Malicious Comments

Yingjie LIU, Ying LAI

Contents

Introduction

Data Preprocessing

Training and Evaluation

- TF-IDF, Word2Vec, FastText
- Transformer

Introduction

The datasets: Dataset using Twitter data, is was used to research hate-speech detection. The text is classified as: hate-speech, offensive language (label 1), and neither (label 0).

- Here we say text with label 1 is malicious.

Our Goal: We use labeled Twitter users' comments to determine whether the comments contain malicious intent.

	Content	Label
0	denial normal con ask comment tragedi emot retard	1
1	abl tweet insuffer bullshit prove trump nazi v...	1
2	retard cute singl life	1
3	thought real badass mongol style declar war at...	1
4	afro american basho	1

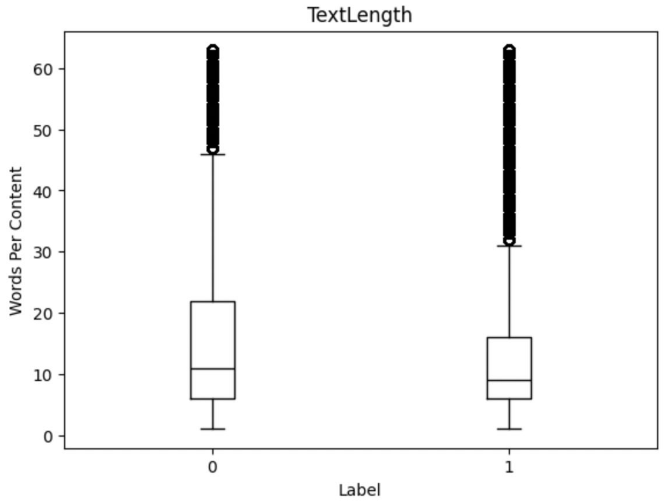
Data Preprocessing

Step 1: Remove useless words and symbol

- Remove HTML tags.
- Remove digits.
- Remove punctuation.
- Convert text to lowercase.
- Tokenize the text.
- Stem the tokens (reducing words to their root form).
- Remove stopwords (common words that typically don't add much meaning to the text).

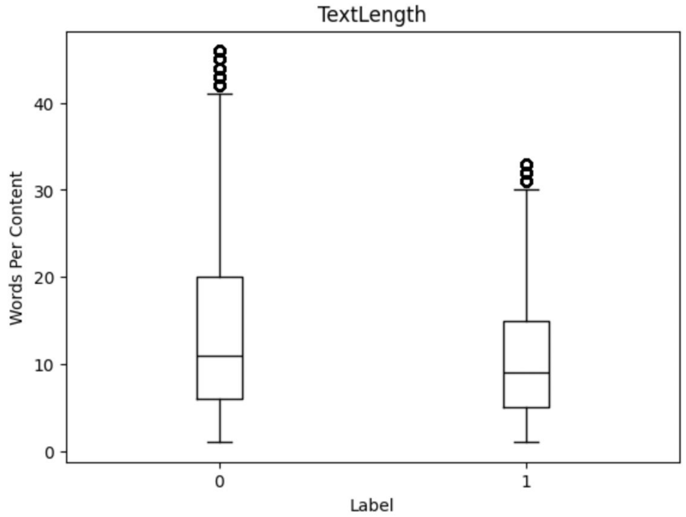
Step 2: Remove outliers

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR



<- Before

After ->



Data Preprocessing

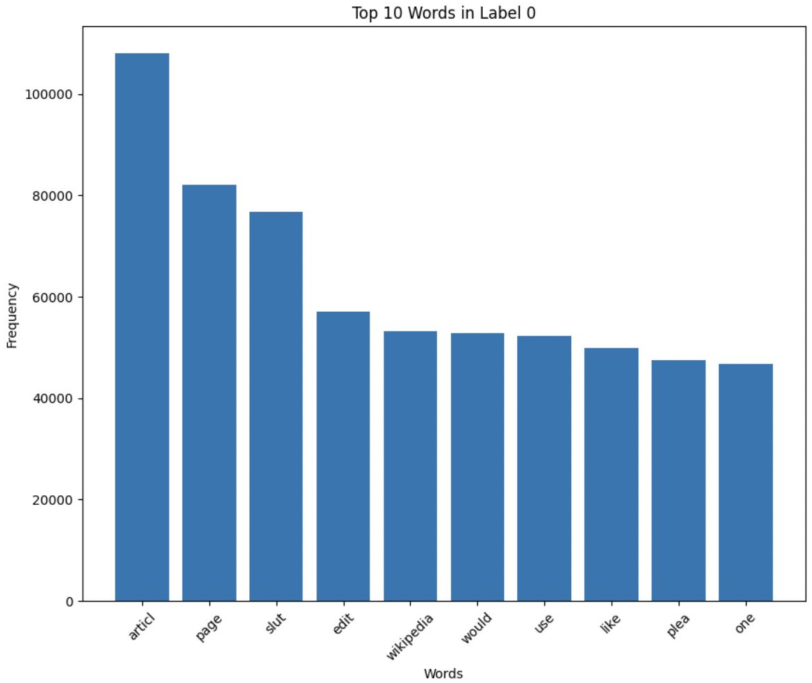
Step 3: Balance the datasets

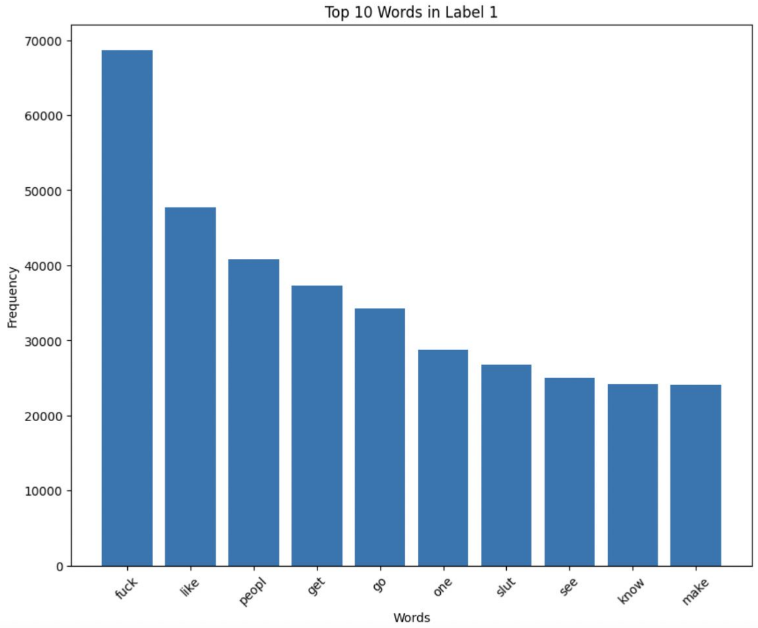
```
Label_counts = filtered_df['Label'].value_counts()
print(Label_counts)
```

```
Label
1    334524
0    314205
Name: count, dtype: int64
```

Reduce the number of samples from the majority class to the same number as the minority class

Find the top10 words in each label





Tokenize the texts, train and predict

Split the datasets:

80% training sets
20% testing sets.

- TF-IDF: a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus.
- Word2Vec: a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.
- FastText: an extension of Word2Vec intended to speed up learning. It treats each word as composed of character n-grams, so it can share representations across words, especially useful for languages with large vocabularies and many rare words. FastText can also predict labels for a document, not just word embeddings

Tokenize the texts, train and predict

- TF-IDF

```
# Create TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit-transform the data
X_tfidf = tfidf_vectorizer.fit_transform(df['Content'])
```

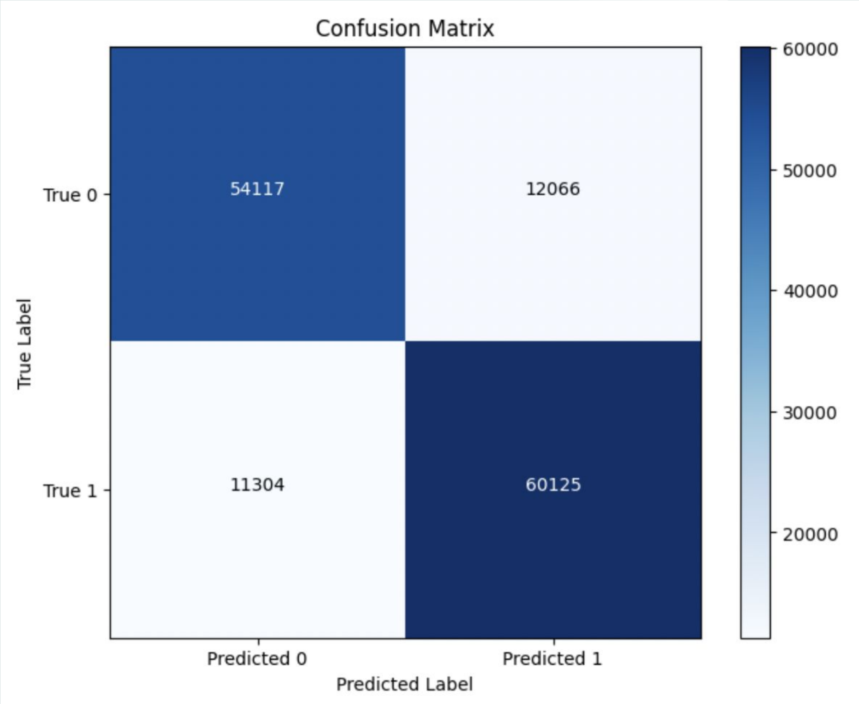
Logistic Regression classifier

```
# Instantiate the Logistic Regression classifier
logreg_classifier = LogisticRegression(max_iter=1000, solver='saga')

# Train the Logistic Regression classifier with progress bar
progress_bar = tqdm(total=100)
logreg_classifier.fit(X_train, y_train)
progress_bar.update(100)
progress_bar.close()

# Predict on the testing set
y_pred = logreg_classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
100%|██████████| 100/100 [00:13<00:00,  7.41it/s]
Accuracy: 0.8301746940673779
```

Split the datasets:

80% training sets
20% testing sets.



Tokenize the texts, train and predict

- Word2Vec

```
# Tokenize the cleaned reviews
tokenized_reviews = [review.split() for review in df['Content']]

# Train the Word2Vec model
w2v_model = Word2Vec(tokenized_reviews, vector_size=100, window=5, min_count=2, workers=4)
```

```
# Generate document feature vectors
def document_vector(word2vec_model, doc):
    # Delete words that are not in the vocabulary
    doc = [word for word in doc if word in word2vec_model.wv]
    if len(doc) == 0:
        return np.zeros(word2vec_model.vector_size)
    # Calculate the average of word vectors
    return np.mean(word2vec_model.wv[doc], axis=0)

# Compute vectors for each document
doc_vectors = np.array([document_vector(w2v_model, doc) for doc in tokenized_reviews])
```

Why choose the mean value of word vector?

Dimensionality consistency, preserve semantic information, and simple and effective.

Split the datasets:

80% training sets
20% testing sets.

PCA (n_component=50) and SVM classifier

```
# Prediction and Evaluation
y_pred = svm.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
negative	0.85	0.84	0.85	4961
positive	0.85	0.86	0.85	5039
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000

Tokenize the texts, train and predict

- FastText

Notice: here we add grid search to find optimal parameters!!

```
# Prepare data, FastText requires each text to be on one line, and the label prefix is "__label__"
df_sampled['fasttext_format'] = '__label__' + df_sampled['Label'].astype(str) + ' ' + df_sampled['Content']
```

```
def train_and_evaluate(parameter_grid, train_file, validate_file):
    best_model = None
    best_accuracy = 0
    best_params = {}

    for lr in parameter_grid['lr']:
        for epoch in parameter_grid['epoch']:
            for wordNgrams in parameter_grid['wordNgrams']:
                model = fasttext.train_supervised(input=train_file,
                                                  lr=lr,
                                                  epoch=epoch,
                                                  wordNgrams=wordNgrams)
                result = model.test(validate_file)
                accuracy = result[1]

                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    best_model = model
                    best_params = {'lr': lr, 'epoch': epoch, 'wordNgrams': wordNgrams}

    return best_model, best_accuracy, best_params


parameter_grid = {
    'lr': [0.1, 0.5, 1.0],
    'epoch': [5, 10, 20],
    'wordNgrams': [1, 2, 3]
}
```

Split the datasets:

80% training sets
20% testing sets.

Result:
Best Accuracy: 0.8335002503755633
Best Parameters: {'lr': 0.1, 'epoch': 10, 'wordNgrams': 2}

Transformer

Split the datasets:

80% training sets
10% validation set
10% testing sets.

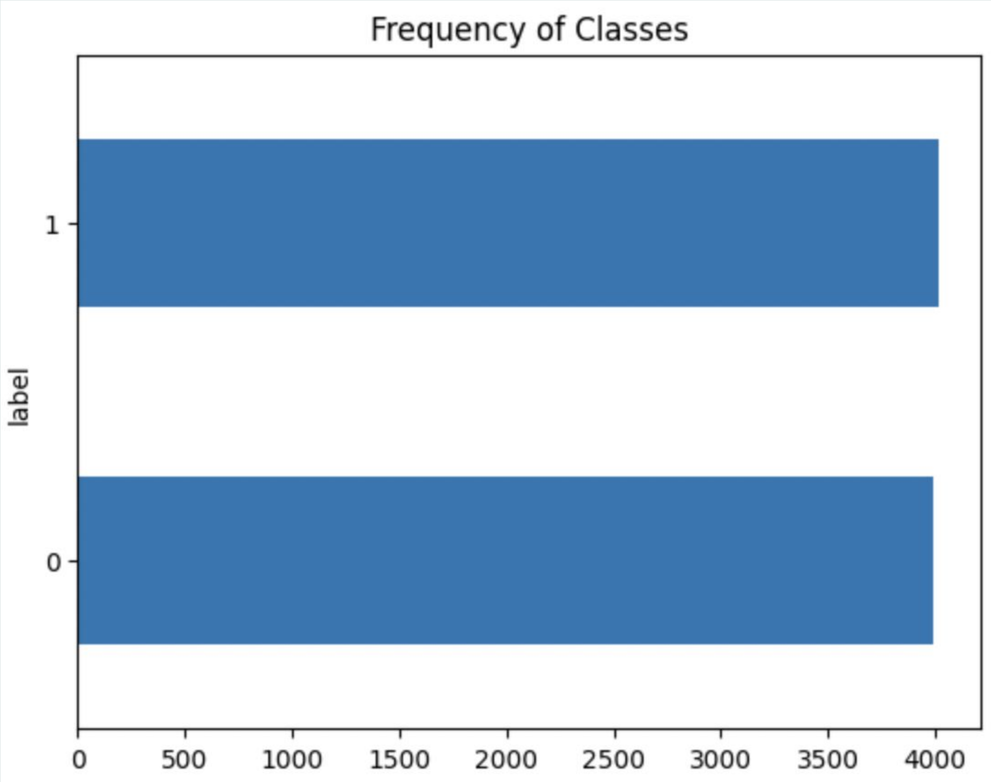
```python
# Divide datasets into training, validation and test set
train_df, temp_df = train_test_split(df, test_size=0.2, random_state=42)
validation_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

# Delete indices
train_df = train_df.reset_index(drop=True)
validation_df = validation_df.reset_index(drop=True)
test_df = test_df.reset_index(drop=True)

# Creat training dataset
train_dataset = Dataset.from_pandas(train_df)

# Create validation dataset
validation_dataset = Dataset.from_pandas(validation_df)

# Create testing dataset
test_dataset = Dataset.from_pandas(test_df)
```

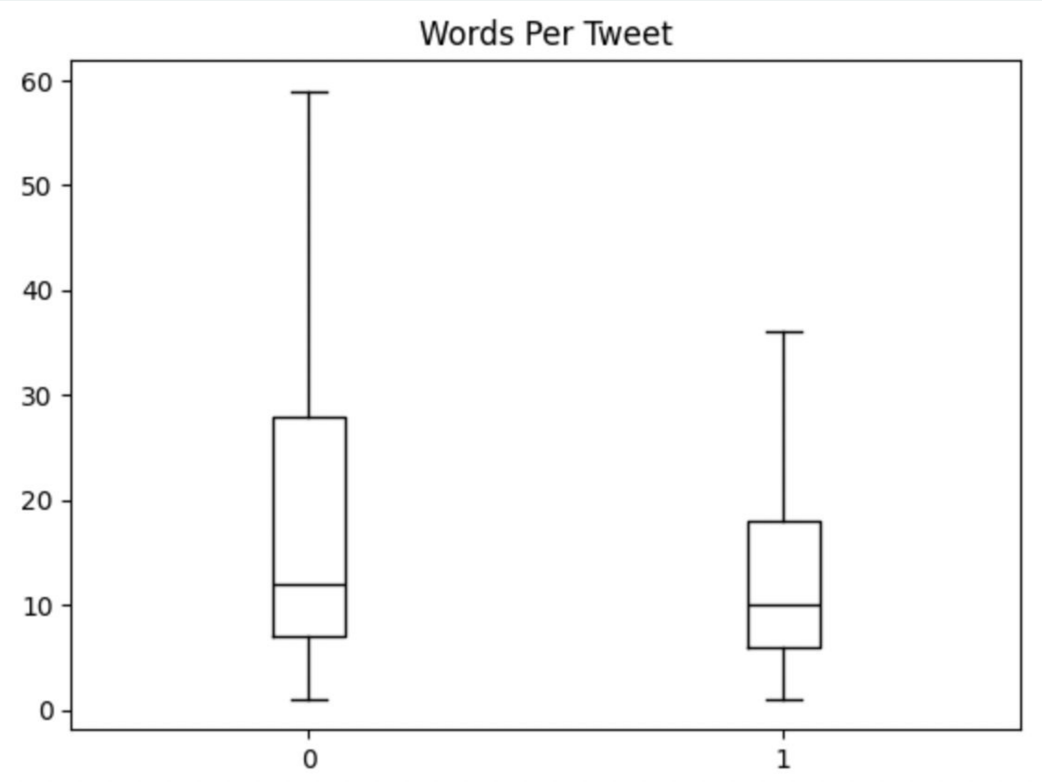
```python
# Create DatasetDict
speeches = DatasetDict({
    'train': train_dataset,
    'validation': validation_dataset,
    'test': test_dataset
})

# print the information of DatasetDict
print(speeches)
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 8000
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 1000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1000
    })
})
```

Transformer

We directly used the dataset saved after previous processing. Here we draw a distribution chart to check the size of the dataset again in different categories, and use box plots to view the length distribution of the text.





Transformer

Tokenization

- Subword tokenization

Test text: "hello this fucking idiot"

```
#example
encoded_text = tokenizer("hello this fucking idiot ")
tokens = tokenizer.convert_ids_to_tokens(encoded_text.input_ids)
encoded_text,tokens
```

```
({'input_ids': [101, 7592, 2023, 8239, 10041, 102], 'attention_mask': [1, 1, 1, 1, 1, 1]},
 ['[CLS]', 'hello', 'this', 'fucking', 'idiot', '[SEP]'])
```

Some special [CLS] and [SEP] tokens have been added to the start and end of the sequence

```
print(tokenizer.convert_tokens_to_string(tokens))
```

```
[CLS] hello this fucking idiot [SEP]
```

Transformer

Tokenization

- Tokenize the whole dataset

```
# Convert individual parts in DatasetDict to DataFrame
train_df = speeches['train'].to_pandas()
validation_df = speeches['validation'].to_pandas()
test_df = speeches['test'].to_pandas()
```

```
# Define a function to vectorize text
def tokenize_texts(df, tokenizer):
    # Initialize two empty lists to store input_ids and attention_mask respectively
    input_ids = []
    attention_masks = []

    # Apply tokenizer to each row of text in the DataFrame
    for text in df['text']:
        #  tokenize the text
        tokenized_text = tokenizer(text, padding=True, truncation=True, max_length=512, return_tensors="pt")
        # Convert the input_ids and attention_mask of the tokenized text into lists and then add them to the cor
        input_ids.append(tokenized_text['input_ids'].squeeze().tolist())  # Use squeeze() to remove unnecessary
        attention_masks.append(tokenized_text['attention_mask'].squeeze().tolist())

    # Add the input_ids and attention_mask lists as new columns to the DataFrame
    df['input_ids'] = input_ids
    df['attention_mask'] = attention_masks
    return df

# Apply tokenize_texts function and re-convert the DataFrame
train_df_encode = tokenize_texts(train_df, tokenizer)
validation_df_encode = tokenize_texts(validation_df, tokenizer)
test_df_encode = tokenize_texts(test_df, tokenizer)
```

Transformer

Tokenization

- Tokenize the whole dataset

```
train_df_encode
```

	text	label	input_ids	attention_mask
0	matter us guy bought everi book cheat two fuck...	1	[101, 3043, 2149, 3124, 4149, 2412, 2072, 2338...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
1	canon recal due allerg reaction caus rubber ma...	0	[101, 9330, 28667, 2389, 2349, 2035, 2121, 229...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
2	american technolog normal censorship promiscu ...	0	[101, 2137, 21416, 21197, 3671, 15657, 20877, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
3	realli dont realli care peopl like friend slut...	1	[101, 2613, 3669, 2123, 2102, 2613, 3669, 2729...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	listen go mood joke pleas shut hell	1	[101, 4952, 2175, 6888, 8257, 22512, 3844, 310...	[1, 1, 1, 1, 1, 1, 1, 1, 1]
...	...	...	...	...
7995	help go along see seem better slightli sober t...	1	[101, 2393, 2175, 2247, 2156, 4025, 2488, 7263...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
7996	gay porn user style color border solid row sty...	1	[101, 5637, 22555, 5310, 2806, 3609, 3675, 502...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7997	jehochman appear work misunderstand index arti...	1	[101, 15333, 6806, 19944, 3711, 2147, 28616, 2...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7998	see first love bowl alley hide bathroom whole ...	0	[101, 2156, 2034, 2293, 4605, 8975, 5342, 5723...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
7999	get close hand argument today alreadi mostli m...	1	[101, 2131, 2485, 2192, 6685, 2651, 2632, 1641...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

8000 rows × 4 columns

Transformer ------ Training

```python
model_ckpt = "distilbert-base-uncased"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

Extract hidden layers:

```python
#Define a function to extract all the hidden states
def extract_hidden_states(input_ids, attention_mask, model):
    # Convert input data to tensor and move it to the correct device
    input_ids = torch.tensor(input_ids).to(device)
    attention_mask = torch.tensor(attention_mask).to(device)

    # Extract last hidden states
    with torch.no_grad():
        outputs = model(input_ids.unsqueeze(0), attention_mask=attention_mask.unsqueeze(0))
        # Return vector for [CLS] token
        hidden_state = outputs.last_hidden_state[:, 0, :].cpu().numpy()

    return hidden_state
```

```python
# Apply it in tensor
```

```python
#This method is used for smaller data sets and does not require the batch method, which puts
def apply_extract_hidden_states(row):
    input_ids = row['input_ids']
    attention_mask = row['attention_mask']
    hidden_state = extract_hidden_states(input_ids, attention_mask, model)
    # Only want to keep the first element in hidden state as an example
    return hidden_state[0]
# Add a new column to store the hidden state of each sample
train_df_encode['hidden_state'] = train_df_encode.apply(apply_extract_hidden_states, axis=1)
```

```python
train_df_encode['hidden_state']
```

```
0       [0.09061651, 0.13126986, 0.00037594105, -0.138...
1       [-0.35385618, -0.08571001, -0.013041183, -0.24...
2       [-0.05068834, 0.06903824, -0.1220773, -0.12001...
3       [-0.20787732, -0.010285065, 0.12884371, -0.261...
4       [0.07289262, 0.12185082, 0.12452027, -0.101105...
                              ...
7995    [-0.027486656, -0.07262253, 0.14432935, -0.116...
7996    [-0.16791086, -0.053512815, 0.025440352, -0.16...
7997    [-0.20645311, -0.07303833, -0.22055058, 0.0110...
7998    [-0.122590944, -0.07989224, -0.008668961, -0.2...
7999    [-0.12268316, 0.053220443, -0.099533305, -0.48...
Name: hidden_state, Length: 8000, dtype: object
```

Transformer ------ Training

train_df_encode

	text	label	input_ids	attention_mask	hidden_state
0	matter us guy bought everi book cheat two fuck...	1	[101, 3043, 2149, 3124, 4149, 2412, 2072, 2338...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	[0.09061651, 0.13126986, 0.00037594105, -0.138...
1	canon recal due allerg reaction caus rubber ma...	0	[101, 9330, 28667, 2389, 2349, 2035, 2121, 229...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	[-0.35385618, -0.08571001, -0.013041183, -0.24...
2	american technolog normal censorship promiscu ...	0	[101, 2137, 21416, 21197, 3671, 15657, 20877, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	[-0.05068834, 0.06903824, -0.1220773, -0.12001...
3	realli dont realli care peopl like friend slut...	1	[101, 2613, 3669, 2123, 2102, 2613, 3669, 2729...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	[-0.20787732, -0.010285065, 0.12884371, -0.261...
4	listen go mood joke pleas shut hell	1	[101, 4952, 2175, 6888, 8257, 22512, 3844, 310...	[1, 1, 1, 1, 1, 1, 1, 1, 1]	[0.07289262, 0.12185082, 0.12452027, -0.101105...
...	...	...	...	...	...
7995	help go along see seem better slightli sober t...	1	[101, 2393, 2175, 2247, 2156, 4025, 2488, 7263...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	[-0.027486656, -0.07262253, 0.14432935, -0.116...
7996	gay porn user style color border solid row sty...	1	[101, 5637, 22555, 5310, 2806, 3609, 3675, 502...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	[-0.16791086, -0.053512815, 0.025440352, -0.16...
7997	jehochman appear work misunderstand index arti...	1	[101, 15333, 6806, 19944, 3711, 2147, 28616, 2...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	[-0.20645311, -0.07303833, -0.22055058, 0.0110...
7998	see first love bowl alley hide bathroom whole ...	0	[101, 2156, 2034, 2293, 4605, 8975, 5342, 5723...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	[-0.122590944, -0.07989224, -0.008668961, -0.2...
7999	get close hand argument today alreadi mostli m...	1	[101, 2131, 2485, 2192, 6685, 2651, 2632, 1641...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	[-0.12268316, 0.053220443, -0.099533305, -0.48...

8000 rows × 5 columns

Transformer ------ Training

```
# create a feature matrix
train_xs = np.array(list(train_df_encode['hidden_state']))

# Create an array of labels
train_ys = np.array(train_df_encode['label'])

# check the shape
print(train_xs.shape)
print(train_ys.shape)
```

```
(8000, 768)
(8000,)
```

```
train_ys
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

Creating a feature matrix

```
# For validation set
valid_xs = np.array(list(validation_df_encode['hidden_state']))
valid_ys = np.array(validation_df_encode['label'])

# For testing set
test_xs = np.array(list(test_df_encode['hidden_state']))
test_ys = np.array(test_df_encode['label'] if 'label' in test_df_encode.columns else [])

# Check shape
print(valid_xs.shape)
print(valid_ys.shape)
print(test_xs.shape)
```

```
(1000, 768)
(1000,)
(1000, 768)
```

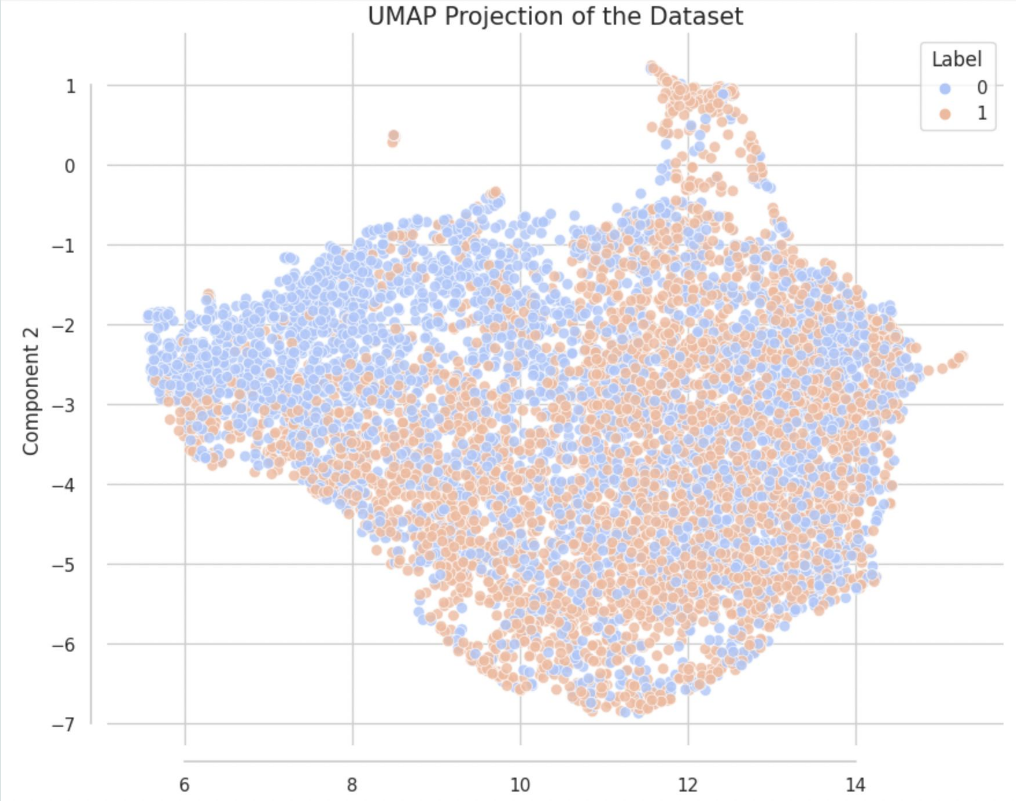
Transformer ------ Training

Visualizing the dataset

Visualizing the hidden state in 768 dimensions is tricky, we will use the powerful UMAPfootnote algorithm to project the vectors into 2D. Since UMAP works best when scaling features to lie within the interval [0,1], we will first apply a MinMaxScaler and then use the umap-learn UMAP implementation from the library to reduce the hidden states:

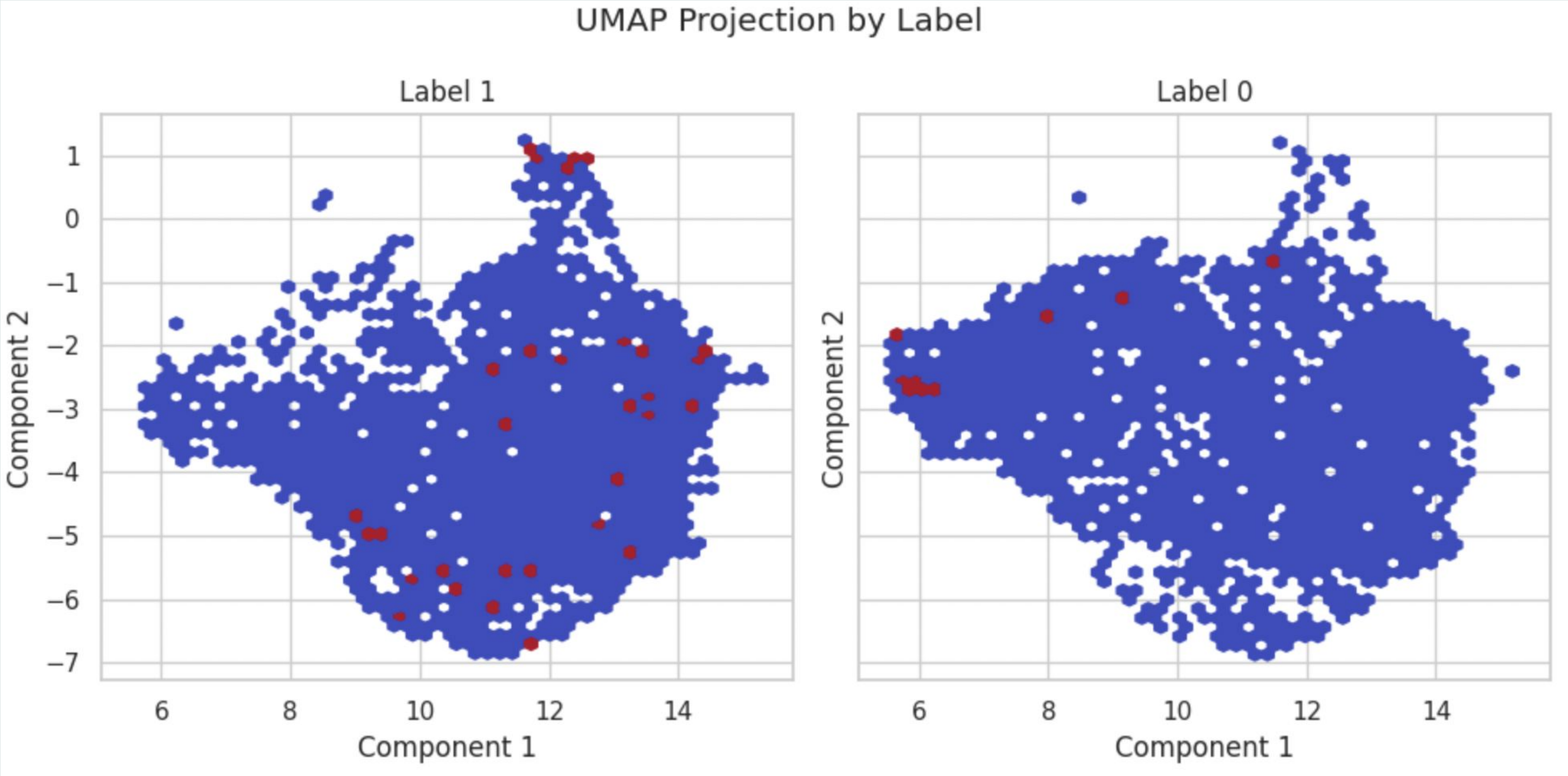
```python
# Scale features to [0,1] range
scaled_xs = preprocessing.MinMaxScaler().fit_transform(train_xs)
# Initialize and fit UMAP
mapper = UMAP(n_components=2, metric="cosine").fit(scaled_xs)
# Create a DataFrame of 2D embeddings
df_emb = pd.DataFrame(mapper.embedding_, columns=["X", "Y"])
df_emb["label"] = train_ys
df_emb.head()
```

	X	Y	label
0	10.855015	-5.579407	1
1	13.744467	-1.453261	0
2	8.038956	-3.498520	0
3	9.617622	-6.344347	1
4	12.086096	-5.394647	1



Transformer ------ Training

Visualizing the dataset



Transformer ------ Prediction

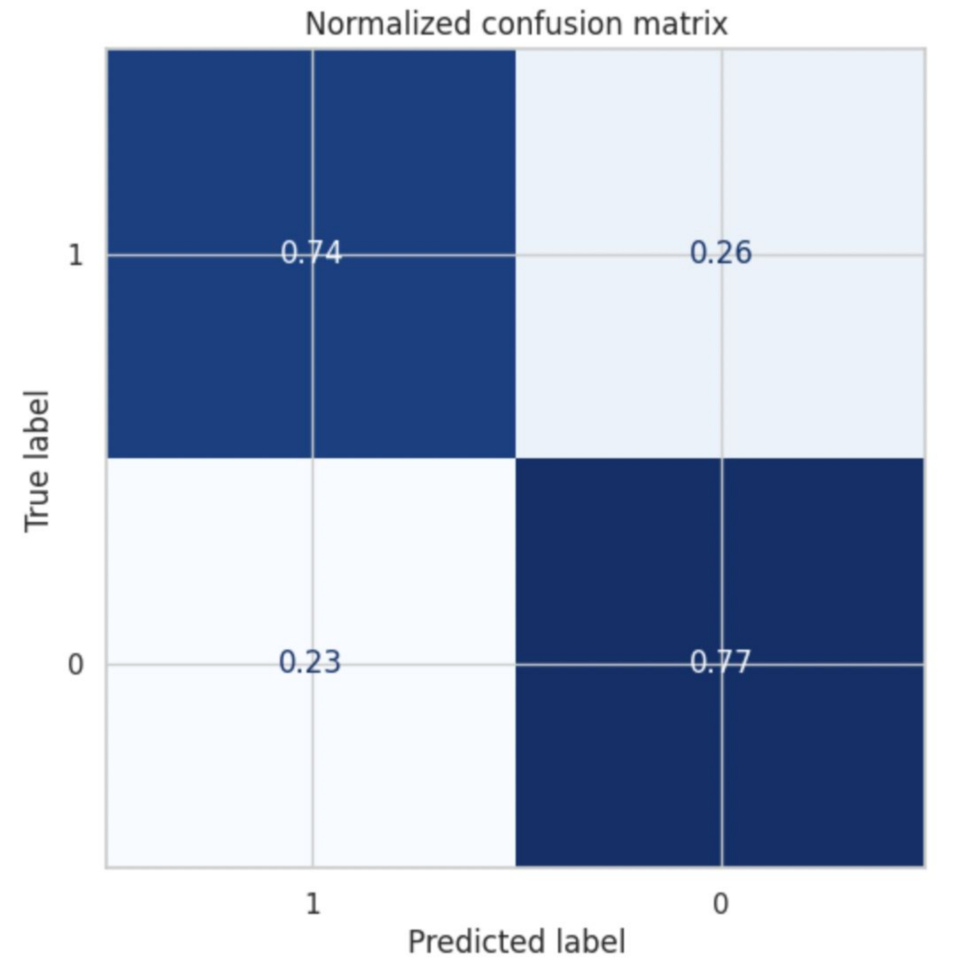
Use simple methods like logistic regression and dummy classifier to do the prediction

```
# Train logistic regression model
from sklearn import linear_model, metrics, dummy
import matplotlib.pyplot as plt

lr_clf = linear_model.LogisticRegression(max_iter=3000)
lr_clf.fit(train_xs, train_ys)

# evaluate model on validation set
accuracy = lr_clf.score(valid_xs, valid_ys)
print(f"Validation Accuracy: {accuracy}")
```

```
Validation Accuracy: 0.752
```

```
#Compare with baseline model
# Use the most frequently used class as the baseline
dummy_clf = dummy.DummyClassifier(strategy="most_frequent")
dummy_clf.fit(train_xs, train_ys)
baseline_accuracy = dummy_clf.score(valid_xs, valid_ys)
print(f"Baseline Accuracy: {baseline_accuracy}")
```

```
Baseline Accuracy: 0.483
```



Transformer

```python
import numpy as np

# Extract predictions and label_ids
predictions = np.array(preds_output.predictions)
label_ids = np.array(preds_output.label_ids)

# Calculate predicted labels: choose the category with the
predicted_labels = np.argmax(predictions, axis=1)

# Calculate accuracy
accuracy = np.mean(predicted_labels == label_ids)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.772
```

Thank you