Natural Language Processing

Assignment 4

Fatemeh Nadi 810101285

May 26, 2023

Problem 1

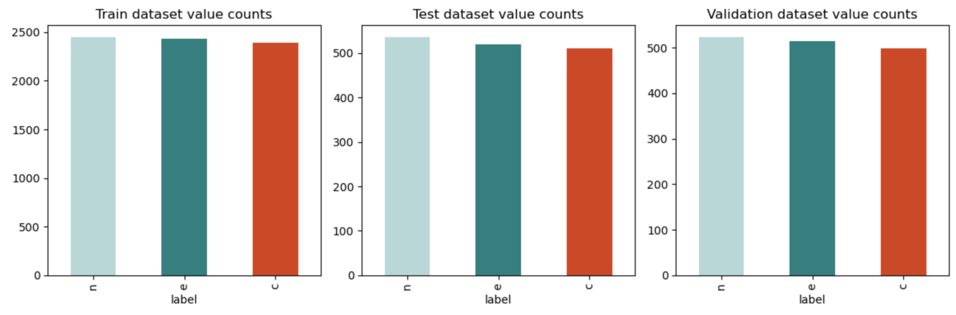
Please see this file: ”CA4_NLP_Q1.ipynb”
Code and explanation are provided

We intend to use Transformers in text applications in this question. Textual entailment is one of the applications studied in Natural Language Processing. To understand the relationship between two sentences, textual entailment can be used. There are three relationships in this assignment:

- Entailment: According to the first sentence, the second sentence is true.
- Contradiction: The two sentences contradict each other
- Neutral: The sentences are not related to each other.

Statistical analysis

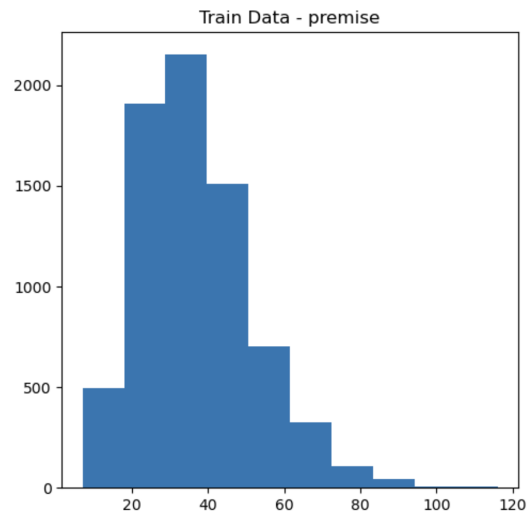
Distribution of each dataframe:

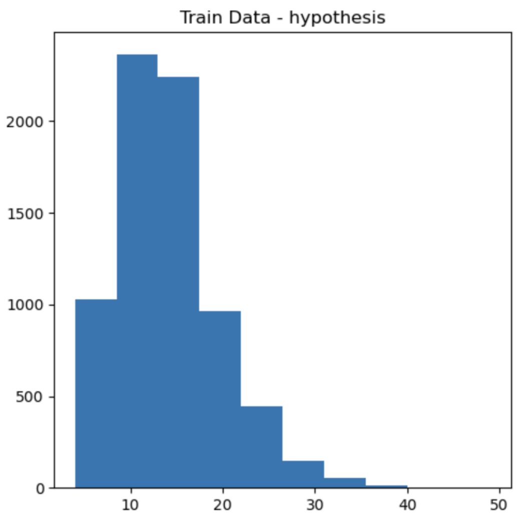


$$Train\ Data = 70.09\%$$
$$Validation\ Data = 14.83\%$$
$$Test\ Data = 15.09\%$$

Data is balanced for each classes.
Minimum and maximum token length of both ’premise’ and ’hypothesis’ for Train:

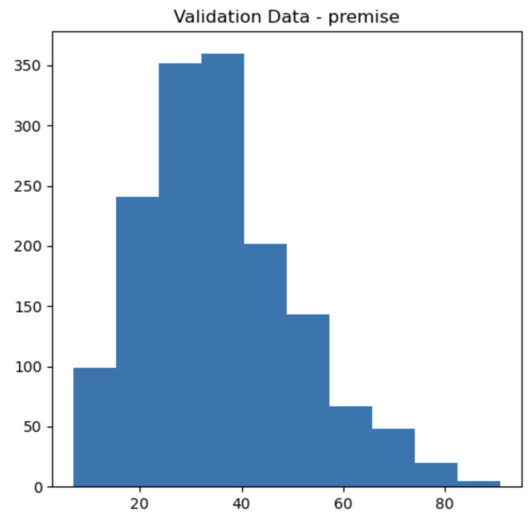
```
max length of premise for Train 116
min length of premise for Train 7
max length of hypothesis for Train 49
min length of hypothesis for Train 4
```

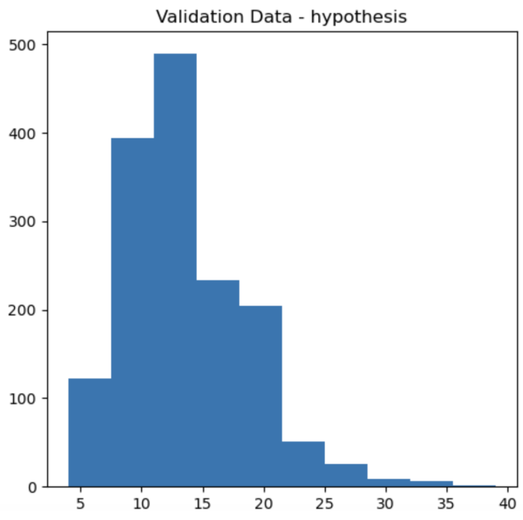




Minimum and maximum token length of both ’premise’ and ’hypothesis’ for Validation:

