In [1]:

```python
import re
import torch
import pickle
import numpy as np
import pandas as pd
from tqdm import tqdm
import torch.nn as nn
from transformers import BertModel
from transformers import BertTokenizer
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

In [3]:

```python
lr = 1e-3
seq_len = 20
dropout = 0.5
num_epochs = 10
label_col = "Product"
tokens_path = "Output/tokens.pkl"
labels_path = "Output/labels.pkl"
data_path = "Input/complaints.csv"
model_path = "Output/bert_pre_trained.pth"
text_col_name = "Consumer complaint narrative"
label_encoder_path = "Output/label_encoder.pkl"
product_map = {'Vehicle loan or lease': 'vehicle_loan',
               'Credit reporting, credit repair services, or other personal consumer reports': 'credit_report',
               'Credit card or prepaid card': 'card',
               'Money transfer, virtual currency, or money service': 'money_transfer',
               'virtual currency': 'money_transfer',
               'Mortgage': 'mortgage',
               'Payday loan, title loan, or personal loan': 'loan',
               'Debt collection': 'debt_collection',
               'Checking or savings account': 'savings_account',
               'Credit card': 'card',
               'Bank account or service': 'savings_account',
               'Credit reporting': 'credit_report',
               'Prepaid card': 'card',
               'Payday loan': 'loan',
               'Other financial service': 'others',
               'Virtual currency': 'money_transfer',
               'Student loan': 'loan',
               'Consumer Loan': 'loan',
               'Money transfers': 'money_transfer'}
```

In [4]:

```
def save_file(name, obj):
    """
    Function to save an object as pickle file
    """
    with open(name, 'wb') as f:
        pickle.dump(obj, f)


def load_file(name):
    """
    Function to load a pickle object
    """
    return pickle.load(open(name, "rb"))
```

Process text data

In [5]:

```
data = pd.read_csv(data_path)
```

In [6]:

```
data.dropna(subset=[text_col_name], inplace=True)
```

In [7]:

```
data.replace({label_col: product_map}, inplace=True)
```

Encode labels

In [8]:

```
label_encoder = LabelEncoder()
label_encoder.fit(data[label_col])
labels = label_encoder.transform(data[label_col])
```

In [9]:

```
save_file(labels_path, labels)
save_file(label_encoder_path, label_encoder)
```

Process the text column

In [10]:

```
input_text = list(data[text_col_name])
```

In [11]:

```
len(input_text)
```

Out[11]:

```
809343
```

Convert text to lower case

In [14]:

```
input_text = [i.lower() for i in tqdm(input_text)]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 8000/8000 [00:00<00:00, 423202.19it/s]
```

Remove punctuations except apostrophe

```
In [15]: input_text = [re.sub(r"[^\w\d'\s]+", " ", i)
                       for i in tqdm(input_text)]
```

```
100%|███████████████████████████████████████████████████████████████████████████| 8000/8000 [00:00<00:00, 16721.68it/s]
```

Remove digits

```
In [16]: input_text = [re.sub("\d+", "", i) for i in tqdm(input_text)]
```

```
100%|███████████████████████████████████████████████████████████████████████████| 8000/8000 [00:00<00:00, 25635.34it/s]
```

Remove more than one consecutive instance of 'x'

```
In [17]: input_text = [re.sub(r'[x]{2,}', "", i) for i in tqdm(input_text)]
```

```
100%|███████████████████████████████████████████████████████████████████████████| 8000/8000 [00:00<00:00, 31373.15it/s]
```

Remove multiple spaces with single space

```
In [18]: input_text = [re.sub(' +', ' ', i) for i in tqdm(input_text)]
```

```
100%|███████████████████████████████████████████████████████████████████████████| 8000/8000 [00:00<00:00, 10778.53it/s]
```

Tokenize the text

