Project Report

on

COVID-19 Twitter Data Sentiment Analysis

Apurv Jain

2017KUCP1016

Natural Language Processing (CST401)

Dr. Basant Agarwal

24 December 2020

1. Introduction

The aim of this project is to analyse the Sentiments of the Tweets made during the COVID-19 Pandemic and classify them into three classes: Positive, Negative and Neutral.

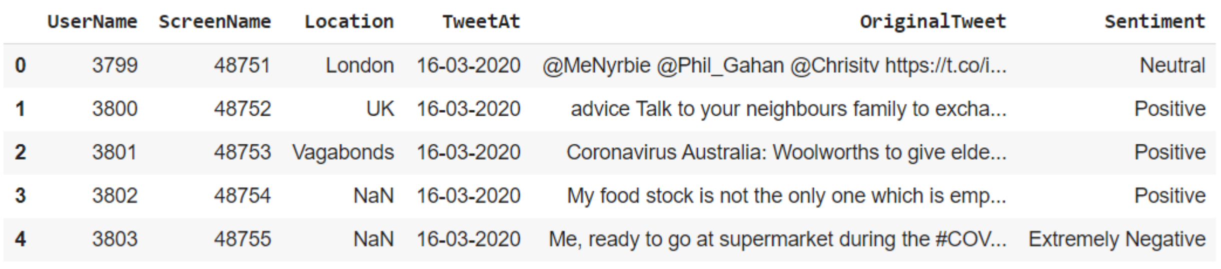
2. Dataset Used

The dataset is available on Kaggle [1]. The data consists of 6 columns (as shown in Fig. 1):

i. UserName
ii. ScreenName
iii. Location
iv. TweetAt
v. OriginalTweet
vi. Sentiment

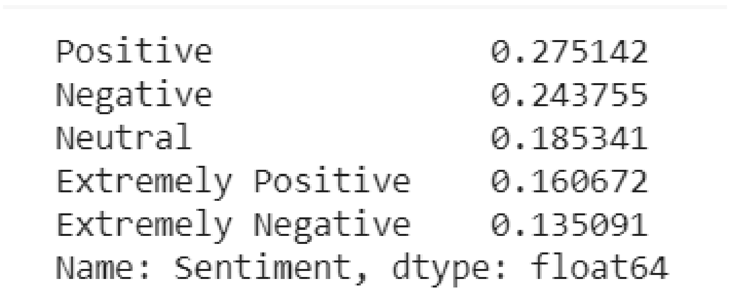
The Sentiment class consists of five classes: Extremely Positive, Positive, Neutral, Negative and Extremely Negative. The class distribution is shown in Fig. 2.

The Training Data consists of 41157 Tweets, while the Testing Data consists of 3798 Tweets, respectively.



	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative

Fig. 1 Original Dataset (first 5 Rows)



```
Positive              0.275142
Negative              0.243755
Neutral               0.185341
Extremely Positive    0.160672
Extremely Negative    0.135091
Name: Sentiment, dtype: float64
```

Fig. 2 Class Distribution of the Dataset

For the ease of doing the analysis, the classes were reduced to three, namely: Positive, Neutral and Negative. The class distribution of the three classes is shown in Fig. 3, 4 and 5.

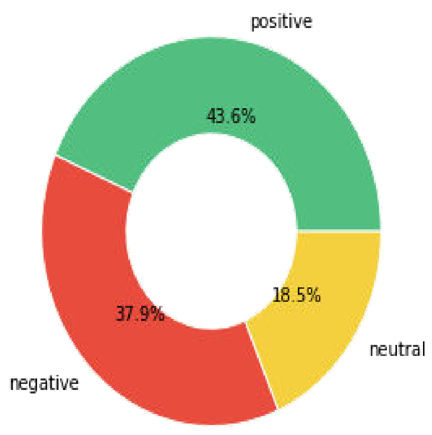
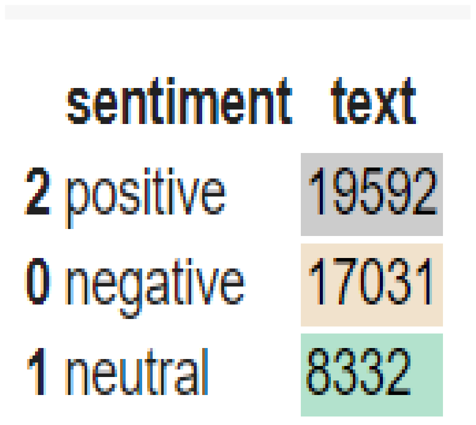
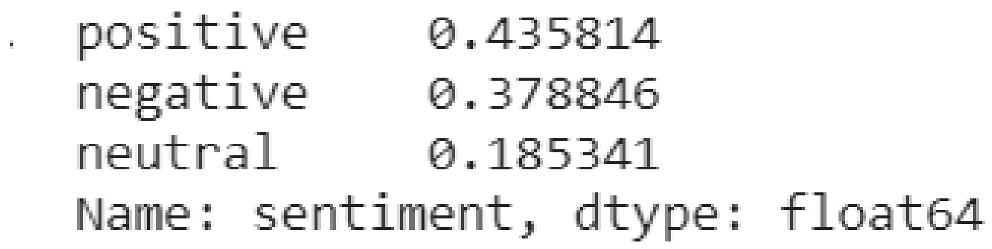


