มหาวิทยาลัยอัสสัมชัญ
ASSUMPTION UNIVERSITY
of THAILAND

VINCENT MARY
SCHOOL of SCIENCE and
TECHNOLOGY

Artificial Intelligence Concepts Report

Hate Speech Detection Report

6238133 San Myint Hlaing,
6237407 Saranya Sangsuk-iem

A. Thitipong Tanprasert

Abstract

This project focuses on leveraging AI to combat hate speech and offensive content online. It has two primary objectives: Content Moderation and Education & Awareness. Content Moderation involves using AI to automatically detect and remove harmful content, making online platforms safer. Education & Awareness aims to flag hate speech and provide educational resources or warnings to encourage responsible online behavior. This report explores the development process, technologies used, ethical considerations, and future possibilities of this AI-based solution. Ultimately, the goal is to create a more respectful online environment.

Table Of Contents

Chapter 1: Introduction **1**
1.1 Scope of the Project 1
1.1.1 Objective 1
1.1.2 Data Source 1

Chapter 2: Processes **2**
2.1 Descriptive Statistics 2
2.2 Clean Text 2
2.2.1 Preprocessing 2
2.2.2 Virtualization 4

Chapter 3: Bert Model **5**
3.1 Train-Test 5

Chapter 4: Result **7**
4.1 Evaluated Model 7
4.2 Prediction 8

Chapter 5: Conclusion **10**

References **11**

Table Of Figures

Chapter 2: Preprocessing

Figure 2.1: Imbalance Class and Balance Class 2

Figure 2.2.1: Preprocessing part in Python 3

Figure 2.2.2: Word Cloud images 4

Figure 2.2.3: Bigrams 4

Chapter 3: Bert Model

Figure 3.1: Three subsets DataFrame 5

Figure 3.2: Build, train, and evaluate a text classification using BERT 5

Figure 3.3: Loss and Accuracy plots 7

Chapter 4: Result 4

Figure 4.1: Confusion Matrix 8

Figure 4.2: Load weights 8

Figure 4.2.1: Pre-trained text classification model 9

Chapter 1: Introduction

In the digital age, the rise of online communication platforms has brought with it a concerning surge in hate speech and offensive content. This project focuses on deploying artificial intelligence (AI) to combat this issue with two key objectives: Content Moderation and Education & Awareness.

Content Moderation aims to use AI to automatically identify and remove hate speech and offensive content from online platforms. This approach leverages advanced natural language processing techniques to distinguish harmful content from regular discourse, creating a safer online environment while reducing the reliance on human moderators.

Education & Awareness goes a step further by not only flagging hate speech but also providing educational resources and warnings to users. This proactive approach encourages responsible online behavior by helping users understand the consequences of their actions.

This project report delves into the development process, technology utilization, ethical considerations, and future possibilities of this AI-driven solution. Ultimately, the goal is to promote a more respectful and inclusive online environment, fostering empathy and tolerance while preserving free expression.

1.1 Scope of the project

1.1.1 Objective

The primary objectives of this project are Content Moderation: Use the AI to automatically detect and remove hate speech and offensive content from online platforms to create a safer and more respectful environment. And Education and Awareness: Use the AI to flag hate speech and provide educational resources or warnings to users, encouraging responsible online behavior.

1.1.2 Data Source

The dataset or data collection for this project will be sourced from Kaggle (https://www.kaggle.com/datasets/mrmorj/hate-speech-and-offensive-language-dataset).