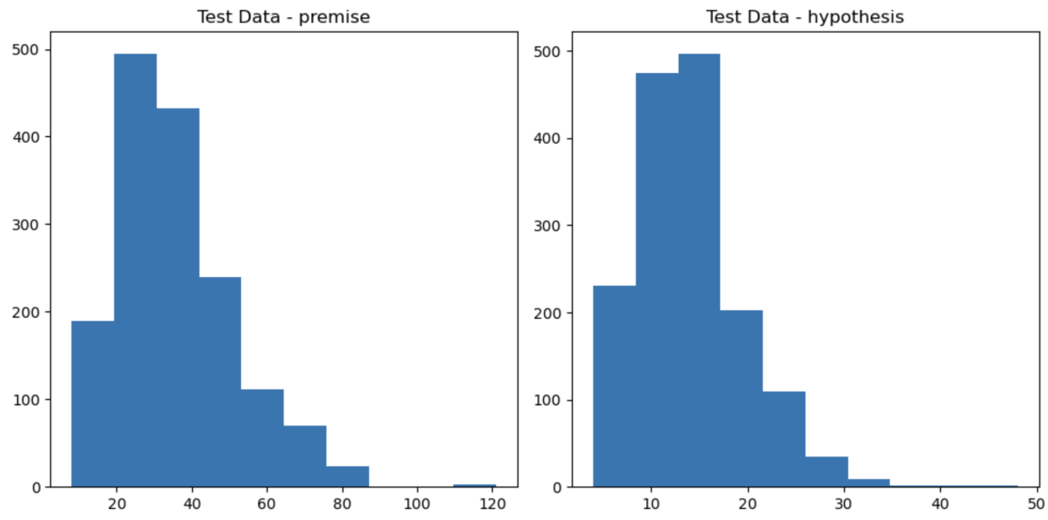
```
max length of premise for Validation 91
min length of premise for Validation 7
max length of hypothesis for Validation 39
min length of hypothesis for Validation 4
```





Minimum and maximum token length of both ’premise’ and ’hypothesis’ for Test:

```
max length of premise for Test 121
min length of premise for Test 8
max length of hypothesis for Test 48
min length of hypothesis for Test 4
```



Reduce premise sentences with more than 69 to minimized sentences.

Preprocessing

Compared to other NLP classification tasks, preprocessing for entailment tasks is slightly different.
In preprocessing, it is very important that the context and point of view of the sentences are not lost. lammatization and removal of stop words are therefore not necessary.
Normalize and replace special characters in premises and hypothesis.

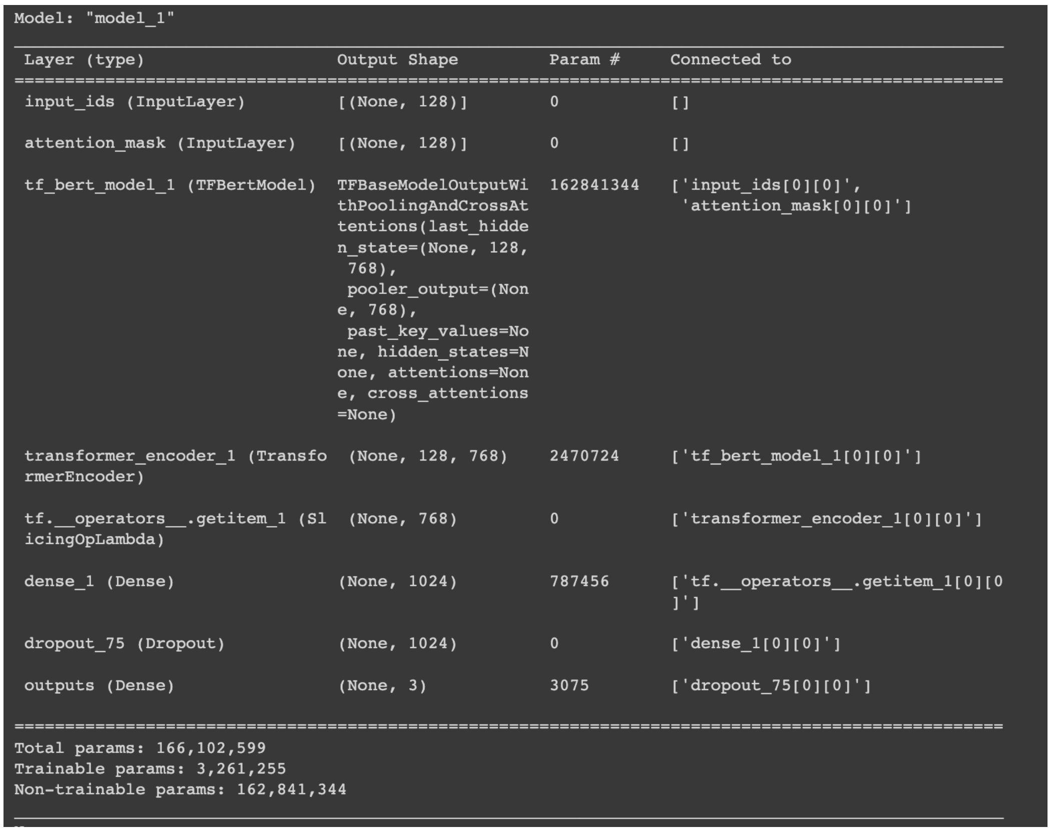
Task 1

Text classification is performed using the BERT model in this code by defining a class called Classifier. Here's how the code works:

- As part of the initialization process, the INIT method initializes the Classifier object with the data set and model checkpoint provided. It also initializes the column names for training and testing data, the preprocessing flag, the batch size, and the sequencing number.

- By using a dictionary, encode_classes maps the class labels in the DataFrame to encoded values.
- Prepare_data creates DataFrames from training, validation, and test datasets. If the preprocess flag is true, it encodes the class labels, and optionally cleans the data.
- Using the specified model checkpoint, the set_model_and_tokenizer method initializes the BERT tokenizer and model.
- A tokenized ID, attention mask, and one-hot encoded label is returned by the get_tokenized_ids method.
- TensorFlow datasets are created by mapping tokenized inputs, masks, and labels to a dictionary format.
- Using tokenized inputs, masks, and labels, the get_dataset method creates a TensorFlow dataset.
- By using BERT as the base model, the create_model method defines and compiles the classification model architecture. The transformer encoder layer is followed by dense layers, and the output layer follows. Except for the top layers, the model's layers are frozen. An optimizer, loss function, and evaluation metric are included in the model.
- As a result of training the model on the training dataset, validating it on the validation dataset, and evaluating it on the test dataset, the train_and_evaluate_model method returns the trained model, training history, evaluation score, and predictions.
- Clean_text cleans the text by tokenizing it, converting it to lowercase, removing stopwords, and lemmatizing it.
- In the clean_data method, the clean_text method is applied to the premise and hypothesis columns of the DataFrame to clean the text data.
- Using Matplotlib, the plot_results method plots training and validation accuracy and loss curves.
- Using Seaborn's heatmap, the print_confusion_matrix method prints and displays a confusion matrix.
- Using the ParsBERT model , an instance of the Classifier class is created. Using the provided dataset, the model is trained and evaluated.

