Predicting Movie genre using NLP

Navsimran Singh (1565066)

Electrical and Computer Engineering, University of Alberta

navsimra@ualberta.ca

Introduction-The project is concentrated on predicting the movie genres using NLP. It uses NLP to analyze the Data,word2 vector model to convert data into vector using genism(**Gensim** is an open-source library for unsupervised topic modeling and natural language processing, using modern statistical machine learning) and finally It uses various Statistical Models to predict the accuracy of the project.

➢ Dataset Details:

The open dataset is taken from the CMU Movie Summary Corpus.This dataset contains multiple files, but we have focused on only two of them for now:

- **Details of movie.metadata.tsv:**
  Metadata for 81,741 movies, extracted from the November 4, 2012 dump of Freebase. The movie genre tags are available in this file. It has a column with unique id for each movie. Then, another one for movie name. Also, there is a column containing the year on which the movie was released. Moreover, we have columns containing the total length of the movie, in which language it was released, the country of origin. Then, at last the dataset contains the column for genre of the movie.
- **Details of plot_summaries.txt:**
  Plot summaries of 42,306 movies extracted from the November 2, 2012 dump of English-language Wikipedia. Each line contains the Wikipedia movie ID (which indexes into movie.metadata.tsv) followed by the plot summary. It is a dataset composed of only two columns containing movie id and plot detail for each movie. The column, movie_id is unique and common between the two datasets. The plot description about the movie describes mostly about the movie that is which characters it has, is it a family movie or it has some comedy scenes etc. which in turn will help in identifying the genres of the movies upon the analysis.

➢ Data Exploration and Pre-processing

The quality of the inputs decides the quality of the output. So, once we have got our hypothesis ready, it makes sense to spend lot of time and efforts here. Approximately, data exploration, cleaning and preparation can take up to 70% of total project time.

Below are the steps that are used to understand, clean and prepare the data for building predictive model:

- **Variable Identification:**
  First, identify **Predictor** (Input) and **Target** (output) variables. Next, identify the data type and category of the variables. As per the data mentioned above, the input variable is taken as the ‘plot’ which is the description about each movie. That is based on the plot, a movie genre is predicted. Next, step is to identify the data type and category of the variables. The data type for genre and plot is character. Variable category is categorical and variable type for genre is target variable and for plot is predictor variable.

- **Missing value treatment:**
  Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification. The missing values decrease the accuracy. For example, in the dataset, the plots having missing values may be predicted with wrong genre if not treated beforehand. There may be multiple reasons why a dataset may have missing values such as

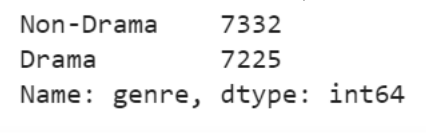
  ✓ **Deletion:** It is of two types: List Wise Deletion and Pair Wise Deletion.

  ✓ **In list wise deletion**, we delete observations where any of the variable is missing. Simplicity is one of the major advantages of this method, but this method reduces the power of model because it reduces the sample size.

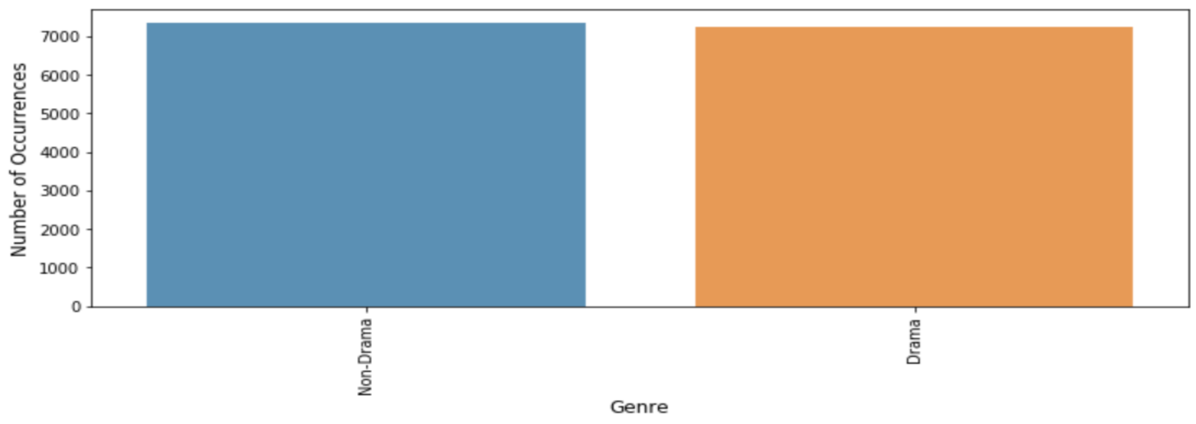
  ✓ **In pair wise deletion**, we perform analysis with all cases in which the variables of interest are present. Advantage of this method is, it keeps as many cases available for analysis. One of the disadvantages of this method, it uses different sample size for different variables. I have deleted some of the rows list wise which had incomplete information about any of the variables.

➢ **Final Dataset**-Final Dataset is the merger of the Movie and Plot dataset. We merge the data with the help of movie_id which is common in the both data set. After that our final dataset consist of movie_id, movie_name, year, genre and plot, we are concentrating on the plot and genre columns in our dataset.

