AML_Project

April 10, 2022

1 Importing Libraries

[1]:
```
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict
%matplotlib inline
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

2 Preprocessing

[2]:
```
train = pd.read_csv('/content/drive/MyDrive/train.csv')
test = pd.read_csv('/content/drive/MyDrive/test.csv')
train.head()
```

[2]:
```
   id keyword location                                               text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M…
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are …
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or…
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as …

   target
0       1
1       1
2       1
3       1
4       1
```

[3]:
```
train = train.drop(['keyword','location'], axis = 1)
test = test.drop(['keyword','location'], axis = 1)
train.head()
```

[3]:
```
   id                                               text  target
0   1  Our Deeds are the Reason of this #earthquake M…       1
1   4             Forest fire near La Ronge Sask. Canada       1
2   5  All residents asked to 'shelter in place' are …       1
3   6  13,000 people receive #wildfires evacuation or…       1
4   7  Just got sent this photo from Ruby #Alaska as …       1
```

[4]:
```
# Checking Shape of Train and Test sets:
print("Shape of Train set:", train.shape)
print("Shape of Test set:", test.shape)
```

```
Shape of Train set: (7613, 3)
Shape of Test set: (3263, 2)
```

[5]:
```
# Removing duplicates of Train set. There are few duplicates in Test set as␣
↪well,
```

```
# however, duplicates of Test set can'b be removed because the final test with␣
↪target has to be uploaded as a submission file

train = train.drop_duplicates(subset=['text'], keep='last')
print("Shape of Train set after removing duplicates:", train.shape)
```

```
Shape of Train set after removing duplicates: (7503, 3)
```

[6]:
```
train[train['text'].map(lambda x: x.isascii())]
test[test['text'].map(lambda x: x.isascii())]

# Cleaning Tweets
def clean_tweets(text):
    text = re.sub(r'@[A-Za-z0-9_]+','',text)    # Removing @mentions
    text = re.sub(r'#','',text)                 # Removing #tag symbol
    text = re.sub(r'RT[\s]+',' ',text)          # Remvoing RT
    text = re.sub(r'\n','',text)
    text = re.sub(r',','',text)
    text = re.sub(r'.[.]+','',text)
    text = re.sub(r'\w+:\/\/\S+','',text)
    text = re.sub(r'https?:\/\/\S+','',text)    # Removing hyperlinks
    text = re.sub(r'/',' ',text)
    text = re.sub(r'-',' ',text)
    text = re.sub(r'_',' ',text)
    text = re.sub(r'!','',text)
    text = re.sub(r':',' ',text)
    text = re.sub(r'$','',text)
    text = re.sub(r'%','',text)
    text = re.sub(r'^','',text)
    text = re.sub(r'&','',text)
    text = re.sub(r'=',' ',text)
    text = re.sub(r' +',' ',text)               # Removing extra whitespaces

    return text

# Removing Emojis
def clean_emoji(inputString):
    return inputString.encode('ascii', 'ignore').decode('ascii')

train['text'] = train['text'].apply(clean_tweets)    # Applying function to␣
↪clean tweets
train['text'] = train['text'].apply(clean_emoji)     # Applying function to␣
↪remove emojis
train['text'] = train.text.str.lower()               # Making all texts to␣
↪lower case
train['text'] = train['text'].str.strip()            # Removing leading and␣
↪trailing whitespaces
```

```
test['text'] = test['text'].apply(clean_tweets)      # Applying function to␣
 ↪clean tweets
test['text'] = test['text'].apply(clean_emoji)       # Applying function to␣
 ↪remove emojis
test['text'] = test.text.str.lower()                 # Making all texts to␣
 ↪lower case
test['text'] = test['text'].str.strip()              # Removing leading and␣
 ↪trailing whitespaces
#pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', -1)
```

Labels are as follows:¶

'target' -> This denotes whether a tweet is about a real disaster (1) or not (0)

[7]:
```
train['target'].value_counts()
```

[7]:
```
0    4307
1    3196
Name: target, dtype: int64
```

3 Setups:

Each of our classification models (SVM, Naive Bayes, Logistic Regression, K Nearest Neighbours, Ada Boost, Gradient boosting and Random Forest) were tested on the following setups:

1. **Setup 1: Removing Punctuation:** All the models are trained and tested after removing punctuations from the corpus.
2. **Setup 2: Removing Stop-words:** All the models are trained and tested after removing stop-words from the corpus.
3. **Setup 3: Removing Numbers:** All the models are trained and tested after removing numbers from the corpus.
4. **Setup 4: Removing Repeating Characters:** All the models are trained and tested after removing repeating characters.
5. **Setup 5: Stemming and Lemmatization:** All the models are trained and tested after applying stemming and lemmatization.
6. **Setup 6: Setup 1–5:** All the models are trained and tested after removing punctuation, stop-words, numbers, repeating words, stemming and lemmatization.
7. **Setup 7: Keeping all above features:** All the models are trained and tested without eliminating any of the above special features.

4 Models:

4.0.1 These models with hyperparameters will be used by all setups, to find the best setup and best model:

[8]:
```
# making a dictionary with four models with some parameters:

model_params = {

    'SVC' :{
        'model' : SVC(),
        'params' : {
            'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01], 'kernel':␣
↪['rbf','linear','poly','sigmoid']
        }
    },

