A Project Report

On

Word Embedding in Text Classification

BY

SHIVANG SINGH

2018A7PS0115H

Under the supervision of

Dr. LOV KUMAR

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F266: STUDY PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(November 2020)

ACKNOWLEDGMENT

This project wouldn’t have been possible without the support from Bits Pilani and I am very grateful to them for granting me with this opportunity.

I would like to express my sincerest gratitude to my mentor, Dr. Lov Kumar, for his guided support throughout the course of the project. I would like to thank my Mentor for providing me with such a wonderful opportunity to work on real life data and get hands on experience on such a fast-growing topic.

I am also grateful to my colleagues and friends who have helped me in making great progress throughout the project. I would also like to thank my parents for their personal support and motivation which has helped me to continue all the way.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "Word Embedding in Text Classification" submitted by Mr. SHIVANG SINGH (ID No. 2018A7PS0115H) in partial fulfillment of the requirements of the course CS F266, Study Project Course, embodies the work done by him under my supervision and guidance.

Date: 23-11-2020 **(Dr. Lov Kumar)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

Due to an increased demand for an accurate vectored representation of textual data in the field of Natural Language Processing, Word Embedding has become more relevant. Important research areas such as Speech Recognition, Text Analysis, Feedback Forums, etc indicate the need for a structured approach to handle and prepare textual data for Machine Learning Algorithms.

This Project broadly deals with the “Classification of Feedback Tweets for various Airlines based on their Sentiment”. The aim is to classify tweets into 3 Categories namely positive, negative or neutral based on the textual information provided through the tweet.

The Project focuses on recognizing the most suitable Word Embedding Technique, Approach to handle Imbalanced Data, Feature Selection and Classification Technique for the given data set to arrive at the most suitable Framework for the solving the problem.

TABLE OF CONTENTS

MODEL FRAMEWORK

1. The Dataset consists of the Classification label (airline_sentiment) and the Text field(text), which specifies the body of the tweet.

2. Apply various **Word Embedding Techniques** such as glove, W2V, CountVectorizer, TFIDF, skip-gram, fasttext, bert, gptmodel, etc to the text field of the dataset.

3. Identifying the Imbalance in the dataset and handling it using various **SMOTE** (Synthetic Minority Over-Sampling Technique) techniques such as SVSMOTE, SMOTE, Borderline SMOTE 1 and 2, SMOTENC.

4. Selecting appropriate features using various **Feature Selection Techniques** such as Correlation analysis, Principal Component Analysis, Significant Features.

5. Applying various **Deep Learning Algorithms** and analyzing their performances to gauge the best combination of Word Embedding, SMOTE Technique, Feature Selection Technique and Deep Learning Algorithm for the given dataset.

DATASET DESCRIPTION

- The Dataset has been obtained through kaggle titled “Twitter US Airline Sentiment”. It provides information about how travelers in February 2015 expressed their sentiments on Twitter. The aim is to analyse the sentiments of a tweet based on its contents.

Twitter US Airline Sentiment

- The Classification label used here is the ‘airline_sentiment’ label. It takes 3 categories namely, Positive, Negative, Neutral.
- The Text field of interest is the ‘text’ label, which contains the actual contents of the tweets provided by the travelers.
- The dataset consists of 14640 data entries and the distribution with respect to the classification label is as follows:

Classification Class	Distribution (%)
Positive	16
Negative	63
Neutral	21

- The distribution indicates imbalance of data among the classification categories and hence indicates the requirement of various SMOTE Techniques.
- The dataset also provides other useful information such as reasons for negative and positive sentiments thereby providing useful feedback to major US Airlines.

WORD EMBEDDING TECHNIQUES

- Word Embedding refers to a set of modelling techniques that are used to provide contextual meaning to Textual information by mapping words or phrases to a learned vector of real numbers.
- They are important as they provide useful mathematical correlations among similar words and act as appropriate inputs for various Deep Learning Algorithms.
- It allows the user to capture the syntactic and semantic meaning of textual information in a format that can be provided as input to Deep Learning models. It improves the performance of text classifiers.
- In this Project, different Word Embedding Techniques are applied to the 'text' labeled field to provide various possible vectored representations of the same dataset.
- The Word Embedding Techniques applied to the dataset are as follows:
    - TFIDF (Term Frequency-Inverse Document Frequency)
    - Count Bag of Words
    - W2V (Word to Vector)
    - GloVe (Global Vectors for Word Representations)
    - Skip-Gram Model
    - FastText
    - BERT Model
    - GPT Model (Generative Pre-Trained Transformer)

- **TFIDF (Term Frequency- Inverse Document Frequency)**

This method specifies the vocabulary of the Word Embedding using the input data. Most common words (specified in terms of minimum occurrences within the dataset) are used as column representations and their normalized frequencies within each document are captured. It captures no contextual meaning and correlation among words. It generates a sparse matrix and is usually used to compare documents. The dataset generated is sparse and large and hence is not used as a Word Embedding Technique since we also apply SMOTE oversampling to handle imbalanced data.