➢ **Genre-Selection**- In the Final Dataset there are Different types of genres available like Comedy, Drama, Family-Film, Crime Fiction and Adventure. By analyzing the dataset, it seems that Drama has more data points as compared to other Genres so after that I categorized the data into Drama and Non drama (Changing -Comedy, Family-Film, Crime Fiction and Adventure into Non-Drama)



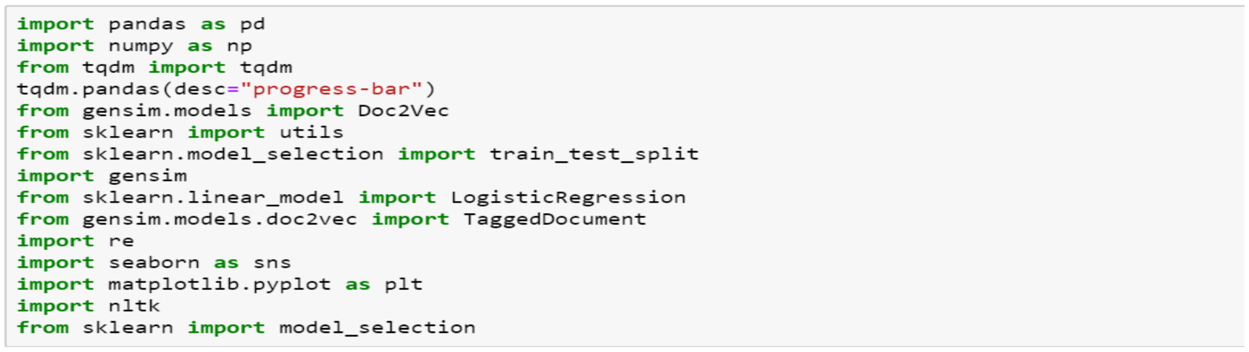
```
Non-Drama    7332
Drama        7225
Name: genre, dtype: int64
```



After applying all the steps on the dataset, we get the data which is ready to be fed as input to the data analysis algorithms.

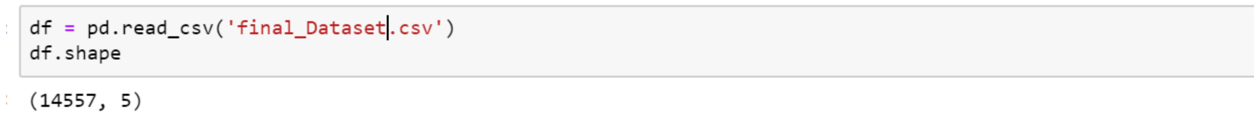
<u>Implementation:</u>

Step1.Importing all the Necessary libraries



```
import pandas as pd
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models import Doc2Vec
from sklearn import utils
from sklearn.model_selection import train_test_split
import gensim
from sklearn.linear_model import LogisticRegression
from gensim.models.doc2vec import TaggedDocument
import re
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn import model_selection
```

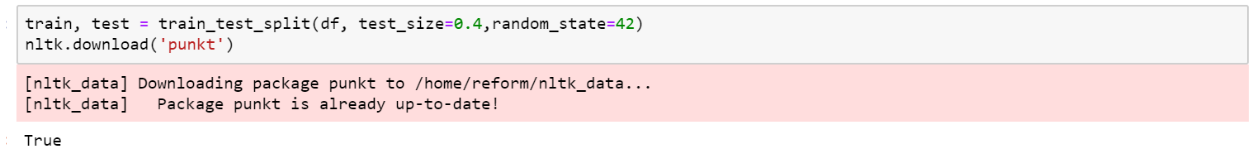
Step2.Loading the data



```
df = pd.read_csv('final_Dataset.csv')
df.shape
```

```
(14557, 5)
```

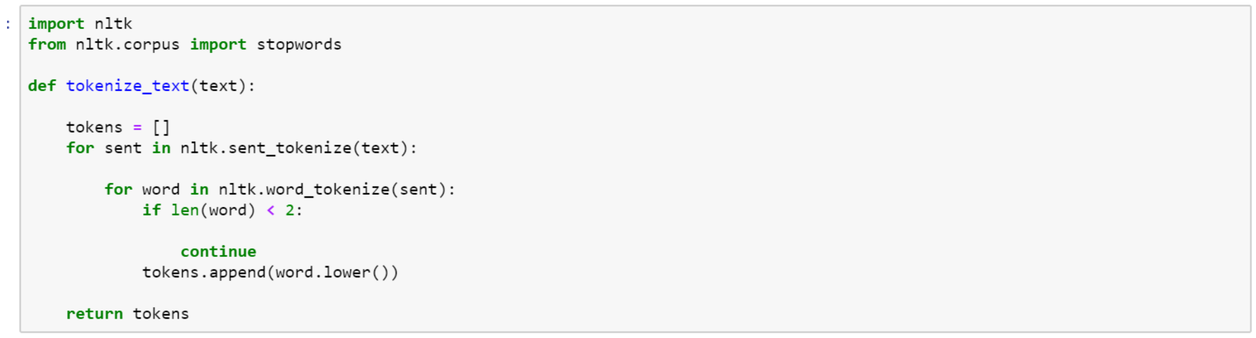
Step3. The following steps include train/test split of 60/40, remove stop-words and tokenize text using NLTK tokenizer.



```
train, test = train_test_split(df, test_size=0.4,random_state=42)
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /home/reform/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```



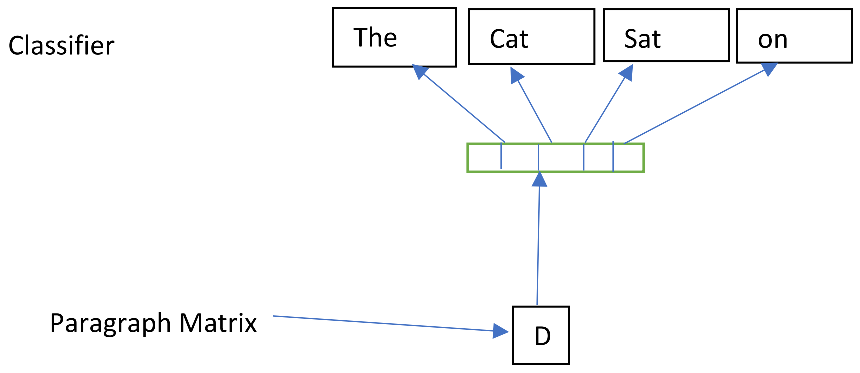
```
import nltk
from nltk.corpus import stopwords

