Project Overview

The world of Natural Language Processing (NLP) continues to evolve with advances in various neural network models, specifically Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models. These models, designed to handle sequential data like text, have brought about exciting developments in text classification. Let's dive into how RNNs and LSTMs are revolutionizing the way businesses classify customer sentiment and feedback.

Business Overview

One application of NLP that holds a central position is text classification, which involves assigning predefined categories to texts based on their content. Models like RNNs and LSTMs are incredibly effective at classifying text data due to their capabilities in understanding context and capturing long-term dependencies.

The rise of deep learning and pre-trained word vectors greatly expands text classification potential in various applications like sentiment analysis, document categorization, speech recognition, facial expression recognition, customer segmentation, and more. By harnessing the power of RNNs and LSTMs, businesses can gain valuable insights from extensive datasets and improve customer satisfaction and service quality.

Aim

The primary objective of this project is to perform text classification on the dataset. RNN and LSTM models will be used to categorize these complaints.

Tech Stack

- **Language:** Python
- **Libraries:** pandas, torch, nltk, numpy, pickle, re, tqdm, sklearn

What is a multiclass classification problem?

Multiclass classification problems are a common type of problem encountered in machine learning, where the goal is to classify input instances into one of several possible classes. Unlike binary classification, which involves distinguishing between two distinct classes, multiclass classification deals with three or more categories simultaneously. In this blog post, we will explore the concept of

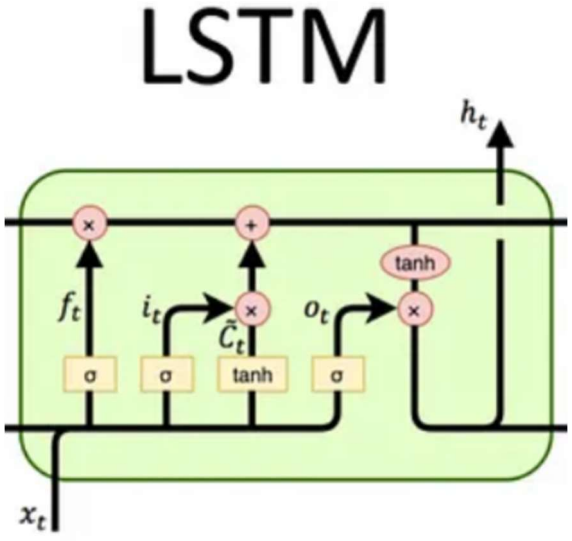
multiclass classification problems in detail and discuss how Long Short-Term Memory (LSTM) models can be used to tackle these challenges effectively.

What Are Multiclass Classification Problems?

Some examples of multiclass classification problems from various domains include sentiment analysis (classifying text documents as positive, negative, or neutral), image classification (categorizing images into distinct categories such as animals, objects, or landmarks), and customer segmentation (grouping customers based on demographics, behavior, or preferences). Other applications include document classification, music genre classification, object detection, and facial expression recognition.

Introduction to LSTM Models

LSTM is a type of recurrent neural network (RNN) architecture specially designed to overcome the limitations of traditional RNNs. These limitations include the vanishing gradient problem and the inability to capture long-term dependencies in sequential data.



The LSTM model addresses these issues by introducing memory cells and gating mechanisms. Memory cells facilitate the storage and updating of information over time. Gating mechanisms - input gate, forget gate, and output gate - control the flow of information within the LSTM model. This advanced architecture enables LSTM models to process sequential data effectively, making them suitable for tasks such as natural language processing (NLP), speech recognition, time-series analysis, and more.

Use Cases of LSTM Models

Prominent companies such as Apple, Google, Microsoft, and Facebook have leveraged LSTM models to enhance their products:

1. Apple: LSTM models are used to improve Siri's memory and contextual understanding capabilities. This allows for enhanced interactive features like "QuickType" in iMessage.