Fig. 3



	sentiment	text
2	positive	19592
0	negative	17031
1	neutral	8332

Fig. 4



```
positive    0.435814
negative    0.378846
neutral     0.185341
Name: sentiment, dtype: float64
```

Fig. 5

3. Data Analysis

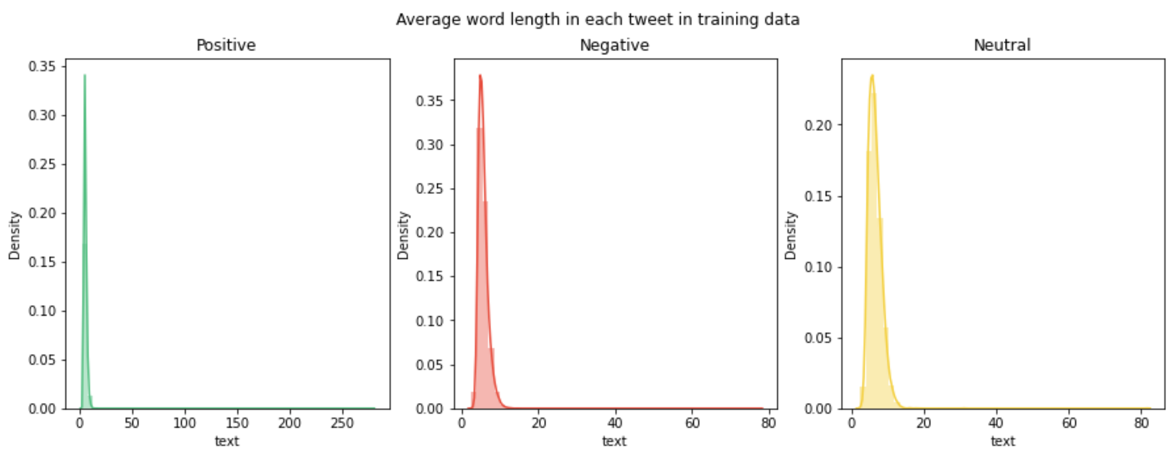


Fig. 6 Average word length in each tweet in training data

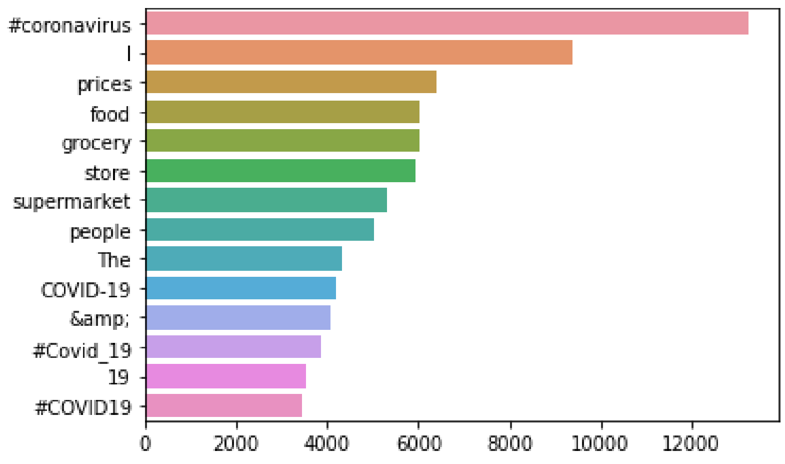


