Report

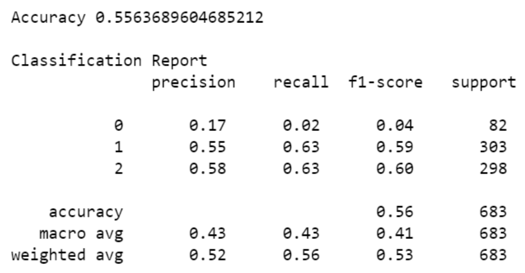
BASELINE METHODS:

1) TFIDF + Classifiers:
   TFIDF consists of two words – TF (Term Frequency) and IDF (Inverse Document Frequency). The idea of TFIDF is to give importance to the words that occur more frequently in one document and less frequently in another document. It diminishes the weight of words that appeared common in many documents.

   Steps:
   - Install libraries and load the dataset (train/test)
   - Preprocess the sentences – (Lowercase sentence and remove punctuations & stop words)
   - TFIDF Vectorizer – Fit transform on the training data which generates tf-idf word count vector. Transform on testing data will generate a term document matrix using the same vocabulary generated in training.
   - Machine Learning models such as SVC and RFC to classify the sentiments.

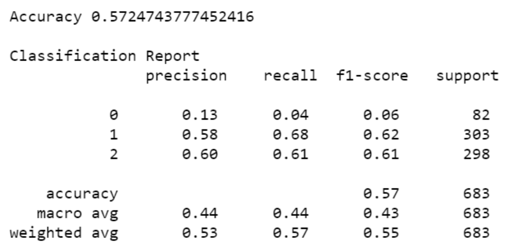
   Results:

RFC



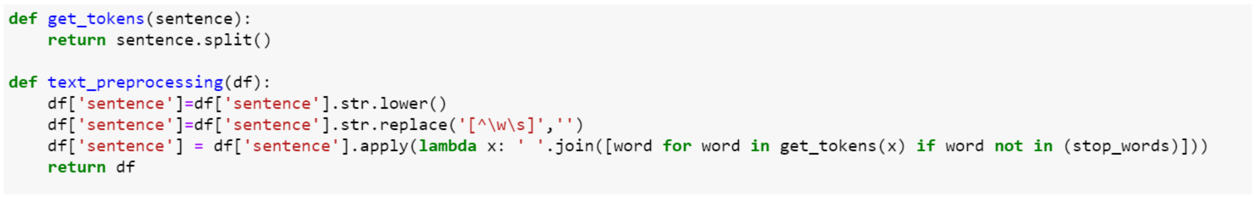
```
Accuracy 0.5563689604685212

Classification Report
              precision    recall  f1-score   support

           0       0.17      0.02      0.04        82
           1       0.55      0.63      0.59       303
           2       0.58      0.63      0.60       298

    accuracy                           0.56       683
   macro avg       0.43      0.43      0.41       683
weighted avg       0.52      0.56      0.53       683
```

SVC

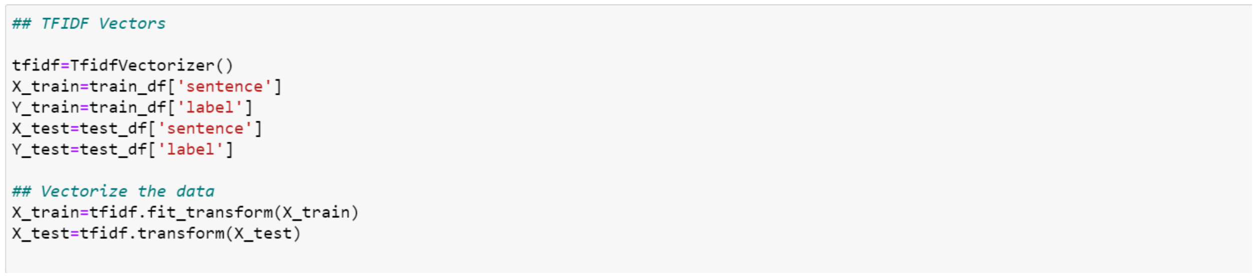


```
Accuracy 0.5724743777452416

Classification Report
              precision    recall  f1-score   support

           0       0.13      0.04      0.06        82
           1       0.58      0.68      0.62       303
           2       0.60      0.61      0.61       298

    accuracy                           0.57       683
   macro avg       0.44      0.44      0.43       683
weighted avg       0.53      0.57      0.55       683
```