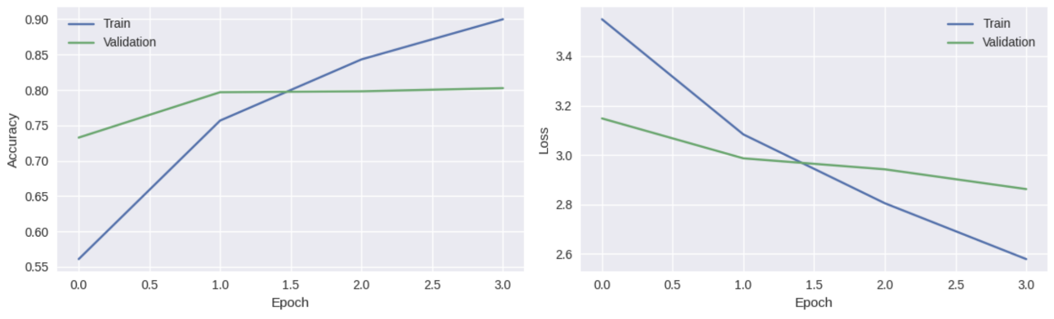
The model is as follows:

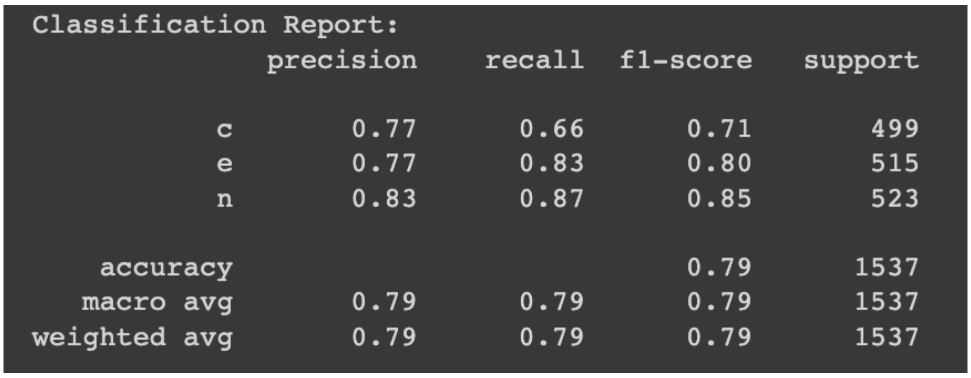


```
Model: "model_1"
______________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
======================================================================================================
 input_ids (InputLayer)         [(None, 128)]        0           []

 attention_mask (InputLayer)    [(None, 128)]        0           []

 tf_bert_model_1 (TFBertModel)  TFBaseModelOutputWi  162841344   ['input_ids[0][0]',
                                thPoolingAndCrossAt               'attention_mask[0][0]']
                                tentions(last_hidde
                                n_state=(None, 128,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 transformer_encoder_1 (Transfo  (None, 128, 768)    2470724     ['tf_bert_model_1[0][0]']
 rmerEncoder)

 tf.__operators__.getitem_1 (Sl  (None, 768)         0           ['transformer_encoder_1[0][0]']
 icingOpLambda)

 dense_1 (Dense)                (None, 1024)         787456      ['tf.__operators__.getitem_1[0][0
                                                                 ]']

 dropout_75 (Dropout)           (None, 1024)         0           ['dense_1[0][0]']

 outputs (Dense)                (None, 3)            3075        ['dropout_75[0][0]']

======================================================================================================
Total params: 166,102,599
Trainable params: 3,261,255
Non-trainable params: 162,841,344
______________________________________________________________________________________________________
```

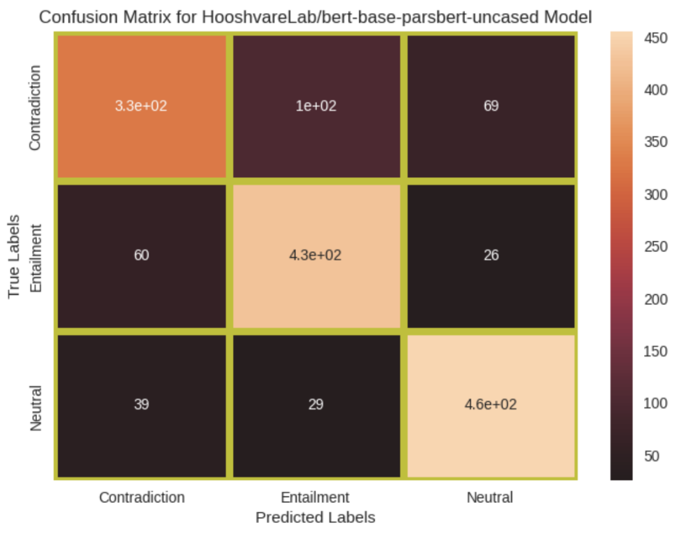
Results

Results for HooshvareLab/bert-base-parsbert-uncased Model





```
Classification Report:
              precision    recall  f1-score   support

           c       0.77      0.66      0.71       499
           e       0.77      0.83      0.80       515
           n       0.83      0.87      0.85       523

    accuracy                           0.79      1537
   macro avg       0.79      0.79      0.79      1537
weighted avg       0.79      0.79      0.79      1537
```

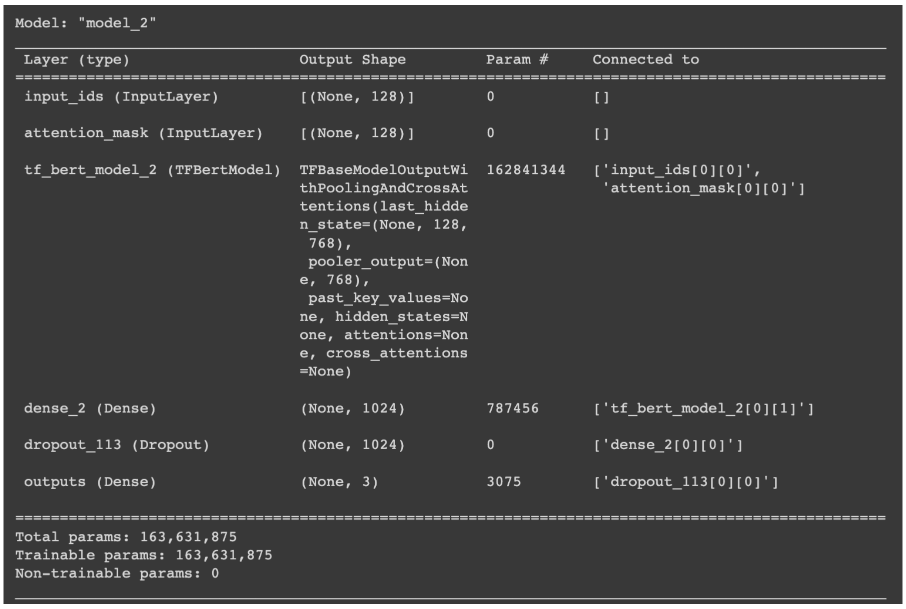


Train accuracy : 89%
Validation accuracy of 80%

It was observed in the test report and confusion matrix that the contradiction and neutral classes can be predicted more accurately than the entailmen class.
Unfreezing the Bert layer and fine-tuning it in accordance with the rest of the model is also expected to result in a higher degree of accuracy.

