

# NLP-4

Narasimha Rohit Katta

April 2024

## 1 Question-2

### 1.1 A - H

#### Part (a)

Implement a function called `read_data(...)`.

```
#a
import os

def read_data(pos_train_path, neg_train_path, pos_test_path, neg_test_path):
    train_texts, train_labels = [], []
    test_texts, test_labels = [], []

    def read_files(directory, texts, labels, label):
        for filename in os.listdir(directory):
            filepath = os.path.join(directory, filename)
            with open(filepath, 'r', encoding='utf-8') as file:
                texts.append(file.read())
            labels.append(label)

    read_files(pos_train_path, train_texts, train_labels, 'pos')
    read_files(neg_train_path, train_texts, train_labels, 'neg')
    read_files(pos_test_path, test_texts, test_labels, 'pos')
    read_files(neg_test_path, test_texts, test_labels, 'neg')

    return train_texts, train_labels, test_texts, test_labels

base_path = '/content/drive/My Drive/dataset'

train_texts, train_labels, test_texts, test_labels = read_data(
    os.path.join(base_path, 'pos_train'),
    os.path.join(base_path, 'neg_train'),
    os.path.join(base_path, 'pos_test'),
    os.path.join(base_path, 'neg_test')
)
```

## Part (b)

Create a development dataset.

```
#b
from sklearn.model_selection import train_test_split

all_texts = train_texts
all_labels = train_labels

label_to_index = {'pos': 1, 'neg': 0}
all_labels_numeric = [label_to_index[label] for label in all_labels]

train_texts, dev_texts, train_labels_numeric, dev_labels_numeric = train_test_split(
    all_texts,
    all_labels_numeric,
    stratify=all_labels_numeric,
    test_size=200,
    random_state=42
)

index_to_label = {1: 'pos', 0: 'neg'}
train_labels = [index_to_label[label] for label in train_labels_numeric]
dev_labels = [index_to_label[label] for label in dev_labels_numeric]
```

## Part (c)

Tokenize the texts using DistilBertTokenizerFast.





```
#c
from transformers import DistilBertTokenizerFast

# Initializing the tokenizer
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

# Tokenizing the training data
train_encodings = tokenizer(train_texts, truncation=True, padding=True)

# Tokenizing the development data
dev_encodings = tokenizer(dev_texts, truncation=True, padding=True)

# Tokenizing the test data
test_encodings = tokenizer(test_texts, truncation=True, padding=True)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: 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 yo
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(
tokenizer.config.json: 100%  28.0/28.0 [00:00<00:00, 1.84kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 11.9MB/s]
tokenizer.json: 100%  466k/466k [00:13<00:00, 35.8kB/s]
config.json: 100%  483/483 [00:00<00:00, 27.0kB/s]
```

## Part (d)

Convert tokenized data and labels into a Dataset object.

```
#d
import torch
from torch.utils.data import Dataset

class PolarityDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = [1 if label == 'pos' else 0 for label in labels]

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

train_dataset = PolarityDataset(train_encodings, train_labels)
dev_dataset = PolarityDataset(dev_encodings, dev_labels)
test_dataset = PolarityDataset(test_encodings, test_labels)
```

## Part (e)

Code for fine-tuning with native PyTorch using DistilBert.

```
#e
import torch
from torch.utils.data import DataLoader
from transformers import DistilBertForSequenceClassification, AdamW
from tqdm import tqdm

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
optimizer = AdamW(model.parameters(), lr=1e-4)
model.train()

for epoch in range(4):
    total_loss = 0
    for batch in tqdm(train_loader, desc=f"Training Epoch {epoch + 1}"):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f"Average loss for epoch {epoch + 1}: {total_loss / len(train_loader)}")

model.eval()
dev_loader = DataLoader(dev_dataset, batch_size=64, shuffle=False)
correct_predictions = 0
total_samples = 0
with torch.no_grad():
    for batch in tqdm(dev_loader, desc="Evaluating on Development Set"):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        preds = torch.argmax(outputs.logits, dim=-1)
        correct_predictions += (preds == batch['labels']).sum().item()
        total_samples += batch['labels'].size(0)
dev_accuracy = correct_predictions / total_samples
print(f"Accuracy on the development set: {dev_accuracy:.4f}")
```

```

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Training Epoch 1: 100% [██████████] 100/100 [01:16:00:00, 1.311t/s]
Average loss for epoch 1: 0.5695039343833923
Training Epoch 2: 100% [██████████] 100/100 [01:16:00:00, 1.301t/s]
Average loss for epoch 2: 0.30907258979976177
Training Epoch 3: 100% [██████████] 100/100 [01:16:00:00, 1.301t/s]
Average loss for epoch 3: 0.159090800671301
Evaluating on Development Set: 100% [██████████] 4/4 [00:03:00:00, 1.071t/s] Accuracy on the development set: 0.8250

```

In this training run, the `DistilBertForSequenceClassification` model was fine-tuned over three epochs. Starting with a training loss of approximately 0.5695, the model showed rapid learning, with the loss decreasing to about 0.3090 in the second epoch and further to roughly 0.1599 in the third epoch. This progression indicates effective learning and model improvement over time. The development set accuracy post-training stood at 82.50%, reflecting a promising level of generalization. The consistent reduction in training loss and a development set accuracy over 80% suggest that the model has a strong potential for accurately classifying new, unseen data.

