Wine Recommendation System Based on BERT

Lochan Basyal
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, United States
lbasyal@stevens.edu

Abstract—Recommendation systems are critical for selecting the right wine for customers, and this paper presents an exploratory data analysis (EDA) and visualization of wine reviews datasets to gain deeper insights. In addition, this paper demonstrates the use of a multi-label classification model, leveraging a BERT pre-trained model to classify different wine varieties. BERT is a bidirectional multilingual transformer-based model that has consistently achieved state-of-the-art results on various NLP tasks. In this paper, the model has been trained in 4 ways: 16 batch size with 5 epochs, 16 batch size with 10 epochs, 32 batch size with 5 epochs, and 32 batch size with 10 epochs. Among these approaches, the model performed well when training in 16 batch sizes with 10 epochs obtaining an overall accuracy of 84% in the wine datasets. The study showcases the effectiveness of this approach in the wine industry and the recommendation systems.

Index Terms—BERT, EDA, NLP, Recommendation System

INTRODUCTION

This project uses the wine reviews datasets [1] having 130k wine reviews with variety, location, winery, price, and description. This dataset contains 10 columns and 130k rows of wine reviews. The interesting problem here is how we can use this dataset and develop the predictive model to suggest the intent of the wine to the customer as per their interest. To develop this model, we need to work on EDA, visualization, model building, performance observation, and testing of the model.

1. Data Pre-processing and Visualization:

In this section I am presenting my work on data preprocessing and visualization to extract insight in the data from diverse angles, simply speaking I tried to understand the data, which is most important to understand to create the model for recommendation. The shape of the datasets I found is (129971, 14) and I checked the data for duplicates since there is no point in using the same descriptions and titles in the datasets. Afterward, I performed the tasks such as dropping the duplicates, getting to know about the missing values, and imputing the missing values. Then

my data transformed into the shape of (119988, 14) and this process has been shown in the attached Notebook along with this report.

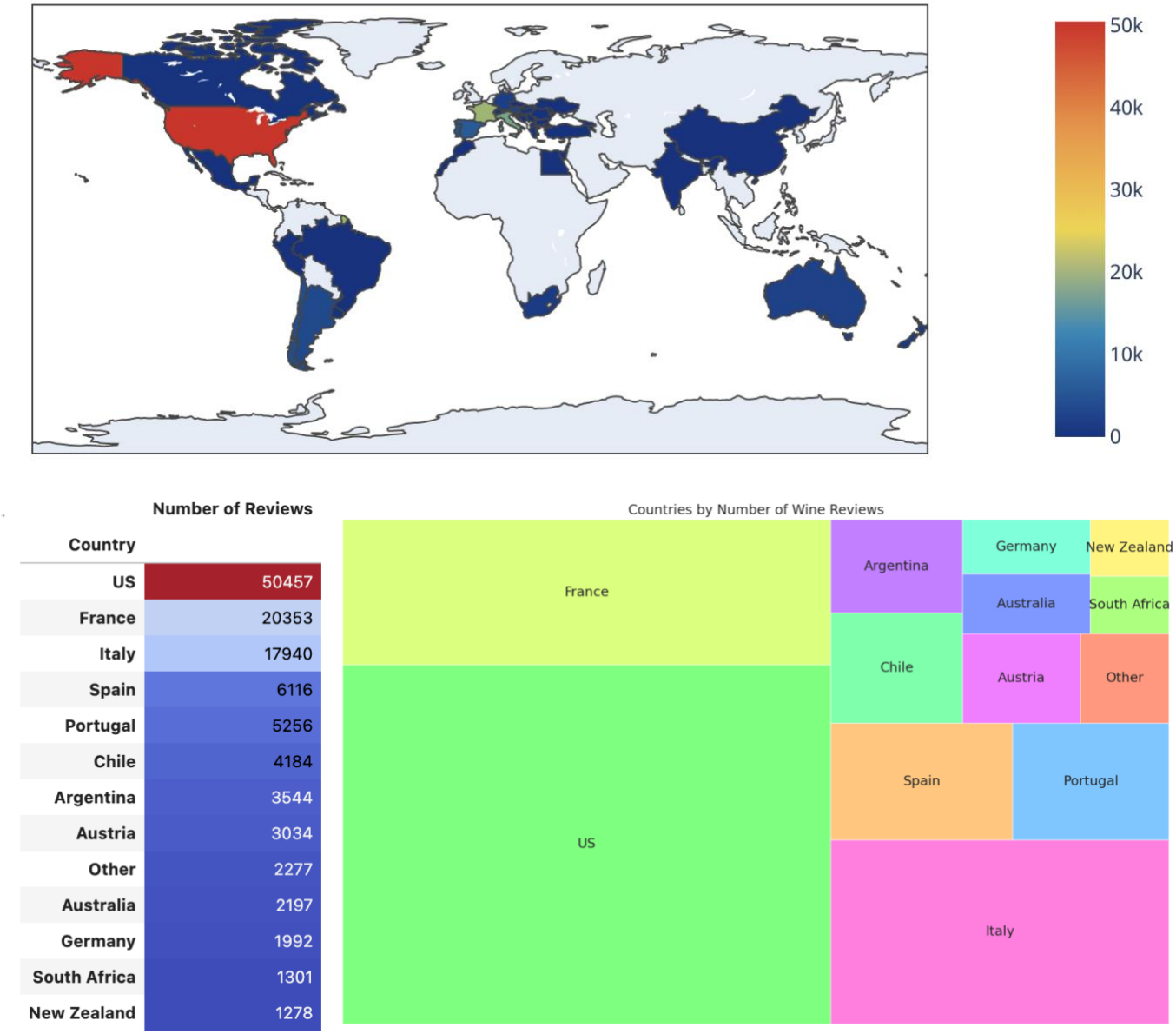

Figure 1: Data Visualization (Wine Reviews by Country)

As per the above visualization, I saw that wine reviews in the US are at the highest level as compared to other countries. Also, I explored the relationship between price and the points in the datasets and I normally found the more points, the wine is expensive. When I visualize the datasets on a variety of wines, I found Bordeaux-style Red Blend as the grapes used for the most expensive wine and chardonnay as the grapes used for the least price wine. Furthermore, France generally produces the most expensive wines while Argentina produces the cheapest wine presented in the figure below.

