Project 3: r/pacmasterrace vs r/mac

Problem Statement

The problem we are tackling is a binary classification problem. The goal is to create a model to accurately classify reddit posts based on the subreddits that it originated from - r/pcmasterrace or r/mac. We will be creating two models to compare - Bayes Naive Classifier and Logistic Regression Model.

Evaluation

The aim is to create a model that is able to generalize well across new observations and accurately predict the origin of the posts from the correct subreddit.

Data Set Used

There are two datasets obtained from the reddit API:

970	**1,000**
r/pcmasterrace	**r/mac**

Data Collection

Functions for data scraping and transforming into Data Frame:

```
def get_posts(url,interactions,header,sleep):
    posts = []
    after = None
    for i in range(interactions):
        print(i)
        if after == None:
            params = {}
        else:
            params = {'after':after}
        res = requests.get(url, params=params, headers=headers)
        if res.status_code == 200:
            the_json = res.json()
            posts.extend(the_json['data']['children'])
            after = the_json['data']['after']
        else:
            print(res.status_code)
            break
        time.sleep(sleep)
    return(posts)
```

```
def create_cols(dataframe):
    dataframe['subreddit'] = dataframe['data'].map(lambda x: x['subreddit'])
    dataframe['title'] = dataframe['data'].map(lambda x: x['title'])
    dataframe['name'] = dataframe['data'].map(lambda x: x['name'])
    dataframe['selftext'] = dataframe['data'].map(lambda x: x['selftext'])
    dataframe['domain'] = dataframe['data'].map(lambda x: x['domain'])
    return dataframe
```

```
header = {'User-agent': 'Bleep blorp bot 0.1'}
url = 'https://www.reddit.com/r/pcmasterrace.json'
interations = 40
sleep_sec = 1.5
pcmasterrace_df = pd.DataFrame(get_posts(url,interations,header,sleep_sec))
pcmasterrace_df = create_cols(pcmasterrace_df)
```

```
header = {'User-agent': 'Bleep blorp bot 0.1'}
url = 'https://www.reddit.com/r/mac.json'
interations = 40
sleep_sec = 1.5
mac_df = pd.DataFrame(get_posts(url,interations,header,sleep_sec))
mac_df = create_cols(mac_df)
```

Data Collection - Remove Duplicates

```
pcmasterrace_df.shape
```

```
(997, 7)
```

```
pcmasterrace_df.head()
```

	kind	data	subreddit	title	name	selftext	domain
0	t3	{'approved_at_utc': None, 'subreddit': 'pcmast...	pcmasterrace	Folding@Home and PCMR team up! Use your PC to ...	t3_dln0o3	This is the 8th iteration of this thread, sinc...	self.pcmasterrace
1	t3	{'approved_at_utc': None, 'subreddit': 'pcmast...	pcmasterrace	I spent all night alone at the MSI CES booth w...	t3_es1t4h	NaN	youtube.com
2	t3	{'approved_at_utc': None, 'subreddit': 'pcmast...	pcmasterrace	A funny title	t3_esooh7	NaN	i.redd.it
3	t3	{'approved_at_utc': None, 'subreddit': 'pcmast...	pcmasterrace	One of the madlad electricians at work made a ...	t3_esdg5b	NaN	i.redd.it
4	t3	{'approved_at_utc': None, 'subreddit': 'pcmast...	pcmasterrace	Linus reached 10 million, but is thinking of r...	t3_esp0ju	NaN	i.redd.it

```
mac_df.shape
```

```
(1000, 7)
```

```
mac_df.head()
```

	kind	data	subreddit	title	name	selftext	domain
0	t3	{'approved_at_utc': None, 'subreddit': 'mac', ...	mac	Picked up an '09 5,5 for $86	t3_esicey	NaN	i.imgur.com
1	t3	{'approved_at_utc': None, 'subreddit': 'mac', ...	mac	Window peel frosting makes for the perfect mou...	t3_es5imf	NaN	i.redd.it
2	t3	{'approved_at_utc': None, 'subreddit': 'mac', ...	mac	I prefer the Macbook Pro 2018 over my Razer Bl...	t3_esogm2	Hey, y'all. I am a programming student and pur...	self.mac
3	t3	{'approved_at_utc': None, 'subreddit': 'mac', ...	mac	My Mac editing suite is finished and the wife ...	t3_esod1h	NaN	i.redd.it
4	t3	{'approved_at_utc': None, 'subreddit': 'mac', ...	mac	Don't know why I can't find a simple solution....	t3_esppmj	On my iPhone I use Aloha app and use their bro...	self.mac

Removing Duplicate Rows

```
pcmasterrace_df.drop_duplicates(subset=['subreddit', 'title', 'name', 'selftext', 'domain'],keep='first', inplace=Tr
mac_df.drop_duplicates(subset=['subreddit', 'title', 'name', 'selftext', 'domain'],keep='first', inplace=True)
```

Checking No. of Rows

```
print("pcmasterrace", pcmasterrace_df.shape)
print("mac", mac_df.shape)
```

```
pcmasterrace (970, 7)
mac (1000, 7)
```

Data Cleaning and EDA

Replacing nulls with string null:

```
#Replacing null with 'null'
pcmasterrace_df['selftext'].fillna('null9999', inplace=True)
mac_df['selftext'].fillna('null9999', inplace=True)
```

Joining Dataframes and Creating Columns

```
final_df = pd.concat([pcmasterrace_df, mac_df], axis=0, join='outer',ignore_index=False)
final_df = final_df.reset_index(drop=True)
```

Creating new columns from website information:

```
final_df['ups'] = final_df['data'].map(lambda x: x['ups'])
final_df['num_comments'] = final_df['data'].map(lambda x: x['num_comments'])
final_df['author'] = final_df['data'].map(lambda x: x['author'])
```

Creating Label column:

```
final_df['label'] = final_df['subreddit'].map({'pcmasterrace':0,'mac':1})
```

Removing un-used columns:

```
drop_columns = ['data','kind','domain','name','subreddit']
final_df.drop(columns=drop_columns,axis=1,inplace=True)
```

Data Cleaning and EDA

Removing space,tab and breakline and stop words:

```
nltk.download()  # Download text data sets, including stop words. Uncomment this if you did not download
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

True
```

```
def clean_text(text_to_clean):
    text_to_clean = re.sub( '[^a-zA-Z0-9]', ' ', text_to_clean) # subs charact in the brackets
    text_to_clean = re.sub( '\s+', ' ', text_to_clean).strip() ## subs tabs,newlines and "whitespace-like"
    words = text_to_clean.lower().split() ## convert to lowercase split indv words
    stops = set(stopwords.words('english')) #converting stop words to set
    meaningful_words = [w for w in words if not w in stops] # removing stop words
    return(" ".join(meaningful_words))
```

```
final_df['clean_title'] = final_df.apply(lambda x: clean_text(x['title']), axis=1)
final_df['clean_selftext'] = final_df.apply(lambda x: clean_text(x['selftext']), axis=1)
```

Data Cleaning and EDA

Creating stemming Function:

```
porter=PorterStemmer()
lancaster=LancasterStemmer()
lemmatizer = WordNetLemmatizer()

def stemtext(sentence,steamer):
    token_words=word_tokenize(sentence)
    token_words
    stem_sentence=[]
    for word in token_words:
        if (str(steamer) == '<WordNetLemmatizer>'):
            stem_sentence.append(steamer.lemmatize(word))
        else:
            stem_sentence.append(steamer.stem(word))

        stem_sentence.append(" ")
    return "".join(stem_sentence)
```

Using function to create stemming columns:

```
# For title column
final_df['clean_title_lemmat'] = final_df.apply(lambda x: stemtext(x['clean_title'], WordNetLemmatizer()), axis=1)
final_df['clean_title_lancast'] = final_df.apply(lambda x: stemtext(x['clean_title'], LancasterStemmer()), axis=1)
final_df['clean_title_port'] = final_df.apply(lambda x: stemtext(x['clean_title'],PorterStemmer()), axis=1)
```

```
# For self text column
final_df['clean_selftext_lemmat'] = final_df.apply(lambda x: stemtext(x['clean_title'], WordNetLemmatizer()), axis=1
final_df['clean_selftext_lancast'] = final_df.apply(lambda x: stemtext(x['clean_title'], LancasterStemmer()), axis=1
final_df['clean_selftext_port'] = final_df.apply(lambda x: stemtext(x['clean_title'],PorterStemmer()), axis=1)
```

