|  |
| --- |
| IE4483 |
| Sentiment Analysis |
|  |

|  |
| --- |
| KOH JIAHUI (U2021624C), SHAO SIYING (U2021377H), PYAE SONE KHIN (U2021752A)  11-21-2023 |

Table of Content

[Chapter 1 Data Processing 2](#_Toc151483339)

[Data Preprocessing Procedures 2](#_Toc151483340)

[Chapter 2 Models 3](#_Toc151483341)

[2.1 BERT model (bert-base-uncased) 3](#_Toc151483342)

[2.1.1 Input and Output Dimensions 3](#_Toc151483343)

[2.1.2 Model Structure 3](#_Toc151483344)

[2.1.3 Loss Function 3](#_Toc151483345)

[2.1.4 Training Strategy 3](#_Toc151483346)

[2.1.5 Feature Format 4](#_Toc151483347)

[2.1.6 Code and Instruction 5](#_Toc151483348)

[2.1.7 Parameter and Setting Selection 7](#_Toc151483349)

[2.2 Convolutional Neural Network 8](#_Toc151483350)

[2.2.1 Input and Output Dimensions 8](#_Toc151483351)

[2.2.2 Model Structure 8](#_Toc151483352)

[2.2.3 Loss Function 8](#_Toc151483353)

[2.2.4 Training Strategy 8](#_Toc151483354)

[2.2.5 Feature Format 9](#_Toc151483355)

[2.2.6 Code and Instruction 9](#_Toc151483356)

[2.3 Recurrent Neural Network 12](#_Toc151483357)

[2.3.1 Input and Output Dimensions 12](#_Toc151483358)

[2.3.2 Model Structure 12](#_Toc151483359)

[2.3.3 Loss Function 13](#_Toc151483360)

[2.3.4 Training Strategy 13](#_Toc151483361)

[2.3.5 Feature Format 13](#_Toc151483362)

[2.3.6 Code and Instruction 14](#_Toc151483363)

[2.4 Model Selection and Result 15](#_Toc151483364)

[Chapter 3 Evaluation of the Model 16](#_Toc151483365)

[Impact on Resource Consumption and Accuracy 16](#_Toc151483366)

[Chapter 4 Hotel Reviews 17](#_Toc151483367)

[4.1.1 Dataset without rating score 17](#_Toc151483368)

[4.1.2 Dataset with inaccurate rating score 18](#_Toc151483369)

[Contribution 19](#_Toc151483370)

# Chapter 1 Data Processing

## Data Preprocessing Procedures

For this project, we have three models which have their own feature format. However, we have general data cleaning procedures before proceeding to train our models. The procedures are as follows.

* Lowercasing:

All text data is converted to lowercase to maintain consistency and prevent case-related variations.

* Removing Punctuation:

Special characters, punctuation marks, and symbols are removed from the text to focus on meaningful words. As removing them simplifies the data and reduce noise.

* Removing Numbers:

Numerical digits or symbols are excluded from the text data, as the focus is often on the emotional tone rather than numerical values. This might help streamline the text data for better model performance.

* Removing Stopwords:

Common, non-discriminative words (stopwords) are eliminated from the text, such as "the" "and" "is". With this, it can help in focusing on words that carry more sentiment.

* Removing Duplicated Rows:

Duplicate entries, if present, are removed to ensure data quality and to avoid bias that may arise from duplication. With this, we can maintain a clean and unbiased dataset.

# Chapter 2 Models

## 2.1 BERT model (bert-base-uncased)

BERT (Bidirectional Encoder Representations from Transformers) is an innovative natural language processing (NLP) model. It was pre-trained on a vast amount of unlabelled data, following a time-consuming and expensive process. It can capture bidirectional context information in input sequences due to its complex architecture [1].

#### 2.1.1 Input and Output Dimensions

For input, BERT takes in sequences of tokens, with each token being represented as a vector which then processed in the model. Additionally, attention masks are used to distinguish between padding tokens and actual input tokens. As for the output, it consists of a series of vectors of size H, each of which is a vector that matches an index-matched input token [2].

#### 2.1.2 Model Structure

BERT architecture starts with a Token Embedding Layer which is for converting input tokens into vector representation. This layer makes it easier for discrete tokens to be converted into continuous-valued vector. This allows the model to capture the meaning found in the input sequence. The model then includes a multi-layer transformer encoder, and each consist of multiple self-attention layers. This allows the model to focus on different parts of the input sequence and identify long-range dependencies. For the bert-base-uncased model, it consists of 12 layers, and each layer helps to extract and refine contextual information from the embedded token. Following that, we have an Outer Layer which is responsible for making the final predictions such as classifying the sentiment or answering a question [3] [4].

#### 2.1.3 Loss Function

We used Cross-Entropy Loss for the loss function which is a standard choice for binary classification tasks. This measures the dissimilarity between predicted outcomes and actual labels [5].

#### 2.1.4 Training Strategy

The training strategy for the model involves batches of data, each with a specified number of input sequences and corresponding labels. Firstly, we used AdamW optimizer which is an improved Adam optimizer with weight decay capabilities. When it is combined with specified learning rate, this optimizer helps to update the weight efficiently throughout the training iterations [6]. Secondly, the number of epochs (num\_train\_epochs) is set 3 in this project. With each epoch, the training dataset is fully explored which allows the model to gradually refine its parament and enhance its predictive capabilities. Moreover, we have evaluation phase after each training which allow us to gain an insights into the model performance and help find any overfitting or underfitting issues.

#### 2.1.5 Feature Format

As we will be using the Bert Base model, the chosen feature format is tokenized and encoded text data. This format is a standard practice in NLP and ensures that textual data can be effectively utilized by the model for sentiment classification. Below are the steps that the unprocessed data goes through to become a suitable format to be used in the model.

1. Text Cleaning:

This process is done using the procedures found in Chapter 1 Data Processing.

1. Tokenization:

Breaking down text into subword units is an important step in NLP. When using BERT, words are represented as subword pieces through subword tokenization. This process relies on a trained tokenizer which can be found in the Hugging Face Transformers library.

1. Padding and Truncation:

BERT models require input sequences with a fixed length. To achieve this, padding is used to extend shorter sequences and truncation is used to shorten longer sequences. This ensures that all input sequences have the same length.

1. Encoding:

Text data is converted into numerical values that can be understood by the BERT model. The encoding process involves:

* input\_ids: A series of integer IDs that represent the tokens.
* attention\_mask: A binary mask that distinguishes between actual tokens and padding tokens.
* token\_type\_ids: Utilized to differentiate between two distinct sentence inputs. It is not used for BERT but included for compatibility.

#### 2.1.6 Code and Instruction

I have written the instruction and explanation of some of the codes below.

# Install the required packages by running the following commands in Jupyter Notebook or Python script

!pip install transformers

!pip install accelerate>=0.20.1

# Copy and paste the following imports at the beginning of your script

import pandas as pd

import torch

from transformers import BertForSequenceClassification, AutoTokenizer, BertTokenizer, Trainer, TrainingArguments

from sklearn.model\_selection import train\_test\_split

from torch.utils.data import DataLoader, Dataset

from sklearn.metrics import classification\_report

# Replace the file path in the pd.read\_csv function with the path to your training data CSV file.

data = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/IE4483/data/processed\_train\_data.csv')

model\_name = "bert-base-uncased"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['reviews'], data['sentiments'], test\_size=0.2, random\_state=42)

X\_train = X\_train.astype(str).tolist()

X\_test = X\_test.astype(str).tolist()

y\_train = y\_train.tolist()

y\_test = y\_test.tolist()

# This is to define a custom dataset class for text data

class SentimentTextDataset(Dataset):

    def \_\_init\_\_(self, text, labels, tokenizer, max\_length):

        self.encodings = tokenizer(

            text,

            truncation=True,

            padding='max\_length',

            max\_length=max\_length,

            return\_tensors='pt',

        )

        self.labels = labels

    def \_\_len\_\_(self):

        return len(self.labels)

    def \_\_getitem\_\_(self, idx):

        item = {

            'input\_ids': self.encodings['input\_ids'][idx],

            'attention\_mask': self.encodings['attention\_mask'][idx],

            'token\_type\_ids': self.encodings['token\_type\_ids'][idx],  # Not used for BERT, but included for compatibility

            'labels': torch.tensor(self.labels[idx]),

        }

        return item

max\_length = 128

train\_dataset = SentimentTextDataset(X\_train, y\_train, tokenizer, max\_length)

eval\_dataset = SentimentTextDataset(X\_test, y\_test, tokenizer, max\_length)

# The training arguments

training\_args = TrainingArguments(

    output\_dir="./sentiment\_model",

    num\_train\_epochs=3,

    per\_device\_train\_batch\_size=8,

    per\_device\_eval\_batch\_size=8,

    evaluation\_strategy="epoch",

    save\_total\_limit=2,

)

model = BertForSequenceClassification.from\_pretrained(model\_name, num\_labels=2)  # num\_labels=2 for binary classification

trainer = Trainer(

    model=model,

    args=training\_args,

    train\_dataset=train\_dataset,

    eval\_dataset=eval\_dataset,

)

trainer.train()

# Save the fine-tuned model to the specified directory

trainer.save\_model("/content/drive/MyDrive/Colab Notebooks/IE4483/model/BERT")

# This is to evaluate the model and generate a classification report

# Replace the file path with the correct path to the trained model

model = BertForSequenceClassification.from\_pretrained("/content/drive/MyDrive/Colab Notebooks/IE4483/model/BERT")

eval\_dataloader = DataLoader(eval\_dataset, batch\_size=8)

model.eval()

predicted\_labels = []

true\_labels = []

with torch.no\_grad():

    for batch in eval\_dataloader:

        input\_ids = batch['input\_ids'].to(model.device)

        attention\_mask = batch['attention\_mask'].to(model.device)

        labels = batch['labels'].to(model.device)

        outputs = model(input\_ids, attention\_mask=attention\_mask)

        predicted\_labels.extend(outputs.logits.argmax(dim=1).cpu().numpy())

        true\_labels.extend(labels.cpu().numpy())

# This is to generate a classification report

report = classification\_report(true\_labels, predicted\_labels, target\_names=["Class 0", "Class 1"])

print(report)

### 2.1.7 Parameter and Setting Selection

Maximum Sequence Length (max\_length)

After weighing the trade-offs between computational efficiency and model complexity, we chose to set max\_length at 128. With this, we can maintain reasonable computing requirements while capturing context. While using a lesser length could mean losing important contextual information, using a greater length would need more computational resources.

Number of Training Epochs (num\_train\_epochs)

Initially, we started with three training epochs. By allowing the model to go through training iterations, this value helps the model identify the sentiment patterns that are underlying the data. We can identify the ideal balance between training time and model convergence by experimenting with the number of training epochs. The model achieve 92% accuracy so I believe the decision to use three training epochs is good.

Number of Labels (num\_labels)

For binary sentiment analysis, it is appropriate to set num\_labels to 2. This selection illustrates the task's binary nature where the objective is to categorize the data as a positive or negative sentiment.

## 2.2 Convolutional Neural Network

A convolutional neural network is a network architecture for deep learning that learns directly from data, it can uncover key information in both time series and image data. 1D CNNs are particularly effective for processing sequential data, such as text, because they can capture local patterns and dependencies in the input sequence. In the context of semantic analysis, the 1D CNN would operate on the sequential structure of the text, learning to recognize patterns of sentiment expressed through word combinations or phrases [7].

#### 2.2.1 Input and Output Dimensions

The input dimension is set to max\_length = 700, after trying a few more max\_length values, the model with 800 as input dimension gave the highest accuracy. The output dimension is set to be determined by the number of unique labels in the training data using num\_classes = len(set(y\_train));

#### 2.2.2 Model Structure

CNN contains several layers that serves different functions [8].

1. Embedding layer that converts sequences of integers to dense vectors
2. Convolutional layer (Conv1D) which learns features by sliding filter over input data. Most computations happen in the convolutional layer.
3. Pooling layer (GlobalMaxPooling1D) that reduces the dimension of the data from output of previous layer. In max pooling, the maximum value from each feature map is extracted.
4. Fully Connected layer (Dense) that learns complex pattern from output of previous layers using the ReLU activation function which helps to add non-linearity.
5. Dropout layer that helps prevent overfitting by randomly setting some input value to 0.
6. Output dense layer that uses SoftMax activation function to output a probability distribution over the possible classes.

#### 2.2.3 Loss Function

The loss function is computed using the Sparse Categorical Cross Entropy, a loss function that is commonly used for classification problem which labels are integers [9]. The output probability distribution obtained is compared against the ground truth and compute the loss. The overall loss is the average of individual loss of all examples.

#### 2.2.4 Training Strategy

Adam optimiser is used in this model training, iteratively adjusting the hyperparameter to increase the accuracy and lower the degree of error. Adam optimiser is a descent-based optimiser that adjust learning rate adaptively based on both the history of gradient calculated and the magnitude of recent gradients [10]. It computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

The data is trained in batches of 32 over an epoch of 10. Various combination of values that involve increasing or decreasing batch sizes and number of epochs are tested out and this gives the highest accuracy.

#### 2.2.5 Feature Format

Apart from the data cleaning adapted in chapter 1, the data is tokenised, from sequence of text to individual unit, which is essential for vocabulary creation and helps segment sentences into meaningful units for analysis. The data is then padded to the maximum length, calculated by finding the longest data. Padding is essential for all the sequences to have the same length. The data is then converted to list and is ready to use.

#### 2.2.6 Code and Instruction

import tensorflow as tf

import numpy as np

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout

# Replace the file path in the pd.read\_csv function with the path to your training data CSV file.

data = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/IE4483/data/processed\_train\_data.csv')

#CNN Model 89.4 (Can replace this part with Cross Vaildation)

max\_words = 700

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

train\_df['reviews'], train\_df['sentiments'], test\_size=0.2, random\_state=1

)

# Tokenization

tokenizer = Tokenizer(num\_words=max\_words, oov\_token='<OOV>')

tokenizer.fit\_on\_texts(X\_train)

X\_train\_sequences = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_sequences = tokenizer.texts\_to\_sequences(X\_test)

# Padding

max\_length = max(len(seq) for seq in X\_train\_sequences)

X\_train\_padded = sequence.pad\_sequences(X\_train\_sequences, maxlen=max\_length)

X\_test\_padded = sequence.pad\_sequences(X\_test\_sequences, maxlen=max\_length)

# Convert labels to NumPy arrays

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

# Create the CNN model

embedding\_dim = 50

num\_classes = len(set(y\_train))

model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=embedding\_dim, input\_length=max\_length))

model.add(Conv1D(128, 5, activation='relu'))

model.add(GlobalMaxPooling1D())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

batch\_size = 32

epochs = 10

model.fit(X\_train\_padded, y\_train, validation\_data=(X\_test\_padded, y\_test), batch\_size=batch\_size, epochs=epochs)

# Evaluate the model on test set

test\_loss, test\_acc = model.evaluate(X\_test\_padded, y\_test)

print(f'Test Accuracy: {test\_acc \* 100:.2f}%')

#End of CNN (Can replace this part with Cross Vaildation)

from sklearn.metrics import classification\_report

00# Make predictions on the test set

y\_pred\_probs = model.predict(X\_test\_padded)

# Convert predicted probabilities to classes

y\_pred = np.argmax(y\_pred\_probs, axis=1)

# Display the classification report

print(classification\_report(y\_test, y\_pred))

# Tokenize and pad sequences

X\_test\_sequences = tokenizer.texts\_to\_sequences(test\_df['reviews'])

X\_test\_padded = pad\_sequences(X\_test\_sequences, maxlen=max\_length)

# Make predictions

y\_test\_pred\_probs = model.predict(X\_test\_padded)

y\_test\_pred = np.argmax(y\_test\_pred\_probs, axis=1)

# Display the predictions

test\_df['predicted\_sentiment'] = y\_test\_pred

print(test\_df[['reviews', 'predicted\_sentiment']])

#Cross Vaildation 90.03

import numpy as np

import pandas as pd

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import accuracy\_score

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout

from tensorflow.keras.preprocessing.sequence import pad\_sequences

X\_data = train\_df['reviews']

y\_data = train\_df['sentiments']

# Reset the index of your DataFrame

df = pd.DataFrame({'X': X\_data, 'y': y\_data}).reset\_index(drop=True)

# Define the number of folds

n\_splits = 10

# Initialize StratifiedKFold for stratified sampling

kf = StratifiedKFold(n\_splits=n\_splits, shuffle=True, random\_state=42)

# Lists to store performance metrics

accuracy\_scores = []

# Hyperparameters

embedding\_dim = 200

max\_words = 700

max\_length = 100

num\_classes = 2

# Loop through each fold

for train\_index, val\_index in kf.split(df['X'], df['y']):

X\_train, X\_val = df.loc[train\_index, 'X'], df.loc[val\_index, 'X']

y\_train, y\_val = df.loc[train\_index, 'y'], df.loc[val\_index, 'y']

# Tokenize and pad sequences

tokenizer = Tokenizer(num\_words=max\_words, oov\_token='<OOV>')

tokenizer.fit\_on\_texts(X\_train)

X\_train\_sequences = tokenizer.texts\_to\_sequences(X\_train)

X\_val\_sequences = tokenizer.texts\_to\_sequences(X\_val)

X\_train\_padded = pad\_sequences(X\_train\_sequences, maxlen=max\_length)

X\_val\_padded = pad\_sequences(X\_val\_sequences, maxlen=max\_length)

# Your model creation and training code

model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=embedding\_dim, input\_length=max\_length))

model.add(Conv1D(128, 5, activation='relu'))

model.add(GlobalMaxPooling1D())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(X\_train\_padded, y\_train, epochs=10, batch\_size=32, verbose=0)

# Make predictions on the validation set

y\_val\_pred\_probs = model.predict(X\_val\_padded)

y\_val\_pred = np.argmax(y\_val\_pred\_probs, axis=1)

# Calculate and store accuracy for this fold

accuracy = accuracy\_score(y\_val, y\_val\_pred)

accuracy\_scores.append(accuracy)

# Calculate and print the average accuracy across all folds

average\_accuracy = np.mean(accuracy\_scores)

print(f'Average Accuracy: {average\_accuracy \* 100:.2f}%')

## 2.3 Recurrent Neural Network

Recurrent Neutral Network (RNN) is a neutral network that is useful for sequential data such as speech, text, and language. Due to having internal memory, RNN remembers the inputs they received and hence used them in their subsequent output predictions [11].

#### 2.3.1 Input and Output Dimensions

The input dimension is set to the max\_words which is 32. 32 gave the highest accuracy after comparing with 64, 128, 192 and 256. The output dimension is 1 as this is a binary sentiment analysis classification.

#### 2.3.2 Model Structure

Embedding layer: This layer is used to transform input text data into low-dimensional vector space. This helps to capture semantic relationships between words as they are being mapped out [12].

Long Short-Term Memory (LSTM) layer: It is used to learn long-range connections for text sequences and understanding sequential patterns within the textual data [13].

Dense layer: Used to make predictions based on the representations learnt from the previous layers.

#### 2.3.3 Loss Function

The loss function used is **'binary\_crossentropy'** which is designed specifically for binary classification. This means that the training process will have the aim of minimising binary crossentropy loss.

#### 2.3.4 Training Strategy

The training strategy for the model involves several steps. [14]

Step 1:

Data processing which includes tokenization, padding, and vocabulary creation.

Step 2:

Model creation including incorporating an embedded layer to convert integer-encoded tokens into meaningful word representations, choosing LTSM as a RNN architecture to capture sequential dependencies and configuring the output layer with a sigmoid actication for binary sentiment analysis.

Step 3:

Utilising ‘binary\_crossentropy’ as the loss function as this is a binary sentiment analysis and Adam optimizer.

Step 4:

Splitting train.csv into training and validation sets and monitor the accuracy of model during training.

Step 5:

Hyperparameter tuning to improve model performance which includes adjusting the input length, embedding size, number of epochs as well as the model architecture.

Step 6:

Conduct a comprehensive evaluation of the performance of the RNN model with a classification report.

#### 2.3.5 Feature Format

The data is first being cleaned with the processes stated in chapter 1. The data is then tokenized, converted into a sequence of numerical tokens. The data is then being padded to ensure that all sequences are of the same length. Furthermore, is it important to ensure that this number should be kept consistent with the model’s input length.

#### 2.3.6 Code and Instruction

# Install the required packages by running the following commands in Jupyter Notebook or python Script

!pip install transformers

!pip install accelerate>=0.20.1

# Copy and paste the following imports at the beginning of your script

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Embedding, LSTM, Dense

# Replace the file path in the pd.read\_csv function with the path to your training data CSV file.

data = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/IE4483/data/processed\_train\_data.csv')

# Putting reviews into the x axis, and sentiments into the y axis

x, y = (train\_df['reviews'].values, train\_df['sentiments'].values)

# Tokenization + padding of training data

tok = Tokenizer(lower=True)

tok.fit\_on\_texts(x)

x\_sequence = tok.texts\_to\_sequences(x)

x\_padding = pad\_sequences(x\_sequence, maxlen=32, padding='post')

# split data into training and validation set

x\_train, x\_val, y\_train, y\_val = train\_test\_split(x\_padding, y, test\_size=0.25, random\_state=1)

# Building RNN model

vocabulary\_size = len(tok.word\_counts.keys()) + 1

max\_words = 32 # To match the input\_length in the Embedding layer

embedding\_size = 200

model = Sequential()

model.add(Embedding(vocabulary\_size, embedding\_size, input\_length=max\_words))

model.add(LSTM(200))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the model

model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), batch\_size=32, epochs=3)

# Accuracy evaluation of model

accuracy = model.evaluate(x\_val, y\_val, verbose=1)

print("Test accuracy is: {:.2%}".format(accuracy[1]))

# Tokenization + padding of test data

tok = Tokenizer(lower=True)

tok.fit\_on\_texts(test\_df['reviews'].values)

x\_test\_sequence = tok.texts\_to\_sequences(test\_df['reviews'].values)

x\_test\_padding = pad\_sequences(x\_test\_sequence, maxlen=32, padding='post')

# Making predictions

y\_test\_prediction\_probability = model.predict(x\_test\_padding)

y\_test\_prediction = np.round(y\_test\_prediction\_probability).astype(int)

# Displaying the predictions

test\_df['predicted\_sentiment'] = y\_test\_prediction

print(test\_df[['reviews', 'predicted\_sentiment']])

# Generating the classification report

ground\_truth\_labels = test\_df['predicted\_sentiment']

predicted\_labels = y\_test\_prediction.astype(int)

class\_report = classification\_report(ground\_truth\_labels, predicted\_labels, target\_names=['0', '1'])

# Displaying the classification report

print(class\_report)

## 2.4 Model Selection and Result

After getting a result from CNN that has an accuracy of 89.4%, also tried out cross validation to see if will get a higher accuracy. It was 90.3 with 5 splits, 200 embedding dimension and 700 for max\_words.

RNN obtained an accuracy of 90.14% after fine-tuning and experimenting with different hyperparameters. The set of parameters that returned with the highest accuracy is with ‘input\_length’ = 32, ‘batch\_size’ = 32, ‘embedding\_size ‘= 200, ‘LTSM’ = 200, ‘epochs’ = 3.

Since Bert model has the highest accuracy of 92% among other models. We choose this model to predict the given test set.

# Chapter 3 Evaluation of the Model

To discuss the strength and weakness of the model, we will analyse two cases, one incorrectly classified and one correctly classified.

Incorrectly Classified Review: “love naughty monkey im happy shoes dont hurt feet”

The review has positive sentiment which are “love” and “happy”, but it was classified as negative. This might be because the model has difficulty understand the specific phrases like “naughty monkey”. Hence, the model might lack context or semantic understanding of certain expressions which lead to misinterpretation of sentiment.

Correctly Classified Review: “shades great buy fast shipping great price good quality”

The review has words such as “great buy”, “fast shipping” and “good quality” and the model correctly identifies this review as a positive sentiment. This demonstrate that the model can capture and combine a various positive phrase to capture positive sentiment.

In summary, the strength is that it is able to identify and group positive phrases which help with accurate positive sentiment classification. However, the weakness is that it has difficulty with some expression which lead to misclassification.

### Impact on Resource Consumption and Accuracy

Since BERT is a pre-trained model, we typically use tokenization and input encoding as the primary feature format. Moreover, BERT handles embeddings, attention masks and streamline the feature preparing process so we do not need to explicitly specify them. Hence, we will be discussing other feature format such as Word Embedding and Bag-of-Word (BoW) that can impact the resource consumption and accuracy if we use other than BERT model.

In general, word embeddings can have a lower memory consumption, but the size of embedding vectors and vocabulary size may influence the memory usage. Additionally, they can capture semantic relationships between words but with complex context, their accuracy will be lower [15]. Similarly, BoW have lower resource consumption which make it suitable for large dataset. However, it disregards word order and context which make it unsuitable for this sematic classification tasks as it requires understanding of sequential information. With that, it might not capture the full semantic meaning which results in lower accuracy [16] [17].

# Chapter 4 Hotel Reviews

### 4.1.1 Dataset without rating score

BERT was pre-trained on unsupervised Wikipedia and Bookcorpus datasets using language modelling. which provided BERT with a strong foundation for understanding context in natural language [18]. Furthermore, BERT is a bidirectional model that was able to capture rich contextual knowledge during pre-training, hence it is capable of understanding sentiment relationships even with unlabelled data. When given unlabelled data, BERT carries out unsupervised training. The model would carry out tokenization with BERT tokenizer and thus obtain contextualised embeddings using the pre-trained BERT model as well as embedding with CLS. These 2 steps are similar to what is being done with labelled data.

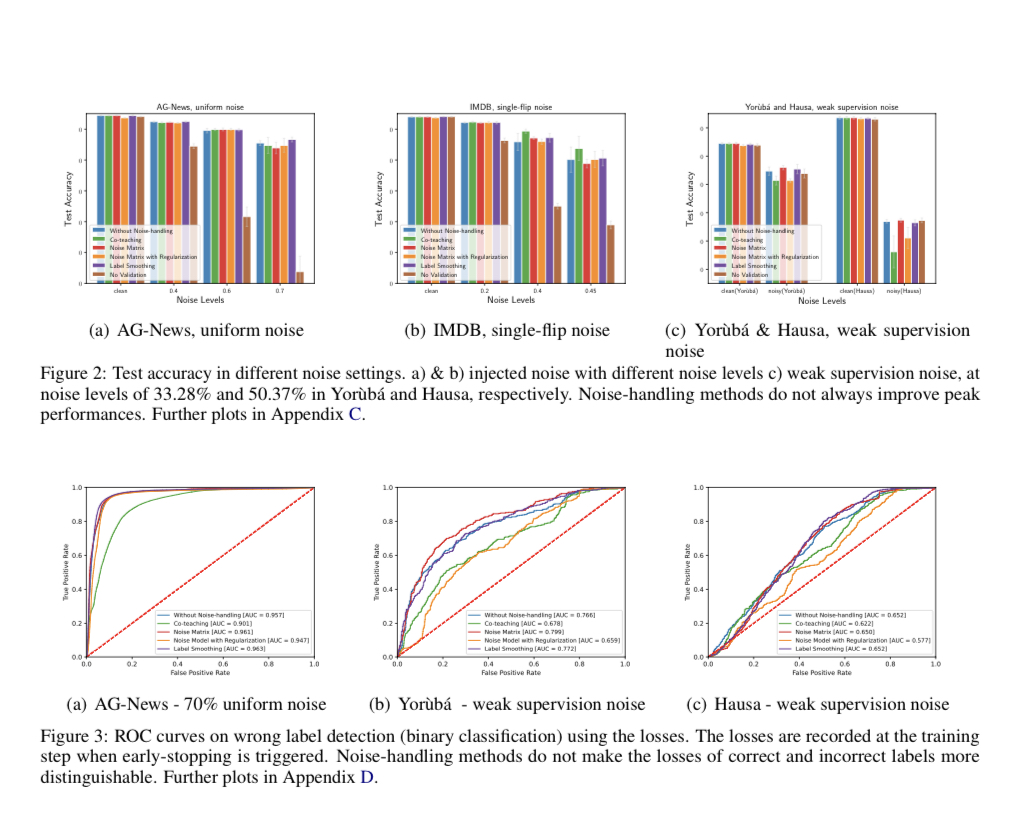
Next, unsupervised clustering algorithm such as k-means clustering would be applied to group similar representations together instead. The centroids of the clusters would then be analysed to interpret the respective sentiment it could represent. For example, if a cluster has embeddings that are predominantly positive, it might be associated with positive sentiment [19]. Lastly, establish thresholds based on the distances between each data point and the centroids to assign the data points to a cluster. This would provide us with a predicted sentiment label for each review.

However, the unsupervised approach would result in a lower accuracy of the prediction as compared to the supervised approach. This is due to the lack of specialised fine-tuning that would enable BERT to capture domain-specific sentiment expressions and patterns, in this case, the hotel. Additionally, in the unsupervised approach, there is a requirement for the user to decipher and analyse the cluster centroids to determine whether they represent positive or negative sentiments. This contrasts with the supervised approach, where the BERT model has already made classifications based on pre-learned patterns from labelled data.

To improve on the model, we can conduct intermediate pre-training by creating a custom tokenizer [18]. As the analytical task if focusing on general sentiment analysis, a custom vocabulary is not required as words used in reviews are generally well represented in standard vocabulary and does not contain unique descriptors or terminology. Hence, we can proceed by defining specific rules for tokenisation with respect to the characteristics specific to the hotel reviews that we are analysing [20]. For example, the hotel name and the unique sentiments or features offered. With a custom tokenizer, this can improve the classification accuracy of the model as this ensures that important information specific to the hotel is not lost and is being considered during the sentiment analysis. In addition, custom tokenizers help to facilitate more meaningful clustering and extraction of sentiment patterns during unsupervised analysis and improves interpretability, making it easier for the users analyse the cluster centroids during k-means clustering.

### 4.1.2 Dataset with inaccurate rating score

Research has found out that BERT is noise robust on injected noise but does not perform as well on weak supervision noise. When injected noise is used, the test accuracy drops only about 10% after injecting 70% wrong labels shown in diagram 1. However, under weak supervision noise, the performance can drop up to 35% in a dataset like Hausa with 50% noise [21].

  
Diagram 1: Test accuracy in different noise settings. a) & b) injected noise with different noise levels c) weak supervision noise, at noise levels of 33.28% and 50.37% in Yorùbá and Hausa, respectively. [21]

Approaches that can improve on algorithm against noise:

1. Data Cleaning

Data cleaning methods such as data validation, normalization, transformation, or correction can be employed to remove or fix any errors or inconsistencies and improve the accuracy. Outlier detection methods like z-score, interquartile range, or clustering can be used to identify and separate any outliers or extreme values.

1. Increase robustness and ability to generalise

To improve the algorithm to perform better with noise, it is important to increase model robustness as a more robust model is better equipped to handle variations, outliers, and uncertainties. Allowing the model to learn to capture the underlying patterns and relationships that generalize well to new, unseen data. This helps the model perform better when facing noisy inputs in real-world scenarios.

Implement techniques like MixUp or CutMix to augment training data, which helps with increase model robustness to noise and generalisation. MixUp is a data arugument technique that generate new samples by linear interpolation of multiple samples and their labels and is proven that it is useful in improving generalisation and robustness [22]. By involving techniques like dropout, regularisation and batch normalisation helps in preventing the model from fitting noise too closely. L1 or L2 regularization terms panelise large weights. Label smoothing is a commonly used method to improve model’s generalization and calibration. It mixes the one-hot label with a uniform vector, preventing the model from getting overconfident on the samples [23].

Another study proposed a new method known as Co-teaching, where two deep neural network is trained simultaneously and let them to teach each other given every mini batch. Letting the network to communicate and upgrades itself, and it proven to be useful than many current methods in robustness of trained deep models [24].

1. Dimension reduction

High-dimensional data often contains noise or irrelevant features. Dimensionality reduction focuses on extracting the most informative features, promoting better generalization to new, unseen data. This can help the model learn underlying patterns and relationships in the data while ignoring noise. Dimensionality reduction methods like Principal Component Analysis (PCA) aims to capture the most significant variance in the data. In doing so, they may filter out components of the data that are considered noise. By retaining only the principal components that contribute the most to the variance, it can effectively reduce the influence of noisy dimensions [25].

## Contribution

|  |  |
| --- | --- |
| KOH JIAHUI (U2021624C) | Data Processing  CNN and its prediction  Hotel Reviews – Inaccurate rating score |
| SHAO SIYING (U2021377H) | Data Processing  RNN and its prediction  Hotel Reviews - Unlabelled |
| PYAE SONE KHIN (U2021752A) | Data Processing  BERT and its prediction  Evaluation of the model  Impact on Resource Consumption and Accuracy |

Reference

[1] S. Khalid, “Bert explained: A Complete Guide with Theory and tutorial,” Medium, https://medium.com/@samia.khalid/bert-explained-a-complete-guide-with-theory-and-tutorial-3ac9ebc8fa7c.

[2] R. Horev, “Bert explained: State of the art language model for NLP,” Medium, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[3] D. Srivastava, “Bert,” LinkedIn, https://www.linkedin.com/pulse/bert-darshika-srivastava-8thqf/?trk=article-ssr-frontend-pulse\_more-articles\_related-content-card.

[4] R. Kumar, “Introduction to Bert and segment embeddings,” Analytics Vidhya, <https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-bert/#:~:text=The%20BERTBase%20model%20uses,has%20around%20110M%20trainable%20parameters>.

[5] S. Saxena, “Binary cross entropy/log loss for binary classification,” Analytics Vidhya, https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/#:~:text=Binary%20Cross%20Entropy%2C%20also%20known,binary%20labels%20of%20a%20dataset.

[6] Optimization, https://huggingface.co/docs/transformers/main\_classes/optimizer\_schedules.

[7] L. Paulacy, “Convolutional Neural Networks for Sequence Processing: Part 1,” *Medium*, Sep. 28, 2018. <https://laxfed.medium.com/convolutional-neural-networks-for-sequence-processing-part-1-420dd9b500>.

[8] “Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network,” *upGrad blog*. https://www.upgrad.com/blog/basic-cnn-architecture/# (accessed Nov. 21, 2023).

[9] M. Rahman, “What You Need to Know about Sparse Categorical Cross Entropy,” *Medium*, Mar. 07, 2023. <https://rmoklesur.medium.com/what-you-need-to-know-about-sparse-categorical-cross-entropy-9f07497e3a6f>

[10] Jason Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” *Machine Learning Mastery*, Jul. 02, 2017. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

[11] N. Donges, “Recurrent neural networks 101: Understanding the basics of RNNs and LSTM,” Built In, 2019. <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>

[12] E. Zvornicanin and E. Zvornicanin, “What Are Embedding Layers in Neural Networks?Baeldung on Computer Science,” *Baeldung on Computer Science*, May 24, 2023. <https://www.baeldung.com/cs/neural-nets-embedding-layers#:~:text=In%20summary%2C%20we%20use%20embedding,is%20a%20specific%20input%20feature>.

[13] K. D. Chaudhuri, “Sentiment Analysis with LSTM,” *Analytics Vidhya*, Mar. 15, 2022. <https://www.analyticsvidhya.com/blog/2022/01/sentiment-analysis-with-lstm/>

[14] S. Li, “A Beginner’s Guide on Sentiment Analysis with RNN - Towards Data Science,” *Medium*, Jun. 21, 2018. <https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e>

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” arXiv.org, https://arxiv.org/abs/1301.3781.

[16] J. Brownlee, “A gentle introduction to the bag-of-words model,” MachineLearningMastery.com, https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[17] “Bag of words: Approach, python code, Limitations,” Quantitative Finance & Algo Trading Blog by QuantInsti, https://blog.quantinsti.com/bag-of-words/.

[18] A. Shankar, “­­­Training BERT from Scratch on Your Custom Domain Data: A Step-by-Step Guide with Amazon SageMaker,” *Medium*, Nov. 13, 2023. [https://medium.com/@shankar.arunp/training-bert-from-scratch-on-your-custom-domain-data-a-step-by-step-guide-with-amazon-25fcbee4316a](mailto:https://medium.com/@shankar.arunp/training-bert-from-scratch-on-your-custom-domain-data-a-step-by-step-guide-with-amazon-25fcbee4316a)

[19]. Mateiionita, “Visualizing BERT, plus an unsupervised solution,” *Kaggle*, Mar. 10, 2019. <https://www.kaggle.com/code/mateiionita/visualizing-bert-plus-an-unsupervised-solution>

[20] “Building a tokenizer, block by block - Hugging Face NLP Course.” [Online]. Available: <https://huggingface.co/learn/nlp-course/chapter6/8?fw=pt>

[21] D. Zhu, M. Hedderich, F. Zhai, D. Ifeoluwa, and D. Klakow, “Is BERT Robust to Label Noise? A Study on Learning with Noisy Labels in Text Classification,” 2022. Accessed: Nov. 21, 2023. [Online]. Available: <https://aclanthology.org/2022.insights-1.8.pdf>

[22] L. Zhang, Z. Deng, K. Kawaguchi, A. Ghorbani, and J. Zou, “HOW DOES MIXUP HELP WITH ROBUSTNESS AND GENERALIZATION?” Accessed: Nov. 21, 2023. [Online]. Available: https://openreview.net/pdf?id=8yKEo06dKNo#:~:text=Mixup%20is%20a%20popular%20data

[23] W. Wong, “What is Label Smoothing?,” *Medium*, Jan. 20, 2020. https://towardsdatascience.com/what-is-label-smoothing-108debd7ef06

[24] B. Han *et al.*, “Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels.” [Online]. Available: <https://proceedings.neurips.cc/paper_files/paper/2018/file/a19744e268754fb0148b017647355b7b-Paper.pdf>

[25] “What Is Principal Component Analysis (PCA) and How It Is Used?,” Sartorius. https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186#:~:text=Principal%20component%20analysis%2C%20or%20PCA