Fig. 7 Words with maximum Word Count

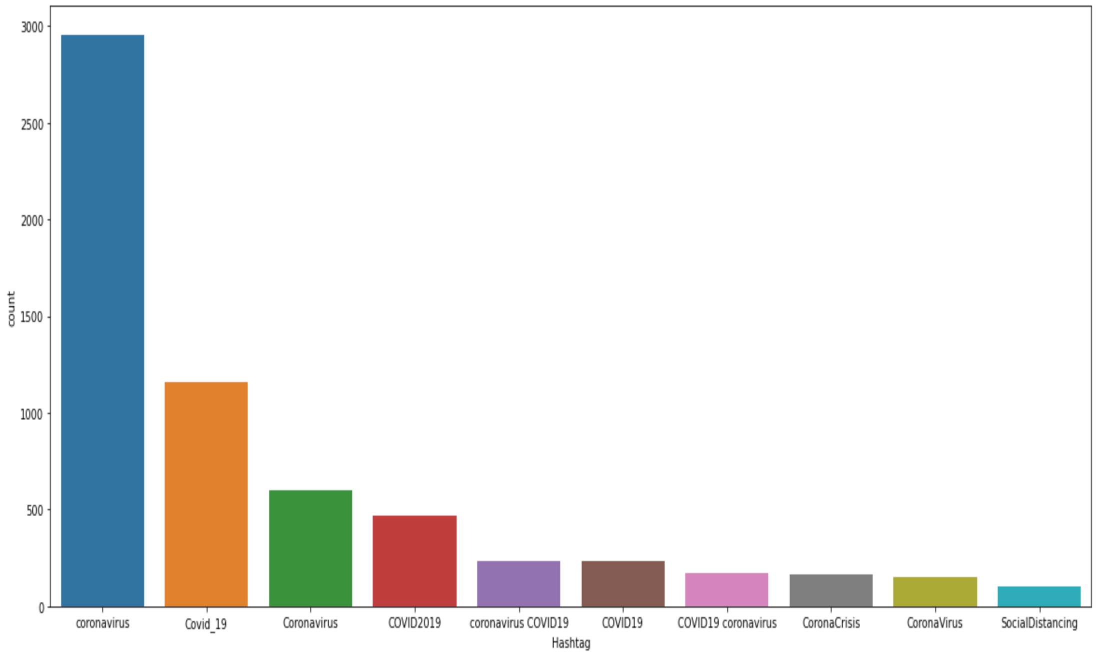


Fig. 8 Top 10 Hashtags

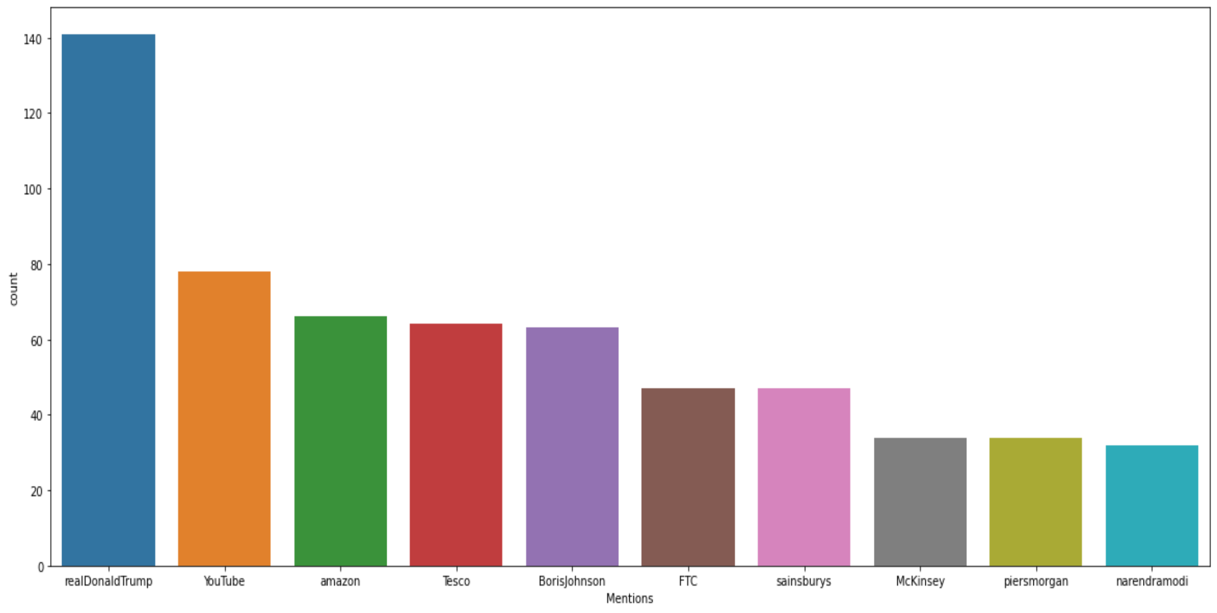


Fig. 9   Top 10 Mentions

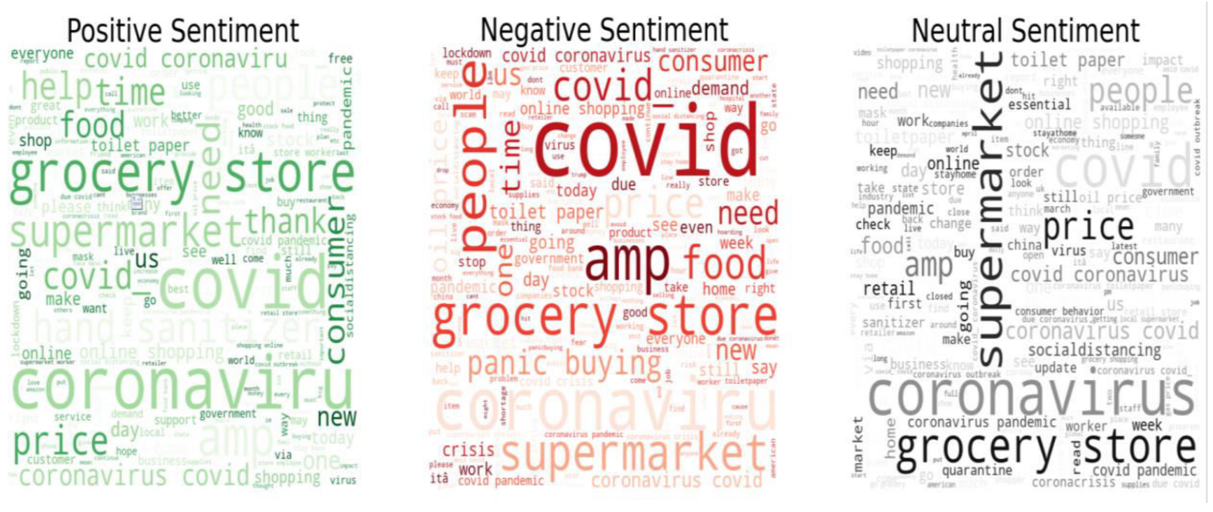


Fig. 10 Word Cloud for each class

4. Data Pre-Processing

The following data pre-processing steps were performed on the text of each tweet in the dataset:

i. Removing of URLs and HTML Links from Tweets
ii. Lower-casing of text
iii. Removing of numbers
iv. Removing of common English Stop words and Punctuations
v. Removing of extra whitespace left
vi. Conversion of classes from string to numbers:
   - ⇒ 0 for 'Neutral'
   - ⇒ 1 for 'Negative'
   - ⇒ 2 for 'Positive'

5. Models Used

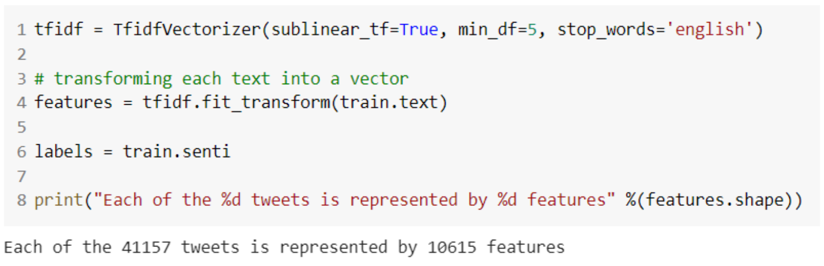
The following models were used for the analysis:

i. Random Forests, Linear SVM, Multinomial Naïve Bayes using TF-IDF.
ii. A simple CNN Network.
iii. CNN using GloVe Embedding (for Twitter Data).
iv. CNN using Word2Vec Embedding (generated using Gensim).

6. Results

A. Random Forests, Linear SVM, Multinomial Naïve Bayes using TF-IDF

The TF-IDF Vectorizer (available in sklearn) was used for generating the Vector Representation of Tweets (as shown in Fig. 11).



```
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, stop_words='english')

# transforming each text into a vector
features = tfidf.fit_transform(train.text)

labels = train.senti

print("Each of the %d tweets is represented by %d features" %(features.shape))
```

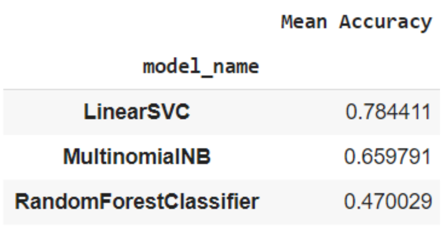
```
Each of the 41157 tweets is represented by 10615 features
```

Fig. 11

A total of 10615 TF-IDF Vector features were generated. The 'sublinear_tf = True' parameter was used to replace the value of 'tf' with '1+log(tf)'. The parameter, 'min_df = 5' was used to ignore the terms with document-frequency less than 5.

a. Comparison between Linear SVM, Random Forests and Multinomial Naïve Bayes

The performance of the three models was compared using 5-fold Cross Validation. Among all, Linear SVM performed the best with 78% Accuracy (as shown in Fig. 12). The 'LinearSVC' model available in 'sklearn' uses One v/s Rest strategy for multi-class classification. More about this in Appendix.

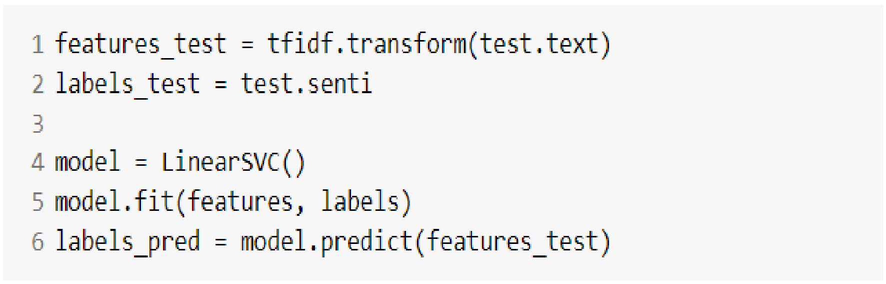


	Mean Accuracy
model_name	
LinearSVC	0.784411
MultinomialNB	0.659791
RandomForestClassifier	0.470029

