Capstone 2 Project Report

Project: Classification of the gender based on Tweets(text) data

I. Introduction:

Sentiment analysis is famous among major brands. It is the way to identify the tone and emotions expressed through written or spoken online communication. It is famous with big companies like

- Twitter
- Amazon
- Facebook
- Netflix.

Based on the dataset we have obtained, there was some analysis which was done based out of the following questions.

- ➢ How well do words in tweets and profiles predict user gender?
- ➢ What are the words that strongly predict male or female gender?
- ➢ How well do stylistic factors (like link color and sidebar color) predict user gender?

The goal of this project is to simply view a Twitter profile and judge whether the user was a male, a female, or a brand (non-individual). NLP is used and its different methods pave the way for achieving a solution for performing the analysis. This is useful for the prospective client in determining a particular gender in analyzing well, based on seeing a user's tweet or a profile.

Here, I have used fastai, a deep learning library, which revolves around the concept of transfer learning, is very effective in predicting the accuracy of classification models using some interesting classes like TextClasDataBunch and a learning model TextLMDataBunch.

II. Dataset:

This data set was used to train a CrowdFlower AI gender predictor. The dataset contains 20,000 rows, each with a username, a random tweet, account profile and image, location, and even link and sidebar color.

Twitter User Gender Classification

Also, this is from the Kaggle competition. The csv file name is gender-classifier-DFE-791531.csv

III. Data Cleaning/Wrangling:

In this part, there have been some NaN's in the dataset. The text and description columns contain loads of information regarding a person's tweet and also, there are some other columns like retweet counts, sidebar link color etc., which might be useful in determining the gender.

Also, there are columns which were not much useful to us in gender prediction. Features like **profile_yn, profile_yn_gold, golden, trusted_judgements** were dropped from the sub DataFrame.

Along with this, there were text and description columns/features which contained tons of information regarding the tweets. So, I had to combine both of them in order to fill the missing values in the description column. The reason I also combined is that I would have a lot of information while training my models later in this section.

IV. Data Story/ EDA:

I have calculated the different gender type counts from the dataset.

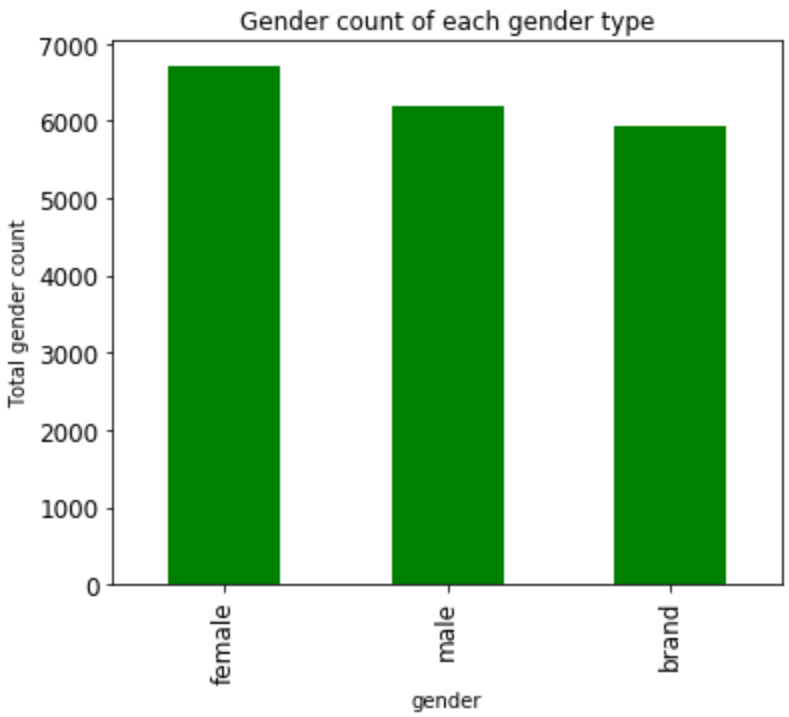
The below bar plot shows the gender count type of how many genders are present in the dataset. There is an unknown gender located as well in the dataset, which was not needed. So, I have removed those values, and retained only the useful information.

Besides this, the tweet counts for each gender were also calculated. This was useful in knowing which gender has more tweet counts.

