Configuration Manual

MSc Research Project
Data Analytics

Pratheep Kumar Baskaran
Student ID: 22142878

School of Computing
National College of Ireland

Supervisor: Prof. Aaloka Anant

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pratheep Kumar Baskaran
Student ID:	22142878
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Aaloka Anant
Submission Due Date:	31st January 2024
Project Title:	Configuration Manual
Word Count:	1292
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pratheep Kumar Baskaran
22142878

1 Introduction

This configuration manual provides a concise yet comprehensive guide for implementing the 'Enhancing Customer Complaint Classification in Banking: A Deep Learning and Natural Language Processing Approach'. It provides prerequisites and step-by-step explanation. Section 2 details hardware and software specification used for NLP and deep learning. Section 3 covers ethical data acquisition and compliance. Finally, Section 4 describes the project's innovative methodologies and execution protocols.

1.1 Project Objective

This project creates a system that combines basic keyword matching with deep learning, to improve how banks classify customer complaints. Also, this includes a comparison of this hybrid approach with a deep learning method to understand which is more effective in a banking context. In addition, develop a GUI with best performing model to help banks manage customer complaints more efficiently.

2 Configuration setup

Configuration Requirements For this research project, specific hardware and software setups are essential. To effectively train our model, which processes substantial text files, we require the power of a GPU. I utilized Google Colab's GPU capabilities for this purpose, ensuring efficient handling of the large-scale data processing involved.

2.1 Hardware Configuration

RAM	8GB Processor
Processor	i5 CPU
System	Windows 11 Operating system-64 Bit
GPU	Google Colab T4

2.2 Software Configuration

The project's software setup utilizes Google Colab and Jupyter Notebook for developing deep learning models. Python 3.6.3 is the chosen programming language for this research.

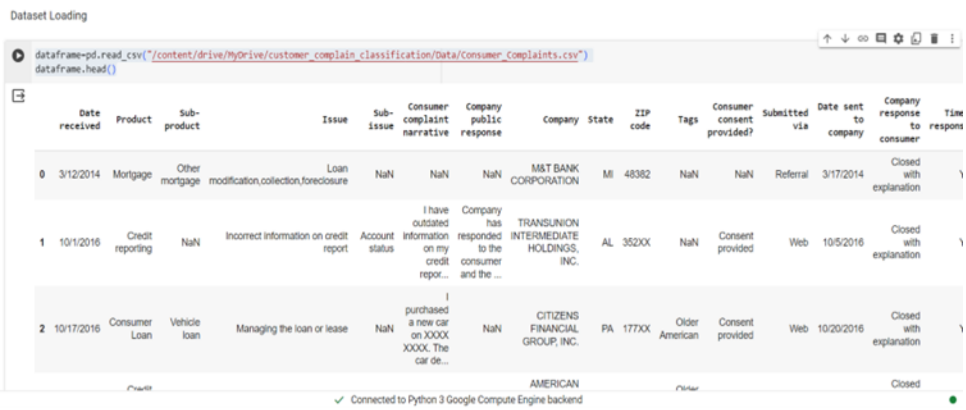
TensorFlow 2.5.0 is a key library for deep learning functionalities. Additional libraries include Keras for model architecture, Matplotlib 3.3.3 for visualization, and sklearn for machine learning tasks.

3 Data Gathering

In my research on classifying customer complaints, I have utilized a dataset from the Consumer Financial Protection Bureau, which contains complaints about various financial products. This dataset offers a detailed view into real consumer issues, with thousands of categorized entries showing authentic customer feedback. To prepare for analysis in Google Colab, I uploaded the dataset to Google Drive and then used specific commands in Colab for data access and preprocessing, setting the stage for in-depth analysis and model development.

4 Data Transformation and Preprocessing

For data transformation, each complaint is stored in a designated dataset folder on the drive. I execute a specific code to mount this data, ensuring it's ready for processing. This step is crucial for accessing and transforming the data, paving the way for further analysis and model training.