Data Cleaning and EDA

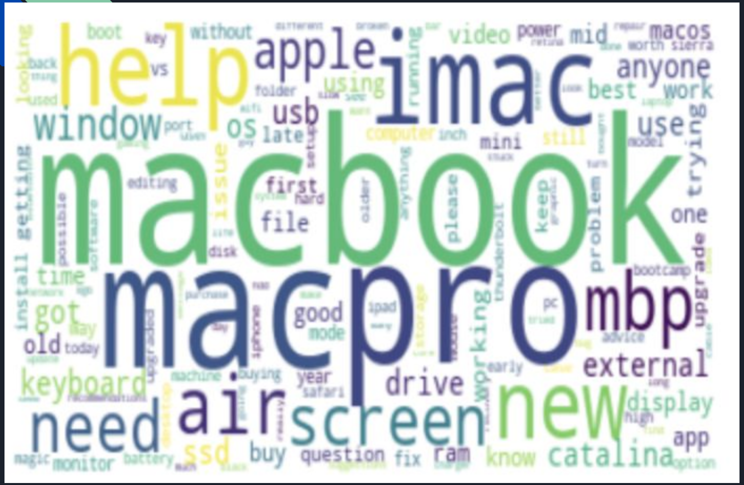
Dealing with empty rows post lemmatizing:

- After lemmatizer the row 825 had no word in the title.
- Dropping drop of 1 row will not have significant affect in the model.

```
#droping column where lemmatizer left no text
display(final_df[final_df['clean_title'].str.len()< 1])
final_df = final_df[~(final_df['clean_title'].str.len()< 1)]
```

	title	selftext	ups	num_comments	author	label	clean_title	clean_selftext	clean_title_lemmat	clean_title_lancast	clean_title_port	clean_selftext_
825	how	how do you fancy dudes put your system specs n...	3	7	pandason89	0		fancy dudes put system specs next usernames				

Data Cleaning and EDA - Word Cloud




```
'mac title common word

[(279, 'macbook'),
 (225, 'pro'),
 (214, 'mac'),
 (96, 'imac'),
 (83, 'help'),
 (64, 'new'),
 (50, '2019'),
 (47, 'air'),
 (45, 'screen'),
 (44, '16'),
 (42, 'mbp'),
 (40, 'apple'),
 (39, 'need'),
 (37, 'get'),
 (35, 'anyone')]
```


```
'pcmasterrace title common words'

[(155, 'pc'),
 (84, 'build'),
 (81, 'new'),
 (68, 'help'),
 (54, 'gpu'),
 (53, 'first'),
 (47, 'monitor'),
 (41, 'cpu'),
 (39, 'need'),
 (36, 'get'),
 (34, 'good'),
 (32, 'gaming'),
 (28, 'one'),
 (25, 'upgrade'),
 (25, 'time')]
```

Preprocessing and Modelling

Base Line Accuracy

```
df['label'].value_counts(normalize=True)[0:1]  # value and pct
```

```
1    0.507872
Name: label, dtype: float64
```

- As calculated above, the baseline accuracy for the dataset is 50.79%.

Split test:

```
#Preparing data
X = df[features].iloc[:,0]
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X,y,stratify=y,random_state=42)
```

Testing which stemmer / lemmatizer give best results in multinomialNB

```
cv = CountVectorizer()
model_mult_nb = MultinomialNB()
alphas = np.linspace(0,2,20)[1:]

pipe = Pipeline([('cv',cv),
                ('model',model_mult_nb)
])
```

Preprocessing and Modelling

```
params = {'cv__stop_words': [['pc','macbook','mac','imac']],
           'cv__max_features':[4000,5000,6000,None],
          'cv__ngram_range': [(1,1)],
          'cv__min_df': [1,5,10,15],
          'cv__max_df':[0.10,0.15,0.2,0.3],
           'model__alpha' : alphas
    }
gs = GridSearchCV(pipe, param_grid=params,cv=5)
print(gs.fit(X_train,y_train))
print(gs.best_params_)
print("Train Score: ", round(gs.best_score_,4))
print("Test Score: ", round(gs.score(X_test,y_test),4))
```

```
                    scoring=None, verbose=0)
{'cv__max_df': 0.15, 'cv__max_features': 4000, 'cv__min_df': 1, 'cv__ngram_range': (1, 1), 'cv__stop_words': ['pc',
'macbook', 'mac', 'imac'], 'model__alpha': 1.1578947368421053}
Train Score:  0.8252
Test Score:  0.8337
```