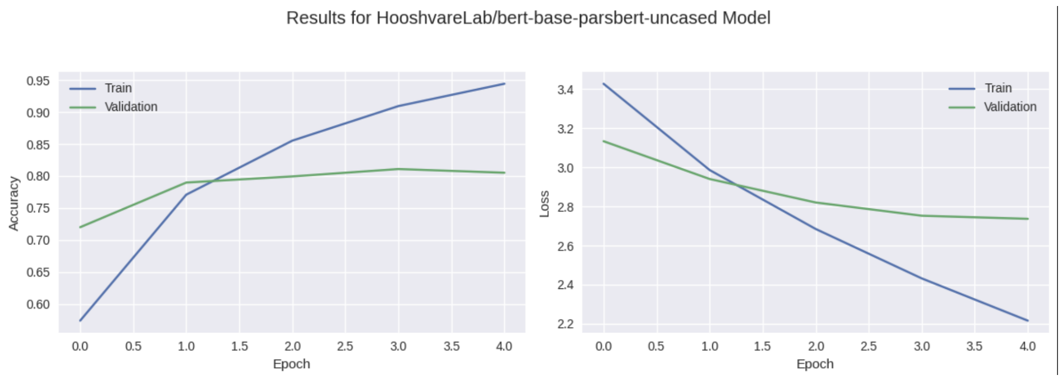
Task 2

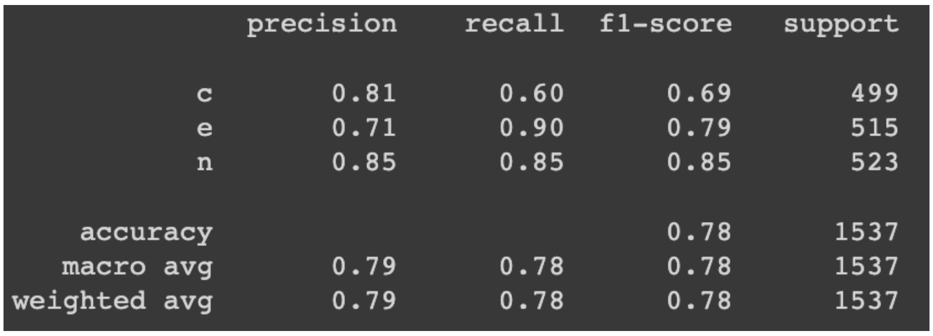
A transformer encoder is not added to this model, and the Bert model can be trained, so it can learn a specific task.



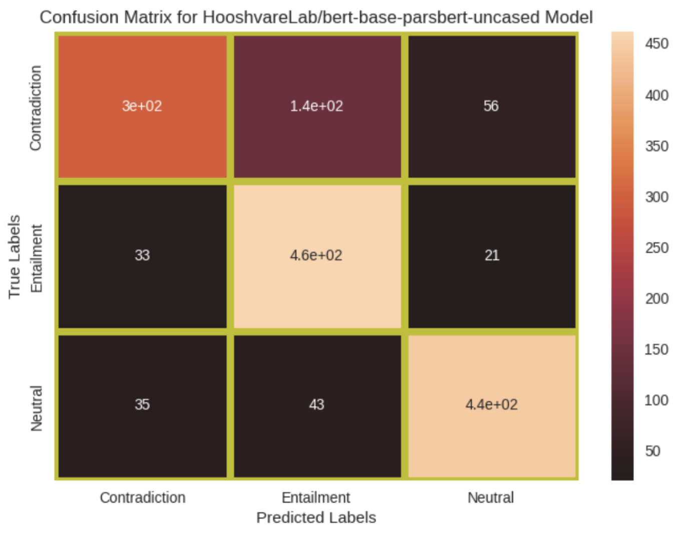
```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_ids (InputLayer)         [(None, 128)]        0           []

 attention_mask (InputLayer)    [(None, 128)]        0           []

 tf_bert_model_2 (TFBertModel)  TFBaseModelOutputWi  162841344   ['input_ids[0][0]',
                                thPoolingAndCrossAt               'attention_mask[0][0]']
                                tentions(last_hidde
                                n_state=(None, 128,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dense_2 (Dense)                (None, 1024)         787456      ['tf_bert_model_2[0][1]']

 dropout_113 (Dropout)          (None, 1024)         0           ['dense_2[0][0]']

 outputs (Dense)                (None, 3)            3075        ['dropout_113[0][0]']

==================================================================================================
Total params: 163,631,875
Trainable params: 163,631,875
Non-trainable params: 0
__________________________________________________________________________________________________
```

Results





```
              precision    recall  f1-score   support

           c       0.81      0.60      0.69       499
           e       0.71      0.90      0.79       515
           n       0.85      0.85      0.85       523

    accuracy                           0.78      1537
   macro avg       0.79      0.78      0.78      1537
weighted avg       0.79      0.78      0.78      1537
```



Train accuracy : 94%
Validation accuracy of 80%

Comparison

In Task 1 :
Accuracy on test data: 0.789
AUC - Contradiction: 0.2
AUC - Entailment: 0.411
AUC - Neutral: 0.879

In Task 2 :
Accuracy on test data: 0.783
AUC - Contradiction: 0.196
AUC - Entailment: 0.415
AUC - Neutral: 0.88

Because we train model 2 in more than one epoch, I think overfitting accuracy or model need more than epochs. Both results are close to each other at all.

Task 3

Natural Language Inference (NLI) tasks require the preparation and loading of data, which is performed by NLIDataset. This class accomplishes the following:

- Using the init method, the dataset is initialized with three dataframes (train_df, val_df, test_df) containing the training, validation, and test data, and model_name which specifies the name of the pre-trained model.