Dataset Loading

```
dataframe=pd.read_csv("/content/drive/MyDrive/customer_complain_classification/Data/Consumer_Complaints.csv")
dataframe.head()
```

	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	Time respons
0	3/12/2014	Mortgage	Other mortgage	Loan modification,collection,foreclosure	NaN	NaN	NaN	M&T BANK CORPORATION	MI	48382	NaN	NaN	Referral	3/17/2014	Closed with explanation	Y
1	10/1/2016	Credit reporting	NaN	Incorrect information on credit report	Account status	I have outdated information on my credit repor...	Company has responded to the consumer and the ...	TRANSUNION INTERMEDIATE HOLDINGS, INC.	AL	352XX	NaN	Consent provided	Web	10/5/2016	Closed with explanation	Y
2	10/17/2016	Consumer Loan	Vehicle loan	Managing the loan or lease	NaN	I purchased a new car on XXXX XXXX. The car de...	NaN	CITIZENS FINANCIAL GROUP, INC.	PA	177XX	Older American	Consent provided	Web	10/20/2016	Closed with explanation	Y

Connected to Python 3 Google Compute Engine backend

Figure 1: Complaints dataset loaded

After loading the dataset, the cleaning phase begins by identifying and removing null values. I then streamline the dataset, retaining only the 'Consumer complaint narrative' and 'Product' columns, discarding all other unnecessary columns for focused analysis.

Next, I embarked on Exploratory Data Analysis (EDA) to gain deeper insights that guide the preprocessing steps. This involved examining patterns, trends, and anomalies within the 'Consumer complaint narrative' and 'Product' columns to inform the subsequent data processing strategies.

Following the initial EDA, I focused on identifying the list of unique products within the dataset. This step was crucial to understand the variety and scope of financial products mentioned in the complaints.

Additionally, I analyzed the distribution of these products, examining how frequently each product category appeared in the dataset, to gain a clearer picture of the data

```python
#taking only useful columns and remove all unnecessary columns
dataframe = dataframe[['Consumer complaint narrative', 'Product']]
dataframe
```

	Consumer complaint narrative	Product
0	NaN	Mortgage
1	I have outdated information on my credit repor...	Credit reporting
2	I purchased a new car on XXXX XXXX. The car de...	Consumer Loan
3	NaN	Credit card
4	NaN	Debt collection
...	...	...
903978	NaN	Debt collection
903979	I had an account with XXXX in XX/XX/XXXX this ...	Debt collection
903980	I was contacted on XX/XX/XXXX email by XXXX fr...	Mortgage
903981	NaN	Debt collection
903982	I had a debit that was included in my chapter ...	Debt collection

903983 rows × 2 columns

Figure 2: Removing of unnecessary columns

```python
# Get unique products
unique_products = dataframe['Product'].unique()

# Print the list of unique products
print("List of Unique Products:")
for product in unique_products:
    print(product)
```

```
List of Unique Products:
Credit reporting
Consumer Loan
Debt collection
Mortgage
Credit card
Other financial service
Bank account or service
Student loan
Money transfers
Payday loan
Prepaid card
Virtual currency
Credit reporting, credit repair services, or other personal consumer reports
Credit card or prepaid card
Checking or savings account
Payday loan, title loan, or personal loan
Vehicle loan or lease
Money transfer, virtual currency, or money service
```

Figure 3: Checked for Unique list of Products

landscape and consumer trends.

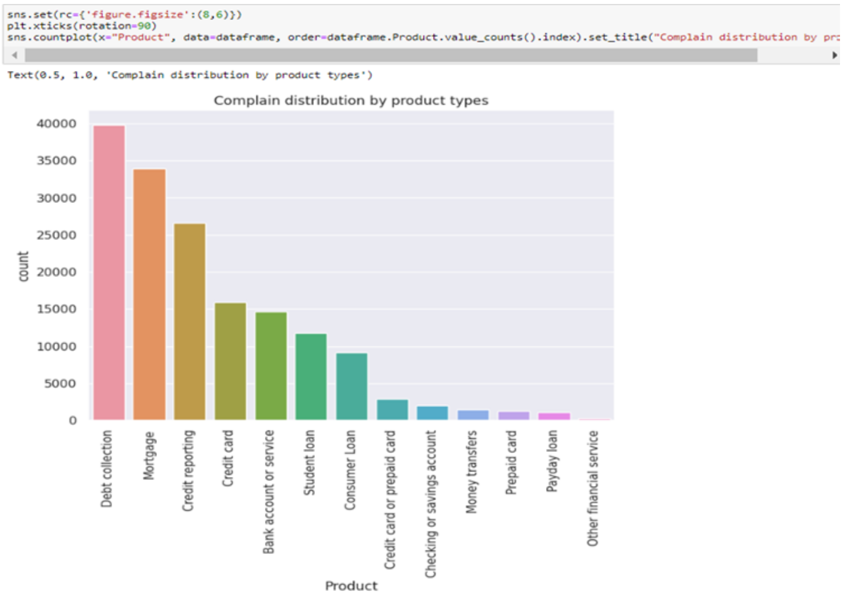


Figure 4: Distribution of Products

To clean the text in the 'Consumer complaint narrative' column of the dataset, I used a straightforward approach focusing on making the text more readable and relevant for analysis. Here's a simplified explanation of the process:

Normalizing Words: I started by simplifying each word to its base form. For instance, converting different tenses of a verb to its root form. This helps in treating different forms of the same word equally during analysis.