- PorterStemmer yielded highest score in a multinomialNB model, using only title (ie. pc and mac root words) as stop words.
- The best hyperparameters from GridSearch are as follows: {'cv__max_df': 0.15, 'cv__max_features': 4000, 'cv__min_df': 1, 'cv__ngram_range': (1, 1), 'cv__stop_words': ['pc', 'macbook', 'mac', 'imac'], 'model__alpha': 1.1578947368421053}

Preprocessing and Modelling

Vectorizing and Appending other features:

```
vectorizer = CountVectorizer(stop_words = ['pc','macbook','mac','imac'],
                             max_features = 4000,
                             ngram_range = (1,1),
                             min_df = 1,
                             max_df = 0.15)
X_train_title_vec = vectorizer.fit_transform(X_train)
X_test_title_vec = vectorizer.transform(X_test)

print("Dic Size:", len(vectorizer.get_feature_names()))
```

```
Dic Size: 2300
```

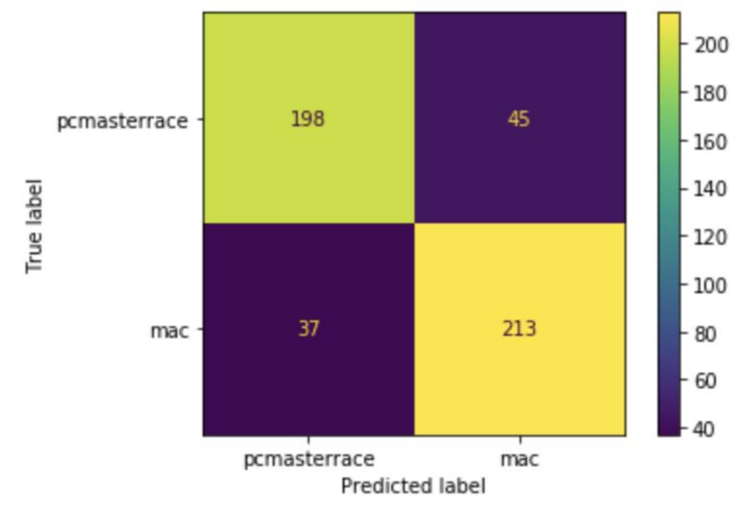
- Vectorizing and appending additional features names into our model to determine whether it helps improve the performance of our model.
- The dictionary size consist of 2,300 aditional feature names.

Preprocessing and Modelling - Naive Bayes

Multinomial Naive Bayes

```
cv = CountVectorizer()
model_mult_nb = MultinomialNB()
params = {'cv__stop_words': [['pc','macbook','mac','imac']],
          'cv__max_features':[4000],
         'cv__ngram_range': [(1,1)],
         'cv__min_df': [1],
         'cv__max_df':[0.15],
          'model__alpha' : alphas
    }
gs = GridSearchCV(pipe, param_grid=params,cv=5)
print(gs.fit(X_train,y_train))
print(gs.best_params_)
print("Train Score: ", round(gs.best_score_,4))
print("Test Score: ", round(gs.score(X_test,y_test),4))
```

```
Train Score:  0.8252
Test Score:  0.8337
```




```
Specificity: 0.8148
Sensitivity/Recall: 0.852
Precision: 0.8256
F1 Score: 0.8386
TN: 198
FP: 45
FN: 37
TP: 213
```

Preprocessing and Modelling - Naive Bayes

- As seen from the slightly higher Sensitivity score (85.2%) compared to the Specificity Score (81.48%), our model does a slightly better job of correctly predicting posts relating to the r/mac subreddit as compared to posts relating to the r/pcmasterrace subreddit.
- The model has a reasonably good recall score of 85.2%. However, we will only select this as our best model when there is a high cost associated with False Negative.
- The model has a reasonably good precision score of 82.56%. However, the cost of False Positive in this context is not high. Hence, we will not use this metric as our sole determinor for selecting our best model.
- The model has a reasonably good overall F1 Score of 83.86%. As there is neither a significant cost of False Positive/ False Negative in this business context, the F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

```
Specificity: 0.8148
Sensitivity/Recall: 0.852
Precision: 0.8256
F1 Score: 0.8386
TN: 198
FP: 45
FN: 37
TP: 213
```

