Open in Colab

BERT for binary or multiclass document classification using the [CLS] token as the document representation; trains a model (on `train.txt`), uses `dev.txt` for early stopping, and evaluates performance on `test.txt`. Reports test accuracy with 95% confidence intervals.

Before executing this notebook on Colab, make sure you're running on cuda (`Runtime > Change runtime type > GPU`) to make use of GPU speedups.

```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers
  Downloading transformers-4.28.1-py3-none-any.whl (7.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.0/7.0 MB 51.7 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.0)
Collecting huggingface-hub<1.0,>=0.11.0
  Downloading huggingface_hub-0.14.1-py3-none-any.whl (224 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 26.9 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.27.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.22.4)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 99.9 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.11.0->transformers
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.11.0->transformers) (2023.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2022.12.7)
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.14.1 tokenizers-0.13.3 transformers-4.28.1
```

```
from transformers import BertModel, BertTokenizer
import nltk
import torch
import torch.nn as nn
import numpy as np
import random
from scipy.stats import norm
import math
```

```
# If you have your folder of data on your Google drive account, you can connect that here
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Change this to the directory with your data
directory="/content/drive/MyDrive/ap_data/"
```

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

```
Running on cuda
```

```
def read_labels(filename):
    labels={}
    with open(filename) as file:
        for line in file:
            cols = line.split("₩t")
            label = cols[2]
            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[2]
                text = cols[3]

                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("%s/train.txt" % directory)
    labels=read_labels("%s/train.txt")


    # train_x, train_y=read_data("%s/train.txt" % directory, labels, max_data_points=None)
    train_x, train_y=read_data("%s/train.txt", labels, max_data_points=None)


    # dev_x, dev_y=read_data("%s/dev.txt" % directory, labels, max_data_points=None)
    dev_x, dev_y=read_data("%s/dev.txt", labels, max_data_points=None)


    # test_x, test_y=read_data("%s/test.txt" % directory, labels, max_data_points=None)
    test_x, test_y=read_data("%s/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["doLowerCase"], do_basic_tokenize=False)
            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=32, max_toks=510):

            """ Get batches for input x, y data, with data tokenized according to the BERT tokenizer
          (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", max_length=max_toks)
            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 0)
      # And use the *last* layer output (layer -1)
      # as a result of this choice, this embedding will be optimized for this purpose during the training process.

        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_size=768, doLowerCase=None):

    bert_model = BERTClassifier(bert_model_name, params={"label_length": len(labels), "doLowerCase":doLowerCase, "embedding_size":embedding_size})
    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" % patience)
            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, embedding_size=embedding_size, doLowerCase=doLowerCase)
```

```
Downloading                          213k/213k
(…)solve/main/vocab.txt:             [00:00<00:00,
100%                                 2.45MB/s]
Downloading                          29.0/29.0
(…)okenizer_config.json:             [00:00<00:00,
100%                                 1.61kB/s]
Downloading                          570/570
(…)lve/main/config.json:             [00:00<00:00,
100%                                 45.8kB/s]
```

```
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, upper))
```

```
Test accuracy for best dev model: 0.564, 95% CIs: [0.468 0.661]
```