20 Newsgroups Text Classification

AML-2304 Natural Language Processing

Final Project Report

Prof. **Harriet Huang**

Group: Valar Morghulis

- Kishan Lakhtariya C0849341
- Harshkumar Patel C0848343
- Kanan Trivedi C0851492
- Mohammaddin Masbi C0850500

Abstract

Analyzed and processed text data of 20 newsgroups text datasets with different techniques to get meaningful words from the dataset and converted them into word vectors using two different word embedding strategies, TF-IDF and Word2Vec, and trained the vectorized data using three different machine learning models, from all the training we got maximum training accuracy of 96% and testing accuracy of 71% using Support Vector Machine Classifier with the vectorizer TF-IDF.

1. Introduction

- It is essential to classify the new topics for the chatbots and AI assistants to give recommendations based on the reader's interest in addressing this problem; we have chosen the news dataset with 20 different topics as labels and trained the model to predict the news text.

2. Dataset

- We have used the fetch_20newsgroups API to get the dataset which contains 18846 samples with 2 columns of data and a target.

```
# Fetching and Storing all data into news_dataset variable with shuffling
news_dataset = fetch_20newsgroups(subset='all',
                                  shuffle=True,
                                  remove=('headers', 'footers', 'quotes'))
```

- As the target is the integer value, we have mapped the target_lables with it for further analysis.

```
# Mapping labels with its integer values (Label Encoding)
lable_map = dict((i, j) for i, j in enumerate(news_dataset.target_names))
lable_map
```

```
{0: 'alt.atheism',
 1: 'comp.graphics',
 2: 'comp.os.ms-windows.misc',
 3: 'comp.sys.ibm.pc.hardware',
 4: 'comp.sys.mac.hardware',
 5: 'comp.windows.x',
 6: 'misc.forsale',
 7: 'rec.autos',
 8: 'rec.motorcycles',
 9: 'rec.sport.baseball',
 10: 'rec.sport.hockey',
 11: 'sci.crypt',
 12: 'sci.electronics',
 13: 'sci.med',
 14: 'sci.space',
 15: 'soc.religion.christian',
 16: 'talk.politics.guns',
 17: 'talk.politics.mideast',
 18: 'talk.politics.misc',
 19: 'talk.religion.misc'}
```

```
# Converting train dataset into pandas dataframe and storing into train_df variable
df = pd.DataFrame({'data': news_dataset.data, 'target': news_dataset.target})
df["target_label"] = df["target"].map(lable_map)
df.head()
```

	data	target	target_label
0	\n\nI am sure some bashers of Pens fans are pr...	10	rec.sport.hockey
1	My brother is in the market for a high-perform...	3	comp.sys.ibm.pc.hardware
2	\n\n\n\n\tFinally you said what you dream abou...	17	talk.politics.mideast
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	3	comp.sys.ibm.pc.hardware
4	1) I have an old Jasmine drive which I cann...	4	comp.sys.mac.hardware

- The initial shape of the data is:

```
df.shape
```

```
(18846, 3)
```

3. Exploratory Data Analysis (EDA):

- We have three data initially
  i. Data: which contains the text of the news
  ii. Target: which is the integer value of labels
  iii. Target_labels: which is the string of the labels

```
# Printing dataset
df
```