Total female tweets: 5725

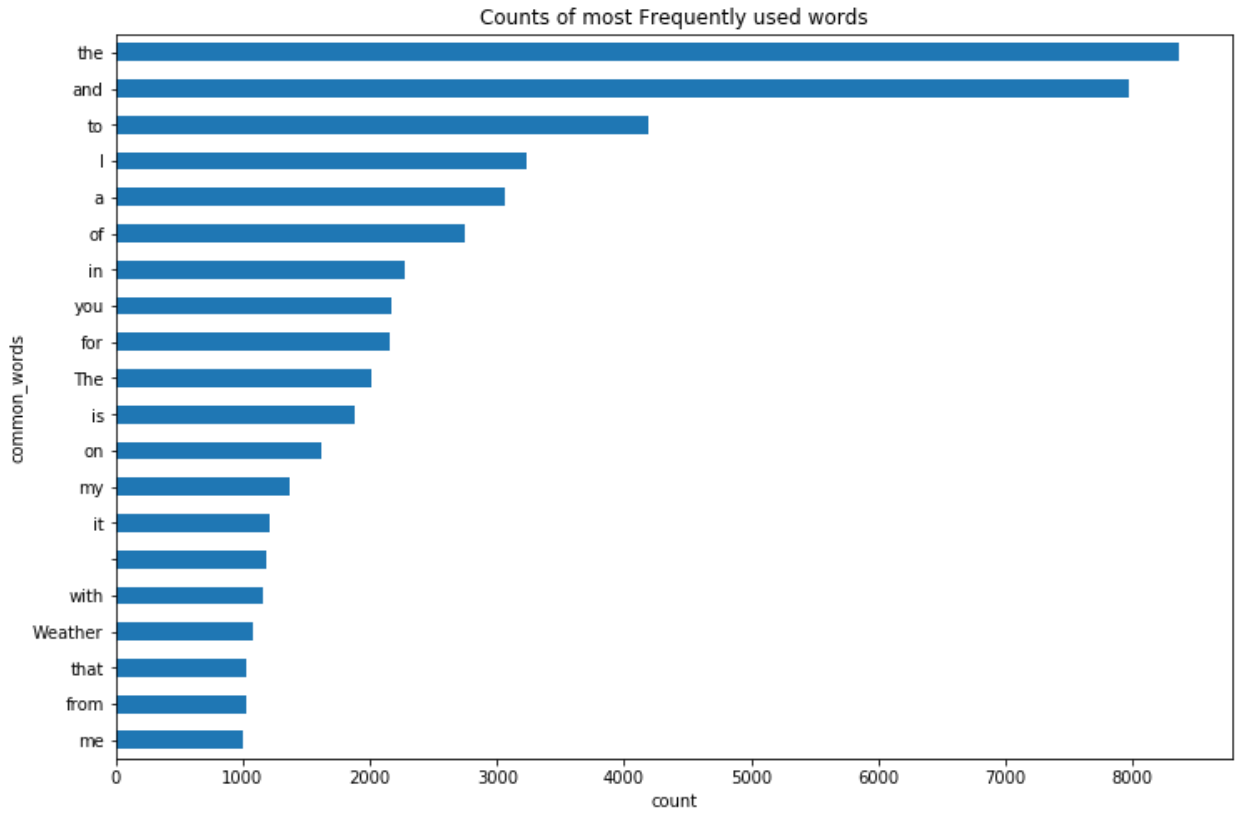
Total male tweets: 5469

Total brand tweets: 4328



We now need to somehow manipulate the text data for our data to preprocess and be useful for our models later. NLP contains a library called NLTK. It stands for Natural Language Toolkit. This toolkit is one of the most powerful NLP libraries which contains packages to make machines understand human language and reply to it with an appropriate response. Tokenization, Stemming, Lemmatization, Punctuation, Character count, word count are some of these packages present.

The below plot shows the most common words count frequently being used throughout the dataset text feature. Later on, the text and description features get combined to impute for the missing values. Also, we might get rich content of data from the combined features. But the above plot contains only normal English language words like a, the, an etc., which is not much from the ‘text’ feature. We will shortly introduce the stopwords concept that can reduce this drawback.