Cleaning Text: The next step involved removing unnecessary characters like punctuation marks and making all the text lowercase. This step ensures consistency across all complaints.

Breaking Down Sentences into Words: I then split the complaints into individual words, which helps in examining each word separately.

Removing Common Words: Which appear very frequently but don't add much value to the analysis. I removed these to focus on more meaningful words.

Reassembling the Words: Finally, I put all the words back together into clean, simplified sentences.

This process made the text in the complaints cleaner and more uniform, allowing for a more accurate and insightful analysis of customer feedback.

The main approach in processing the complaints involved using keyword matching. This method was crucial for filtering out misclassified complaints. I defined a set of keywords for each product. These keywords were carefully selected to accurately represent the essence of complaints related to each product. Then, using these keywords, I filtered the complaints, ensuring that each was correctly categorized according to the relevant product. The correctly classified complaints were then stored in a new CSV file. This file served as a cleaned and more organized dataset, ready for further in-depth analysis.

Text Vectorization with a Tokenizer: I converted the words in the complaints into numbers. This is like replacing each word with a unique code so that the computer can understand and analyze the text.

```
#text cleaning using nltk
_wnl = nltk.WordNetLemmatizer()

def normalize_word(w):
    return _wnl.lemmatize(w).lower()

def get_tokenized_lemmas(s):
    return [normalize_word(t) for t in nltk.word_tokenize(s)]

def clean(text):
    # Cleans a string: Lowercasing, trimming, removing non-alphanumeric
    text = text.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation),''))
    return " ".join(re.findall(r'\w+', text, flags=re.UNICODE)).lower()

def remove_stopwords(l):
    # Removes stopwords from a list of tokens
    return [w for w in l if w not in feature_extraction.text.ENGLISH_STOP_WORDS]
def join_tok(text):
    return " ".join(text).lower()

def process(texts):
    lst=[]
    for text in tqdm(texts):
        clean_text= clean(text)
        tok_text= get_tokenized_lemmas(clean_text)
        remov_stp= remove_stopwords(tok_text)
        lst.append(join_tok(remov_stp))
    return lst

dataframe['Consumer complaint narrative'] =  process(dataframe['Consumer complaint narrative'])

100%|██████████| 183081/183081 [04:59<00:00, 610.50it/s]
```

