NLP Project MS1

Mohamed Ashraf

May 26, 2023

1 Milestone 2 Description

In this Milestone we will be using Feature Extraction to get different feature sets

1. Bag of Words
2. TF-IDF
3. Word2Vec

After that we will start testing different Models and compare their overall F Scores to check which model works the best with the given set of features. The models we will be testing are

1. Logistic Regression Model
2. Support Vector Machine Model
3. XGBoost

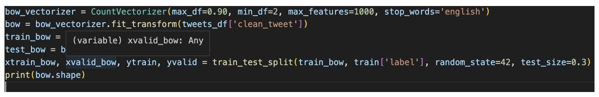
Finally we will tune our Models to adjust the given parameters and compare our results to pick the best Model to be used

2 Preparing Feature sets

Preprocessed data must be transformed into features in order to be analysed. Depending on their intended use, text features can be created using a variety of methods. Word embeddings, TF-IDF, and the Bag of Words

2.1 Bag of Words Feature Set

Consider a Corpus C of D documents d1,d2. . . . .dD and N unique tokens extracted out of the corpus C. The N tokens (words) will form a dictionary and the size of the bag-of-words matrix M will be given by D X N. Each row in the matrix M contains the frequency of tokens in document D(i).



```
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')
bow = bow_vectorizer.fit_transform(tweets_df['clean_tweet'])
train_bow =   (variable) xvalid_bow: Any
test_bow = b
xtrain_bow, xvalid_bow, ytrain, yvalid = train_test_split(train_bow, train['label'], random_state=42, test_size=0.3)
print(bow.shape)
```

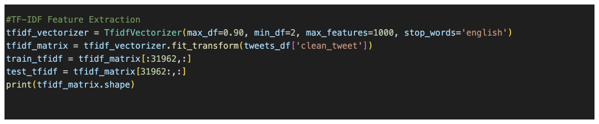
Figure 1: Bag of Words Feature Extraction

2.2 TF-IDF Feature set

The TF-IDF algorithm diminishes the importance of common terms by assigning them lower weights, and elevates the significance of words that are infrequent across the entire corpus but appear frequently in a small subset of documents.

1. TF = (Number of times term t appears in a document)/(Number of terms in the document)

2. IDF = log(N/n), where, N is the number of documents and n is the number of documents a term t has appeared in.
3. TF-IDF = TF*IDF

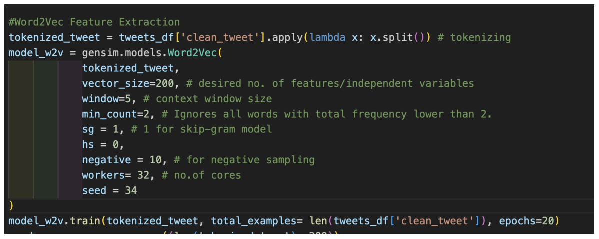


```
#TF-IDF Feature Extraction
tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(tweets_df['clean_tweet'])
train_tfidf = tfidf_matrix[:31962,:]
test_tfidf = tfidf_matrix[31962:,:]
print(tfidf_matrix.shape)
```

Figure 2: TF-IDF Feature Extraction

2.3 Word2Vec Feature set

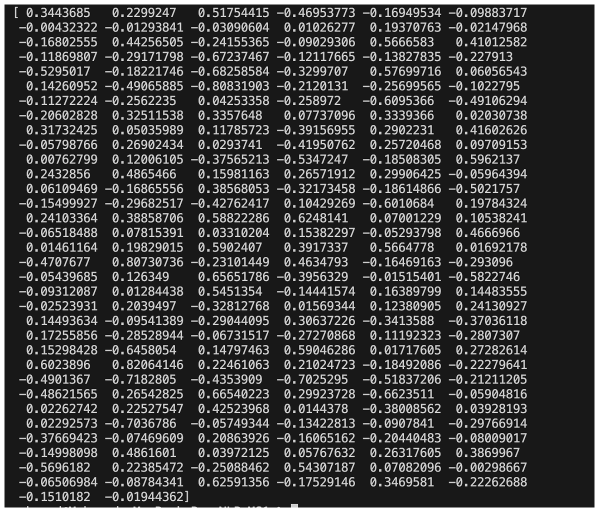
Word embeddings focus on the words representation by transforming them into vectorized forms. The main goal of word embeddings is to convert complex, high-dimensional word features into compact, low-dimensional feature vectors while maintaining contextual similarity within the text corpus. This remarkable approach enables word embeddings to perform various tasks.



```
#Word2Vec Feature Extraction
tokenized_tweet = tweets_df['clean_tweet'].apply(lambda x: x.split()) # tokenizing
model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            vector_size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2, # Ignores all words with total frequency lower than 2.
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 32, # no.of cores
            seed = 34
)
model_w2v.train(tokenized_tweet, total_examples= len(tweets_df['clean_tweet']), epochs=20)
```

