```
from transformers import BertModel, BertTokenizer
import nltk
import pandas as pd
import torch
import torch.nn as nn
import seaborn as sns
import numpy as np
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from scipy.stats import norm
import math
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/tqdm/auto.py:22: TqdmWarning: IProgress not
  from .autonotebook import tqdm as notebook_tqdm
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Running on {}".format(device))
```

```
Running on cpu
```

```
def read_labels(filename):
    labels={}
    with open(filename) as file:
        for line in file:
            cols = line.split("\t")
            label = cols[1]
            if label not in labels:
                labels[label]=len(labels)
    return labels
```

```
def read_data(filename, labels, max_data_points=1000):

    data = []
    data_labels = []
    with open(filename) as file:
        for line in file:
            cols = line.split("\t")
            label = cols[1]
            text = cols[2]
```

```
            data.append(text)
            data_labels.append(labels[label])


    # shuffle the data
    tmp = list(zip(data, data_labels))
    random.shuffle(tmp)
    data, data_labels = zip(*tmp)

    if max_data_points is None:
        return data, data_labels

    return data[:max_data_points], data_labels[:max_data_points]
```

```
labels=read_labels("train.txt")
train_x, train_y=read_data("train.txt", labels, max_data_points=None)
dev_x, dev_y=read_data("dev.txt", labels, max_data_points=None)
test_x, test_y=read_data("test.txt", labels, max_data_points=None)
```

```
def evaluate(model, x, y):
    model.eval()
    corr = 0.
    total = 0.
    with torch.no_grad():
        for x, y in zip(x, y):
            y_preds=model.forward(x)
            for idx, y_pred in enumerate(y_preds):
                prediction=torch.argmax(y_pred)
                if prediction == y[idx]:
                    corr += 1.
                total+=1
    return corr/total, total
```

```
class BERTClassifier(nn.Module):

    def __init__(self, bert_model_name, params):
        super().__init__()

        self.model_name=bert_model_name
        self.tokenizer = BertTokenizer.from_pretrained(self.model_name, do_lower_case=params[
        self.bert = BertModel.from_pretrained(self.model_name)

        self.num_labels = params["label_length"]

        self.fc = nn.Linear(params["embedding_size"], self.num_labels)

    def get_batches(self, all_x, all_y, batch_size=16, max_toks=510):

        """ Get batches for input x, y data, with data tokenized according to the BERT tokeni
```

```
        (and limited to a maximum number of WordPiece tokens """

          batches_x=[]
          batches_y=[]

          for i in range(0, len(all_x), batch_size):

              current_batch=[]

              x=all_x[i:i+batch_size]

              batch_x = self.tokenizer(x, padding=True, truncation=True, return_tensors="pt", m
              batch_y=all_y[i:i+batch_size]

              batches_x.append(batch_x.to(device))
              batches_y.append(torch.LongTensor(batch_y).to(device))

          return batches_x, batches_y

      def forward(self, batch_x):

          bert_output = self.bert(input_ids=batch_x["input_ids"],
                          attention_mask=batch_x["attention_mask"],
                          token_type_ids=batch_x["token_type_ids"],
                          output_hidden_states=True)

        # We're going to represent an entire document just by its [CLS] embedding (at position
        # And use the *last* layer output (layer -1)
        # as a result of this choice, this embedding will be optimized for this purpose during

          bert_hidden_states = bert_output['hidden_states']

          out = bert_hidden_states[-1][:,0,:]

          out = self.fc(out)

          return out.squeeze()
```

```
def confidence_intervals(accuracy, n, significance_level):
    critical_value=(1-significance_level)/2
    z_alpha=-1*norm.ppf(critical_value)
    se=math.sqrt((accuracy*(1-accuracy))/n)
    return accuracy-(se*z_alpha), accuracy+(se*z_alpha)
```

```
def train(bert_model_name, model_filename, train_x, train_y, dev_x, dev_y, labels, embedding_

    bert_model = BERTClassifier(bert_model_name, params={"label_length": len(labels), "doLowe
    bert_model.to(device)

    batch_x, batch_y = bert_model.get_batches(train_x, train_y)
```

```
    dev_batch_x, dev_batch_y = bert_model.get_batches(dev_x, dev_y)

    optimizer = torch.optim.Adam(bert_model.parameters(), lr=1e-5)
    cross_entropy=nn.CrossEntropyLoss()

    num_epochs=30
    best_dev_acc = 0.
