Sentiment Analysis for yelp Reviews

Real People. Real Reviews.™

Group 2
Anuradha Tidke
Nitin Godi
Rakshith Eleti

Problem Statement

- The goal of this project is to estimate the Yelp ratings polarity (0 or 1) of a local business review based on the review text.
- This was done using NLP models with an emphasis on sentiment analysis.
- Yelp is a review-based website where people can discover and exchange information about local businesses.
- When consumers conduct a rapid search for businesses, they are more inclined to assess quality only on the star rating without reading the review language.
- As a result, our group is looking for probable connections between review wording and the corresponding rating.

Understanding the Dataset

- We have taken our dataset from Kaggle.
- The Yelp reviews polarity dataset is constructed by considering stars 1 and 2 negative (label: 0), and 3 and 4 positive (label: 1).
- Number of training sample points: 559,922
- Number of testing sample points: 37,997
- Train and test sample points are distributed equally for both labels.
- Feature column: Review text
- Target column: Label {0, 1}

Project Description

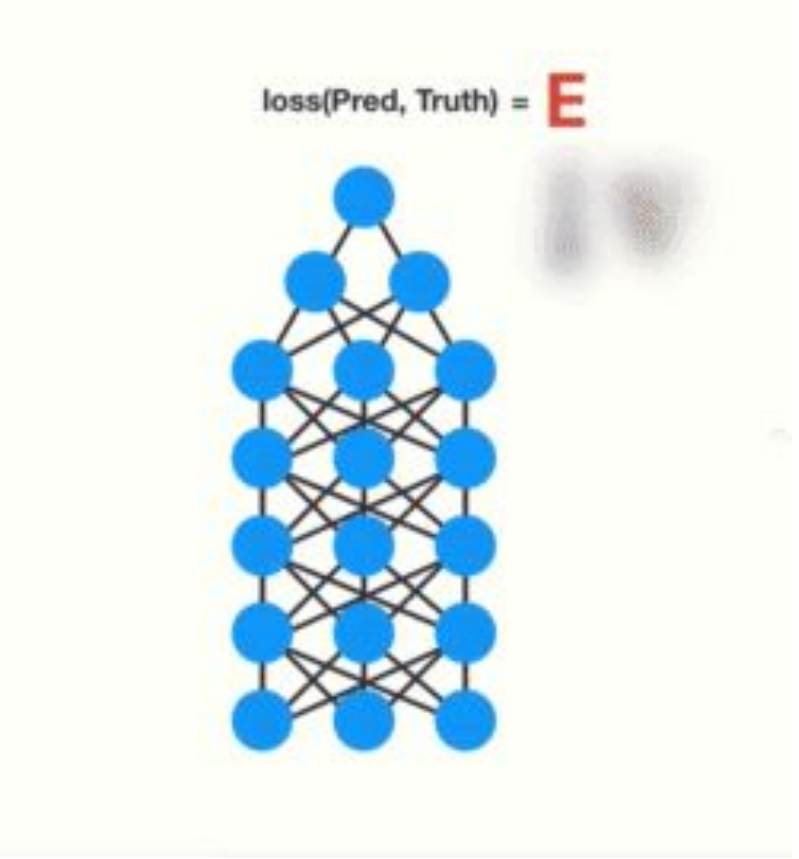
- In this project we will be exploring NLP methods like RNN model (LSTM), pre-trained models (DistillBert and ELECTRA) and Ensemble of DistilBERT and ELECTRA.
- Project pipeline:
  - Data cleaning
  - Padding and packing
  - Vectorizing the word
  - Training the model
  - Calculating model accuracy on the test dataset

Data Preprocessing

- The dataset was downloaded using Kaggle API.
- Data cleaning steps:
  - Tokenize the data
  - Remove stopwords (nltk package)
  - Remove punctuation marks
  - Lemmatize the tokens (WordNet lemmatizer from nltk)
  - Dropped zero-length sequences (0.01 percentage in train and 0.007 percentage in test datasets)
  - Split train dataset into train and validation with 90% split.
    - Resulting number of datapoints:
      - train_data length = 503,924
      - val_data length = 55,993

