Sentiment Analysis of Online Product Reviews

Venkatesh Gopinath, Monica Ram, Mahitha Guduri
Dept. of Mechanical and Industrial Engineering
College of Engineering, Northeastern University
Boston, MA
bogem.v@northeastern.edu, ram.m@northeastern.edu, guduri.m@northeastern.edu

***Abstract*—Sentiment analysis is a method to extract opinions and attitudes from text data using natural language processing (NLP). The aim of this research project is to perform sentiment analysis on Amazon customer reviews using two different techniques from NLTK: VADER (Valence Aware Dictionary and Sentiment Reasoner) and Roberta Pretrained Model from Hugging Face and three Neural Network models. We will preprocess the text data, perform tokenization, and train the model using the Keras deep learning library.**

***Index Terms*—Reviews, VADER, Roberta, Sequential, LSTM, glove embedding, Pad Sequences, Keras.**

I. Introduction

Sentiment analysis has become increasingly popular with the increasing amount of user-generated content on the internet. This technique is used to extract subjective information from text data sources such as reviews, social media posts, and news articles. Sentiment analysis can be used to understand the overall sentiment of a particular brand or product, analyze customer feedback, and identify trends in customer behavior. In recent years, deep learning techniques, such as neural networks, have shown impressive results in sentiment analysis tasks.

In this paper, we performed few basic exploratory data analysis to understand the distribution of the data. We later performed sentiment analysis using few models from NLTK and neural network models from keras. The study uses the Amazon review dataset for product reviews. We utilized two approaches from NLTK to perform sentiment analysis: VADER (Valence Aware Dictionary and Sentiment Reasoner) and Roberta Pretrained Models. The neural network model used in this project is a sequential model from Keras with an embedding layer, an LSTM layer, and a dense output layer. We conducted a study to compare the performance of these methods.

We will use the Amazon customer reviews dataset to train and test our model. The Amazon reviews dataset is a collection consisting of 3,744,047 reviews on a variety of products. It contains product reviews as text and has a score which is star reviews from 1 to 5 with 5 being most liked and 1 being least liked.

II. Backgrounds

With the advances in Deep Learning in parallel with higher computation capabilities using cloud computing techniques, approaches like Recurrent Neural Networks, Long Short-Term Memory and the latest Transformer based methods have been a huge success. One such approach of text-based sentiment analysis, Recurrent Neural Networks is considered the most powerful method.

III. Approach

A. Dataset

The Amazon reviews dataset [1] consists of features like 'Id', 'Product Id', 'User Id', 'Profile Name', 'Helpfulness Numerator', 'Helpfulness Denominator', 'Score', 'Time', 'Summary', 'Text'. We will be working on 'Text', 'Id', and 'Score'.The dataset is split into training, validation, and test datasets. The training dataset contains 60% of the data, the validation dataset contains 20% of the data and the test dataset contains the remaining 20% of the data. Then the test dataset is further split into validation andtesting sets with stratified random sampling to ensure the classes are balanced in all the training, validation, and test sets.

B. Pre-processing

- In Exploratory Data Analysis (EDA), for each label in the train set to get an overviewof the most frequent words. Many HTML tags like" p"," href", etc. are identified. These words are removed.
- We later plotted a bar plot to compare the count of reviews based on the star reviews they received. We observed that most of the reviews have a five-star rating. And the count decreased gradually with the ratings.
- We wrote a function to perform a few sorts of pre-processing like:
  - Lowercase the entire text
  - remove all html tags like "p", "href", etc. from dataset
  - Replace special characters and punctuation with a blank (except for full stop(.))
  - split the text into words
  - remove stop words like to, an, the, of, over, his, her…

- Punctuation, numbers are removed and converted into tokens with" " as a delimiter.
- In the same function we also collected pos_tags for all the words and lemmatized them using WordNetLemmatizer ().
- We also plotted a bar plot to compare the count of reviews and the length of the words in them.
- We observed that most of the reviews have less than 200 words. Hence, we put a threshold of 200 words and considered the reviews only whose length is less than 200.