Preprocessing and Modelling - Logistic Regression

Logistic Regression Model

```
#Without Hyperparameters
vectorizer = CountVectorizer(stop_words = ['pc','macbook','mac','imac'],
                             max_features = 4000,
                             ngram_range = (1,1),
                             min_df = 1,
                             max_df = 0.15)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

lr = LogisticRegression()

lr.fit(X_train_vec,y_train)
print("Cross Val Score: ",cross_val_score(lr,X_train_vec,y_train,cv=5).mean())
print("Train Score: ",lr.score(X_train_vec,y_train))
print("Test Score: ",lr.score(X_test_vec,y_test))
```

```
Cross Val Score:  0.8102954649564819
Train Score:  0.9728997289972899
Test Score:  0.8275862068965517
```

```
lr3 = LogisticRegression(max_iter=1000,
                         penalty='l2',
                         C=1,
                         class_weight={1: 0.4, 0: 0.6},
                         solver='saga',
                         )

lr3.fit(X_train_vec,y_train)
print("Cross Val Score: ",cross_val_score(lr3,X_train_vec,y_train,cv=5).mean())
print("Train Score: ",lr3.score(X_train_vec,y_train))
print("Test Score: ",lr3.score(X_test_vec,y_test))
```

```
Cross Val Score:  0.7967498854786991
Train Score:  0.9220867208672087
Test Score:  0.8093306288032455
```

```
penalty = ['l1', 'l2']
C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
solver = ['liblinear', 'saga']

lr2 = LogisticRegression(max_iter=1000)

param_grid = dict(penalty=penalty,
                  C=C,
                  class_weight=class_weight,
                  solver=solver)

grid = GridSearchCV(estimator=lr2,
                    param_grid=param_grid,
                    scoring='roc_auc',
                    verbose=1,
                    n_jobs=-1)

#Fit the model
best_model = grid.fit(X_train_vec,y_train)

#Print The value of best Hyperparameters
print('Best penalty:', best_model.best_estimator_.get_params()['penalty'])
print('Best C:', best_model.best_estimator_.get_params()['C'])
print('Best class_weight:', best_model.best_estimator_.get_params()['class_weight'])
print('Best solver:', best_model.best_estimator_.get_params()['solver'])
```

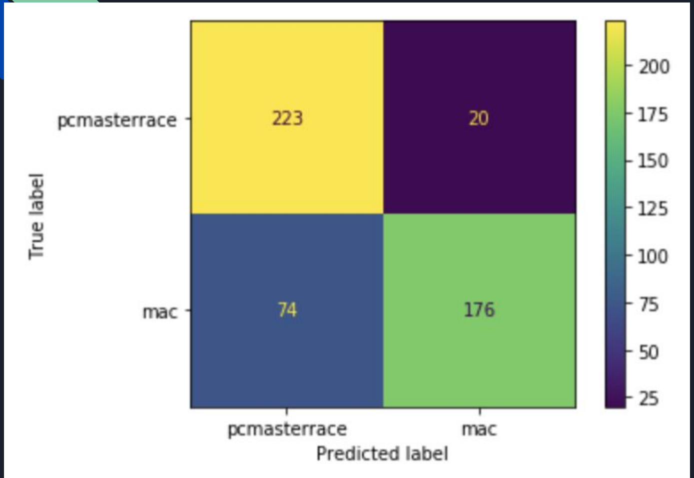
```
Fitting 5 folds for each of 128 candidates, totalling 640 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

Best penalty: l2
Best C: 1
Best class_weight: {1: 0.4, 0: 0.6}
Best solver: saga

[Parallel(n_jobs=-1)]: Done 640 out of 640 | elapsed:  2.0min finished
```

