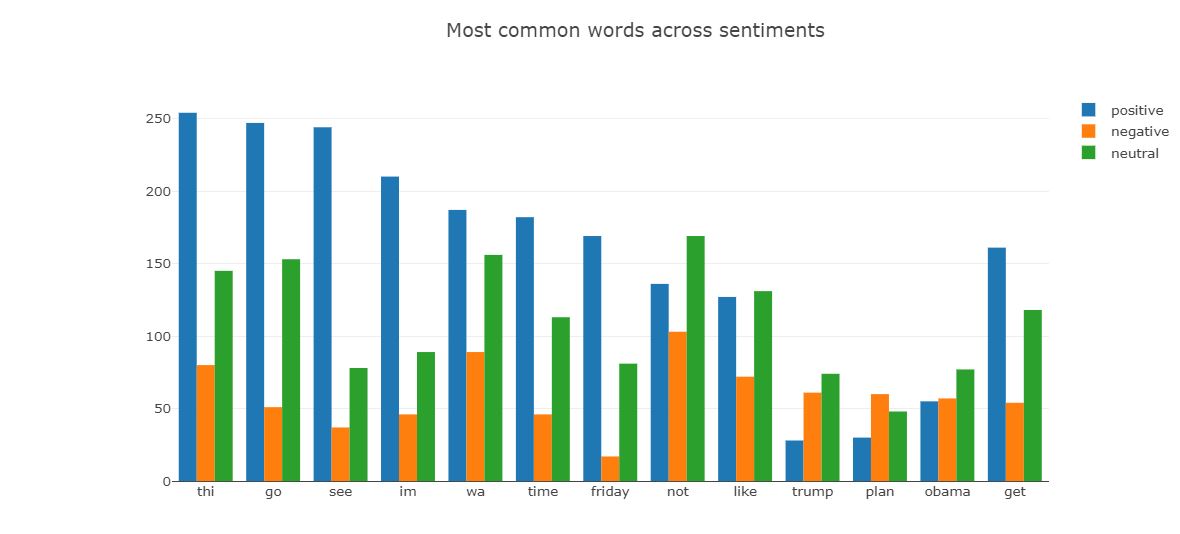
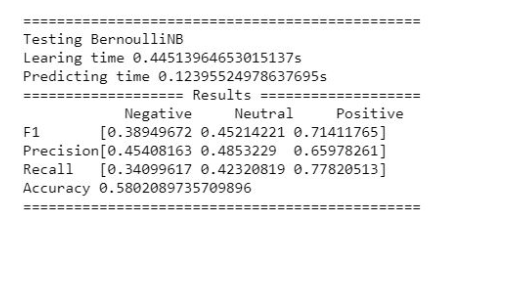