Figure 3: Word2Vec Feature Extraction

We will try to get the most simmilar words using our trained W2V model by using cosine similarity between word vectors to find words that can come in the context of the given word

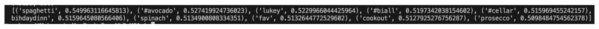


```
[ 0.3443685   0.2299247   0.51754415 -0.46953773 -0.16949534 -0.09883717
 -0.00432322 -0.01293841 -0.03090604  0.01026277  0.19370763 -0.02147968
 -0.16802555  0.44256505 -0.24155365 -0.09029306  0.5666583   0.41012582
 -0.11869807 -0.29171798 -0.67237467 -0.12117665 -0.13827835 -0.227913
 -0.5295017  -0.18221746 -0.68258584 -0.3299707   0.57699716  0.06056543
  0.14260952 -0.49065885 -0.80831903 -0.2120131  -0.25699565 -0.1022795
 -0.11272224 -0.2562235   0.04253358 -0.258972   -0.6095366  -0.49106294
 -0.20602828  0.32511538  0.3357648   0.07737096  0.3339366   0.02030738
  0.31732425  0.05035989  0.11785723 -0.39156955  0.2902231   0.41602626
 -0.05798766  0.26902434  0.0293741  -0.41950762  0.25720468  0.09709153
  0.00762799  0.12006105 -0.37565213 -0.5347247  -0.18508305  0.5962137
  0.2432856   0.4865466   0.15981163  0.26571912  0.29906425 -0.05964394
  0.06109469 -0.16865556  0.38568053 -0.32173458 -0.18614866 -0.5021757
 -0.15499927 -0.29682517 -0.42762417  0.10429269 -0.6010684   0.19784324
  0.24103364  0.38858706  0.58822286  0.6248141   0.07001229  0.10538241
 -0.06518488  0.07815391  0.03310204  0.15382297 -0.05293798  0.4666966
  0.01461164  0.19829015  0.5902407   0.3917337   0.5664778   0.01692178
 -0.4707677   0.80730736 -0.23101449  0.4634793  -0.16469163 -0.293096
 -0.05439685  0.126349    0.65651786 -0.3956329  -0.01515401 -0.5822746
 -0.09312087  0.01284438  0.5451354  -0.14441574  0.16389799  0.14483555
 -0.02523931  0.2039497  -0.32812768  0.01569344  0.12380905  0.24130927
  0.14493634 -0.09541389 -0.29044095  0.30637226 -0.3413588  -0.37036118
  0.17255856 -0.28528944 -0.06731517 -0.27270868  0.11192323 -0.2807307
  0.15298428 -0.6458054   0.14797463  0.59046286  0.01717605  0.27282614
  0.6023896   0.82064146  0.22461063  0.21024723 -0.18492086 -0.22279641
 -0.4901367  -0.7182805  -0.4353909  -0.7025295  -0.51837206 -0.21211205
 -0.48621565  0.26542825  0.66540223  0.29923728 -0.6623511  -0.05904816
  0.02262742  0.22527547  0.42523968  0.0144378  -0.38008562  0.03928193
  0.02292573 -0.7036786  -0.05749344 -0.13422813 -0.0907841  -0.29766914
 -0.37669423 -0.07469609  0.20863926 -0.16065162 -0.20440483 -0.08009017
 -0.14998098  0.4861601   0.03972125  0.05767632  0.26317605  0.3869967
 -0.5696182   0.22385472 -0.25088462  0.54307187  0.07082096 -0.00298667
 -0.06506984 -0.08784341  0.62591356 -0.17529146  0.3469581  -0.22262688
 -0.1510182  -0.01944362]
```

Figure 4: Vector Representation of word "Dinner"