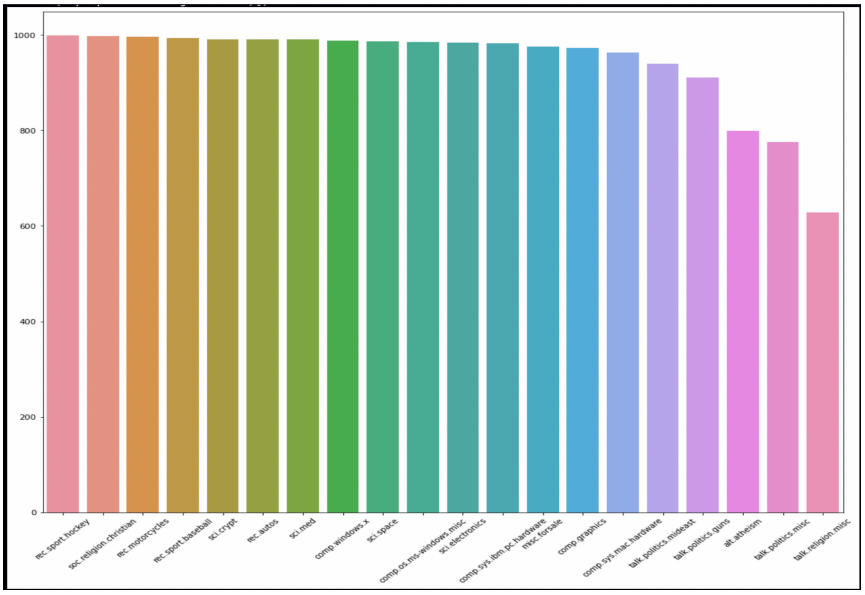
	data	target	target_label
0	\n\nI am sure some bashers of Pens fans are pr...	10	rec.sport.hockey
1	My brother is in the market for a high-perform...	3	comp.sys.ibm.pc.hardware
2	\n\n\n\n\tFinally you said what you dream abou...	17	talk.politics.mideast
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	3	comp.sys.ibm.pc.hardware
4	1) I have an old Jasmine drive which I cann...	4	comp.sys.mac.hardware
...	...	...	...
18841	DN> From: nyeda@cnsvax.uwec.edu (David Nye)\nD...	13	sci.med
18842	\nNot in isolated ground recepticles (usually ...	12	sci.electronics
18843	I just installed a DX2-66 CPU in a clone mothe...	3	comp.sys.ibm.pc.hardware
18844	\nWouldn't this require a hyper-sphere. In 3-...	1	comp.graphics
18845	After a tip from Gary Crum (crum@fcom.cc.utah....	7	rec.autos

18846 rows × 3 columns

- The total count and frequency of the labels are:

```
# Getting the exact count of Labels
df.target_label.value_counts()
```

```
rec.sport.hockey           999
soc.religion.christian     997
rec.motorcycles            996
rec.sport.baseball         994
sci.crypt                  991
rec.autos                  990
sci.med                    990
comp.windows.x             988
sci.space                  987
comp.os.ms-windows.misc    985
sci.electronics            984
comp.sys.ibm.pc.hardware   982
misc.forsale               975
comp.graphics              973
comp.sys.mac.hardware      963
talk.politics.mideast      940
talk.politics.guns         910
alt.atheism                799
talk.politics.misc         775
talk.religion.misc         628
Name: target_label, dtype: int64
```



4. Preprocessing

- We performed pre-processing steps to convert text data into meaningful data to make the computer understandable the text data.
- First, we plot the word cloud to know the frequency of the words, and we discover that many un-useful words have higher frequencies than essential words.

```
# plot word count for data
wc = wordcloud.WordCloud()
wc.generate(str(df['data']))
fig = plt.figure(figsize=[20, 22])
plt.title('WordCloud of News')
plt.axis('off')
plt.imshow(wc)
plt.show()
```

- Here we can see that the occurrence of the “n” is higher in the text data.
- So, we must perform data cleaning, including removing symbols and ASCII characters, stopping word removal, steaming and lemmatizing.
- We tokenized the data using NLTK’s word_tokenize function and stored all tokenized words in separate columns.

```
# Converting text paragraph into tokens using nltk.work_tokenize function and also removing special charecters and un-usefull text
# And returning tokens list
def tokenize(data):
    tokens = nltk.word_tokenize(data)
    return [word for word in tokens if word.isalpha()]
```

```
# Applying tokenize function to data column of dataset and storing results into column name tokenized
df['tokenized'] = df.apply(lambda x: tokenize(x['data']), axis=1)
df.head()
```

	data	target	target_label	tokenized
0	\n\nI am sure some bashers of Pens fans are pr...	10	rec.sport.hockey	[I, am, sure, some, bashers, of, Pens, fans, a...
1	My brother is in the market for a high-perform...	3	comp.sys.ibm.pc.hardware	[My, brother, is, in, the, market, for, a, vid...
2	\n\n\n\n\tFinally you said what you dream abou...	17	talk.politics.mideast	[Finally, you, said, what, you, dream, about, ...
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	3	comp.sys.ibm.pc.hardware	[Think, It, the, SCSI, card, doing, the, DMA, ...
4	1) I have an old Jasmine drive which I cann...	4	comp.sys.mac.hardware	[I, have, an, old, Jasmine, drive, which, I, c...

- We applied the English stop word removal on the tokenized word using “stopwords.words()” of the NLTK function.

```
# Definning function to remove english stopwords from tokenized data with the help of stopwords function of NLTK
def remove_stopword(data):

    stopword = stopwords.words("english")
    return [word for word in data if not word in stopword]
```

```
# Appling remove_stopword function to column tokenized of dataset and storing returned values into stopword_removed column

df['stopword_removed'] = df.apply(lambda x: remove_stopword(x['tokenized']),
                                  axis=1)
df[["data", "stopword_removed"]].head()
```

	data	stopword_removed
0	\n\nI am sure some bashers of Pens fans are pr...	[I, sure, bashers, Pens, fans, pretty, confuse...
1	My brother is in the market for a high-perform...	[My, brother, market, video, card, supports, V...
2	\n\n\n\n\tFinally you said what you dream abou...	[Finally, said, dream, Mediterranean, That, ne...
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	[Think, It, SCSI, card, DMA, transfers, NOT, d...
4	1) I have an old Jasmine drive which I cann...	[I, old, Jasmine, drive, I, use, new, system, ...

- After removing the stop words, we convert the words to their root form, which is called stemming

```
# Converting words to it's root form using stemming technique
# Word stemming using PorterStemmer function of NLTK
# And return the stemmed word list


def stemming(data):

    stemmer = PorterStemmer()
    return [stemmer.stem(word) for word in data]
```

```
# Appling stemming function to dataset with column name stopword_removed

df['stemmed_words'] = df.apply(lambda x: stemming(x['stopword_removed']),
                               axis=1)
df[['data', 'stemmed_words']].head()
```

	data	stemmed_words
0	\n\nI am sure some bashers of Pens fans are pr...	[i, sure, basher, pen, fan, pretti, confus, la...
1	My brother is in the market for a high-perform...	[my, brother, market, video, card, support, ve...
2	\n\n\n\n\tFinally you said what you dream abou...	[final, said, dream, mediterranean, that, new,...
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	[think, it, scsi, card, dma, transfer, not, di...
4	1) I have an old Jasmine drive which I cann...	[i, old, jasmin, drive, i, use, new, system, m...

- And also applied the lemmatizing to the stop words.

```
# lemmatization also comvert words to its root form
# But the only difference between lemmatization and stemmeing is that stemming just removes or stem
# lemmatization considers the context and converts the word to its meaningful base form, which is c


def lemmatization(data):
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(word) for word in data]
```

```
df['lemmatized_words'] = df.apply(lambda x: stemming(x['stopword_removed']),
                                  axis=1)
df[['data', 'lemmatized_words']].head()
```

	data	lemmatized_words
0	\n\nI am sure some bashers of Pens fans are pr...	[i, sure, basher, pen, fan, pretti, confus, la...
1	My brother is in the market for a high-perform...	[my, brother, market, video, card, support, ve...
2	\n\n\n\n\tFinally you said what you dream abou...	[final, said, dream, mediterranean, that, new,...
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	[think, it, scsi, card, dma, transfer, not, di...
4	1) I have an old Jasmine drive which I cann...	[i, old, jasmin, drive, i, use, new, system, m...

- The only difference between lemmatization and stemming is that stemming removes or stems the last few characters of a word, often leading to incorrect meanings and spelling.

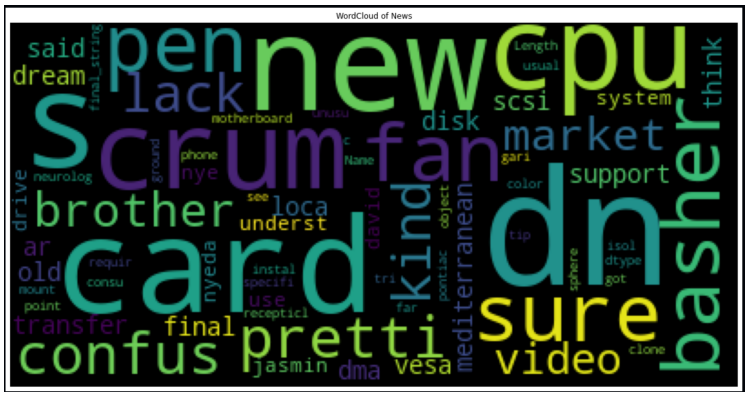
- Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma.
- The final step of the preprocessing is the joining of the lemmatized_words into a string.

```
# Function that join and return lemmatized words to string.
def word_rejoing(data):
    return (" ".join(data))
```

```
# Appling word_rejoing function to lemmatized_words and
# Stroing final string to dataframe with column name final_string
df['final_string'] = df.apply(lambda x: word_rejoing(x['lemmatized_words']),
                              axis=1)
df[['data', 'final_string']].head()
```

	data	final_string
0	\n\nI am sure some bashers of Pens fans are pr...	i sure basher pen fan pretti confus lack kind ...
1	My brother is in the market for a high-perform...	my brother market video card support vesa loca...
2	\n\n\n\n\tFinally you said what you dream abou...	final said dream mediterranean that new the ar...
3	\nThink!\n\nIt's the SCSI card doing the DMA t...	think it scsi card dma transfer not disk the s...
4	1) I have an old Jasmine drive which I cann...	i old jasmin drive i use new system my underst...