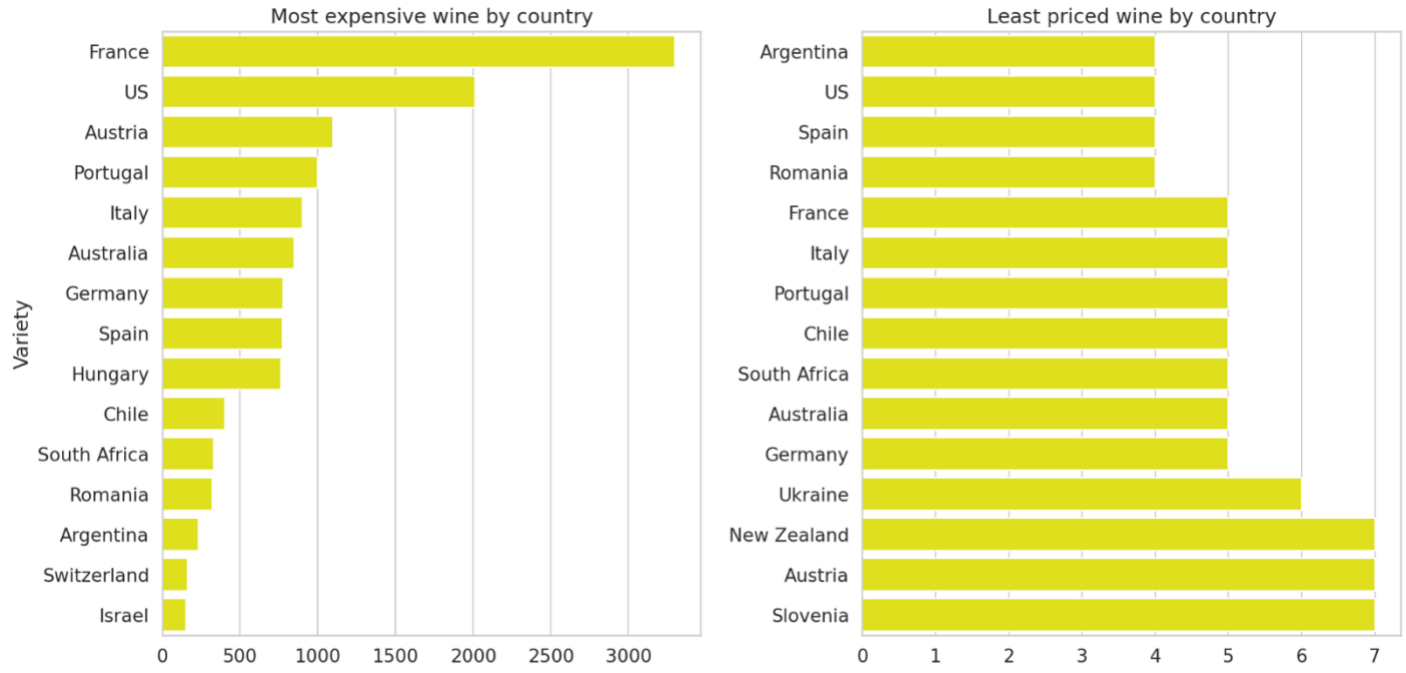

Figure 2: Most Expensive/Least Expensive Wine by Country

When exploring the highest and least-rated wines, Italy produces the most-rated wine and Argentina the least.

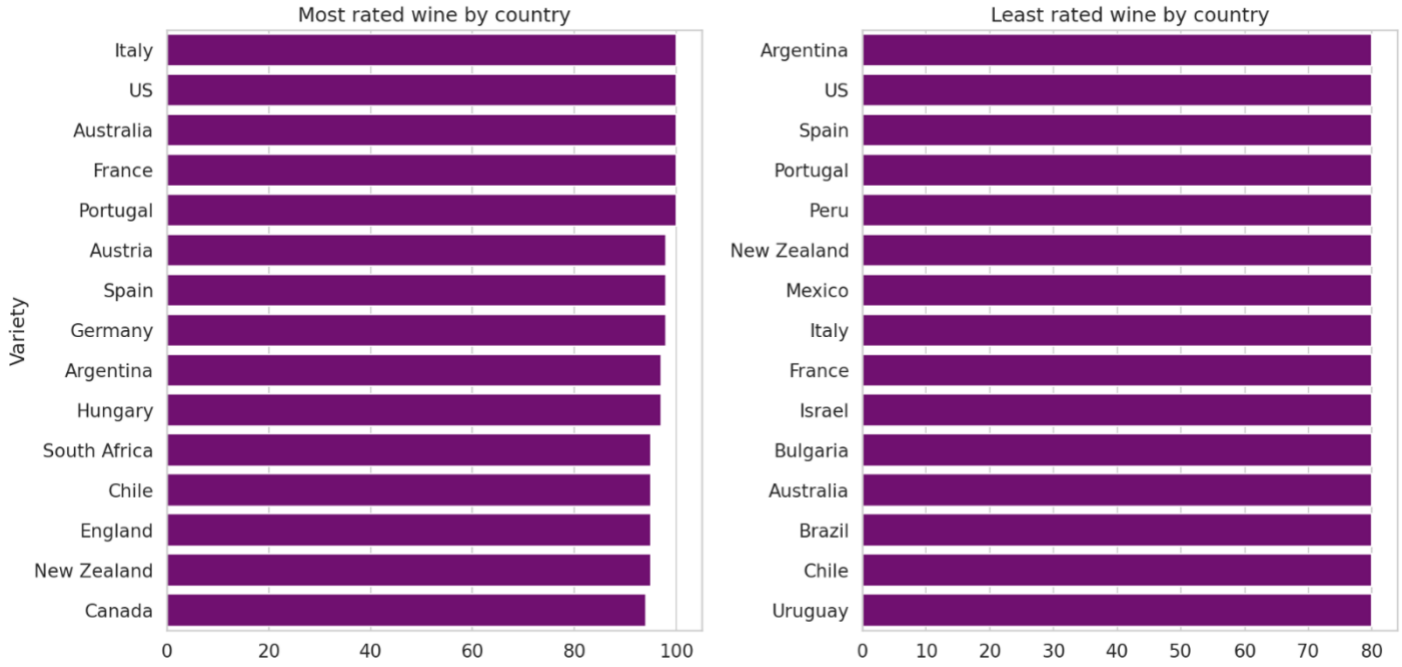

Figure 3: Most Rated/Least Rated Wine by Country

When reviewing the wines in terms of Province, California has the highest number of wine reviews, followed by Washington state. Also, I performed the visualization in Vineyards and wineries and presented the word clouds in terms of Description and Variety which has been presented below.