    'MultinomialNB' :{
        'model' : MultinomialNB(),
        'params' : {
            'alpha' : np.linspace(0.5, 1.5, 6), 'fit_prior' : [True, False]
        }
    },

    'logistics_regression' :{
        'model' : LogisticRegression(solver = 'lbfgs', multi_class = 'auto'),
        'params' : {
            'C' : [0.1, 1, 20, 40, 60, 80, 100], 'solver' : ['lbfgs',␣
↪'liblinear']
        }
    },
    'K_Nearest_Neighbors' :{
        'model' : KNeighborsClassifier(),
        'params' : {
            'n_neighbors' : [5, 10, 20, 50, 80, 100, 200], 'weights' :␣
↪['uniform', 'distance']
        }
    },

    'random_forest' :{
        'model' : RandomForestClassifier(),
        'params' : {
            'n_estimators' : [50,100,150],
            'max_depth':[2,3,None], 'criterion':['gini','entropy']
        }
    },
```

```
    'AdaBoost' :{
        'model' : AdaBoostClassifier(),
        'params' : {
            'n_estimators' : [50,100,150], 'learning_rate' : [0.5,1,1.5]
        }
    }
    #'Gradient_Boosting' :{ 'model' : GradientBoostingClassifier(),'params' :␣
 ↪{'n_estimators' : [50,100,150], 'criterion':['friedman_mse',␣
 ↪'squared_error', 'mae']}}
}
```

4.1 Setup 1: Models after removing Punctuations:

[9]:
```
# Creating a df that is copy of the train set.
df = train.copy()
```

4.1.1 Removing Punctuations:

[10]:
```
import string
string.punctuation
```

[10]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

[11]:
```
punctuations_list = string.punctuation
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)

df['text'] = df['text'].apply(lambda x: cleaning_punctuations(x))
```

4.1.2 Splitting data into Train and Test :

[12]:
```
# Splitting data into Train and Test sets:
X = df['text']
y = df['target']
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 3)
```

4.1.3 Transforming dataset using TF-IDF Vectorizer:

[13]:
```python
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  62381
```

4.1.4 Results:

[14]:
```python
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
↪created dictionary
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.


scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                          # Training the␣
↪model
    clf_pred = best_model.predict(X_test)                          # Predicting␣
↪the results
    print(confusion_matrix(y_test,clf_pred))                        # Printing␣
↪Confusion Matrix
    print(metrics.classification_report(y_test, clf_pred))      # Printing␣
↪Classification Report
    scores.append({                                                 # Appending␣
↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')
```

```python
# Creating data frame with model, best scores and best params:
res1 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()

Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[735 108]
 [185 473]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       843
           1       0.81      0.72      0.76       658

    accuracy                           0.80      1501
   macro avg       0.81      0.80      0.80      1501
weighted avg       0.81      0.80      0.80      1501


Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[778  65]
 [237 421]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       843
           1       0.87      0.64      0.74       658

    accuracy                           0.80      1501
   macro avg       0.82      0.78      0.79      1501
weighted avg       0.81      0.80      0.79      1501


Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[708 135]
 [179 479]]
              precision    recall  f1-score   support
```

```
           0       0.80      0.84      0.82       843
           1       0.78      0.73      0.75       658

    accuracy                           0.79      1501
   macro avg       0.79      0.78      0.79      1501
weighted avg       0.79      0.79      0.79      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[741 102]
 [221 437]]
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       843
           1       0.81      0.66      0.73       658

    accuracy                           0.78      1501
   macro avg       0.79      0.77      0.78      1501
weighted avg       0.79      0.78      0.78      1501


Score is appended.

RandomForestClassifier()

Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[768  75]
 [255 403]]
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       843
           1       0.84      0.61      0.71       658

    accuracy                           0.78      1501
   macro avg       0.80      0.76      0.77      1501
weighted avg       0.79      0.78      0.77      1501


Score is appended.

AdaBoostClassifier()
```

```
Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[686 157]
 [230 428]]
              precision    recall  f1-score   support

           0       0.75      0.81      0.78       843
           1       0.73      0.65      0.69       658

    accuracy                           0.74      1501
   macro avg       0.74      0.73      0.73      1501
weighted avg       0.74      0.74      0.74      1501


Score is appended.

