# “DEDUCTION OF SUBJECTIVE REMARKS TO STATISTICAL EQUIVALENCE BY SENTIMENT ANALYSIS”

PROGRAMMING LANGUAGE: Python

**TEAM MEMBERS:**

Abhishek Rattihalli (800959105)
Sanju K H (800953525)
Sujal T V (800969246)
Venkatesh Bonageri (800964302)
Vikas Deshpande (800964852)
Rakesh Harish (800984018)

UNDER THE GUIDANCE OF

**Dr. Ali Sever**

`

**DEPARTMENT OF COMPUTER SCIENCE**

**COLLEGE OF COMPUTING AND INFORMATICS**
**CHARLOTTE,**

**NORTH CAROLINA - 28262**

**SPRING 2017**

# TABLE OF CONTENTS

## 1. ABSTRACT

This work is intended to aim at textual analysis of subjective/sentimental remarks and infer their statistical equivalence. This proposal considers comments made on and stars rated to online product purchases made by users in Amazon. Amazon data are pursued especially since they are abundantly available for Amazon product reviews, although this implementation can be extended to any instance of this paradigm.

## 2. INTRODUCTION

Sentiment Analysis is one of the interesting applications of text analytics. Although it is often associated with sentiment classification of documents, broadly speaking it refers to the use of text analytics approaches applied to the set of problems related to identifying and extracting subjective material in text sources.

In a survey, for 9 out of 10 customers, reviews are as important as personal recommendations. Nowadays, customers are likely to spend on a product only after reading positive reviews. Rating depicts the sentiment of the review and how strongly a user is recommending the product.

The goal of the project is to predict the rating of a product based on reviews, so that a user can decide whether to buy the product or not just looking at ratings instead of reading all the reviews. For the implementation, we will be using **Amazon Fine Food Review Dataset** *[1]*. The dataset consists of about 568,454 reviews. The project can be used to any dataset which involves user reviews and can predict its ratings.

## 3. MOTIVATION:

The motivation for doing this project was primarily from the paper/research activity steered by three Stanford PhD students **- Predicting Star Ratings of Movie Review Comments** *[2]*. We have considered the product reviews from users of E Commerce company – Amazon, and have analyzed the ratings using Machine Learning algorithms. Using machine learning techniques to infer the polarity of text-based comments is significant in this age of information. The aim of this project is to predict the star rating of a user’s comment about a product on a scale of 1, 2, 3 … to 5. Though a user self-classifies a comment on a scale of 1 to 5 in our dataset from Amazon, this research can potentially be applied to settings where a comment about any product is available.

## 4. **LANGUAGE USED**: Python 3.6

### a. Paradigm of Python

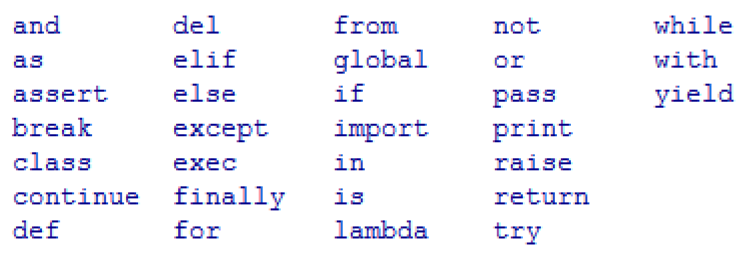
Python is a multi-paradigm programming language - object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming.

### b. History of Python

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need programming interfaces. Finally, Python is portable across all major hardware and software platforms.

The first development of Python was started in 1989 and its initial version was released in 1991. Many of Python's features originated from an interpreted language called ABC. The most recent version of Python is version 3 that was released in year 2008. The latest variant of it is 3.6.1 which was released in 2017.

Python’s primitive built-in data types include Booleans, numbers (machine integers, arbitrary-precision integers, and real and complex floating point numbers), and strings (8-bit and Unicode). The following identifiers are used as reserved words, or keywords of the Python



| | | | | |
|---|---|---|---|---|
| and | del | from | not | while |
| as | elif | global | or | with |
| assert | else | if | pass | yield |
| break | except | import | print | |
| class | exec | in | raise | |
| continue | finally | is | return | |
| def | for | lambda | try | |

## c. Strengths of Python

- Python is easy to read, use and maintain.
- Extremely popular in academia, creating a large talent pool.
- Simple syntax and readability
- Dynamically typed and flexible, with code that is less verbose.
- Python is multi paradigm and supports OO, procedural, and functional programming styles

## d. Weaknesses of Python

- Because it is an interpreted language, it is often many times slower than compiled languages
- Python is present on many server and desktop platforms, but it is weak in mobile computing and browsers
- Because the language is dynamically typed, it requires more testing and has errors that only show up at runtime

## e. Why Python for Machine Learning algorithms?

Python is mainly a scripting Language very easy to learn. SCIKIT-learn - machine learning in Python is one of the easiest and most advanced library used for this purpose. R comes with lots of great packages used for Data Analysis and Machine Learning, etc. However, Python is much easier to learn and use.

**f.** Characteristics of Python

➢ **Readability**

At first glance, Python seems a little weird. The syntax is all okay; everything is either C++ or English understandable, but there is something missing: braces. For block comments, Python uses a strict whitespace with indents and detents. In his article titled “Why Python?” on how he came to chose Python over Perl, Eric Raymond writes “I immediately tripped over the first odd feature of Python that everyone notices: the fact that whitespace (indentation) is actually significant in the language syntax. The language has no analog of the C and Perl brace syntax; instead, changes in indentation delimit statement groups. And, like most hackers on first realizing this fact, I recoiled in reflexive disgust.”

➢ **Writability**

Python is certainly readable only to the degree that it is writable. One cannot hope to understand or be able to write anything more complex than standard programming without thorough study of the documentation of the Python website. The documentation is superb, showing concrete examples and proving the lasting epithet that “there should be one- and preferably only one- way to do it.”

➢ **Reliability**

Judging by the gleaming reviews all over the Internet, one could say that Python surpasses all expectations in terms of reliability. Being as intensely reusable as all the code you write is, Python is certainly designed to last. With every updated version, only a few things ever get replaced; the developers opt to keep old things as long as they can while plugging the new features as better and easier to use. One example is the "not equal to" operator, which, in version 1.0, was simply "<>" like in other languages at the time. Nowadays, Python recommends the use of the "!=" syntax, noting that the "<>" still works, but is obsolete.

➢ **Abstraction**

Abstracting something means to give names to things, so that the name captures the core of what a program segment does.
Python like any other high level programming languages supports Object oriented concepts. Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy.
Abstraction in Python can be achieved by several ways as follows:

- Variables
- Methods or functions
- Classes and Objects
- modules

➢ **Syntax and Semantics of Python**

When writing programs, we must take care of

▪ **Semantics** - meaning of the program
There should not be more than one meaning associated with any statement because computer cannot figure out which is the correct intended meaning.
E.g.: In English if I say, "I'm having a friend for dinner", the statement can be interpreted in two different ways. Such things should not happen in Python.

▪ **Syntax** - specify grammar using programming language.
E.g.: "Mouse Cat Dog" is not in a correct syntax as far as English language is concerned. Similarly, there is a syntax in python.

## g. Basic control abstractions of Python

### i. if...else statement:

In the case, you want one thing to happen when a condition it true, and something else to happen when it is false; we use if...else in this scenario.

```
if expression:
        statement (s)
else:
        statement (s)
```

### ii. for loop:

Executes a sequence of statements multiple times and shortens the code that manages the loop variable.

```
for loop in sequence:
        statements
```

### iii. while statement:

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

```
while expression:
        statements
```

### iv. Nested Loops:

You can use one or more loop inside any other while or for loops.

```
if a < b:
  ST_A
else:
  if a > b:
     ST_B
  else:
     ST_C
```

v. **Break Statements:**

The loop statement would be terminated, and the execution would be transferred to the first available statement after the loop.

```
for loop in sequence:
        If statement:
                break
statements...
```

vi. **continue:**

Skips the rest of the body and immediately retests the condition before reiterating

## 5. PROBLEM DEFINITION

This work is intended to aim at textual analysis of subjective/sentimental remarks and infer their statistical equivalence. This proposal considers comments and stars(ratings) rated to online product purchases made by users. E-Commerce data are pursued especially since they are abundantly available, and this implementation can be extended to any instance of this paradigm.

## 6. IMPLEMENTED TECHNIQUES

- Decision Tree Classifier
- Neural Networks
- Support Vector Machines
- Random Forest Classification
- K Nearest Neighbor
- Gradient Boosting classifier

## 7. TECHNICAL DOCUMENTATION

**Programming languages used**: Python 3.6

**Reason:**
The proposed system has been implemented using Python programming language as it involves analysis of historical data with prediction and graphical representation of the output results, and Python is one of the most preferred languages for statistical data analysis.

**Tools and environments**:
Python 3.6  IDE: PyCharm

**Code:**

## Text Parser.py

```
import csv
import re
import string
import pandas as pd
from sklearn.svm import SVC
from collections import Counter
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.neural_network import MLPClassifier
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier as knc
from sklearn.ensemble import GradientBoostingClassifier as gbc

stopWords = []
dataSet = []
emoticons_str = r"""
    (?:
        [:=;] # Eyes
        [oO\-]? # Nose (optional)
        [D\)\]\(\]/\\OpP] # Mouth
    )"""
regex_str = [
    r'<[^>]+>',  # HTML tags
    r"(?:[a-z][a-z\-_]+[a-z])",  # words with - and '
    r'(?:[\w_]+)',  # other words
    r'(?:\S)'  # anything else
]
tokens_re = re.compile(r'(' + '|'.join(regex_str) + ')', re.VERBOSE | re.IGNORECASE)
emoticon_re = re.compile(r'^' + emoticons_str + '$', re.VERBOSE | re.IGNORECASE)
stemmer = PorterStemmer()
lemmatiser = WordNetLemmatizer()

def initializeSystem():
    stop = stopwords.words('english') + list(string.punctuation) + ['rt', 'via', 'i\'m', 'us', 'it']
    for x in stop:
        stopWords.append(stemmer.stem(lemmatiser.lemmatize(x, pos="v")))

def preprocess(s, lowercase=True):
    tokens = tokens_re.findall(s)
    if lowercase:
        tokens = [token if emoticon_re.search(token) else
stemmer.stem(lemmatiser.lemmatize(token.lower(), pos="v")) for
                    token in tokens]
    return tokens

def processString(string):
    terms_stop = [term for term in preprocess(string) if
                    term not in stopWords and len(str(term)) > 1 and not term.isnumeric()]
    return terms_stop

def loadFile(filePath):
    fileRead = open(filePath, "r")
    reader = csv.reader(fileRead, dialect='excel')
    for row in reader:
        temp = (row[1], row[-1])
        dataSet.append(temp)
    return dataSet

def prepareSparseMatrix(convertedReviews, decisionAttributes):
    sparseMatrix = []
    for cr in convertedReviews:
        newCr = [0] * len(decisionAttributes)
        for word in cr:
            if word in decisionAttributes:
                index = decisionAttributes.index(word)
                newCr[index] += 1
            else:
                pass
        sparseMatrix.append(newCr)
    return sparseMatrix

def convertReviews(reviews):
    convertedReviews = []
    for a in reviews:
        convertedReviews.append(processString(str(a).lower()))
    return convertedReviews

def getDecisionAttributes(convertedReviews):
    toCount = []
    decisionAttributes = []
    for a in convertedReviews:
        toCount.append(" ".join(a))
    str1 = ""
    for a in toCount:
        str1 += "".join(a)
    x = Counter(str1.split(" "))
    for (k, v) in x.most_common(min(500, len(x))):
        decisionAttributes.append(k)
    return decisionAttributes

def model_data(training_data):
    dtc = DecisionTreeClassifier(random_state=9, min_samples_split=5)
    dtc.fit(training_data['data'], training_data['result'])

    nn = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=1)
    nn.fit(training_data['data'], training_data['result'])

    svc = SVC(C=100, kernel="linear")
    svc.fit(training_data['data'], training_data['result'])

    rfc = RFC(n_estimators=10, criterion='entropy', max_depth=10, min_samples_split=5,
bootstrap='true', random_state=None)
    rfc.fit(training_data['data'], training_data['result'])


    knc_map = knc(n_neighbors=15, weights='distance')
    knc_map.fit(training_data['data'], training_data['result'])

    gbc_map = gbc(n_estimators=150, verbose=0)
    gbc_map.fit(training_data['data'], training_data['result'])

    return {
        'Decision Tree Classifier': dtc,
        'Neural Networks': nn,
        'Support Vector Machines': svc,
        'Random Forest Classification': rfc,
        'k Nearest Neighbours': knc_map,
        'Gradient Boosting Classifier': gbc_map
    }

def test_models(test_data, models):
    print("Prediction rating:\n")
    for model in models:
        prediction = models[model].score(test_data['data'], test_data['result'])*100.00
        print(str(model) + ": " + "%.2f" % prediction + "%")


initializeSystem()
```

```
training_data = loadFile("../data/training_data_small.csv")
trainDataFeaturesReviews = pd.DataFrame(training_data, columns=["review", "rating"])
targetRating = (trainDataFeaturesReviews['rating'])
targetReview = trainDataFeaturesReviews['review']
trainReviews = convertReviews(targetReview)
decisionAttributes = getDecisionAttributes(trainReviews)
trainSparseMatrix = prepareSparseMatrix(trainReviews, decisionAttributes)
dataFeatures = pd.DataFrame(trainSparseMatrix, columns=decisionAttributes)
training_data = {
    'data': dataFeatures,
    'result': targetRating
}

test_data = loadFile("../data/test_data_small.csv")
testDataFeaturesReviews = pd.DataFrame(test_data, columns=["review", "rating"])
testReview = testDataFeaturesReviews['review']
testRating = testDataFeaturesReviews['rating']
testSparseMatrix = prepareSparseMatrix(convertReviews(testReview), decisionAttributes)
testDataFeatures = pd.DataFrame(testSparseMatrix, columns=decisionAttributes)
test_data = {
    'data': testDataFeatures,
    'result': testRating
}

models = model_data(training_data)
test_models(test_data, models)
```

## grapher.py

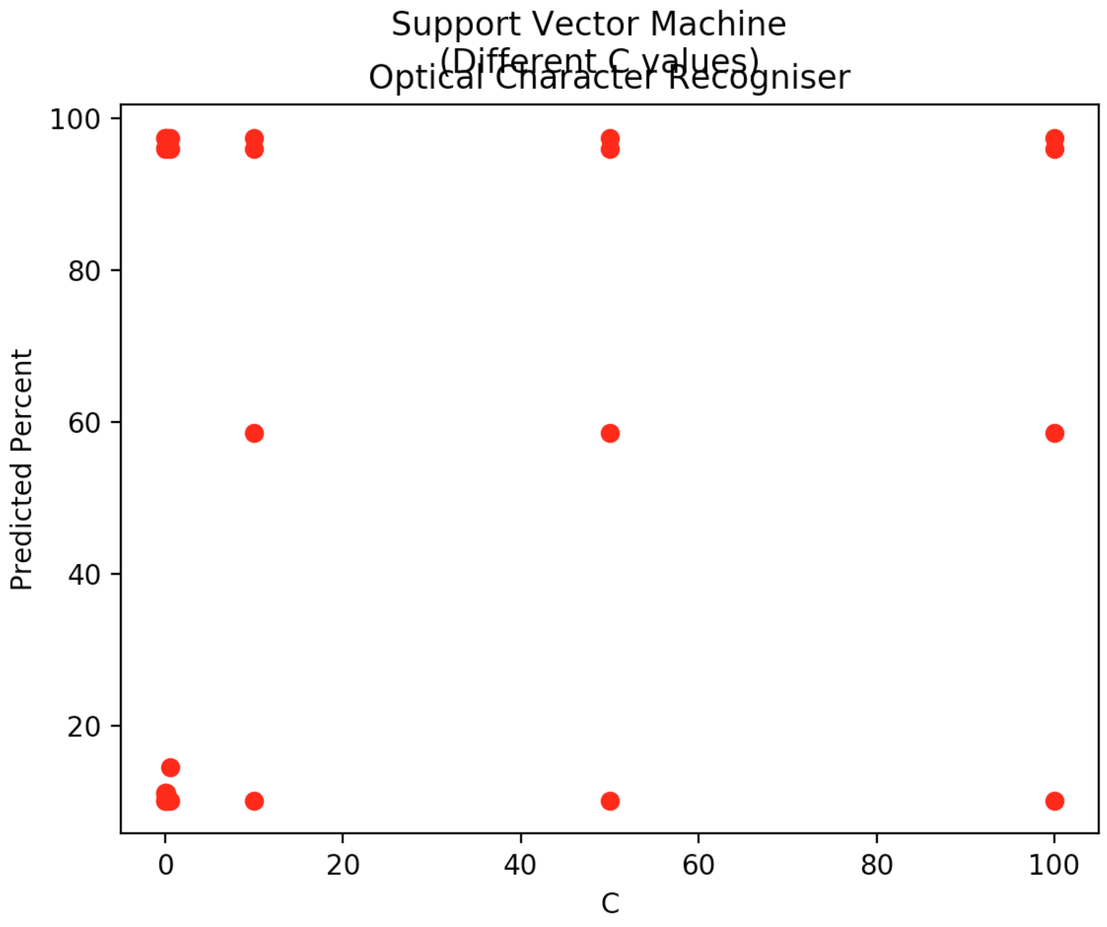
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


c = [0.01, 0.05, 0.5, 0.1, 10, 50, 100, 0.01, 0.05, 0.5, 0.1, 10, 50, 100, 0.01, 0.05, 0.5, 0.1, 10, 50,
100, 0.01, 0.05, 0.5, 0.1, 10, 50, 100]
y = [11.18530885, 11.18530885, 14.57985531, 11.18530885, 58.59766277, 58.59766277,
58.59766277, 97.38452977, 97.38452977, 97.38452977, 97.38452977, 97.38452977,
97.38452977, 97.38452977, 10.1836394, 10.1836394, 10.1836394, 10.1836394, 10.1836394,
10.1836394, 10.1836394, 95.9933222, 96.10461881, 95.9933222, 96.04897051, 95.9933222,
95.9933222, 95.9933222]