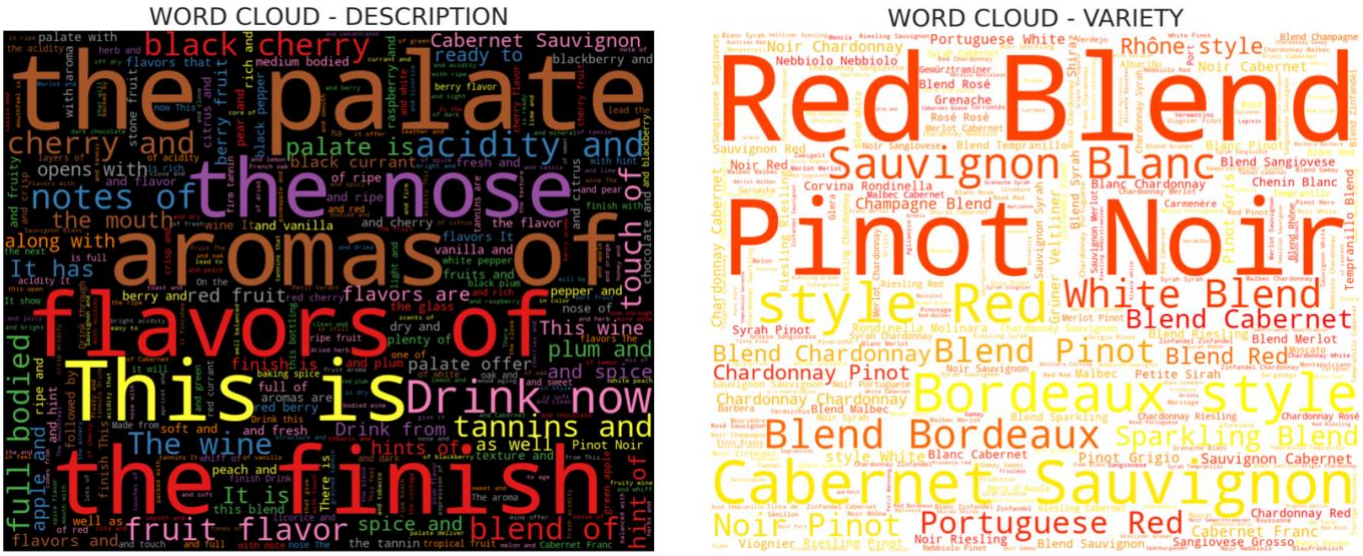

Figure 4: Word Cloud of Description and Variety

2. Model Building and Performance Observation:

In this section, we only required the description and variety columns of the datasets. I already filtered out the duplicate entry on descriptions and titles in the datasets. Here, I am presenting what it looks like after extracting the description and variety columns only.

	description	variety
0	Aromas include tropical fruit, broom, brimston...	White Blend
1	This is ripe and fruity, a wine that is smooth...	Portuguese Red
2	Tart and snappy, the flavors of lime flesh and...	Pinot Gris
3	Pineapple rind, lemon pith and orange blossom ...	Riesling
4	Much like the regular bottling from 2012, this...	Pinot Noir

Table I: Showing Description and Variety of Datasets

For building the model, I took the 7 most described varieties in the datasets which have been shown in the below bar chart.