2. Google: LSTM models are employed in Google's speech recognition system, improving recognition rates by nearly 50%. LSTMs are also used in image caption generation, automatic email answering, Google Assistant Allo, and Google Translate.

3. Microsoft: Microsoft utilizes LSTM for enhanced speech recognition, creating photoreal talking heads and incorporating it into programming applications like code writing.

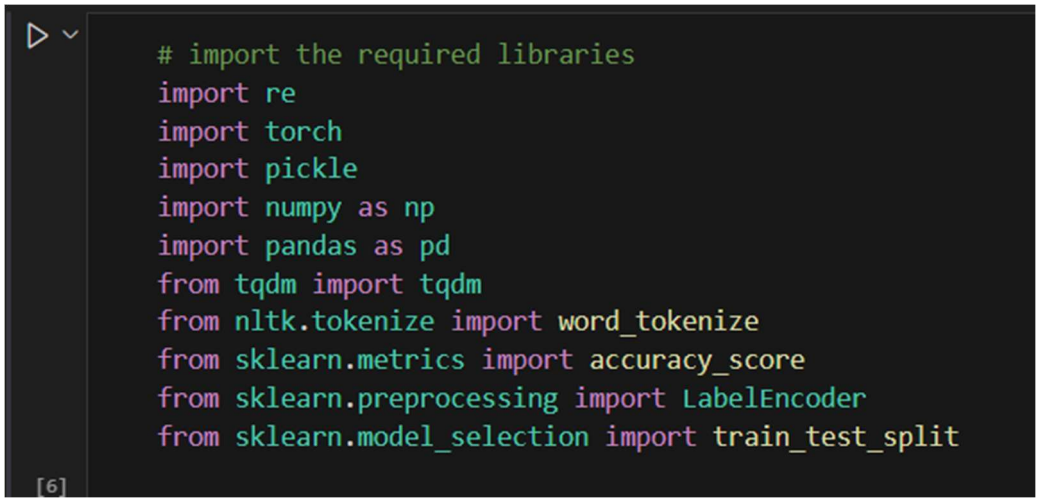
4. Facebook: The social media giant uses LSTM models for handling more than 4.5 billion translations per day across over 2,000 language pairs.

LSTM for Classification Python Project- Solution Approach

1. Understand the Multiclass Classification Problem.
2. Install necessary Python packages.
3. Import required libraries like pandas, torch, nltk, numpy, etc.
4. Process GloVe embeddings and text data while handling missing values and duplicate labels.

5. Perform data preprocessing tasks, such as tokenization and converting tokens to integer indices.
6. Build data loaders to obtain batches of rows with embeddings and corresponding labels.
7. Create RNN and LSTM models using PyTorch to address the vanishing gradient problem.
8. Train RNN and LSTM models using PyTorch.
9. Predict on test data and evaluate model performance.

Code



```
[6]
# import the required libraries
import re
import torch
import pickle
import numpy as np
import pandas as pd
from tqdm import tqdm
from nltk.tokenize import word_tokenize
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

```
lr = 0.0001
input_size = 50
num_epochs = 50
hidden_size = 50
label_col = "Product"
tokens_path = "Output/tokens.pkl"
labels_path = "Output/labels.pkl"
data_path = "Input/complaints.csv"
rnn_model_path = "Output/model_rnn.pth"
lstm_model_path = "Output/model_lstm.pth"
vocabulary_path = "Output/vocabulary.pkl"
embeddings_path = "Output/embeddings.pkl"
glove_vector_path = "Input/glove.6B.50d.txt"
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'}
```

```python
# define function for saving a file
def save_file(name, obj):
    """
    Function to save an object as pickle file
    """
    with open(name, 'wb') as f:
        pickle.dump(obj, f)

# define function for loading a file
def load_file(name):
    """
    Function to load a pickle object
    """
    return pickle.load(open(name, "rb"))
```

[4]

Process glove embeddings