3 Evaluation Metric

The F1 score is employed as an evaluation measure, which combines Precision and Recall in a weighted average. As a result, it considers both false positives and false negatives when assessing performance.



```
[('spaghetti', 0.549963116645813), ('#avocado', 0.527419924736023), ('lukey', 0.5229966044425964), ('#biall', 0.5197342038154602), ('#cellar', 0.515969455242157),
bihdaydinn', 0.5159645080566406), ('spinach', 0.5134900808334351), ('fav', 0.5132644772529602), ('cookout', 0.5127925276756287), ('prosecco', 0.5098484754562378)]
```

Figure 5: Most simmilar words to the word ”Dinner”

1. True Positives (TP) - These are the correctly predicted positive values, which means that the actual class is ”yes” and the predicted class is also ”yes.”
2. True Negatives (TN) - These are the correctly predicted negative values, which means that the actual class is ”no” and the predicted class is also ”no.”
3. False Positives (FP) - When the actual class is ”no” and the predicted class is ”yes.”
4. False Negatives (FN) - When the actual class is ”yes” but the predicted class is ”no.”

- **Precision** $= \frac{TP}{TP+FP}$
- **Recall** $= \frac{TP}{TP+FN}$
- **F1 Score** $= \frac{2\cdot(Recall\cdot Precision)}{Recall+Precision}$

4 Models Testing

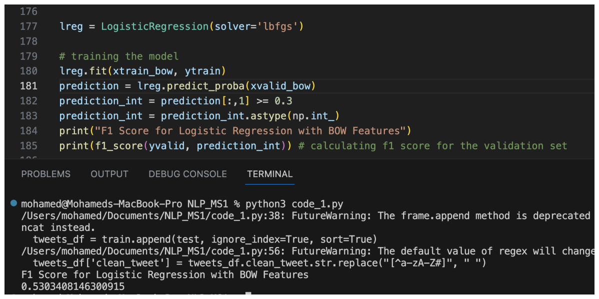
In this section, our objective is to train various models using the extracted set of features. The aim is to identify the feature set that achieves the highest F1 Score for each model.

4.1 Logistic Regression Model

Logistic Regression is a classification technique that enables the prediction of binary outcomes based on a given set of independent variables. The algorithm is used to determine whether an outcome will be a binary choice such as Yes or No

4.1.1 Logistic Regression - Bag of Words

We will fit logistic regression model on the Bag-of-Words (BoW) features.
We have achieved an **F1 Score** of 0.5303408146300915 using TF-IDF with Logistic



```python

lreg = LogisticRegression(solver='lbfgs')

# training the model
lreg.fit(xtrain_bow, ytrain)
prediction = lreg.predict_proba(xvalid_bow)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int_)
print("F1 Score for Logistic Regression with BOW Features")
print(f1_score(yvalid, prediction_int)) # calculating f1 score for the validation set
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method is deprecated
ncat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex will change
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
F1 Score for Logistic Regression with BOW Features
0.5303408146300915
```

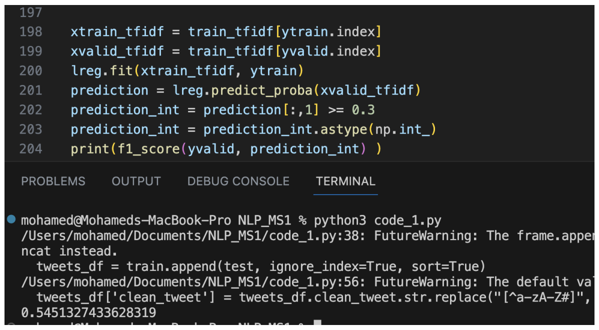
Figure 6: Logistic Regression with BOW

4.1.2 Logistic Regression - TF-IDF

We will fit a logistic regression model on TF-IDF features.
We have achieved an **F1 Score** of 0.5451327433628319 using TF-IDF with Logistic Regression.

4.1.3 Logistic Regression - Word2Vec

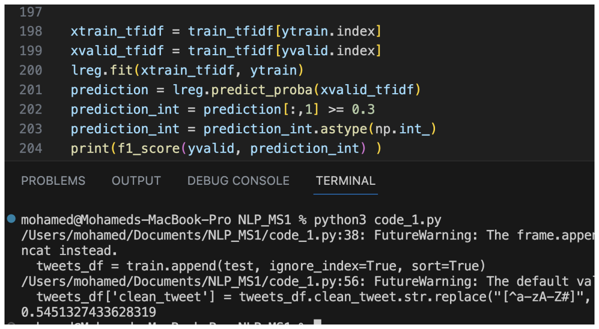
We will fit logistic regression model on Word2Vec features.
We have achieved an **F1 Score** of 0.6150442477876106 using Word2Vec with Logistic Regression.



```
xtrain_tfidf = train_tfidf[ytrain.index]
xvalid_tfidf = train_tfidf[yvalid.index]
lreg.fit(xtrain_tfidf, ytrain)
prediction = lreg.predict_proba(xvalid_tfidf)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int_)
print(f1_score(yvalid, prediction_int) )
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.apper
ncat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default va
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]",
0.5451327433628319
```

Figure 7: Logistic Regression with TF-IDF



```
xtrain_tfidf = train_tfidf[ytrain.index]
xvalid_tfidf = train_tfidf[yvalid.index]
lreg.fit(xtrain_tfidf, ytrain)
prediction = lreg.predict_proba(xvalid_tfidf)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int_)
print(f1_score(yvalid, prediction_int) )
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.apper
ncat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default va
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]",
0.5451327433628319
```

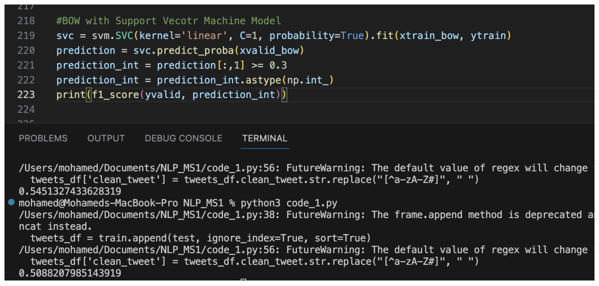
Figure 8: Logistic Regression with Word2Vec

4.2 Support Vector Machine (SVM)

SVM is primarily employed in solving classification problems. In SVM, data items are represented as points in a multi-dimensional space, where each feature corresponds to a specific coordinate. The objective is to classify the data points into different classes based on their positions in this space.

4.2.1 Support Vector Machine - Bag of Words

We will fit Support Vector Machine model on the Bag-of-Words (BoW) features.
We have achieved an **F1 Score** of 0.5092936802973977 using Bag-of-Words with SVM



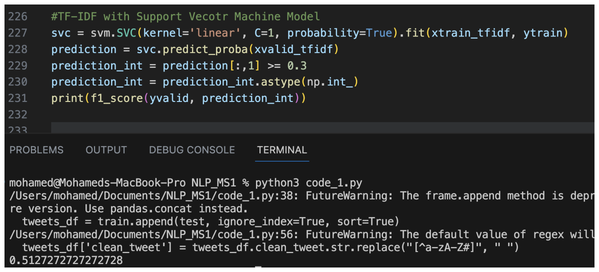
```
#BOW with Support Vecotr Machine Model
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_bow, ytrain)
prediction = svc.predict_proba(xvalid_bow)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int_)
print(f1_score(yvalid, prediction_int))
```

```
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex will change
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.5451327433628319
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method is deprecated a
ncat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex will change
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.5088207985143919
```

Figure 9: SVM with BOW

4.2.2 Support Vector Machine - TF-IDF

We will fit a Support Vector Machine model on TF-IDF features.
We have achieved an **F1 Score** of 0.5114155251141552 using TF-IDF with SVM.



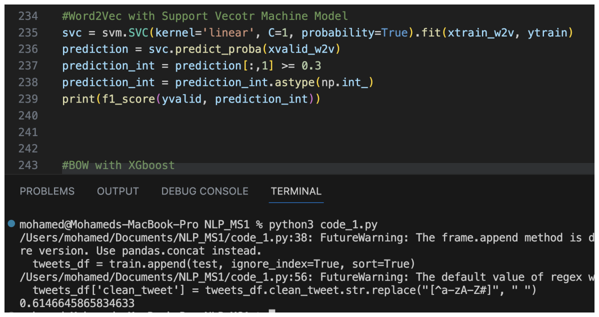
```python
#TF-IDF with Support Vecotr Machine Model
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_tfidf, ytrain)
prediction = svc.predict_proba(xvalid_tfidf)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int_)
print(f1_score(yvalid, prediction_int))
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method is depr
re version. Use pandas.concat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex will
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.5127272727272728
```

Figure 10: SVM with TF-IDF

4.2.3 Support Vector Machine - Word2Vec

We will fit Support Vector Machine model on Word2Vec features.
We have achieved an **F1 Score** of 0.6174980872226473 using Word2Vec with SVM.



```python
#Word2Vec with Support Vecotr Machine Model
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_w2v, ytrain)
prediction = svc.predict_proba(xvalid_w2v)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int_)
print(f1_score(yvalid, prediction_int))



#BOW with XGboost
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method is d
re version. Use pandas.concat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex w
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.6146645865834633
```

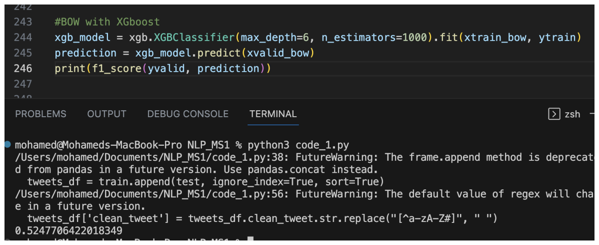
Figure 11: SVM with Word2Vec

4.3 XGBoost

XGBoost is an enhanced algorithm of gradient boosting algorithm is t has ability to handle both classification and regression problems. It achieves this by optimizing an objective function through an iterative process of adding new models that minimize the errors of the previous models.

4.3.1 XGBoost - Bag of Words

We will fit XGBoost model on the Bag-of-Words (BoW) features.
We have achieved an **F1 Score** of 0.5130687318489837 using Bag-of-Words with XGBoost



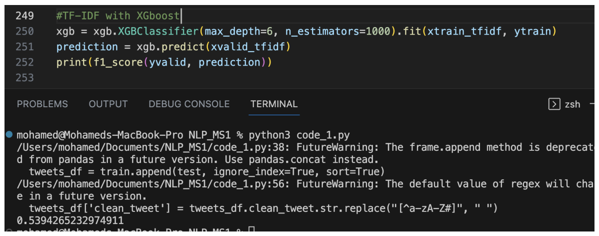
```python
#BOW with XGboost
xgb_model = xgb.XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_bow, ytrain)
prediction = xgb_model.predict(xvalid_bow)
print(f1_score(yvalid, prediction))
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method is deprecat
d from pandas in a future version. Use pandas.concat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex will cha
e in a future version.
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.5247706422018349
```

Figure 12: XGBoost with BOW

4.3.2 XGBoost - TF-IDF

We will fit a XGBoost model on TF-IDF features.
We have achieved an **F1 Score** of 0.5185891325071497 using TF-IDF with XGBoost.



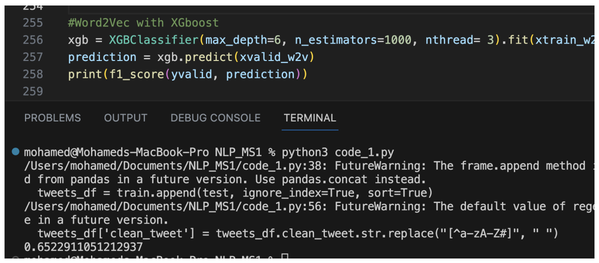
```
#TF-IDF with XGboost
xgb = xgb.XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_tfidf, ytrain)
prediction = xgb.predict(xvalid_tfidf)
print(f1_score(yvalid, prediction))
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method is deprecat
d from pandas in a future version. Use pandas.concat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of regex will cha
e in a future version.
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.5394265232974911
```

Figure 13: XGBoost with TF-IDF

4.3.3 XGBoost - Word2Vec

We will fit XGBoost model on Word2Vec features.
We have achieved an **F1 Score** of 0.6588447653429602 using Word2Vec with XGBoost.



```
#Word2Vec with XGboost
xgb = XGBClassifier(max_depth=6, n_estimators=1000, nthread= 3).fit(xtrain_w2
prediction = xgb.predict(xvalid_w2v)
print(f1_score(yvalid, prediction))
```

```
mohamed@Mohameds-MacBook-Pro NLP_MS1 % python3 code_1.py
/Users/mohamed/Documents/NLP_MS1/code_1.py:38: FutureWarning: The frame.append method 
d from pandas in a future version. Use pandas.concat instead.
  tweets_df = train.append(test, ignore_index=True, sort=True)
/Users/mohamed/Documents/NLP_MS1/code_1.py:56: FutureWarning: The default value of reg
e in a future version.
  tweets_df['clean_tweet'] = tweets_df.clean_tweet.str.replace("[^a-zA-Z#]", " ")
0.6522911051212937
```

Figure 14: XGBoost with Word2Vec

5 Fine Tuning

XGBoost with Word2Vec model has the best F Score so far.we will try to tune it to try to get the best Precision out of this model.