- At the of preprocessing, we again generated a word cloud on final_string, and we got a clean and clear view of the words.



5. Feature Extraction:

- Extracting features from the text data is called word vectorization or word embeddings. This is a very crucial part of Natural Language Processing because this feature vector will go into the model training. If this is not done correctly, the model will lead to misprediction.
- Word Vectorization is the technique of converting words to numerical form to perform model training on text-based data.
- For this problem, we have used two-word vectorizing techniques.

1. TF-IDF Vectorizer:

a. Term Frequency-Inverse Document Frequency is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

b. This is done by multiplying two metrics: how many times a word appears in a document and the inverse document frequency of the word across a set of documents.

c. Here we have used the TfidfVectorizer class of the Sklearn library to convert the words to vectors.

d.

```
# Creating instance of TfidfVectorzer function of sklearn library
tf_idf = TfidfVectorizer()
# Converting final_strings to its meaning full words vectors from dataframe
# Storing final traing data to X variable
X = tf_idf.fit_transform(df["final_string"])
# Storing Labels to Y variable
Y = df["target"]
# Printing Shape of data
X.shape
```

```
(18846, 57689)
```

e.
```
# Displaying average non-zero values from dataset
X.nnz / float(X.shape[0])
```
```
61.29640241961159
```

f. Here we discovered that from all features, 61.29% of data is none zero, and 39% of features contain zeros because of the uneven length of news text.

g. To train and evaluate the model, we split the dataset into train-test with a ratio of 80-20%.

h. We have trained the model with three different classifiers.

i. Naive Bayes Classifier

- With Naive Bayes, we got a training accuracy of 81% and test accuracy of 69%

```
# Defining naive bayes classifier model with
# fitting train data into model

naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train, y_train)
```
```
▾ MultinomialNB
MultinomialNB()
```

```
# getting training accuracy of model
y_pred = naive_bayes_classifier.predict(X_train)
NB_train_acc = metrics.accuracy_score(y_train, y_pred)
print("Train Accuracy : ", NB_train_acc)
tf_idf_train_acc["NaiveBayes"] = NB_train_acc

# Performing prediction on test dataset
y_pred = naive_bayes_classifier.predict(X_test)
NB_test_acc = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy : ", NB_test_acc)
tf_idf_test_acc["NaiveBayes"] = NB_test_acc
```
```
Train Accuracy :  0.8114884584770496
Test Accuracy :  0.6896551724137931
```

ii. SGD Classifier

- With SGD, we got a training accuracy of 85% and test accuracy of 70%

```
# Creating model of SGD Classifier with SGDClassifier class of sklearn
# Fitting the traing dataset into model
sgd = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random_state=42)
sgd.fit(X_train, y_train)
```
```
▾ SGDClassifier
SGDClassifier(alpha=0.001, random_state=42)
```

```
# getting training accuracy of model
y_pred = sgd.predict(X_train)
SGD_train_acc = metrics.accuracy_score(y_train, y_pred)
print("Train Accuracy : ", SGD_train_acc)
tf_idf_train_acc["SGD"] = SGD_train_acc

# Performing prediction on test dataset
y_pred = sgd.predict(X_test)
SGD_test_acc = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy : ", SGD_test_acc)
tf_idf_test_acc["SGD"] = SGD_test_acc
```
```
Train Accuracy :  0.857389227911913
Test Accuracy :  0.7026525198938992
```