Chapter 2: Processes

2.1 Descriptive Statistics

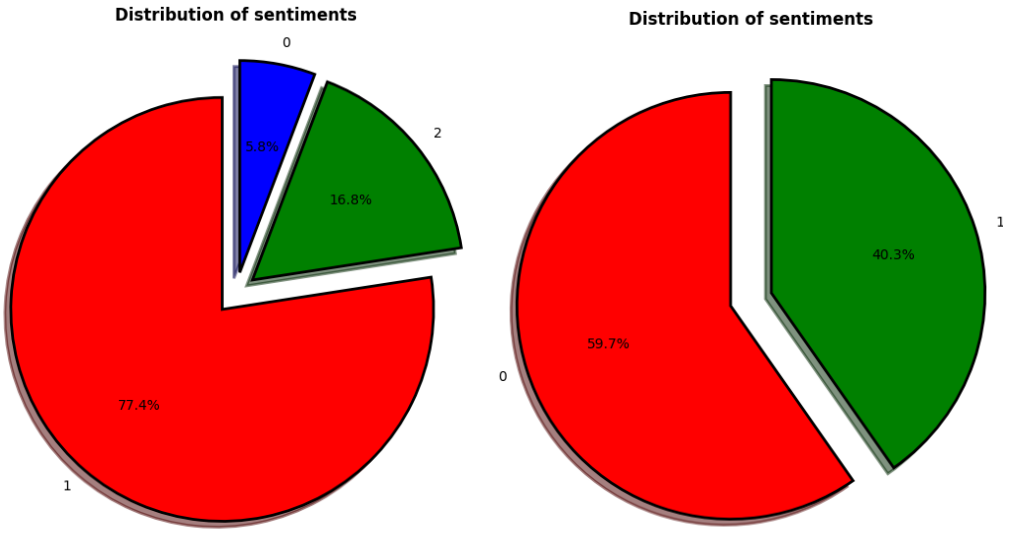


Figure 2.1: (1)Imbalance Class (2)Balance Class

We met an imbalanced class problem as most of the data is 77% of hate speech. Therefore, the problem was solved by increasing the number of non-hate speech messages so that the data had an equal number of classes (find more from Kaggle). Then we generate a pie chart for visualizing the distribution of sentiments or labels in a dataset. It uses Matplotlib to create the chart with specific attributes, such as colors, separation of segments, and a title, to make it visually informative and appealing.

2.2 Clean Text

2.2.1 Preprocessing

We will detail the preprocessing steps applied to the collected data, such as tokenization, stop word removal, and stemming or lemmatization. Additionally, we will explore various text analysis techniques, including sentiment lexicon-based approaches and machine learning-based methods.

```
stopword = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return ''

def clean_text(text):
    text = str(text).lower()
    patterns = [
        ('\[.*?\]', ''),
        ('https?://\S+|www\.\S+', ''),
        ('<.*?>+', ''),
        (r"\@w+|\#", ''),
        (r"[^\w\s]", ''),
        (f'[{re.escape(string.punctuation)}]', ''),
        ('\n', ''),
        ('\w*\d\w*', ''),
    ]
    for pattern, repl in patterns:
        text = re.sub(pattern, repl, text)
    tokens = word_tokenize(text)
    lemmatized_tokens = [lemmatizer.lemmatize(word, wordnet.VERB) for word in tokens if word not in stopword]
```

Figure 2.2.1: Preprocessing part in Python

The provided code is a Python script for text preprocessing, specifically designed to prepare text data for natural language processing (NLP) or text analysis tasks. It accomplishes this by performing the following main steps:

1. **Stopword Removal:** It removes common stopwords (e.g., "the," "and," "in") from the text, which are often semantically insignificant and can be safely omitted for analysis.
2. **Lemmatization:** It lemmatizes words in the text, reducing them to their base or dictionary form. This step ensures that different word forms (e.g., "running" and "ran") are treated as the same word, simplifying analysis.
3. **Text Cleaning:** It applies a series of regular expression patterns to clean the text. This includes removing URLs, HTML tags, Twitter handles, hashtags, special characters, punctuation, newline characters, and alphanumeric sequences containing digits.
4. **Tokenization:** It tokenizes the cleaned text into individual words or tokens, making it ready for further analysis.

Overall, this code is crucial for preparing text data for various NLP tasks, enhancing data quality, and simplifying subsequent analysis.

2.2.2 Visualization

We create a word cloud visualization from text data in a DataFrame. It collects the text, removes common stopwords, and generates a word cloud image to visually represent the most frequently occurring words in the text. The resulting word cloud is displayed using Matplotlib.