Steps that will be followed

- Tune parameters such as maxDepth, minChildWeight, subsample, colsampleBytree
- Tune the learning rate.
- Finally, tune gamma to avoid overfitting.

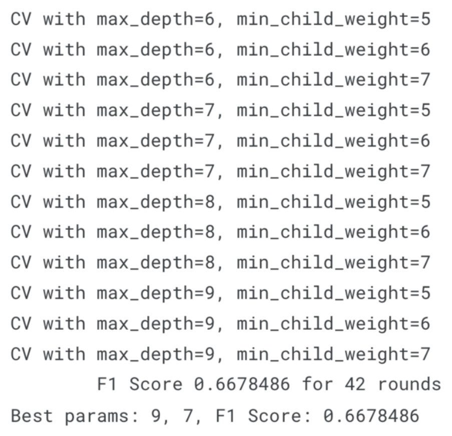
5.1 Tune maxDepth and minChildWeight

We will tune each parameter and choose the values that produces best F1 Score and update the params given to our model in order to achieve better precision.
we will do the same Tuning process to all of our parameters in order to determine the best values to be used

5.2 Tune Learning Rate

```
max_f1 = 0.0
best_params = None
for eta in [0.3, 0.2, 0.1, 0.05, 0.01, 0.005]:
    print("CV with eta={}".format(eta))
    # Update ETA
    params['eta'] = eta

    # Run CV
```



```
CV with max_depth=6, min_child_weight=5
CV with max_depth=6, min_child_weight=6
CV with max_depth=6, min_child_weight=7
CV with max_depth=7, min_child_weight=5
CV with max_depth=7, min_child_weight=6
CV with max_depth=7, min_child_weight=7
CV with max_depth=8, min_child_weight=5
CV with max_depth=8, min_child_weight=6
CV with max_depth=8, min_child_weight=7
CV with max_depth=9, min_child_weight=5
CV with max_depth=9, min_child_weight=6
CV with max_depth=9, min_child_weight=7
	F1 Score 0.6678486 for 42 rounds
Best params: 9, 7, F1 Score: 0.6678486
```

Figure 15: XGBoost with Word2Vec Tuning

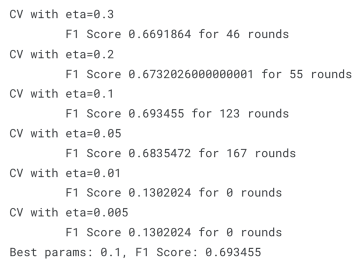
```
    cv_results = xgb.cv(
        params,
        dtrain,
        feval=custom_eval,
        num_boost_round=1000,
        maximize=True,
        seed=16,
        nfold=5,
        early_stopping_rounds=20
    )

    # Finding the best F1 Score
    mean_f1 = cv_results['test-f1_score-mean'].max()
    boost_rounds = cv_results['test-f1_score-mean'].idxmax()
    print("\tF1 Score {} for {} rounds".format(mean_f1, boost_rounds))

    if mean_f1 > max_f1:
        max_f1 = mean_f1
        best_params = eta

print("Best params: {}, F1 Score: {}".format(best_params, max_f1))
```

5.2.1 Learning Rate Tuning Result



```
CV with eta=0.3
	F1 Score 0.6691864 for 46 rounds
CV with eta=0.2
	F1 Score 0.6732026000000001 for 55 rounds
CV with eta=0.1
	F1 Score 0.693455 for 123 rounds
CV with eta=0.05
	F1 Score 0.6835472 for 167 rounds
CV with eta=0.01
	F1 Score 0.1302024 for 0 rounds
CV with eta=0.005
	F1 Score 0.1302024 for 0 rounds
Best params: 0.1, F1 Score: 0.693455
```

Figure 16: Learning Rate Results

6 Results

Based on our findings Word2Vec set of features works the best with XGBoost Model giving accuracy of 65 before fine tuning. and after tuning the parameters we acheived higher score of 69 that can be used for our project to make sentiment analysis and make classification for the tweets dataset we use