C. Vector Embeddings

- We performed embedding of the text using pre-trained word vectors, 'glove.6B.100d.txt' which is publicly available at the Stanford website.
- The dimension of each word embedding is set to 100.
- We then created an embedding matrix for directly feeding to Embedding () layer of Keras.
- We wrote a function to average out embedding for each corresponding word in a review.

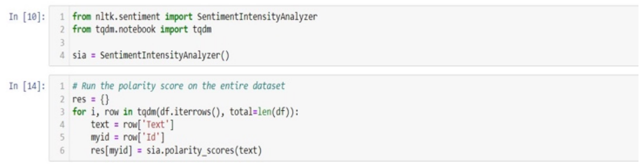
D. Loss Function

Since our problem is Binary (Positive and Negative), we will be using Binary Cross Entropy Loss is used.

$$L_{binarycrossentropy}(y, p) = \text{abs}\ (Y\_pred - Y\_actual)$$

E. Evaluation Metrics

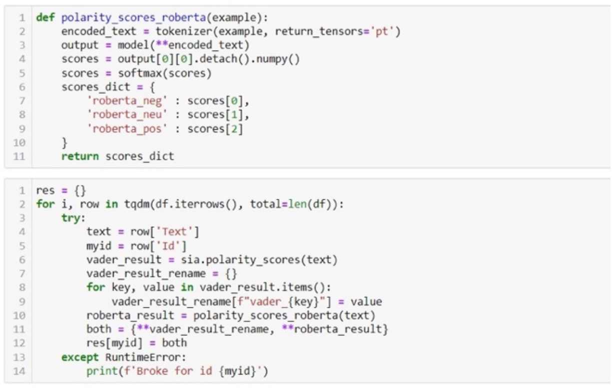
We chose to use Accuracy as a performance measure of our Train, Validation, and Test sets.



```
In [10]:
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

sia = SentimentIntensityAnalyzer()
```

```
In [14]:
# Run the polarity score on the entire dataset
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    text = row['Text']
    myid = row['Id']
    res[myid] = sia.polarity_scores(text)
```

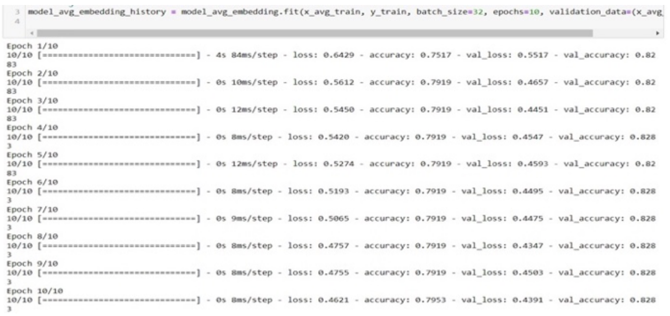
Fig.1. Evaluation metrics VADER Polarity Scores



```
def polarity_scores_roberta(example):
    encoded_text = tokenizer(example, return_tensors='pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'roberta_neg' : scores[0],
        'roberta_neu' : scores[1],
        'roberta_pos' : scores[2]
    }
    return scores_dict
```

```
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    try:
        text = row['Text']
        myid = row['Id']
        vader_result = sia.polarity_scores(text)
        vader_result_rename = {}
        for key, value in vader_result.items():
            vader_result_rename[f"vader_{key}"] = value
        roberta_result = polarity_scores_roberta(text)
        both = {**vader_result_rename, **roberta_result}
        res[myid] = both
    except RuntimeError:
        print(f'Broke for id {myid}')
```