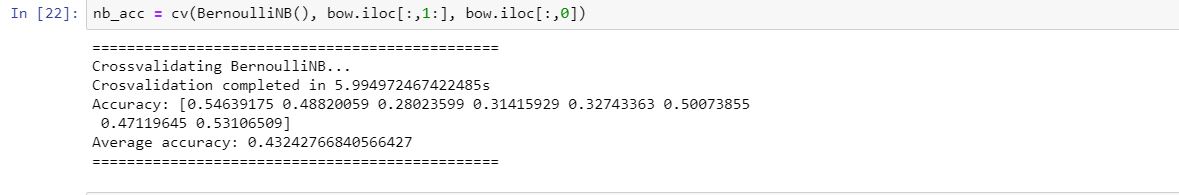
Fig 3

**Use classification**

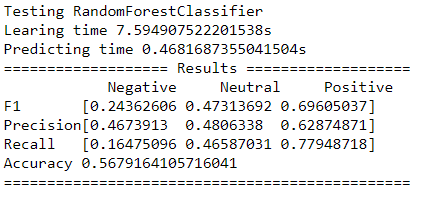
**Bag of word +naïve Bayes**

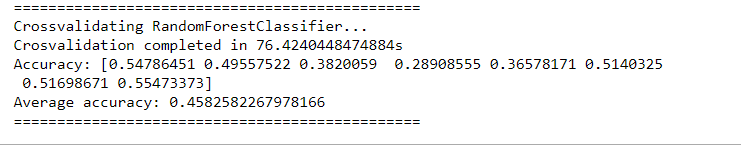


**Cross validation**



**Use RandomForestClassifier**





Word to vector

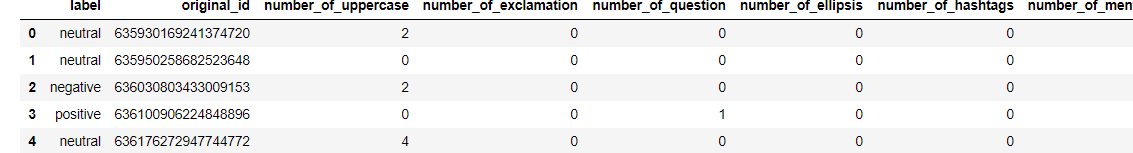


Table 4

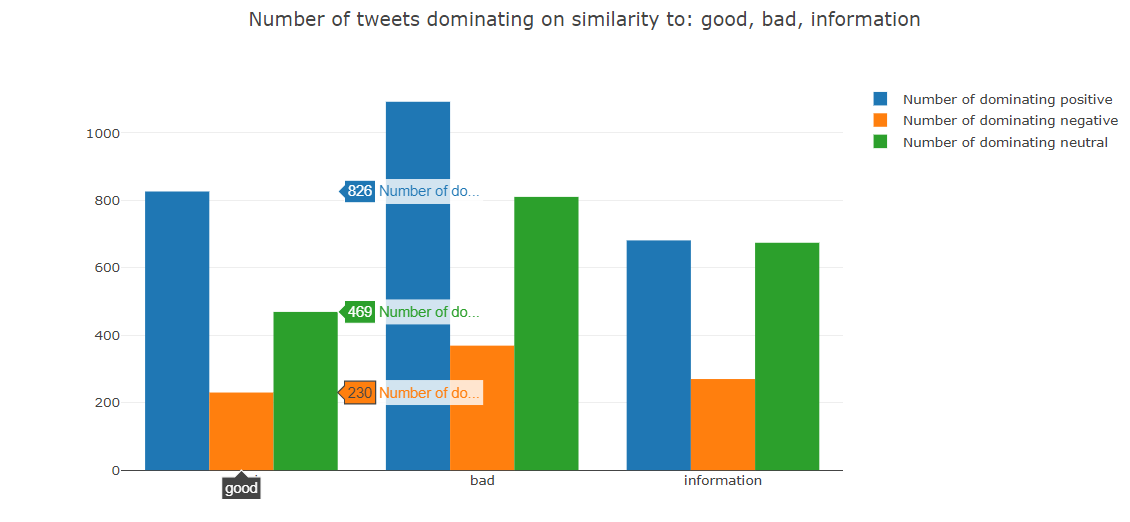
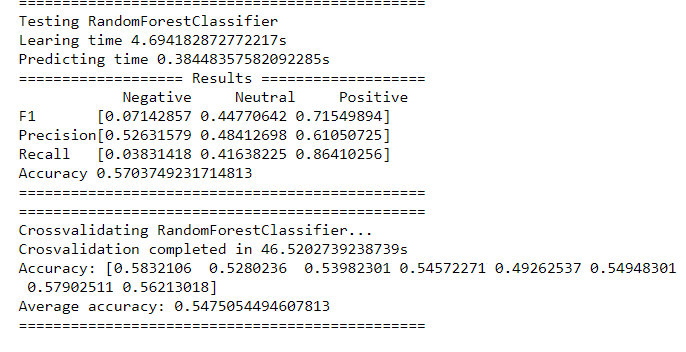
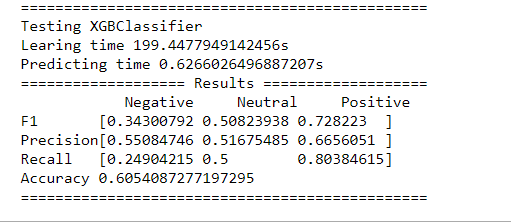