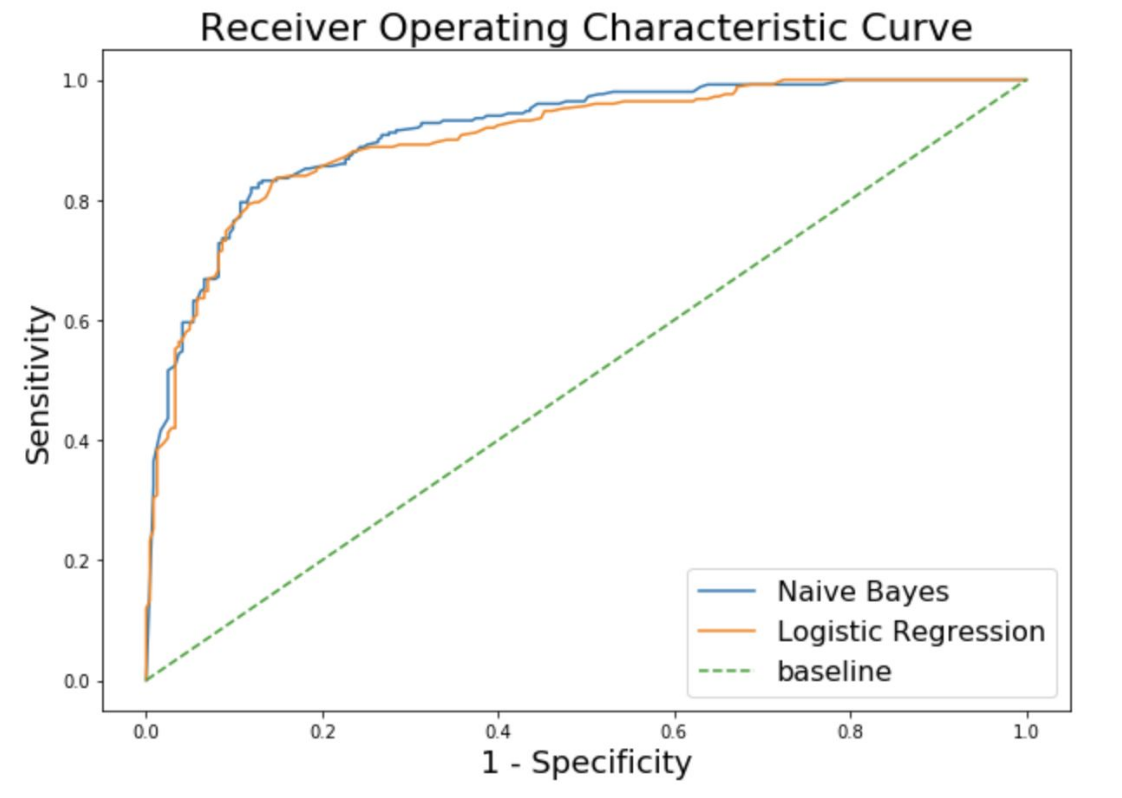
Preprocessing and Modelling - Logistic Regression



Specificity: 0.9177
Sensitivity/Recall: 0.704
Precision: 0.898
F1 Score: 0.7892
TN: 223
FP: 20
FN: 74
TP: 176

- As seen from the significantly higher Specificity score (91.77%) compared to the Specificity Score (70.4%), our model does a significantly better job of correctly predicting posts relating to the r/pcmasterrace subreddit as compared to posts relating to the r/mac subreddit.
- The model has a comparatively poor recall score of 70.4%. However, we will only select this as our best model when there is a high cost associated with False Negative.
- The model has a extremely good precision score of 89.8%. However, the cost of False Positive in this context is not high. Hence, we will not use this metric as our sole determinor for selecting our best model.
- The model has a decent overall F1 Score of 78.92%. As there is neither a significant cost of False Positive/ False Negative in this business context, the F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

Evaluation



Receiver Operating Characteristic Curve
Sensitivity
1 - Specificity
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes
Logistic Regression
baseline

Evaluation

Results:

- Based on our problem statement, the aim is to create a model that is able to generalize well across new observations and accurately predict the origin of the posts from the correct subreddit.
- As there is neither a significant cost of False Positive/ False Negative in this business context, the F1 Score is a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).
- Comparing the F1 Score of the Multinomial Naive Bayes Model (83.86%) and the Logistic Regression Model (78.92%), we can see that the Multinomial Naive Bayes Model is the better model in solving our problem statement.