Figure 2.2.2: Word Cloud images

We also analyze text data in a DataFrame (df) to find and visualize the top 10 most frequent bigrams (two-word combinations) for a specific category of data (where 'label' is equal to 0).

Data Preparation: We first collect and tokenize the text data, splitting it into individual words then extracts bigrams (pairs of consecutive words) from the tokenized text.

Bigram Counting: The code counts the occurrences of each unique bigram and stores them in a DataFrame, sorted by frequency.

Visualization: We use Matplotlib and Seaborn to create a horizontal bar plot. The top 10 most frequent bigrams are displayed on the y-axis, and their respective counts are shown on the x-axis. Text labels with the counts are added to each bar in the plot for clarity.

Plot Customization: The plot is customized with labels and titles to make it informative and visually appealing.

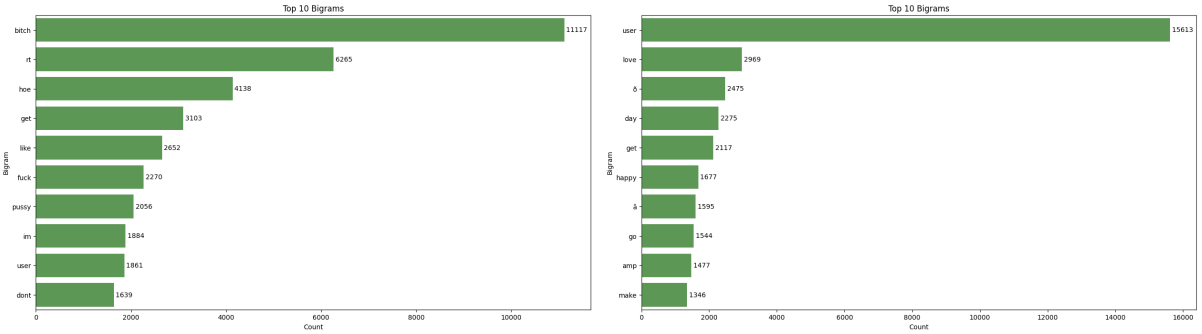


Figure 2.2.3: Bigrams

In summary, we extract and visualize the top 10 most common two-word combinations (bigrams) in a specific category of text data, providing insights into the language patterns within that category.

Chapter 3: Bert Model

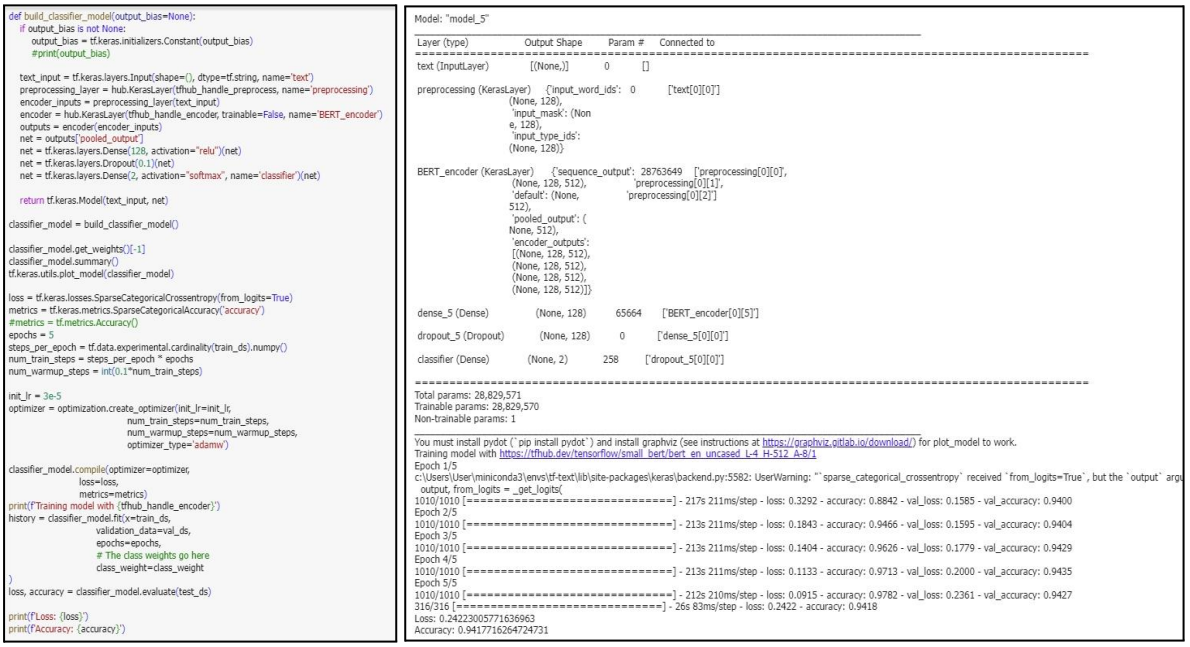
3.1 Train-Test

```
df_train = df.loc[df["data_type"]=="train"].sample(frac=1)
df_val = df.loc[df["data_type"]=="val"].sample(frac=1)
df_test = df.loc[df["data_type"]=="test"].sample(frac=1)

train_ds = tf.data.Dataset.from_tensor_slices((df_train.text.values, df_train.label.values)).batch(32, drop_remainder=False)
val_ds = tf.data.Dataset.from_tensor_slices((df_val.text.values, df_val.label.values)).batch(32, drop_remainder=False)
test_ds = tf.data.Dataset.from_tensor_slices((df_test.text.values, df_test.label.values)).batch(32, drop_remainder=False)
```