CPU times: user 56.8 s, sys: 2.49 s, total: 59.3 s
Wall time: 45min 34s
```

[15]:
```python
res1
```

[15]:
```
                   model  best_score  \
0  SVC                    0.804797
1  MultinomialNB          0.798801
2  logistics_regression   0.790806
3  K_Nearest_Neighbors    0.784810
4  random_forest          0.780147
5  AdaBoost               0.742172

                                                         best_params
0  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 20, 'solver': 'liblinear'}
3  {'n_neighbors': 50, 'weights': 'distance'}
4  {'criterion': 'gini', 'max_depth': None, 'n_estimators': 100}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

4.2 Setup 2: Models after removing Stop-words:

[16]:
```python
# Creating a df that is copy of the train set.
df = train.copy()
```

4.2.1 Removing Stop-words:

[17]:
```
sw = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if␣
↪word not in (sw)]))
```

4.2.2 Splitting data into Train and Test :

[18]:
```
# Splitting data into Train and Test sets:
X = df['text']
y = df['target']
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
↪random_state = 3)
```

4.2.3 Transforming dataset using TF-IDF Vectorizer:

[19]:
```
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  51017
```

4.2.4 Results:

[20]:
```
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
↪created dictionary
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.


scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
```

```
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                          # Training the␣
 ↪model
    clf_pred = best_model.predict(X_test)                           # Predicting␣
 ↪the results
    print(confusion_matrix(y_test,clf_pred))                        # Printing␣
 ↪Confusion Matrix
    print(metrics.classification_report(y_test, clf_pred))          # Printing␣
 ↪Classification Report
    scores.append({                                                 # Appending␣
 ↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')

# Creating data frame with model, best scores and best params:
res2 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()

Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[728 115]
 [182 476]]
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       843
           1       0.81      0.72      0.76       658

    accuracy                           0.80      1501
   macro avg       0.80      0.79      0.80      1501
weighted avg       0.80      0.80      0.80      1501


Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[767  76]
 [220 438]]
              precision    recall  f1-score   support
```

```
           0       0.78      0.91      0.84       843
           1       0.85      0.67      0.75       658

    accuracy                           0.80      1501
   macro avg       0.81      0.79      0.79      1501
weighted avg       0.81      0.80      0.80      1501


Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[703 140]
 [183 475]]
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       843
           1       0.77      0.72      0.75       658

    accuracy                           0.78      1501
   macro avg       0.78      0.78      0.78      1501
weighted avg       0.78      0.78      0.78      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[738 105]
 [222 436]]
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       843
           1       0.81      0.66      0.73       658

    accuracy                           0.78      1501
   macro avg       0.79      0.77      0.77      1501
weighted avg       0.79      0.78      0.78      1501


Score is appended.

RandomForestClassifier()
```

```
Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[705 138]
 [219 439]]
              precision    recall  f1-score   support

           0       0.76      0.84      0.80       843
           1       0.76      0.67      0.71       658

    accuracy                           0.76      1501
   macro avg       0.76      0.75      0.75      1501
weighted avg       0.76      0.76      0.76      1501


Score is appended.

AdaBoostClassifier()

Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[747  96]
 [270 388]]
              precision    recall  f1-score   support

           0       0.73      0.89      0.80       843
           1       0.80      0.59      0.68       658

    accuracy                           0.76      1501
   macro avg       0.77      0.74      0.74      1501
weighted avg       0.76      0.76      0.75      1501


Score is appended.

CPU times: user 38.7 s, sys: 3.43 s, total: 42.1 s
Wall time: 39min 31s
```

[21]:
```
res2
```

[21]:
```
                  model  best_score  \
0  SVC                   0.802132
1  MultinomialNB         0.802798
2  logistics_regression  0.784810
3  K_Nearest_Neighbors   0.782145
4  random_forest         0.762159
5  AdaBoost              0.756163
```

```
                                          best_params
0  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 40, 'solver': 'lbfgs'}
3  {'n_neighbors': 100, 'weights': 'distance'}
4  {'criterion': 'gini', 'max_depth': None, 'n_estimators': 50}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

4.3 Setup 3: Models after removing numbers:

[22]:
```python
# Creating a df that is copy of the train set.
df = train.copy()
```

4.3.1 Removing numbers:

[23]:
```python
def cleaning_numbers(text):
    return re.sub('[0-9]+', '', text)

df['text'] = df['text'].apply(lambda text: cleaning_numbers(text))
```

4.3.2 Splitting data into Train and Test :

[24]:
```python
# Splitting data into Train and Test sets:
X = df['text']
y = df['target']
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 3)
```

4.3.3 Transforming dataset using TF-IDF Vectorizer:

[25]:
```python
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  60344
```

4.3.4 Results:

[26]:
```
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
↪created dictionary
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.


scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                          # Training the␣
↪model
    clf_pred = best_model.predict(X_test)                           # Predicting␣
↪the results
    print(confusion_matrix(y_test,clf_pred))                        # Printing␣
↪Confusion Matrix
    print(metrics.classification_report(y_test, clf_pred))          # Printing␣
↪Classification Report
    scores.append({                                                 # Appending␣
↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')

# Creating data frame with model, best scores and best params:
res3 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()

Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[732 111]
 [187 471]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       843
           1       0.81      0.72      0.76       658
```