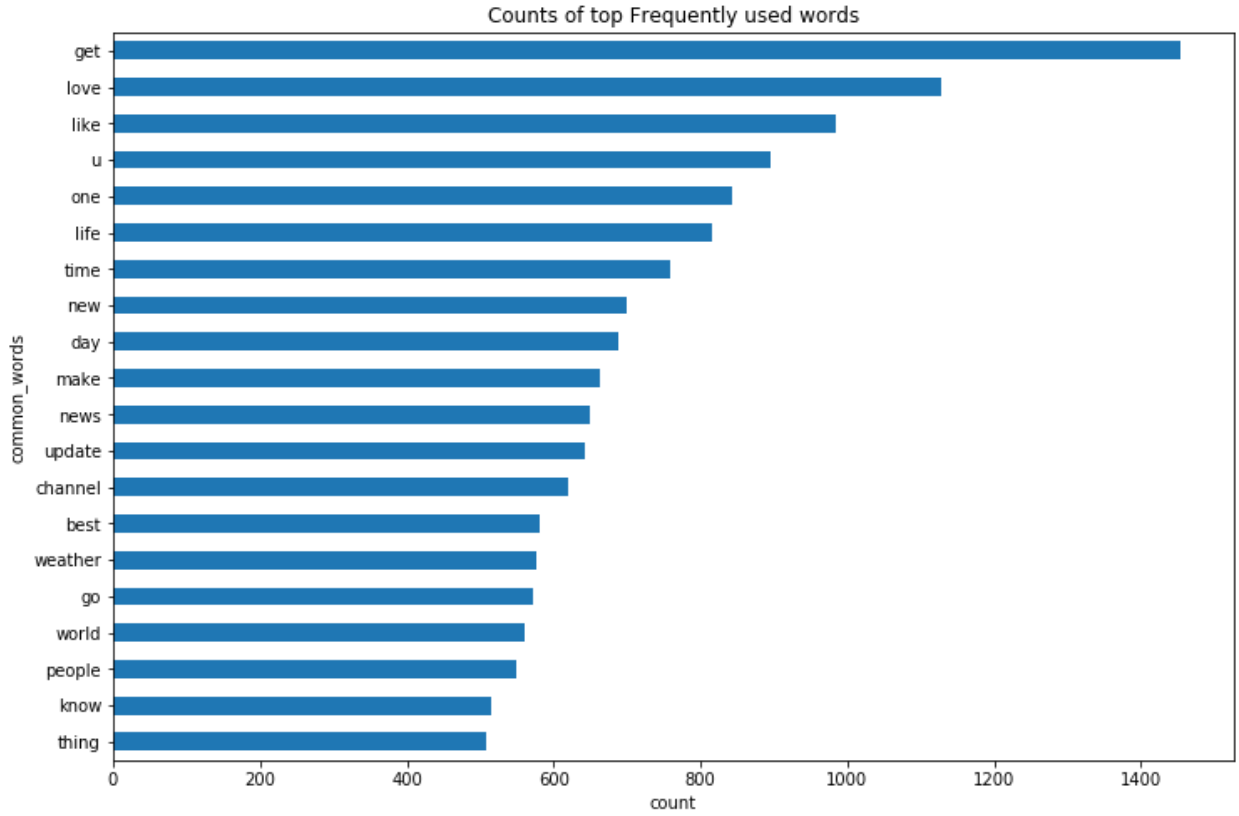
Removing stop words and applying Lemmatization:

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore. In natural language processing, useless words (data), are referred to as stop words.

Lemmatization is also stemming but produces results which are all valid words.

Also, a regular expression is used below to specify a set of strings that matches it; they allow you to check if a particular string matches a given regular expression.

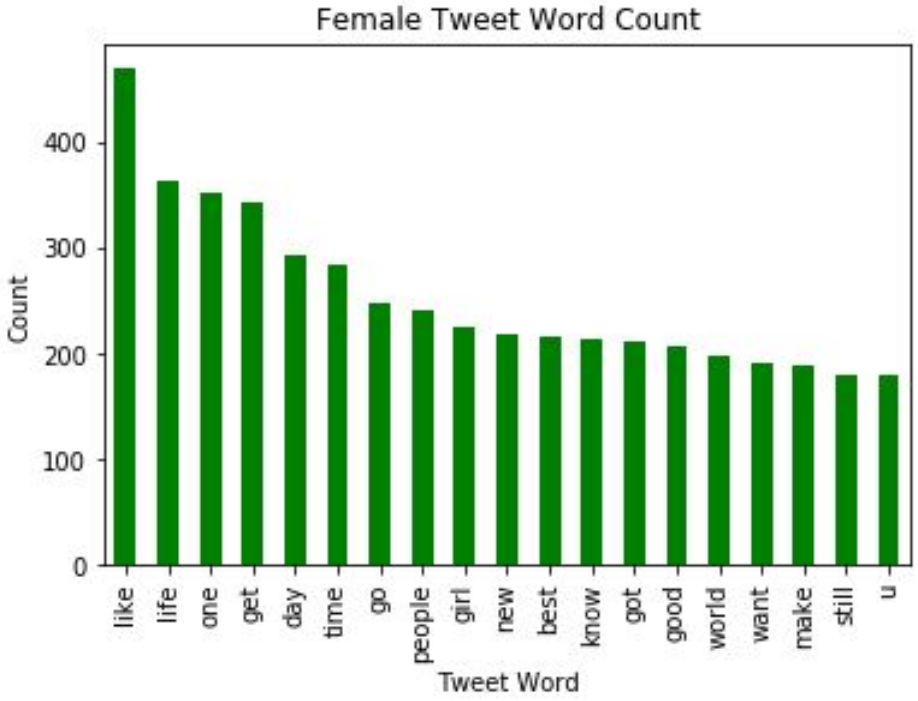
The plot now is cleaned up using regular expressions, tokenizing and using stopwords to the combined features of text and description. We will next see how to generate the most common frequency of words using a concept called **‘WordCloud’.**