Fig 4

Tesing RandomClassifier with cross validation



Testing XGBClassifier



Final model

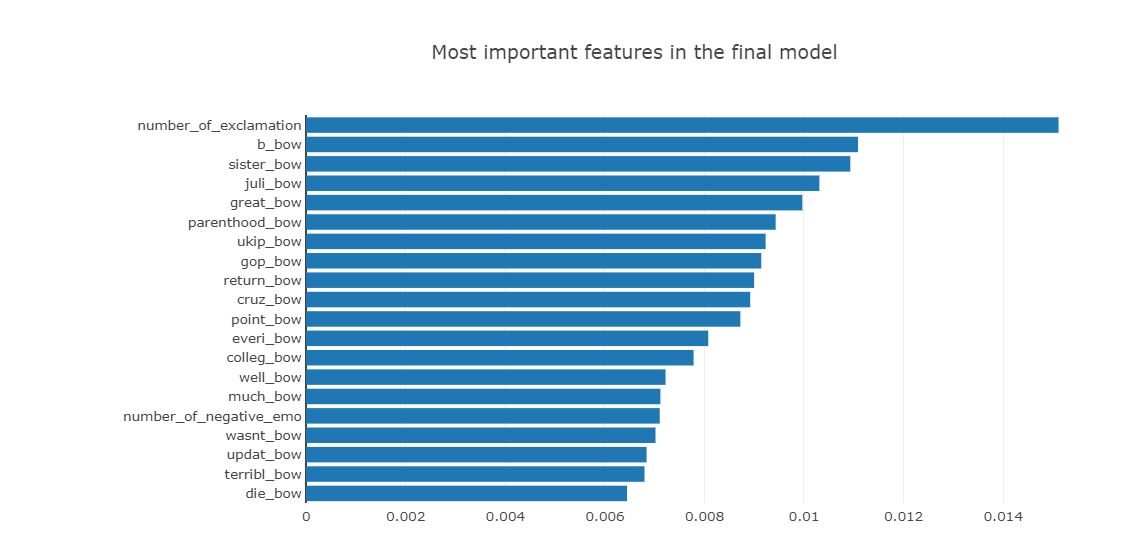


Fig 5

**Conclusion**

Experiment showed that prediction of text sentiment is a non-trivial task for machine learning. A lot of pre-processing is required just to be able to run any algorithm and see - usually not great - results. Main problem for sentiment analysis is to craft the machine representation of the text. Simple bag-of-words was definitely not enough to obtain satisfying results, thus a lot of additional features were created basing on common sense (number of emoticons, exclamation marks etc.). Word2vec representation significantly raised the predictions quality. I think that a slight improvement in classification accuracy for the given training dataset could be developed, but since it contained highly skewed data (small number of negative cases), the difference will be probably in the order of a few percent.

**Future Enhancements**

From future perspective, we would like to extend this project by implementing some machine learning algorithms for applications like election results, product ratings, movies' outcomes and running the project on clusters to expand its functionalities. Moreover, we would like to make a web application for users to input keywords, and get analysed results. In this project, we have worked only with unigram models, but we would like to extend it to bigram and further which will increase linkage between the data and provide accurate sentiment analysis results. Computation of overall tweet score can be done for a single keyword which can provide an overall sentiment of public regarding a topic.

**Printout of the code sheet**

#use python library

from collections import Counter

import nltk

import gensim.models.keyedvectors as word2vec

import pandas as pd

from emoticons import EmoticonDetector

import re as regex

import numpy as np

import plotly

from plotly import graph\_objs

from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV, RandomizedSearchCV

from time import time

import gensim

# plotly configuration

plotly.offline.init\_notebook\_mode()

class TwitterData\_Initialize():

data = []

processed\_data = []

wordlist = []

data\_model = None

data\_labels = None

is\_testing = False

def initialize(self, csv\_file, is\_testing\_set=False, from\_cached=None):

if from\_cached is not None:

self.data\_model = pd.read\_csv(from\_cached)

return

self.is\_testing = is\_testing\_set

if not is\_testing\_set:

self.data = pd.read\_csv(csv\_file, header=0, names=["id", "emotion", "text"])

self.data = self.data[self.data["emotion"].isin(["positive", "negative", "neutral"])]

else:

self.data = pd.read\_csv(csv\_file, header=0, names=["id", "text"],dtype={"id":"int64","text":"str"},nrows=4000)

not\_null\_text = 1 ^ pd.isnull(self.data["text"])

not\_null\_id = 1 ^ pd.isnull(self.data["id"])

self.data = self.data.loc[not\_null\_id & not\_null\_text, :]

self.processed\_data = self.data

self.wordlist = []

self.data\_model = None

self.data\_labels = None

data = TwitterData\_Initialize()

data.initialize("data\\train.csv")

data.processed\_data.head(5)

#data Distribution

df = data.processed\_data

neg = len(df[df["emotion"] == "negative"])

pos = len(df[df["emotion"] == "positive"])

neu = len(df[df["emotion"] == "neutral"])

dist = [

graph\_objs.Bar(

x=["negative","neutral","positive"],

y=[neg, neu, pos],

)]

plotly.offline.iplot({"data":dist, "layout":graph\_objs.Layout(title="Sentiment type distribution in training set")})

class TwitterCleanuper:

def iterate(self):

for cleanup\_method in [self.remove\_urls,

self.remove\_usernames,

self.remove\_na,

self.remove\_special\_chars,

self.remove\_numbers]:

yield cleanup\_method

@staticmethod

def remove\_by\_regex(tweets, regexp):

tweets.loc[:, "text"].replace(regexp, "", inplace=True)

return tweets

def remove\_urls(self, tweets):

return TwitterCleanuper.remove\_by\_regex(tweets, regex.compile(r"http.?://[^\s]+[\s]?"))

def remove\_na(self, tweets):

return tweets[tweets["text"] != "Not Available"]

def remove\_special\_chars(self, tweets): # it unrolls the hashtags to normal words

for remove in map(lambda r: regex.compile(regex.escape(r)), [",", ":", "\"", "=", "&", ";", "%", "$",