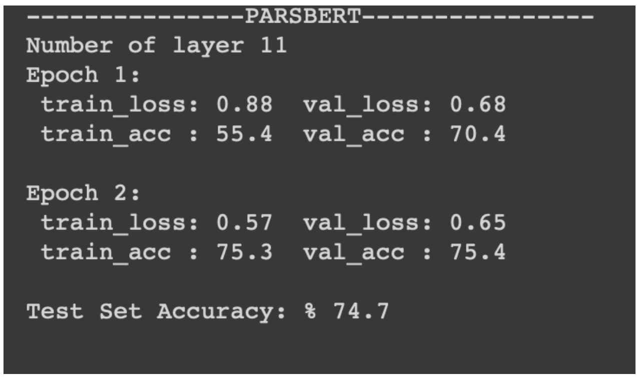
- In the prepare_data method, the load_data method is called for each dataframe (train, val, test).
- Using the specified tokenizer (AutoTokenizer from the Hugging Face library), the load_data method loads data from a dataframe. By encoding the text into token IDs, it creates attention mask IDs. The label_dict is also used to assign numerical labels to each data instance.
- Get_data_loaders returns data loaders for training, validation, and test data. To create data loaders with specified batch size and shuffle settings, it uses the PyTorch DataLoader class.

DeleteEncodingLayers is a utility function that takes in a full BERT model and a number of layers to keep. BERT model is modified by removing the specified number of encoding layers.

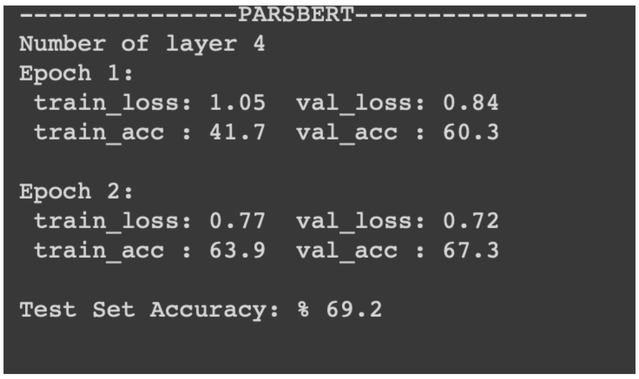
For fine-tuning or experimentation, you can use this function to remove certain layers from a BERT model.

Each iteration removes one layer and reports accuracy in three epochs.
Accuracy decreased when each layer was removed.

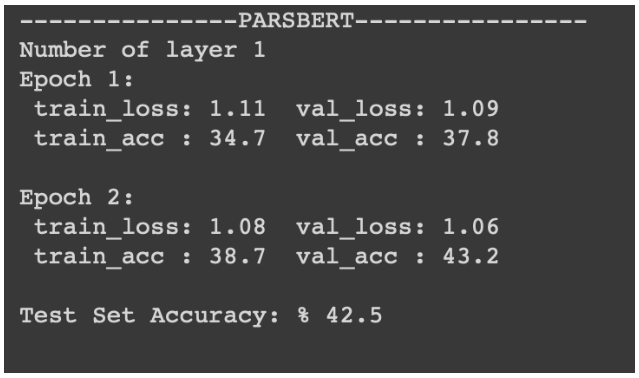
Results are listed below:



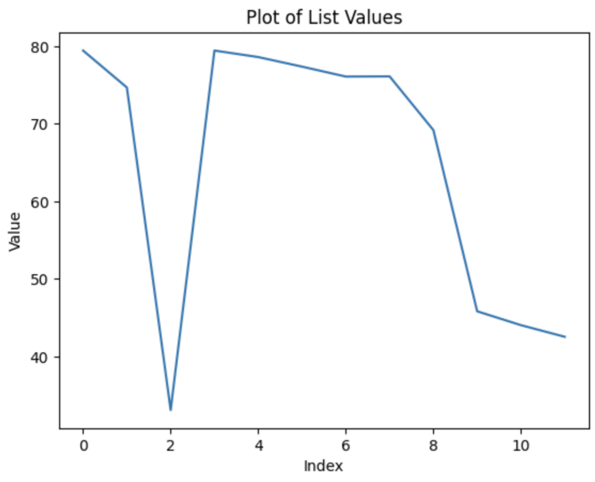
```
---------------PARSBERT----------------
Number of layer 11
Epoch 1:
 train_loss: 0.88  val_loss: 0.68
 train_acc : 55.4  val_acc : 70.4

Epoch 2:
 train_loss: 0.57  val_loss: 0.65
 train_acc : 75.3  val_acc : 75.4

Test Set Accuracy: % 74.7
```



```
---------------PARSBERT----------------
Number of layer 4
Epoch 1:
 train_loss: 1.05  val_loss: 0.84
 train_acc : 41.7  val_acc : 60.3

Epoch 2:
 train_loss: 0.77  val_loss: 0.72
 train_acc : 63.9  val_acc : 67.3

Test Set Accuracy: % 69.2
```



```
---------------PARSBERT----------------
Number of layer 1
Epoch 1:
 train_loss: 1.11  val_loss: 1.09
 train_acc : 34.7  val_acc : 37.8

Epoch 2:
 train_loss: 1.08  val_loss: 1.06
 train_acc : 38.7  val_acc : 43.2

Test Set Accuracy: % 42.5
```



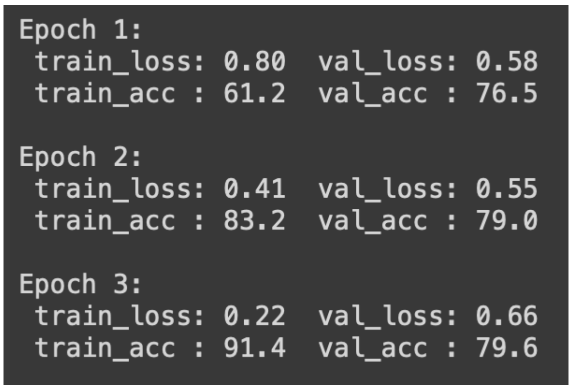
With ten layers, the accuracy is 33%, but there may be some errors during training. Except that this accuracy starts at 79% and ends at 44%.

Task 4

Parsbert has 12 layers, with each layer having 12 layers, so we have 144 layers.
A dictionary with a specific pattern of values is generated by delete_heads(p).
This function takes a parameter p that represents the desired proportion of the total number of values compared to the maximum possible number (144 in this case).
It enters a loop that continues until N is greater than 0.

