Week 12: Project Deliverables

Project Topic: Hate Speech Detection using Transformers (Deep Learning)

Group Name: Data Defenders

Batch Code: LISUM34

Team Members' Details:

Name	Malika Hafiza Pasha	Biswadip Bhattacharyya	Oluseun Omobulejo
Email	malikahafizap@gmail.com	bbhattac@syr.edu	Seun.omobulejo@outlook.com
Country	United States	United States	United Kingdom
College/Company	California State University – Dominguez Hills	Syracuse University	University of Westminster
Specialization	Data Science		

Problem Description:

Any verbal, written, or behavioral communication that targets or uses derogatory or discriminatory language against an individual or group on the basis of who they are—that is, based on their religion, ethnicity, nationality, race, color, ancestry, sex, or another identity factor—is referred to as hate speech. We will walk you through a Python and machine learning hate speech detection model in this problem.

Sentiment categorization is often the process involved in hate speech detection. Therefore, training a model on data that is often used to classify attitudes can result in a model that can identify hate speech from a given text passage. Therefore, in order to complete the objective of developing a hate speech recognition model, we will use Twitter to find tweets that include hate speech.

Business Understanding:

The rise of social media and online communication platforms has increased the dissemination of hate speech. It is crucial for businesses, social media platforms, and communities to identify and address hate speech to maintain a safe and inclusive environment. Effective detection and moderation can enhance user experience, comply with regulations, and protect brand reputation.

Data Collection:

Dataset Details:	
1. Dataset Name	train_E6oV3lV
2. Dataset storage location	[Twitter hate speech (kaggle.com)]
3. Base format of the file	csv
4. Size of the data	2.95 MB
5. Total number of observations	31962
6. Total number of files	1
7. Total number of features	3
8. Proposed Approach	There are no missing vales in this dataset

Data Information and Data Preprocessing

A. Data Information

1. Importing Libraries:

The necessary libraries for data preprocessing are imported. This typically includes pandas, numpy, re (regular expressions), and string libraries.

2. Loading the Dataset:

The dataset is loaded into a pandas DataFrame. The specific dataset and its location are not specified in the extracted content, but typically, it would involve using pd.read_csv or similar functions.

3. Initial Data Inspection:

Basic data inspection techniques are used, such as displaying the first few rows of the dataset using df.head() and getting the summary of the dataset with df.info().

```
In [5]:  # total number of observations and features
         print(f'Number of Observations: {df.shape[0]}')
         print(f'Number of Features: {df.shape[1]}')

         Number of Observations: 31962
         Number of Features: 3

In [6]:  # features that exists in this data
         df.columns

Out[6]:  Index(['id', 'label', 'tweet'], dtype='object')

In [7]:  # type of data in the dataset
         df.dtypes

Out[7]:  id        int64
         label     int64
         tweet    object
         dtype: object

In [8]:  # information about the data
         df.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 31962 entries, 0 to 31961
         Data columns (total 3 columns):
          #   Column  Non-Null Count  Dtype
         ---  ------  --------------  -----
          0   id      31962 non-null  int64
          1   label   31962 non-null  int64
          2   tweet   31962 non-null  object
         dtypes: int64(2), object(1)
         memory usage: 749.2+ KB

In [9]:  # size of the data
         df.size

Out[9]:  95886
```

```
In [10]:  # Predictor Attribute
          text = df.iloc[:, 1:]
          text.tail()

Out[10]:
```

	label	tweet
31957	0	ate @user isz that youuu?ð□□□ð□□□ð□□□ð□□□ð□□□ð...
31958	0	to see nina turner on the airwaves trying to...
31959	0	listening to sad songs on a monday morning otw...
31960	1	@user #sikh #temple vandalised in in #calgary,...
31961	0	thank you @user for you follow

```
In [11]:  # target Attribute
          label = df.iloc[:, 0:1]
          label.tail()

Out[11]:
```

	id
31957	31958
31958	31959
31959	31960
31960	31961
31961	31962

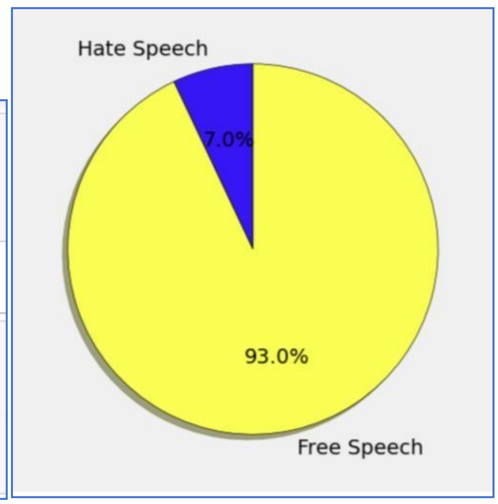
```
In [12]:  # checking the missing values
          df.isnull().sum()

Out[12]:  id       0
          label    0
          tweet    0
          dtype: int64
```

```
In [13]:  # Extract the Label Feature for Each Class
          hate_speech = df[df['label'] == 1].shape[0]
          free_speech = df[df['label'] == 0].shape[0]
          print('Hate Speech =', hate_speech)
          print('Free Speech =', free_speech)

          Hate Speech = 2242
          Free Speech = 29720

In [14]:  # Visualize the Label Class
          speech = [hate_speech, free_speech]
          label = ["Hate Speech", "Free Speech"]
          plt.pie(speech, labels = label, shadow = True, wedgeprops = {'edgecolor': 'black'},
                  autopct = '%1.1f%%', startangle= 90, colors=['blue', 'yellow'])
          plt.tight_layout()
          plt.show()
```



2. Data Preprocessing - Text Cleaning:

a. Removing Punctuation: This function removes all punctuation from the text data.

```
# Text Cleaning: Removing Punctuation
def remove_punct(text):
    return text.translate(str.maketrans('','',string.punctuation))
df['tweet'] = df['tweet'].apply(remove_punct)
```

b. Removing URLs: This function removes URLs from the text data.

```
# Text Cleaning: Removing URLs
def remove_punct(text):
    return text.translate(str.maketrans('','',string.punctuation))
df['tweet'] = df['tweet'].apply(remove_punct)
```

c. Removing Tags: This function removes tags (e.g., @username) from the text data.

```
# Text Cleaning: Removing Tags
def remove_tag(text):
    newtext= re.sub(r'(@[A-Za-z0-9]+)',"",text)
    return newtext
df['tweet'] = df['tweet'].apply(remove_tag)
```

d. Removing Special Characters: This function removes special characters from the text data, keeping only alphanumeric characters.

```
# Text Cleaning: Removing Special Characters
def remove_special(text):
    return " ".join(e for e in text.split() if e.isalnum())
df['tweet'] = df['tweet'].apply(remove_special)
```

e. Tokenization: Involves splitting text into individual words or tokens.

```python
# tokenizing
from nltk.tokenize import sent_tokenize, word_tokenize
def tokenize(text):
    text = word_tokenize(text)
    return text
df['tweet'] = df['tweet'].apply(tokenize)
```

f. Removing Stopwords: It is usually removed to focus on the meaningful words in the text.

```python
# removing stopwords
from nltk.corpus import stopwords
def remove_stop(text):
    text  = [i for i in text if not i in stopwords.words('english')]
    return text
df['tweet'] = df['tweet'].apply(remove_stop)
```

g. Lemmatization: It reduce words to their root forms. Lemmatization considers the context and converts words to their base forms.

```python
# Lemmatization
from nltk.stem import WordNetLemmatizer
def Lemmatize(text):
    word_lem = WordNetLemmatizer()
    text = [word_lem.lemmatize(token) for token in text]
    return text
df['tweet'] = df['tweet'].apply(Lemmatize)
```

3. Feature Extraction- Word Embedding with BERT:

Using the BERT Model to convert the processed text into dense vector representations. These embeddings capture the semantic meaning of the words in context, which is crucial for effective hate speech detection. The embeddings are derived from the final hidden layers of BERT, providing a rich feature set for model training.

a. Importing necessary libraries such as torch and Transformers

```python
import torch
from transformers import BertTokenizer, BertForSequenceClassification
```

b. Load the pre-trained Bert tokenizer and model

```python
# Load pre-trained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=2)
```
✓ 1.2s

c. Use the tokenizer to tokenize the text data (train and test)

```python
encoded_inputs = tokenizer(balanced_data['tweet'].tolist(), padding=True, truncation=True, max_length=512, return_tensors='pt')
```
✓ 11.6s

```python
# Tokenize the test data
test_encoded_inputs = tokenizer(df_test['tweet'].tolist(),
                                padding=True,
                                truncation=True,
                                max_length=128,
                                return_tensors='pt')

# Extract input IDs and attention masks
test_input_ids = test_encoded_inputs['input_ids']
test_attention_masks = test_encoded_inputs['attention_mask']

# Create a DataLoader for the test set
test_dataset = TensorDataset(test_input_ids, test_attention_masks)
test_dataloader = DataLoader(test_dataset, sampler=SequentialSampler(test_dataset), batch_size=32)
```

4. Model Training:

```python
from torch.optim import AdamW
from transformers import get_linear_schedule_with_warmup

# Optimizer and learning rate scheduler
optimizer = AdamW(model.parameters(), lr=2e-5)
epochs = 3
total_steps = len(train_dataloader) * epochs

# Scheduler for learning rate decay
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)

for epoch in range(epochs):
    model.train()
    total_loss = 0

    for step, batch in enumerate(train_dataloader):
        batch_input_ids = batch[0].to(device)
        batch_input_mask = batch[1].to(device)
        batch_labels = batch[2].to(device)

        # Clear gradients
        model.zero_grad()

        # Forward pass
        outputs = model(batch_input_ids, attention_mask=batch_input_mask, labels=batch_labels)
        loss = outputs.loss
        total_loss += loss.item()

        # Backward pass
        loss.backward()

        # Update weights
        optimizer.step()
        scheduler.step()

    print(f"Epoch {epoch + 1} Loss: {total_loss / len(train_dataloader)}")
```

```
Epoch 1 Loss: 0.2407190567784984
Epoch 2 Loss: 0.06772707958632396
Epoch 3 Loss: 0.027000677799207472
```

```python
from sklearn.metrics import accuracy_score, classification_report

model.eval()
predictions, true_labels = [], []

with torch.no_grad():
    for batch in val_dataloader:
        batch_input_ids = batch[0].to(device)
        batch_input_mask = batch[1].to(device)
        batch_labels = batch[2].to(device)

        outputs = model(batch_input_ids, attention_mask=batch_input_mask)
        logits = outputs.logits
        predictions.append(logits.argmax(dim=1).cpu().numpy())
        true_labels.append(batch_labels.cpu().numpy())

# Flatten predictions and true labels
predictions = [item for sublist in predictions for item in sublist]
true_labels = [item for sublist in true_labels for item in sublist]

# Print accuracy and classification report
print("Accuracy:", accuracy_score(true_labels, predictions))
print(classification_report(true_labels, predictions))
```

```
Accuracy: 0.9803162853297442
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      5996
           1       0.97      0.99      0.98      5892

    accuracy                           0.98     11888
   macro avg       0.98      0.98      0.98     11888
weighted avg       0.98      0.98      0.98     11888
```

5. Testing with the test data and save this file in csv:

```python
predictions = []

# No gradient calculations during inference
with torch.no_grad():
    for batch in test_dataloader:
        # Move inputs to the appropriate device
        batch_input_ids = batch[0].to(device)
        batch_attention_masks = batch[1].to(device)

        # Forward pass
        outputs = model(batch_input_ids, attention_mask=batch_attention_masks)

        # Get the predicted class (e.g., 0 for non-hate speech, 1 for hate speech)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1).cpu().numpy()

        # Store predictions
        predictions.extend(preds)

# Example: Print or save the predictions
print(predictions)
```

Python

```
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```python
import pandas as pd

# Create a DataFrame with predictions
pred_df = pd.DataFrame({'text': df_test['tweet'], 'prediction': predictions})

# Save to a CSV file
pred_df.to_csv('test_predictions.csv', index=False)
```

Python

GitHub Repository Link:

https://github.com/malikahafizap/Data_Glacier_Internship/tree/main/Week%208%20(19%20July%20%E2%80%93%2025%20July%202024)