Fig. 12

b. Using Linear SVM for Testing

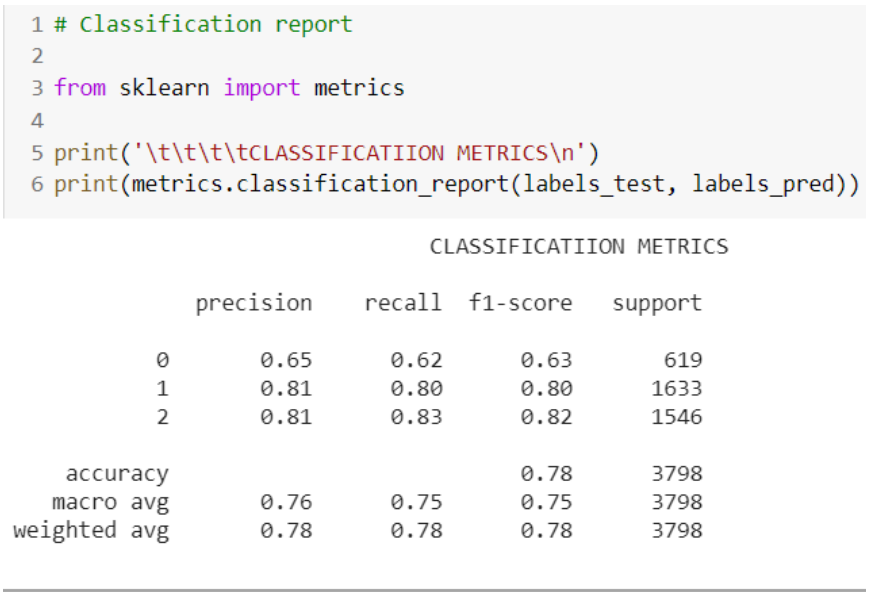
Since, Linear SVM performed the best among the three models, it was used for Testing. The testing accuracy achieved was 78% (as shown in Fig. 13 and 14).



```
features_test = tfidf.transform(test.text)
labels_test = test.senti

model = LinearSVC()
model.fit(features, labels)
labels_pred = model.predict(features_test)
```

Fig. 13



```
# Classification report

from sklearn import metrics

print('\t\t\t\tCLASSIFICATIION METRICS\n')
print(metrics.classification_report(labels_test, labels_pred))
```

```
                                CLASSIFICATIION METRICS

              precision    recall  f1-score   support

           0       0.65      0.62      0.63       619
           1       0.81      0.80      0.80      1633
           2       0.81      0.83      0.82      1546

    accuracy                           0.78      3798
   macro avg       0.76      0.75      0.75      3798
weighted avg       0.78      0.78      0.78      3798
```

Fig. 14 Classification Report for Testing (Linear SVM)

B. CNN Based Models

Three models were used based on CNN:

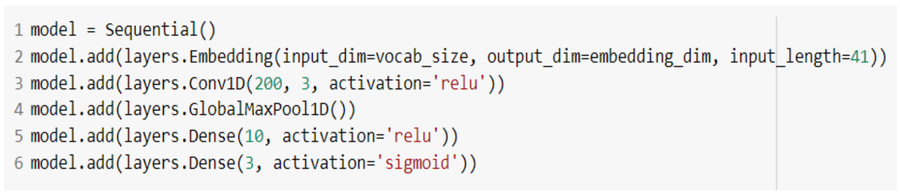
i. Simple CNN
ii. CNN using GloVe Twitter Embedding
iii. CNN using Word2Vec Embedding (generated using Gensim)

Before using these models, some more pre-processing steps were applied, given as follows:

i. One-hot encoded the labels (classes).
ii. Pad-sequences with 'maxlen = 41' (the maximum length of a sentence in the dataset)
iii. Used Tokenizer available in Keras. Vocab Size of the vectors was 59192.

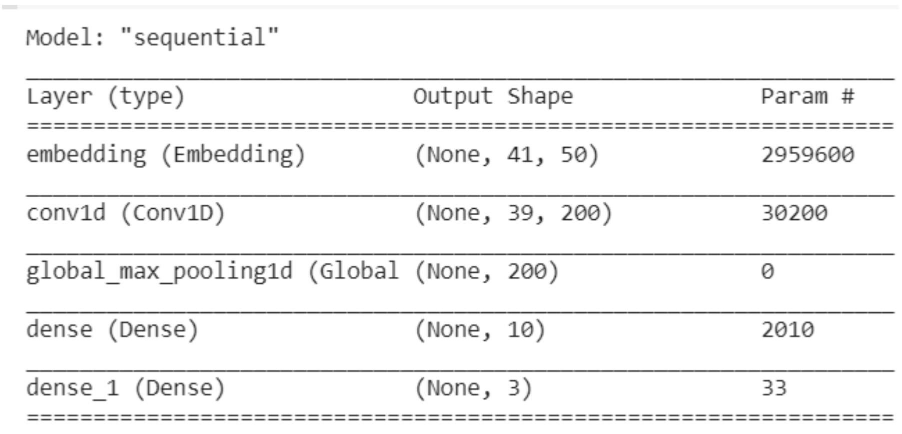
a. Simple CNN Model

Firstly, a simple CNN model was used over the Vectors generated using Keras, with 'Embedding size as 50', 'Filter Size as 200' and 'Kernel Size as 3'. The layers of the architecture are shown in Fig. 15 and 16.



```python
model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=41))
model.add(layers.Conv1D(200, 3, activation='relu'))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(3, activation='sigmoid'))
```