- It selects a random key from my_dict.
- When the value list for the selected key is less than 12 (to limit the number of values per key to 12), it generates a random value between 0 and 11.
- It adds the new value to the value list corresponding to the selected key and subtracts 1 from N if the new value is not already present.

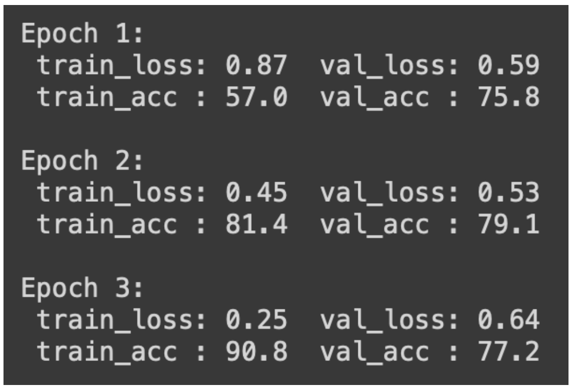
50%



```
Epoch 1:
 train_loss: 0.80  val_loss: 0.58
 train_acc : 61.2  val_acc : 76.5

Epoch 2:
 train_loss: 0.41  val_loss: 0.55
 train_acc : 83.2  val_acc : 79.0

Epoch 3:
 train_loss: 0.22  val_loss: 0.66
 train_acc : 91.4  val_acc : 79.6
```

Test Set Accuracy: % 73.3

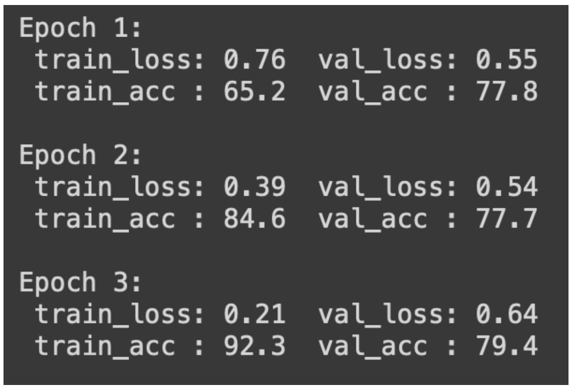
67%



```
Epoch 1:
 train_loss: 0.87  val_loss: 0.59
 train_acc : 57.0  val_acc : 75.8

Epoch 2:
 train_loss: 0.45  val_loss: 0.53
 train_acc : 81.4  val_acc : 79.1

Epoch 3:
 train_loss: 0.25  val_loss: 0.64
 train_acc : 90.8  val_acc : 77.2
```

Test Set Accuracy: % 80.1

83%



```
Epoch 1:
 train_loss: 0.76  val_loss: 0.55
 train_acc : 65.2  val_acc : 77.8

Epoch 2:
 train_loss: 0.39  val_loss: 0.54
 train_acc : 84.6  val_acc : 77.7

Epoch 3:
 train_loss: 0.21  val_loss: 0.64
 train_acc : 92.3  val_acc : 79.4
```

Test Set Accuracy: % 80.9

There may be a more important head in part one randomly deleted when I delete 83% heads.

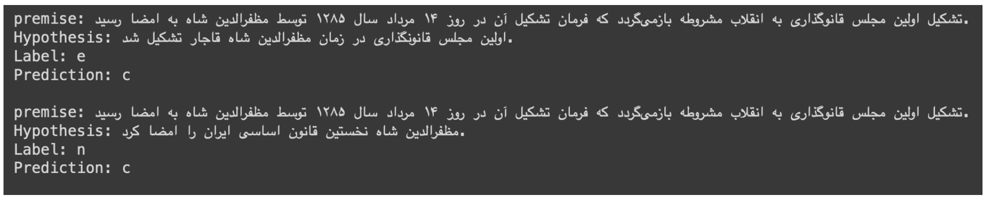
Task 5

To determine what is biased in this model, use the previous model:
Two significant biases can affect entailment classification. Hypothesis-only bias occurs when the model relies solely on the information provided in the hypothesis to make class predictions. Various factors can contribute to this bias, such as negation words like ”not,” ”never,” and ”no,” which are often associated with contradictions. In addition, the model may learn that entailment hypotheses tend to be shorter than contradictions or neutral statements based on the dataset.

Consequently, the model may classify examples solely based on the content of the hypothesis, without considering thoroughly the information presented in the premise. Due to this hypothesis-centric approach, the model overlooks contextual cues and logical relationships provided by the premises, resulting in biased predictions.

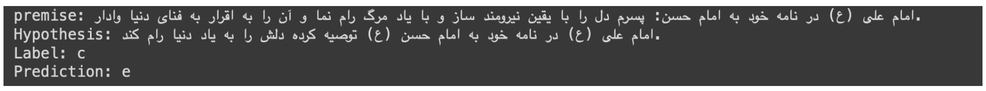
By relying solely on the information provided in the hypotheses, the model correctly identifies contradictions. Nevertheless, it does not take into account the context and additional details that can be derived from the premises.

To overcome the hypothesis-only bias, it is necessary to develop more robust models that effectively integrate both premise and hypothesis information, leveraging their synergistic relationships. The use of attention mechanisms, contextual embeddings, or fine-tuning on larger and more diverse datasets can mitigate this bias and improve the overall performance and fairness of entailment classification systems.

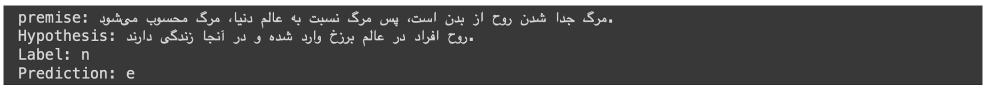


```
premise: تشکیل اولین مجلس قانونگذاری به انقلاب مشروطه بازمی‌گردد که فرمان تشکیل آن در روز ۱۴ مرداد سال ۱۲۸۵ توسط مظفرالدین شاه به امضا رسید.
Hypothesis: اولین مجلس قانونگذاری در زمان مظفرالدین شاه قاجار تشکیل شد.
Label: e
Prediction: c

premise: تشکیل اولین مجلس قانونگذاری به انقلاب مشروطه بازمی‌گردد که فرمان تشکیل آن در روز ۱۴ مرداد سال ۱۲۸۵ توسط مظفرالدین شاه به امضا رسید.
Hypothesis: مظفرالدین شاه نخستین قانون اساسی ایران را امضا کرد.
Label: n
Prediction: c
```