```
    accuracy                           0.80      1501
   macro avg       0.80      0.79      0.80      1501
weighted avg       0.80      0.80      0.80      1501


Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[772  71]
 [236 422]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.83       843
           1       0.86      0.64      0.73       658

    accuracy                           0.80      1501
   macro avg       0.81      0.78      0.78      1501
weighted avg       0.81      0.80      0.79      1501


Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[706 137]
 [182 476]]
              precision    recall  f1-score   support

           0       0.80      0.84      0.82       843
           1       0.78      0.72      0.75       658

    accuracy                           0.79      1501
   macro avg       0.79      0.78      0.78      1501
weighted avg       0.79      0.79      0.79      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
```

```
[[746  97]
 [228 430]]
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       843
           1       0.82      0.65      0.73       658

    accuracy                           0.78      1501
   macro avg       0.79      0.77      0.77      1501
weighted avg       0.79      0.78      0.78      1501


Score is appended.

RandomForestClassifier()

Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[760  83]
 [254 404]]
              precision    recall  f1-score   support

           0       0.75      0.90      0.82       843
           1       0.83      0.61      0.71       658

    accuracy                           0.78      1501
   macro avg       0.79      0.76      0.76      1501
weighted avg       0.78      0.78      0.77      1501


Score is appended.

AdaBoostClassifier()

Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[684 159]
 [228 430]]
              precision    recall  f1-score   support

           0       0.75      0.81      0.78       843
           1       0.73      0.65      0.69       658

    accuracy                           0.74      1501
   macro avg       0.74      0.73      0.73      1501
weighted avg       0.74      0.74      0.74      1501
```

```
Score is appended.

CPU times: user 46.1 s, sys: 3.82 s, total: 49.9 s
Wall time: 44min
```

[27]:
```
res3
```

[27]:
```
                    model  best_score  \
0  SVC                     0.801466
1  MultinomialNB           0.795470
2  logistics_regression    0.787475
3  K_Nearest_Neighbors     0.783478
4  random_forest           0.775483
5  AdaBoost                0.742172

                                                  best_params
0  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 40, 'solver': 'lbfgs'}
3  {'n_neighbors': 80, 'weights': 'distance'}
4  {'criterion': 'gini', 'max_depth': None, 'n_estimators': 50}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

4.4 Setup 4: Models after removing repeating characters:

[28]:
```
# Creating a df that is copy of the train set.
df = train.copy()
```

4.4.1 Removing repeating characteres:

[29]:
```
tokens = (word_tokenize(i) for i in df.text)
df['text'] = df['text'].apply(nltk.word_tokenize)

pattern = re.compile(r'(.)\1*')

def reduce_sequence_word(word):
    return ''.join([match.group()[:2] if len(match.group()) > 2 else match.
↪group() for match in pattern.finditer(word)])

def reduce_sequence_tweet(tweet):
    return [reduce_sequence_word(word) for word in tweet]

df.text = df.text.apply(lambda tweet: reduce_sequence_tweet(tweet))
```

4.4.2 Splitting data into Train and Test :

[30]:
```
# Splitting data into Train and Test sets:
X = df['text'].astype(str)
y = df['target'].astype(str)
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
↪random_state = 3)
```

4.4.3 Transforming dataset using TF-IDF Vectorizer:

[31]:
```
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  62005
```

4.4.4 Results:

[32]:
```
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
↪created dictionary
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.


scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                          # Training the␣
↪model
    clf_pred = best_model.predict(X_test)                           # Predicting␣
↪the results
    print(confusion_matrix(y_test,clf_pred))                        # Printing␣
↪Confusion Matrix
```

```python
    print(metrics.classification_report(y_test, clf_pred))       # Printing␣
↪Classification Report
    scores.append({                                              # Appending␣
↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')

# Creating data frame with model, best scores and best params:
res4 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()

Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[732 111]
 [187 471]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       843
           1       0.81      0.72      0.76       658

    accuracy                           0.80      1501
   macro avg       0.80      0.79      0.80      1501
weighted avg       0.80      0.80      0.80      1501


Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[779  64]
 [233 425]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       843
           1       0.87      0.65      0.74       658

    accuracy                           0.80      1501
   macro avg       0.82      0.78      0.79      1501
weighted avg       0.81      0.80      0.80      1501
```

```
Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[702 141]
 [172 486]]
              precision    recall  f1-score   support

           0       0.80      0.83      0.82       843
           1       0.78      0.74      0.76       658

    accuracy                           0.79      1501
   macro avg       0.79      0.79      0.79      1501
weighted avg       0.79      0.79      0.79      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[750  93]
 [231 427]]
              precision    recall  f1-score   support

           0       0.76      0.89      0.82       843
           1       0.82      0.65      0.72       658

    accuracy                           0.78      1501
   macro avg       0.79      0.77      0.77      1501
weighted avg       0.79      0.78      0.78      1501


Score is appended.

RandomForestClassifier()

Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[755  88]
 [259 399]]
              precision    recall  f1-score   support

           0       0.74      0.90      0.81       843
           1       0.82      0.61      0.70       658
```

```
    accuracy                           0.77      1501
   macro avg       0.78      0.75      0.76      1501
weighted avg       0.78      0.77      0.76      1501


Score is appended.

AdaBoostClassifier()

Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[691 152]
 [231 427]]
              precision    recall  f1-score   support

           0       0.75      0.82      0.78       843
           1       0.74      0.65      0.69       658

    accuracy                           0.74      1501
   macro avg       0.74      0.73      0.74      1501
weighted avg       0.74      0.74      0.74      1501


Score is appended.

CPU times: user 1min 14s, sys: 4.19 s, total: 1min 18s
Wall time: 46min 15s
```

[33]:
```
res4
```

[33]:
```
                   model  best_score  \
0  SVC                    0.801466
1  MultinomialNB          0.802132
2  logistics_regression   0.791472
3  K_Nearest_Neighbors    0.784144
4  random_forest          0.768821
5  AdaBoost               0.744837

                                                           best_params
0  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 80, 'solver': 'lbfgs'}
3  {'n_neighbors': 80, 'weights': 'distance'}
4  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 150}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

4.5 Setup 5: Applying Stemming and Lemmatization:

[34]:
```
# Creating a df that is copy of the train set.
df = train.copy()
```

4.5.1 Applying Stemming:

[35]:
```
# Tokenizing tweets:
tokens = (word_tokenize(i) for i in df.text)
df['text'] = df['text'].apply(nltk.word_tokenize)

stemm = SnowballStemmer('english')
df['text'] = df['text'].apply(lambda x: [stemm.stem(y) for y in x])
```

4.5.2 Splitting data into Train and Test :

[36]:
```
# Splitting data into Train and Test sets:
X = df['text'].astype(str)
y = df['target'].astype(str)
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 3)
```

4.5.3 Transforming dataset using TF-IDF Vectorizer:

[37]:
```
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  57871
```

4.5.4 Results:

[38]:
```
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
 ↪created dictionary
```

```
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.


scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                    # Training the␣
↪model
    clf_pred = best_model.predict(X_test)                     # Predicting␣
↪the results
    print(confusion_matrix(y_test,clf_pred))                  # Printing␣
↪Confusion Matrix
    print(metrics.classification_report(y_test, clf_pred))    # Printing␣
↪Classification Report
    scores.append({                                           # Appending␣
↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')

# Creating data frame with model, best scores and best params:
res5 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()

Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[709 134]
 [184 474]]
              precision    recall  f1-score   support

           0       0.79      0.84      0.82       843
           1       0.78      0.72      0.75       658

    accuracy                           0.79      1501
   macro avg       0.79      0.78      0.78      1501
weighted avg       0.79      0.79      0.79      1501
```

```
Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[774  69]
 [232 426]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       843
           1       0.86      0.65      0.74       658

    accuracy                           0.80      1501
   macro avg       0.81      0.78      0.79      1501
weighted avg       0.81      0.80      0.79      1501


Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[695 148]
 [178 480]]
              precision    recall  f1-score   support

           0       0.80      0.82      0.81       843
           1       0.76      0.73      0.75       658

    accuracy                           0.78      1501
   macro avg       0.78      0.78      0.78      1501
weighted avg       0.78      0.78      0.78      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[745  98]
 [217 441]]
              precision    recall  f1-score   support

           0       0.77      0.88      0.83       843
           1       0.82      0.67      0.74       658
```

```
    accuracy                           0.79      1501
   macro avg       0.80      0.78      0.78      1501
weighted avg       0.79      0.79      0.79      1501


Score is appended.

RandomForestClassifier()

Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[752  91]
 [247 411]]
              precision    recall  f1-score   support

           0       0.75      0.89      0.82       843
           1       0.82      0.62      0.71       658

    accuracy                           0.77      1501
   macro avg       0.79      0.76      0.76      1501
weighted avg       0.78      0.77      0.77      1501


Score is appended.

AdaBoostClassifier()

Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[708 135]
 [233 425]]
              precision    recall  f1-score   support

           0       0.75      0.84      0.79       843
           1       0.76      0.65      0.70       658

    accuracy                           0.75      1501
   macro avg       0.76      0.74      0.75      1501
weighted avg       0.76      0.75      0.75      1501


Score is appended.

CPU times: user 1min 11s, sys: 3.68 s, total: 1min 14s
Wall time: 45min
```

[39]:
```
res5
```

[39]:
```
                     model  best_score  \
0  SVC                      0.788141
1  MultinomialNB            0.799467
2  logistics_regression     0.782811
3  K_Nearest_Neighbors      0.790140
4  random_forest            0.774817
5  AdaBoost                 0.754830

                                                         best_params
0  {'C': 10, 'gamma': 0.1, 'kernel': 'sigmoid'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 20, 'solver': 'lbfgs'}
3  {'n_neighbors': 80, 'weights': 'distance'}
4  {'criterion': 'gini', 'max_depth': None, 'n_estimators': 150}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

4.6 Setup 6: Models after removing all the features:

[40]:
```
# Creating a df that is copy of the train set.
df = train.copy()
```

4.6.1 Removing Punctuation:

[41]:
```
import string
string.punctuation

punctuations_list = string.punctuation
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)

df['text'] = df['text'].apply(lambda x: cleaning_punctuations(x))
```

4.6.2 Removing Stop-words:

[42]:
```
sw = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if␣
 ↪word not in (sw)]))
```

4.6.3 Removing Numbers:

[43]:
```
def cleaning_numbers(text):
    return re.sub('[0-9]+', '', text)

df['text'] = df['text'].apply(lambda text: cleaning_numbers(text))
```

4.6.4 Removing repeating characters:

[44]:
```
tokens = (word_tokenize(i) for i in df.text)
df['text'] = df['text'].apply(nltk.word_tokenize)

pattern = re.compile(r'(.)\1*')

def reduce_sequence_word(word):
    return ''.join([match.group()[:2] if len(match.group()) > 2 else match.
↪group() for match in pattern.finditer(word)])

def reduce_sequence_tweet(tweet):
    return [reduce_sequence_word(word) for word in tweet]

df.text = df.text.apply(lambda tweet: reduce_sequence_tweet(tweet))
```

4.6.5 Applying Stemming and Lemmatization:

[45]:
```
stemm = SnowballStemmer('english')
df['text'] = df['text'].apply(lambda x: [stemm.stem(y) for y in x])
```

4.6.6 Splitting data into Train and Test :

[46]:
```
# Splitting data into Train and Test sets:
X = df['text'].astype(str)
y = df['target'].astype(str)
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
↪random_state = 3)
```

4.6.7 Transforming dataset using TF-IDF Vectorizer:

[47]:
```
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
```

```
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  46081
```

4.6.8 Results:

[48]:
```
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
↪created dictionary
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.



scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                         # Training the␣
↪model
    clf_pred = best_model.predict(X_test)                          # Predicting␣
↪the results
    print(confusion_matrix(y_test,clf_pred))                       # Printing␣
↪Confusion Matrix
    print(metrics.classification_report(y_test, clf_pred))         # Printing␣
↪Classification Report
    scores.append({                                                # Appending␣
↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')

# Creating data frame with model, best scores and best params:
res6 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()
```

```
Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[713 130]
 [179 479]]
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       843
           1       0.79      0.73      0.76       658

    accuracy                           0.79      1501
   macro avg       0.79      0.79      0.79      1501
weighted avg       0.79      0.79      0.79      1501


Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[760  83]
 [222 436]]
              precision    recall  f1-score   support

           0       0.77      0.90      0.83       843
           1       0.84      0.66      0.74       658

    accuracy                           0.80      1501
   macro avg       0.81      0.78      0.79      1501
weighted avg       0.80      0.80      0.79      1501


Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[701 142]
 [180 478]]
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       843
           1       0.77      0.73      0.75       658

    accuracy                           0.79      1501
   macro avg       0.78      0.78      0.78      1501
```

```
weighted avg       0.78      0.79      0.78      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[730 113]
 [212 446]]
              precision    recall  f1-score   support

           0       0.77      0.87      0.82       843
           1       0.80      0.68      0.73       658

    accuracy                           0.78      1501
   macro avg       0.79      0.77      0.78      1501
weighted avg       0.78      0.78      0.78      1501


Score is appended.

RandomForestClassifier()

Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[742 101]
 [231 427]]
              precision    recall  f1-score   support

           0       0.76      0.88      0.82       843
           1       0.81      0.65      0.72       658

    accuracy                           0.78      1501
   macro avg       0.79      0.76      0.77      1501
weighted avg       0.78      0.78      0.77      1501


Score is appended.

AdaBoostClassifier()

Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[747  96]
 [270 388]]
              precision    recall  f1-score   support
```

```
           0       0.73      0.89      0.80       843
           1       0.80      0.59      0.68       658

    accuracy                           0.76      1501
   macro avg       0.77      0.74      0.74      1501
weighted avg       0.76      0.76      0.75      1501


Score is appended.

CPU times: user 1min 1s, sys: 2.34 s, total: 1min 3s
Wall time: 36min 49s
```

[49]:

```
res6
```

[49]:

```
                   model  best_score  \
0  SVC                    0.794137
1  MultinomialNB          0.796802
2  logistic_regression    0.785476
3  K_Nearest_Neighbors    0.783478
4  random_forest          0.778814
5  AdaBoost               0.756163

                                                       best_params
0  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 20, 'solver': 'liblinear'}
3  {'n_neighbors': 50, 'weights': 'distance'}
4  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 150}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

4.7 Setup 7: Models without removing any setup:

[50]:

```
# Creating a df that is copy of the train set.
df = train.copy()
```

4.7.1 Splitting data into Train and Test :

[51]:

```
# Splitting data into Train and Test sets:
X = df['text']
y = df['target']
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 3)
```

4.7.2 Transforming dataset using TF-IDF Vectorizer:

[52]:
```
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  62117
```

4.7.3 Results:

[53]:
```
%%time