Figure 5: Text Cleaning

```
product_keywords = {
    'Mortgage': ['Mortgage', 'Foreclosure', 'Interest Rate', 'Bankruptcy', 'Fraud', 'Loan', 'Escrow', 'Down Payment', 'Legal', 'T
    'Credit card': ['Unauthorized Charges', 'Dispute', 'Billing Error', 'Contract', 'Cancellation', 'Chargeback', 'Terms and Cond
    'Credit reporting': ['Credit Reporting', 'Score', 'Credit', 'Inquiry', 'Dispute', 'History', 'Reporting', 'Creditworthiness',
    'Debt collection': ['Debt Collection', 'Debt', 'Reinsertion', 'HIPAA', 'Legal', 'Bounced', 'Credit', 'Collection', 'Cease',
    'Other financial service': ['Check', 'Money', 'Loan', 'Account', 'Company', 'Payment', 'Bank', 'Credit', 'Service', 'Debt'],
    'Consumer Loan': ['Payment', 'Loan', 'Credit', 'Car', 'Account', 'Vehicle', 'Company', 'Paid', 'Late', 'Report'],
    'Money transfers': ['Transfer', 'Transaction', 'Fund', 'Sent', 'Bank', 'PayPal', 'Western Union', 'Payment', 'Received', 'Ser
    'Credit card or prepaid card': ['Charge', 'Balance', 'Fee', 'Statement', 'Customer', 'Chase', 'Transaction', 'Limit', 'Rewar
    'Prepaid card': ['Fund', 'Transaction', 'Express', 'RushCard', 'Service', 'Customer', 'Balance', 'Charge', 'Reload', 'Withdr
    'Payday loan': ['Payday', 'Cash', 'Advance', 'Short-term', 'Lender', 'Finance', 'Emergency', 'Borrower', 'Interest', 'Credit
    'Student loan': ['Navient', 'Student', 'Repayment', 'Plan', 'Income', 'School', 'Education', 'Financial aid', 'Forbearance',
    'Checking or savings account': ['Checking', 'Savings', 'Deposit', 'Bank', 'Card', 'Transaction', 'Fee', 'Branch', 'Charge',
    'Bank account or service': ['Account', 'Bank', 'Check', 'Fee', 'Money', 'Card', 'Credit', 'Checking', 'Savings', 'Transactio
}


# Create a dictionary to store filtered and remaining complaints for each product
filtered_complaints = defaultdict(list)
remaining_complaints = defaultdict(list)

# Function to process a product
def process_product(product):
    # Get the keywords for the current product
    keywords = product_keywords[product]

    # Filter complaints based on keywords (case-insensitive)
    product_complaints = dataframe[dataframe['Product'] == product]['Consumer complaint narrative']
    filtered_product_complaints = product_complaints[product_complaints.str.contains('|'.join(keywords), case=False)].tolist()

    # Remove duplicates
    filtered_product_complaints = list(set(filtered_product_complaints))

    # Store filtered complaints in the dictionary
    filtered_complaints[product] = filtered_product_complaints

    # Store remaining complaints in the dictionary
    remaining_product_complaints = product_complaints[~product_complaints.isin(filtered_product_complaints)].tolist()
    remaining_complaints[product] = remaining_product_complaints

# Process each product
for product in product_keywords.keys():
    process_product(product)

# Store filtered complaints in a new CSV file
output_data_filtered = {'Product': [], 'Consumer complaint narrative': []}
for product, complaints in filtered_complaints.items():
    for complaint in complaints:
        output_data_filtered['Product'].append(product)
        output_data_filtered['Consumer complaint narrative'].append(complaint)

output_dataframe_filtered = pd.DataFrame(output_data_filtered)

# Save the DataFrame to CSV for filtered complaints
output_csv_path_filtered = '/content/drive/MyDrive/customer_complain_classification/filtered_complaints_all_products.csv'
```

Figure 6: Filtering using keyword Matching

```python
tok= Tokenizer()
tok.fit_on_texts(complaint_text)

global vocab_size
vocab_size= len(tok.word_index)+1
embed_len= 300
embedd_matrix= np.zeros((vocab_size,embed_len))
titles=tok.texts_to_sequences(complaint_text)
titles= pad_sequences(titles,maxlen=300,padding='post')

max_len=300
Y = dataframe['Product']
```

```python
with open('/content/drive/MyDrive/customer_complain_classification/tokenizer.pickle', 'wb') as handle:
  pickle.dump(tok, handle)
```

```python
class_labels = LabelBinarizer()
labels = class_labels.fit_transform(Y)
pickle.dump(class_labels,open('/content/drive/MyDrive/customer_complain_classification/label_transform.pkl', 'wb'))
```

```python
cls = len(class_labels.classes_)
print(class_labels.classes_)
```

```
['Bank account or service' 'Checking or savings account' 'Consumer Loan'
 'Credit card' 'Credit card or prepaid card' 'Credit reporting'
 'Debt collection' 'Money transfers' 'Mortgage' 'Other financial service'
 'Payday loan' 'Prepaid card' 'Student loan']
```

Figure 7: Text vectorization and Binarizer

Label Binarization for Products: For the different types of products mentioned in the complaints, I used a method to turn their names into a simple 0s and 1s format. This made easier for the computer to recognize and classify the product. After cleaning the data, I split it into three segments of 60,20, and 20 for training, testing and validation respectively to check how well the classification model works. This way, the model learns from a substantial portion of the data and gets thoroughly tested and validated.

```python
X_train, X_test, y_train, y_test = train_test_split( titles, labels, test_size=0.2, shuffle= True)
X_train, X_val, y_train, y_val = train_test_split( X_train, y_train, test_size=0.2, shuffle= True)
print("Training Data size: ", X_train.shape[0])
print("Testing data size: ", X_test.shape[0])
embed_len = 300
```

```
Training Data size:  117171
Testing data size:  36617
```

Figure 8: Splitting of Data

5 Implementation and Evaluation

5.1 Modelling on cleaned data without Keyword matching

I designed a layered model that first turns words into numbers, then uses filters to detect patterns in the text, and finally, it has decision-making layers that categorize customer complaints. The model is trained to improve accuracy over time. I measure its success by how accurately it classifies these complaints.