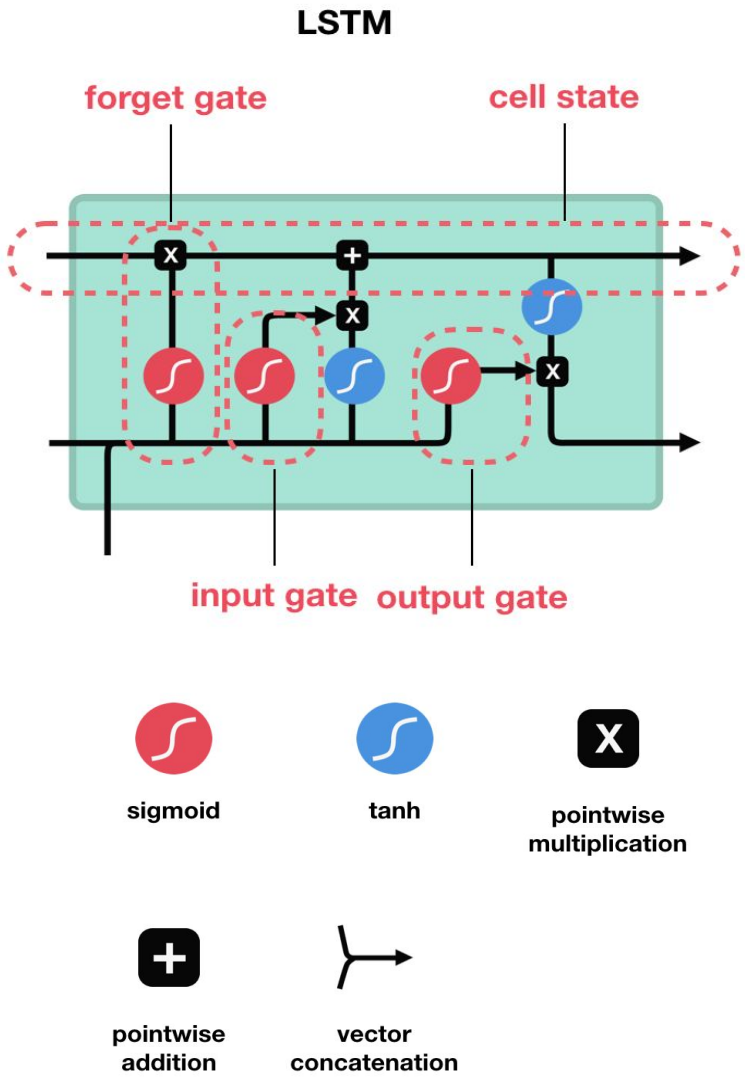
Architecture of RNN

- Class of neural networks that allow previous outputs to be used as inputs while having hidden states.
- Issue of vanishing gradient.
- Layers that get a small gradient update stop learning. Those are usually the earlier layers.
- These layers don’t learn.
- So RNN’s can forget what it seen in longer sequences, thus having a short-term memory.



LSTM to the rescue

- Internal mechanisms called gates that can regulate the flow of information.
- The Forget gate decides what is relevant to keep from prior steps.
- The input gate decides what information is relevant to add from the current step.
- The output gate determines what the next hidden state should be.



