(Model Test)INM434 Natural Language Processing_Yumi Heo_code

May 8, 2024

INM434 Natural Language Processing MSc Data Science | Yumi Heo | 230003122

Google Colab Folder Link: https://drive.google.com/drive/folders/1Yn99YR6d5iJ79NYjdZwLLUTEPmucnNqH?u

Model Training Code Google Colab Link:https://colab.research.google.com/drive/1nTohqQt6vPr6GIj6mX9jmtQia

Model Test Code Google Colab Link: https://colab.research.google.com/drive/1OgBrPqldusG9K2o68J3SBCF-pTexFih0?usp=sharing

[ ]:
```
# Mount the drive to load files and models
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

1 TEST

1.1 DistilBERT (ver. trained with preprocessed training set)

[ ]:
```
# Import libraries for test
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```

[ ]:
```
# Specify the testset file path
test_file_path = "/content/drive/MyDrive/1. NLP CW/DistilBERT/test set/testset.
↪csv"

# Load the test set
test_df = pd.read_csv(test_file_path)

# Print the test dataframe
print(test_df)
```

```
                                   text  label
0                          locate card     11
```

```
1     still received new card ordered week ago     11
2             ordered card arrived help please     11
3                         way know card arrive     11
4                             card arrived yet     11
…                                          …    …
3075                      im uk still get card     24
3076                    many countries support     24
3077                        countries business     24
3078                         countries operate     24
3079                   card mailed used europe     24

[3080 rows x 2 columns]
```

[ ]:
```
# Install datasets from huggingface
!pip install datasets
```

[ ]:
```
# Import the library to change the test set to Hugging Face Dataset
from datasets import Dataset

# Convert the dataframe to a Hugging Face Dataset
testset = Dataset.from_dict(test_df)
```

[ ]:
```
# Check the inside of testset
testset
```

[ ]:
```
Dataset({
    features: ['text', 'label'],
    num_rows: 3080
})
```

[ ]:
```
# This code is derived from lab tutorial 8 and the training code notebook
# Import libraries to tokenize
from transformers import DistilBertTokenizer

# Tokenization
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132:
FutureWarning: `resume_download` is deprecated and will be removed in version
1.0.0. Downloads always resume when possible. If you want to force a new
download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
```

```
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

[ ]:
```python
# This code is derived from lab tutorial 8 and the training code notebook
# Tokenize the test data
def tokenize(batch):
    return tokenizer(batch['text'], padding='max_length', truncation=True,␣
 ↪max_length=29) # Define the maximum length as 29

test_set = testset.map(tokenize, batched=True)
```

```
Map:   0%|          | 0/3080 [00:00<?, ? examples/s]
```

[ ]:
```python
# Set the data format
test_set.set_format('pt', columns=['input_ids', 'attention_mask', 'label'])
```

[ ]:
```python
# Set up the data loader
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32,␣
 ↪shuffle=False)
```

[ ]:
```python
# Define the folder path to load the model's state dictionary
folder_path = '/content/drive/MyDrive/1. NLP CW/DistilBERT/'

# Define the dictionary file path
model_save_path = folder_path + 'distilBERT_model.pth'
```

[ ]:
```python
# This code is derived from lab tutorial 8 and the training code notebook
# Define the model architecture to load the model
class DistilBERT(nn.Module):
    def __init__(self, model):
        super(DistilBERT, self).__init__()
        self.model = model
        self.linear = nn.Linear(768, 77) # 77 classes

    def forward(self, input_ids, attention_mask):
        outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)
        last_hidden_state = outputs.last_hidden_state[:, 0, :]
        logits = self.linear(last_hidden_state)
        return logits
```

[ ]:
```python
# Load the best model to test
test_model = torch.load('/content/drive/MyDrive/1. NLP CW/DistilBERT/
 ↪processed_distilbert.bin', map_location=torch.device('cpu')) # Make sure to␣
 ↪run in CPU
# Match the state dictionary to the loaded model
```

```
state_dict = torch.load(model_save_path, map_location=torch.device('cpu')) #␣
 ↪Make sure to run in CPU
test_model.load_state_dict(state_dict)
```

[ ]:
```
<All keys matched successfully>
```

[ ]:
```
# Import the library to evaluate the final model
from sklearn.metrics import precision_score, recall_score, f1_score

# Define the test function
def evaluate(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    predictions_list = []
    labels_list = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids']
            attention_mask = batch['attention_mask']
            labels = batch['label']

            outputs = model(input_ids, attention_mask)
            predictions = torch.argmax(outputs, dim=1)
            correct += (predictions == labels).sum().item()
            total += labels.size(0)

            predictions_list.extend(predictions.cpu().numpy()) # Make sure it␣
 ↪will run in CPU
            labels_list.extend(labels.cpu().numpy()) # Make sure it will run in␣
 ↪CPU

    accuracy = correct / total
    precision = precision_score(labels_list, predictions_list,␣
 ↪average='weighted')
    recall = recall_score(labels_list, predictions_list, average='weighted')
    f1 = f1_score(labels_list, predictions_list, average='weighted')

    return accuracy, precision, recall, f1
```

[ ]:
```
# Test the model
test_model.eval()

# Get the test accuracy
test_accuracy, test_precision, test_recall, test_f1 = evaluate(test_model,␣
 ↪test_loader)
```

```
print(f'Test Accuracy: {round((test_accuracy*100), 2)}')
print(f'Test Precision: {round((test_precision*100), 2)}')
print(f'Test Recall: {round((test_recall*100), 2)}')
print(f'Test F1 Score: {round((test_f1*100), 2)}')
```

```
Test Accuracy: 89.55
Test Precision: 90.08
Test Recall: 89.55
Test F1 Score: 89.56
```

2 TEST

2.1 LSTM (ver. tuned with Word2Vec embedding and dropout)

[ ]:
```
# Import libraries for test
from tensorflow.keras.models import load_model
import numpy as np
```

[ ]:
```
# load the LSTM model
model = load_model('/content/drive/MyDrive/1. NLP CW/LSTM/
  ↪dropout_tuned_LSTM_word2vec_model.keras')
```

[ ]:
```
# Load the test data for the LSTM model
X_test_padded = np.load('/content/drive/MyDrive/1. NLP CW/LSTM/test set/
  ↪X_test_padded.npy')
y_test_array = np.load('/content/drive/MyDrive/1. NLP CW/LSTM/test set/
  ↪y_test_array.npy')
```

[ ]:
```
# Test the model with test dataset
loss, accuracy = model.evaluate(X_test_padded, y_test_array)

print("Test Loss:", loss)
print("Test Accuracy:", round((accuracy*100), 2))
```

```
97/97 [==============================] - 2s 12ms/step - loss: 0.7372 - accuracy:
0.7984
Test Loss: 0.7371559143066406
Test Accuracy: 79.84
```

[ ]:
```
# Import the library to check precision, recall, and F1 score
from sklearn.metrics import precision_score, recall_score, f1_score

# Check predictions with the test set
y_test_prob = model.predict(X_test_padded)

# Convert probabilities to class labels
y_test_pred = np.argmax(y_test_prob, axis=1)
```

```python
# Calculate precision, recall, and f1 score
precision = precision_score(y_test_array, y_test_pred, average='weighted')
recall = recall_score(y_test_array, y_test_pred, average='weighted')
f1 = f1_score(y_test_array, y_test_pred, average='weighted')

print("Precision:", round((precision*100), 2))
print("Recall:", round((recall*100), 2))
print("F1 Score:", round((f1*100), 2))
```

```
97/97 [==============================] - 2s 12ms/step
Precision: 81.46
Recall: 79.84
F1 Score: 79.54
```