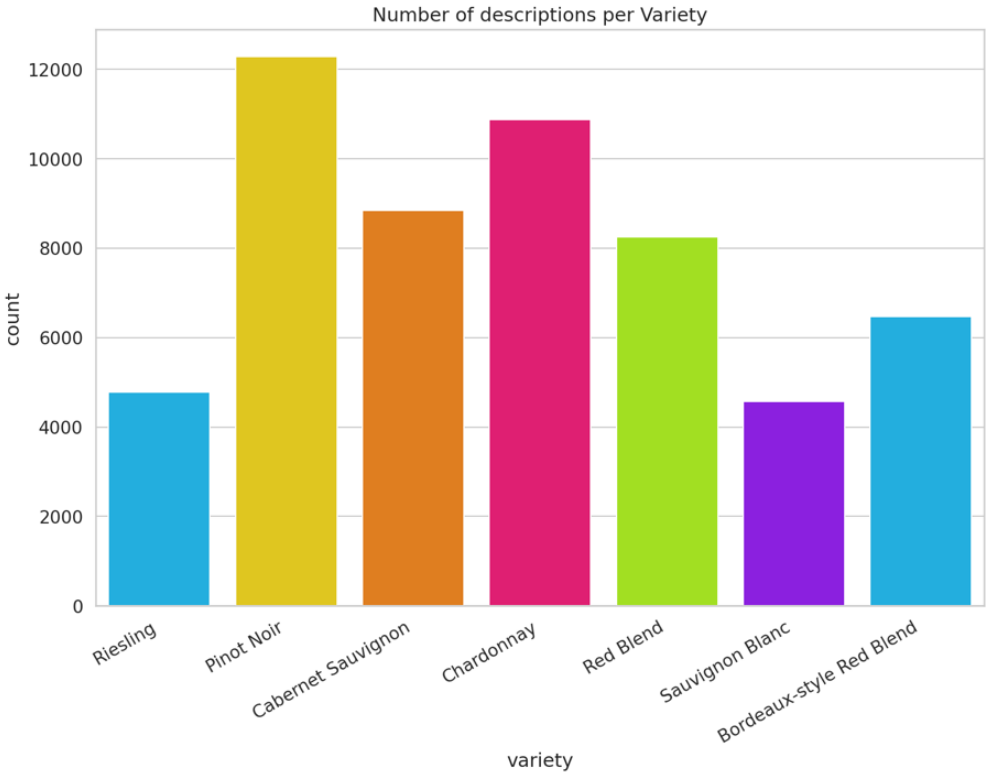

Figure 5: Number of Description Per Variety of Whole Datasets

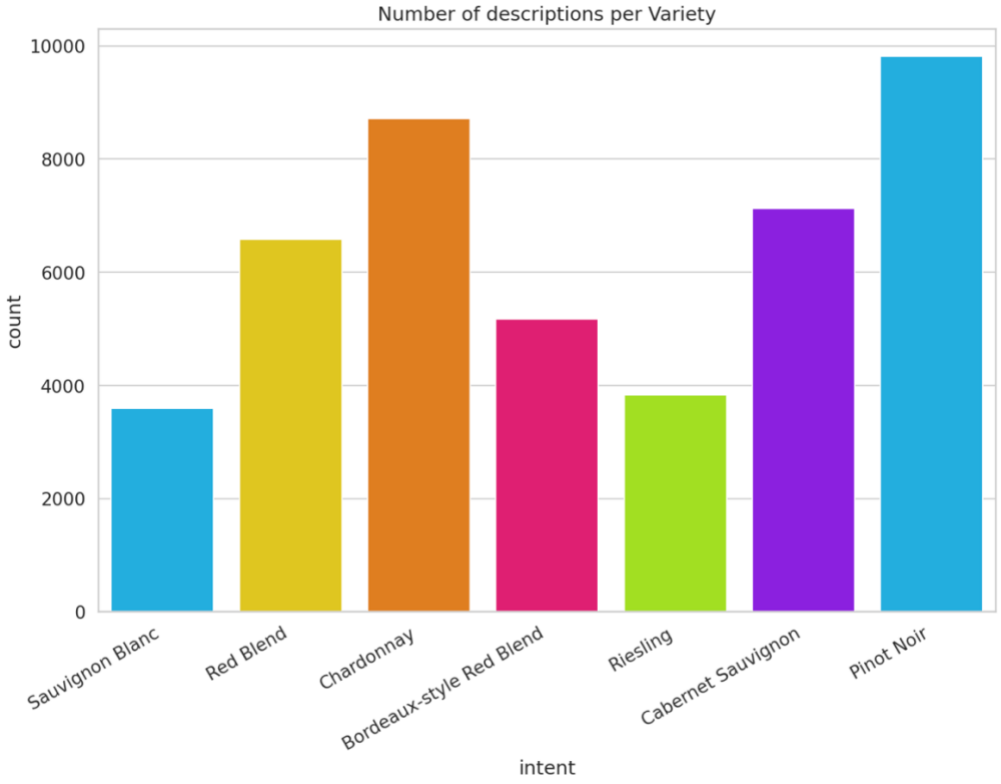

Figure 6: Number of Description Per Variety of Training Datasets

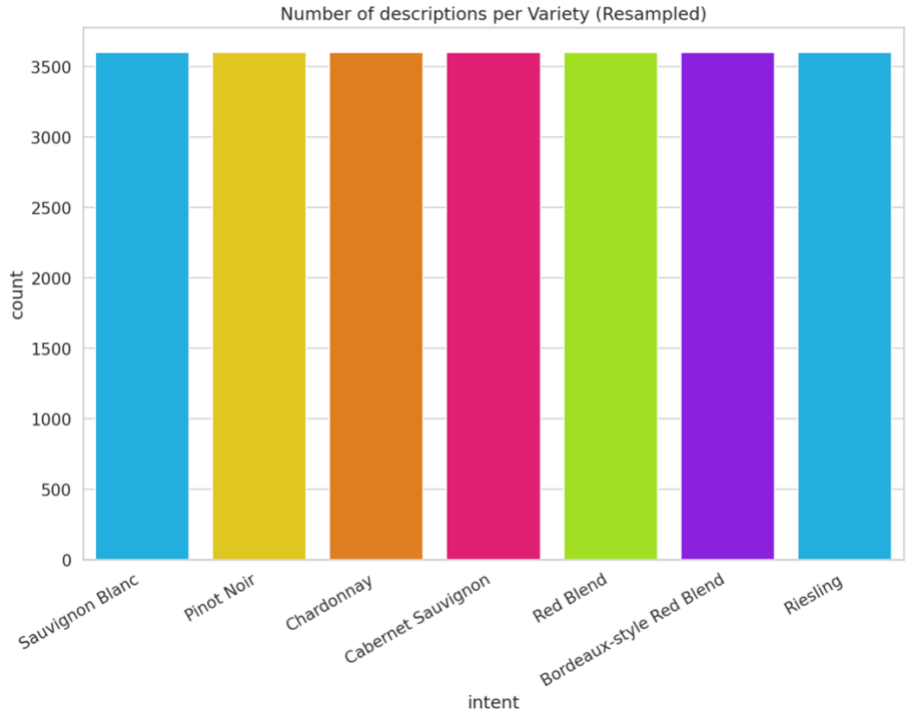

Figure 7: Number of Description Per Variety of Resampled Datasets for Model Training

The datasets have been split with 0.2 as the test size. Also, made the training dataset uniform by taking the wine with the least number of counts (i.e., Sauvignon Blanc), setting the other's variety equal to the least count, and adding all the data together which produce the shape of (25200, 2) data. The data with full tokenization has been fed into the model.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_ids (InputLayer)       [(None, 128)]             0
_________________________________________________________________
bert (BertModelLayer)        (None, 128, 768)          108890112
_________________________________________________________________
lambda_1 (Lambda)            (None, 768)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 768)               0
_________________________________________________________________
dense_2 (Dense)              (None, 768)               590592
_________________________________________________________________
dropout_3 (Dropout)          (None, 768)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 5383
=================================================================
Total params: 109,486,087
Trainable params: 109,486,087
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Model Summary

The presented model is a deep learning model built for text classification tasks, which uses the Bidirectional Encoder Representations from Transformers (BERT) architecture as a pre-trained feature extractor. The model architecture is built in TensorFlow Keras, and it consists of three main layers: an input layer, a BERT model layer, and two dense layers.

The input layer takes a sequence of up to 128 tokens as input, which is the maximum length of the input sequence. The BERT model layer takes the input sequence and applies a pre-trained BERT transformer model to it, generating a sequence of hidden states of dimension 768 for each token in the sequence. The 'BertModelLayer' has '108,890,112' trainable parameters.

The output of the BERT layer is then passed through a lambda layer, which selects the hidden state corresponding to the first token in the sequence, CLS. The CLS token is used as a representation of the entire input sequence, which is then fed into two dense layers with dropout regularization and activation functions. The final dense layer outputs probabilities for each of the 7 classes, which is the number of classes in the classification task.

The model has a total of 109,486,087 parameters, all of which are trainable. These parameters include the weights of the pre-trained BERT model, which are loaded using the 'load_stock_weights' function. The model is optimized using the categorical cross-entropy loss function and the Adam optimization algorithm with a learning rate of 0.00001.

During the model training, the 'fit()' function of the Kera's API was used to train the model using the training data with a ***batch size of 16, for 5 epochs***. The training data was shuffled randomly for each epoch and 10% of the training data was used for validation. A 'TensorBoard' callback was created to log the training progress to a directory defined by the 'log_dir' parameter, with a timestamp for each training run. The training progress was logged with the use of the 'tensorboard_callback' parameter in the 'fit' function.