```python
# open the glove embeddings file and read
with open(glove_vector_path, "rt") as f:
    emb = f.readlines()
```

[5]

400000 unique words are there in the embeddings

+ Code + Markdown

```python
# length of embeddings
len(emb)
```

Python

```
400000
```

Check the first record

```python
# check first record
emb[0]
```

Python

```
'the 0.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.49688 -0.17862 -0.00066023 -0.6566 0.27843 -0.14767 -0.55677 0.14658 -0.0095095 0.011658 0.10204 -0.12792 -0.8443 -0.12181 -0.01
```

```python
# split the first record and check for vocabulary
emb[0].split()[0]
```

Python

```
'the'
```

```
[8]
# split the first record and check for vocabulary
emb[0].split()[0]
```

```
'the'
```

```
[9]
# split the first record and check for embeddings
emb[0].split()[1:]
```

```
['0.418',
 '0.24968',
 '-0.41242',
 '0.1217',
 '0.34527',
 '-0.044457',
 '-0.49688',
 '-0.17862',
 '-0.00066023',
 '-0.6566',
 '0.27843',
 '-0.14767',
 '-0.55677',
 '0.14658',
 '-0.0095095',
 '0.011658',
 '0.10204',
 '-0.12792',
 '-0.8443',
 '-0.12181',
 '-0.016801',
 '-0.33279',
 '-0.1552',
 '-0.23131',
```

Separate embeddings and vocabulary

```
[10]
vocabulary, embeddings = [], []

for item in emb:
    vocabulary.append(item.split()[0])
    embeddings.append(item.split()[1:])
```

Convert embeddings to numpy float array

```
[11]
embeddings = np.array(embeddings, dtype=np.float32)
```

```
[12]
embeddings.shape
```

```
(400000, 50)
```

Add embeddings for padding and unknown items

```
[13]
vocabulary[:10]
```

```
['the', ',', '.', 'of', 'to', 'and', 'in', 'a', '"', "'s"]
```

```
[14]  vocabulary = ["<pad>", "<unk>"] + vocabulary
```

```
[15]  embeddings = np.vstack([np.ones(50, dtype=np.float32), np.mean(embeddings, axis=0),
                              embeddings])
```

```
[16]  print(len(vocabulary), embeddings.shape)
```

```
...   400002 (400002, 50)
```

Save embeddings and vocabulary

```
[17]  save_file(embeddings_path, embeddings)
      save_file(vocabulary_path, vocabulary)
```

Process text data

Read the data file

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

Drop rows where the text column is empty

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

Replace duplicate labels

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

Encode the label column and save the encoder and encoded labels

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

```
labels[0]
```

```
8
```

```
label_encoder.classes_
```

```
array(['card', 'credit_report', 'debt_collection', 'loan',
       'money_transfer', 'mortgage', 'others', 'savings_account',
       'vehicle_loan'], dtype=object)
```

```
data[label_col]
```

```
1             vehicle_loan
7            credit_report
8            credit_report
10           credit_report
13           credit_report
```

```
[25]
save_file(labels_path, labels)
save_file(label_encoder_path, label_encoder)
```

Process the text column

```
[26]
input_text = data[text_col_name]
```

Convert text to lower case

```
[27]
input_text = [i.lower() for i in tqdm(input_text)]
```

```
...  100%|██████████| 809343/809343 [00:03<00:00, 263904.71it/s]
```

Remove punctuations except apostrophe

```
[28]
input_text = [re.sub(r"[^\w\d'\s]+", " ", i) for i in tqdm(input_text)]
```

```
...  100%|██████████| 809343/809343 [00:34<00:00, 23770.26it/s]
```

Remove digits

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

```
100%|██████████| 809343/809343 [00:22<00:00, 36054.62it/s]
```

Remove more than one consecutive instance of 'x'

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

```
100%|██████████| 809343/809343 [00:18<00:00, 43254.56it/s]
```

Replace multiple spaces with single space

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