def tokenize_text(text):

    tokens = []
    for sent in nltk.sent_tokenize(text):

        for word in nltk.word_tokenize(sent):
            if len(word) < 2:

                continue
            tokens.append(word.lower())

    return tokens
```

Step4.Setup Doc2Vec Training & Evaluation Models

First, we instantiate a doc2vec model- Distributed Bag of Words (DBOW).

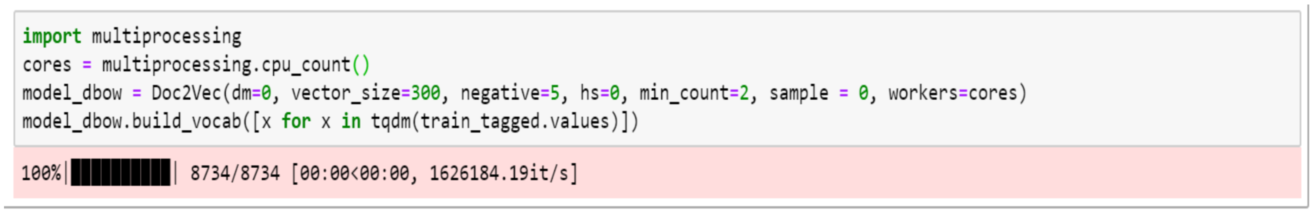
Distributed Bag of Words (DBOW)

DBOW is the doc2vec model analogous to Skip-gram model in word2vec. The DBOW model "ignores the context words in the input but force the model to predict words randomly sampled from the paragraph in the output". For Example, given a sentence "The cat sat on sofa" The model is learning by predicting 2 sampled words. So, in order to learn the document vector, two words are sampled from {the, cat, sat, on, the, sofa}, as shown in the diagram.



For the Plot, the paragraph vectors are obtained by training a neural network on the task of predicting a probability distribution of words in a paragraph given a randomly sampled word from the paragraph.

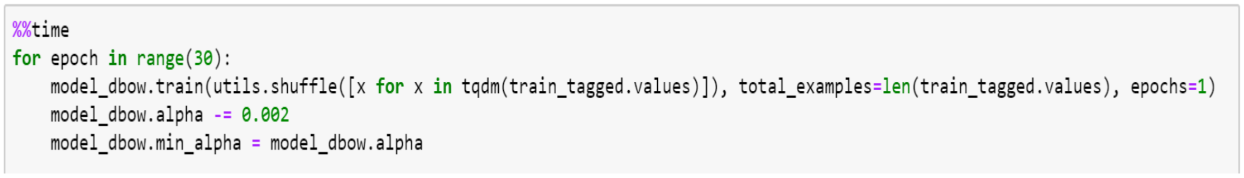
Building a Vocabulary- Distributed Bag of Words (DBOW) is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training and testing set.



```
import multiprocessing
cores = multiprocessing.cpu_count()
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0, min_count=2, sample = 0, workers=cores)
model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 8734/8734 [00:00<00:00, 1626184.19it/s]
```

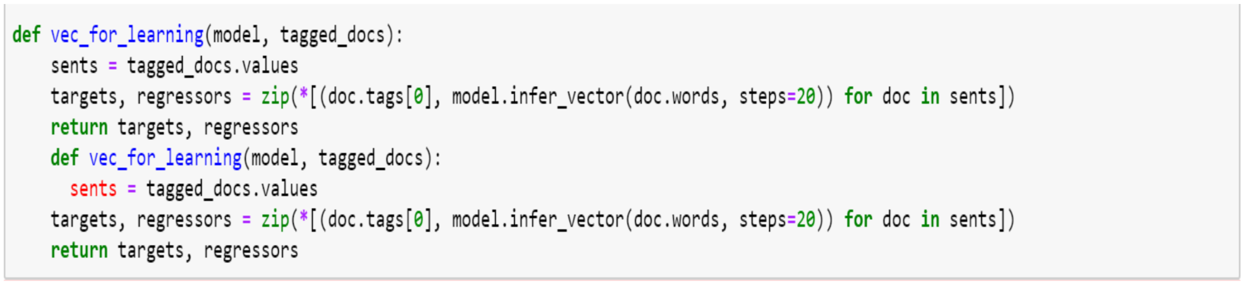
Step5.Training a doc2vec model-