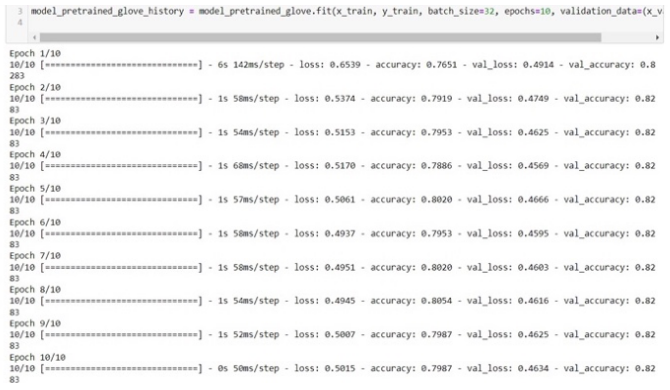
Fig.2. Evaluation metrics Roberta Polarity Scores



```
model_avg_embedding_history = model_avg_embedding.fit(x_avg_train, y_train, batch_size=32, epochs=10, validation_data=(x_avg_
```

```
Epoch 1/10
10/10 [==============================] - 4s 84ms/step - loss: 0.6429 - accuracy: 0.7517 - val_loss: 0.5517 - val_accuracy: 0.82
83
Epoch 2/10
10/10 [==============================] - 0s 10ms/step - loss: 0.5612 - accuracy: 0.7919 - val_loss: 0.4657 - val_accuracy: 0.82
83
Epoch 3/10
10/10 [==============================] - 0s 12ms/step - loss: 0.5450 - accuracy: 0.7919 - val_loss: 0.4451 - val_accuracy: 0.82
83
Epoch 4/10
10/10 [==============================] - 0s 8ms/step - loss: 0.5420 - accuracy: 0.7919 - val_loss: 0.4547 - val_accuracy: 0.828
3
Epoch 5/10
10/10 [==============================] - 0s 12ms/step - loss: 0.5274 - accuracy: 0.7919 - val_loss: 0.4593 - val_accuracy: 0.82
83
Epoch 6/10
10/10 [==============================] - 0s 8ms/step - loss: 0.5193 - accuracy: 0.7919 - val_loss: 0.4495 - val_accuracy: 0.828
3
Epoch 7/10
10/10 [==============================] - 0s 9ms/step - loss: 0.5065 - accuracy: 0.7919 - val_loss: 0.4475 - val_accuracy: 0.828
3
Epoch 8/10
10/10 [==============================] - 0s 8ms/step - loss: 0.4757 - accuracy: 0.7919 - val_loss: 0.4347 - val_accuracy: 0.828
3
Epoch 9/10
10/10 [==============================] - 0s 8ms/step - loss: 0.4755 - accuracy: 0.7919 - val_loss: 0.4503 - val_accuracy: 0.828
3
Epoch 10/10
10/10 [==============================] - 0s 8ms/step - loss: 0.4621 - accuracy: 0.7953 - val_loss: 0.4391 - val_accuracy: 0.828
3
```

Fig.3. Evaluation for average embedding



```
model_pretrained_glove_history = model_pretrained_glove.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_v
```

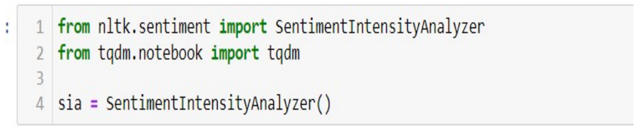
```
Epoch 1/10
10/10 [==============================] - 6s 142ms/step - loss: 0.6539 - accuracy: 0.7651 - val_loss: 0.4914 - val_accuracy: 0.8
283
Epoch 2/10
10/10 [==============================] - 1s 58ms/step - loss: 0.5374 - accuracy: 0.7919 - val_loss: 0.4749 - val_accuracy: 0.82
83
Epoch 3/10
10/10 [==============================] - 1s 54ms/step - loss: 0.5153 - accuracy: 0.7953 - val_loss: 0.4625 - val_accuracy: 0.82
83
Epoch 4/10
10/10 [==============================] - 1s 68ms/step - loss: 0.5170 - accuracy: 0.7886 - val_loss: 0.4569 - val_accuracy: 0.82
83
Epoch 5/10
10/10 [==============================] - 1s 57ms/step - loss: 0.5061 - accuracy: 0.8020 - val_loss: 0.4666 - val_accuracy: 0.82
83
Epoch 6/10
10/10 [==============================] - 1s 58ms/step - loss: 0.4937 - accuracy: 0.7953 - val_loss: 0.4595 - val_accuracy: 0.82
83
Epoch 7/10
10/10 [==============================] - 1s 58ms/step - loss: 0.4951 - accuracy: 0.8020 - val_loss: 0.4603 - val_accuracy: 0.82
83
Epoch 8/10
10/10 [==============================] - 1s 54ms/step - loss: 0.4945 - accuracy: 0.8054 - val_loss: 0.4616 - val_accuracy: 0.82
83
Epoch 9/10
10/10 [==============================] - 1s 52ms/step - loss: 0.5007 - accuracy: 0.7987 - val_loss: 0.4625 - val_accuracy: 0.82
83
Epoch 10/10
10/10 [==============================] - 0s 50ms/step - loss: 0.5015 - accuracy: 0.7987 - val_loss: 0.4634 - val_accuracy: 0.82
83
```