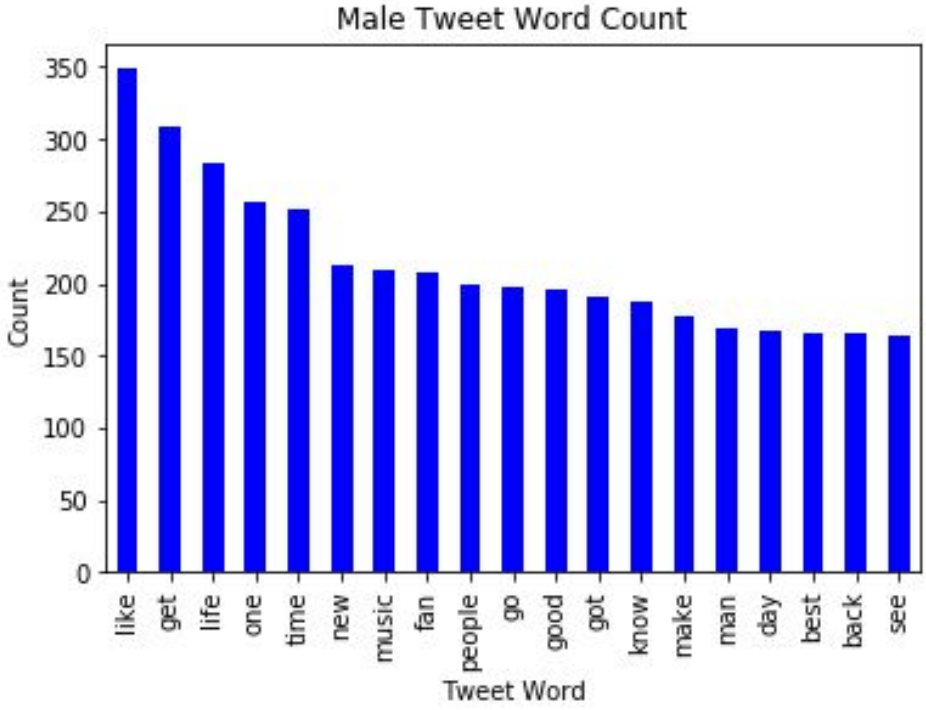
Wordcloud:

Ususally, whenever we deal with NLP or text, we see a cloud filled with lots of words in different sizes, which represent the frequency or the importance of each word. Wordcloud is the exact representation of this. It is a data Visualization technique used for text analysis.

From the image, we now see that the words new, love, life, one etc.. are displayed in a large font, meaning that these words occur very frequently and the frequency for these words is more. The duplicates also have been removed by the script, so that we would know the importance of the word once.



The bar plot shows the female Tweet word count. The frequency is described as to which word occurs more frequently among female gender. The same applies to male gender as well on how frequent the word is common and how many times it appears.



V. Statistical Test (t-test):

Hypothesis testing is usually used for inferencing from a population of the dataset. I have used a t-test statistic for my dataset. This is a two-sided test for the null hypothesis that 2 independent samples have identical average (expected) values. It calculates the mean of two independent samples of scores.

Male vs Female: Statisitcally Significant
Ttest_indResult(statistic=6.822643594255689, pvalue=9.454733981475721e-12)

Here, from the above result we see that the p-value is almost ~0. Now we can say that there is a statistically signifincant difference between male and female average length of words in a text. Let us now try performing the same test on a cleaned column, combined with the description to see if this holds the same for the below scenario as well.

Male vs Female: Statisitcally Significant
Ttest_indResult(statistic=18.992209470544758, pvalue=4.7982449742186716e-79)

As we can see from the above result we see that the p-value is almost ~0. Now we can say that there is a statistically significant difference between male and female average length of words in the cleaned version of the text as well. This really helps in comparing a sample mean to a hypothesized value.

VI. Bag of Words(BOW) model:

Whenever we apply any algorithm in NLP, it works on numbers. We cannot directly feed our text into that algorithm. Hence, the Bag of Words model is used to preprocess the text by converting it into a bag of words, which keeps a count of the total occurrences of most frequently used words. This is a

This model can be visualized using a table, which contains the count of words corresponding to the word itself.

There are three ways of doing this. I used the CountVectorizer method for doing this.

This creates vectors that have a dimensionality equal to the size of our vocabulary, and if the text data features that vocab word, we will put a one in that dimension. Every time we encounter that word again, we will increase the count, leaving 0s everywhere we did not find the word even once. The result of this will be very large vectors.