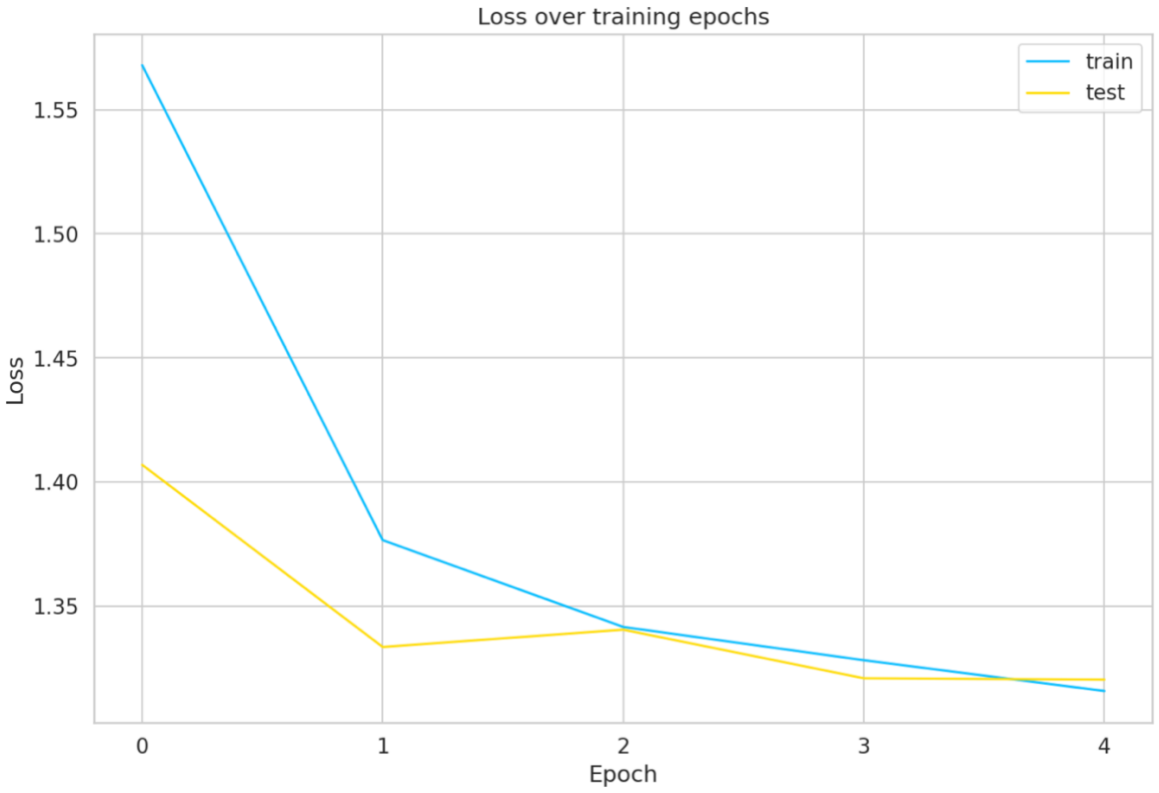

Figure 9: Loss over Training (batch size = 16 and epochs = 5)

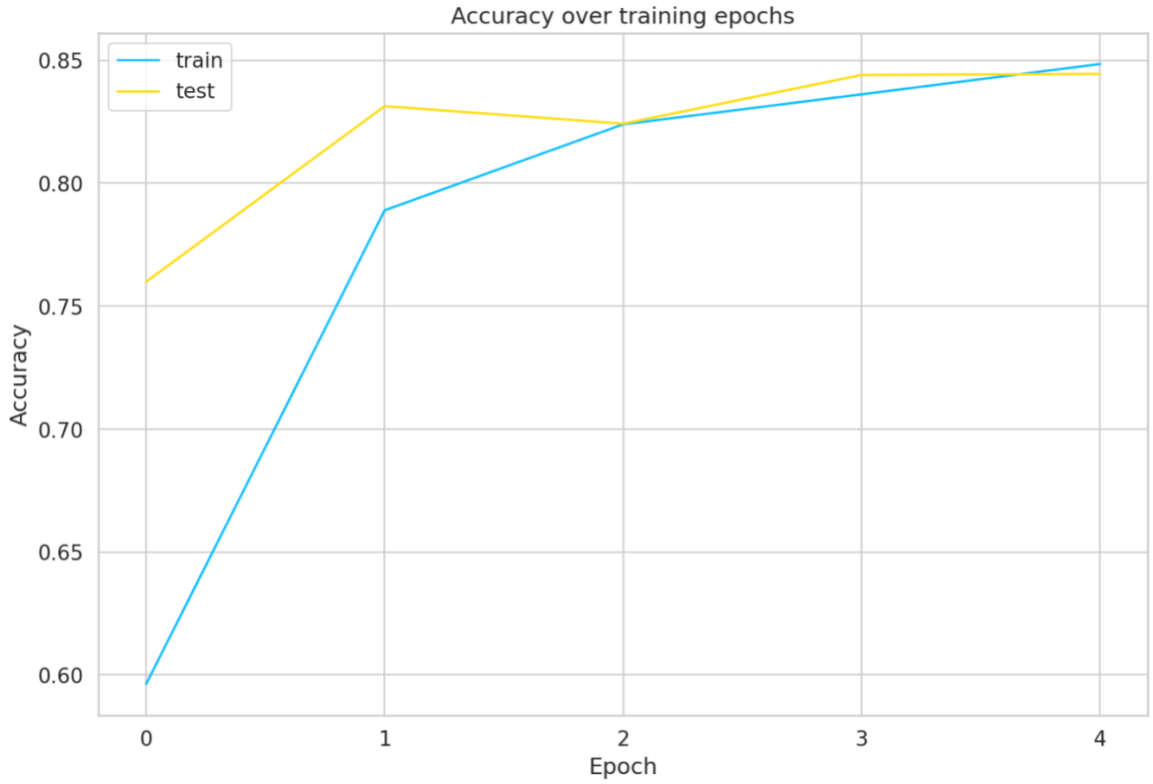

Figure 10: Accuracy over Training (batch size = 16 and epochs = 5)

The above graphical representation between loss and training epochs shows the model is performing well as the training epochs increase. Also, the gap between the training and testing losses is decreasing, indicating that the model is good.

Graphical representation in between accuracy and the training epochs, we can see that the model's accuracy is improving with each epoch, except for a slight decrease in validation accuracy in the later epochs. The training accuracy is increasing rapidly, indicating that the model is learning to fit the training data well. However, we need to be careful to prevent overfitting, as indicated by the slight decrease in validation accuracy in the later epochs. Overall, the accuracy values indicate that the model is improving its performance, but we need to monitor it closely to prevent overfitting and ensure that it generalizes well to unseen data.

From this experiment, I obtained a training accuracy of 86.90 % and a test accuracy of 83.11 % which is considered as good.

	precision	recall	f1-score	support
Pinot Noir	0.81	0.86	0.83	2461
Red Blend	0.92	0.74	0.82	1661
Chardonnay	0.90	0.86	0.88	2155
Cabernet Sauvignon	0.80	0.79	0.79	1717
Riesling	0.89	0.87	0.88	941
Sauvignon Blanc	0.76	0.83	0.80	975
Bordeaux-style Red Blend	0.75	0.89	0.81	1300
accuracy			0.83	11210
macro avg	0.83	0.83	0.83	11210
weighted avg	0.84	0.83	0.83	11210

Figure 11: Classification Report (batch size = 16 and epochs = 5)

This is a classification report that evaluates the performance of a model on a dataset of 11,210 samples. The report shows the precision, recall, F1-score, and support for each class in the dataset, as well as the macro and weighted averages for all the classes. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. A recall is the ratio of correctly predicted positive observations to the total actual positive observations. F1-score is the weighted harmonic mean of precision and recall.