```
100%|██████████| 809343/809343 [00:48<00:00, 16776.33it/s]
```

Tokenize the text

```
2]  tokens = [word_tokenize(t) for t in tqdm(input_text)]
```

```
100%|██████████| 809343/809343 [20:52<00:00, 646.29it/s]
```

Take the first 20 tokens in each complaint text

```
3]  tokens = [i[:20] if len(i) > 19 else ['<pad>'] * (20 - len(i)) + i for i in tqdm(tokens)]
```

```
100%|██████████| 809343/809343 [18:40<00:00, 722.10it/s]
```

Convert tokens to integer indices from vocabulary

```
def token_index(tokens, vocabulary, missing='<unk>'):
    """
    :param tokens: List of word tokens
    :param vocabulary: All words in the embeddings
    :param missing: Token for words not present in the vocabulary
    :return: List of integers representing the word tokens
    """
    idx_token = []
    for text in tqdm(tokens):
        idx_text = []
        for token in text:
            if token in vocabulary:
                idx_text.append(vocabulary.index(token))
            else:
                idx_text.append(vocabulary.index(missing))
        idx_token.append(idx_text)
    return idx_token
```

[34]

```
tokens = token_index(tokens, vocabulary)
```

[35]

```
100%|██████████| 809343/809343 [2:52:22<00:00, 78.25it/s]
```

```
len(tokens)
```

```
809343
```

```
tokens[0]
```

```
[43,
 5909,
 3660,
 15,
 187,
 51,
 2333,
 563,
 15,
 3115,
 447,
 6,
 136,
 68,
 5,
 163,
 12,
 9,
 638,
 568]
```

```
data.head()
```

Python

	Date received	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	Company public response	Company	State	ZIP code	Tags	Consumer consent provided?	Submitted via	Date sent to company	Company response to consumer	Timely response?	Consu dispu
1	2019-11-01	vehicle_loan	Loan	Struggling to pay your loan	Denied request to lower payments	I contacted Ally on Friday XX/XX/XXXX after fa...	Company has responded to the consumer and the ...	ALLY FINANCIAL INC.	NJ	088XX	NaN	Consent provided	Web	2019-11-01	Closed with explanation	Yes	
7	2019-07-08	credit_report	Credit reporting	Problem with a credit reporting company's inve...	Their investigation did not fix an error on yo...	Hello This complaint is against the three cred...	Company has responded to the consumer and the ...	TRANSUNION INTERMEDIATE HOLDINGS, INC.	NY	109XX	NaN	Consent provided	Web	2019-07-08	Closed with explanation	Yes	
8	2020-06-10	credit_report	Credit reporting	Improper use of your report	Credit inquiries on your report that you don't...	I am a victim of Identity Theft & currently ha...	Company has responded to the consumer and the ...	Experian Information Solutions Inc.	MT	NaN	Servicemember	Consent provided	Web	2020-06-10	Closed with explanation	Yes	
10	2019-07-03	credit_report	Credit reporting	Incorrect information on your report	Account information incorrect	Two accounts are still on my credit history af...	Company has responded to the consumer and the ...	Experian Information Solutions Inc.	FL	328XX	NaN	Consent provided	Web	2019-07-03	Closed with non-monetary relief	Yes	

Create PyTorch Dataset

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

    def __init__(self, tokens, embeddings, labels):
        """
        :param tokens: List of word tokens
        :param embeddings: Word embeddings (from glove)
        :param labels: List of labels
        """
        self.tokens = tokens
        self.embeddings = embeddings
        self.labels = labels

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

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

Create Models

RNN Model

