National College of Ireland

Configuration Manual

MSc Research Project

Artificial Intelligence for Business

Usman Ghani

Student ID: x22186514

School of Computing

National College of Ireland

Supervisor: VICTOR DEL ROSAL

1. Introduction

Our primary goal is to provide a cutting-edge sentiment analysis framework tailored specifically for FlickCard, a platform for user-generated content. This system leverages advanced deep learning models, including LSTM networks, to accurately classify sentiments as positive, negative, or neutral. We created the notification alert system using the pushbullet for fast replies when we received negative reviews on our products from our clients. In this document, important code snippets are present that can be used to recreate the project code.

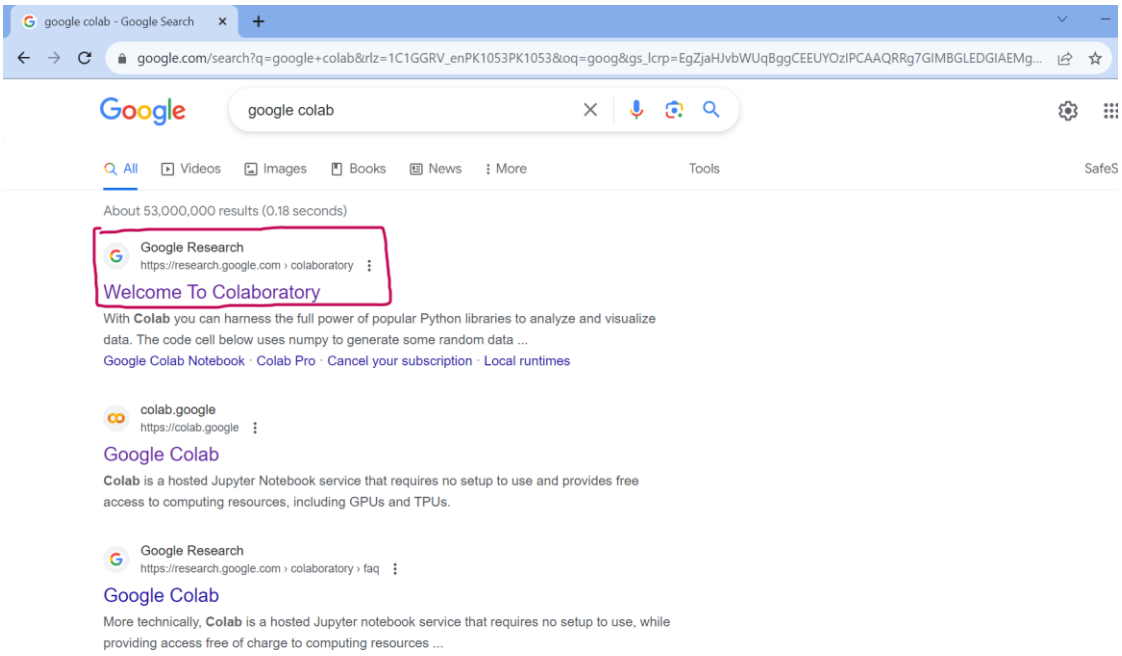
2. System Requirements

There are two methods to execute and set up our code. The first method involves installing the Anaconda Navigator on our local machine. Alternatively, if we prefer not to install the Anaconda Navigator locally, we can utilize Google Colab. Google Colab is an online service that offers computational power and fast, free services for running our machine learning tasks. In our case, we use the google colab.

