

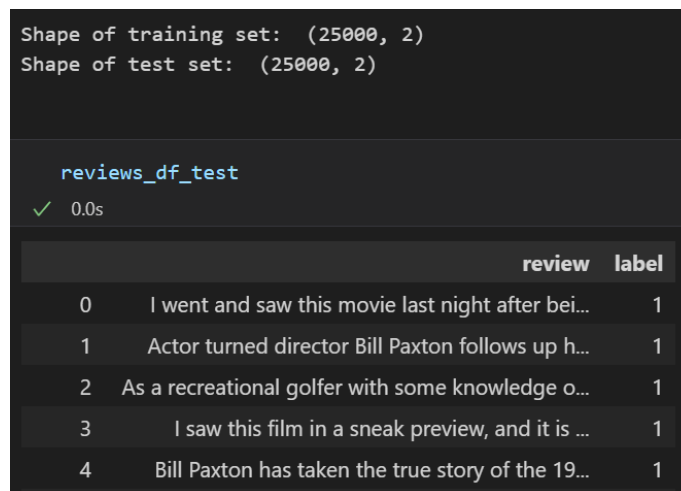
Machine Learning Algorithm Description

Introduction

The purpose of this project was to use natural language processing and machine learning techniques learned during ENEL 645 on a data set from Stanford containing 50,000 IMDb movie reviews. The goal of the model is to be able to predict the sentiment of a users review as either positive or negative. The dataset provided contains two folders one labelled as training and another labelled as testing. Within each of those folders there is a positive and negative folder with each of those containing 12,500 .txt files with the reviews.

Loading Data

The .txt files were loaded into a Pandas data frame by using python to parse through each .txt file and reading in the user review from the .txt file. The user reviews were stored in either a training or testing data frame depending on if they came from the training or testing folder. The labels (1 or 0 for positive and negative, respectively) were then added to the next column beside review. The training and test set each contain 25,000 reviews, with 12,500 of them being positive and 12,500 being negative.



```
Shape of training set: (25000, 2)
Shape of test set: (25000, 2)

reviews_df_test
✓ 0.0s
```

	review	label
0	I went and saw this movie last night after bei...	1
1	Actor turned director Bill Paxton follows up h...	1
2	As a recreational golfer with some knowledge o...	1
3	I saw this film in a sneak preview, and it is ...	1
4	Bill Paxton has taken the true story of the 19...	1

Figure 1: The first 5 rows of the test set data frame.

Data Preprocessing

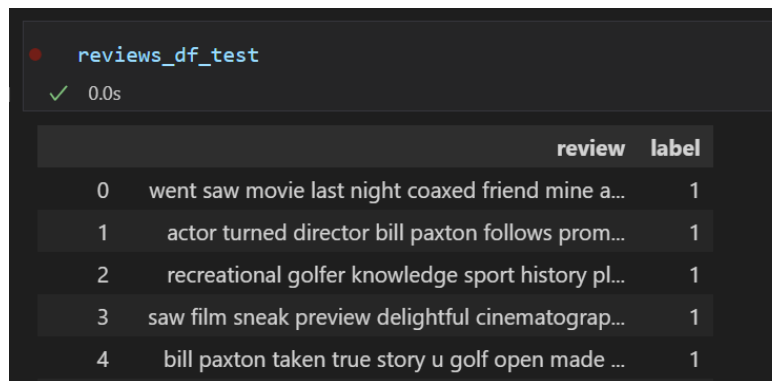
The reviews were then preprocessed by passing them to a function that completes the following data cleaning processes:

- Tokenization: This converts the user review text into smaller units to be processed (typically individual words)
- Stop Word Removal: Remove common words that don't typically provide much information regarding sentiment. Words such as "it", "me", "as", "but", etc. will be removed leaving more meaningful words.
- Lemmatization: The tokenized remaining words were then lemmatized which involves converting words to their root words based on context. For example, "caring" would be converted to "care".

This helps the model with feature extraction and reducing the vocabulary needed to be learned by the model.

- Removal of special characters and punctuation: Special characters such as “@\$%” or other are removed as they aren’t needed to predict sentiment. Additionally, punctuation is also removed for the same reason.
- Lower case: All words in the reviews are converted to lower case so that the model doesn’t isn’t required to model the same words with different cases.
- Finally, the words or tokens are joined back together and saved in the review column.

Below is the same reviews as in Figure 1 except with preprocessing complete.



The screenshot shows a Jupyter Notebook cell with the variable `reviews_df_test` and a green checkmark indicating successful execution. Below the code, a table displays the first five rows of the test set. The table has two columns: `review` and `label`. The reviews are all positive, as indicated by the `label` column containing the value 1.

	review	label
0	went saw movie last night coaxed friend mine a...	1
1	actor turned director bill paxton follows prom...	1
2	recreational golfer knowledge sport history pl...	1
3	saw film sneak preview delightful cinematograp...	1
4	bill paxton taken true story u golf open made ...	1

Figure 2: The first 5 rows of the test set with preprocessing complete.

Vectorization

For a computer to model the data (user reviews) it must have features that it understands. In NLP words are the input features and so they need to be turned into numbers, through a process called vectorization. Bag of Words is a common vectorizer that counts the frequency of words in a text and then creates a vector based on the words. Then by knowing the vectors that result in a positive or negative sentiment from the data sets (i.e. supervised learning) the model can learn this and make predictions on user reviews that are not labelled positive or negative.

4. Vectorization

```
# Use Bag of Words to vectorize the reviews
from sklearn.feature_extraction.text import CountVectorizer

# Create the Bag of Words model
bow = CountVectorizer()

# Transform the training and test sets
X_train = bow.fit_transform(reviews_df_train["review"])
X_test = bow.transform(reviews_df_test["review"])

# Get the labels
y_train = reviews_df_train["label"]
y_test = reviews_df_test["label"]

# Print the shapes of the training and test sets
print("Shape of training set: ", X_train.shape)
print("Shape of test set: ", X_test.shape)
print("Shape of training labels: ", y_train.shape)
print("Shape of test labels: ", y_test.shape)
```

✓ 2.9s

```
Shape of training set: (25000, 63875)
Shape of test set: (25000, 63875)
Shape of training labels: (25000,)
Shape of test labels: (25000,)
```

Figure 3: Bag-of-Words vectorization code.

Above you can see that the training and testing set both have 25,000 rows and 63,875 columns with each column representing a word. Each sample will have a value in each column representing the presence and frequency of that word in that sample.

Data Split

It was decided to use the 25,000 reviews in the training folder for training only. The 25,000 reviews in the test folder were used for validation/testing only. Typically, we use 70% of the total data for training and 30% for testing but in this case because the data was split before we received it and we have no information from the client. We do not want to mix the data sets and re-split because perhaps the testing set data was chosen for a specific reason. Alternatively, if we wanted a 70%/30% split we could have extracted 15,000 reviews from the test set to become a new test set and then added the remaining 10,000 reviews to the training set. As we have 25,000 reviews in the training set, which is a large dataset, it did not seem necessary and the test set of 25,000 reviews and the training set of 25,000 reviews were used as intended from the “client”.

Model Type and Architecture

Next the model is trained on the vectorized data previously created. In this project we are using a neural network to predict the sentiment of the user reviews. A neural network was chosen over a traditional

machine learning model as they can effectively capture complex problems through feature extraction. Neural networks are also good at modeling non-linear data which NLP is. Furthermore, a neural network was also chosen for their ability to not require much feature engineering. Below is the neural network architecture:

```
# Create the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation="relu", input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.6),
    tf.keras.layers.Dense(8, activation="relu"),
    tf.keras.layers.Dropout(0.6),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
✓ 0.1s

# Compile the model
mod_1 = model.compile(tf.keras.optimizers.Adam(learning_rate = 0.001),
                      loss="binary_crossentropy",
                      metrics=["accuracy"])

# Train the model
mod_1 = model.fit(X_train, y_train, epochs=5, batch_size=128, validation_data=(X_test, y_test), verbose=2)
✓ 1m 34.8s
```

Figure 4: Neural network architecture.

In neural networks there are several hyperparameters that must be chosen to produce a good model. Some of the major hyperparameters tuned in this model are:

- **Layers and Neurons** – These can be specified by the machine learning engineer and are chosen to extract features of varying complexity.
- **Activation Function** – Typically Relu is chosen as it is commonly used and one of the best performing activation functions. In a categorical problem the last layer cannot be Relu as it is a linear function, and the model must predict the class in the last layer (not linear). In this case the sigmoid activation function was chosen which can predict binary classifications (i.e. positive or negative)
- **Epoch** – Epoch is the number of time that the model will train on the data set and update the parameters. Generally, with each epoch the model becomes more accurate although it will plateau and is computationally heavy to increase.
- **Learning Rate** – The model uses a loss function to determine the variance from the actual value from what it predicted. Minimizing the loss function will provide a more accurate model. The learning rate is used to adjust the iteration distance between each epoch and can help find the minimum of the loss function.
- **Regularization** – This involves techniques that can prevent the model from overfitting the training data and poorly generalizing to new unseen data. Dropout was used to deal with overfitting. Dropout randomly turns off certain neurons during testing forcing the model to not focus too much on any one extracted feature.

ENEL 645: Final Project

Sam Rainbow

UCID: 30084292

You can see in Figure 4 that there are 2 layers with the first layer containing 16 neurons and using a relu activation function. Dropout is applied to 60% of the neurons during each training update in the first layer. The next layer has 8 neurons and there is also a dropout of 60% on this layer. The last layer has one neuron since the prediction is binary. The sigmoid activation function is used and provides a number for each sample between 0 and 1 with sentiment of 0.5 and above considered positive.

Results

During training, the initial iterations of architecture contained more layers and neurons, but the training accuracy was 99% while the testing accuracy was 88% indicating high variance. As such, less neurons and layers were used, and dropout was added to prevent overfitting the model and allowing for better generalization.

Below is the final model compilation using the architecture outlined in Figure 4 the following training accuracy and validation accuracy was achieved after 5 epochs. Training accuracy was 89.55% and the validation accuracy was 87.38%. This falls within the required accuracy outlined in the project requirements.

```
# Compile the model
mod_1 = model.compile(tf.keras.optimizers.Adam(learning_rate = 0.001),
                      loss="binary_crossentropy",
                      metrics=["accuracy"])

# Train the model
mod_1 = model.fit(X_train, y_train, epochs=5, batch_size=128, validation_data=(X_test, y_test), verbose=2)
✓ 1m 34.8s
```

Epoch	1/5	2/5	3/5	4/5	5/5
196/196	15s	17s	24s	20s	19s
loss	0.5941	0.4432	0.3771	0.3305	0.2910
accuracy	0.6816	0.8136	0.8566	0.8795	0.8955
val_loss	0.4174	0.3331	0.3077	0.3144	0.3255
val_accuracy	0.8720	0.8788	0.8775	0.8703	0.8738
15s/epoch	75ms/step	85ms/step	124ms/step	101ms/step	97ms/step

Figure 5: Model performance per epoch.

Below is the above data as a graph with accuracy on the y-axis and epoch on the x axis for both the training and validation.

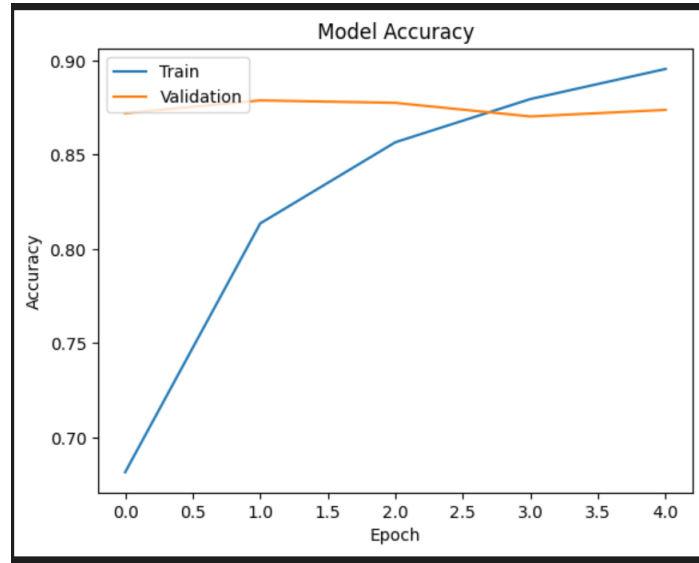


Figure 6: Training and validation accuracy vs epoch.

We can see that with more epochs the model's accuracy on the training set increases but the validation accuracy is high after the first epoch and slightly decreases by the last epoch. This is likely because of the large amount of data in the training set. After 1 epoch the model is trained well and can already predict the sentiment of the validation set with 87.2%. In the next 4 epochs the model becomes better at predicting the training but not the validation.

Below is the predicted sentiment for the test set shown in a confusion matrix:

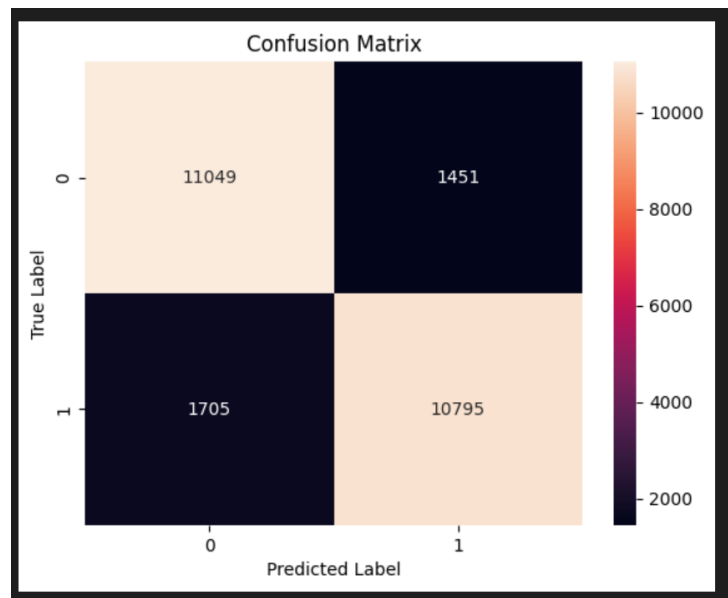


Figure 7: Confusion matrix showing the true label vs the predicted label.

As we can see the model incorrectly predicted 1 (positive) sentiment 1451 times, and incorrectly predicted 0 (negative) sentiment 1705 times. It correctly predicted 0 (negative sentiment) 11049 times and 1 (positive sentiment) 10795 times.