```
In [19]: tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
```

```
In [20]: input_text[0]
```

```
Out[20]: 'i contacted ally on friday after falling behind on payments due to being out of work for a short period of time due to an illness i chated with a representative after logging
into my account regarding my opitions to ensure i protect my credit and bring my account current \n\nshe advised me that before an extenstion could be done i had to make a pay
ment in the amount of i reviewed my finances as i am playing catch up on all my bills and made this payment on monday this rep advised me once this payment posts to my account
to contact ally back for an extention or to have a payment deffered to the end of my loan \n\nwith this in mind i contacted ally again today and chatted with i explained all o
f the above and the information i was provided when i chatted with the rep last week she asked several questions and advised me that a one or two month extension deffered paym
ent could be done however partial payment is needed what she advised me or there abouts would be due within days from me accepting the agreement and then the remaining bal of
or there abouts would be due in in my payments of per month would resume \n\nif this was the case i should have just been offered this when i just made my payment so that i co
uld catch up on my bills \n\nthis company was working with in new jersey which has since closed most likely due to illegal practices they changed my loan company to this compa
ny after i had signed paperwork for another kill you with interest rates and has never once considered refiancing my vechile for a lower interest rate due to the age of the ve
chile other companies will not take it and they do not work with you '
```

```
In [21]: sample_tokens = tokenizer(input_text[0], padding="max_length",
                                   max_length=seq_len, truncation=True,
                                   return_tensors="pt")
```

```
In [22]: sample_tokens
```

```
Out[22]: {'input_ids': tensor([[  101,   178, 12017, 11989,  1113,   175, 22977,  1183,  1170,  4058,
                   1481,  1113, 10772,  1496,  1106,  1217,  1149,  1104,  1250,   102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'at
tention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

In [23]:

```
sample_tokens["input_ids"]
```

Out[23]:

```
tensor([[  101,   178, 12017, 11989,  1113,   175, 22977,  1183,  1170,  4058,
          1481,  1113, 10772,  1496,  1106,  1217,  1149,  1104,  1250,   102]])
```

In [24]:

```
sample_tokens["attention_mask"]
```

Out[24]:

```
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

In [25]:

```
tokens = [tokenizer(i, padding="max_length", max_length=seq_len,
                    truncation=True, return_tensors="pt")
         for i in tqdm(input_text)]
```

```
100%|█████████████████████████████████████████████████████████████████████████████| 8000/8000 [00:56<00:00, 142.26it/s]
```

Save the tokens

In [26]:

```
save_file(tokens_path, tokens)
```

Create Bert model

In [27]:

```
class BertClassifier(nn.Module):

    def __init__(self, dropout, num_classes):
        super(BertClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-cased')
        for param in self.bert.parameters():
            param.required_grad = False
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(768, num_classes)
        self.activation = nn.ReLU()

    def forward(self, input_ids, attention_mask):
        _, bert_output = self.bert(input_ids=input_ids,
                                   attention_mask=attention_mask,
                                   return_dict=False)
        dropout_output = self.activation(self.dropout(bert_output))
        final_output = self.linear(dropout_output)
        return final_output
```

Create PyTorch Dataset

In [28]:

```
class TextDataset(torch.utils.data.Dataset):

    def __init__(self, tokens, labels):
        self.tokens = tokens
        self.labels = labels

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

    def __getitem__(self, idx):
        return self.labels[idx], self.tokens[idx]
```

Function to train the model

In [29]:

```
def train(train_loader, valid_loader, model, criterion, optimizer,
          device, num_epochs, model_path):
    """
    Function to train the model
    :param train_loader: Data loader for train dataset
    :param valid_loader: Data loader for validation dataset
    :param model: Model object
    :param criterion: Loss function
    :param optimizer: Optimizer
    :param device: CUDA or CPU
    :param num_epochs: Number of epochs
    :param model_path: Path to save the model
    """
    best_loss = 1e8
    for i in range(num_epochs):
        print(f"Epoch {i+1} of {num_epochs}")
        valid_loss, train_loss = [], []
        model.train()
        # Train loop
        for batch_labels, batch_data in tqdm(train_loader):
            input_ids = batch_data["input_ids"]
            attention_mask = batch_data["attention_mask"]
            # Move data to GPU if available
            batch_labels = batch_labels.to(device)
            input_ids = input_ids.to(device)
            attention_mask = attention_mask.to(device)
            input_ids = torch.squeeze(input_ids, 1)
            # Forward pass
            batch_output = model(input_ids, attention_mask)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            ###batch_labels = batch_labels.type(torch.LongTensor)
            loss = criterion(batch_output, batch_labels)
            train_loss.append(loss.item())
            optimizer.zero_grad()
            # Backward pass
            loss.backward()
            # Gradient update step
            optimizer.step()
        model.eval()
        # Validation loop
        for batch_labels, batch_data in tqdm(valid_loader):
            input_ids = batch_data["input_ids"]
            attention_mask = batch_data["attention_mask"]
            # Move data to GPU if available
            batch_labels = batch_labels.to(device)
            input_ids = input_ids.to(device)
            attention_mask = attention_mask.to(device)
            input_ids = torch.squeeze(input_ids, 1)
            # Forward pass
            batch_output = model(input_ids, attention_mask)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            ###batch_labels = batch_labels.type(torch.LongTensor)
            loss = criterion(batch_output, batch_labels)
            valid_loss.append(loss.item())
        t_loss = np.mean(train_loss)
        v_loss = np.mean(valid_loss)
        print(f"Train Loss: {t_loss}, Validation Loss: {v_loss}")
        if v_loss < best_loss:
            best_loss = v_loss
            # Save model if validation loss improves
            torch.save(model.state_dict(), model_path)
        print(f"Best Validation Loss: {best_loss}")
```

Function to test the model

In [30]:

```
def test(test_loader, model, criterion, device):
    """
    Function to test the model
    :param test_loader: Data loader for test dataset
    :param model: Model object
    :param criterion: Loss function
    :param device: CUDA or CPU
    """
    model.eval()
    test_loss = []
    test_accu = []
    for batch_labels, batch_data in tqdm(test_loader):
        input_ids = batch_data["input_ids"]
        attention_mask = batch_data["attention_mask"]
        # Move data to GPU if available
        batch_labels = batch_labels.to(device)
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)
        input_ids = torch.squeeze(input_ids, 1)
        # Forward pass
        batch_output = model(input_ids, attention_mask)
        batch_output = torch.squeeze(batch_output)
        # Calculate loss
        ###batch_labels = batch_labels.type(torch.LongTensor)
        loss = criterion(batch_output, batch_labels)
        test_loss.append(loss.item())
        batch_preds = torch.argmax(batch_output, axis=1)
        # Move predictions to CPU
        if torch.cuda.is_available():
            batch_labels = batch_labels.cpu()
            batch_preds = batch_preds.cpu()
        # Compute accuracy
        test_accu.append(accuracy_score(batch_labels.detach().
                                        numpy(),
                                        batch_preds.detach().
                                        numpy()))
    test_loss = np.mean(test_loss)
    test_accu = np.mean(test_accu)
    print(f"Test Loss: {test_loss}, Test Accuracy: {test_accu}")
```

Train Bert model

Load the files

In [31]:

```
tokens = load_file(tokens_path)
labels = load_file(labels_path)
label_encoder = load_file(label_encoder_path)
num_classes = len(label_encoder.classes_)
```

Split data into train, validation and test sets

In [33]:

```
X_train, X_test, y_train, y_test = train_test_split(tokens, labels,
                                                   test_size=0.2)
X_train, X_valid, y_train, y_valid = train_test_split(X_train,
                                                      y_train,
                                                     test_size=0.25)
```

Create PyTorch datasets

In [34]:

```
train_dataset = TextDataset(X_train, y_train)
valid_dataset = TextDataset(X_valid, y_valid)
test_dataset = TextDataset(X_test, y_test)
```

Create data loaders

In [35]:

```
train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=16,
                                           shuffle=True,
                                           drop_last=True)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                                           batch_size=16)
test_loader = torch.utils.data.DataLoader(test_dataset,
                                         batch_size=16)
```

Create model object

In [36]:

```
device = torch.device("cuda:0" if torch.cuda.is_available()
                     else "cpu")
```

In [37]:

```
model = BertClassifier(dropout, num_classes)
```

Define loss function and optimizer

In [38]:

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

Move the model to GPU if available

In [39]:

```
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
```

Training loop ¶

In [40]:

```
train(train_loader, valid_loader, model, criterion, optimizer,
     device, num_epochs, model_path)
```

```
  0%|                                                                                          | 0/300 [00:00<?, ?it/s]

Epoch 1 of 2

100%|████████████████████████████████████████████████████████████████████████████████| 300/300 [13:16<00:00,  2.66s/it]
100%|████████████████████████████████████████████████████████████████████████████████| 100/100 [00:53<00:00,  1.88it/s]

Train Loss: 1.7121455442905427, Validation Loss: 1.7283893287181855


  0%|                                                                                          | 0/300 [00:00<?, ?it/s]

Best Validation Loss: 1.7283893287181855
Epoch 2 of 2

100%|████████████████████████████████████████████████████████████████████████████████| 300/300 [13:54<00:00,  2.78s/it]
100%|████████████████████████████████████████████████████████████████████████████████| 100/100 [01:01<00:00,  1.62it/s]

Train Loss: 1.6853400252262751, Validation Loss: 1.681002470254898
Best Validation Loss: 1.681002470254898
```

Test the model

In [41]:

```
test(test_loader, model, criterion, device)
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 100/100 [00:57<00:00,  1.73it/s]

Test Loss: 1.6601403439044953, Test Accuracy: 0.453125
```

Predict on new text

In [42]:
```
input_text = '''I am a victim of Identity Theft & currently have an Experian account that
I can view my Experian Credit Report and getting notified when there is activity on
my Experian Credit Report. For the past 3 days I've spent a total of approximately 9
hours on the phone with Experian. Every time I call I get transferred repeatedly and
then my last transfer and automated message states to press 1 and leave a message and
someone would call me. Every time I press 1 I get an automatic message stating than you
before I even leave a message and get disconnected. I call Experian again, explain what
is happening and the process begins again with the same end result. I was trying to have
this issue attended and resolved informally but I give up after 9 hours. There are hard
hit inquiries on my Experian Credit Report that are fraud, I didn't authorize, or recall
and I respectfully request that Experian remove the hard hit inquiries immediately just
like they've done in the past when I was able to speak to a live Experian representative
in the United States. The following are the hard hit inquiries : BK OF XXXX XX/XX/XXXX
XXXX XXXX XXXX  XX/XX/XXXX XXXX  XXXX XXXX  XX/XX/XXXX XXXX  XX/XX/XXXX XXXX  XXXX
XX/XX/XXXX'''
```

In [43]:
```
input_text = input_text.lower()
input_text = re.sub(r"[^\w\d'\s]+", " ", input_text)
input_text = re.sub("\d+", "", input_text)
input_text = re.sub(r'[x]{2,}', "", input_text)
input_text = re.sub(' +', ' ', input_text)
```

In [44]:
```
tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
```

In [45]:
```
tokens = tokenizer(input_text, padding="max_length",
                 max_length=seq_len, truncation=True,
                 return_tensors="pt")
```

In [46]:
```
input_ids = tokens["input_ids"]
attention_mask = tokens["attention_mask"]
```

In [47]:
```
device = torch.device("cuda:0" if torch.cuda.is_available()
                     else "cpu")
```

In [48]:
```
input_ids = input_ids.to(device)
attention_mask = attention_mask.to(device)
```

In [49]:
```
input_ids = torch.squeeze(input_ids, 1)
```

In [50]:
```
label_encoder = load_file(label_encoder_path)
num_classes = len(label_encoder.classes_)
```

In [51]:

```
# Create model object
model = BertClassifier(dropout, num_classes)

# Load trained weights
model.load_state_dict(torch.load(model_path))

# Move the model to GPU if available
if torch.cuda.is_available():
    model = model.cuda()

# Forward pass
out = torch.squeeze(model(input_ids, attention_mask))

# Find predicted class
prediction = label_encoder.classes_[torch.argmax(out)]
print(f"Predicted Class: {prediction}")
```

```
Predicted Class: credit_report
```

In [ ]: