Project uses multiple models to get the best accuracy

Project by Anmol Sharma

In [1]:
```
import random
import json
from sklearn.model_selection import train_test_split
```

Creating various classes to optimise and classify data to our needs

In [2]:
```
class Review:
    def __init__(self, text, score):
        self.text = text
        self.score = score
        self.sentiment = self.get_sentiment()

    def get_sentiment(self):
        if self.score <= 2:
            return "NEGATIVE"
        elif self.score == 3:
            return "NEUTRAL"
        if self.score >= 3:
            return "POSITIVE"
```

In [3]:
```
class ReviewContainer:
    def __init__(self, rev):
        self.rev = rev

    def evenly_distribute(self):
        negative = list(filter(lambda x: x.sentiment == "NEGATIVE", self.rev))
        positive = list(filter(lambda x: x.sentiment == "POSITIVE", self.rev))
        positive_shrunk = positive[:len(negative)]
        self.rev = positive_shrunk + negative
        random.shuffle(self.rev)
```

Importing the reviews csv and splitting data evenly as positive and negative (to train optimally)

In [4]:
```
file_name = 'c:/Users/Lenovo/PycharmProjects/Giraffe/Books_small_10000.json'
reviews = []
with open(file_name) as f:
    for line in f:
        review = json.loads(line)
        reviews.append(Review(review['reviewText'], review['overall']))

training, test = train_test_split(reviews, test_size=0.33, random_state=42)

train_cont = ReviewContainer(training)
test_cont = ReviewContainer(test)

train_cont.evenly_distribute()
test_cont.evenly_distribute()
```

```
train_x = [x.text for x in train_cont.rev]
train_y = [x.sentiment for x in train_cont.rev]

test_x = [x.text for x in test_cont.rev]
test_y = [x.sentiment for x in test_cont.rev]

print(test_y.count("POSITIVE"), test_y.count("NEGATIVE"))
```

```
208 208
```

In [5]:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Tfidfvectoriser is better as it ignores the words that are excessively used like was is,
# Term Frequency Inverse Document Frequency Vectoriser

# vec = CountVectorizer()
vec = TfidfVectorizer()
train_x_vecs = vec.fit_transform(train_x)
test_x_vecs = vec.transform(test_x)
```

Classifiers:

SVM

In [6]:

```
from sklearn import svm

clf_svm = svm.SVC(kernel='rbf', C=4)
clf_svm.fit(train_x_vecs, train_y)

print(clf_svm.predict(test_x_vecs[0]))
```

```
['POSITIVE']
```

Decision tree

In [7]:

```
from sklearn.tree import DecisionTreeClassifier

clf_dec = DecisionTreeClassifier()
clf_dec.fit(train_x_vecs, train_y)

print(clf_svm.predict(test_x_vecs[0]))
```

```
['POSITIVE']
```

Checking the score for each models

In [8]:

```
print(clf_svm.score(test_x_vecs, test_y))
print(clf_dec.score(test_x_vecs, test_y))
```

```
0.8197115384615384
0.6322115384615384
```

In [9]:

```
# F1 scores, which are more important

from sklearn.metrics import f1_score

print(f1_score(test_y, clf_svm.predict(test_x_vecs), average=None, labels=['POSITIVE', 'NE
test_set = ["Very high quality product, loved it.", "Horrible, doesn't work, waste of mone
            "Excellent quality, suggested to buy."]
new_test = vec.transform(test_set)
print(clf_svm.predict(new_test))
```

```
[0.82269504 0.81662592]
['POSITIVE' 'NEGATIVE' 'POSITIVE']
```

Tuning the model (with grid search)

In [10]:

```
from sklearn.model_selection import GridSearchCV

parameters = {'kernel': ('linear', 'rbf'), 'C': (1, 4, 8, 16, 32)}

tuned_svm = svm.SVC()

# cv is for how many times we want cross validation
clf = GridSearchCV(tuned_svm, parameters, cv=5)
clf.fit(train_x_vecs, train_y)
res = clf.best_params_
print(res)
```

```
{'C': 1, 'kernel': 'linear'}
```

Saving the model

In [11]:

```
import pickle

with open('c:/Users/Lenovo/PycharmProjects/Giraffe//models/sentiment_classifier.pkl', 'wb'
    pickle.dump(clf, f)

# loading the mmoel

with open('c:/Users/Lenovo/PycharmProjects/Giraffe//models/sentiment_classifier.pkl', 'rb'
    loaded_clf = pickle.load(f)
```

Final test

In [12]:

```
print(test_x[1])
print(loaded_clf.predict(test_x_vecs[1]))
```

```
I am forced to write twenty words, and don't have twenty words to say about these books. A
ll I have to say is, loved the movie, but what I've read of these so far is mind numbingly
boring.
['POSITIVE']
```