- **Count Bag of Words (CBOW)**

This model allows us to provide context to words by targeting its surroundings. It uses surrounding words (context) to predict the current word. It requires a context-window length which specifies how far from a given word does the relevant context extend. It is an implementation approach to the Word2Vec model but in this case we use our own dataset to train the model.

- **Skip-Gram (Skg)**

Similar to the Count Bag of Words model, even this model uses surrounding words as context for a target word but it is different in the sense that it works in an opposite manner. It takes a target word as input and predicts using various probability distributions the most likely context window for it. It is also an implementation approach to the Word2Vec model. In this case as well, we use our dataset to train the model. It works on a semi-supervised learning based approach.

- **GloVe**

It is an Unsupervised learning algorithm, that is used to create global representation of words. The model for our dataset is trained using a glove 6B 300d dictionary. The model provides us with a [300X1] vectored representation for each word in our document. We can compute the semantic similarity between words using the Euclidean distance between two vector representations. It's drawback lies with the fact that there are only few relationships between words that can be captured using a single number. The [300X1] representation for a sentence has been obtained by averaging the [300X1] representations for all its constituent words.

- **FastText**

It is an extension of the Word2Vec model. Instead of having a vectored representation for a word, it produces n-gram of characters for each word. This consequently captures the meanings of prefixes and suffixes. It is especially useful for getting the embedding for words that have not been seen before while training the model since it can be broken down and analyzed. The data corpus used to train the model is the English Wikipedia.

- **Word2Vec**

It is a basic implementation of Word2Vec model which maps a given word to a learned vector. We have already seen its two implementation approaches namely, Count bag of words and Skip-gram model, but in both those approaches the model was trained using the dataset. In this case, we use an existing data corpus provided by Google called Google News Vector negative 300.

- **BERT Model**

It is based on a transformer architecture rather than the conventional LSTM (Long Short Term Memory) architecture. It stands for Bidirectional Encoder Representations from Transformers. It is trained on a very large data corpus of unlabeled text including the entire Wikipedia Book Corpus. This large corpus allows it to deduce deeper meaning and relations between words. The bidirectional aspect of it comes from taking into account the left and the right context for a word. Once the model has been trained using the corpus it is fine-tuned to specific NLP tasks. It is pre-trained on two tasks namely, Masked Language Modelling and Next Sentence Prediction.

- **GPT Model**

It is also based on a transformer architecture. It works on the principle of using pre-trained models to create the corresponding word embedding for our dataset. It finds its application in Text Auto-correct, Auto-complete feature on IDE's. Unlike BERT, it has decoder blocks instead of encoder blocks. It is an autoregressive (predict future behaviour based on past behaviour) learning approach that produces human-like text. It is proven to be extremely powerful at Summarizing Texts and Answering Questions.

IMBALANCED DATA TREATMENT

- As was observed while analyzing the dataset, the distribution of the classification parameter (airline_sentiment) was found to be skewed/biased.

- Since most machine learning algorithms operate on the basic assumption that there are equal numbers of examples for each classification category, balancing the data becomes really important and it also increases the predictive power of the model.

- The source of such imbalance could be attributed to various factors such as biased sampling, measurement errors, problem domain, etc.

- One way to treat imbalanced data is by oversampling the minority class using existing data entries. This method is known as SMOTE (Synthetic Minority Oversampling Technique). It generates new data entries for minority classes to balance the training examples for each class.

- For each Word Embedding obtained through various techniques for the given dataset, we apply various SMOTE Techniques to generate various balanced datasets depicting all combinations of Word Embedding Techniques and SMOTE Techniques.

- The various types of SMOTE Techniques applied to the dataset are:
  - SMOTE
  - SVSMOTE
  - Borderline-SMOTE,
  - SMOTE-NC,
  - ADASYN.

SMOTE TECHNIQUES

- **SMOTE**

It is the most default form of SMOTE Technique which just over samples the minority class. It works only for continuous data features.

- **SVM SMOTE**

It is an extension to borderline SMOTE but it uses the SVM algorithm to identify misclassifications. It focuses more on the parts of the distribution where data is separated rather than generating data on minority class overlap.

- **Borderline-SMOTE**

It is similar to SMOTE except for the fact that it generates data only along the decision boundary between the two classes whereas SMOTE generates data randomly between two data points.

It has two variations :

1. **Borderline SMOTE1** - Over Samples the majority class as well in case of misclassifications along the decision boundary.
2. **Borderline SMOTE2** - Just over samples the minority class.

- **SMOTE-NC**

It is just an extension to the default SMOTE Technique but it works for categorical and continuous features as well. It is especially useful for cases when we have mixed features in our dataset.