j = plt.figure(1)
j.suptitle('Support Vector Machine \n (Different C values)')
```

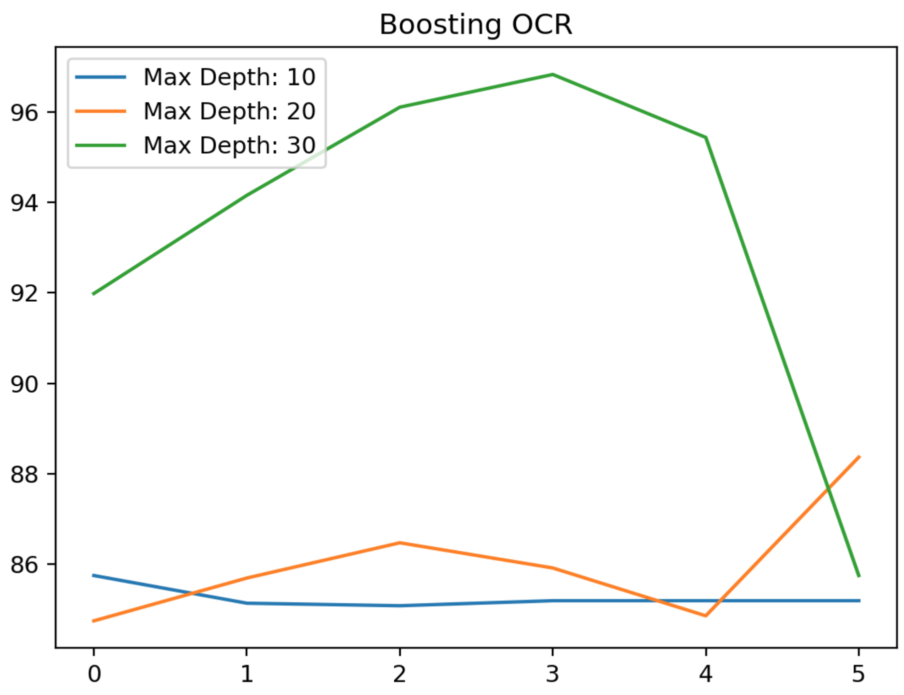
```
plt.title('Optical Character Recogniser')
plt.plot(c, y, 'ro')
plt.xlabel('C')
plt.ylabel('Predicted Percent')
plt.show()


y1 = [58.19672131, 58.19672131, 58.19672131, 58.19672131]
y2 = [58.19672131, 58.19672131, 59.83606557, 59.01639344]
y3 = [58.19672131, 58.19672131, 58.19672131, 58.19672131]
y4 = [60.6557377, 77.04918033, 93.44262295, 87.70491803]
z1 = [0.01, 0.05, 0.5, 0.1]
z2 = [0.01, 0.05, 0.5, 0.1]
z3 = [0.01, 0.05, 0.5, 0.1]
z4 = [0.01, 0.05, 0.5, 0.1]
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(y1, z1, label='RBF')
ax.plot(y2, z2, label='Poly')
ax.plot(y3, z3, label='Sigmoid')
ax.plot(y4, z4, label='Linear')
plt.title('Product Review Analysis')
ax.set_xlabel('Kernel', )
ax.set_ylabel('Predicted Percent')
plt.legend(loc=2)
plt.show()


x1 = [85.7540345, 85.14190317, 85.08625487, 85.19755147, 85.19755147, 85.19755147]
x2 = [84.75236505, 85.6983862, 86.47746244, 85.92097941, 84.86366166, 88.36950473]
x3 = [91.98664441, 94.15692821, 96.10461881, 96.82804674, 95.43683918, 85.7540345]
plt.figure()
plt.title('Boosting OCR')
plt.plot(x1, label='Max Depth: 10')
plt.plot(x2, label='Max Depth: 20')
plt.plot(x3, label='Max Depth: 30')
plt.legend(loc=2)
plt.show()

x1 = [0.01, 0.05, 0.1, 0.5, 1, 1.5]
x2 = [0.01, 0.05, 0.1, 0.5, 1, 1.5]
x3 = [0.01, 0.05, 0.1, 0.5, 1, 1.5]
y1 = [84.42622951, 90.16393443, 90.98360656, 91.80327869, 91.80327869, 91.80327869]
y2 = [91.80327869, 92.62295082, 92.62295082, 90.16393443, 90.98360656, 90.98360656]
```

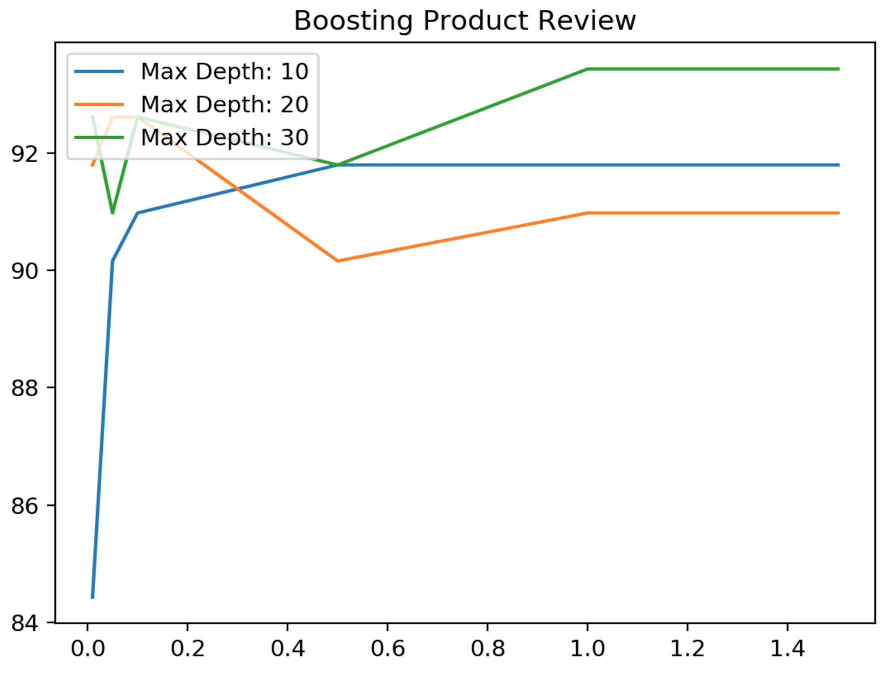
```
y3 = [92.62295082, 90.98360656, 92.62295082, 91.80327869, 93.44262295, 93.44262295]
plt.figure()
plt.title('Boosting Product Review')
plt.plot(x1, y1, label='Max Depth: 10')
plt.plot(x2, y2, label='Max Depth: 20')
plt.plot(x3, y3, label='Max Depth: 30')
plt.legend(loc=2)
plt.show()

x = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
y = [92.62295082, 92.62295082, 92.62295082, 92.62295082, 92.62295082, 92.62295082,
92.62295082, 92.62295082, 92.62295082, 92.62295082]
plt.title('KNN Product Review')
plt.plot(x, y)
plt.show()

x = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
y = [97.82971619, 97.10628826, 96.82804674, 96.60545353, 96.38286032, 95.8263773,
95.38119087, 94.88035615, 94.54646633, 94.54646633]
plt.title('KNN OCR')
plt.plot(x, y)
plt.show()

x = [1, 2, 3, 4, 5, 6]
y = [79, 64.86, 75.57, 65.95, 77.62, 76.62]
l = ['Decision Tree Classifier', 'Neural Networks', 'Support Vector Machines', 'Random Forest
Classification',\
    'k Nearest Neighbours', 'Gradient Boosting Classifier']
plt.xticks(x, l)
plt.title("Algorithms Vs Accuracy")
plt.xlabel('Algorithm')
plt.ylabel('Accuracy')
plt.plot(x, y)
for i, j in zip(x, y):
    plt.annotate(str(j), xy=(i,j))
plt.show()
```

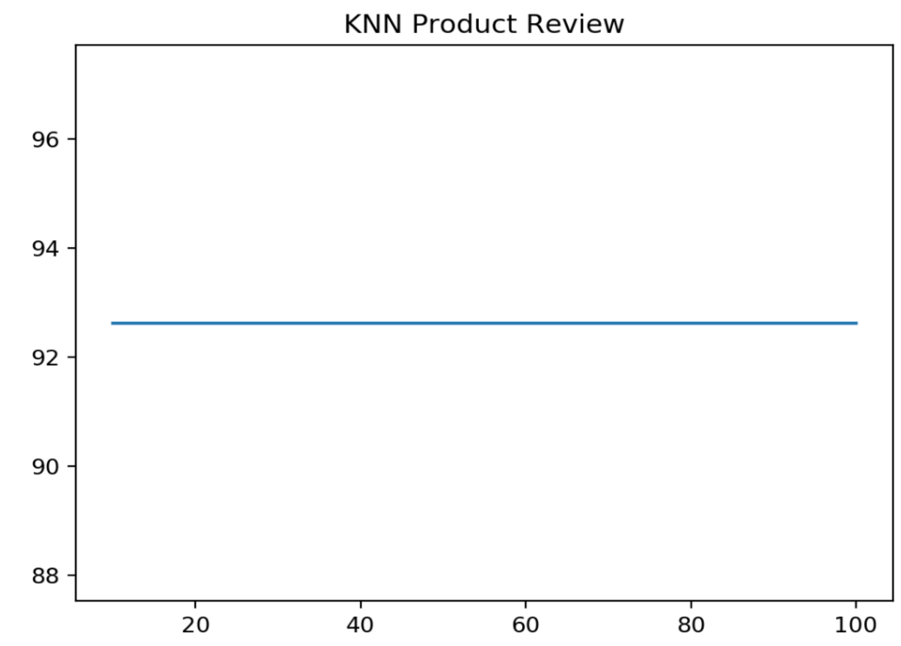
## Results:

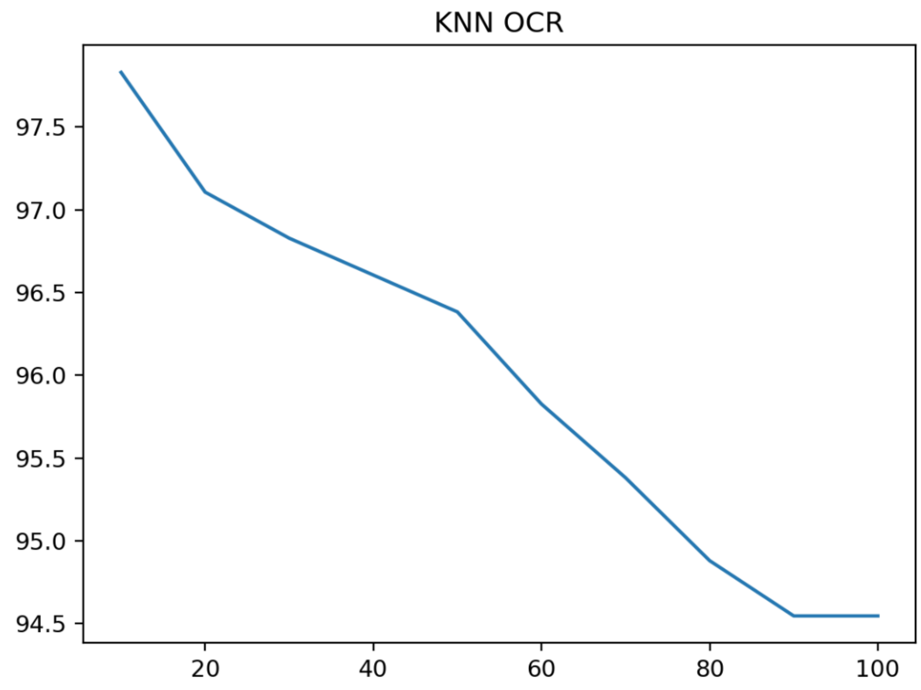
### Experiments and Observations

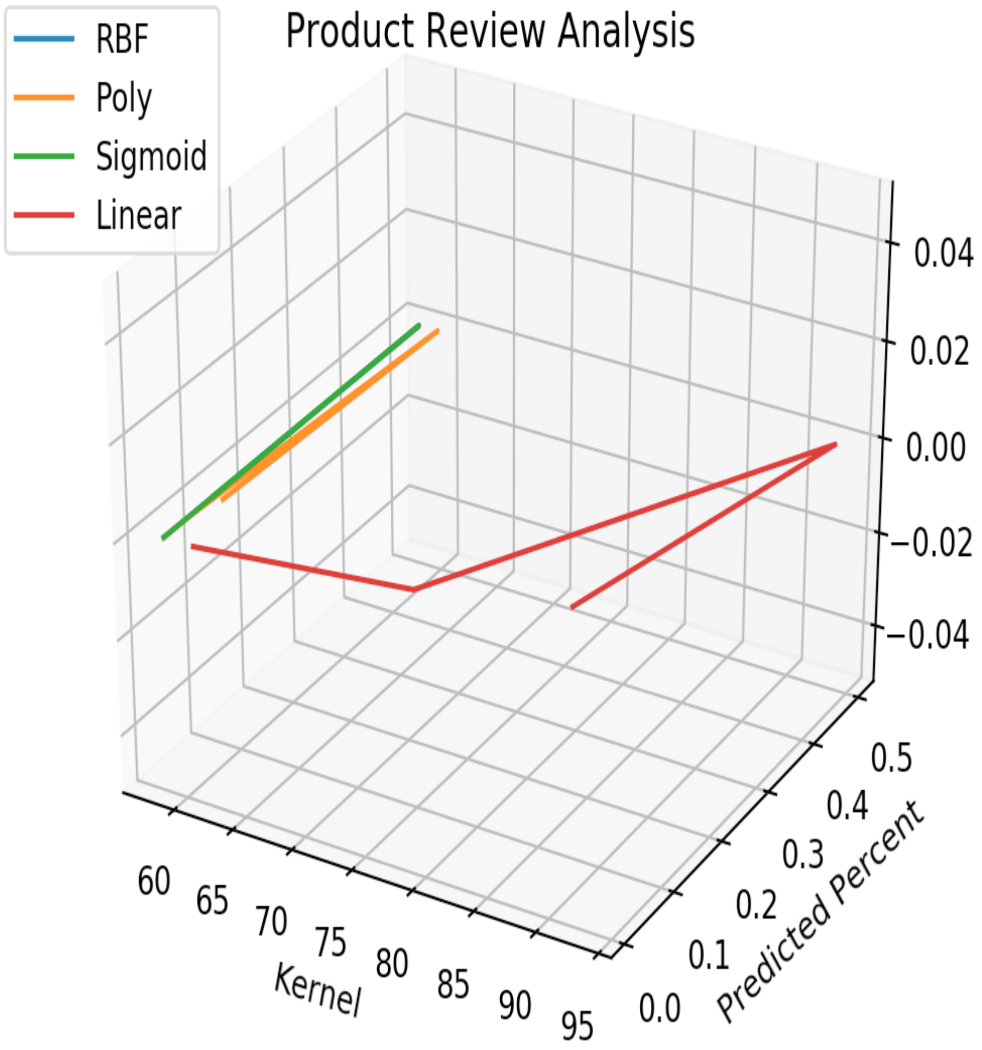



Boosting OCR
Max Depth: 10
Max Depth: 20
Max Depth: 30
96
94
92
90
88
86
0
1
2
3
4
5


Boosting Product Review
Max Depth: 10
Max Depth: 20
Max Depth: 30
92
90
88
86
84
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

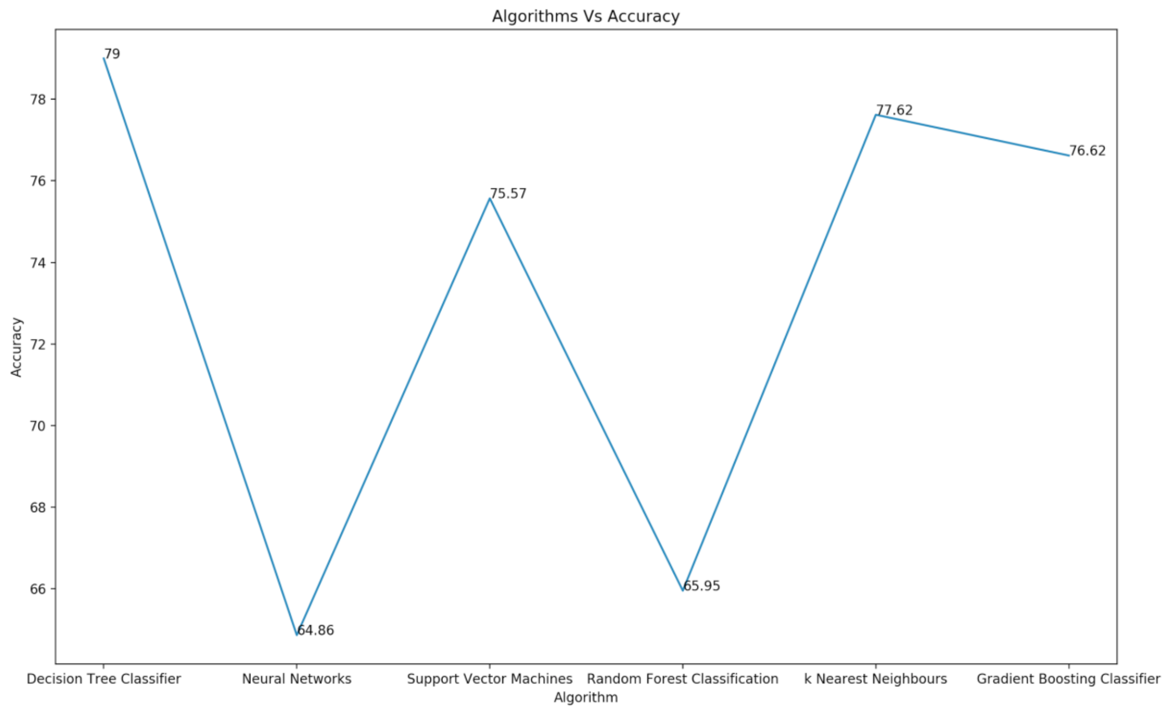

KNN Product Review
96
94
92
90
88
20
40
60
80
100


KNN OCR
97.5
97.0
96.5
96.0
95.5
95.0
94.5
20
40
60
80
100



Product Review Analysis
RBF
Poly
Sigmoid
Linear
0.04
0.02
0.00
−0.02
−0.04
60
65
70
75
80
85
90
95
Kernel
0.0
0.1
0.2
0.3
0.4
0.5
Predicted Percent

## Comparisons of different Models



## 8. CONCLUSION:

We have developed some explanatory models to predict the helpfulness of Amazon Fine Food Reviews using different Machine Learning algorithms to see which of the methods predicts with a best accuracy. This dataset comes from over 568,0454 Amazon Fine Food Reviews. This type of analysis will improve the company's selection of helpful reviews at the top of the review section and improve customer's purchasing decisions. It could also help other reviewers as a guide to writing helpful reviews. Further we can try to implement different algorithms and techniques to see if we can get a better prediction and also try using different EDA techniques and data cleansing techniques to refine the data in a better way before analysis using Machine algorithms so that we may get a better accuracy.

## 9. REFERENCES:

- https://www.kaggle.com/snap/amazon-fine-food-reviews
- http://cs229.stanford.edu/proj2011/MehtaPhilipScaria-Predicting%20Star%20Ratings%20from%20Movie%20Review%20Comments.pdf
- http://marcobonzanini.com/