```
class RNNNetwork(torch.nn.Module):

    def __init__(self, input_size, hidden_size, num_classes):
        """
        :param input_size: Size of embedding
        :param hidden_size: Hidden vector size
        :param num_classes: Number of classes in the dataset
        """
        super(RNNNetwork, self).__init__()
        # RNN Layer
        self.rnn = torch.nn.RNN(input_size=input_size,
                                hidden_size=hidden_size,
                                batch_first=True)
        # Linear Layer
        self.linear = torch.nn.Linear(hidden_size, num_classes)

    def forward(self, input_data):
        _, hidden = self.rnn(input_data)
        output = self.linear(hidden)
        return output
```

LSTM Model

```
class LSTMNetwork(torch.nn.Module):

    def __init__(self, input_size, hidden_size, num_classes):
        """
        :param input_size: Size of embedding
        :param hidden_size: Hidden vector size
        :param num_classes: Number of classes in the dataset
        """
        super(LSTMNetwork, self).__init__()
        # LSTM Layer
        self.rnn = torch.nn.LSTM(input_size=input_size,
                                 hidden_size=hidden_size,
                                 batch_first=True)
        # Linear Layer
        self.linear = torch.nn.Linear(hidden_size, num_classes)

    def forward(self, input_data):
        _, (hidden, _) = self.rnn(input_data)
        output = self.linear(hidden[-1])
        return output
```

Define train function

```
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):
            # Move data to GPU if available
            batch_labels = batch_labels.to(device)
            batch_labels = batch_labels.type(torch.LongTensor)
            batch_data = batch_data.to(device)
            # Forward pass
            batch_output = model(batch_data)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            loss = criterion(batch_output, batch_labels)
            train_loss.append(loss.item())
```

```
            batch_output = model(batch_data)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            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):
            # Move data to GPU if available
            batch_labels = batch_labels.to(device)
            batch_labels = batch_labels.type(torch.LongTensor)
            batch_data = batch_data.to(device)
            # Forward pass
            batch_output = model(batch_data)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            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}")
```

Define test function

```
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):
        # Move data to device
        batch_labels = batch_labels.to(device)
        batch_labels = batch_labels.type(torch.LongTensor)
        batch_data = batch_data.to(device)
        # Forward pass
        batch_output = model(batch_data)
        batch_output = torch.squeeze(batch_output)
        # Calculate loss
        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()))
```

```
    # Calculate loss
    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 RNN Model

Load the files

```
[46]
    tokens = load_file(tokens_path)
    labels = load_file(labels_path)
    embeddings = load_file(embeddings_path)
    label_encoder = load_file(label_encoder_path)
    num_classes = len(label_encoder.classes_)
```

Split data into train, validation and test sets

```
[47]
    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

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

Create data loaders

```
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)
```

Train RNN model

```
model = RNNNetwork(input_size, hidden_size, num_classes)
```

Define loss function and optimizer

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Training loop

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

```
  0%|          | 0/30350 [00:00<?, ?it/s]
Epoch 1 of 50
100%|██████████| 30350/30350 [02:13<00:00, 227.67it/s]
100%|██████████| 10117/10117 [00:17<00:00, 570.33it/s]
  0%|          | 14/30350 [00:00<03:41, 137.18it/s]
Train Loss: 1.3165217890986893, Validation Loss: 1.1515936910652462
Best Validation Loss: 1.1515936910652462
Epoch 2 of 50
100%|██████████| 30350/30350 [02:09<00:00, 234.65it/s]
100%|██████████| 10117/10117 [00:17<00:00, 572.80it/s]
  0%|          | 16/30350 [00:00<03:09, 159.91it/s]
Train Loss: 1.0885461262261258, Validation Loss: 1.0318552850616465
Best Validation Loss: 1.0318552850616465
Epoch 3 of 50
100%|██████████| 30350/30350 [02:08<00:00, 237.09it/s]
100%|██████████| 10117/10117 [00:17<00:00, 587.58it/s]
  0%|          | 27/30350 [00:00<03:25, 184.65it/s]
```