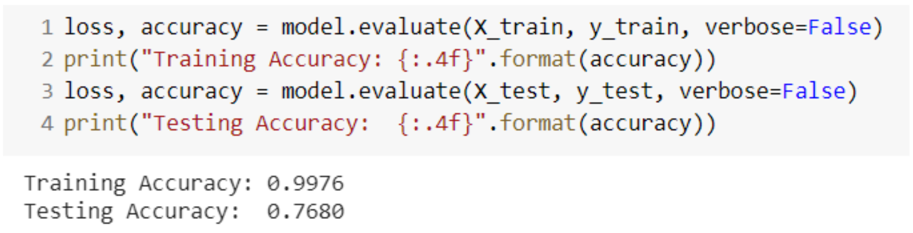
Fig. 15



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 41, 50)            2959600
_________________________________________________________________
conv1d (Conv1D)              (None, 39, 200)           30200
_________________________________________________________________
global_max_pooling1d (Global (None, 200)               0
_________________________________________________________________
dense (Dense)                (None, 10)                2010
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 33
=================================================================
```

Fig. 16

The Testing Accuracy of 76.8% was achieved using this model (Fig. 17).



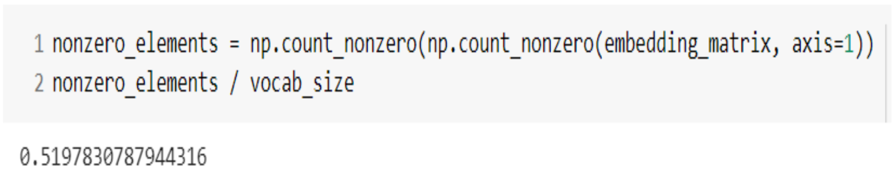
```python
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9976
Testing Accuracy:  0.7680
```

Fig. 17

b. CNN Model using GloVe Embedding

The Embedding Matrix for this model was generated using GloVe Twitter Data pre-trained vectors, available at [2]. The embedding dimensions used were 50 and embedding was generated for about 52% of the Vocabulary (Fig, 18).

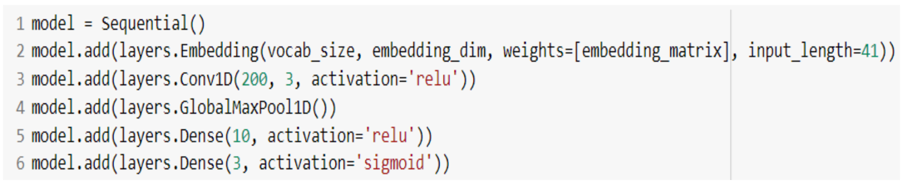


```
nonzero_elements = np.count_nonzero(np.count_nonzero(embedding_matrix, axis=1))
nonzero_elements / vocab_size
```

```
0.5197830787944316
```

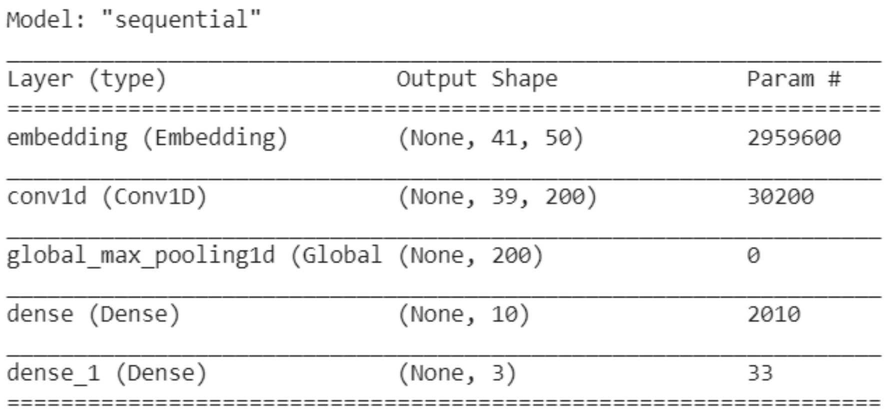
Fig. 18

The layers of the architecture are shown in Fig. 19 and 20.



```
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, weights=[embedding_matrix], input_length=41))
model.add(layers.Conv1D(200, 3, activation='relu'))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(3, activation='sigmoid'))
```

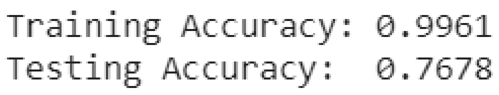
Fig. 19



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 41, 50)            2959600
_________________________________________________________________
conv1d (Conv1D)              (None, 39, 200)           30200
_________________________________________________________________
global_max_pooling1d (Global (None, 200)               0
_________________________________________________________________
dense (Dense)                (None, 10)                2010
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 33
=================================================================
```

Fig. 20

The testing accuracy achieved was 76.78%, using this model (as shown in Fig. 21).



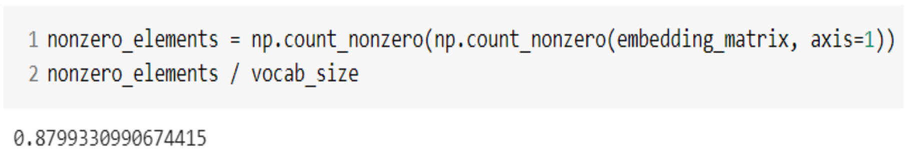
```
Training Accuracy: 0.9961
Testing Accuracy:  0.7678
```

Fig. 21

c. **CNN Model using Word2Vec Embedding**

The Embedding Matrix was generated using the Word2Vec Skip-gram vectors and CBOW vectors of dimensions 50 each. These vectors were generated using Gensim module [3].