Also, here we use “One Hot Encoding” technique because our target variable is a multi class(male, female, brand), so the machine learning models would not be able to work using the classes as the target variable. So I have used LabelEncoder to transform the classes to binary numbers for Machine Learning models.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y = encoder.fit_transform(y)
```

VII. Data Modeling:

In this step, We will find a model to predict gender by accounting and taking into factors of the tweets data(which is text). Before using machine learning algorithms to get the best model, we need to prepare a dataset. I have prepared all the training features into one dataframe called X and the target variable y into another dataframe.

```
X.columns.values[:50]
```

```
array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23',
       '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34',
       '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45',
       '46', '47', '48', '49'], dtype=object)
```

```
y.columns.values
```

```
array(['0'], dtype=object)
```

As we can see the y-value is encoded using a LabelEncoder. All the values will be later converted to their original classes with inverse_transform.

VIII. Predictive Modeling:

a. Using Fastai:

Using Google Colab:

I have used Google Colab notebook for my fastai library for classification. Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

Here, I have used the Fastai deep learning Library as one of the options to train my model on. I would not say I have trained from scratch, because the fastai library comprises the method called Transfer **Learning.** It is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones. It consists of a pre-trained model where I

can use my dataset to classify the gender on the text data. It is a very useful technique which makes the classification pretty accurate. It is a Pre-trained Model Approach.

We use TextLMDataBunch to get the data ready for a language model. This does all the necessary preprocessing behind the scene.We can create a learner object(language_model_learner) that will directly create a model, download the pretrained weights and be ready for fine-tuning. We can use the data_lm object we created earlier to fine-tune a pretrained language model.

```
[ ]  # Train language model
     data_lm = TextLMDataBunch.from_df(train_df = sub_df_trn, valid_df = sub_df_val, path = "")

     learn = language_model_learner(data_lm, AWD_LSTM, drop_mult=0.5)
     learn.fit_one_cycle(1, 1e-2)
```

Also, we use TextClasDataBunch to get the data ready for a text classifier. We now use the data_clas object we created earlier to build a classifier with our fine-tuned encoder. The learner object can be done in a single line.

```
[ ]  # Build classifier
     data_clas = TextClasDataBunch.from_df(train_df = sub_df_trn, valid_df = sub_df_val, path=".", vocab=data_lm.train_ds.vocab, bs=32)

     learn = text_classifier_learner(data_clas, AWD_LSTM, drop_mult=0.5)
     learn.load_encoder('ft_enc')
```

To improve the accuracy further, we use freeze_to method with different layers in it. We don't unfreeze the whole thing but to unfreeze one layer at a time. unfreeze the last two layers freeze_to(-2), train it a little bit more. unfreeze the next layer freeze_to(-3), train it a little bit more. unfreeze the whole thing unfreeze(), train it a little bit more.

```
[ ] # unfreeze model and fine tune it
    learn.freeze_to(-3)
    learn.fit_one_cycle(1, slice(1e-3/2., 1e-3))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.729154	0.667992	0.692322	00:14