The report shows that the model has an overall accuracy of 83%, meaning that it correctly predicted the class of 83% of the samples in the dataset. The precision, recall, and F1-score values for each class show how well the model performs for each class. For example, the model has high precision, recall, and F1-score for Riesling, which

means that it performs well in this class. However, the model has a lower precision, recall, and F1-score for Cabernet Sauvignon, which means that it performs less well for this class.

The macro and weighted averages show the overall performance of the model across all the classes. The macro average calculates the average precision, recall, and F1 score across all the classes, while the weighted average considers the number of samples in each class. In this case, the macro and weighted averages are the same because the dataset is balanced, meaning that there is roughly the same number of samples for each class.

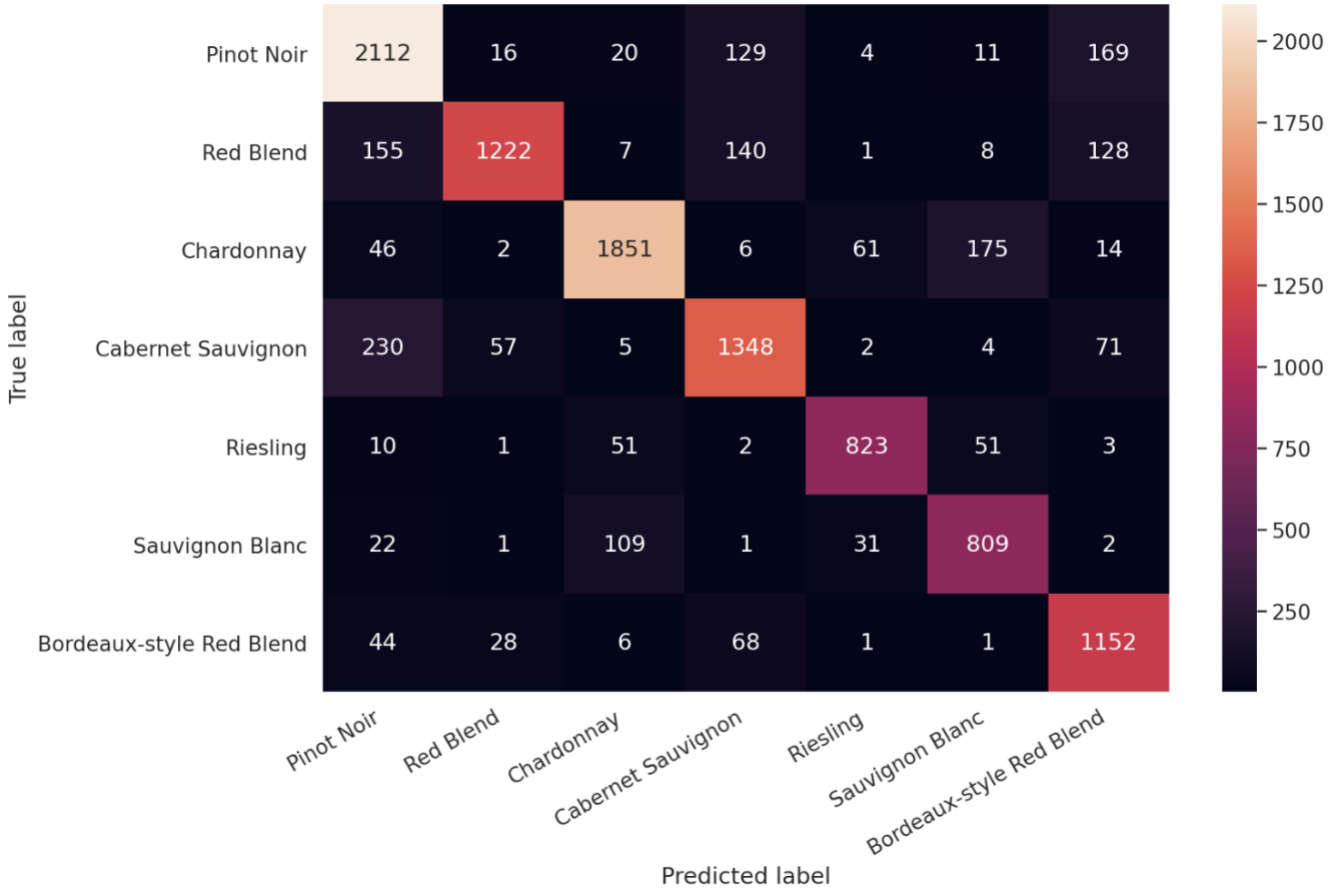

Figure 12: Confusion Matrix (batch size = 16 and epochs = 5)

The diagonal entries represent the number of correctly classified labels, while the off-diagonal entries represent the number of misclassifications. For instance, the cell in row 1, and column 2 indicates that this model predicted 16 samples to be Red Blend, but they were Pinot Noir. Furthermore, 1222 samples are correctly classified as Red Blend but (1661-1222) which is equal to 439 samples were misclassified. These metrics can provide insights into the strengths and weaknesses of the model and can help guide further improvements to the classification algorithm.

```
text: Strong wine made of red grapes
intent: Pinot Noir

text: Grapy plummy and juicy taste
intent: Bordeaux-style Red Blend
```

Figure 13: Testing of the recommendation system (batch size = 16 and epochs = 5)

Similarly, I am presenting the graphical representation for the remaining cases,

When model training with (batch size = 16 and epochs = 10)