iii. Support Vector Machine(SVM) Classifier

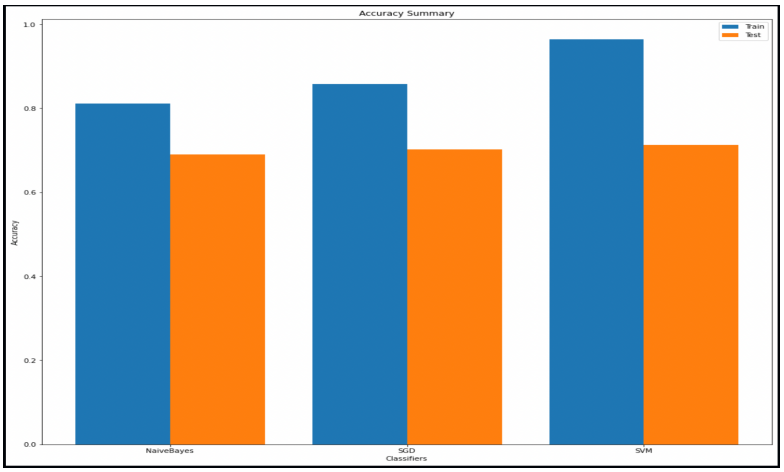
- With SVM, we got a training accuracy of 96% and test accuracy of 71%

```
svm = SVC()
svm.fit(X_train, y_train)
```
```
▾ SVC
SVC()
```

```
# getting training accuracy of model
y_pred = svm.predict(X_train)
SVM_train_acc = metrics.accuracy_score(y_train, y_pred)
print("Train Accuracy : ", SVM_train_acc)
tf_idf_train_acc["SVM"] = SVM_train_acc

# Performing prediction on test dataset
y_pred = svm.predict(X_test)
SVM_test_acc = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy : ", SVM_test_acc)
tf_idf_test_acc["SVM"] = SVM_test_acc
```
```
Train Accuracy :  0.964181480498806
Test Accuracy :  0.7124668435013263
```

iv. Train-Test Summary:



- From all three classification models, Support Vector Machine Classifier performed well with train accuracy of 96% and test accuracy of 71% other than Naïve Bayes with 81%-69% Train-Test and SGD with 85%-70% train-test accuracy.

2. Word2Vect Vectorizer:

a. Word2vec is a technique for natural language processing to convert words into vectors. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest different words for a partial sentence.

b.
```
# Splitting the lemmatized_wrods list into train test dataset
X_train, X_test, y_train, y_test = train_test_split(df['lemmatized_words'],
                                                    df['target'],
                                                    test_size=0.2)
```

```
# Train the word2vec model with vector_size 800 and window size 600
w2v_model = gensim.models.Word2Vec(X_train,
                                   vector_size=500,
                                   window=100,
                                   min_count=20)
```

c. Here we have taken the lemmatized_words for training the network.

d. We have trained the word2vec model using Gensim's Word2Vec function with vector_size=500 and window=100 with a min_count of 20.

e.
```
# Generating aggregated sentence vectors based on the word vectors for each word in the sentence
words = set(w2v_model.wv.index_to_key)
X_train_vect = np.array(
    [np.array([w2v_model.wv[i] for i in ls if i in words]) for ls in X_train])
X_test_vect = np.array(
    [np.array([w2v_model.wv[i] for i in ls if i in words]) for ls in X_test])
```

f. Generating aggregated sentence vectors based on the word vectors for each word in the sentence

g.
```
# Compute sentence vectors by averaging the word vectors for the words contained in the sentence
X_train_vect_avg = []
for v in X_train_vect:
    if v.size:
        X_train_vect_avg.append(v.mean(axis=0))
    else:
        X_train_vect_avg.append(np.zeros(500, dtype=float))

X_test_vect_avg = []
for v in X_test_vect:
    if v.size:
        X_test_vect_avg.append(v.mean(axis=0))
    else:
        X_test_vect_avg.append(np.zeros(500, dtype=float))
```

h. Compute final sentence vectors by averaging the word vectors for the words contained in the sentence

i. We have performed model training with three different classifiers.

i. Stochastic Gradient Descent(SGD) Classifier

- We have achieved a training accuracy of 59.9 % and a test accuracy of 59.3% with the SGD Classifier.

```
# Creating SGD Classifier model and
sgd = SGDClassifier()
sgd.fit(X_train_vect_avg, y_train)
```

```
SGDClassifier()
```

```
# getting training accuracy of model
y_pred = sgd.predict(X_train_vect_avg)
SGD_train_acc = metrics.accuracy_score(y_train, y_pred)
print("Train Accuracy : ", SGD_train_acc)
w2v_train_acc["SGD"] = SGD_train_acc

# Performing prediction on test dataset
y_pred = sgd.predict(X_test_vect_avg)
SGD_test_acc = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy : ", SGD_test_acc)
w2v_test_acc["SGD"] = SGD_test_acc
```

```
Train Accuracy :  0.599230565136641
Test Accuracy :  0.5952254641909814
```

ii. Random Forest Classifier

- With a random forest classifier, we got a training accuracy of 97% and test accuracy of 61%

```
# Train data with randomforest classifer
rf = RandomForestClassifier()
rf_model = rf.fit(X_train_vect_avg, y_train)
```

```
# getting training accuracy of model
y_pred = rf_model.predict(X_train_vect_avg)
rf_train_acc = metrics.accuracy_score(y_train, y_pred)
print("Train Accuracy : ", rf_train_acc)
w2v_train_acc["RandomForest"] = rf_train_acc

# Performing prediction on test dataset
y_pred = rf_model.predict(X_test_vect_avg)
rf_test_acc = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy : ", rf_test_acc)
w2v_test_acc["RandomForest"] = rf_test_acc
```

```
Train Accuracy :  0.9710798620323693
Test Accuracy :  0.6127320954907162
```

iii. Support Vector Machine(SVM) Classifier

- Using the SVM classifier, we got a training accuracy of 96% and test accuracy of 71%

```
svm = SVC()
svm.fit(X_train_vect_avg, y_train)
```

```
SVC()
```

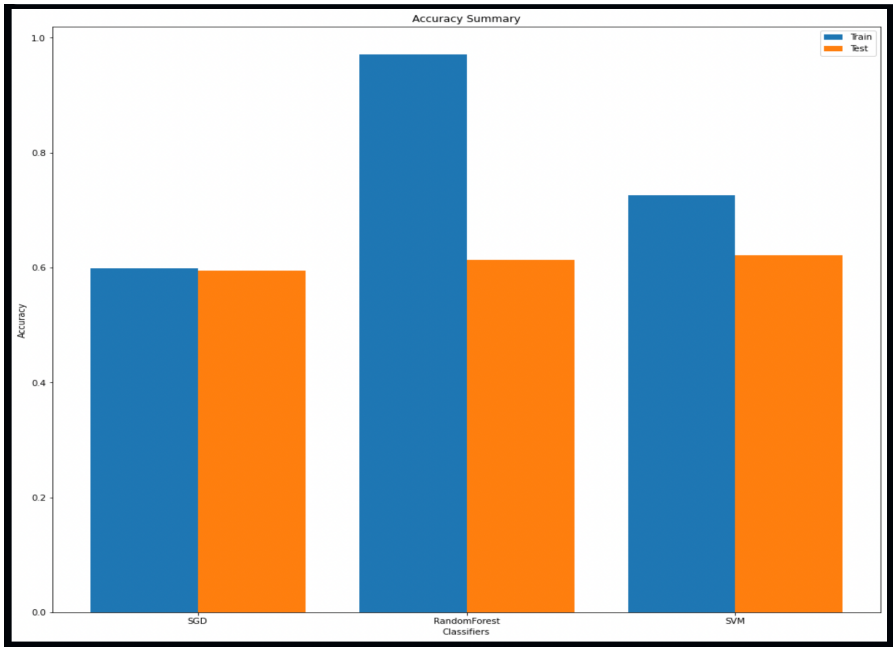
```
# getting training accuracy of model
y_pred = svm.predict(X_train_vect_avg)
SVM_train_acc = metrics.accuracy_score(y_train, y_pred)
print("Train Accuracy : ", SVM_train_acc)
w2v_train_acc["SVM"] = SVM_train_acc

# Performing prediction on test dataset
y_pred = svm.predict(X_test_vect_avg)
SVM_test_acc = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy : ", SVM_test_acc)
w2v_test_acc["SVM"] = SVM_test_acc
```

```
Train Accuracy :  0.7263863093658796
Test Accuracy :  0.6209549071618037
```

iv. Classifier Summary

- For the word2vec word embedding technique from all three classification models, the Support Vector Machine classifier performed well with the train-test, 73%-61% accuracy score,



Conclusion

- **To conclude, we performed classification on 20 newsgroup datasets with 2 different word embedding techniques with all text pre-processing steps with feature selection to train the classification models.**
- **We observe that the TF-IDF vectorization technique performed better for this problem, with a 71% accuracy score higher than Word2Vec embedding using the SVM Classifier.**
- **Because the Word2Vec technique is most suitable for recommendation problems, but this is the problem of Multiclass text classification, so for that TI-IDF is best.**