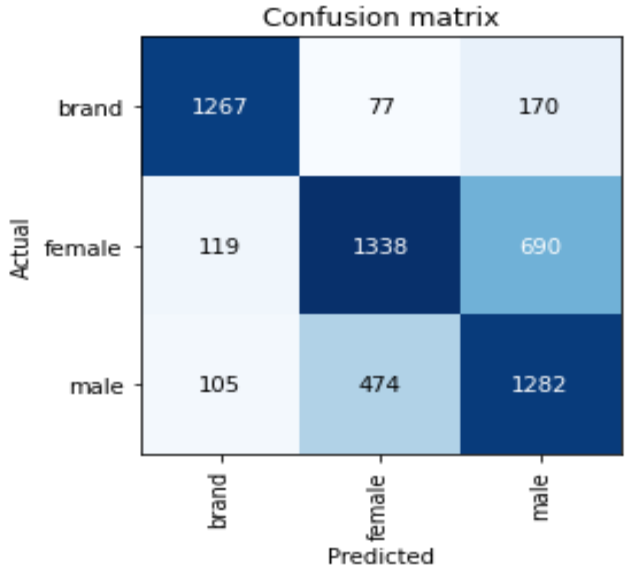
```
[ ] learn.unfreeze()
    learn.fit_one_cycle(1, slice(1e-3/100, 1e-3))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.667758	0.660029	0.705541	00:18

We can see that after training for a few layers here, the accuracy jumped to almost 7% more than the original. This is a technique used in Transfer Learning. The Validation loss and the training loss decreased too. Here it is important to note that we are exactly not overfitting the data because the training and validation loss are almost the same, so we don't have to train the model more.

Note: We use the original Text data here(not the cleaned version), to predict the accuracy and it did a very good job in detecting the accuracy

Plotting the confusion matrix. Confusion-matrix is a good technique to summarize the performance of a classification algorithm. We use ClassificationInterpretationclass here.



b. <u>Models built from scratch:</u>

Besides fastai, I also wanted to build models and see how they perform in comparison to their counterparts in fastai. So, here I mainly used 3 algorithms.

i. LogisticRegression :

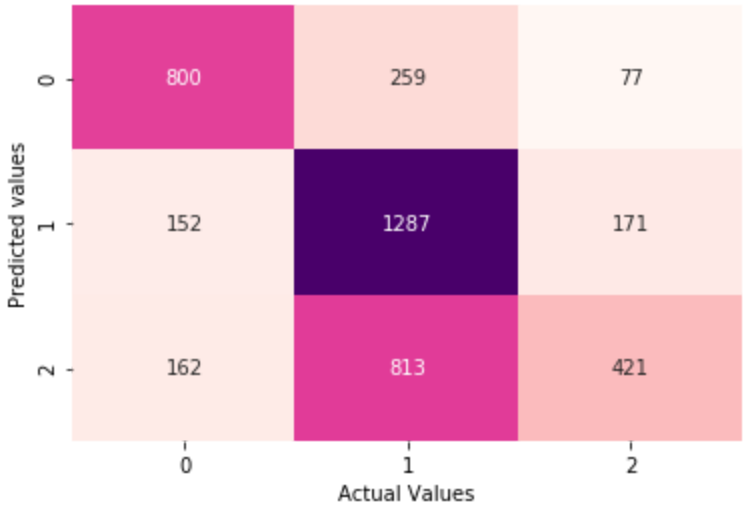
Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. It is a special case of linear regression where the target variable is categorical in nature. The hypothesis of logistic regression tends it to limit the cost function between 0 and 1.

For this, I set the parameter grid for selecting the best parameters using GridSearchCV for my RandomForestRegressor. It also contains cross validation within the Gridsearch on the dataset. This method is called hyperparameter tuning, where optimization is the key factor for selecting the best parameters. I also plotted the confusion matrix as shown below.

```
Accuracy: 0.6055045871559633
Confusion matrix:
 [[ 800  259   77]
 [ 152 1287  171]
 [ 162  813  421]]
Classification report:
              precision    recall  f1-score   support

           0       0.72      0.70      0.71      1136
           1       0.55      0.80      0.65      1610
           2       0.63      0.30      0.41      1396

    accuracy                           0.61      4142
   macro avg       0.63      0.60      0.59      4142
weighted avg       0.62      0.61      0.58      4142
```



Note: ***Here class 0 : Brand, class 1: Female, class 2: Male***

We can know this from the below code snippet where I have decoded the values too.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y = encoder.fit_transform(y)
```

```
y[:10]
```

```
array([2, 2, 2, 1, 1, 0, 2, 1, 1, 0])
```

```
decoded = encoder.inverse_transform(y)
decoded[:10]
```

```
array(['male', 'male', 'male', 'female', 'female', 'brand', 'male',
       'female', 'female', 'brand'], dtype=object)
```

From the above, let me elaborate on what the metrics really are.

Confusion Matrix: It is a performance measurement for machine learning classification problems where output can be two or more classes. It is a table with different combinations of predicted and actual values. It helps us understand the True Positive, False positives, True Negatives and False Negatives.

Precision: Out of all the positive classes we have predicted correctly, how many are actually positive. It is a ratio of TP/TP+FP.

Recall: Out of all the positive classes, how much we predicted correctly. It should be high as possible. It is a ratio of TP/TP+FN.

F1-score: The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

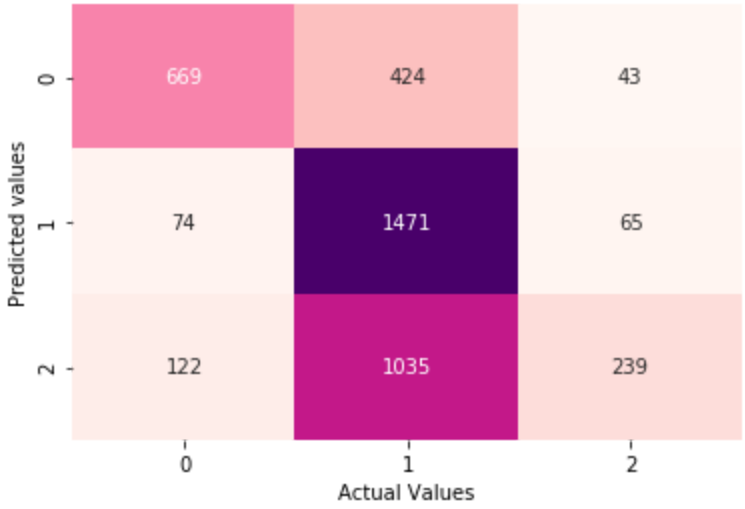
ii. RandomForestClassifier:

I have also used the Random Forest ensemble method which is a non linear model. In this, I use a Classifier, since my output contains multiple classes to determine. It uses multiple decision trees and a technique called bagging. This combines multiple decision trees in determining the final output rather than relying on individual decision trees.

```
Accuracy: 0.5743602124577499
Confusion matrix:
 [[ 669  424   43]
 [  74 1471   65]
 [ 122 1035  239]]
Classification report:
              precision    recall  f1-score   support

           0       0.77      0.59      0.67      1136
           1       0.50      0.91      0.65      1610
           2       0.69      0.17      0.27      1396

    accuracy                           0.57      4142
   macro avg       0.65      0.56      0.53      4142
weighted avg       0.64      0.57      0.53      4142
```



iii. SVM

This is a C-Support Vector Classification. SVM's are mainly used for the below points.

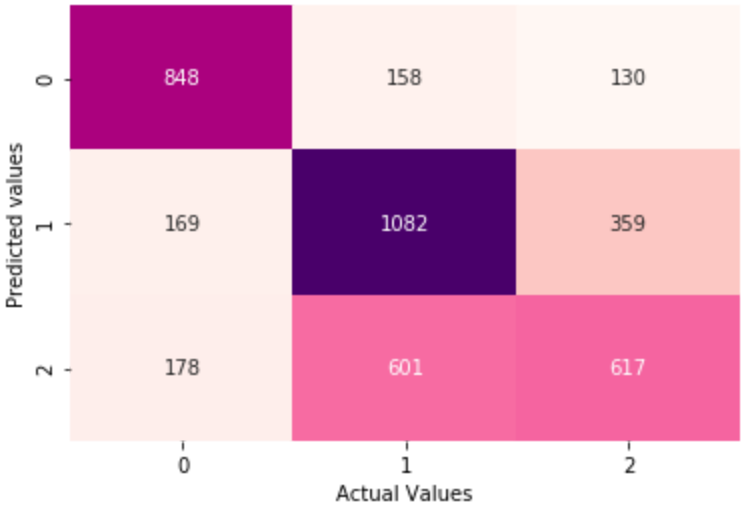
- SVM maximizes margin, so the model is slightly more robust (compared to linear regression), but more importantly: SVM supports kernels, so it can be modelled using non linear points too.
- These are effective in high dimensional spaces.
- Still effective in cases where the number of dimensions is greater than the number of samples.

- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

```
Accuracy: 0.6149203283437953
Confusion matrix:
 [[ 848  158  130]
 [ 169 1082  359]
 [ 178  601  617]]
Classification report:
              precision    recall  f1-score   support

           0       0.71      0.75      0.73      1136
           1       0.59      0.67      0.63      1610
           2       0.56      0.44      0.49      1396

    accuracy                           0.61      4142
   macro avg       0.62      0.62      0.62      4142
weighted avg       0.61      0.61      0.61      4142
```

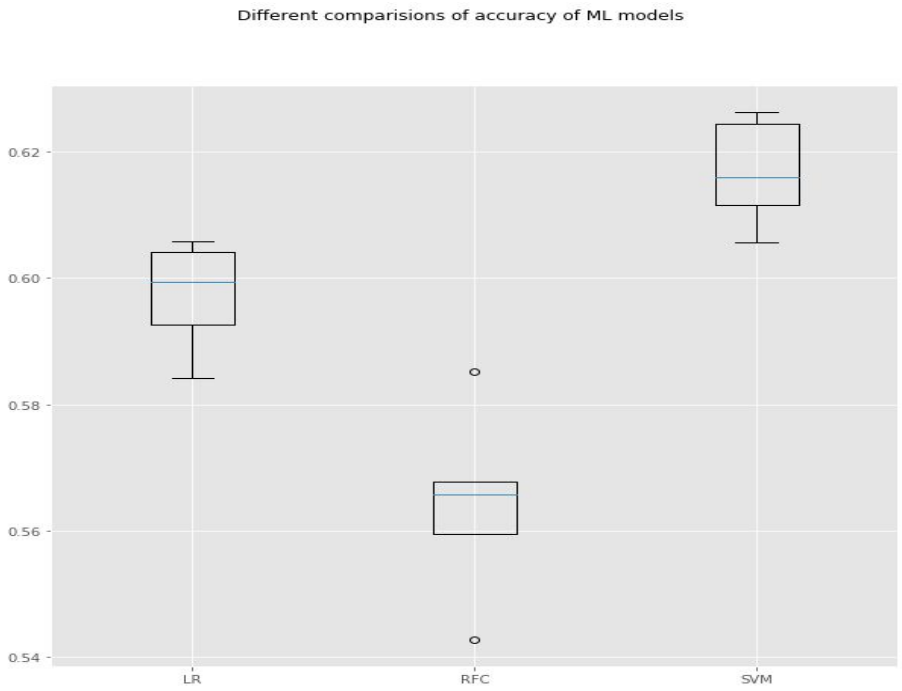


The below box plots show comparison of accuracies of different models which have been developed above. I used a K fold cross validation method to obtain a specific score on how well the model performed. Even though the results might differ a bit because we used KFold for checking the score, It's almost similar to the results of what we obtained above. This is separately trained and the results are used.