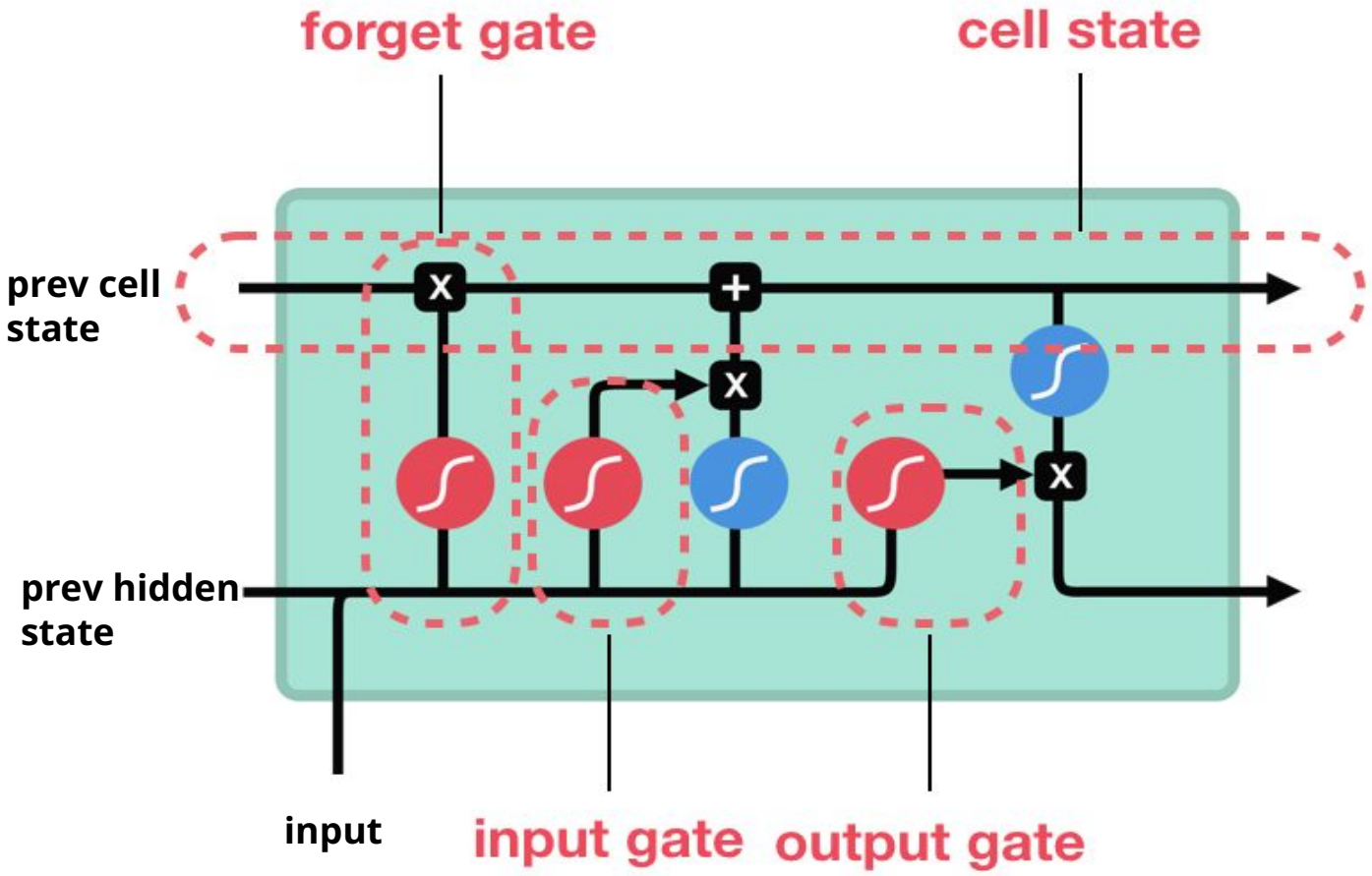
LSTM Base Code

```
def LSTMCELL(prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)
    Ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(Ct)
    return ht, Ct



ct = [0, 0, 0]
ht = [0, 0, 0]

for input in inputs:
    ct, ht = LSTMCELL(ct, ht, input)
```



Code Structure

Define LSTM class:

- Initialize the embedding and all the model layers
- Create an array of lengths of all the sequences
- Add padding to all the sequences
- Pack the padded sequences so that the zeros will be ignored

Data Prep	Train Prep

• Extract vocab dictionary and maximum sequence length. This value is used to do the padding.
• Use the glove embeddings, to calculate the embedding dictionary for our vocab
• Sentence to token ID sequence conversion
• Create a lookup table with the embedding vectors of all the tokens in the vocab
	• We chose adam optimizer and cross entropy loss
	• For each batch,
	  ○ Initialize all the optimizer gradients to zero
	  ○ Calculate the predictions and the loss.
	  ○ Loss is back propagated to calculate the gradients of previous layers
	  ○ The optimizer uses the gradients to update the parameters