In my project, I also experimented with two advanced techniques: LSTM (Long Short-Term Memory) and Bi-LSTM (Bidirectional Long Short-Term Memory). While both methods are effective for understanding text data, Bi-LSTM performed better in the first experiment, showcasing its strength in capturing information from the complaints more comprehensively.

```
model= Sequential()
model.add(Embedding(vocab_size,embed_len,input_length=titles.shape[1],trainable=False))
model.add(Conv1D(32, 2, activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Flatten())
model.add(Dense(16,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(cls,activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 300)          23689800

 conv1d (Conv1D)             (None, 299, 32)           19232

 dense (Dense)               (None, 299, 32)           1056

 flatten (Flatten)           (None, 9568)              0

 dense_1 (Dense)             (None, 16)                153104

 dense_2 (Dense)             (None, 16)                272

 dense_3 (Dense)             (None, 13)                221

=================================================================
Total params: 23863685 (91.03 MB)
Trainable params: 173885 (679.24 KB)
Non-trainable params: 23689800 (90.37 MB)
_________________________________________________________________
```

Figure 9: Building Model

```
model= Sequential()
model.add(Embedding(vocab_size,embed_len,input_length=titles.shape[1],trainable=False))
model.add(Bidirectional(LSTM(256,return_sequences=True)))
model.add(Dense(256,activation='relu'))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(cls,activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 300, 300)          23689800

 bidirectional (Bidirection  (None, 300, 512)          1140736
 al)

 dense_8 (Dense)             (None, 300, 256)          131328

 flatten_2 (Flatten)         (None, 76800)             0

 dense_9 (Dense)             (None, 128)               9830528

 dense_10 (Dense)            (None, 64)                8256

 dense_11 (Dense)            (None, 13)                845

=================================================================
Total params: 34801493 (132.76 MB)
Trainable params: 11111693 (42.39 MB)
Non-trainable params: 23689800 (90.37 MB)
```

Figure 10: Building Bi-LSTM Model

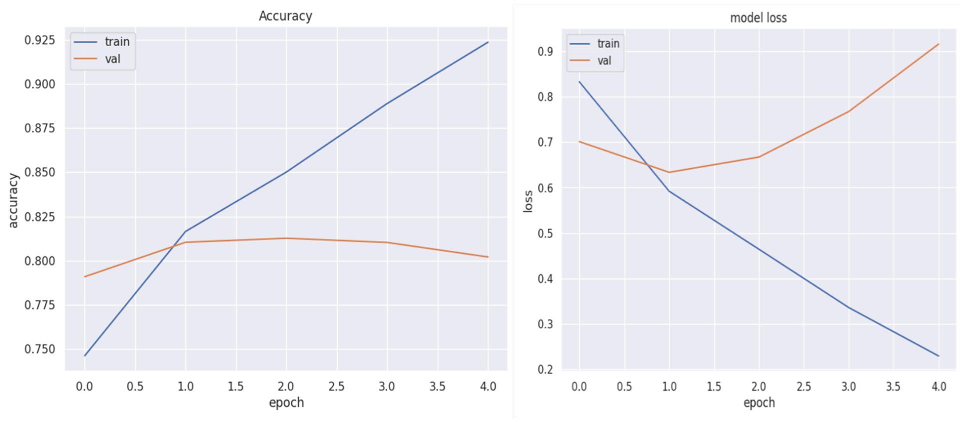


Figure 11: Accuracy and loss plot for Bi-LSTM without keyword matching

5.2 Modelling on cleaned data with Keyword matching

In my research, I employed CNN, LSTM and Bi-LSTM models on datasets that were specifically filtered using keyword matching. This approach enabled me to assess the impact of keyword filtering on the performance of these models, providing insights into their effectiveness with and without the use of keyword-focused data refinement.

```
from keras.optimizers import SGD
model= Sequential()
model.add(Embedding(vocab_size,embed_len,input_length=titles.shape[1],trainable=False))
model.add(Conv1D(32, 2, activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Flatten())
model.add(Dense(16,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(cls,activation='softmax'))

# model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics = ['accuracy'])
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 300, 300)          22805700

 conv1d_2 (Conv1D)           (None, 299, 32)           19232

 dense_12 (Dense)            (None, 299, 32)           1056

 flatten_3 (Flatten)         (None, 9568)              0

 dense_13 (Dense)            (None, 16)                153104

 dense_14 (Dense)            (None, 16)                272

 dense_15 (Dense)            (None, 13)                221

=================================================================
Total params: 22979585 (87.66 MB)
Trainable params: 173885 (679.24 KB)
Non-trainable params: 22805700 (87.00 MB)
```