# implemented GridSearchCV for four models using a loop and a previously␣
↪created dictionary
# in the created variable 'scores', results are stored for each model such as:␣
↪model, best_score and best_params.


scores = []

for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1, verbose=1) #␣
↪Using Cross Validation of 5 and n_jobs=-1 for fast training by using all the␣
↪processors
    print(mp['model'])
    print('\nTraining the model...')
    best_model = clf.fit(X_train, y_train)                     # Training the␣
↪model
    clf_pred = best_model.predict(X_test)                      # Predicting␣
↪the results
    print(confusion_matrix(y_test,clf_pred))                   # Printing␣
↪Confusion Matrix
    print(metrics.classification_report(y_test, clf_pred))     # Printing␣
↪Classification Report
    scores.append({                                            # Appending␣
↪results to 'scores' list
        'model' : model_name,
        'best_score' : best_model.score(X_test, y_test),
        'best_params' : clf.best_params_
    })
    print('\nScore is appended.\n')
```

```python
# Creating data frame with model, best scores and best params:
res7 = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
```

```
SVC()

Training the model…
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[[736 107]
 [187 471]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       843
           1       0.81      0.72      0.76       658

    accuracy                           0.80      1501
   macro avg       0.81      0.79      0.80      1501
weighted avg       0.81      0.80      0.80      1501


Score is appended.

MultinomialNB()

Training the model…
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[[777  66]
 [233 425]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       843
           1       0.87      0.65      0.74       658

    accuracy                           0.80      1501
   macro avg       0.82      0.78      0.79      1501
weighted avg       0.81      0.80      0.80      1501


Score is appended.

LogisticRegression()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[705 138]
 [180 478]]
              precision    recall  f1-score   support
```

```
           0       0.80      0.84      0.82       843
           1       0.78      0.73      0.75       658

    accuracy                           0.79      1501
   macro avg       0.79      0.78      0.78      1501
weighted avg       0.79      0.79      0.79      1501


Score is appended.

KNeighborsClassifier()

Training the model…
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[[753  90]
 [230 428]]
              precision    recall  f1-score   support

           0       0.77      0.89      0.82       843
           1       0.83      0.65      0.73       658

    accuracy                           0.79      1501
   macro avg       0.80      0.77      0.78      1501
weighted avg       0.79      0.79      0.78      1501


Score is appended.

RandomForestClassifier()

Training the model…
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[[764  79]
 [277 381]]
              precision    recall  f1-score   support

           0       0.73      0.91      0.81       843
           1       0.83      0.58      0.68       658

    accuracy                           0.76      1501
   macro avg       0.78      0.74      0.75      1501
weighted avg       0.78      0.76      0.75      1501


Score is appended.

AdaBoostClassifier()
```

```
Training the model…
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[[718 125]
 [260 398]]
              precision    recall  f1-score   support

           0       0.73      0.85      0.79       843
           1       0.76      0.60      0.67       658

    accuracy                           0.74      1501
   macro avg       0.75      0.73      0.73      1501
weighted avg       0.75      0.74      0.74      1501


Score is appended.

CPU times: user 58 s, sys: 4.29 s, total: 1min 2s
Wall time: 44min 58s
```

[54]:

```
res7
```

[54]:

```
                  model  best_score  \
0  SVC                   0.804131
1  MultinomialNB         0.800799
2  logistic_regression   0.788141
3  K_Nearest_Neighbors   0.786809
4  random_forest         0.762825
5  AdaBoost              0.743504

                                                          best_params
0  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
1  {'alpha': 0.5, 'fit_prior': True}
2  {'C': 40, 'solver': 'lbfgs'}
3  {'n_neighbors': 80, 'weights': 'distance'}
4  {'criterion': 'entropy', 'max_depth': None, 'n_estimators': 100}
5  {'learning_rate': 0.5, 'n_estimators': 150}
```

5 Creating Submission file:

It can be observed that **Setup-1 and 7** is performing best for SVM model. **Setup 6** will be used. Let's just train this model with 100% training data. This model will be used for predicting test file.

[55]:

```
# Creating a df that is copy of the train set.
df = train.copy()
```

5.0.1 Removing Punctuation:

[56]:
```python
import string
string.punctuation

punctuations_list = string.punctuation
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)

df['text'] = df['text'].apply(lambda x: cleaning_punctuations(x))
```

5.0.2 Removing Stop-words:

[57]:
```python
sw = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (sw)]))
```

5.0.3 Removing Numbers:

[58]:
```python
def cleaning_numbers(text):
    return re.sub('[0-9]+', '', text)

df['text'] = df['text'].apply(lambda text: cleaning_numbers(text))
```

5.0.4 Removing repeating characters:

[59]:
```python
tokens = (word_tokenize(i) for i in df.text)
df['text'] = df['text'].apply(nltk.word_tokenize)

pattern = re.compile(r'(.)\1*')