The second bias that can occur in entailment tasks is known as the overlap bias. This bias arises when the premise and hypothesis share a significant number of words, leading the model to classify them as entailment. This can be problematic because the model may rely heavily on surface-level word matching rather than understanding the underlying meaning of the statements. the premise and hypothesis have a high degree of lexical overlap, resulting in the model correctly predicting entailment. However, it's important to note that the model's decision is primarily based on surface-level similarity rather than capturing the deeper semantic relationship between the statements. This overlap bias highlights the challenges in effectively modeling entailment and the need to consider alternative approaches to mitigate this bias.



```
premise: امام علی (ع) در نامه خود به امام حسن: پسرم دل را با یقین نیرومند ساز و با یاد مرگ رام نما و آن را به اقرار به فنای دنیا وادار.
Hypothesis: امام علی (ع) در نامه خود به امام حسن (ع) توصیه کرده دلش را به یاد دنیا رام کند.
Label: c
Prediction: e
```



```
premise: مرگ جدا شدن روح از بدن است، پس مرگ نسبت به عالم دنیا، مرگ محسوب می‌شود.
Hypothesis: روح افراد در عالم برزخ وارد شده و در آنجا زندگی دارند.
Label: n
Prediction: e
```

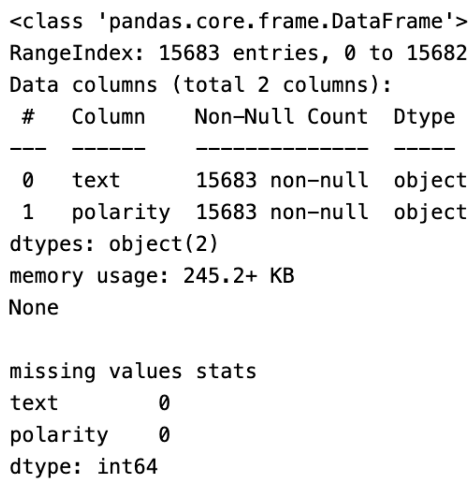
Problem 2

Please see this file: ”CA4_NLP_Q2.ipynb”
Code and explanation are provided

Part 1

Preprocessing

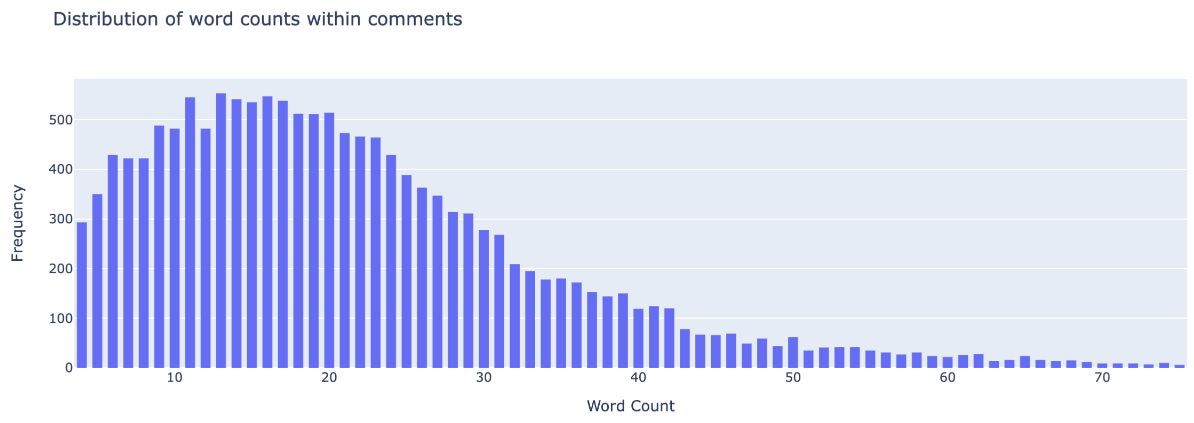
For simplicity, I fix this problem by removing rows with the rate value of None. Furthermore, the dataset contains duplicated rows and missing values in the comment section.
data information

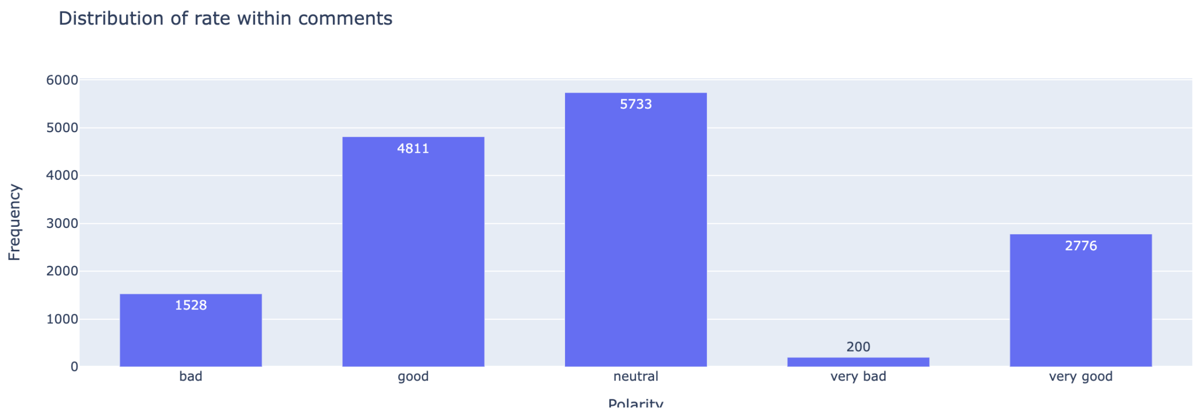


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15683 entries, 0 to 15682
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   text      15683 non-null  object
 1   polarity  15683 non-null  object
dtypes: object(2)
memory usage: 245.2+ KB
None