Figure 12: CNN model Building with keyword matching

Then fitted the model, involved running the model through the training data (X_train and y_train) for 10 cycles, called epochs.

The bidirectional nature of Bi-LSTM allows it to process data from both forward and backward directions, offering a more comprehensive analysis of the text, especially

```
history = model.fit(X_train,y_train,epochs=10,batch_size=32,verbose=1,validation_data=(X_val,y_val), shuffle=True)
```

```
Epoch 1/10
3217/3217 [==============================] - 21s 6ms/step - loss: 2.0154 - accuracy: 0.2893 - val_loss: 1.9662 - val_accuracy:
0.3119
Epoch 2/10
3217/3217 [==============================] - 19s 6ms/step - loss: 1.8265 - accuracy: 0.3566 - val_loss: 1.6709 - val_accuracy:
0.3828
Epoch 3/10
3217/3217 [==============================] - 20s 6ms/step - loss: 1.5787 - accuracy: 0.4218 - val_loss: 1.9622 - val_accuracy:
0.3253
Epoch 4/10
3217/3217 [==============================] - 20s 6ms/step - loss: 1.4690 - accuracy: 0.4636 - val_loss: 1.3601 - val_accuracy:
0.5071
Epoch 5/10
3217/3217 [==============================] - 20s 6ms/step - loss: 1.2916 - accuracy: 0.5338 - val_loss: 1.2833 - val_accuracy:
0.5468
Epoch 6/10
3217/3217 [==============================] - 20s 6ms/step - loss: 1.1130 - accuracy: 0.6410 - val_loss: 1.0283 - val_accuracy:
0.6821
Epoch 7/10
3217/3217 [==============================] - 22s 7ms/step - loss: 0.9826 - accuracy: 0.6997 - val_loss: 1.1904 - val_accuracy:
0.6206
Epoch 8/10
3217/3217 [==============================] - 20s 6ms/step - loss: 0.9004 - accuracy: 0.7286 - val_loss: 0.8569 - val_accuracy:
0.7458
Epoch 9/10
3217/3217 [==============================] - 20s 6ms/step - loss: 0.8436 - accuracy: 0.7470 - val_loss: 0.8551 - val_accuracy:
0.7436
Epoch 10/10
3217/3217 [==============================] - 19s 6ms/step - loss: 0.8037 - accuracy: 0.7601 - val_loss: 0.7928 - val_accuracy:
0.7668
```

Figure 13: Model Fitting

```
#classification report
print(classification_report(y_test_new, prediction))
```

```
              precision    recall  f1-score   support

           0       0.63      0.78      0.70      2934
           1       0.00      0.00      0.00       408
           2       0.49      0.39      0.43      1821
           3       0.61      0.69      0.65      3112
           4       0.00      0.00      0.00       587
           5       0.81      0.79      0.80      5320
           6       0.75      0.88      0.81      7967
           7       0.00      0.00      0.00       306
           8       0.91      0.92      0.91      6771
           9       0.00      0.00      0.00        62
          10       0.00      0.00      0.00       229
          11       0.00      0.00      0.00       239
          12       0.88      0.82      0.85      2406

    accuracy                           0.76     32162
   macro avg       0.39      0.40      0.40     32162
weighted avg       0.72      0.76      0.74     32162
```

Figure 14: Classification report of CNN with keyword matching

beneficial in scenarios where keyword context plays a key role.

```
model= Sequential()
model.add(Embedding(vocab_size,embed_len,input_length=titles.shape[1],trainable=False))
model.add(Bidirectional(LSTM(256,return_sequences=True)))
model.add(Dense(256,activation='relu'))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(cls,activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 300, 300)          23689800

 bidirectional (Bidirection  (None, 300, 512)          1140736
 al)

 dense_8 (Dense)             (None, 300, 256)          131328

 flatten_2 (Flatten)         (None, 76800)             0

 dense_9 (Dense)             (None, 128)               9830528

 dense_10 (Dense)            (None, 64)                8256

 dense_11 (Dense)            (None, 13)                845

=================================================================
Total params: 34801493 (132.76 MB)
Trainable params: 11111693 (42.39 MB)
Non-trainable params: 23689800 (90.37 MB)
```