Figure 3.1: Three subsets DataFrame

This code is used to split a DataFrame (df) into three subsets: training data (df_train), validation data (df_val), and test data (df_test). These subsets are typically used for training, fine-tuning, and evaluating machine learning models, such as a BERT model. The “32” number is the dataset we import to train the model each time.



```
def build_classifier_model(output_bias=None):
    if output_bias is not None:
        output_bias = tf.keras.initializers.Constant(output_bias)
        #print(output_bias)

    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dense(128, activation="relu")(net)
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(2, activation="softmax", name='classifier')(net)

    return tf.keras.Model(text_input, net)

classifier_model = build_classifier_model()

classifier_model.get_weights()[-1]
classifier_model.summary()
tf.keras.utils.plot_model(classifier_model)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
#metrics = tf.metrics.Accuracy()
epochs = 5
steps_per_epoch = tf.data.experimental.cardinality(train_ds).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
                                          num_train_steps=num_train_steps,
                                          num_warmup_steps=num_warmup_steps,
                                          optimizer_type='adamw')

classifier_model.compile(optimizer=optimizer,
                         loss=loss,
                         metrics=metrics)
print(f'Training model with {tfhub_handle_encoder}')
history = classifier_model.fit(x=train_ds,
                               validation_data=val_ds,
                               epochs=epochs,
                               # The class weights go here
                               class_weight=class_weight
)
loss, accuracy = classifier_model.evaluate(test_ds)

print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

```
Model: "model_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
text (InputLayer)               [(None,)]            0           []

preprocessing (KerasLayer)      {'input_word_ids':   0           ['text[0][0]']
                                 (None, 128),
                                'input_mask': (Non
                                e, 128),
                                'input_type_ids':
                                (None, 128)}

BERT_encoder (KerasLayer)       {'sequence_output':  28763649    ['preprocessing[0][0]',
                                 (None, 128, 512),               'preprocessing[0][1]',
                                'default': (None,                'preprocessing[0][2]']
                                512),
                                'pooled_output': (
                                None, 512),
                                'encoder_outputs':
                                [(None, 128, 512),
                                (None, 128, 512),
                                (None, 128, 512),
                                (None, 128, 512)]}

dense_5 (Dense)                 (None, 128)          65664       ['BERT_encoder[0][5]']

dropout_5 (Dropout)             (None, 128)          0           ['dense_5[0][0]']

classifier (Dense)              (None, 2)            258         ['dropout_5[0][0]']