LSTM version1:
Full train dataset

- **train_data length: 503,924**
- val_data length: 55,993
- test_data length: 37,997
- Parameters:
  - Learning rate: 1e-3
  - Batch size: 16
  - Epoch size: 10
- **Results:**
  - Best val_acc: 92.22 %
  - Time req for all epochs = 2 hrs 45 min
  - val_acc decreased after epoch 1
    - overfitting
  - **test_acc using the above best model: 93.63 %**

```
Starting training loop...
Epoch 0: 15748it [16:30, 15.90it/s, Training Loss: 0.24869]
Epoch 0 | Train Loss 0.24869, Train Acc 93.18, Val Acc 92.02
The model has been saved!
Epoch 1: 15748it [16:27, 15.95it/s, Training Loss: 0.18541]
Epoch 1 | Train Loss 0.18541, Train Acc 94.56, Val Acc 92.22
The model has been saved!
Epoch 2: 15748it [16:27, 15.95it/s, Training Loss: 0.15792]
Epoch 2 | Train Loss 0.15792, Train Acc 95.30, Val Acc 92.02
Epoch 3: 15748it [16:26, 15.96it/s, Training Loss: 0.13661]
Epoch 3 | Train Loss 0.13661, Train Acc 95.28, Val Acc 91.36
Epoch 4: 15748it [16:28, 15.94it/s, Training Loss: 0.11903]
Epoch 4 | Train Loss 0.11903, Train Acc 96.37, Val Acc 91.93
Epoch 5: 15748it [16:26, 15.96it/s, Training Loss: 0.10402]
Epoch 5 | Train Loss 0.10402, Train Acc 97.04, Val Acc 91.87
Epoch 6: 15748it [16:32, 15.87it/s, Training Loss: 0.09042]
Epoch 6 | Train Loss 0.09042, Train Acc 97.10, Val Acc 91.57
Epoch 7: 15748it [16:31, 15.88it/s, Training Loss: 0.07957]
Epoch 7 | Train Loss 0.07957, Train Acc 97.33, Val Acc 91.13
Epoch 8: 15748it [16:30, 15.90it/s, Training Loss: 0.06997]
Epoch 8 | Train Loss 0.06997, Train Acc 97.27, Val Acc 90.81
Epoch 9: 15748it [16:27, 15.95it/s, Training Loss: 0.06335]
Epoch 9 | Train Loss 0.06335, Train Acc 97.82, Val Acc 90.91
The accuracy on the test set is 93.63
The confusion matrix is
[[17720  1278]
 [ 1141 17858]]
```

LSTM version2:
25% of train data

- Train data further split to 25%
- **train_data length: 125,982**
- **Results:**
  - Best val_acc: 90.67 %
  - Time req for all epochs > 40 min
  - val_acc decreased after epoch 1
    - Overfitting
    - Solution is to reduce the learning rate after epoch 1
  - **test_acc using the above best model: 91.98%**

- Fortunately, shrinking the training data didn't affect our test accuracy by a lot (decreased from 93.63% to 91.98, 1.7% decrease)
- So we decided to consider the shrinked data for all the pretrained models as well, so that we can compare their accuracy with LSTM.

```
Starting training loop...
Epoch 0: 3937it [03:56, 16.65it/s, Training Loss: 0.31155]
Epoch 0 | Train Loss 0.31155, Train Acc 91.65, Val Acc 90.19
The model has been saved!
Epoch 1: 3937it [03:54, 16.80it/s, Training Loss: 0.21484]
Epoch 1 | Train Loss 0.21484, Train Acc 93.98, Val Acc 90.67
The model has been saved!
Epoch 2: 3937it [03:54, 16.81it/s, Training Loss: 0.17009]
Epoch 2 | Train Loss 0.17009, Train Acc 95.23, Val Acc 90.29
Epoch 3: 3937it [03:55, 16.74it/s, Training Loss: 0.13860]
Epoch 3 | Train Loss 0.13860, Train Acc 96.05, Val Acc 89.97
Epoch 4: 3937it [03:54, 16.80it/s, Training Loss: 0.11315]
Epoch 4 | Train Loss 0.11315, Train Acc 96.87, Val Acc 89.61
Epoch 5: 3937it [03:53, 16.83it/s, Training Loss: 0.09169]
Epoch 5 | Train Loss 0.09169, Train Acc 97.54, Val Acc 89.56
Epoch 6: 3937it [03:53, 16.83it/s, Training Loss: 0.07487]
Epoch 6 | Train Loss 0.07487, Train Acc 97.53, Val Acc 88.93
Epoch 7: 3937it [03:54, 16.81it/s, Training Loss: 0.06261]
Epoch 7 | Train Loss 0.06261, Train Acc 97.90, Val Acc 88.87
Epoch 8: 3937it [03:54, 16.81it/s, Training Loss: 0.05364]
Epoch 8 | Train Loss 0.05364, Train Acc 98.45, Val Acc 89.06
Epoch 9: 3937it [03:54, 16.81it/s, Training Loss: 0.04750]
Epoch 9 | Train Loss 0.04750, Train Acc 97.87, Val Acc 88.48
The accuracy on the test set is 91.98
The confusion matrix is
[[17466  1532]
 [ 1514 17485]]
```