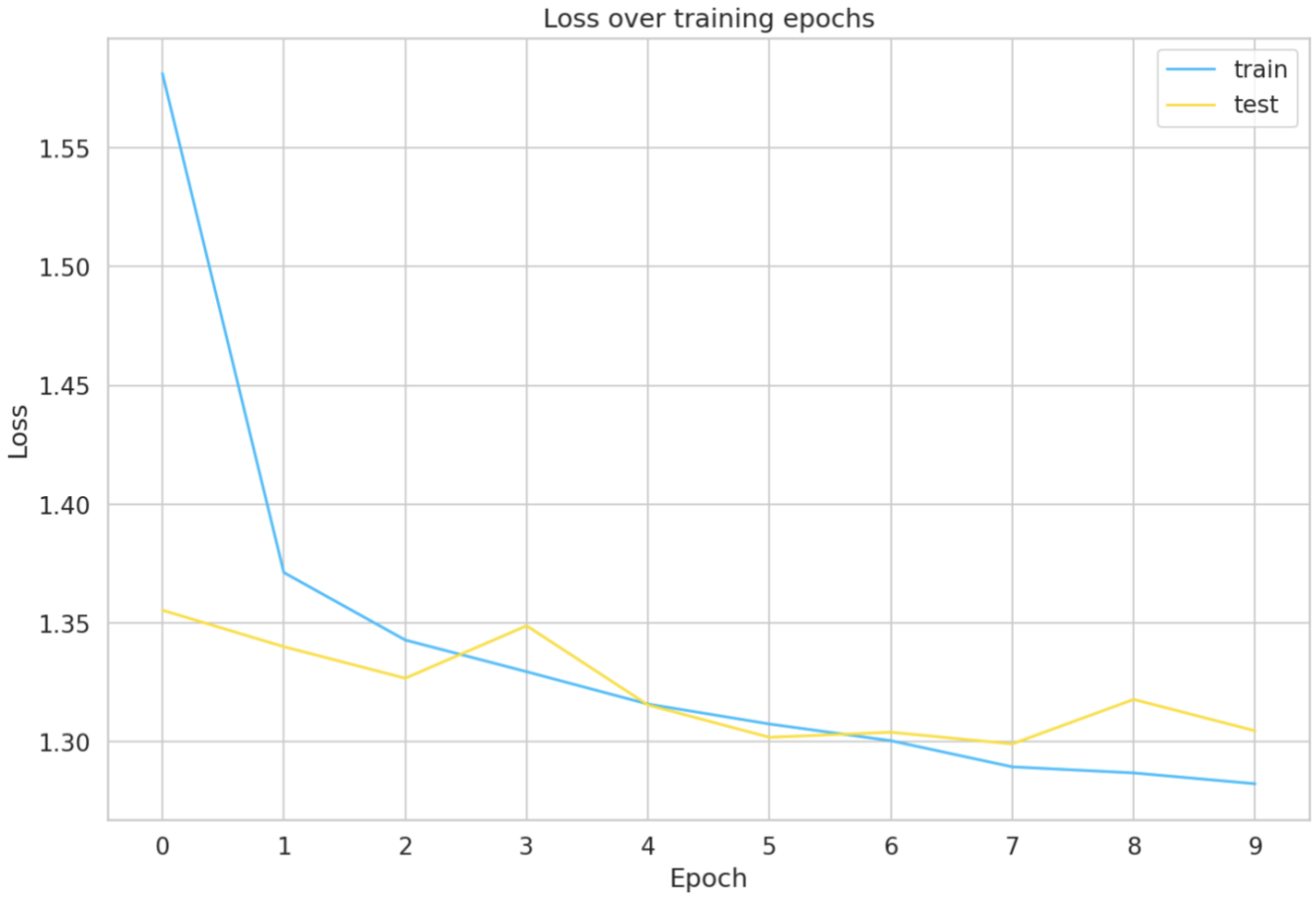

Figure 14: Loss over Training Epochs (batch size = 16 and epochs = 10)

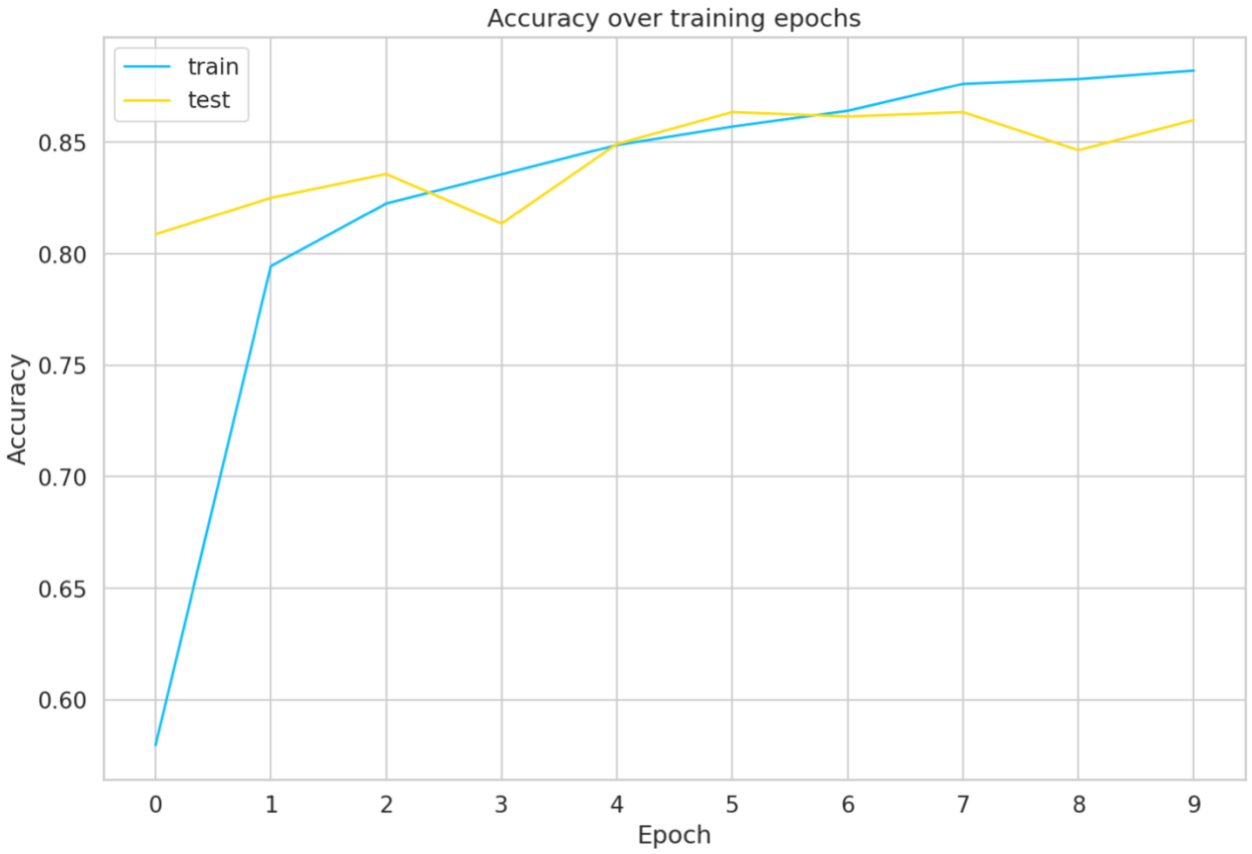

Figure 15: Accuracy over Training Epochs (batch size = 16 and epochs = 10)

In this case, the model obtained a training accuracy of 89.48% and a test accuracy of 83.64%

	precision	recall	f1-score	support
Pinot Noir	0.88	0.81	0.85	2461
Red Blend	0.84	0.79	0.82	1661
Chardonnay	0.88	0.89	0.89	2155
Cabernet Sauvignon	0.79	0.78	0.78	1717
Riesling	0.86	0.91	0.88	941
Sauvignon Blanc	0.82	0.81	0.81	975
Bordeaux-style Red Blend	0.74	0.89	0.81	1300
accuracy			0.84	11210
macro avg	0.83	0.84	0.83	11210
weighted avg	0.84	0.84	0.84	11210