missing values stats
text        0
polarity    0
dtype: int64
```

Next we Normalized texts, the comments have different lengths based on words! Detecting the most normal range could help us find the maximum length of the sequences for the preprocessing step. On the other hand, we suppose that the minimum word combination for having a meaningful phrase for our learning process is 3.

So we reduce tokens, texts with word length of greater than 3 and less than 75 includes 95.95% of the whole!.





Cleaning is the final step in this section. Our cleaned method includes these steps:

- fixing unicodes
- removing specials like a phone number, email, url, new lines, ...
- cleaning HTMLs
- normalizing
- removing emojis

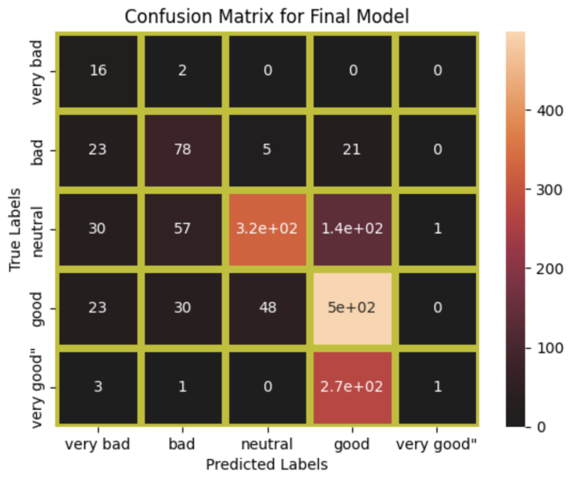
Finally map labels to integers and use bert tokenizer.
In create_input function we convert a list of input strings into tokenized and encoded inputs that can be fed into a BERT-based model with a specified maximum sequence length.

The model is summarized below:

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_word_ids (InputLayer)    [(None, 128)]        0           []

 input_mask (InputLayer)        [(None, 128)]        0           []

 segment_ids (InputLayer)       [(None, 128)]        0           []

 keras_layer (KerasLayer)       [(None, 768),        470926849   ['input_word_ids[0][0]',
                                 (None, 128, 768)]                'input_mask[0][0]',
                                                                  'segment_ids[0][0]']

 dropout (Dropout)              (None, 768)          0           ['keras_layer[0][0]']

 dense (Dense)                  (None, 5)            3845        ['dropout[0][0]']

==================================================================================================
Total params: 470,930,694
Trainable params: 470,930,693
Non-trainable params: 1
__________________________________________________________________________________________________
```

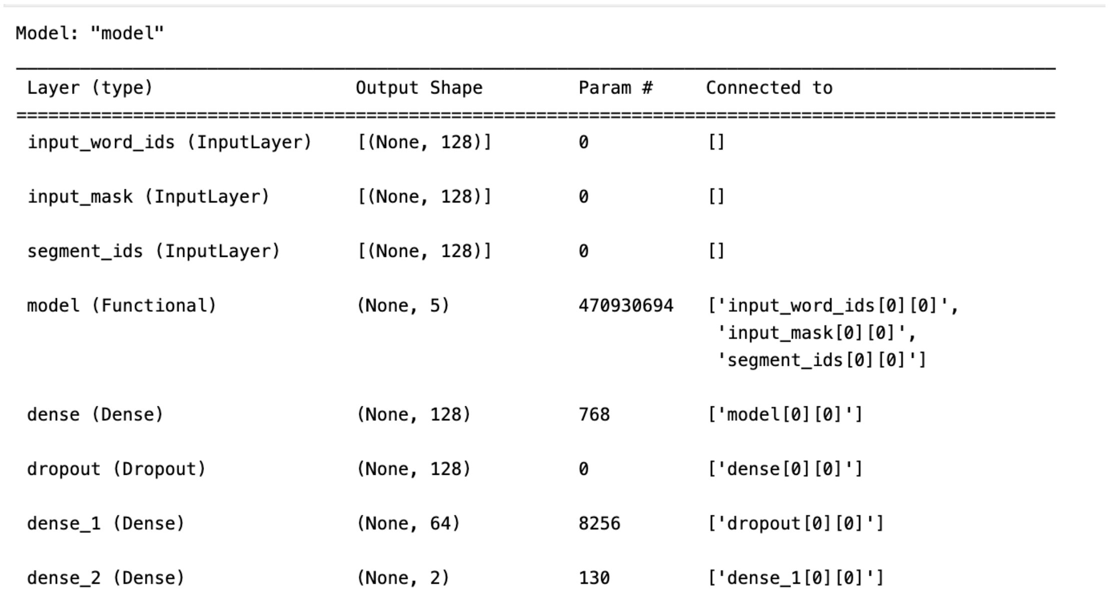
Train model in 5 epochs and results:

```
              precision    recall  f1-score   support

           0       0.17      0.89      0.28        18
           1       0.46      0.61      0.53       127
           2       0.86      0.59      0.70       550
           3       0.54      0.83      0.65       599
           4       0.50      0.00      0.01       275

    accuracy                           0.59      1569
   macro avg       0.51      0.59      0.43      1569
weighted avg       0.63      0.59      0.54      1569
```



Part 2

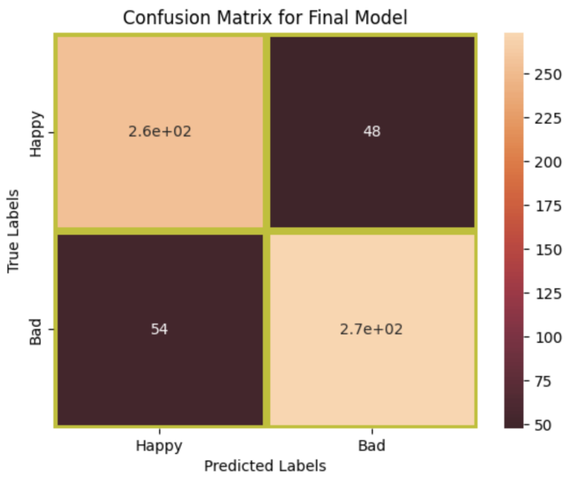
Use snapfood dataset to test preivous model.
map 5 classes of sentiper to 3 level with a dense layer here is the model:



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_word_ids (InputLayer)    [(None, 128)]        0           []

 input_mask (InputLayer)        [(None, 128)]        0           []

 segment_ids (InputLayer)       [(None, 128)]        0           []

 model (Functional)             (None, 5)            470930694   ['input_word_ids[0][0]',
                                                                  'input_mask[0][0]',
                                                                  'segment_ids[0][0]']

 dense (Dense)                  (None, 128)          768         ['model[0][0]']

 dropout (Dropout)              (None, 128)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 64)           8256        ['dropout[0][0]']

 dense_2 (Dense)                (None, 2)            130         ['dense_1[0][0]']
```

results

```
40/40 [==============================] - 6s 133ms/step
              precision    recall  f1-score   support

           0       0.83      0.84      0.83       303
           1       0.85      0.83      0.84       327

    accuracy                           0.84       630
   macro avg       0.84      0.84      0.84       630
weighted avg       0.84      0.84      0.84       630
```



The model is strong enough to make accurate predictions for the snap task despite a relatively low accuracy on previous data.
By placing a linear layer and testing the data snap, it is reasonable to reach such results if the fine-tuned model is too complex or has the most data.