- **ADASYN**

It stands for Adaptive Synthetic Sampling. It is similar to borderline SMOTE except for the fact that it generates the data according to data density. Data generation is inversely proportional to the density distribution of the minority class.

```
import pandas as pd
import numpy as np
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import BorderlineSMOTE
from imblearn.over_sampling import SVMSMOTE
from imblearn.over_sampling import ADASYN
# sm = SVMSMOTE(random_state=42)
# sm= SMOTE(random_state=42)
# sm= BorderlineSMOTE(random_state=42,kind='borderline-1')
# sm= BorderlineSMOTE(random_state=42,kind='borderline-2')
sm= ADASYN(random_state=42)
cou=0


for i in range(1,8):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    df=np.genfromtxt(fname,delimiter=',')
    datan=df[:,0:-1]
    out=df[:,-1]
    X_res, y_res = sm.fit_resample(datan,out)
    y=y_res.reshape(-1,1)
    d=np.concatenate((X_res,y),axis=1)
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(35+i)+'.csv'
    np.savetxt(fname,d, delimiter=',', fmt='%f')
```

SMOTE TECHNIQUES USED

```
# Feature Extraction with PCA
import numpy as np
import pandas as pd
from pandas import read_csv
from sklearn.decomposition import PCA

for i in range(1,43):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    dataframe=np.genfromtxt(fname,delimiter=',')
    X = dataframe[:,0:-1]
    Y = dataframe[:,-1]
    Y_data=pd.DataFrame(data=Y)
    # feature extraction
    pca = PCA(n_components=100)
    dfpc=pca.fit_transform(X)
    df = pd.DataFrame(data = dfpc)
    df_final=pd.concat([df,Y_data],axis=1)
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(126+i)+'.csv'
    np.savetxt(fname,df_final, delimiter=',', fmt='%f')
```

FEATURE SELECTION USING PCA

FEATURE SELECTION

- After removing imbalance from the data, we are now left with multiple datasets having numerous features (column labels) representing each document in our dataset.

- An important step to be completed before the dataset is ready to be given as an input to a Deep Learning model is Feature Selection.

- Feature Selection refers to the set of methods/techniques that are used to select a subset of input features by eliminating redundant, irrelevant and non-informative features from the dataset. It improves the predictive accuracy of the DL models.

- It reduces the size of the dataset and hence allows the model to train faster. Often the features are ranked based on their significance in predicting the classification class and only the top log n features are chosen (varies based on the total features).

- The need for applying Feature Selection arises from the presence of Correlated input features which reduce the relative importance of the actual significant features. Some of the features are relatively more important than the others in identifying the class for a data entry. Usually, the features are ranked based on certain specific property which is specified through Domain knowledge and Requirements.

- The Feature Selection Techniques used for the datasets are as follows:
    1. Significant Features
        i. Gini Values
        ii. Confidence Intervals
    2. Correlation Analysis
    3. Principal Component Analysis (PCA)

- **SIGNIFICANT FEATURES**

  - This method essentially depicts the class of techniques that are used to determine the most relevant/significant subset of the set of input features which are useful to determine the class of a given input. It essentially involves different properties based on which the features are ranked for significance.

  -We use two common implementations of this approach:

  1. **Confidence Interval -** Out of all the features in the dataset, only those features are selected for which the 95% Confidence Intervals for different class categories do not overlap. CAn be visualized more easily using Box plots.
  2. **Gini Index -** It calculates the probability that a specific feature is classified incorrectly when chosen randomly. It works for discrete values only.

- **CORRELATION ANALYSIS**

  -This method is used to remove correlated input features from the dataset. For our dataset, we have used **Pearson** Correlation as a measure for correlation.

  -For each input feature, Pearson Correlation returns a value between -1 and 1. Usually, an absolute value of >=0.7 is treated to be highly correlated and hence one of those two features is dropped from the dataset.

  -The above method is applied for each pair of input features until only uncorrelated features are left. This can also be visualized using correlation heatmaps (present in seaborn library)

- **PRINCIPAL COMPONENT ANALYSIS (PCA)**

  - It is not essentially a feature selection technique but can be used as one if the property that depicts the significance of an input feature based on its variation.

  -It creates new uncorrelated variables that can be expressed as linear combination of old variables thereby minimizing the size of the input features without having drastic effects on the information conveyed.

  -It is one of the most common dimensionality-reduction techniques.

```
def civalue(x):
    mx=np.mean(x)
    ci=np.zeros((2))
    ci[0]=mx-1.96*x.std()/math.sqrt(len(x))
    ci[1]=mx+1.96*x.std()/math.sqrt(len(x))
    return ci

civ=np.zeros((len(x[0]),6))
for i in range(0,x.shape[1]):
    fv=x[:,i]
    civ[i,0:2]=civalue(fv[in0])
    civ[i,2:4]=civalue(fv[in1])
    civ[i,4:6]=civalue(fv[in2])

checkciv=np.zeros((len(x[0]),6))
for i in range(0,x.shape[1]):
    fv=x[:,i]
    checkciv[i,0:2]=civalue(fv[inn0])
    checkciv[i,2:4]=civalue(fv[inn1])
    checkciv[i,4:6]=civalue(fv[inn2])

count=0;
impfeatures=[]
for i in range(0,x.shape[1]):
    if (civ[i][0]>checkciv[i][1] or civ[i][1]<checkciv[i][0]):
        if (civ[i][2]>checkciv[i][3] or civ[i][3]<checkciv[i][2]):
            if (civ[i][4]>checkciv[i][5] or civ[i][5]<checkciv[i][4]):
                impfeatures.append(i)

finalarr=np.empty((x.shape[0],0))

for i in impfeatures:
  col=x[:,i]
  col=np.reshape(col,(x.shape[0],1))
  finalarr = np.append(finalarr, col, axis=1)
```