Code Snippets:



```python
def get_tokens(sentence):
    return sentence.split()

def text_preprocessing(df):
    df['sentence']=df['sentence'].str.lower()
    df['sentence']=df['sentence'].str.replace('[^\w\s]','')
    df['sentence'] = df['sentence'].apply(lambda x: ' '.join([word for word in get_tokens(x) if word not in (stop_words)]))
    return df
```



```python
## TFIDF Vectors

tfidf=TfidfVectorizer()
X_train=train_df['sentence']
Y_train=train_df['label']
X_test=test_df['sentence']
Y_test=test_df['label']

## Vectorize the data
X_train=tfidf.fit_transform(X_train)
X_test=tfidf.transform(X_test)
```

```
# Support Vector Classifier
svc=LinearSVC()
svc.fit(X_train,Y_train)
y_pred=svc.predict(X_test)
print('Accuracy %s \n' % accuracy_score(y_pred, Y_test))
print ("Classification Report \n",classification_report(Y_test,y_pred))
```

2) Word2vec + ML Classifier (Gensim)

Word2vec - It is basically a mapping of words to vector representations. It inputs a text corpus and outputs a set of vectors (feature vectors). I have used the Google's pretrained model on google news data. We build the vocabulary for our model by also considering pretrained model's vocabulary. We train on our sentences in order to add new words to the pre trained model.

Steps:

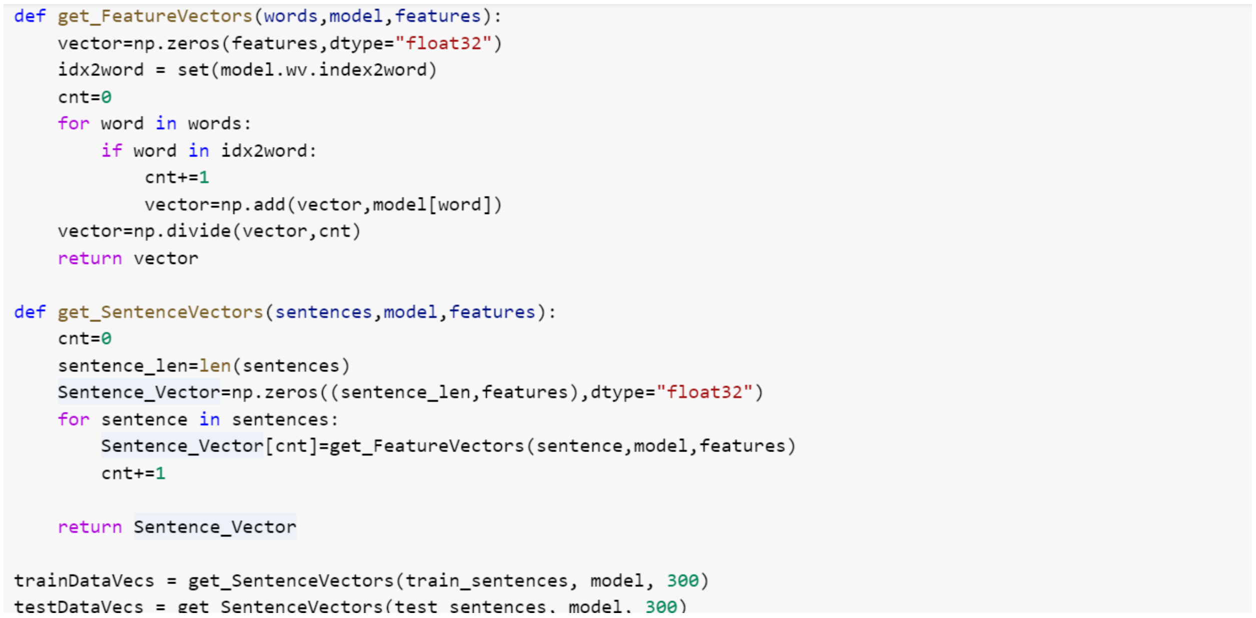
- Download Google's pretrained model , install libraries and load the dataset (train/test)
- Preprocess the sentences – (Lowercase sentence and remove punctuations & stop words)
- Retrain the model by updating the vocabulary with new words from training dataset.
- Get the vector representation for sentences. We determine the vectors by averaging the vector scores.
- Machine Learning model such as RFC are used to classify the sentiments.

