Data Mining HW2 Report

In this homework,I chose to fine-tune pre-trained BERT model for the competition. BERT is a transformer-based machine learning model introduced

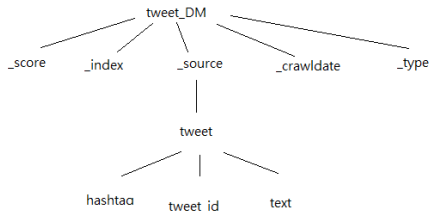
By Google. It's designed for NLP tasks and has become a foundational model for applications like text classification, question answering, and sentiment analysis.

The following are the steps I walked through:

First comes with some preprocessing step, I use google colab environment to run my code , and I have upload data of competition to my google drive then mount my drive to access these data.

```
from google.colab import drive
drive.mount('/content/drive')
```

After reading all the data needed as the dataframes,I observe the structure of the json dataset. The structure is like



Thus,I extract the columns under tweet to tweet_dm

```
# extract the columns under "tweet" to tweets_dm for simplification
tweets_dm["hashtags"] = tweets_dm['_source'].apply(lambda x: x['tweet']['hashtags'])
tweets_dm["tweet_id"] = tweets_dm['_source'].apply(lambda x: x['tweet']['tweet_id'])
tweets_dm["text"] = tweets_dm['_source'].apply(lambda x: x['tweet']['text'])
```

Then,apply identification tag by merging dataframe of tweet_dm and identification.

```
# merge tweets_dm and identification(append identification label to tweets_dm)
df_merged = tweets_dm.merge(identification,how="left",on="tweet_id")
```

Now we can spilt the dataset in training and testing by the identification tag.

```
# split the dataset
df_train = df_merged[df_merged["identification"]=="train"]
df_test = df_merged[df_merged["identification"]=="test"]
print(df_train.shape)
print(df_test.shape)
```

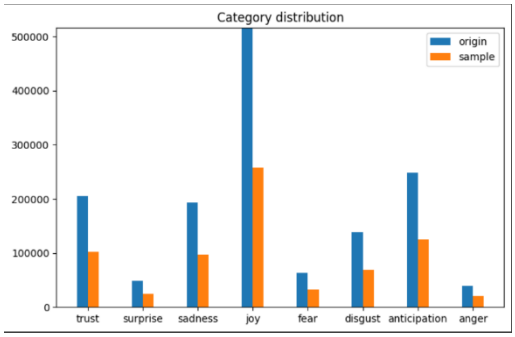
And apply emotion tag on the training set by merging dataframe of df_train and emotion

```
df_train = df_train.merge(emotion,how="left",on="tweet_id")
```

Here, I sample 50% of the training data because it's too big for my computation resource.

```
#sample half of data
df_train = df_train.sample(frac=0.5, random_state=42)
```

Plot the distribution of all categories before and after sampling.



The hyperparameter I define for mechine learning(but only train for one epoch at the end)

```
# define the hyperparameters
lr = 3e-5
epochs = 3
batch_size = 64
```

There are 8 categories in total and I also use 2 dictionaries for the convertion between emotion(e.x. “fear”) and label(e.x. 3) since we need numeric labels for machine learning.

```
category = ["anger", "anticipation", "disgust", "fear", "sadness", "surprise", "trust", "joy"]
emotion_to_label = {c: i for i, c in enumerate(category)}
label_to_emotion = {i: c for i, c in enumerate(category)}
```

I load the bert-base-uncased tokenizer here for tokenization of text.

```
tokenizer = T.BertTokenizer.from_pretrained("google-bert/bert-base-uncased", cache_dir="./cache/")
```

Definition of dataset and collate function here.They’re help for machine learning by stack a batch of data and package them to pytensor type.Also, I do the tokenization during collate function.

Another worth noting thing is we only use three columns of the data.They’re tweet_id(identify a data point), text(token for input data), and emotion(label for prediction)

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

# define dataset for mechine learning
class Dataset(Dataset):
  def __init__(self, dataframe):
    self.dataframe = dataframe
```

```
8.
9.    def __len__(self):
10.     return len(self.dataframe)
11.
12.   def __getitem__(self, index):
13.     id = self.dataframe.iloc[index,0]
14.     x = self.dataframe.iloc[index,1]
15.     y = self.dataframe.iloc[index,2]
16.     return id, x, y
17.
18. # define collate function for mechine learning
19. def collate_fn(batch):
20.   id = [data[0] for data in batch]
21.   # tokenize and package into PyTorch
22.   sentence = [data[1] for data in batch]
23.   token = tokenizer(sentence, truncation=True, padding=True, return_tensors="pt").to(d
    evice)
24.   # convert emotions to labels here also package into PyTorch
25.   label = torch.tensor([emotion_to_label[data[2]] for data in batch]).to(device)
26.
27.   return id, token, label
28.
29. # build dataset of training set
30. ds_train = Dataset(df_train[["tweet_id","text","emotion"]])
31.
32. # build dataloader of training set
33. dl_train = DataLoader(ds_train, batch_size=batch_size, shuffle=True, collate_fn=collat
    e_fn)
```

Definition of the model, I use the bert-base-uncased pre-trained model for this task and add a linear layer for classification.

During forward,model take the tokens as input, and we use pooler_output to feed in linear layer then produce prediction(score for every categories)

```
1.  # construct the model
2.  class MyModel(torch.nn.Module):
```

```
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # define the modules used in my model
        # use pretrained BERT model here
        self.bert = T.BertModel.from_pretrained("google-bert/bert-base-uncased", cache_dir="./cache/")
        hidden_size = self.bert.config.hidden_size
        # apply a linear layer for classification
        self.classifier = torch.nn.Linear(hidden_size, len(category))

    def forward(self, **kwargs):
        # forward pass
        outputs = self.bert(input_ids = kwargs["input_ids"], attention_mask=kwargs["attention_mask"])
        # use pooled_output for feature of whole sentence instead of single word
        pooled_output = outputs.pooler_output

        # predictions
        classification_output = self.classifier(pooled_output)

        return classification_output

model = MyModel().to(device)
```

I use Adam as optimizer and cross entropy as loss function.

```
# define the optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# define the loss functions
criterion_classification = torch.nn.CrossEntropyLoss()  # for classification
```

It takes about 2 hours for training one epoch using sampled data(50% of training set).Sine colab has limited free GPU resource, final model on trained for one epoch.

For testing, I apply a dummy column for df_test dataframe to pass through the dataset and dataloader.

```
df_test["dummy"] = "fear"
```

Argmax() will find the index of max value in the prediction of all categories, and I convert them to origin emotion by label_to_emotion dictionary.Then write the result in csv file.

```
import csv

with open('output.csv', 'w', newline='') as csvfile:

  writer = csv.writer(csvfile)

  writer.writerow(["id","emotion"])

  pbar = tqdm(dl_test)
  eval_model.eval()
  with torch.no_grad():
    for ids, batch_tokens, batch_labels in pbar:
      # predict
      batch_preds = eval_model(**batch_tokens)

      # convert label to emotion
      label_preds = torch.argmax(batch_preds, dim=1)

      for item in zip(ids,label_preds):
        # write output csv file
        id = item[0]
        emotion = label_to_emotion[item[1].item()]

        writer.writerow([id,emotion])
```