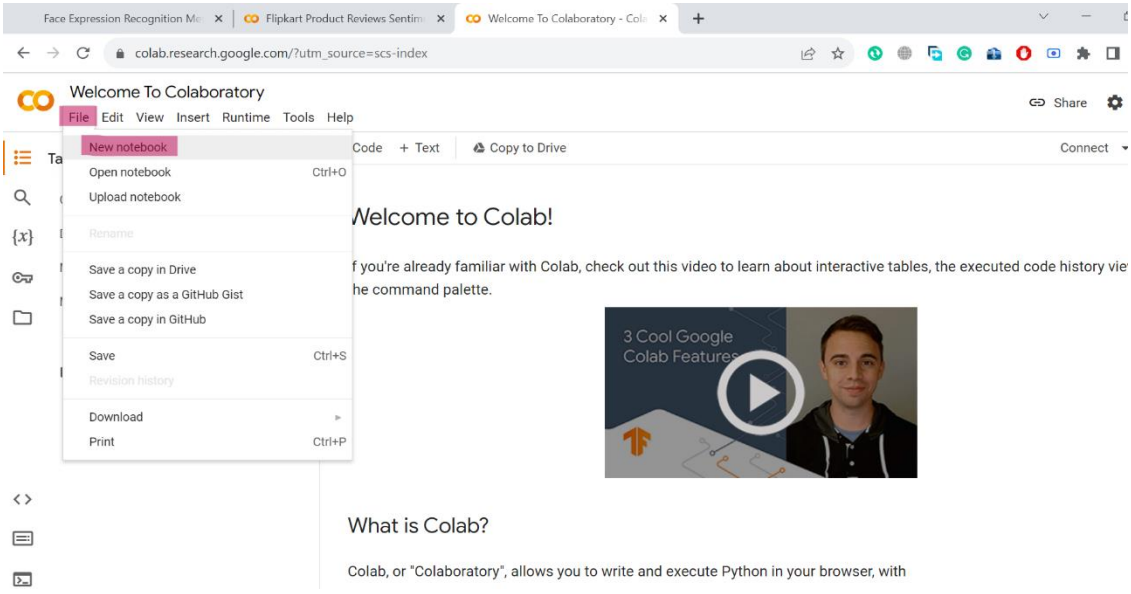
2.1 System Configuration

Google Colab setup

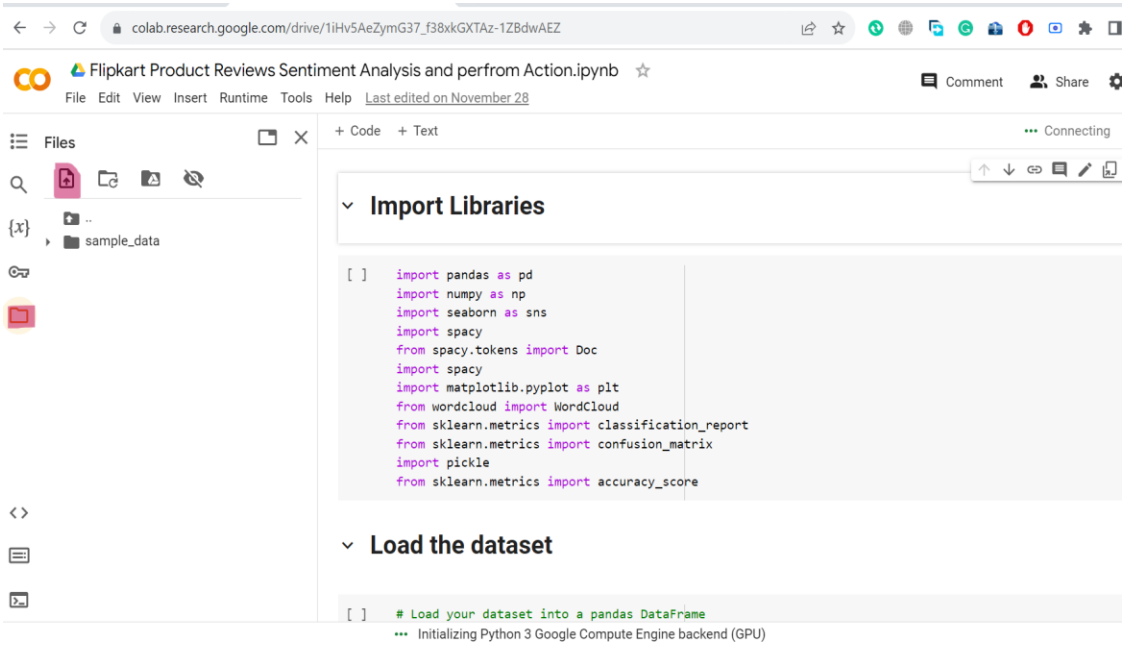
- Open Google and search for Google Colab, then click on the first link in the search results.

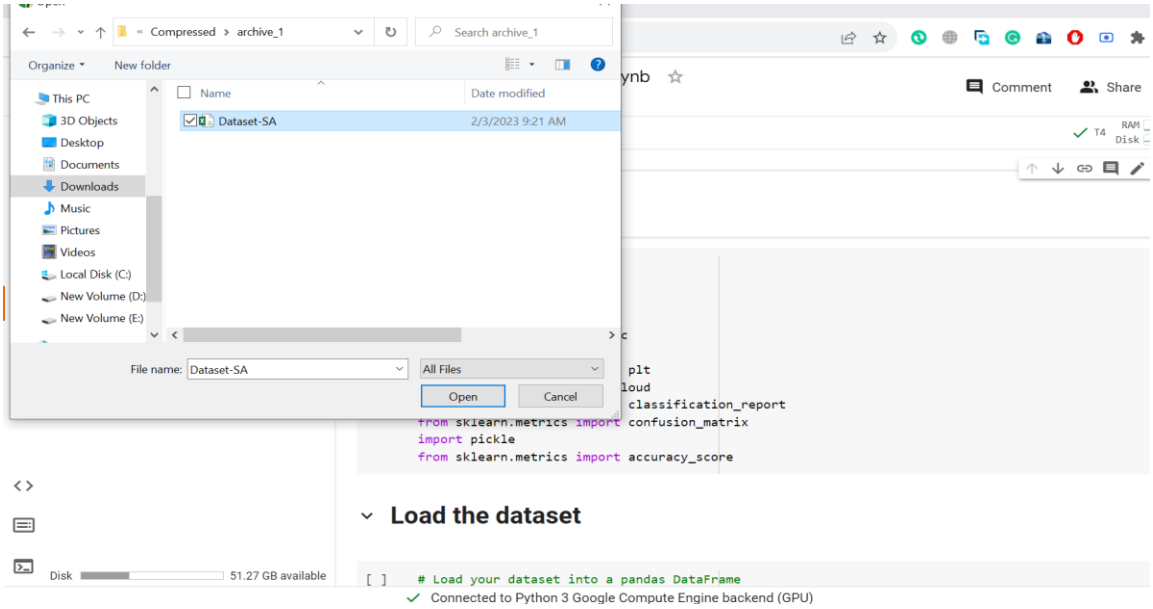


- After clicking on the URL, you will be directed to Google Colab. Here, locate the "File" option, and upon clicking, you will find the "New Notebook" option. Clicking on it will open a new notebook where you can write and execute your desired code.



- As we are utilizing online services, loading the dataset directly from our local machine isn't possible. Therefore, we need to upload our dataset. To do this, navigate to the "File" option, then select "Upload" and choose the data from your local folder to upload it to the drive.





3. Environment Setup

Google Colab typically comes pre-installed with many commonly used libraries. To use them, you just need to import them into your notebook. If any libraries are not already installed, you can easily install them using the pip command. In our specific case, we import the necessary libraries required for our work.

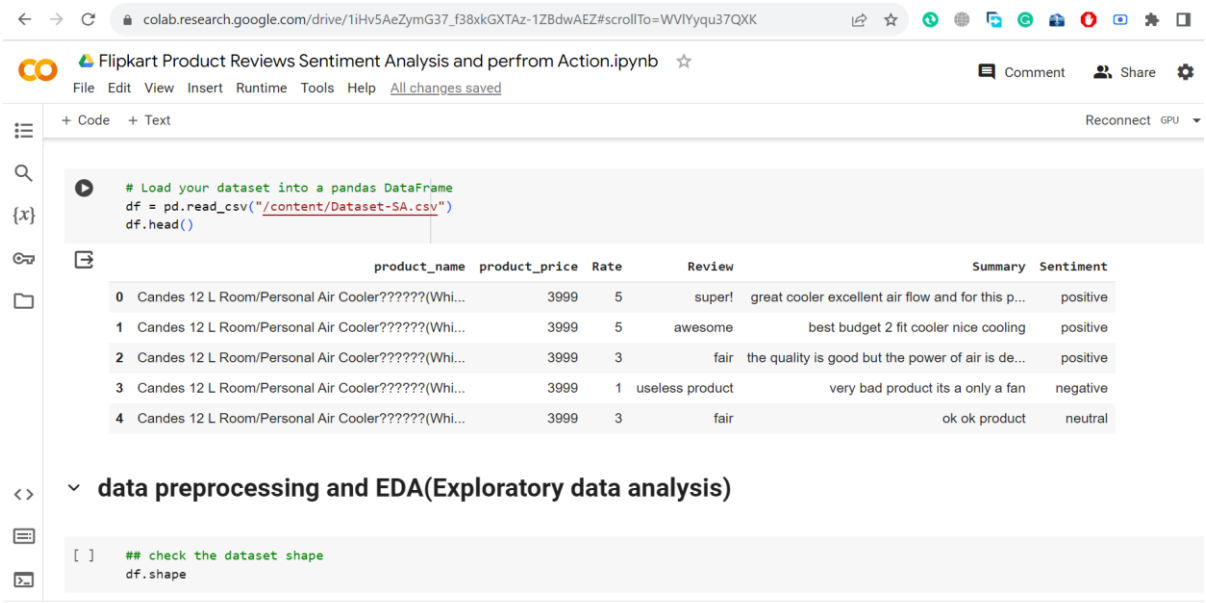
- Pandas
- Numpy
- Seaborn
- Spacy
- Matplotlib
- Wordcloud
- Tensorflow
- Sklearn
- Pickle

4. Implementation

4.1 Data Collection

- The dataset was obtained from Kaggle, and it's necessary to unzip the downloaded dataset before utilizing it. You can find the link to the dataset provided below.
  https://www.kaggle.com/datasets/niraliivaghani/flipkart-product-customer-reviews-dataset

- As discussed in the earlier section on setting up Google Colab, we loaded the dataset onto the drive. Now, we proceed to read the dataset using the pandas read command.



4.2 Data Preprocessing

- Check and remove missing values using the following code

```
[ ]    ## Check Missing values in our dataset
       df.isnull().sum()
```

```
product_name          0
product_price         0
Rate                  0
Review            24664
Summary              11
Sentiment             0
dtype: int64
```

```
[ ]    df.dropna(inplace=True)
```

- Data visualization is done using matplotlib, seaborn
- Implement a comprehensive cleaning process on the "Summary" column:
  - Lowercase the text.
  - Tokenize using SpaCy.
  - Remove stop words.
  - Apply lemmatization.
  - Save cleaned text in a new "cleaned_text" column.

```python
# function for text cleaning
def clean_text(text_column):
    cleaned_texts = []

    for text in text_column:
        # Step 1: Text Lowercasing
        text = text.lower()

        # Step 2: Tokenization
        doc = nlp(text)
        tokens = [token.text for token in doc]

        # Step 3: Stop Word Removal
        filtered_tokens = [token for token in tokens if not nlp.vocab[token].is_stop]

        # Step 4: Lemmatization
        lemmatized_tokens = [token.lemma_ for token in nlp(" ".join(filtered_tokens))]

        # Step 5: Reconstruct the cleaned text
        cleaned_text = " ".join(lemmatized_tokens)
        cleaned_texts.append(cleaned_text)

    return cleaned_texts
```

```python
# Clean the 'Summary review column' and create a new 'cleaned_text' column
df["cleaned_text"] = clean_text(df["Summary"])
```

4.3 Data Splitting

- Map sentiment labels to numerical values for LSTM model training.
- Split the dataset into input (X) and output (y) for model development
- Utilize scikit-learn's train_test_split function to divide the dataset into training and testing sets for robust model evaluation.

Create our independent and dependent columns

```python
# Define a dictionary to map sentiment labels to numerical values
sentiment_mapping = {"negative": 0, "neutral": 1, "positive": 2}

# Replace values in the 'Sentiment' column using the dictionary
df['Sentiment'] = df['Sentiment'].map(sentiment_mapping)
```

```python
X=df['cleaned_text']
y=df['Sentiment']
```

Splitting the Data:

```python
#Split  dataset into training and testing sets for model training and evaluation.
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
```

4.4 Tokenization and Padding

- Tokenize and pad the text data using Keras Tokenizer and pad_sequences to maintain consistent input lengths for the LSTM model.

```
#Tokenize the text and pad sequences to ensure consistent input lengths for the LSTM model.


# Create a tokenizer and fit it on the training data
tokenizer = Tokenizer(num_words=5000)  # You can adjust the vocabulary size as needed
tokenizer.fit_on_texts(X_train)

# Save the tokenizer to a file
with open('tokenizer.pkl', 'wb') as tokenizer_file:
    pickle.dump(tokenizer, tokenizer_file)

# Convert text to sequences and pad them
X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

max_sequence_length = 100  # Adjust the sequence length as needed
X_train_padded = pad_sequences(X_train_sequences, maxlen=max_sequence_length)
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_sequence_length)
```

4.5 Model Building

- Train the LSTM model on the prepared dataset, adjusting key parameters based on validation results.

```
model = Sequential()

# Embedding layer: Maps words to dense vectors
model.add(Embedding(input_dim=5000, output_dim=32, input_length=max_sequence_length))

# LSTM layer: Processes sequential data
model.add(LSTM(100, return_sequences=True))  # Use return_sequences=True to stack LSTM layers
model.add(LSTM(75, return_sequences=True))
model.add(LSTM(50))



# Dense layers: Fully connected layers for learning complex patterns
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))  # Dropout layer to reduce overfitting
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))


# Output layer: Use softmax for multi-class classification
model.add(Dense(3, activation='softmax'))  # 3 classes: negative, neutral, positive

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


# Train the model
model.fit(X_train_padded, y_train,
          validation_data=(X_test_padded, y_test),
          epochs=15,
          batch_size=128)
```

5. Model Evaluation

- Evaluate the trained LSTM model on the test set, considering metrics such as accuracy, loss, size, and latency.
- Generate a confusion matrix for a detailed performance assessment.

```
from sklearn.metrics import confusion_matrix

# Assuming you have a trained model 'model'
y_pred = model.predict(X_test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)


confusion = confusion_matrix(y_test, y_pred_classes)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(5,3))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

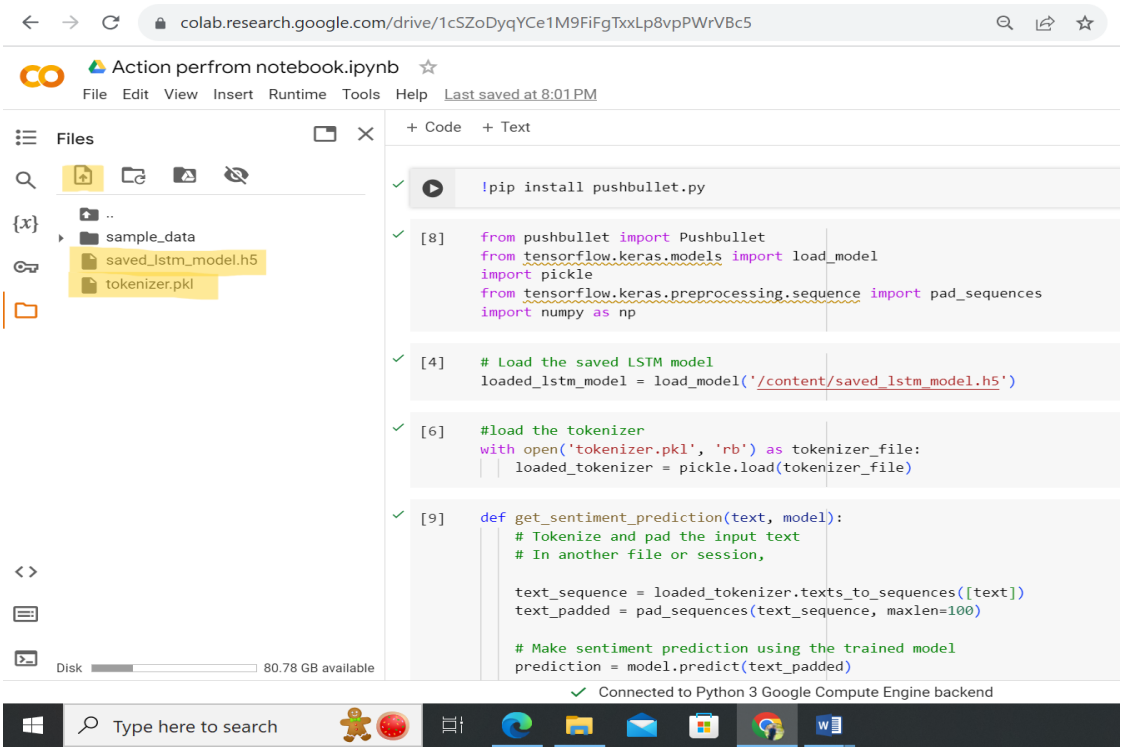
- Save the trained model for later use

```
# Save the LSTM model to a file
lstm_model_filename = 'saved_lstm_model.h5'
model.save(lstm_model_filename)
```

6. Negative Feedback Handling

6.1 Load The model

For negative feedback, we made a notification system. We save the model in the previous so we load that on the new notebook and also load the tokenizer pickle file.



6.2 Model Prediction

- Made a function in which we will pass the message it will return the prediction.

```
def get_sentiment_prediction(text, model):
    # Tokenize and pad the input text
    # In another file or session,

    text_sequence = loaded_tokenizer.texts_to_sequences([text])
    text_padded = pad_sequences(text_sequence, maxlen=100)

    # Make sentiment prediction using the trained model
    prediction = model.predict(text_padded)

    # Map the prediction to the corresponding sentiment class
    sentiment_classes = ["Negative", "Neutral", "Positive"]
    predicted_class = sentiment_classes[np.argmax(prediction)]

    return predicted_class
```

6.3 Notification System

- Implement a notification system for negative feedback using the Pushbullet library.
- Send notifications based on the model's predictions, enhancing user engagement and allowing timely responses to negative sentiments.

```python
# Example usage:
# Replace 'your_text_here' with the actual text you want to predict
input_text = input("Please enter the feedback:\n")

# Call the function to get sentiment prediction
predicted_sentiment = get_sentiment_prediction(input_text, loaded_lstm_model)

# Define your notification message based on the prediction
if predicted_sentiment == 'Negative':
    # Replace 'YOUR_API_KEY' with your actual Pushbullet API key
    pb = Pushbullet('o.9TzXjIdLL0vxGdFqaiODkvwZ8xPFH1t2')

    # Send a notification
    push = pb.push_note("Negative Sentiment Detected", "The sentiment analysis model has detected a negative sentiment. Please check the results.")
    print("Notification sent:", push)

# Print the prediction and notification details
print("Prediction:", predicted_sentiment)
```

```
Please enter the feedback:
```