```
%%time
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(train_tagged.values)]), total_examples=len(train_tagged.values), epochs=1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

Step6.Building the Final Vector Feature for the Classifier-

A feature is an individual measurable property or characteristic of a phenomenon being observed. Choosing informative, discriminating and independent features is a crucial step for effective algorithms in classification and regression. A set of numeric features can be conveniently described by a feature vector.



```
def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words, steps=20)) for doc in sents])
    return targets, regressors
    def vec_for_learning(model, tagged_docs):
      sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words, steps=20)) for doc in sents])
    return targets, regressors
```

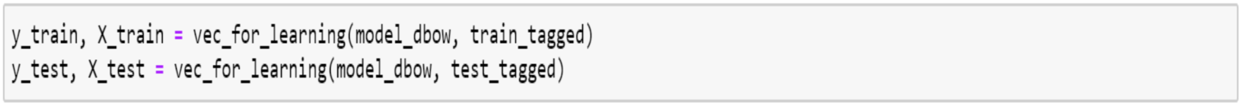
Step7.Prediction Using Different Models

1.**<u>Logistic Regression</u>**- Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest and set of independent variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a logit transformation of the probability of presence of the characteristic of interest

$\mathrm{logit}(p)=b_0+b_1X_1+ b_2X_2 +b_3X_3 \ldots\ldots+ b_KX_k$

Train the Logistic Regression Classifier



```
y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)
```

```
logreg = LogisticRegression(solver='newton-cg',n_jobs=20, C=1.0, class_weight='balanced')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
from sklearn.metrics import accuracy_score, f1_score
print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
print('Testing F1 score: {}'.format(f1_score(y_test, y_pred, average='weighted')))
```

```
Testing accuracy 0.5440494590417311
Testing F1 score: 0.5440283735696272
```

Step8.Concluding the results

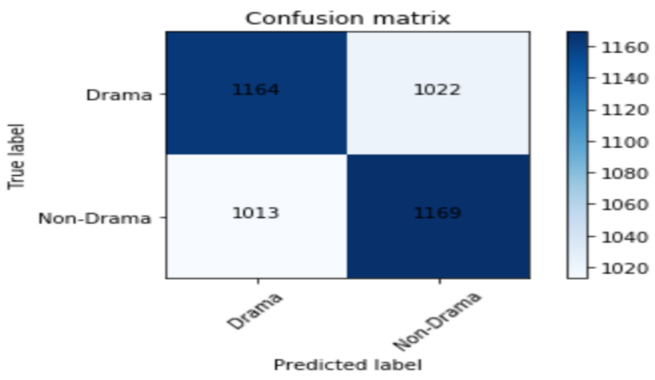
Confusion Matrix

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

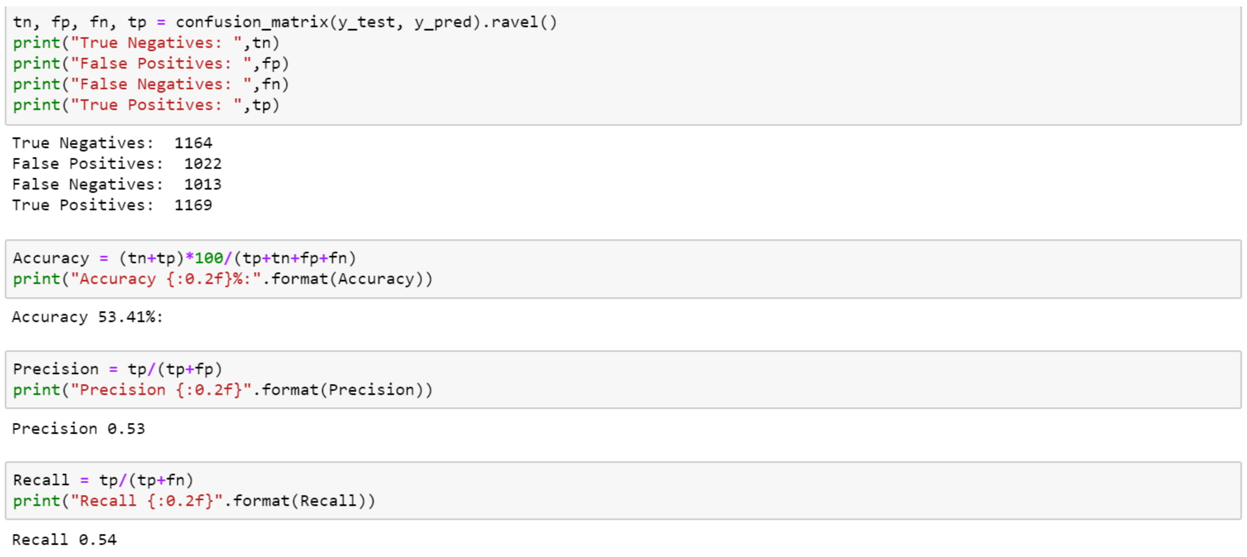
```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("confusion matrix")
    else:
        print('Confusion matrix')

    print(cm)
```

Output:

```
Confusion matrix
[[1164 1022]
 [1013 1169]]
```



Step9.Evaluation Metrics-



```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1164
False Positives:  1022
False Negatives:  1013
True Positives:  1169
```

```
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy {:0.2f}%:".format(Accuracy))
```

```
Accuracy 53.41%:
```

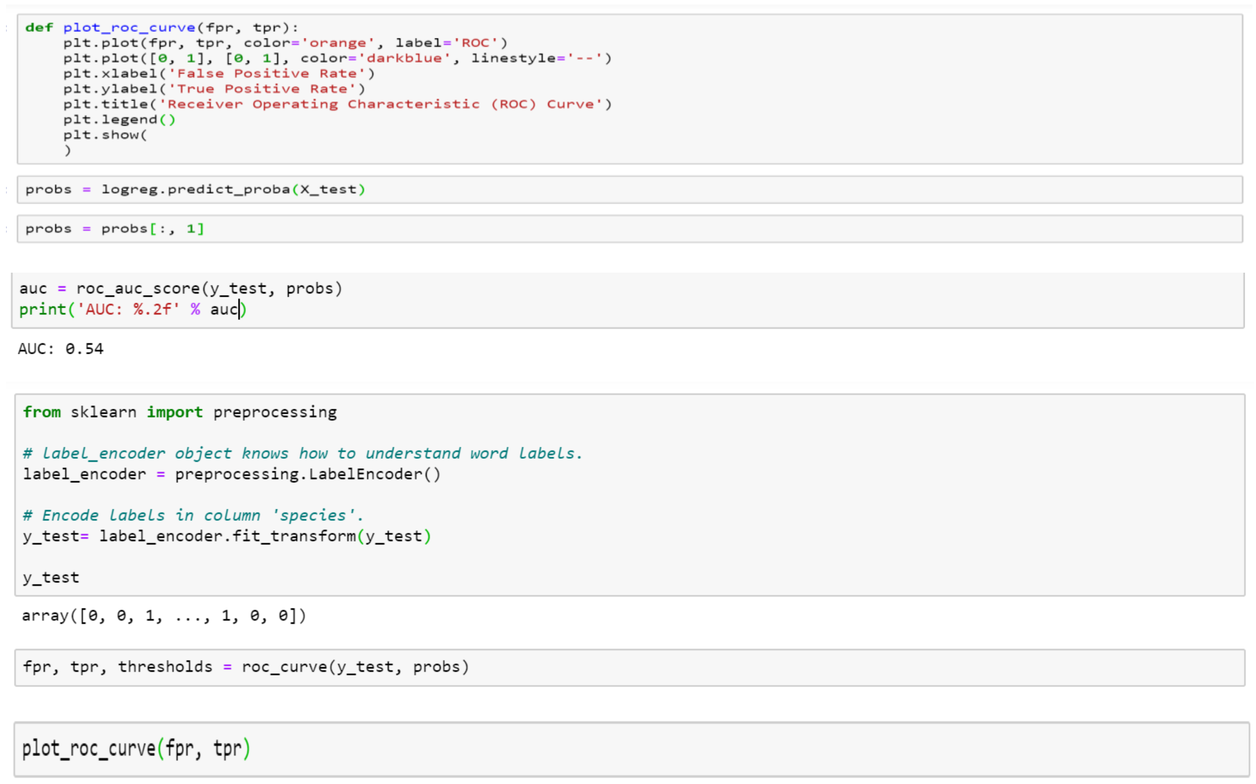
```
Precision = tp/(tp+fp)
print("Precision {:0.2f}".format(Precision))
```

```
Precision 0.53
```

```
Recall = tp/(tp+fn)
print("Recall {:0.2f}".format(Recall))
```

```
Recall 0.54
```

Step10.Plotting the Roc Curve



```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show(
    )
```

```
probs = logreg.predict_proba(X_test)
```

```
probs = probs[:, 1]
```

```
auc = roc_auc_score(y_test, probs)
print('AUC: %.2f' % auc)
```

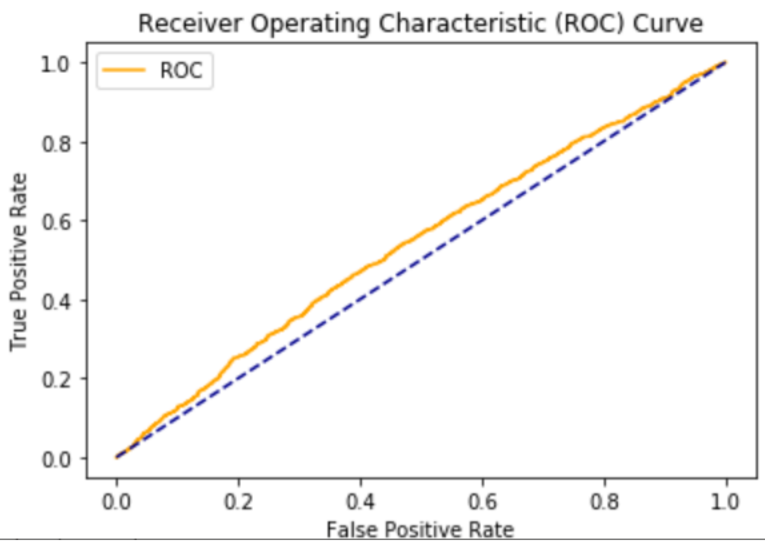
```
AUC: 0.54
```

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
y_test= label_encoder.fit_transform(y_test)

