# Gender and Character Classifier

In this task, we were giving two files which contained EastEnders script one named “training.csv” and another which contained test data named “test.csv”. The files contained 3 different columns which were:

- Lines (The lines of each character)
- Character
- Gender

So, there were two tasks that needed to be classified:

1. Classifier to be able to predict the gender based on the lines (Not using character label)
2. Classifier to be able to predict the character based on the line (Not using gender label)

For my task I decided two separate the tasks while they are very similar,  decided to organize it in two files.

## Tools

- **import pandas as pd** : pandas is used to read files and a general library for python
- **import numpy as np :** For doing maths operations and creating a numpy array
- **from sklearn.model_selection import train_test_split** :  Used for splitting training data into validation and training set

**Used for making TF-IDF vector**

- from sklearn.feature_extraction.text import TfidfVectorizer
- from sklearn.feature_extraction.text import CountVectorizer

**used for computing precision_recall_fscore**

- from sklearn.metrics import precision_recall_fscore_support

**used for computing accuracy**

- from sklearn.metrics import accuracy_score

**Random Forest classifier**

- from sklearn.ensemble import RandomForestClassifier

**SVM classifier**

- from sklearn.svm import SVC

**Compute F1-score,precision and recall**

- from sklearn.metrics import f1_score
- from sklearn.metrics import precision_score
- from sklearn.metrics import recall_score

**NLTK lib for word tokenization and other pre-processing**

- from nltk import word_tokenize,WordLemmanizer
- import nltk
- import re

## Features

As with any classification problem, feature extraction and selection is one of the most important steps to creating a accurate classifier. With NLP, this is no exception. Throughout the course we had learned many ways of extracting features from text. In principle, what you have to do is convert the words to number for the classification to use. The values and representation of these values (weights) are what make a difference in your accuracy in NLP. This is why I have decided to use these features on my classification algorithm:

- **Bigrams**

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

As I had used bigrams previously in my lab 1, I felt that the use of it again would be very effective since the context word and adjacent words can be helpful and specific to each character. This can help both gender and character classification.

- **Unigrams**

Like bigrams, unigrams take in just the one word and the frequency, this was a great starting point from me, and it was easy to implement as a base for features. You can see the difference in my results in the table at the end where I experiment with different features.

- **POS Tags**

POS Tags use a tag set to and assigns the tag set label to each token within a corpus. You can see in my data that I basically go through the corpus and assign a POS tag to each token. Different characters can have different categorical sentence structures and therefore I tried to use

- **TF-IDF**

Within any text classification problem, TF-IDF is such a common feature to use mainly because of the effectiveness of it. In my classifiers, I apply Term frequency – Inverse document frequency where document is each line and I apply the “ngram_value” to get the TF-IDF of bigrams and unigrams within my code.

## Pre-processing

As pre-processing is an important step, I decided to use these steps to clean and pre-process my data:

- **Stop word removal:** Removing stop words like you've could affect the accuracy of my data and therefore its better to get a classifier without stop words in my training data, this had improved my classifier.
- **Lemmatisation:** This turns words into their original format which makes it much easier to parse through and it improves the accuracy on bigrams and unigrams. Note that I have only applied it to verbs.
- **Regular expression:** I used the regular expression '['+x+']' which filters special characters which I found was quite frequent in the lines of the script.
- **Lower casing:** making all words lower case will improve the prediction model based on certain context. As I am using n-gram features, I don't want my model treating words nouns as the starting point of a sentence, so I used lower casing as part of my pre-processing steps.

**Word Embedding / Text Numeric Representation (Features)**

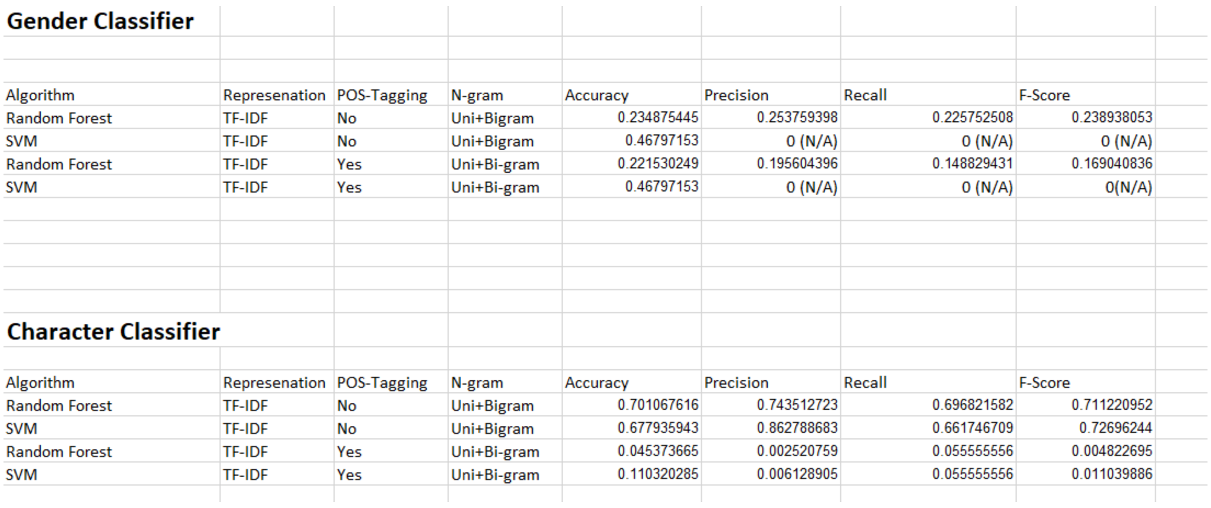
For representation of my data in numeric form, I decided to go with TF-IDF in the testing as this yielded the best results from my training data. At first it was the easiest to implement to represent my feature set, but I did experiment with word2vec in my training data, however I found that it resulted in my precision and accuracy being lower, I have left this in the code and it is tagged on the method I did it with.

For word2vec, I used the genism library and simply trained my word2vec model on the corpus of the training data. This is most likely the reason the accuracy is so low because I would probably need to train my model on a very large corpus set (More than the 10,000 lines) in order to get higher accuracies, but for the nature of this task, I was not able to do so.

## Classifiers

In my program, I decided to start with a simple SVM (Support Vector Machine) as I started with the gender classification task. From then on after some research, I decided to incorporate a Random forest classifier. The way my classifier was configured was that I split my training data so that my parameters can be fine tuned on at least 33% validation.

I decided to split it and test my different classifiers with different baselines to see my most accurate results, after this evaluation, I decided on which features were the best and which classifier was the best (after various baselines). You can see the scores here below (next page)



**Gender Classifier**

| Algorithm | Represenation | POS-Tagging | N-gram | Accuracy | Precision | Recall | F-Score |
|---|---|---|---|---|---|---|---|
| Random Forest | TF-IDF | No | Uni+Bigram | 0.234875445 | 0.253759398 | 0.225752508 | 0.238938053 |
| SVM | TF-IDF | No | Uni+Bigram | 0.46797153 | 0 (N/A) | 0 (N/A) | 0 (N/A) |
| Random Forest | TF-IDF | Yes | Uni+Bi-gram | 0.221530249 | 0.195604396 | 0.148829431 | 0.169040836 |
| SVM | TF-IDF | Yes | Uni+Bi-gram | 0.46797153 | 0 (N/A) | 0 (N/A) | 0(N/A) |

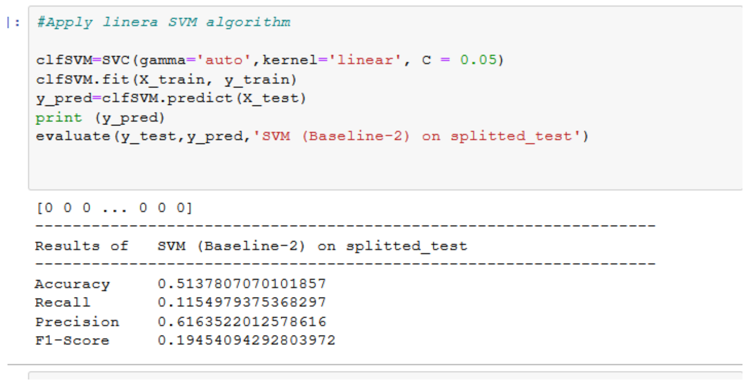
**Character Classifier**

| Algorithm | Represenation | POS-Tagging | N-gram | Accuracy | Precision | Recall | F-Score |
|---|---|---|---|---|---|---|---|
| Random Forest | TF-IDF | No | Uni+Bigram | 0.701067616 | 0.743512723 | 0.696821582 | 0.711220952 |
| SVM | TF-IDF | No | Uni+Bigram | 0.677935943 | 0.862788683 | 0.661746709 | 0.72696244 |
| Random Forest | TF-IDF | Yes | Uni+Bi-gram | 0.045373665 | 0.002520759 | 0.055555556 | 0.004822695 |
| SVM | TF-IDF | Yes | Uni+Bi-gram | 0.110320285 | 0.006128905 | 0.055555556 | 0.011039886 |

**Table Showing Score Comparison**

## Metrics

Within my code, I had 4 metrics which I used to measure how well my classifier was able to predict the correct labels. The most important metric must be my F1 Score. An instance of where this was extremely important was on my gender classification task here:



```
#Apply linera SVM algorithm

clfSVM=SVC(gamma='auto',kernel='linear', C = 0.05)
clfSVM.fit(X_train, y_train)
y_pred=clfSVM.predict(X_test)
print (y_pred)
evaluate(y_test,y_pred,'SVM (Baseline-2) on splitted_test')
```

```
[0 0 0 ... 0 0 0]
------------------------------------------------------------------
Results of   SVM (Baseline-2) on splitted_test
------------------------------------------------------------------
Accuracy     0.5137807070101857
Recall       0.1154979375368297
Precision    0.6163522012578616
F1-Score     0.19454094292803972
```

The accuracy score will always indicate 0.51 as half the split test data was showing labels 0 while this is true, it does not give an accurate indication of the results, that's why the F1-Score is 0.1945, which factors in that no labels were returned 50% of the time. The most important metric was to increase this F1 score.

## Improvement/conclusion

In conclusion, there is quite a lot of aspects of my classifier that could be improved. Because I started of quite basic and improved my classifier, if I had more time, I would experiment with more features such as Parse trees and dependency relationships. Also I could use pre-trained word2vec models that might give us better results. Character specific TD-IDF ,treating each character as a separate document, might give me better results.

I also split my training data to tune my parameters but I could in the future use cross-validation or hyper-tune my algorithm.  The most important focus was for me to not overfit or underfit, that is why I split my training data into 33% and did a random state. Overall  I am satisfied with the overall work.