The Embedding Matrix was generated for about 88% of vocabulary (Fig. 22).

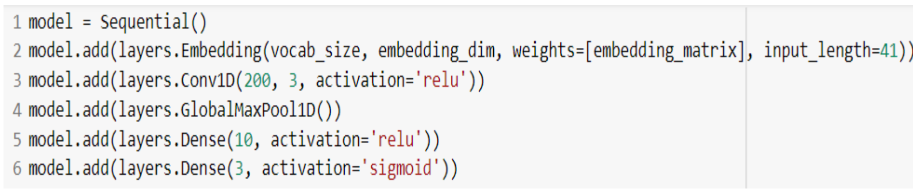


```
nonzero_elements = np.count_nonzero(np.count_nonzero(embedding_matrix, axis=1))
nonzero_elements / vocab_size
```

```
0.8799330990674415
```

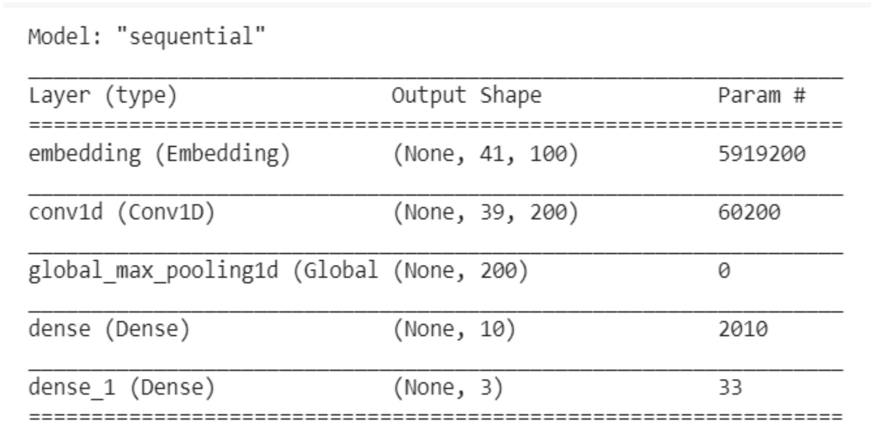
Fig. 22

The resulting dimension of the Embedding Matrix in this case was, 100 (50 + 50). The Layers of the architecture of this model are shown in Fig. 23 and 24.



```
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, weights=[embedding_matrix], input_length=41))
model.add(layers.Conv1D(200, 3, activation='relu'))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(3, activation='sigmoid'))
```

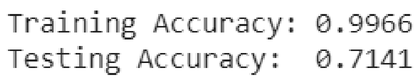
Fig. 23



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 41, 100)           5919200   
_________________________________________________________________
conv1d (Conv1D)              (None, 39, 200)           60200     
_________________________________________________________________
global_max_pooling1d (Global (None, 200)               0         
_________________________________________________________________
dense (Dense)                (None, 10)                2010      
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 33        
=================================================================
```

Fig. 24

The testing accuracy achieved using this model was 71.41% (as shown in Fig. 25).



```
Training Accuracy: 0.9966
Testing Accuracy:  0.7141
```

Fig. 25

7. Results Comparison

The Comparison between Linear SVM and CNN based models is presented in Table I.

Table I Comparison between Linear SVM and CNN based models

Model	**Tokenizer Used**	**Testing Accuracy**
Linear SVM	TF-IDF	78%
Simple CNN	Keras	76.8%
CNN (GloVe Embedding)	Keras	76.78%
CNN (Word2Vec Embedding)	Keras	71.41%

Appendix
Using Linear SVM for Multi-class Classification

SVM, by default, separates the data points into two classes using a hyperplane in two dimensions, that is, a line; and thus performs binary classification.

But, it does not support Multi-class classification, rather it can use wither of the two approaches, based on the above principle: i) One v/s One or ii) One v/s Rest [4].

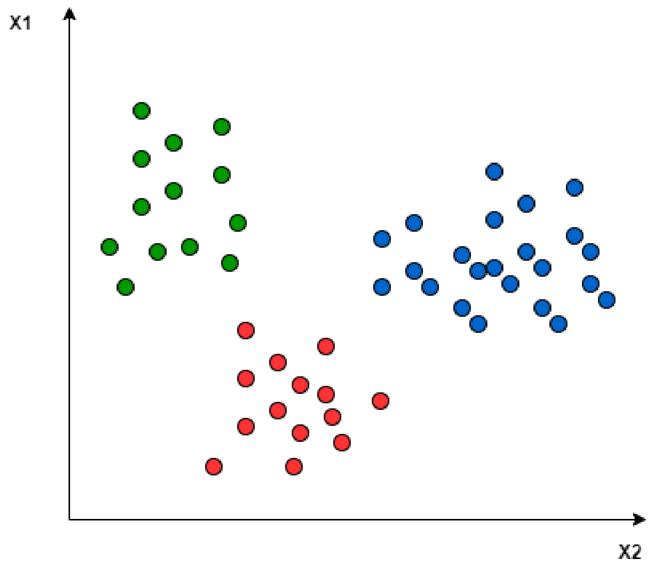


Fig. 26

The One v/s One approach maps the data points into high-dimensional space and separated them using linear separation between every pair of classes.

For example, for the points given in Fig. 26, the points can be separated into three classes (represented by Red, Blue and Green respectively), using One v/s One approach as shown in Fig. 27. In this approach, the points taken into account for separation belong to only two classes at one time. Therefore, in Fig. 27, the blue-green line separates the points belonging to the blue and green class respectively. The similar approach is followed for the other pairs of classes as well. Thus, it breaks down the problem for **k** classes into $\frac{k(k-1)}{2}$ binary classification problems.

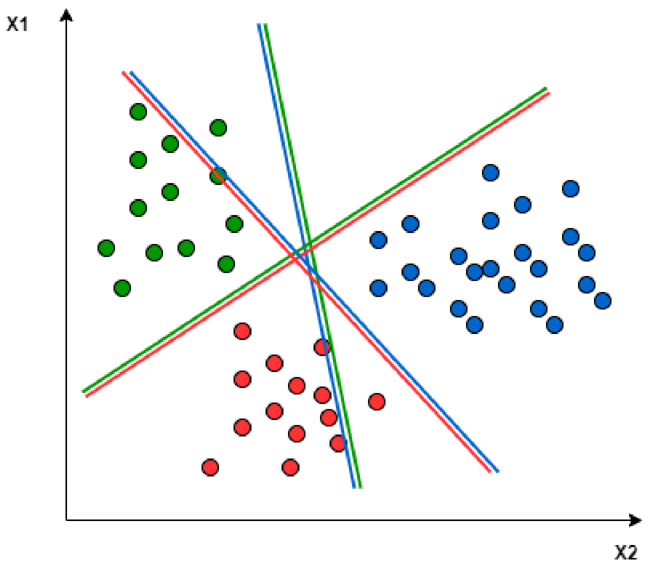


Fig. 27 One v/s One

In the One v/s Rest approach, binary classification is performed for each class. The separation is performed between points belonging to a particular class and all other points, at a time. Therefore, at one instance, the points are divided into two groups: one of the points belonging to one of the classes and other group of the rest of the points. Thus, for **k** classes, **k** linear SVMs are used, with each SVM predicting for each of the **k** classes.

For example, for the points given in Fig. 26, the One v/s Rest approach can be applied as shown in Fig. 28. Here, the Red line separates the points of the class represented by Red colour from all the other points. Similar task is performed for each of the other classes as well.

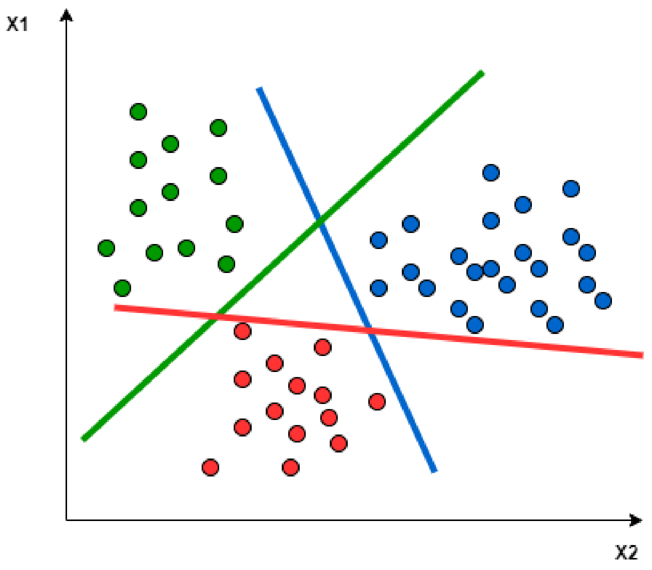


Fig. 28 One v/s Rest

The 'LinearSVC' model available in 'sklearn' library (used in this project) uses 'One v/s Rest' approach for Multi-class classification problems [5].

References

[1] *Coronavirus tweets NLP - Text Classification*, Kaggle, Sept. 2020. [Online]. Available: http://www.kaggle.com/datatattle/covid-19-nlp-text-classification.

[2] J. Pennington, R. Socher, C. D. Manning. *GloVe: Global Vectors for Word Representation*. (2014). [Online]. Available: http://nlp.stanford.edu/projects/glove/.

[3] R. Řehůřek. *models.word2vec – Word2vec embeddings*. Gensim 4.0.0. (2009). [Online]. Available: http://radimrehurek.com/gensim/models/word2vec.html.

[4] Baeldung, "Multiclass Classification Using Support Vector Machines." baeldung.com. http://www.baeldung.com/cs/svm-multiclass-classification.

[5] *sklearn.svm.LinearSVC*. scikit-learn (2007). [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.

Note: For the information regarding various files present in the project directory, please refer to the 'FileOfFiles.txt' file. The project code can be executed by running the corresponding Jupyter Notebook.