y_test
```

```
array([0, 0, 1, ..., 1, 0, 0])
```

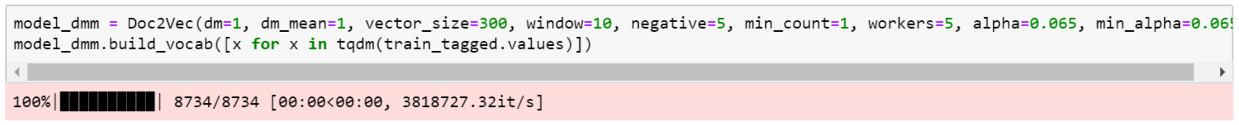
```
fpr, tpr, thresholds = roc_curve(y_test, probs)
```

```
plot_roc_curve(fpr, tpr)
```



To improve accuracy, the approach I use, is the distributed memory. **Distributed Memory** acts as a memory that remembers what is missing from the current context-or as a topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document. We again instantiate a Doc2Vec model with a vector size of 300 and iterating over the training corpus 300 times.

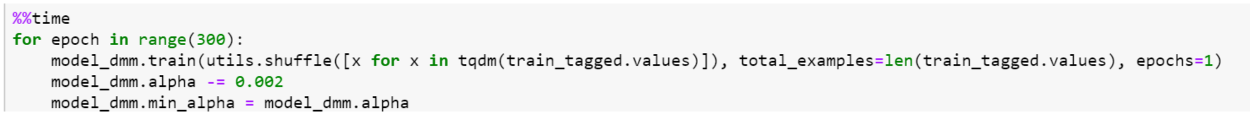
Step1.Building the Vocabulary.



```
model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10, negative=5, min_count=1, workers=5, alpha=0.065, min_alpha=0.06
model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])
```

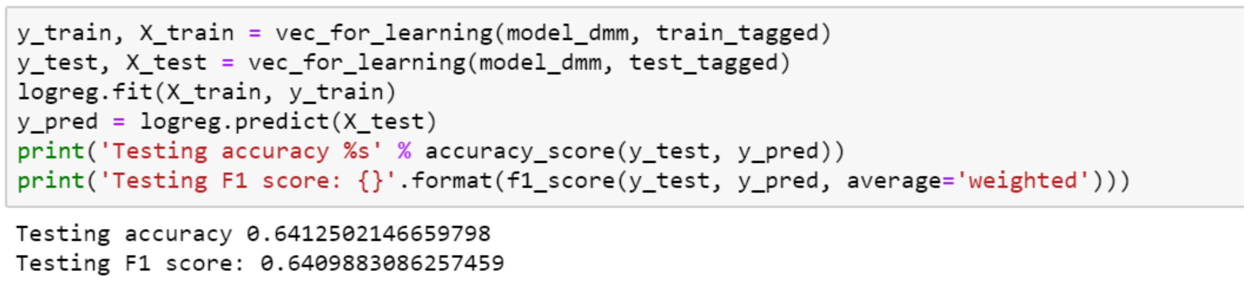
```
100%|██████████| 8734/8734 [00:00<00:00, 3818727.32it/s]
```

Step2.Training a doc2vec model-



```
%%time
for epoch in range(300):
    model_dmm.train(utils.shuffle([x for x in tqdm(train_tagged.values)]), total_examples=len(train_tagged.values), epochs=1)
    model_dmm.alpha -= 0.002
    model_dmm.min_alpha = model_dmm.alpha
```

Step3.Training the Logistic Regressor Classifier.

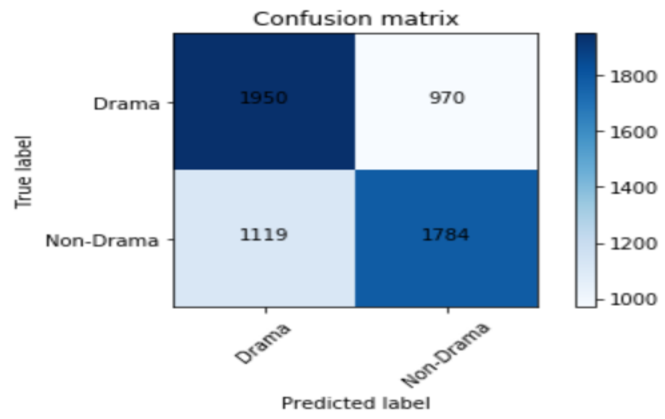


```
y_train, X_train = vec_for_learning(model_dmm, train_tagged)
y_test, X_test = vec_for_learning(model_dmm, test_tagged)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
print('Testing F1 score: {}'.format(f1_score(y_test, y_pred, average='weighted')))
```

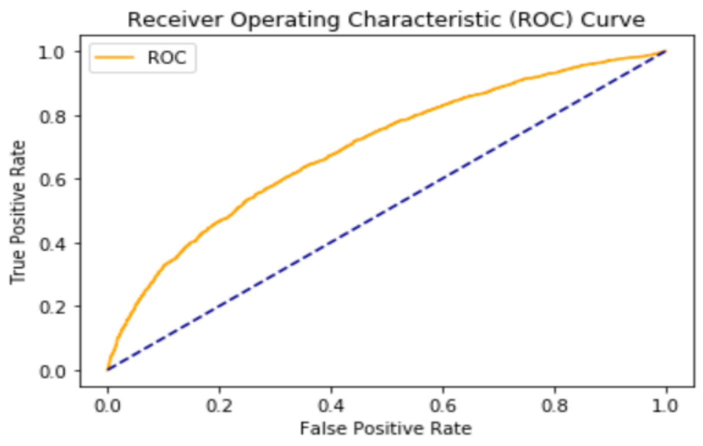
```
Testing accuracy 0.6412502146659798
Testing F1 score: 0.6409883086257459
```

Step4.Confusion Matrix.

```
Confusion matrix
[[1950  970]
 [1119 1784]]
```



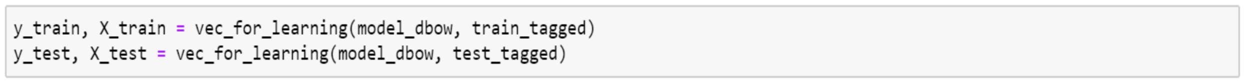
Step5.ROC Curve-



Second Model-Support Vector Machine-

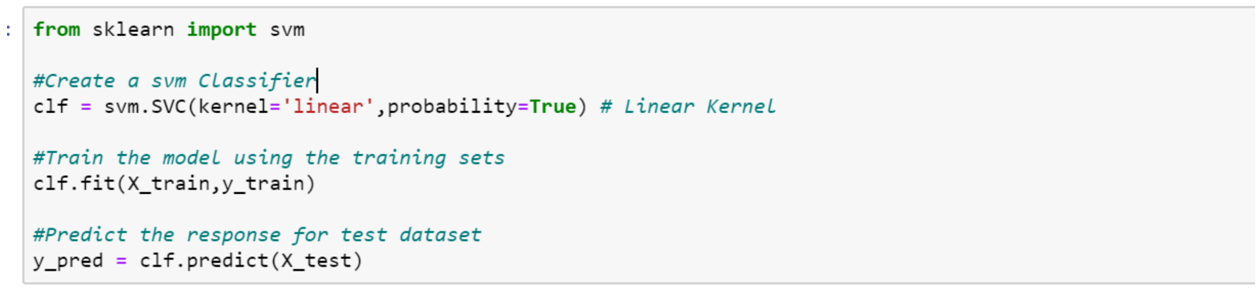
Support vector Machine is a supervised machine learning algorithm which can be used for both classification and regression. However, it is mostly used in the classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a coordinate.

Step1.Train the SVM Classifier-

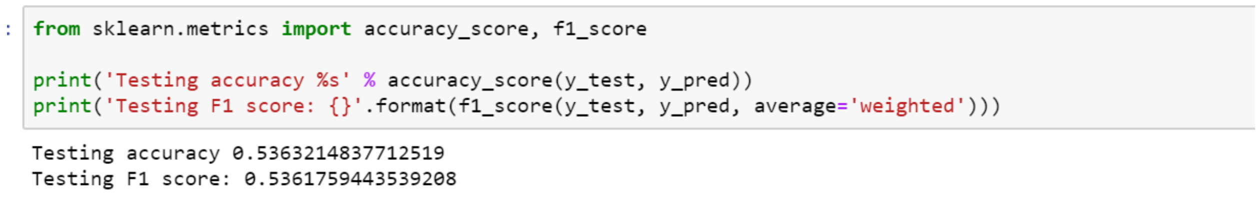


```
y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)
```

Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are many Features in a Data Set. One of the examples where there are a lot of features, is Text Classification, as each alphabet is a new feature. So, I use Linear Kernel as we have only two categories of genre. I also try to implement rbf Kernel, but it effects the accuracy of the model. Accuracy decreases up to 44% while using the rbf kernel



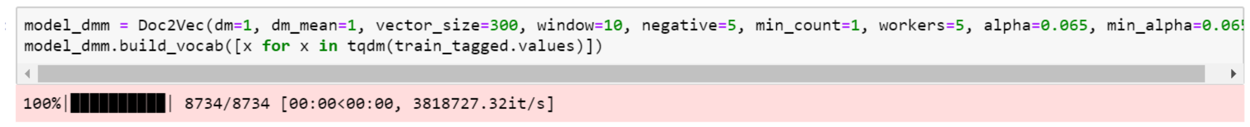
```
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear',probability=True) # Linear Kernel

#Train the model using the training sets
clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```



```
from sklearn.metrics import accuracy_score, f1_score

print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
print('Testing F1 score: {}'.format(f1_score(y_test, y_pred, average='weighted')))
```

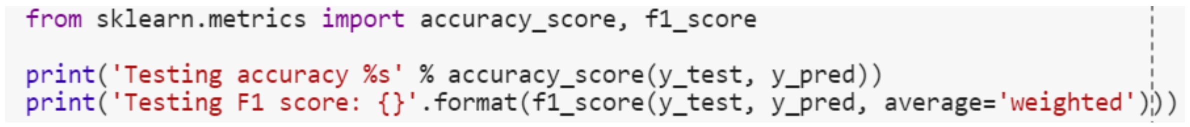
```
Testing accuracy 0.5363214837712519
Testing F1 score: 0.5361759443539208
```

Step2.With Distributed Memory-

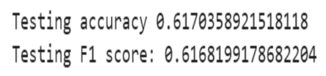


```
model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10, negative=5, min_count=1, workers=5, alpha=0.065, min_alpha=0.06
model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 8734/8734 [00:00<00:00, 3818727.32it/s]
```



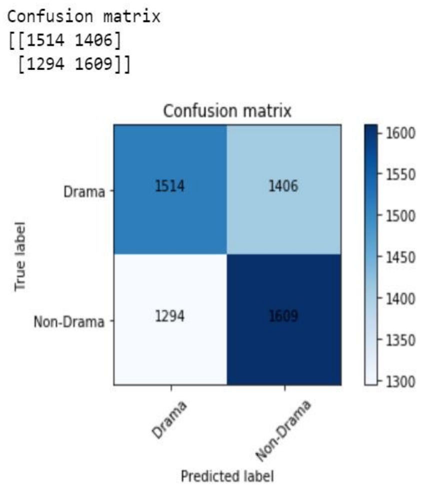
```
from sklearn.metrics import accuracy_score, f1_score

print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
print('Testing F1 score: {}'.format(f1_score(y_test, y_pred, average='weighted')))
```



```
Testing accuracy 0.6170358921518118
Testing F1 score: 0.6168199178682204
```

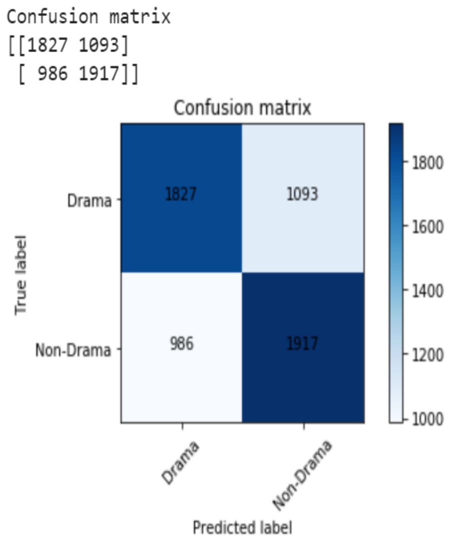
Step3.Confusion Matrix-

Without Distributed Memory



```
Confusion matrix
[[1514 1406]
 [1294 1609]]
```

With Distributed Memory

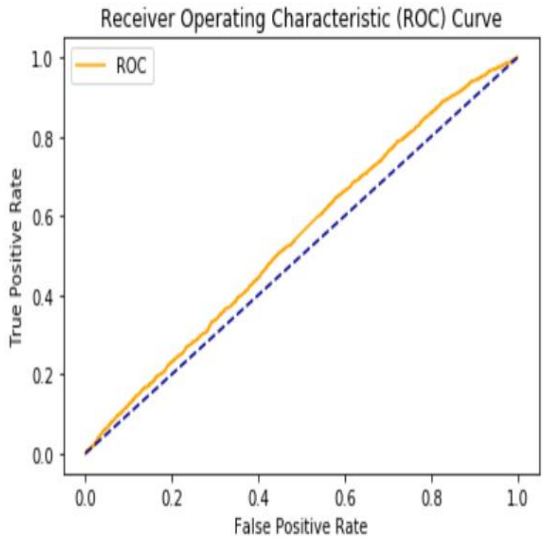


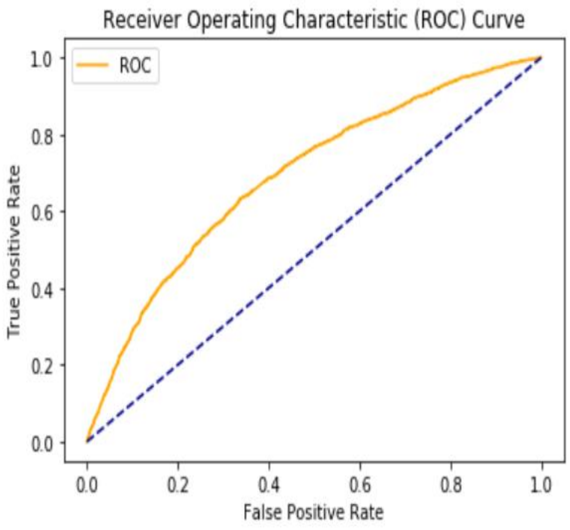
```
Confusion matrix
[[1827 1093]
 [ 986 1917]]
```

Step4.ROC Curve-

Without Distributed Memory

With Distributed Memory





<u>Conclusion</u>-Accuracy with Logistic regression is more as compared to SVM.Without the use of Distributed memory Accuracy is quite Decent with the both algorithms. While applying the distributed memory accuracy made a quite jump up to 10 percent in case of Logistic Regression and 8 percent in case of SVM.Finally, Logistic Regression model able to predict the output with the accuracy of 65 percent.