Fig. 4. Evaluation for average glove embedding

IV. Models

A. VADER Model from NLTK library

We used NLTK's Sentiment Intensity Analyzer to get the neg/neu/pos scores of the text. This uses a "bag of words" approach and stop words are removed. Also, each word is scored and combined to a total score.

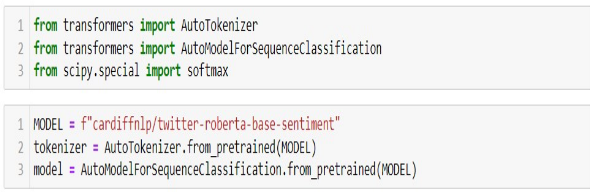


```
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

sia = SentimentIntensityAnalyzer()
```

Fig. 5. VADAR Model

B. Roberta Pretrained Model from Hugging face library

It builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.



```
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
```

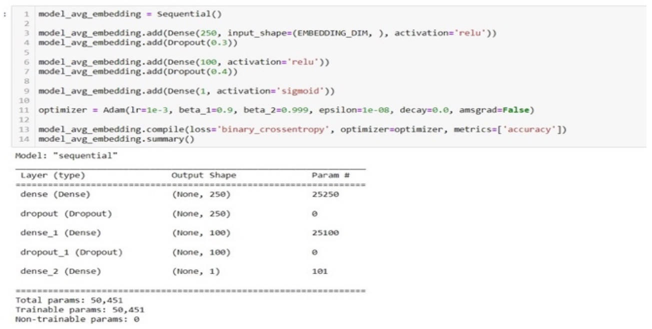
```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

Fig. 6. Roberta Model

C. Sequential model from Keras

The sequential model allows us to create layer-by-layer for most problems. It is limited. It does not allow us to create models that share layers or have multiple inputs or outputs.

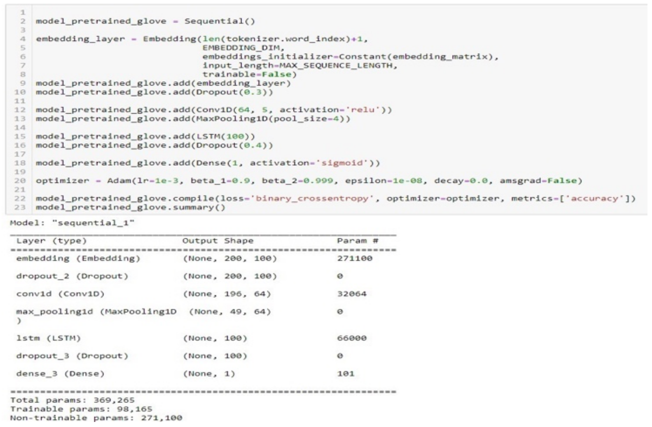
The same model is used for both average word embedded and average glove-embedded data frames.



```
model_avg_embedding = Sequential()

model_avg_embedding.add(Dense(250, input_shape=(EMBEDDING_DIM, ), activation='relu'))
model_avg_embedding.add(Dropout(0.3))

model_avg_embedding.add(Dense(100, activation='relu'))
model_avg_embedding.add(Dropout(0.4))

model_avg_embedding.add(Dense(1, activation='sigmoid'))

optimizer = Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0, amsgrad=False)

model_avg_embedding.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
model_avg_embedding.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 250)               25250

 dropout (Dropout)           (None, 250)               0

 dense_1 (Dense)             (None, 100)               25100

 dropout_1 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 50,451
Trainable params: 50,451
Non-trainable params: 0
```

Fig. 7. Sequential Model for Word Embedded