def reduce_sequence_word(word):
    return ''.join([match.group()[:2] if len(match.group()) > 2 else match.group() for match in pattern.finditer(word)])

def reduce_sequence_tweet(tweet):
    return [reduce_sequence_word(word) for word in tweet]

df.text = df.text.apply(lambda tweet: reduce_sequence_tweet(tweet))
```

5.0.5 Applying Stemming and Lemmatization:

[60]:
```
stemm = SnowballStemmer('english')
df['text'] = df['text'].apply(lambda x: [stemm.stem(y) for y in x])
```

5.0.6 Splitting data into Train and Test :

[61]:
```
# Splitting data into Train and Test sets:
X = df['text'].astype(str)
y = df['target'].astype(str)
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size = 0.2,␣
↪random_state = 3)
```

5.0.7 Transforming dataset using TF-IDF Vectorizer:

[62]:
```
# Extracting features using TF-IDF (1,2) - unigrams and bigrams
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

# Transforming the data using TD-IDF Vectorizer
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

```
No. of feature_words:  46081
```

5.0.8 SVC model:

[63]:
```
svc = SVC()
hyperParam = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01], 'kernel':␣
↪['rbf','linear','poly','sigmoid']}

gsv = GridSearchCV(svc,hyperParam,cv=5,verbose=1,n_jobs=-1)  # Using Cross␣
↪Validation of 5 and n_jobs=-1 for fast training by using all the processors
best_model = gsv.fit(X_train, y_train)                        # Training model␣
↪with X_train and y_train
svc_pred = best_model.predict(X_test)                         # Predicting the␣
↪results

print("Best HyperParameter: ", gsv.best_params_)
```

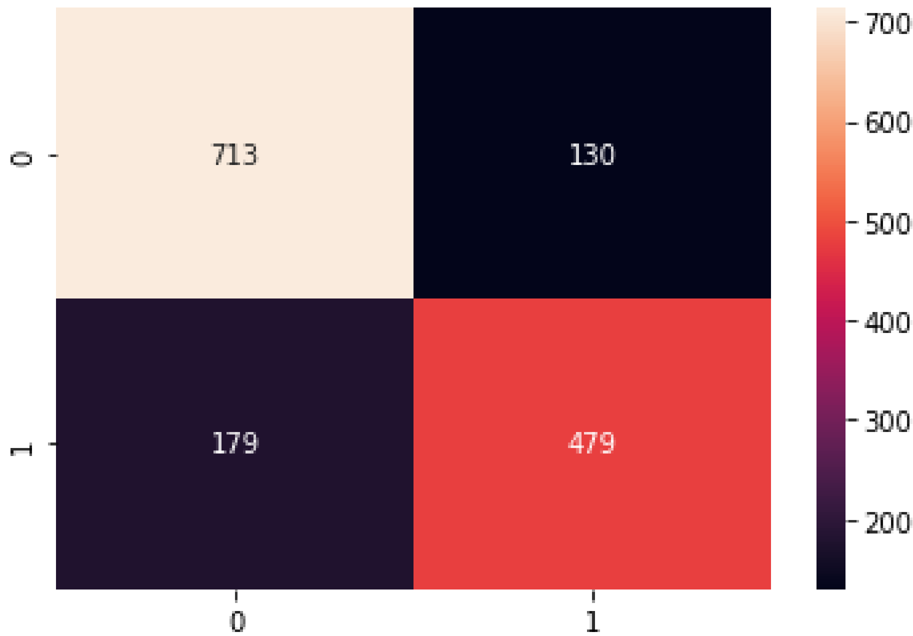
```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best HyperParameter:  {'C': 1, 'gamma': 1, 'kernel': 'linear'}
```

```
[96]: from sklearn.metrics import accuracy_score
      print("The accuracy is:", accuracy_score(y_test,svc_pred))
```

```
The accuracy is: 0.7941372418387741
```

```
[97]: cv=confusion_matrix(y_test,svc_pred)
      sns.heatmap(cv,annot=True, fmt='g')
```

```
[97]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc7ede82f50>
```



5.0.9 Submission file:

```
[95]: print(svc_pred)
      print(type(svc_pred))

      my_array = svc_pred
      print(len(my_array))
      submission = pd.DataFrame({'Ground Truth':y_test,'Predicted':my_array})
      submission['id'] = test['id']
      submission = submission[['id','Ground Truth','Predicted']]
      submission.to_csv('submission.csv', index=False)
      submission
```

```
['1' '0' '1' … '0' '0' '1']
<class 'numpy.ndarray'>
```

```
1501
```

[95]:

```
            id Ground Truth Predicted
7090 NaN      1            1
1275  4193.0  0            0
5721 NaN      1            1
2308  7710.0  0            0
244   780.0   0            0
…     …  ..           ..
1785  6028.0  0            0
485   1578.0  0            0
3756 NaN      0            0
1604  5419.0  1            0
2664  8895.0  1            1

[1501 rows x 3 columns]
```