Results:

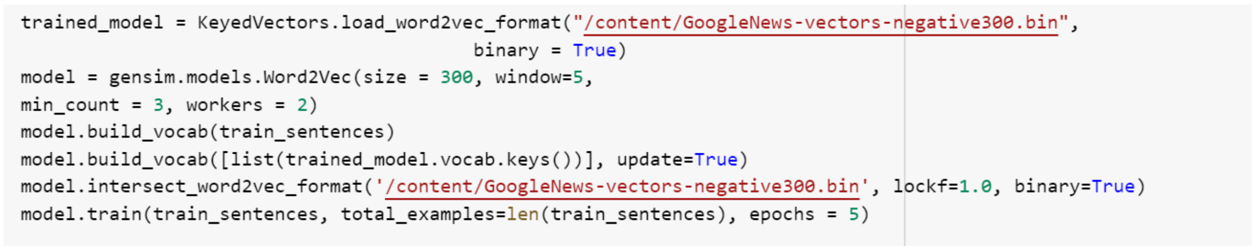
```
Fitting random forest to training data....
accuracy 0.4612005856515373
              precision    recall  f1-score   support

           0       0.08      0.11      0.10        82
           1       0.51      0.57      0.54       303
           2       0.56      0.45      0.50       298

    accuracy                           0.46       683
   macro avg       0.38      0.38      0.38       683
weighted avg       0.48      0.46      0.47       683
```

Code Snippet:

```
def get_tokens(sentence):
    return sentence.split()

def text_preprocessing(df):
    stop_words=stopwords.words('english')
    df['sentence']=df['sentence'].str.lower()
    df['sentence']=df['sentence'].str.replace('[^\w\s]','')
    df['sentence'] = df['sentence'].apply(lambda x: ' '.join([word for word in get_tokens(x) if word not in (stop_words)]))
    return df
```



```
def get_FeatureVectors(words,model,features):
    vector=np.zeros(features,dtype="float32")
    idx2word = set(model.wv.index2word)
    cnt=0
    for word in words:
        if word in idx2word:
            cnt+=1
            vector=np.add(vector,model[word])
    vector=np.divide(vector,cnt)
    return vector

def get_SentenceVectors(sentences,model,features):
    cnt=0
    sentence_len=len(sentences)
    Sentence_Vector=np.zeros((sentence_len,features),dtype="float32")
    for sentence in sentences:
        Sentence_Vector[cnt]=get_FeatureVectors(sentence,model,features)
        cnt+=1

    return Sentence_Vector

trainDataVecs = get_SentenceVectors(train_sentences, model, 300)
testDataVecs = get_SentenceVectors(test_sentences, model, 300)
```



```
trained_model = KeyedVectors.load_word2vec_format("/content/GoogleNews-vectors-negative300.bin",
                                                   binary = True)
model = gensim.models.Word2Vec(size = 300, window=5,
min_count = 3, workers = 2)
model.build_vocab(train_sentences)
model.build_vocab([list(trained_model.vocab.keys())], update=True)
model.intersect_word2vec_format('/content/GoogleNews-vectors-negative300.bin', lockf=1.0, binary=True)
model.train(train_sentences, total_examples=len(train_sentences), epochs = 5)
```

OTHER METHODS:

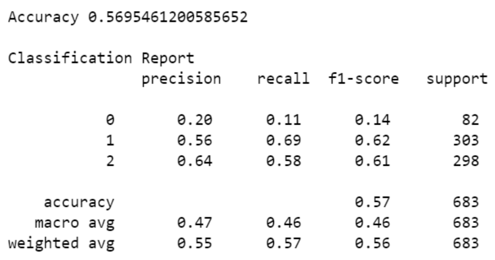
Other methods apart from baseline was to try out with spacy’s pretrained model and BERT model. Spacy has its inbuilt way of determining vector representation for sentences and seems to give better results as compared to baseline models. BERT uses its inbuilt preprocessing techniques and neural network architecture to get better results.

1) Word2vec + ML Classifier (Spacy)
   Word2vec - It is basically a mapping of words to vector representations. It inputs a text corpus and outputs a set of vectors (feature vectors). Spacy has a pretrained model ('en_core_web_lg') which is trained on written text in English language. It generates a mean vector for entire sentence which can be used in determining the sentiment of the sentences.

   Steps:
   - Download spacy language model, install libraries and load the dataset (train/test)
   - Preprocess the sentences – (Lowercase sentence and remove punctuations & stop words)
   - Get the vector representation for sentences. The spacy library gives the mean vector for the entire sentence as an output.
   - Reshape the vectors to make it suitable for input to Machine Learning Model.
   - Machine Learning models such as SVC and RFC are used to classify the sentiments.

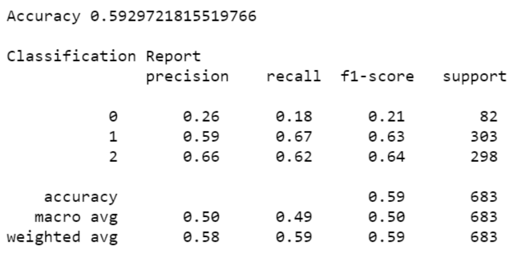
   Results:

   RFC



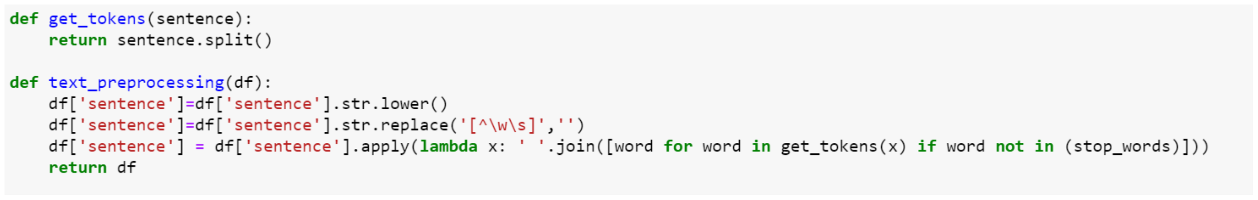
```
Accuracy 0.5695461200585652

Classification Report
              precision    recall  f1-score   support

           0       0.20      0.11      0.14        82
           1       0.56      0.69      0.62       303
           2       0.64      0.58      0.61       298

    accuracy                           0.57       683
   macro avg       0.47      0.46      0.46       683
weighted avg       0.55      0.57      0.56       683
```

   SVC

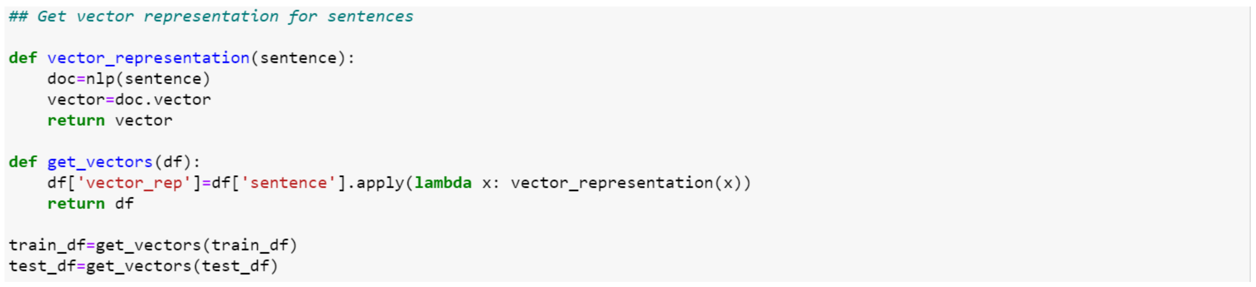


```
Accuracy 0.5929721815519766

Classification Report
              precision    recall  f1-score   support

           0       0.26      0.18      0.21        82
           1       0.59      0.67      0.63       303
           2       0.66      0.62      0.64       298

    accuracy                           0.59       683
   macro avg       0.50      0.49      0.50       683
weighted avg       0.58      0.59      0.59       683
```

   Code Snippet:



```python
def get_tokens(sentence):
    return sentence.split()

def text_preprocessing(df):
    df['sentence']=df['sentence'].str.lower()
    df['sentence']=df['sentence'].str.replace('[^\w\s]','')
    df['sentence'] = df['sentence'].apply(lambda x: ' '.join([word for word in get_tokens(x) if word not in (stop_words)]))
    return df
```



```python
## Get vector representation for sentences

def vector_representation(sentence):
    doc=nlp(sentence)
    vector=doc.vector
    return vector

def get_vectors(df):
    df['vector_rep']=df['sentence'].apply(lambda x: vector_representation(x))
    return df

train_df=get_vectors(train_df)
test_df=get_vectors(test_df)
```

```
# Support Vector Classifier
svc=LinearSVC()
svc.fit(train_X,train_Y)
y_pred=svc.predict(test_X)
print('Accuracy %s \n' % accuracy_score(y_pred, test_Y))
print ("Classification Report \n",classification_report(test_Y,y_pred))
```

2) BERT MODEL (PROPOSED)

BERT is a pretrained model on generic datasets and is useful in sentiment analysis of sentences due to it's huge training corpus. Here we used the latest pretrained model (uncased_L-12_H-768_A-12) which ensures every sentence are lower cased before tokenizing. ktrain is a deep learning Tensorflow Keras library which is used to build and train neural network models.

STEPS:

- Download ktrain library, stopwords (nltk) and then install libraries.
- Load the datasets. Consider 10% of train data for validating bert model.
- Ktrain library downloads the bert model weights and is used to train the dataset.
- The preprocessing is done based on inbuilt Bert's preprocessing techniques.
- Build the Bert model on train data and fit the data on the model.

Results:

```
              precision    recall  f1-score   support

           0       0.34      0.39      0.36        72
           1       0.72      0.68      0.70       319
           2       0.72      0.74      0.73       292

    accuracy                           0.67       683
   macro avg       0.59      0.60      0.60       683
weighted avg       0.68      0.67      0.68       683
```

```
0.6734992679355783
```

Code Snippets:

```
## Preprocess the dataset based on BERT's inbuilt preprocessing
(X_train,Y_train),(X_val,Y_val),preprocess=text.texts_from_df(train_df=train_df,
                   text_column='sentence',
                   label_columns='label',
                   val_df=val_df,
                   maxlen=100,
                   preprocess_mode='bert')
```

```
## Build the model classifier
model=text.text_classifier(name='bert',
                           train_data=(X_train,Y_train),
                           preproc=preprocess)
learner=ktrain.get_learner(model=model,
                           train_data=(X_train,Y_train),
                           val_data=(X_val,Y_val),
                           batch_size=32)
```

COMPARISION OF MODELS:

METRIC:

ACCURACY

- TFIDF – 0.57
- Word2vec (Gensim) – 0.46
- Word2vec (Spacy) – 0.59
- BERT – 0.67

The ML models in TFIDF give values in between 55-57% whereas Word2vec with google's pretrained model (Gensim) gives accuracy between 45-51%. To improve accuracy, I experimented with Word2vec spacy and Bert model. Word2vec spacy gives the vector values for sentences based on the pretrained model en_core_web_lg and gives accuracy between 57% - 61%. Out of all, the proposed model BERT gives the highest accuracy ranging between 62% - 68%. BERT considers its own preprocessing techniques which seems to perform better then the preprocessing techniques used in earlier models.

The F-1 score seems to be very close for class-1 and class-2 labels in all the models. The F-1 score is low for class 0 labels in all the models.

Further, I experimented with consideration of Stemming of the words (Porter Stemming). However, I could not see any significant improvement in the accuracy for the baseline models.

Other findings from models:

- In all the models, we see that the model performs poorly to classify class label 0 as compared to other classes.
- All the models except BERT has higher precision value for class 2 labels and higher recall value for class 1 labels.

Based on accuracy and F-1 Score, BERT model seems to perform best over all other models in classifying the sentiments. However, it can be more improved by considering a larger training data and finding a very good learning rate for the model.

REFERENCES:

- Ktrain (TensorFlow/Keras Library) - https://github.com/amaiya/ktrain
- NLP Articles on Word2vec, TFIDF and BERT