"@", "%", "^", "\*", "(", ")", "{", "}",

"[", "]", "|", "/", "\\", ">", "<", "-",

"!", "?", ".", "'",

"--", "---", "#"]):

tweets.loc[:, "text"].replace(remove, "", inplace=True)

return tweets

def remove\_usernames(self, tweets):

return TwitterCleanuper.remove\_by\_regex(tweets, regex.compile(r"@[^\s]+[\s]?"))

def remove\_numbers(self, tweets):

return TwitterCleanuper.remove\_by\_regex(tweets, regex.compile(r"\s?[0-9]+\.?[0-9]\*"))

class TwitterData\_Cleansing(TwitterData\_Initialize):

def \_\_init\_\_(self, previous):

self.processed\_data = previous.processed\_data

def cleanup(self, cleanuper):

t = self.processed\_data

for cleanup\_method in cleanuper.iterate():

if not self.is\_testing:

t = cleanup\_method(t)

else:

if cleanup\_method.\_\_name\_\_ != "remove\_na":

t = cleanup\_method(t)

self.processed\_data = t

class TwitterData\_TokenStem(TwitterData\_Cleansing):

def \_\_init\_\_(self, previous):

self.processed\_data = previous.processed\_data

def stem(self, stemmer=nltk.PorterStemmer()):

def stem\_and\_join(row):

row["text"] = list(map(lambda str: stemmer.stem(str.lower()), row["text"]))

return row

self.processed\_data = self.processed\_data.apply(stem\_and\_join, axis=1)

def tokenize(self, tokenizer=nltk.word\_tokenize):

def tokenize\_row(row):

row["text"] = tokenizer(row["text"])

row["tokenized\_text"] = [] + row["text"]

return row

self.processed\_data = self.processed\_data.apply(tokenize\_row, axis=1)

data = TwitterData\_TokenStem(data)

data.tokenize()

data.stem()

data.processed\_data.head(5)

#building wordlist

words = Counter()

for idx in data.processed\_data.index:

words.update(data.processed\_data.loc[idx, "text"])

words.most\_common(5)

stopwords=nltk.corpus.stopwords.words("english")

whitelist = ["n't", "not"]

for idx, stop\_word in enumerate(stopwords):

if stop\_word not in whitelist:

del words[stop\_word]

words.most\_common(5)

class TwitterData\_Wordlist(TwitterData\_TokenStem):

def \_\_init\_\_(self, previous):

self.processed\_data = previous.processed\_data

whitelist = ["n't","not"]

wordlist = []

def build\_wordlist(self, min\_occurrences=3, max\_occurences=500, stopwords=nltk.corpus.stopwords.words("english"),

whitelist=None):

self.wordlist = []

whitelist = self.whitelist if whitelist is None else whitelist

import os

if os.path.isfile("data\\wordlist.csv"):

word\_df = pd.read\_csv("data\\wordlist.csv")

word\_df = word\_df[word\_df["occurrences"] > min\_occurrences]

self.wordlist = list(word\_df.loc[:, "word"])

return

words = Counter()

for idx in self.processed\_data.index:

words.update(self.processed\_data.loc[idx, "text"])

for idx, stop\_word in enumerate(stopwords):

if stop\_word not in whitelist:

del words[stop\_word]

word\_df = pd.DataFrame(data={"word": [k for k, v in words.most\_common() if min\_occurrences < v < max\_occurences],

"occurrences": [v for k, v in words.most\_common() if min\_occurrences < v < max\_occurences]},

columns=["word", "occurrences"])

word\_df.to\_csv("data\\wordlist.csv", index\_label="idx")

self.wordlist = [k for k, v in words.most\_common() if min\_occurrences < v < max\_occurences]

data = TwitterData\_Wordlist(data)

data.build\_wordlist()

words = pd.read\_csv("data\\wordlist.csv")

x\_words = list(words.loc[0:10,"word"])

x\_words.reverse()

y\_occ = list(words.loc[0:10,"occurrences"])

y\_occ.reverse()

dist = [

graph\_objs.Bar(

x=y\_occ,

y=x\_words,

orientation="h"

)]

plotly.offline.iplot({"data":dist, "layout":graph\_objs.Layout(title="Top words in built wordlist")})

#bag of words

class TwitterData\_BagOfWords(TwitterData\_Wordlist):

def \_\_init\_\_(self, previous):

self.processed\_data = previous.processed\_data

self.wordlist = previous.wordlist

def build\_data\_model(self):

label\_column = []

if not self.is\_testing:

label\_column = ["label"]

columns = label\_column + list(

map(lambda w: w + "\_bow",self.wordlist))

labels = []

rows = []

for idx in self.processed\_data.index:

current\_row = []

if not self.is\_testing:

# add label

current\_label = self.processed\_data.loc[idx, "emotion"]

labels.append(current\_label)

current\_row.append(current\_label)

# add bag-of-words

tokens = set(self.processed\_data.loc[idx, "text"])

for \_, word in enumerate(self.wordlist):

current\_row.append(1 if word in tokens else 0)

rows.append(current\_row)

self.data\_model = pd.DataFrame(rows, columns=columns)

self.data\_labels = pd.Series(labels)

return self.data\_model, self.data\_labels

data = TwitterData\_BagOfWords(data)

bow, labels = data.build\_data\_model()

bow.head(5)

grouped = bow.groupby(["label"]).sum()

words\_to\_visualize = []

sentiments = ["positive","negative","neutral"]

#get the most 7 common words for every sentiment

for sentiment in sentiments:

words = grouped.loc[sentiment,:]

words.sort\_values(inplace=True,ascending=False)

for w in words.index[:7]:

if w not in words\_to\_visualize:

words\_to\_visualize.append(w)

#visualize it

plot\_data = []

for sentiment in sentiments:

plot\_data.append(graph\_objs.Bar(

x = [w.split("\_")[0] for w in words\_to\_visualize],

y = [grouped.loc[sentiment,w] for w in words\_to\_visualize],

name = sentiment

))

plotly.offline.iplot({

"data":plot\_data,

"layout":graph\_objs.Layout(title="Most common words across sentiments")

})

#classification

import random

seed = 666

random.seed(seed)

def test\_classifier(X\_train, y\_train, X\_test, y\_test, classifier):

log("")

log("===============================================")

classifier\_name = str(type(classifier).\_\_name\_\_)

log("Testing " + classifier\_name)

now = time()

list\_of\_labels = sorted(list(set(y\_train)))

model = classifier.fit(X\_train, y\_train)

log("Learing time {0}s".format(time() - now))

now = time()

predictions = model.predict(X\_test)

log("Predicting time {0}s".format(time() - now))

precision = precision\_score(y\_test, predictions, average=None, pos\_label=None, labels=list\_of\_labels)

recall = recall\_score(y\_test, predictions, average=None, pos\_label=None, labels=list\_of\_labels)

accuracy = accuracy\_score(y\_test, predictions)

f1 = f1\_score(y\_test, predictions, average=None, pos\_label=None, labels=list\_of\_labels)

log("=================== Results ===================")

log(" Negative Neutral Positive")

log("F1 " + str(f1))

log("Precision" + str(precision))

log("Recall " + str(recall))

log("Accuracy " + str(accuracy))

log("===============================================")

return precision, recall, accuracy, f1

def log(x):

#can be used to write to log file

print(x)

#bag of word +Naive bayes

from sklearn.naive\_bayes import BernoulliNB

X\_train, X\_test, y\_train, y\_test = train\_test\_split(bow.iloc[:, 1:], bow.iloc[:, 0],

train\_size=0.7, stratify=bow.iloc[:, 0],

random\_state=seed)

precision, recall, accuracy, f1 = test\_classifier(X\_train, y\_train, X\_test, y\_test, BernoulliNB())

#cross validation

def cv(classifier, X\_train, y\_train):

log("===============================================")

classifier\_name = str(type(classifier).\_\_name\_\_)

now = time()

log("Crossvalidating " + classifier\_name + "...")

accuracy = [cross\_val\_score(classifier, X\_train, y\_train, cv=8, n\_jobs=-1)]

log("Crosvalidation completed in {0}s".format(time() - now))

log("Accuracy: " + str(accuracy[0]))

log("Average accuracy: " + str(np.array(accuracy[0]).mean()))

log("===============================================")

return accuracy

nb\_acc = cv(BernoulliNB(), bow.iloc[:,1:], bow.iloc[:,0])

#finding emojis

class EmoticonDetector:

emoticons = {}

def \_\_init\_\_(self, emoticon\_file="data\\emoticons.txt"):

from pathlib import Path

content = Path(emoticon\_file).read\_text()

positive = True

for line in content.split("\n"):

if "positive" in line.lower():

positive = True

continue

elif "negative" in line.lower():

positive = False

continue

self.emoticons[line] = positive

def is\_positive(self, emoticon):

if emoticon in self.emoticons:

return self.emoticons[emoticon]

return False

def is\_emoticon(self, to\_check):

return to\_check in self.emoticons

class TwitterData\_ExtraFeatures(TwitterData\_Wordlist):

def \_\_init\_\_(self):

pass

def build\_data\_model(self):

extra\_columns = [col for col in self.processed\_data.columns if col.startswith("number\_of")]

label\_column = []

if not self.is\_testing:

label\_column = ["label"]

columns = label\_column + extra\_columns + list(

map(lambda w: w + "\_bow",self.wordlist))

labels = []

rows = []

for idx in self.processed\_data.index:

current\_row = []

if not self.is\_testing:

# add label

current\_label = self.processed\_data.loc[idx, "emotion"]

labels.append(current\_label)

current\_row.append(current\_label)

for \_, col in enumerate(extra\_columns):

current\_row.append(self.processed\_data.loc[idx, col])

# add bag-of-words

tokens = set(self.processed\_data.loc[idx, "text"])

for \_, word in enumerate(self.wordlist):

current\_row.append(1 if word in tokens else 0)

rows.append(current\_row)

self.data\_model = pd.DataFrame(rows, columns=columns)

self.data\_labels = pd.Series(labels)

return self.data\_model, self.data\_labels

def build\_features(self):

def count\_by\_lambda(expression, word\_array):

return len(list(filter(expression, word\_array)))

def count\_occurences(character, word\_array):

counter = 0

for j, word in enumerate(word\_array):

for char in word:

if char == character:

counter += 1

return counter

def count\_by\_regex(regex, plain\_text):

return len(regex.findall(plain\_text))

self.add\_column("splitted\_text", map(lambda txt: txt.split(" "), self.processed\_data["text"]))

# number of uppercase words

uppercase = list(map(lambda txt: count\_by\_lambda(lambda word: word == word.upper(), txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_uppercase", uppercase)

# number of !

exclamations = list(map(lambda txt: count\_occurences("!", txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_exclamation", exclamations)

# number of ?

questions = list(map(lambda txt: count\_occurences("?", txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_question", questions)

# number of ...

ellipsis = list(map(lambda txt: count\_by\_regex(regex.compile(r"\.\s?\.\s?\."), txt),

self.processed\_data["text"]))

self.add\_column("number\_of\_ellipsis", ellipsis)

# number of hashtags

hashtags = list(map(lambda txt: count\_occurences("#", txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_hashtags", hashtags)

# number of mentions

mentions = list(map(lambda txt: count\_occurences("@", txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_mentions", mentions)

# number of quotes

quotes = list(map(lambda plain\_text: int(count\_occurences("'", [plain\_text.strip("'").strip('"')]) / 2 +

count\_occurences('"', [plain\_text.strip("'").strip('"')]) / 2),

self.processed\_data["text"]))

self.add\_column("number\_of\_quotes", quotes)

# number of urls

urls = list(map(lambda txt: count\_by\_regex(regex.compile(r"http.?://[^\s]+[\s]?"), txt),

self.processed\_data["text"]))

self.add\_column("number\_of\_urls", urls)

# number of positive emoticons

ed = EmoticonDetector()

positive\_emo = list(

map(lambda txt: count\_by\_lambda(lambda word: ed.is\_emoticon(word) and ed.is\_positive(word), txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_positive\_emo", positive\_emo)

# number of negative emoticons

negative\_emo = list(map(

lambda txt: count\_by\_lambda(lambda word: ed.is\_emoticon(word) and not ed.is\_positive(word), txt),

self.processed\_data["splitted\_text"]))

self.add\_column("number\_of\_negative\_emo", negative\_emo)

def add\_column(self, column\_name, column\_content):

self.processed\_data.loc[:, column\_name] = pd.Series(column\_content, index=self.processed\_data.index)

data = TwitterData\_ExtraFeatures()

data.initialize("data\\train.csv")

data.build\_features()

data.cleanup(TwitterCleanuper())

data.tokenize()

data.stem()

data.build\_wordlist()

data\_model, labels = data.build\_data\_model()

data\_model.head(5)

sentiments = ["positive","negative","neutral"]

plots\_data\_ef = []

for what in map(lambda o: "number\_of\_"+o,["positive\_emo","negative\_emo","exclamation","hashtags","question"]):

ef\_grouped = data\_model[data\_model[what]>=1].groupby(["label"]).count()

plots\_data\_ef.append({"data":[graph\_objs.Bar(

x = sentiments,

y = [ef\_grouped.loc[s,:][0] for s in sentiments],

)], "title":"How feature \""+what+"\" separates the tweets"})

for plot\_data\_ef in plots\_data\_ef:

plotly.offline.iplot({

"data":plot\_data\_ef["data"],

"layout":graph\_objs.Layout(title=plot\_data\_ef["title"])

})

from sklearn.ensemble import RandomForestClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_model.iloc[:, 1:], data\_model.iloc[:, 0],

train\_size=0.7, stratify=data\_model.iloc[:, 0],

random\_state=seed)

precision, recall, accuracy, f1 = test\_classifier(X\_train, y\_train, X\_test, y\_test, RandomForestClassifier(random\_state=seed,n\_estimators=403,n\_jobs=-1))

rf\_acc = cv(RandomForestClassifier(n\_estimators=403,n\_jobs=-1, random\_state=seed),data\_model.iloc[:, 1:], data\_model.iloc[:, 0])

class Word2VecProvider(object):

word2vec = None

dimensions = 0

def load(self, path\_to\_word2vec):

self.word2vec = gensim.models.KeyedVectors.load\_word2vec\_format(path\_to\_word2vec, binary=False)

self.word2vec.init\_sims(replace=True)

self.dimensions = self.word2vec.vector\_size

def get\_vector(self, word):

if word not in self.word2vec.vocab:

return None

return self.word2vec.syn0norm[self.word2vec.vocab[word].index]

def get\_similarity(self, word1, word2):

if word1 not in self.word2vec.vocab or word2 not in self.word2vec.vocab:

return None

return self.word2vec.similarity(word1, word2)

class TwitterData(TwitterData\_ExtraFeatures):

def build\_final\_model(self, word2vec\_provider, stopwords=nltk.corpus.stopwords.words("english")):

whitelist = self.whitelist

stopwords = list(filter(lambda sw: sw not in whitelist, stopwords))

extra\_columns = [col for col in self.processed\_data.columns if col.startswith("number\_of")]

similarity\_columns = ["bad\_similarity", "good\_similarity", "information\_similarity"]

label\_column = []

if not self.is\_testing:

label\_column = ["label"]

columns = label\_column + ["original\_id"] + extra\_columns + similarity\_columns + list(

map(lambda i: "word2vec\_{0}".format(i), range(0, word2vec\_provider.dimensions))) + list(

map(lambda w: w + "\_bow",self.wordlist))

labels = []

rows = []

for idx in self.processed\_data.index:

current\_row = []

if not self.is\_testing:

# add label

current\_label = self.processed\_data.loc[idx, "emotion"]

labels.append(current\_label)

current\_row.append(current\_label)

current\_row.append(self.processed\_data.loc[idx, "id"])

for \_, col in enumerate(extra\_columns):

current\_row.append(self.processed\_data.loc[idx, col])

# average similarities with words

tokens = self.processed\_data.loc[idx, "tokenized\_text"]

for main\_word in map(lambda w: w.split("\_")[0], similarity\_columns):

current\_similarities = [abs(sim) for sim in

map(lambda word: word2vec\_provider.get\_similarity(main\_word, word.lower()), tokens) if

sim is not None]

if len(current\_similarities) <= 1:

current\_row.append(0 if len(current\_similarities) == 0 else current\_similarities[0])

continue

max\_sim = max(current\_similarities)

min\_sim = min(current\_similarities)

current\_similarities = [((sim - min\_sim) / (max\_sim - min\_sim)) for sim in

current\_similarities] # normalize to <0;1>

current\_row.append(np.array(current\_similarities).mean())

# add word2vec vector

tokens = self.processed\_data.loc[idx, "tokenized\_text"]

current\_word2vec = []

for \_, word in enumerate(tokens):

vec = word2vec\_provider.get\_vector(word.lower())

if vec is not None:

current\_word2vec.append(vec)

averaged\_word2vec = list(np.array(current\_word2vec).mean(axis=0))

current\_row += averaged\_word2vec

# add bag-of-words

tokens = set(self.processed\_data.loc[idx, "text"])

for \_, word in enumerate(self.wordlist):

current\_row.append(1 if word in tokens else 0)

rows.append(current\_row)

self.data\_model = pd.DataFrame(rows, columns=columns)

self.data\_labels = pd.Series(labels)

return self.data\_model, self.data\_labels

td = TwitterData()

td.initialize("data\\train.csv")

td.build\_features()

td.cleanup(TwitterCleanuper())

td.tokenize()

td.stem()

td.build\_wordlist()

td.build\_final\_model(word2vec)

td.data\_model.head(5)

data\_model = td.data\_model

data\_model.drop("original\_id",axis=1,inplace=True)

columns\_to\_plot = ["bad\_similarity", "good\_similarity", "information\_similarity"]

bad, good, info = columns\_to\_plot

sentiments = ["positive","negative","neutral"]

only\_positive = data\_model[data\_model[good]>=data\_model[bad]]

only\_positive = only\_positive[only\_positive[good]>=only\_positive[info]].groupby(["label"]).count()

only\_negative = data\_model[data\_model[bad] >= data\_model[good]]

only\_negative = only\_negative[only\_negative[bad] >= only\_negative[info]].groupby(["label"]).count()

only\_info = data\_model[data\_model[info]>=data\_model[good]]

only\_info = only\_info[only\_info[info]>=only\_info[bad]].groupby(["label"]).count()

plot\_data\_w2v = []

for sentiment in sentiments:

plot\_data\_w2v.append(graph\_objs.Bar(

x = ["good","bad", "information"],

y = [only\_positive.loc[sentiment,:][0], only\_negative.loc[sentiment,:][0], only\_info.loc[sentiment,:][0]],

name = "Number of dominating " + sentiment

))

plotly.offline.iplot({

"data":plot\_data\_w2v,

"layout":graph\_objs.Layout(title="Number of tweets dominating on similarity to: good, bad, information")

})

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_model.iloc[:, 1:], data\_model.iloc[:, 0],

train\_size=0.7, stratify=data\_model.iloc[:, 0],

random\_state=seed)

precision, recall, accuracy, f1 = test\_classifier(X\_train, y\_train, X\_test, y\_test, RandomForestClassifier(n\_estimators=403,n\_jobs=-1, random\_state=seed))

rf\_acc = cv(RandomForestClassifier(n\_estimators=403,n\_jobs=-1,random\_state=seed),data\_model.iloc[:, 1:], data\_model.iloc[:, 0])

from xgboost import XGBClassifier as XGBoostClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_model.iloc[:, 1:], data\_model.iloc[:, 0],

train\_size=0.7, stratify=data\_model.iloc[:, 0],

random\_state=seed)

precision, recall, accuracy, f1 = test\_classifier(X\_train, y\_train, X\_test, y\_test, XGBoostClassifier(seed=seed))

xgb\_acc = cv(XGBoostClassifier(seed=seed),data\_model.iloc[:, 1:], data\_model.iloc[:, 0])

def report(results, n\_top=3):

for i in range(1, n\_top + 1):

candidates = np.flatnonzero(results['rank\_test\_score'] == i)

for candidate in candidates:

log("Model with rank: {0}".format(i))

log("Mean validation score: {0:.3f} (std: {1:.3f})".format(

results['mean\_test\_score'][candidate],

results['std\_test\_score'][candidate]))

log("Parameters: {0}".format(results['params'][candidate]))

log("")

def best\_fit(X\_train, y\_train, n\_iter=5):

parameters = {

"n\_estimators":[103,201, 403],

"max\_depth":[3,10,15, 30],

"objective":["multi:softmax","binary:logistic"],

"learning\_rate":[0.05, 0.1, 0.15, 0.3]

}

rand\_search = RandomizedSearchCV(XGBoostClassifier(seed=seed),param\_distributions=parameters,

n\_iter=n\_iter,scoring="accuracy",

n\_jobs=-1,cv=8)

import time as ttt

now = time()

log(ttt.ctime())

rand\_search.fit(X\_train, y\_train)

report(rand\_search.cv\_results\_, 10)

log(ttt.ctime())

log("Search took: " + str(time() - now))

test\_data = TwitterData()

test\_data.initialize("data\\test.csv", is\_testing\_set=True)

test\_data.build\_features()

test\_data.cleanup(TwitterCleanuper())

test\_data.tokenize()

test\_data.stem()

test\_data.build\_wordlist()

test\_data.build\_final\_model(word2vec)

test\_data.data\_model.head(5)

results = pd.DataFrame([],columns=["Id","Category"])

results["Id"] = test\_model["original\_id"].astype("int64")

results["Category"] = predictions

results.to\_csv("results\_xgb.csv",index=False)

features = {}

for idx, fi in enumerate(xgboost.feature\_importances\_):

features[test\_model.columns[1+idx]] = fi

important = []

for f in sorted(features,key=features.get,reverse=True):

important.append((f,features[f]))

# print(f + " " + str(features[f]))

to\_show = list(filter(lambda f: not f[0].startswith("word2vec") and not f[0].endswith("\_similarity"),important))[:20]

to\_show.reverse()

features\_importance = [

graph\_objs.Bar(

x=[f[1] for f in to\_show],

y=[f[0] for f in to\_show],

orientation="h"

)]

plotly.offline.iplot({"data":features\_importance, "layout":graph\_objs.Layout(title="Most important features in the final model",

margin=graph\_objs.Margin(

l=200,

pad=3

),)})

**Bibliography/references**

[1] M.Rambocas, and J. Gama, “Marketing Research:TheRoleof

Sentiment Analysis”. The 5th SNA-KDD Workshop’11. University of Porto,2013.

[2] A. K. Jose, N. Bhatia, and S. Krishna, “Twitter Sentiment Analysis”.

National Institute of TechnologyCalicut,2010.

[3] P. Lai, “Extracting Strong Sentiment TrendfromTwitter”. Stanford

University, 2012.

[4] Y. Zhou, and Y. Fan, “ A Sociolinguistic Study of American Slang,”

Theory and Practice in Language Studies, 3(12), 2209–2213, 2013.

doi:10.4304/tpls.3.12.2209-2213