Figure 15: Model Building BI-LSTM with Keyword Matching

```
#classification report
print(classification_report(y_test_new, prediction))
```

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.78      2934
           1       0.63      0.18      0.28       408
           2       0.73      0.73      0.73      1821
           3       0.75      0.77      0.76      3112
           4       0.50      0.31      0.38       587
           5       0.87      0.89      0.88      5320
           6       0.88      0.89      0.88      7967
           7       0.66      0.65      0.66       306
           8       0.93      0.97      0.95      6771
           9       0.00      0.00      0.00        62
          10       0.65      0.45      0.54       229
          11       0.62      0.67      0.65       239
          12       0.94      0.93      0.94      2406

    accuracy                           0.85     32162
   macro avg       0.69      0.64      0.65     32162
weighted avg       0.85      0.85      0.85     32162
```

Figure 16: Classification report

To evaluate the performance of the Bi-LSTM model with keyword matching, I created plots for both validation loss and accuracy. These plots serve as visual tools to understand how well the model is learning and performing over time.

The best performing model in my project, the Bi-LSTM with keyword matching, was saved as an .h5 file. This file format is ideal for storing the complete architecture of the model along with its trained weights and biases, ensuring that all the learning it has undergone is preserved.

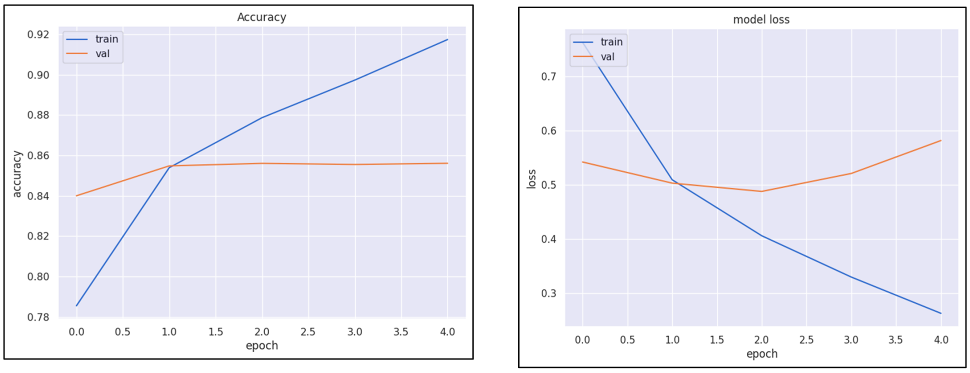


Figure 17: Accuracy and loss plot