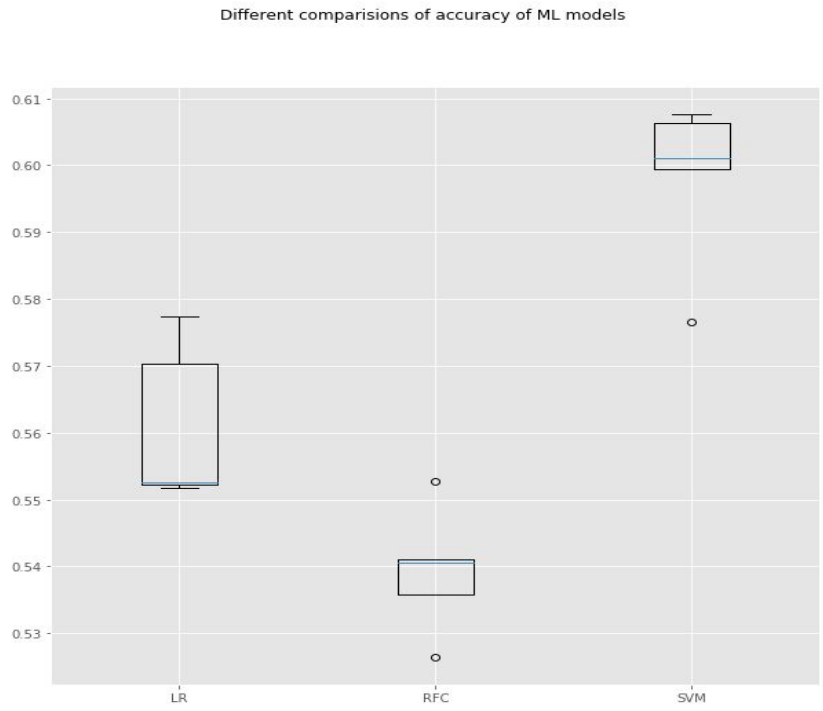
LR: 0.597185 (0.007982)

RFC: 0.564169 (0.013691)
SVM: 0.616746 (0.007782)

These are the cross validation scores with the parameters set for the KFold validation of n_splits=5 for the original text and description columns combined of tweet data.



I have performed the same algorithms on the cleaned version of the text as well. Below you can see the results comparison to the above on the original text data.



The above box plot shows the comparison of accuracies of different models which have been developed above using KFold cross Validation. These are the cross validation scores with the parameters set for the KFold validation of n_splits=5 for the cleaned(applying Lemmatization, stopwords etc.) text and description columns combined of tweet data.

LR: 0.560856 (0.010851)

RFC: 0.539329 (0.008562)
SVM: 0.598219 (0.011240)

IX. Results:

As we can see from the above, even though we built our models from scratch and trained them on the whole text, we get good accuracy scores. But in comparison, we used a transfer learning technique using fastai library and we are able to get a whopping increase in the results in classifying different genders. So, that is a pretty powerful technique to implement later when we can add a few more features along with

text, since we can decrease the training and validation loss too, which is important for not overfitting or underfitting the data.

Models trained on the entire text data:

ML Model	Accuracy scores on original text	Accuracy scores on cleaned text
Logistic Regression	60.55%	61.03%
Random Forest Classifier	57.43%	54.85%
Support Vector Machine(SVM)	61.49%	59.07%

Pre-defined Model(Transfer Learning approach):

epoch	train_loss	valid_loss	accuracy	time
0	0.667758	0.660029	0.705541	00:18

X. Future work:

There was a lot which I got to work on in this project, where I had to predict the gender based on text data. I have used fastai deep learning library to correctly identify the gender with the pre-trained models. In this dataset, I tried to use Images as profileImage for classification as well, but I did not have the chance to work on it. Definitely, this can drastically improve on the work I have been doing in the future. Apart from this, I can incorporate some more features along with the text data into my predictors, so I can see some improvement on the classification. With the models, SVM can be worked along with PCA for reducing the feature space, where I encode my text data as separate vectors. It takes time to execute SVM alone. Probably adding dimensionality reduction can work. These mentioned topics are worthwhile to work on and improve the accuracy much more.