```
Epoch 49 of 50
100%|██████████| 30350/30350 [01:52<00:00, 270.83it/s]
100%|██████████| 10117/10117 [00:15<00:00, 650.74it/s]
  0%|          | 17/30350 [00:00<03:01, 167.37it/s]
Train Loss: 0.775322395792903, Validation Loss: 0.7769622505712712
Best Validation Loss: 0.7757040077323875
Epoch 50 of 50
100%|██████████| 30350/30350 [01:50<00:00, 275.40it/s]
100%|██████████| 10117/10117 [00:15<00:00, 649.10it/s]
Train Loss: 0.7750829347430481, Validation Loss: 0.77966645701481
Best Validation Loss: 0.7757040077323875
```

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

```
100%|██████████| 10117/10117 [00:17<00:00, 574.71it/s]
Test Loss: 0.7853627795005748, Test Accuracy: 0.7361638065404004
```

Train LSTM Model

```
model = LSTMNetwork(input_size, hidden_size, num_classes)
```

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

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

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

```
Epoch 48 of 50
100%|██████████| 30350/30350 [05:32<00:00, 91.39it/s]
100%|██████████| 10117/10117 [00:41<00:00, 242.26it/s]
  0%|          | 4/30350 [00:00<14:13, 35.54it/s]
Train Loss: 0.6460606603364919, Validation Loss: 0.6751768548714322
Best Validation Loss: 0.6733405589122092
Epoch 49 of 50
100%|██████████| 30350/30350 [05:07<00:00, 98.72it/s]
100%|██████████| 10117/10117 [00:45<00:00, 220.00it/s]
  0%|          | 6/30350 [00:00<08:38, 58.53it/s]
Train Loss: 0.6453242170360228, Validation Loss: 0.6768645386698454
Best Validation Loss: 0.6733405589122092
Epoch 50 of 50
100%|██████████| 30350/30350 [05:23<00:00, 93.94it/s]
100%|██████████| 10117/10117 [00:52<00:00, 192.34it/s]
Train Loss: 0.6445044340511179, Validation Loss: 0.6743432611611458
Best Validation Loss: 0.6733405589122092
```

Predict on new text

```
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'''
```

Process input text

```
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)
tokens = word_tokenize(input_text)
```

Add padding if the length of tokens is less than 20

```
tokens = ['<pad>']*(20-len(tokens))+tokens
```

Tokenize the input text

```
idx_token = []
for token in tokens:
    if token in vocabulary:
        idx_token.append(vocabulary.index(token))
    else:
        idx_token.append(vocabulary.index('<unk>'))
```

Get embeddings for tokens

```
token_emb = embeddings[idx_token,:]
```

Convert to torch tensor

```
inp = torch.from_numpy(token_emb)
```

Move the tensor to GPU if available

```
inp = inp.to(device)
```

Create a batch of one record

```
inp = torch.unsqueeze(inp, 0)
```

RNN prediction

```
# Create model object
model = RNNNetwork(input_size, hidden_size, num_classes)

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

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

# Forward pass
out = torch.squeeze(model(inp))

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

```
Predicted  Class: credit_report
```

LSTM prediction

```
# Create model object
model = LSTMNetwork(input_size, hidden_size, num_classes)

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

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

# Forward pass
out = torch.squeeze(model(inp))

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

```
Predicted  Class: credit_report
```

Conclusion

In conclusion, the implementation of RNN and LSTM-based multi-class text classification models has proven to be a significant step forward in automating the process of categorizing consumer complaints about various financial products and services. These deep learning models have the capability to understand the context and semantics of text data, providing a more accurate and efficient way to classify complaints into categories such as 'card', 'credit_report', 'debt_collection', 'loan', 'money_transfer', 'mortgage', 'others', 'savings_account', and 'vehicle_loan'.

By leveraging the power of natural language processing (NLP), FinText has successfully utilized this technology to classify consumer complaints. In doing so, finance companies are able to ensure that complaints are directed to the appropriate departments or entities, thereby facilitating a quicker and more effective resolution.