```
model.save('/content/drive/MyDrive/customer_complain_classification/Models/bilstm_model.h5')
```

Figure 18: Classification report of CNN with keyword matching

6 Deployment

For deploying my project, I set up a simple website using Flask, a tool that helps create web applications. This website has a user-friendly interface where anyone can enter text, like a customer complaint. When they submit their text, my trained Bi-LSTM model, which I previously saved and loaded onto the website, analyzes it. The model, trained to understand and classify customer complaints, predicts the category of the complaint based on the text provided. The website then displays this predicted category to the user. This setup makes it easy for anyone, even without technical know-how, to see how the model works and to use it for classifying text in real-time.

To set up and run a Flask web application like the one described, which includes text processing and machine learning model deployment, several prerequisite libraries and tools are needed. Here's a brief overview:

Flask: The core library to create and manage the web application.

NumPy & Pandas: Essential for data manipulation and numerical operations.

Matplotlib & Seaborn: Used for data visualization, though they're more relevant during the model development phase than in the Flask app itself.

Pickle: For loading saved models and other objects like the tokenizer.

TensorFlow & Keras: Necessary for loading and running the deep learning model (in this case, the Bi-LSTM model).

Scikit-Learn (sklearn): For various machine learning utilities; here, it's specifically used for feature extraction from text.

NLTK (Natural Language Toolkit): A toolkit for natural language processing which provides functions for text cleaning and preprocessing.

WordNetLemmatizer (from NLTK): Used for lemmatization, which reduces words to their base or root form.

Regular Expressions (re): For text cleaning, such as removing non-alphanumeric characters.

String Library: For basic string operations, essential in the text cleaning process.

TQDM: A library for displaying progress bars, helpful during lengthy processing tasks.

Stopwords (from NLTK): To remove common words that usually don't contribute to the meaning of the text.

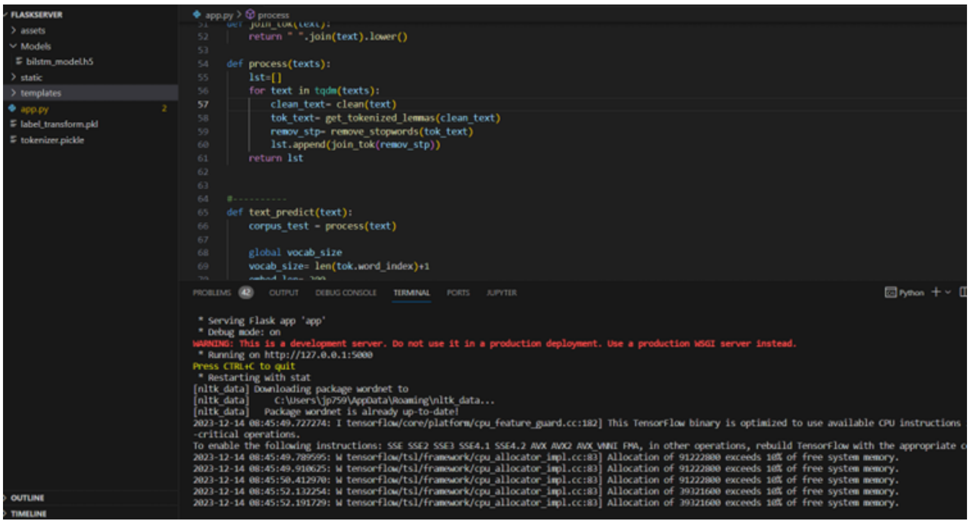


Figure 19: Classification report of CNN with keyword matching

To deploy Flask application and test the text classification locally, I followed these steps in Visual Studio Code. First, I made sure to have my Flask project folder open, which, in my case, was the "flaskserver" folder. Then, I used the VS Code terminal to navigate to the project folder. After confirming that I was inside the project directory, I ran my Flask application by executing the command "python app.py" in the terminal. Once the application was running, I received a URL, such as http://127.0.0.1:5000/, in the terminal. With this URL, I opened a web browser, pasted it into the address bar, and accessed the GUI for testing text classification. This straightforward process allowed me to deploy and interact with the application locally on my computer for testing purposes.

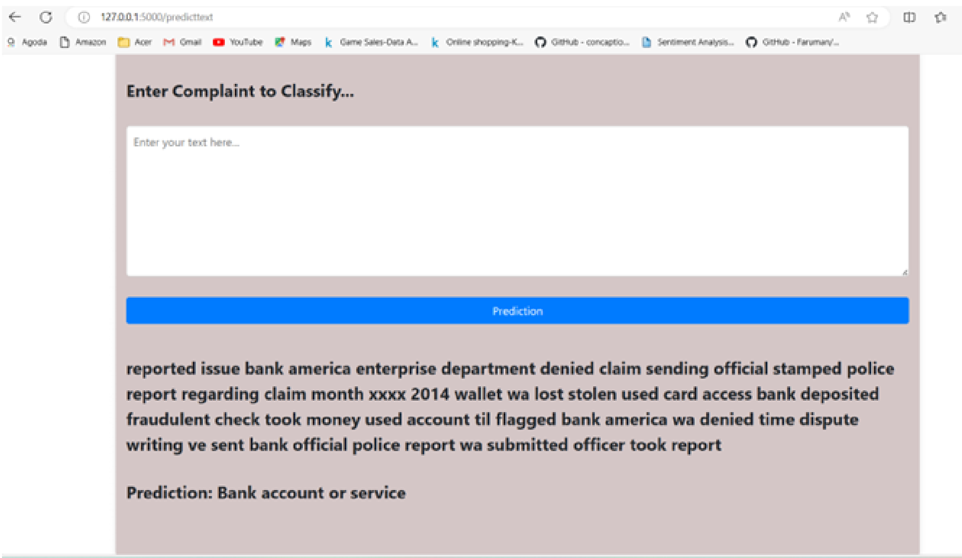


Figure 20: Classification report of CNN with keyword matching