FEATURE SELECTION USING CONFIDENCE INTERVAL

```
for loopvar in range(1,43):
    paribas_data = pd.read_csv('C:/Users/Shivang/Desktop/Twitter airline/'+str(loopvar)+'.csv',header=None)

    y=paribas_data.iloc[:,-1]
    paribas_data=paribas_data.iloc[:,:-1]

    correlated_features = set()
    correlation_matrix = paribas_data.corr()

    for i in range(len(correlation_matrix.columns)):
        for j in range(i):
            if abs(correlation_matrix.iloc[i, j]) > 0.7:
                colname = correlation_matrix.columns[i]
                correlated_features.add(colname)

    paribas_data.drop(labels=correlated_features, axis=1, inplace=True)
    final_arr=paribas_data.join(y)
    np.savetxt('C:/Users/Shivang/Desktop/Twitter airline/'+str(84+loopvar)+'.csv', final_arr ,delimiter=',')
```

FEATURE SELECTION USING CROSS CORRELATION

CLASSIFICATION MODELS

- After performing various feature selection techniques on the dataset, it is now prepared to be entered as an input to different classification models.

- The Classification Models are trained using a part of the dataset to gauge the relationship between the input features and the classification label i.e. airline_sentiment (positive, negative and neutral).
- The Dataset was split (80:20 ratio) into Training and Testing Datasets to ensure Cross Validation and prevent overfitting in data. The model was trained on the training set and then the accuracy score was measured based on the model's performance on the testing dataset.

- The trained model is then used to classify the remaining part of the dataset and then compared with its actual value. The classifications and misclassification ratios are then used to calculate the overall Model Accuracy.

- The model with the best accuracy provides the estimate for the most optimal model framework (Word Embedding, SMOTE Technique, Feature Selection Technique and Classification model).

- The Classification Models used in the project are as follows:
    - Logistic Regression
    - Decision Tree
    - Naive Bayes
    - Linear Support Vector Machine
    - MLP Classifier

- **LOGISTIC REGRESSION**

- A Multiclass Logistic Regression model is used because the target feature (airline_sentiment) has three classes namely positive, negative and neutral.
- Its foundation is based on the Odds of an Event happening when provided with a set of values for the predictor variables.
- It assumes no collinearity among the predictor variables, independent observations and mutually exclusive class categories.

```
logistic_scores=[]

for i in range(85,127):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    df=np.genfromtxt(fname,delimiter=',')
    logisticRegr = LogisticRegression(random_state=4,max_iter=4000)
    x_train, x_test, y_train, y_test = train_test_split(df[:,0:-1], df[:,-1], test_size=0.2,random_state=0)
    logisticRegr.fit(x_train, y_train)
    predictions = logisticRegr.predict(x_test)
    score = logisticRegr.score(x_test, y_test)
    logistic_scores.append(score)
```

- **DECISION TREE**

- The Decision Tree Classifier creates a flow-chart based tree structure consisting of ‘test’ conditions and their outputs for the predictor variables.
- The conditions of the Decision Tree varies depending on the maximum allowable depth of the tree. The goal is to ‘answer’ each test condition and follow the path to a root node.
- It works at each step on the principles of largest information gain and divide and conquer.

```
decision_scores=[]

for i in range(127,169):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    df=np.genfromtxt(fname,delimiter=',')
    classifier = DecisionTreeClassifier(max_depth=50, random_state=42)
    x_train, x_test, y_train, y_test = train_test_split(df[:,0:-1], df[:,-1], test_size=0.2,random_state=0)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    decision_scores.append(accuracy)
```

- **NAIVE BAYES**

- It is a classification technique based on the principles of Bayes Theorem. The classifier built is highly sensitive to the Independence of Input features.
- It calculates the individual membership probabilities for each data point for each class category and then chooses the category with maximum probability.
- It is considered extremely useful in conditions where there is scarcity of data points or scalability and cost requirements. It also works well with high dimensional data.

```
NB_scores=[]

for i in range(127,169):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    df=np.genfromtxt(fname,delimiter=',')
    classifier = GaussianNB()
    x_train, x_test, y_train, y_test = train_test_split(df[:,0:-1], df[:,-1], test_size=0.2,random_state=0)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    NB_scores.append(accuracy)
```

- **LINEAR SUPPORT VECTOR MACHINE**

- It uses a linear model for classifying the data points into classes. It creates a hyperplane that is used to separate various data points into their corresponding class categories.
- It calculates support vectors (points closest to the line belonging to separate classes) for a fit and then compares various fits based on their margin values(distance of support vectors from fit).
- The Linear version of SVC is recommended for multi class problems and hence we use the Linear version. The critical aspect of the algorithm is to estimate the best line that not only distinguishes the given data points, but is generalized enough to classify new unseen examples.

- It is memory efficient and works extremely well for high dimensional data. However, it is sensitive to noise and underperforms for large datasets.

```
SVC_scores=[]

for i in range(1,85):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    df=np.genfromtxt(fname,delimiter=',')
    classifier = LinearSVC()
    x_train, x_test, y_train, y_test = train_test_split(df[:,0:-1], df[:,-1], test_size=0.2,random_state=0)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    SVC_scores.append(accuracy)
```

- **MLP CLASSIFIER**

- It stands for Multilayer Perceptron. It is based on feedforward (anticipated) Artificial Neural Network and backpropagation.
- The performance of the Classification model is gauged using a loss function. The value of the loss function is minimized using the Neural Network.
- It consists of an input and output layer separated by multiple hidden layers. It uses an activation function to calculate the values for each internal node.
- It is different from a true perceptron as a true perceptron can only be used for binary classification whereas a neuron in MLP model can be used for multiclass classification as well.
- It is extremely useful in cases where the data points are not linearly separable.

```
mlp_scores=[]

for i in range(1,85):
    fname='C:/Users/Shivang/Desktop/Twitter airline/'+str(i)+'.csv'
    df=np.genfromtxt(fname,delimiter=',')
    classifier = MLPClassifier(random_state=1,max_iter=8000)
    x_train, x_test, y_train, y_test = train_test_split(df[:,0:-1], df[:,-1], test_size=0.2,random_state=0)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    mlp_scores.append(accuracy)
```

RESULTS

All the possible combinations for the model framework (Word Embedding, SMOTE Technique, Feature Selection and Classification Model) were evaluated based on their accuracies on the datasets. The Datasets were split (80:20 ratio) into training and testing sets respectively. The models were trained using the training set and then were used to predict values for the test set. The accuracy was measured for predicting the correct class category for each model.

1. WORD EMBEDDING:

The **Mean Accuracy Scores** for different Word Embedding Techniques are :

Model Accuracy * Word Embedding

Model Accuracy

Word Embedding	Mean	N	Std. Deviation	Minimum	Maximum	Range
bert - Word Embedding	.68253	120	.100799	.435	.899	.464
cbow - Word Embedding	.65965	120	.081816	.445	.899	.454
fasttext - Word Embedding	.57533	120	.094389	.438	.902	.464
glove - Word Embedding	.70179	120	.092658	.445	.901	.456
gptmodel - Word Embedding	.52202	120	.116601	.336	.820	.484
skg - Word Embedding	.70640	120	.070845	.566	.899	.333
W2V - Word Embedding	.72535	120	.085412	.445	.909	.464
Total	.65330	840	.116153	.336	.909	.573

2. SMOTE TECHNIQUES:

The **Mean Accuracy Scores** for different SMOTE Techniques are :

Model Accuracy * SMOTE Techniques

Model Accuracy

SMOTE Techniques	Mean	N	Std. Deviation	Minimum	Maximum	Range
ADASYN	.62690	140	.126581	.348	.908	.560
Borderline SMOTE 1	.63558	120	.130666	.337	.906	.569
Borderline SMOTE 2	.62752	160	.116648	.336	.902	.566
None	.68397	140	.076114	.458	.899	.441
SMOTE	.65969	140	.124775	.396	.909	.513
SVMSMOTE	.68726	140	.099899	.372	.898	.526
Total	.65330	840	.116153	.336	.909	.573

3. FEATURE SELECTION TECHNIQUES:

The **Mean Accuracy Scores** for different Feature Selection Techniques are :

Model Accuracy * Feature Selection techniques

Model Accuracy

Feature Selection techniques	Mean	N	Std. Deviation	Minimum	Maximum	Range
Confidence Interval	.65153	210	.117575	.395	.899	.504
Cross Correlation Analysis	.65395	210	.116395	.336	.902	.566
None	.65633	210	.119575	.398	.909	.511
Principal Component Analysis	.65138	210	.111684	.336	.901	.565
Total	.65330	840	.116153	.336	.909	.573

4. CLASSIFICATION MODELS:

The **Mean Accuracy Scores** for different Classification models are :

Model Accuracy * Classification Model

Model Accuracy

Classification Model	Mean	N	Std. Deviation	Minimum	Maximum	Range
Decision Tree	.65748	168	.081772	.431	.766	.335
Linear SVC	.66218	168	.102585	.426	.787	.361
Logistic Regression	.65776	168	.102530	.425	.787	.362
MLP Classifier	.73059	168	.130210	.431	.909	.478
Naive Bayes	.55847	168	.089054	.336	.728	.392
Total	.65330	840	.116153	.336	.909	.573

The Best and Worst Combinations specific to a particular classification model. The combinations are w.r.t the tuple (Map Index, Word Embedding, SMOTE Technique, Feature Selection, Classification Model, Accuracy Score).

- **Logistic Regression**

- The Best 10 combinations are:

9	W2V - Word Embedding	SVMSMOTE	None	Logistic Regression	0.787
93	W2V - Word Embedding	SVMSMOTE	Cross Correlation Analysis	Logistic Regression	0.787
2	W2V - Word Embedding	None	None	Logistic Regression	0.776
86	W2V - Word Embedding	None	Cross Correlation Analysis	Logistic Regression	0.776
16	W2V - Word Embedding	SMOTE	None	Logistic Regression	0.775
100	W2V - Word Embedding	SMOTE	Cross Correlation Analysis	Logistic Regression	0.775
128	W2V - Word Embedding	None	Principal Component Analysis	Logistic Regression	0.775
14	bert - Word Embedding	SVMSMOTE	None	Logistic Regression	0.773
51	W2V - Word Embedding	SVMSMOTE	Confidence Interval	Logistic Regression	0.772
6	glove - Word Embedding	None	None	Logistic Regression	0.771

- The Worst 10 combinations are:

99	gptmodel - Word Embedding	SMOTE	Cross Correlation Analysis	Logistic Regression	0.425
15	gptmodel - Word Embedding	SMOTE	None	Logistic Regression	0.431
57	gptmodel - Word Embedding	SMOTE	Confidence Interval	Logistic Regression	0.431
120	gptmodel - Word Embedding	ADASYN	Cross Correlation Analysis	Logistic Regression	0.437
36	gptmodel - Word Embedding	ADASYN	None	Logistic Regression	0.445
78	gptmodel - Word Embedding	ADASYN	Confidence Interval	Logistic Regression	0.445
162	gptmodel - Word Embedding	ADASYN	Principal Component Analysis	Logistic Regression	0.445
106	gptmodel - Word Embedding	Borderline SMOTE 1	Cross Correlation Analysis	Logistic Regression	0.448
22	gptmodel - Word Embedding	Borderline SMOTE 1	None	Logistic Regression	0.452
64	gptmodel - Word Embedding	Borderline SMOTE 1	Confidence Interval	Logistic Regression	0.452

- **Decision Tree**

- The Best 10 combinations are:

220	skg - Word Embedding	SVMSMOTE	Confidence Interval	Decision Tree	0.766
221	cbow - Word Embedding	SVMSMOTE	Confidence Interval	Decision Tree	0.765
178	skg - Word Embedding	SVMSMOTE	None	Decision Tree	0.762
192	skg - Word Embedding	Borderline SMOTE 1	None	Decision Tree	0.76
179	cbow - Word Embedding	SVMSMOTE	None	Decision Tree	0.759
262	skg - Word Embedding	SVMSMOTE	Cross Correlation Analysis	Decision Tree	0.758
304	skg - Word Embedding	SVMSMOTE	rincipal Component Analys	Decision Tree	0.758
185	skg - Word Embedding	SMOTE	None	Decision Tree	0.755
186	cbow - Word Embedding	SMOTE	None	Decision Tree	0.755
227	skg - Word Embedding	SMOTE	Confidence Interval	Decision Tree	0.754

- The Worst 10 combinations are:

183	gptmodel - Word Embedding	SMOTE	None	Decision Tree	0.431
225	gptmodel - Word Embedding	SMOTE	Confidence Interval	Decision Tree	0.431
204	gptmodel - Word Embedding	ADASYN	None	Decision Tree	0.445
288	gptmodel - Word Embedding	ADASYN	Cross Correlation Analysis	Decision Tree	0.445
330	gptmodel - Word Embedding	ADASYN	rincipal Component Analys	Decision Tree	0.445
246	gptmodel - Word Embedding	ADASYN	Confidence Interval	Decision Tree	0.446
274	gptmodel - Word Embedding	Borderline SMOTE 1	Cross Correlation Analysis	Decision Tree	0.451
316	gptmodel - Word Embedding	Borderline SMOTE 1	rincipal Component Analys	Decision Tree	0.451
190	gptmodel - Word Embedding	Borderline SMOTE 1	None	Decision Tree	0.452
232	gptmodel - Word Embedding	Borderline SMOTE 1	Confidence Interval	Decision Tree	0.452

- **Naive Bayes**

- The Best 10 combinations are:

422	W2V - Word Embedding	None	Cross Correlation Analysis	Naive Bayes	0.728
464	W2V - Word Embedding	None	Principal Component Analysis	Naive Bayes	0.728
339	skg - Word Embedding	None	None	Naive Bayes	0.709
381	skg - Word Embedding	None	Confidence Interval	Naive Bayes	0.706
338	W2V - Word Embedding	None	None	Naive Bayes	0.693
380	W2V - Word Embedding	None	Confidence Interval	Naive Bayes	0.693
423	skg - Word Embedding	None	Cross Correlation Analysis	Naive Bayes	0.678
465	skg - Word Embedding	None	Principal Component Analysis	Naive Bayes	0.678
352	W2V - Word Embedding	SMOTE	None	Naive Bayes	0.674
430	skg - Word Embedding	SVMSMOTE	Cross Correlation Analysis	Naive Bayes	0.67

- The Worst 10 combinations are:

449	gptmodel - Word Embedding	Borderline SMOTE 2	Cross Correlation Analysis	Naive Bayes	0.336
491	gptmodel - Word Embedding	Borderline SMOTE 2	Principal Component Analysis	Naive Bayes	0.336
442	gptmodel - Word Embedding	Borderline SMOTE 1	Cross Correlation Analysis	Naive Bayes	0.337
484	gptmodel - Word Embedding	Borderline SMOTE 1	Principal Component Analysis	Naive Bayes	0.337
456	gptmodel - Word Embedding	ADASYN	Cross Correlation Analysis	Naive Bayes	0.348
498	gptmodel - Word Embedding	ADASYN	Principal Component Analysis	Naive Bayes	0.348
428	gptmodel - Word Embedding	SVMSMOTE	Cross Correlation Analysis	Naive Bayes	0.372
470	gptmodel - Word Embedding	SVMSMOTE	Principal Component Analysis	Naive Bayes	0.372
407	gptmodel - Word Embedding	Borderline SMOTE 2	Confidence Interval	Naive Bayes	0.395
393	gptmodel - Word Embedding	SMOTE	Confidence Interval	Naive Bayes	0.396

- **Linear SVM**

- The Best 10 combinations are:

513	W2V - Word Embedding	SVMSMOTE	None	Linear SVC	0.787
597	W2V - Word Embedding	SVMSMOTE	Cross Correlation Analysis	Linear SVC	0.787
518	bert - Word Embedding	SVMSMOTE	None	Linear SVC	0.78
506	W2V - Word Embedding	None	None	Linear SVC	0.779
520	W2V - Word Embedding	SMOTE	None	Linear SVC	0.779
590	W2V - Word Embedding	None	Cross Correlation Analysis	Linear SVC	0.779
604	W2V - Word Embedding	SMOTE	Cross Correlation Analysis	Linear SVC	0.779
555	W2V - Word Embedding	SVMSMOTE	Confidence Interval	Linear SVC	0.774
632	W2V - Word Embedding	None	Principal Component Analysis	Linear SVC	0.773
510	glove - Word Embedding	None	None	Linear SVC	0.772

- The Worst 10 combinations are:

603	gptmodel - Word Embedding	SMOTE	Cross Correlation Analysis	Linear SVC	0.426
561	gptmodel - Word Embedding	SMOTE	Confidence Interval	Linear SVC	0.431
519	gptmodel - Word Embedding	SMOTE	None	Linear SVC	0.433
624	gptmodel - Word Embedding	ADASYN	Cross Correlation Analysis	Linear SVC	0.435
540	gptmodel - Word Embedding	ADASYN	None	Linear SVC	0.436
582	gptmodel - Word Embedding	ADASYN	Confidence Interval	Linear SVC	0.445
666	gptmodel - Word Embedding	ADASYN	Principal Component Analysis	Linear SVC	0.445
610	gptmodel - Word Embedding	Borderline SMOTE 1	Cross Correlation Analysis	Linear SVC	0.447
526	gptmodel - Word Embedding	Borderline SMOTE 1	None	Linear SVC	0.453
568	gptmodel - Word Embedding	Borderline SMOTE 1	Confidence Interval	Linear SVC	0.453

- **MLP Classifier**

- The Best 10 combinations are:

688	W2V - Word Embedding	SMOTE	None	MLP Classifier	0.909
709	W2V - Word Embedding	ADASYN	None	MLP Classifier	0.908
695	W2V - Word Embedding	Borderline SMOTE 1	None	MLP Classifier	0.906
789	fasttext - Word Embedding	Borderline SMOTE 2	Cross Correlation Analysis	MLP Classifier	0.902
832	glove - Word Embedding	Borderline SMOTE 2	Principal Component Analysis	MLP Classifier	0.901
713	glove - Word Embedding	ADASYN	None	MLP Classifier	0.899
737	W2V - Word Embedding	Borderline SMOTE 1	Confidence Interval	MLP Classifier	0.899
763	bert - Word Embedding	None	Cross Correlation Analysis	MLP Classifier	0.899
793	W2V - Word Embedding	ADASYN	Cross Correlation Analysis	MLP Classifier	0.899
836	skg - Word Embedding	ADASYN	Principal Component Analysis	MLP Classifier	0.899

- The Worst 10 combinations are:

687	gptmodel - Word Embedding	SMOTE	None	MLP Classifier	0.431
729	gptmodel - Word Embedding	SMOTE	Confidence Interval	MLP Classifier	0.431
708	gptmodel - Word Embedding	ADASYN	None	MLP Classifier	0.445
750	gptmodel - Word Embedding	ADASYN	Confidence Interval	MLP Classifier	0.445
776	glove - Word Embedding	SMOTE	Cross Correlation Analysis	MLP Classifier	0.445
788	cbow - Word Embedding	Borderline SMOTE 2	Cross Correlation Analysis	MLP Classifier	0.445
814	W2V - Word Embedding	SMOTE	Principal Component Analysis	MLP Classifier	0.445
736	gptmodel - Word Embedding	Borderline SMOTE 1	Confidence Interval	MLP Classifier	0.45
694	gptmodel - Word Embedding	Borderline SMOTE 1	None	MLP Classifier	0.451
831	fasttext - Word Embedding	Borderline SMOTE 2	Principal Component Analysis	MLP Classifier	0.463

OVERALL:

Considering all possible combinations not specific to a particular Classification model. The combinations are wrt the tuple (Word Embedding, SMOTE Technique, Feature Selection, Classification Methods, Accuracy Score) .

The Best 10 combinations are :

688	W2V - Word Embedding	SMOTE	None	MLP Classifier	0.909
709	W2V - Word Embedding	ADASYN	None	MLP Classifier	0.908
695	W2V - Word Embedding	Borderline SMOTE 1	None	MLP Classifier	0.906
789	fasttext - Word Embedding	Borderline SMOTE 2	Cross Correlation Analysis	MLP Classifier	0.902
832	glove - Word Embedding	Borderline SMOTE 2	Principal Component Analysis	MLP Classifier	0.901
713	glove - Word Embedding	ADASYN	None	MLP Classifier	0.899
737	W2V - Word Embedding	Borderline SMOTE 1	Confidence Interval	MLP Classifier	0.899
763	bert - Word Embedding	None	Cross Correlation Analysis	MLP Classifier	0.899
793	W2V - Word Embedding	ADASYN	Cross Correlation Analysis	MLP Classifier	0.899
836	skg - Word Embedding	ADASYN	Principal Component Analysis	MLP Classifier	0.899

The Worst 10 combinations are :

449	gptmodel - Word Embedding	Borderline SMOTE 2	Cross Correlation Analysis	Naive Bayes	0.336
491	gptmodel - Word Embedding	Borderline SMOTE 2	Principal Component Analysis	Naive Bayes	0.336
442	gptmodel - Word Embedding	Borderline SMOTE 1	Cross Correlation Analysis	Naive Bayes	0.337
484	gptmodel - Word Embedding	Borderline SMOTE 1	Principal Component Analysis	Naive Bayes	0.337
456	gptmodel - Word Embedding	ADASYN	Cross Correlation Analysis	Naive Bayes	0.348
498	gptmodel - Word Embedding	ADASYN	Principal Component Analysis	Naive Bayes	0.348
428	gptmodel - Word Embedding	SVMSMOTE	Cross Correlation Analysis	Naive Bayes	0.372
470	gptmodel - Word Embedding	SVMSMOTE	Principal Component Analysis	Naive Bayes	0.372
407	gptmodel - Word Embedding	Borderline SMOTE 2	Confidence Interval	Naive Bayes	0.395
393	gptmodel - Word Embedding	SMOTE	Confidence Interval	Naive Bayes	0.396

CONCLUSION

After analyzing the **Accuracy Scores** for all possible combinations of the Model Framework, it can be seen that the following **Model Frameworks** are most suitable for the **Text Classification of Twitter AIrline Sentiments.**

1. Word2Vec -> SMOTE -> No Feature Selection ->MLP Classifier -> 90.9% Accuracy
2. Word2Vec -> ADASYN -> No Feature Selection ->MLP Classifier -> 90.8% Accuracy
3. Word2Vec -> Borderline SMOTE 1 -> No Feature Selection ->MLP Classifier -> 90.6% Accuracy
4. Fasttext -> Borderline SMOTE 2 -> Cross Correlation Analysis ->MLP Classifier -> 90.2% Accuracy
5. GloVe -> Borderline SMOTE 2 -> Principal Component Analysis ->MLP Classifier -> 90.1% Accuracy

On comparing the **Mean Accuracy Scores** individually for each Component of the framework, it is found that the following give the best results:

Individual Component	**Most Optimal Choice**
WORD EMBEDDING	Word2Vec (W2V)
SMOTE TECHNIQUE	SVMSMOTE
FEATURE SELECTION TECHNIQUE	Cross Correlation Analysis
CLASSIFICATION MODEL	MLP Classifier

REFERENCES

- CountVectorizer
- TFIDF
- Word2Vec
- GloVe NLP , GloVe Embeddings
- Cbow vs Skip-Gram Model
- FastText
- LSTM vs Transformer Architecture
- GPT Model
- BERT Model
- Identifying Data Imbalance
- SMOTE Techniques
- Feature Selection Importance
- Feature Ranking and Feature Subset Selection
- Confidence Interval
- Pearson Correlation
- Principal Component Analysis
- Linear SVM

- Logistic Regression
- Multi Class Classification
- MLP Classifier
- Decision Tree
- Naive Bayes
- SPSS Tutorial