    patience=5

    best_epoch=0

    for epoch in range(num_epochs):
        bert_model.train()

        # Train
        for x, y in zip(batch_x, batch_y):
            y_pred = bert_model.forward(x)
            loss = cross_entropy(y_pred.view(-1, bert_model.num_labels), y.view(-1))
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        # Evaluate
        dev_accuracy, _=evaluate(bert_model, dev_batch_x, dev_batch_y)
        if epoch % 1 == 0:
            print("Epoch %s, dev accuracy: %.3f" % (epoch, dev_accuracy))
            if dev_accuracy > best_dev_acc:
                torch.save(bert_model.state_dict(), model_filename)
                best_dev_acc = dev_accuracy
                best_epoch=epoch
        if epoch - best_epoch > patience:
            print("No improvement in dev accuracy over %s epochs; stopping training" % patien
            break

    bert_model.load_state_dict(torch.load(model_filename))
    print("\nBest Performing Model achieves dev accuracy of : %.3f" % (best_dev_acc))
    return bert_model
```

```
# small BERT -- can run on laptop
bert_model_name="google/bert_uncased_L-2_H-128_A-2"
model_filename="mybert.model"
embedding_size=128
doLowerCase=True

# # bert-base -- slow on laptop; better on Colab
# bert_model_name="bert-base-cased"
# model_filename="mybert.model"
# embedding_size=768
# doLowerCase=False

model=train(bert_model_name, model_filename, train_x, train_y, dev_x, dev_y, labels, embeddin
```

```
Some weights of the model checkpoint at google/bert_uncased_L-2_H-128_A-2 were not used when initi
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on ano
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you e
Epoch 0, dev accuracy: 0.590
Epoch 1, dev accuracy: 0.590
Epoch 2, dev accuracy: 0.595
Epoch 3, dev accuracy: 0.595
Epoch 4, dev accuracy: 0.595
Epoch 5, dev accuracy: 0.590
Epoch 6, dev accuracy: 0.600
Epoch 7, dev accuracy: 0.600
Epoch 8, dev accuracy: 0.610
Epoch 9, dev accuracy: 0.595
Epoch 10, dev accuracy: 0.600
Epoch 11, dev accuracy: 0.625
Epoch 12, dev accuracy: 0.625
Epoch 13, dev accuracy: 0.635
Epoch 14, dev accuracy: 0.635
Epoch 15, dev accuracy: 0.650
Epoch 16, dev accuracy: 0.650
Epoch 17, dev accuracy: 0.650
Epoch 18, dev accuracy: 0.645
Epoch 19, dev accuracy: 0.650
Epoch 20, dev accuracy: 0.645
Epoch 21, dev accuracy: 0.650
No improvement in dev accuracy over 5 epochs; stopping training

Best Performing Model achieves dev accuracy of : 0.650
```

```
class_names = ['Primary', 'Secondary', 'Tertiary']
ax = sns.countplot(np.array(train_y))
plt.xlabel('Class')
plt.title('Train')
ax.set_xticklabels(class_names)
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pa
  FutureWarning
```

```
class_names = ['Primary', 'Secondary', 'Tertiary']
ax = sns.countplot(np.array(dev_y))
plt.xlabel('Class')
plt.title('Dev')
ax.set_xticklabels(class_names)
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pa
  FutureWarning
```

```
class_names = ['Primary', 'Secondary', 'Tertiary']
ax = sns.countplot(np.array(test_y))
plt.xlabel('Class')
```

```
plt.title('Test')
ax.set_xticklabels(class_names)
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pa
  FutureWarning
```

Matthew Moon / **INFO 159 Final Project** Published at Apr 22, 2022 Private

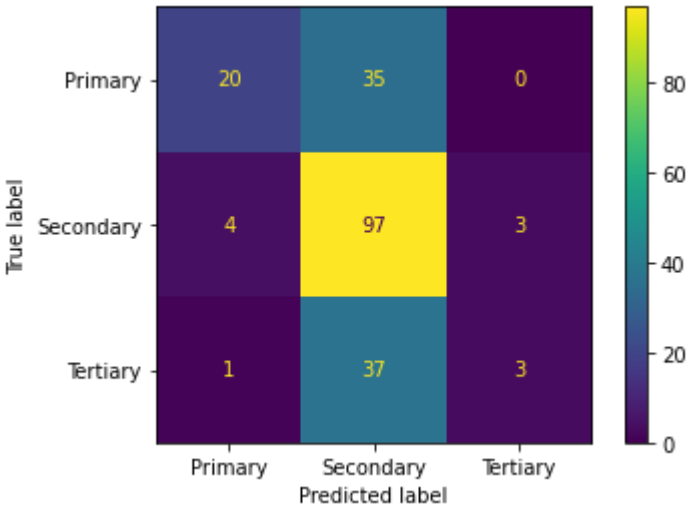
```
test_batch_x, test_batch_y = model.get_batches(test_x, test_y)
accuracy, test_n=evaluate(model, test_batch_x, test_batch_y)

lower, upper=confidence_intervals(accuracy, test_n, .95)
print("Test accuracy for best dev model: %.3f, 95%% CIs: [%.3f %.3f]\n" % (accuracy, lower, u
```

```
Test accuracy for best dev model: 0.600, 95% CIs: [0.532 0.668]
```

```
y_predictions = []
with torch.no_grad():
    for x, y in zip(test_batch_x, test_batch_y):
        y_preds=model.forward(x)
        for idx, y_pred in enumerate(y_preds):
            prediction=torch.argmax(y_pred)
            y_predictions += [prediction]
y_predictions_array = np.array(tf.concat(y_predictions, 0))
```

```
cm = confusion_matrix(test_y, y_predictions_array)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1288f0aad0>
```




```
len(y_predictions_array)
```