==================================================================================================
Total params: 28,829,571
Trainable params: 28,829,570
Non-trainable params: 1
__________________________________________________________________________________________________
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot_model to work.
Training model with https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Epoch 1/5
c:\Users\User\miniconda3\envs\tf-text\lib\site-packages\keras\backend.py:5582: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argu
  output, from_logits = _get_logits(
1010/1010 [==============================] - 217s 211ms/step - loss: 0.3292 - accuracy: 0.8842 - val_loss: 0.1585 - val_accuracy: 0.9400
Epoch 2/5
1010/1010 [==============================] - 213s 211ms/step - loss: 0.1843 - accuracy: 0.9466 - val_loss: 0.1595 - val_accuracy: 0.9404
Epoch 3/5
1010/1010 [==============================] - 213s 211ms/step - loss: 0.1404 - accuracy: 0.9626 - val_loss: 0.1779 - val_accuracy: 0.9429
Epoch 4/5
1010/1010 [==============================] - 213s 211ms/step - loss: 0.1133 - accuracy: 0.9713 - val_loss: 0.2000 - val_accuracy: 0.9435
Epoch 5/5
1010/1010 [==============================] - 212s 210ms/step - loss: 0.0915 - accuracy: 0.9782 - val_loss: 0.2361 - val_accuracy: 0.9427
316/316 [==============================] - 26s 83ms/step - loss: 0.2422 - accuracy: 0.9418
Loss: 0.24223005771636963
Accuracy: 0.941771626472473
```

Figure 3.2: Build, train, and evaluate a text classification using BERT

This code builds, trains, and evaluates a text classification model using BERT embeddings.

1. **def build_classifier_model:** This function constructs a text classification model using TensorFlow and BERT embeddings. It takes an optional `output_bias` parameter, which can be used to initialize the output layer's bias. This can be useful for handling class imbalance in the data.
2. **Model Architecture:** The text input is defined as a TensorFlow input layer, expecting text data as input with a shape of `()`. A preprocessing layer is added using the TensorFlow Hub (`hub.KerasLayer`) to preprocess text input data using a specified pre-trained BERT model (`tfhub_handle_preprocess`). The BERT encoder layer is added using another TensorFlow Hub layer (`tfhub_handle_encoder`). The encoder is marked as non-trainable (`trainable=False`) since the BERT model is pre-trained. The model's output is obtained from the BERT encoder's pooled output. Two dense layers are added for classification, with a ReLU activation function in the first layer and a softmax activation function in the final layer.
3. **Model Compilation:** The model is compiled with an optimization strategy using a specified learning rate schedule. Loss is defined as sparse categorical cross-entropy. The accuracy metric is used to evaluate the model's performance during training.
4. **Training the Model:** The model is trained using the `fit` method with the training dataset (`train_ds`) and validated using the validation dataset (`val_ds`). The training configuration includes the number of epochs, steps per epoch, and class weights if needed. Class weights can help address class imbalance in the dataset. Training progress is stored in the `history` variable.
5. **Model Evaluation:** After training, the model is evaluated on the test dataset (`test_ds`) to measure its performance. The loss and accuracy of the model on the test data are printed.

Overall, this code demonstrates the construction, training, and evaluation of a text classification model with BERT embeddings. It utilizes TensorFlow and TensorFlow Hub to simplify the process of working with pre-trained BERT models and fine-tuning them for specific classification tasks.

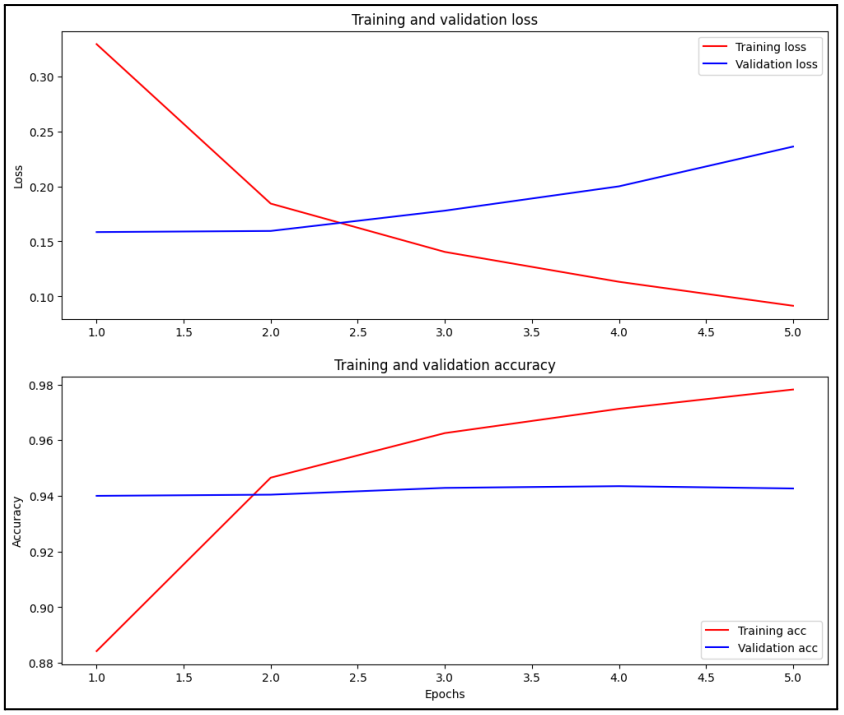


Figure 3.3: Loss and Accuracy plots

We visualize the training and validation performance of a machine learning model over multiple training epochs. It displays two subplots: one for loss (indicating how well the model is fitting the data) and another for accuracy (indicating the model's prediction accuracy). These visualizations help monitor the model's training progress and identify potential issues like overfitting.

Chapter 4: Result

4.1 Evaluated Model

We evaluate the performance of a trained machine learning model (classifier_model) on a test dataset. It computes and displays a confusion matrix, which helps assess how well the model is classifying data, and a classification report that provides detailed metrics like precision, recall, and F1-score for each class. This evaluation allows you to gauge the model's classification accuracy and other performance characteristics on unseen data.

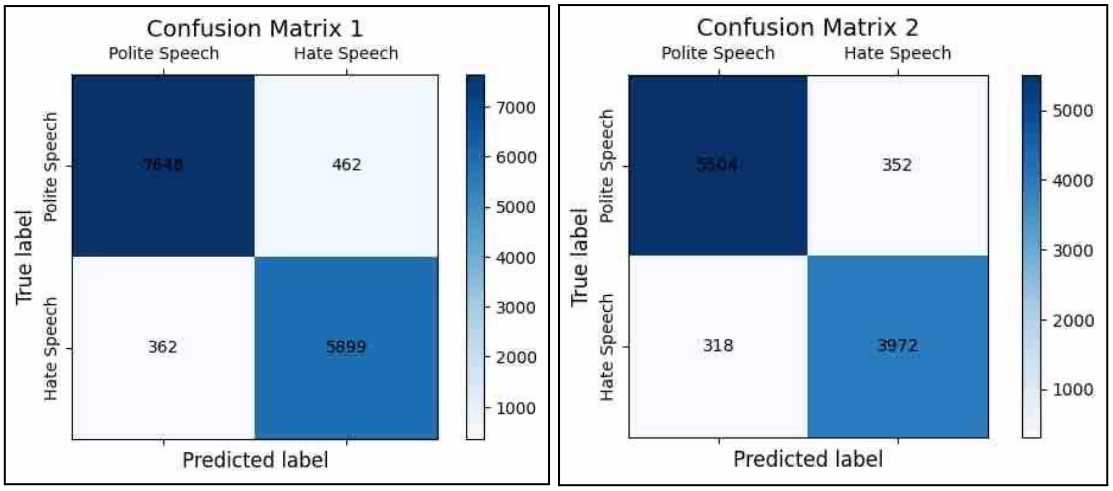


Figure 4.1: Confusion Matrix

4.2 Prediction

```
def build_classifier_model(output_bias=None):
    if output_bias is not None:
        output_bias = tf.keras.initializers.Constant(output_bias)
        #print(output_bias)

    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=False, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dense(128, activation="relu")(net)
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(2, activation="softmax", name='classifier')(net)

    return tf.keras.Model(text_input, net)

tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1'
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1'

classifier_model = build_classifier_model()
# Load weight model
classifier_model.load_weights('/content/drive/MyDrive/hatespeech/BERT_HateSpeechDetection_weights.h5')
# Ensure its architecture is the same as the original model
classifier_model.summary()

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Model: "model_1"
_____
Layer (type)                 Output Shape              Param #   Connected to
==========
text (InputLayer)            [(None,)]                 0         []

✓ เชื่อมต่อกับ ของแบ็กเอนด์ Google Compute Engine ที่ใช้ Python 3 แล้ว
```

Figure 4.2: Load weights

This code is used to load a pre-trained BERT-based text classification model and its weights for hate speech detection. Here's a brief explanation of each part of the code:

1. **Building a BERT-based Model:** build_classifier_model(output_bias=None): This function constructs a text classification model using TensorFlow and BERT embeddings. It defines the architecture of the model, including input layers for text, preprocessing using a BERT-specific preprocessing layer, and the BERT encoder layer.

The model includes additional layers for classification, such as dense layers with ReLU activation functions and a softmax output layer. The output_bias parameter allows for bias initialization to address class imbalance if provided.

2. **TensorFlow Hub Handles:** tfhub_handle_encoder and tfhub_handle_preprocess: These variables store the URLs of TensorFlow Hub models for BERT encoder and preprocessing. They define the architecture and preprocessing steps for the BERT-based model.
3. **Loading the Model Weights:** This loads the pre-trained weights of the BERT-based model from a specified file path. The weights should correspond to the architecture defined in the build_classifier_model function.
4. **Model Summary:** classifier_model.summary(): This command prints a summary of the loaded model, displaying the model's architecture and the number of parameters in each layer.

```
# Prediction
text = input()
y_pred = classifier_model.predict([text])
y_pred = np.argmax(y_pred, axis=-1)

print(f'This contains harsh words. Please use polite words.') if y_pred == 1 else print(text)
```

```
You are a bitch.
1/1 [==============================] - 1s 1s/step
This contains harsh words. Please use polite words.
```

Figure 4.2.1: Pre-trained text classification model

Finally, This code snippet takes user-provided text input and uses a pre-trained text classification model to predict whether the input contains hate speech. If hate speech is detected, it issues a warning message asking for polite language; otherwise, it displays the original input text.

Chapter 5: Conclusion

In summary, this project successfully achieved its objectives by developing a Hate Speech Detection system using advanced natural language processing and machine learning techniques. Key highlights include rigorous data preprocessing, the construction of a robust hate speech detection model based on BERT embeddings, and thorough performance evaluation. The system provides a user-friendly interface for immediate feedback on hate speech content, promoting responsible online behavior. Ethical considerations were prioritized, and future enhancements were identified to further improve the system's effectiveness in creating a safer online environment.

References

1. Papers with Code - Hate Speech Detection. (n.d.). https://paperswithcode.com/task/hate-speech-detection
2. GeeksforGeeks. (2022). Hate Speech Detection using Deep Learning. GeeksforGeeks. https://www.geeksforgeeks.org/hate-speech-detection-using-deep-learning/
3. Roshancyriacmathew. (n.d.). GitHub - roshancyriacmathew/hate-speech-detection-using-machine-learning: https://github.com/roshancyriacmathew/hate-speech-detection-using-machine-learning
4. Buvaneshwaran, K. (2022, November 30). Hate speech detection with Python - CopyAssignment. CopyAssignment. https://copyassignment.com/hate-speech-detection/
5. Mehta, H., & Passi, K. (2022). Social media hate speech detection using Explainable Artificial Intelligence (XAI). Algorithms, 15(8), 291. https://doi.org/10.3390/a15080291