## Part (f)

Add code to print training and validation loss.

```

#f
import torch
from torch.utils.data import DataLoader
from transformers import DistilBertForSequenceClassification, AdamW
from tqdm import tqdm

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
optimizer = AdamW(model.parameters(), lr=1e-4)
model.train()

for epoch in range(4): # Train for 3 epochs
    total_train_loss = 0
    for batch in tqdm(train_loader, desc=f"Training Epoch {epoch + 1}"):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_train_loss += loss.item()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    avg_train_loss = total_train_loss / len(train_loader)
    print(f"Average training loss for epoch {epoch + 1}: {avg_train_loss:.4f}")
    model.eval()
    dev_loader = DataLoader(dev_dataset, batch_size=64, shuffle=False)
    total_dev_loss = 0
    with torch.no_grad():
        for batch in tqdm(dev_loader, desc=f"Validating Epoch {epoch + 1}"):
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            loss = outputs.loss
            total_dev_loss += loss.item()

    avg_dev_loss = total_dev_loss / len(dev_loader)
    print(f"Average validation loss for epoch {epoch + 1}: {avg_dev_loss:.4f}")
    model.train()

```

```

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
FutureWarning: This implementation of AdamW is deprecated and will be
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:429: FutureWarning: This implementation of AdamW is deprecated and will be
warnings.warn(
Training Epoch 1: 100% [██████████] 100/100 [01:17:00:00, 1.291t/s]
Average training loss for epoch 1: 0.6661
Validating epoch 1: 100% [██████████] 4/4 [00:03:00:00, 1.001t/s]
Average validation loss for epoch 1: 0.5153
Training Epoch 2: 100% [██████████] 100/100 [01:16:00:00, 1.301t/s]
Average training loss for epoch 2: 0.4330
Validating epoch 2: 100% [██████████] 4/4 [00:03:00:00, 1.071t/s]
Average validation loss for epoch 2: 0.4237
Training Epoch 3: 100% [██████████] 100/100 [01:16:00:00, 1.311t/s]
Average training loss for epoch 3: 0.1974
Validating epoch 3: 100% [██████████] 4/4 [00:03:00:00, 1.071t/s] Average validation loss for epoch 3: 0.4376

```

During the three-epoch fine-tuning of the DistilBertForSequenceClassification model, training loss showed a steady decline from 0.6661 in the first epoch to 0.1974 by the third, indicating effective learning. However, the validation loss, after initially decreasing, slightly increased to 0.4376 in the third epoch, hinting at potential overfitting. This discrepancy suggests the model's increasing specialization to the training data, which may not fully generalize to the validation data.

## Part (g)

Obtain and print the performance metrics on the test data.

```
#g
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
model.eval()
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
true_labels = []
predictions = []

with torch.no_grad():
    for batch in tqdm(test_loader, desc="Evaluating on Test Set"):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        preds = torch.argmax(outputs.logits, dim=-1)
        predictions.extend(preds.cpu().numpy())
        true_labels.extend(batch['labels'].cpu().numpy())
predictions = np.array(predictions)
true_labels = np.array(true_labels)
accuracy = accuracy_score(true_labels, predictions)
precision = precision_score(true_labels, predictions, pos_label=1) # 1 corresponds to 'pos'
recall = recall_score(true_labels, predictions, pos_label=1)
f1 = f1_score(true_labels, predictions, pos_label=1)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision (pos): {precision:.4f}")
print(f"Recall (pos): {recall:.4f}")
print(f"F1 Score (pos): {f1:.4f}")
```

```
Evaluating on Test Set: 100%|██████████| 4/4 [00:03<00:00, 1.18it/s]Accuracy: 0.8800
Precision (pos): 0.8519
Recall (pos): 0.9200
F1 Score (pos): 0.8846
```

The evaluation on the test set yielded a high accuracy of 88%, indicating strong overall model performance. The precision for the 'pos' class was 85.19%, suggesting that when the model predicts a positive outcome, it is correct most of the time. The recall for 'pos' was higher at 92%, indicating the model's proficiency in identifying most true positive cases. The F1 score, a balance of precision and recall, was also high at 88.46%, reflecting the model's robustness in classifying the positive class.

## Part (h)

Run the code with specified parameters and report the results:-

During the fine-tuning of the DistilBERT model with the AdamW optimizer, a learning rate of  $5 \times 10^{-5}$ , and a batch size of 16, the training loss decreased significantly over the course of 3 epochs. By the end of the third epoch, the training loss had reduced to approximately 0.1974, indicating substantial learning from the training data. However, the validation loss showed a slight increase to around 0.4376 in the third epoch, suggesting potential overfitting to the training data.

On the test set, the model achieved an accuracy of 88%. Precision for the positive class was around 85.19%, while recall was higher at 92%, showing a strong ability to identify positive instances. The F1 score was calculated to be 88.46%, demonstrating a good balance between precision and recall. These metrics indicate a proficient model that performs well in classifying the positive class, but with room for improvement in generalization, as indicated by the validation loss trend.

## Part (I)

Explore different hyperparameters and discuss the changes.

## Learning Rate Change

### Results

After increasing the learning rate to  $1 \times 10^{-4}$ , the model's training loss continued to decrease across epochs, indicative of ongoing learning. However, the higher learning rate may have led to a higher validation loss in the final epoch, suggesting the potential for overfitting.

## Batch Size Change

### Results

The increase in batch size to 32 could have contributed to a more stable gradient estimation. However, the validation loss did not decrease proportionally, which could be attributed to the larger batch size or the increased potential for the model to overfit.

## Epoch Number Change

### Results

Extending training to 4 epochs resulted in a significant decrease in training loss, yet the validation loss increased. This suggests that while the model was continuing to learn, it might also have been overfitting to the training data.

## Test Set Performance

### Results

The model's accuracy on the test set slightly decreased to 79.00%. Precision for the positive class dropped to 71.97%, indicating more false positives, potentially due to overfitting. Recall remained very high at 95.00%, and the F1 score was at 81.90%, reflecting a decrease in precision but maintaining a strong overall performance.

## Part (J)

```
class CustomLayer(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(CustomLayer, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)
        self.relu = nn.ReLU()

    def forward(self, x):
        return self.relu(self.linear(x))

additional_layer = CustomLayer(model.classifier.in_features, model.classifier.in_features)

original_classifier = model.classifier
model.classifier = nn.Sequential(
    additional_layer,
    original_classifier
)

model.to(device)
```

The 'CustomLayer' addition enhances the model's complexity, introducing an extra linear and ReLU activation step. This change aims to improve the model's ability to learn and represent more intricate data patterns, potentially boosting its predictive performance. However, the increased complexity raises concerns about overfitting, necessitating vigilant evaluation of the model's performance on unseen data. The ultimate goal of this modification is to strike a balance between increased representational power and the model's ability to generalize well to new data.

An extra processing step is added to the model by stacking the CustomLayer before the original classifier. The data passes through this new layer before going to the final classification layer. The model may learn from the training data more effectively thanks to the CustomLayer. The model can capture more

```

Using device: cuda
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Training Epoch 1: 100% |██████████| 57/57 [01:24<00:00, 1.48s/it]
Average loss for epoch 1: 0.55914304593713
Training Epoch 2: 100% |██████████| 57/57 [01:24<00:00, 1.48s/it]
Average loss for epoch 2: 0.339913860047060
Training Epoch 3: 100% |██████████| 57/57 [01:24<00:00, 1.48s/it]
Average loss for epoch 3: 0.1500671936297183
Evaluating on Development Set: 100% |██████████| 4/4 [00:03<00:00, 1.11it/s]
Accuracy on the development set: 0.8300
Evaluating on Test Set: 100% |██████████| 7/7 [00:03<00:00, 2.11it/s] Accuracy: 0.9600
Precision (pos): 0.9225
Recall (pos): 1.0000
F1 Score (pos): 0.9615

```

Figure 1: Enter Caption

intricate correlations in the data than linear models can because of the non-linear activation function.

article amsmath graphicx

## Model Performance Comparison

The performance metrics before and after the addition of the custom layer are as follows:

### Before Custom Layer Addition:

- Test Set Accuracy: 88.00%
- Precision (Positive Class): 85.19%
- Recall (Positive Class): 92.00%
- F1 Score (Positive Class): 88.46%

### After Custom Layer Addition:

- Test Set Accuracy: 96.00%
- Precision (Positive Class): 92.59%
- Recall (Positive Class): 100.00%
- F1 Score (Positive Class): 96.15%

### Discussion of Performance Change:

The addition of the CustomLayer likely contributed to the model capturing more complex data patterns, thereby improving the representational power of the neural network. The ReLU activation in the custom layer may have introduced beneficial non-linearity that helped model complex relationships within the data.

The increase in precision and recall to the point of perfect recall could suggest that the additional parameters from the CustomLayer were able to fit the distinctions in the data that were previously missed. However, it is also crucial to ensure that the improvement isn't a result of overfitting. A more robust



validation process, possibly with k-fold cross-validation, would help confirm the model's true generalization ability.

The improvements across accuracy, precision, recall, and F1 score demonstrate that the custom layer's integration has strengthened the model's classification abilities, contributing to its higher performance on the test set. This suggests that the new architecture can better understand the nuances of the dataset, possibly capturing features and interactions that were not accounted for by the original model configuration.