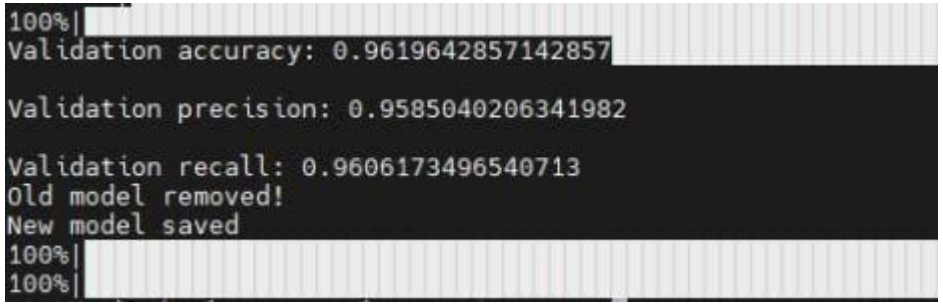
Pretrained Model 1: DistilBERT

What is DistilBert and how does it work?

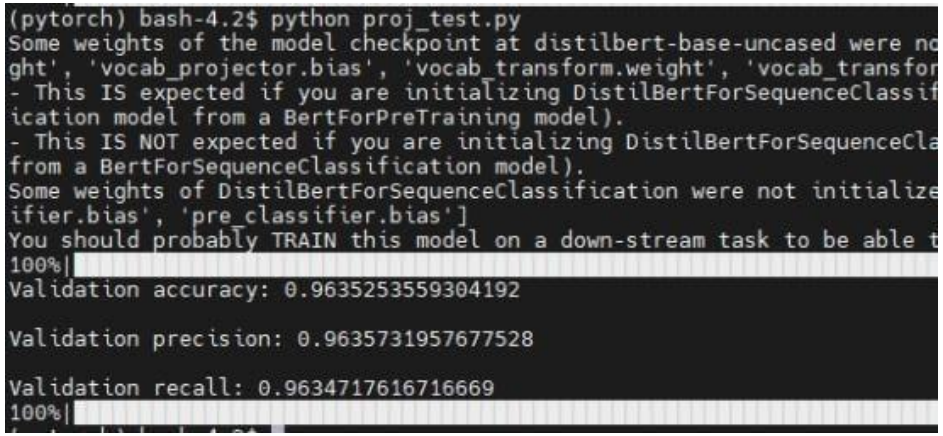
- DistilBERT is a compact, quick, inexpensive, and light Transformer model that has been trained using BERT basis.
- It has 40% less parameters than bert-base-uncased, and it runs 60% quicker while keeping over 95% of BERT's performance on the GLUE language comprehension benchmark.
- DistilBERT employs a process known as distillation to simulate Google's BERT, which entails replacing a huge neural network with a smaller one.
- After a big neural network has been trained, the whole output distributions of the network can be approximated using a smaller network.

Pretrained Model 1: DistilBERT

- **train_data length: 112,000**
- val_data length: 28,000
- test_data length: 37,997
- Parameters:
  - Learning rate: 5e-5
  - Batch size: 8
  - Epoch size: 3
- **Results:**
  - val_acc: 96.19%
  - **test_acc: 96.35%**



Validation data accuracy

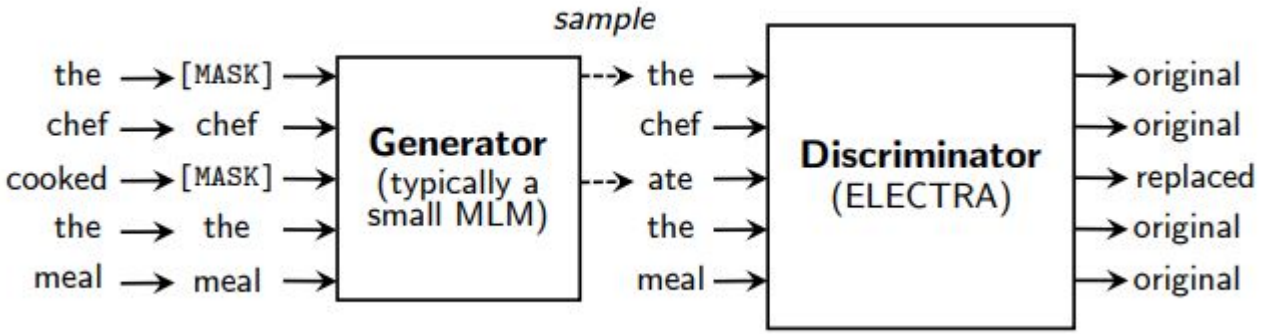


Test data accuracy

Pretrained Model 2: ELECTRA

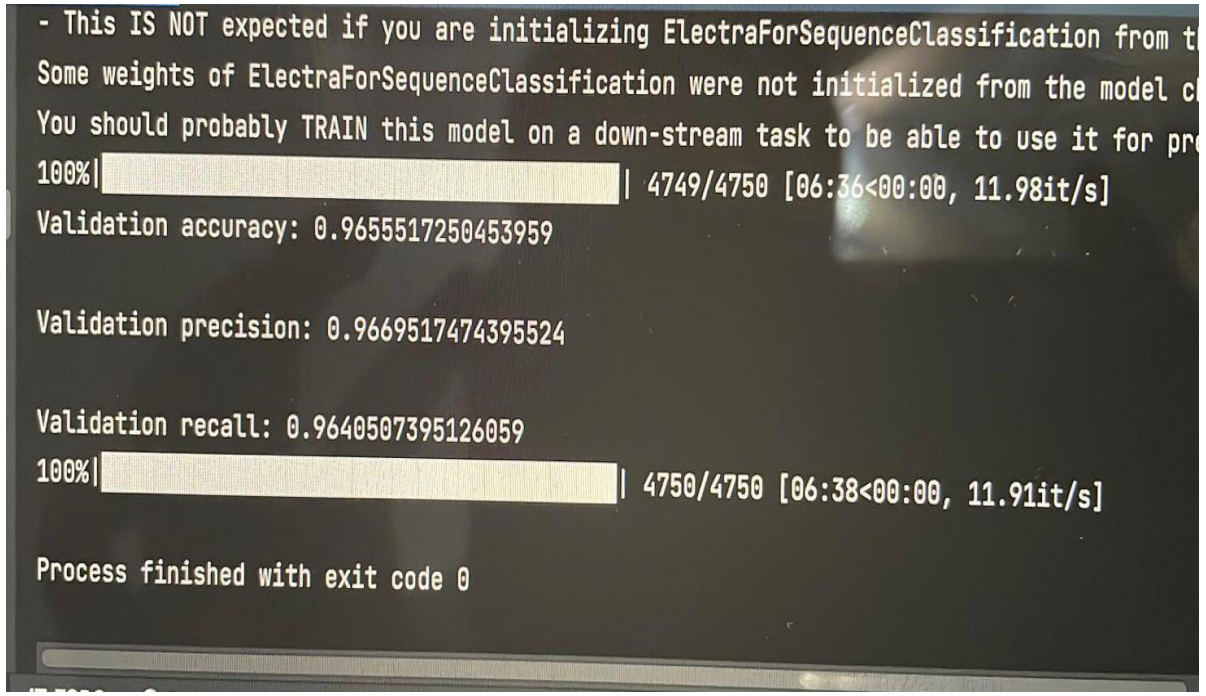
What is Electra and How does it work?

- ELECTRA is a new pre-training approach which trains two transformer models: the generator and the discriminator.
- The generator's role is to replace tokens in a sequence, and is therefore trained as a masked language model.
- The discriminator, attempts to determine which tokens in the sequence were substituted by the generator.



Pretrained Model 2: ELECTRA

- **train_data length: 112,000**
- val_data length: 28,000
- test_data length: 37,997
- Parameters:
  - Learning rate: 5e-5
  - Batch size: 8
  - Epoch size: 3
- **Results:**
  - Best val_acc: 96.21%
  - **test_acc using the above best model: 96.55%**



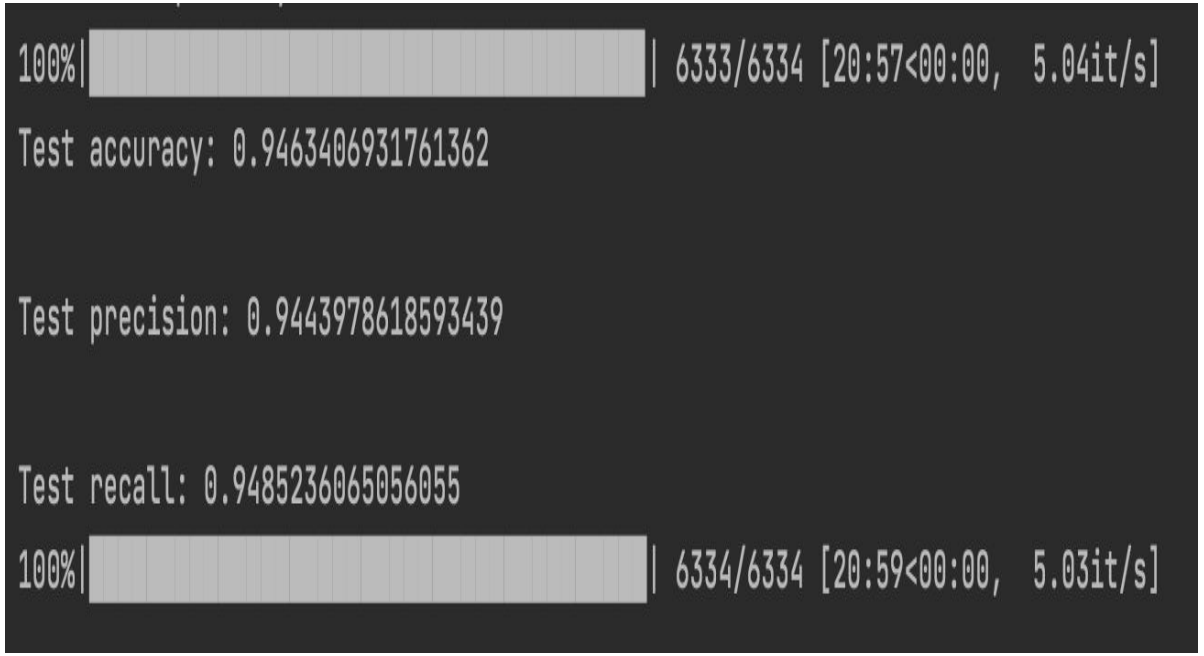
Ensemble of DistilBERT and ELECTRA

- Training dataset were fed to both DistilBert and Electra models separately.
- Outputs of these two models were fed to a Linear layer which acts as a classifier.
- The code snippet below shows the structure of the ensemble.

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.m1 = DistilBertForSequenceClassification.from_pretrained(checkpoint1)
        self.m2 = ElectraForSequenceClassification.from_pretrained(checkpoint2)
        self.dropout = nn.Dropout(0.3)
        self.out3 = nn.Linear(4,2)
    def forward(self, ids):
        output_1 = self.m1(ids, return_dict=False)
        output_2 = self.dropout(output_1[0])
        output_3 = self.m2(ids, return_dict=False)
        output_4 = self.dropout(output_3[0])
        output_5 = torch.cat((output_2, output_4), dim=1)
        output = self.out3(output_5)
        return output
```

Ensemble of DistilBERT and ELECTRA

- **train_data length: 112,000**
- val_data length: 28,000
- test_data length: 37,997
- Parameters:
  - Learning rate: 5e-5
  - Batch size: 8
  - **Epoch size: 1**
- **Results:**
  - Best val_acc: 94.03%
  - **test_acc using the above best model: 94.63%**



Observations and Conclusion

- The pretrained models have given higher accuracy on the test dataset than the LSTM model, even by using lesser training data, which was expected.
- It is worth noticing that we have used the raw data without any cleaning for our pretrained models.
- This is because they utilise the structure of the sentence from both directions to connect every output element to every input element. So, removing stopwords and punctuations might confuse the model and reduce the accuracy.

Model	Number of train data points	Accuracy
LSTM	125,982	91.98%
DistilBERT	112,000	96.35%
ELECTRA	112,000	96.55%
Ensemble*	112,000	94.63%

* The ensemble model was trained using only 1 epoch.

Reference

1. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
2. https://huggingface.co/docs/transformers/index
3. https://www.kdnuggets.com/2019/09/bert-roberta-distilbert-xlnet-one-use.html
4. http://ankit-ai.blogspot.com/2021/02/understanding-state-of-art-language.html
5. https://github.com/amir-jafari/Deep-Learning/blob/master/Pytorch/RNN/2_TextClassification/example_LSTM_sentiment_analysis.py

Any questions?