```

model_pretrained_glove = Sequential()

embedding_layer = Embedding(len(tokenizer.word_index)+1,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
model_pretrained_glove.add(embedding_layer)
model_pretrained_glove.add(Dropout(0.3))

model_pretrained_glove.add(Conv1D(64, 5, activation='relu'))
model_pretrained_glove.add(MaxPooling1D(pool_size=4))

model_pretrained_glove.add(LSTM(100))
model_pretrained_glove.add(Dropout(0.4))

model_pretrained_glove.add(Dense(1, activation='sigmoid'))

optimizer = Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0, amsgrad=False)

model_pretrained_glove.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
model_pretrained_glove.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 100)          271100

 dropout_2 (Dropout)         (None, 200, 100)          0

 conv1d (Conv1D)             (None, 196, 64)           32064

 max_pooling1d (MaxPooling1D  (None, 49, 64)           0
 )

 lstm (LSTM)                 (None, 100)               66000

 dropout_3 (Dropout)         (None, 100)               0

 dense_3 (Dense)             (None, 1)                 101

=================================================================
Total params: 369,265
Trainable params: 98,165
Non-trainable params: 271,100
```

Fig. 8. Sequential Model for glove Embedded

V. RESULTS

A. VADAR Results

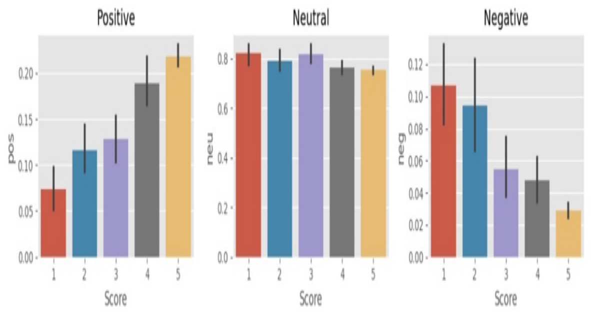


Fig. 9. VADAR Score Vs Star Rating

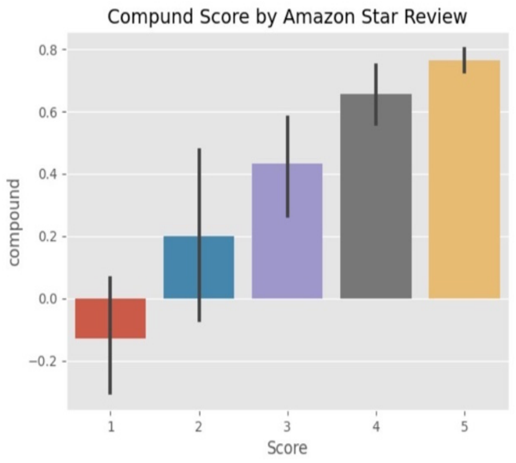


Fig. 10. VADAR Compound Score VS Star Rating

B. Roberta Results

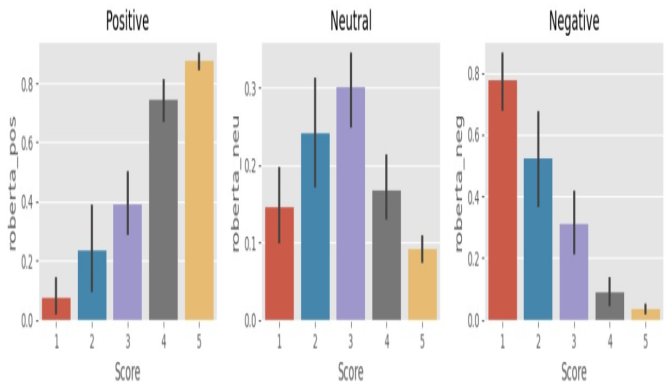


Fig. 11. Roberta Score Vs Star Rating

C. Comparing Roberta and VADER Results

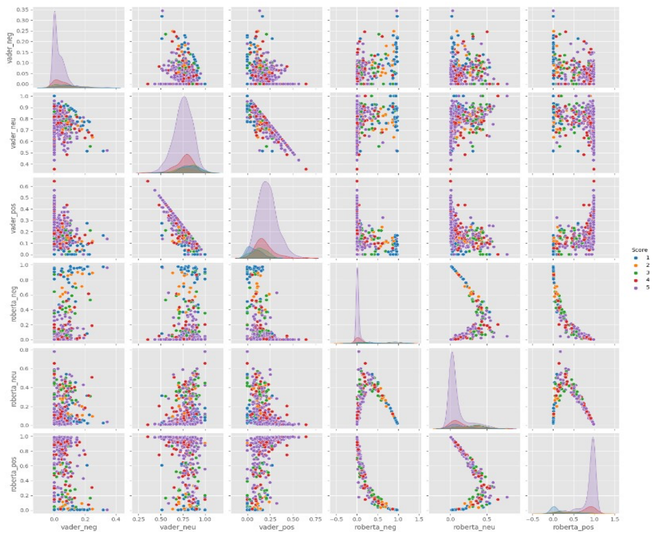


Fig. 12. Roberta Vs VADER Results

D. *Sequential results*

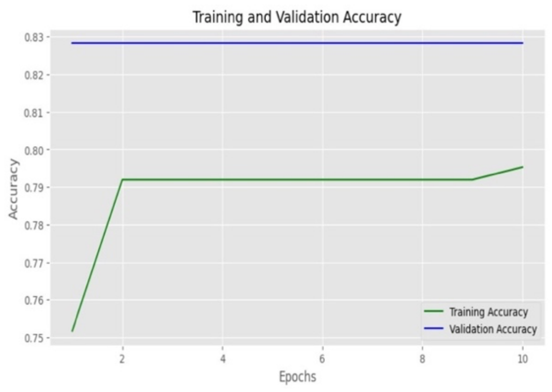
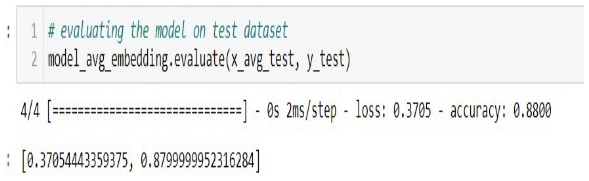


Fig. 13. Accuracy Vs Epochs for Train & Validation



```
# evaluating the model on test dataset
model_avg_embedding.evaluate(x_avg_test, y_test)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.3705 - accuracy: 0.8800

[0.37054443359375, 0.8799999952316284]
```

Fig. 14. Loss & Accuracy Score for Word Embedded

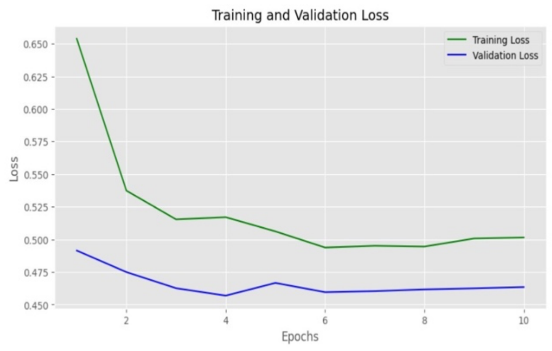
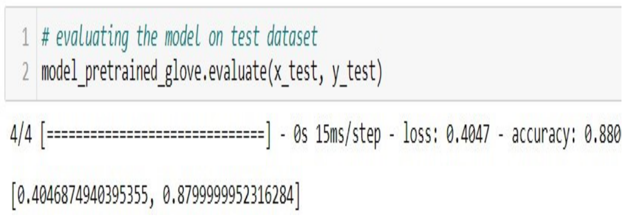


Fig. 15. Loss Vs Epochs for Glove Embedded



```
# evaluating the model on test dataset
model_pretrained_glove.evaluate(x_test, y_test)
```

```
4/4 [==============================] - 0s 15ms/step - loss: 0.4047 - accuracy: 0.880

[0.40468749403955355, 0.8799999952316284]
```

Fig. 16. Loss & Accuracy Scores for Glove Embedded

VI. CONCLUSION

Based on our comparison between VADER and Roberta Models, we can tell that Roberta has performed a lot better.

Also, between word embedded Sequential model and Glove embedded Sequential model, later resulted in better scores

The next steps could be

- To use a Recurrent neural network which works best for text-based models.
- We can extend our NLTK model comparison to transform that token comparison to transformers that tokenises automatically.

REFERENCES

[1] https://keras.io/api/models/sequential/

[2] https://www.nltk.org/api/nltk.sentiment.vader.html#module-nltk.sentiment.vader

[3] Dataset: https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews