Smart Test Selection Mechanism Model Approach

Model Introduction:

" Introducing a "Smart Test Selection Mechanism" using PyTorch and deep learning. This model automates test case selection for code changes in a continuous integration pipeline, optimizing efficiency and scalability. With LSTM architecture and label encoding, it streamlines software development workflows by intelligently recommending pertinent tests. "

1. Data Loading and Exploration

- Load the dataset using Pandas and inspect its structure.

```
import pandas as pd
df1 = pd.read_csv('data.csv')
```

2. Data Preprocessing and Tokenization

- Extract code changes and smart test selection mechanism columns.
- Tokenize code changes using the Keras Tokenizer.
- Pad sequences to ensure consistent input size.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

code_changes = df1['Code Changes'].tolist()
smart_test_selection = df1['Smart Test Selection Mechanism'].tolist()

tokenizer = Tokenizer()
tokenizer.fit_on_texts(code_changes)
total_words = len(tokenizer.word_index) + 1
```

```
input_sequences = tokenizer.texts_to_sequences(code_changes)
input_sequences = pad_sequences(input_sequences)
```

3. Label Encoding

- Convert the 'smart_test_selection' column to numerical labels.

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(smart_test_selection)
```

4. PyTorch Dataset and DataLoader

- Implement a custom PyTorch Dataset for efficient data handling.
- Create a DataLoader for batching and shuffling.

```
import torch
from torch.utils.data import Dataset, DataLoader

class CustomDataset(Dataset):
    def __init__(self, sequences, labels):
        self.sequences = torch.LongTensor(sequences)
        self.labels = torch.LongTensor(labels)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.sequences[idx], self.labels[idx]

# Create PyTorch Dataset and DataLoader
dataset = CustomDataset(input_sequences, encoded_labels)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

5. Model Architecture

- Design a PyTorch model for sequence classification using LSTM.
- Embedding layer, LSTM layer, and fully connected layer with softmax activation.

```
class PyModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, lstm_hidden_dim, output_size):
        super(PyModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, lstm_hidden_dim, batch_first=True)
        self.fc = nn.Linear(lstm_hidden_dim, output_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.embedding(x)
        _, (x, _) = self.lstm(x)
        x = x.squeeze(0)
        x = self.fc(x)
        x = self.softmax(x)
        return x

# Instantiate PyTorch Model
pymodel = PyModel(vocab_size=total_words, embedding_dim=100,lstm_hidden_dim=100,
output_size=len(set(encoded_labels)))
```

6. Loss and Optimizer

- Choose the appropriate loss function (CrossEntropyLoss) and optimizer (Adam).

```
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(pymodel.parameters(), lr=0.001)
```

7. Training Loop

- Train the model using the prepared DataLoader.

```
num_epochs = 17
for epoch in range(num_epochs):
    for batch_inputs, batch_labels in dataloader:
        # Training steps
    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')
```

8. Evaluation and Testing

- Evaluate the trained model on a separate test set.
- Analyze performance metrics such as accuracy, precision, and recall.

Gradually, the loss value is decreasing, indicating a better performance of the application

```
        print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')

[8]                                                                    Python

Epoch 1/17, Loss: 1.7637227773666382
Epoch 2/17, Loss: 1.7418612241744995
Epoch 3/17, Loss: 1.7014516592025757
Epoch 4/17, Loss: 1.6553884744644165
Epoch 5/17, Loss: 1.4415475130081177
Epoch 6/17, Loss: 1.3485851287841797
Epoch 7/17, Loss: 1.2781450748443604
Epoch 8/17, Loss: 1.254122257232666
Epoch 9/17, Loss: 1.2256451845169067
Epoch 10/17, Loss: 1.1572619676589966
Epoch 11/17, Loss: 1.1806937456130981
Epoch 12/17, Loss: 1.2686513662338257
Epoch 13/17, Loss: 1.2339955568313599
Epoch 14/17, Loss: 1.276078701019287
Epoch 15/17, Loss: 1.14839768409729
Epoch 16/17, Loss: 1.179030179977417
Epoch 17/17, Loss: 1.0586559772491455
```

confusion matrix

```
Confusion Matrix:
[[3 0 0 0 0 0]
 [0 2 0 0 0 0]
 [0 0 3 0 0 0]
 [0 0 0 2 0 0]
 [0 0 0 0 1 0]
 [0 0 0 0 0 2]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         3
...
    accuracy                           1.00        13
   macro avg       1.00      1.00      1.00        13
weighted avg       1.00      1.00      1.00        13

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

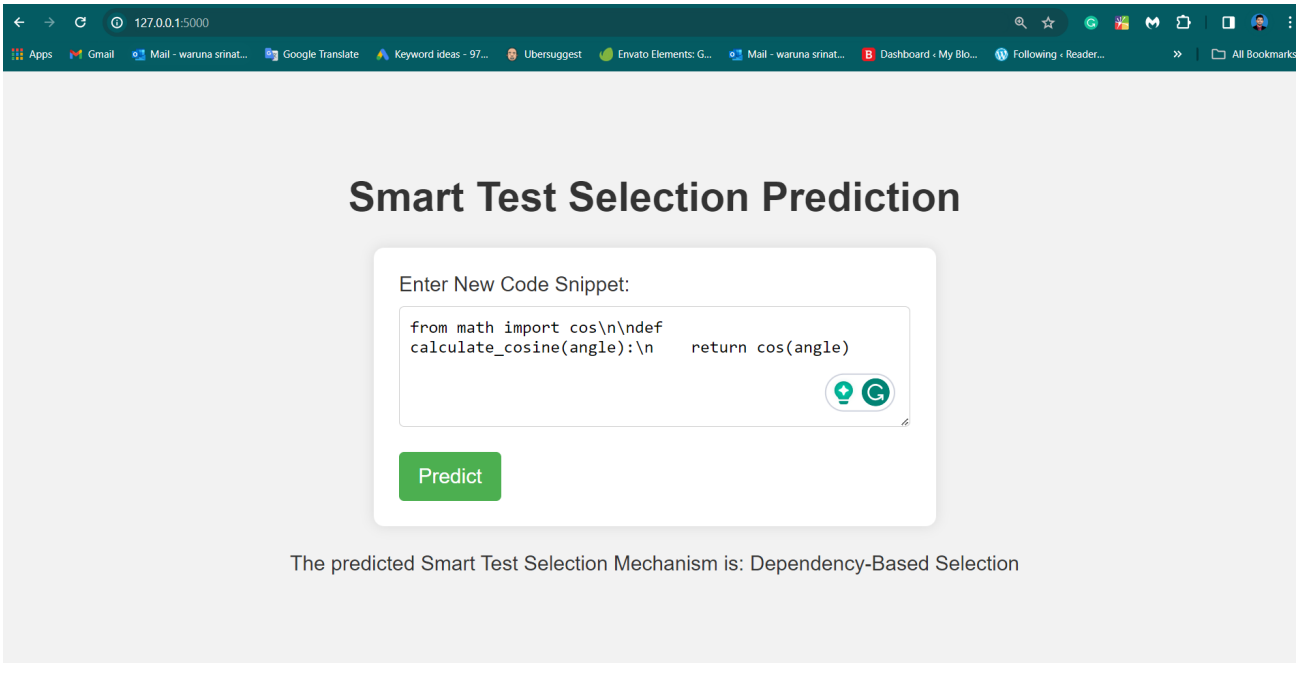
After training the dataset using the TensorFlow framework, I compared the accuracy with the PyTorch model-trained accuracy. The PyTorch model demonstrated superior performance. Therefore, I chose the PyTorch-trained model for making predictions.

9. Saving The Trained Model & Loarding The Saved Model

Save the PyTorch model (in .pth format) after training, and later load it for making future predictions.

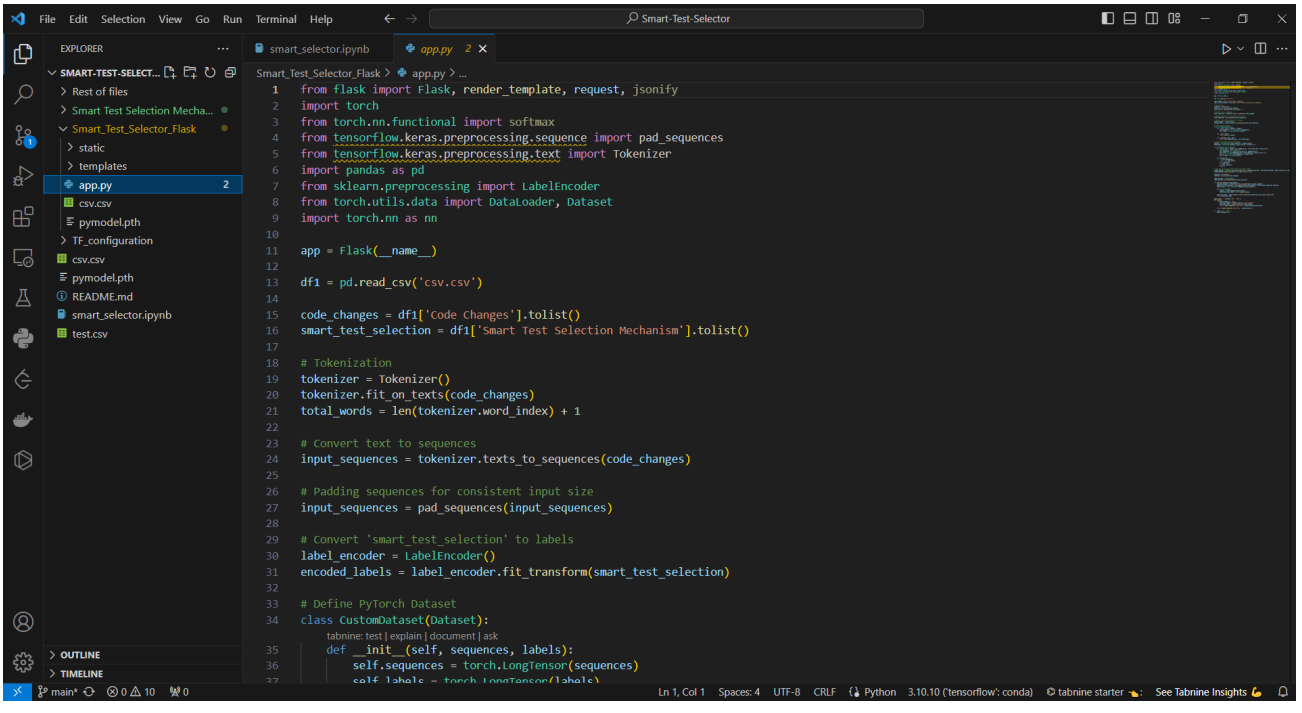
10. Integration and Deployment

After training the model, I successfully deployed the application using Flask. This is the Flask user interface (UI)



```
(tensorflow) C:\Users\warun\Documents\code\Smart-Test-Selector\Smart_Test_Selector_Flask>python app.py
C:\Users\warun\anaconda3\envs\tensorflow\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\warun\anaconda3\envs\tensorflow\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
C:\Users\warun\anaconda3\envs\tensorflow\lib\site-packages\numpy\.libs\libopenblas64__v0.3.23-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
C:\Users\warun\anaconda3\envs\tensorflow\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\warun\anaconda3\envs\tensorflow\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
C:\Users\warun\anaconda3\envs\tensorflow\lib\site-packages\numpy\.libs\libopenblas64__v0.3.23-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
 * Debugger is active!
 * Debugger PIN: 841-570-895
127.0.0.1 - - [27/Dec/2023 19:26:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Dec/2023 19:26:56] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - - [27/Dec/2023 19:27:19] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [27/Dec/2023 19:27:19] "GET /static/style.css HTTP/1.1" 304 -
```

Flask Application Local Port:- http://127.0.0.1:5000



Conclusion

The PyTorch model, incorporating LSTM for sequence analysis, accurately predicts test case relevance in the Smart Test Selection Mechanism. Through rigorous training, the model gains a robust understanding of code changes, presenting an intelligent solution for optimized test case selection. By leveraging deep learning principles, the model excels in capturing intricate patterns within input sequences. Its deployment shows promise in enhancing testing processes, offering a reliable and efficient approach to smart test selection in software development pipelines.

Thank you!

Document by Waruna Srinath