Figure 16: Classification report (when the model is trained with 16 batch sizes and 10 epochs)

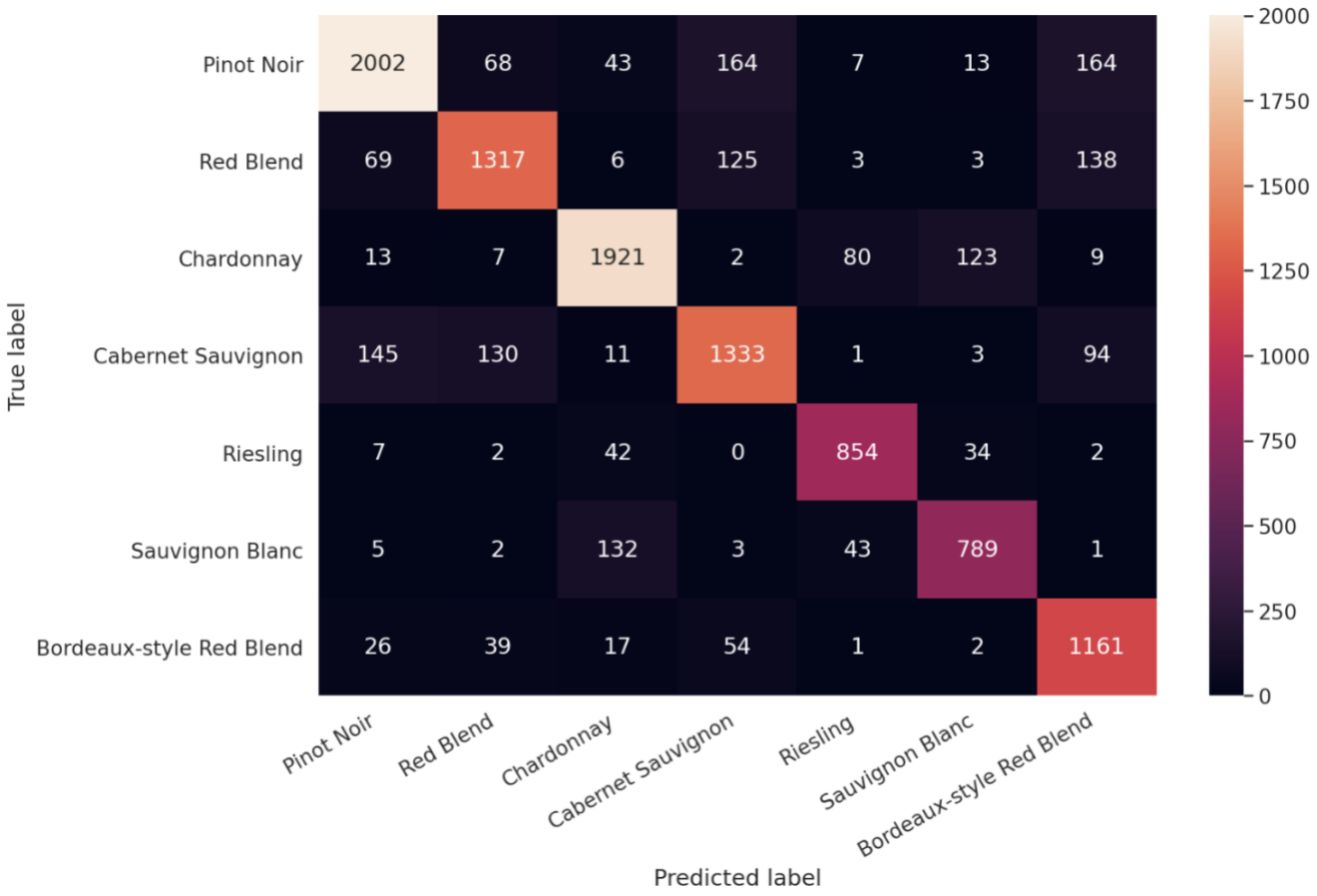

Figure 17: Confusion matrix (when model trained with 16 batch sizes and 10 epochs)

```
text: Strong wine made of red grapes
intent: Red Blend

text: Grapy plummy and juicy taste
intent: Cabernet Sauvignon
```

Figure 18: Model Testing (when the model trained with 16 batch sizes and 10 epochs)

Model Training with (Batch size = 32 and epochs = 5)

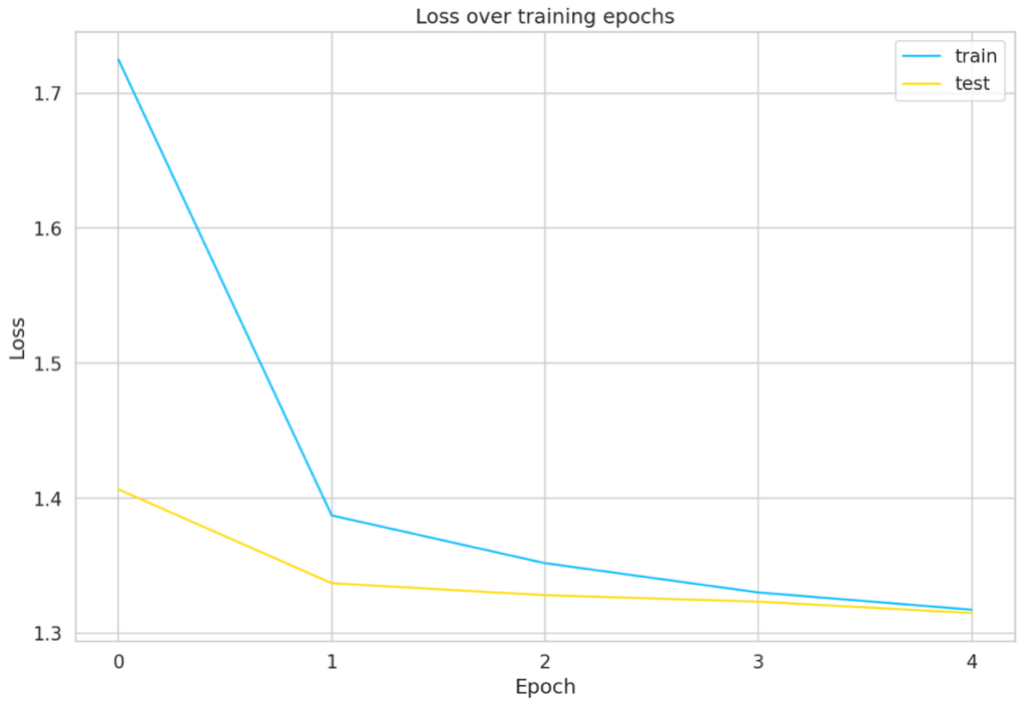

Figure 19: Loss over Training Epochs (batch size = 32 and epochs = 5)

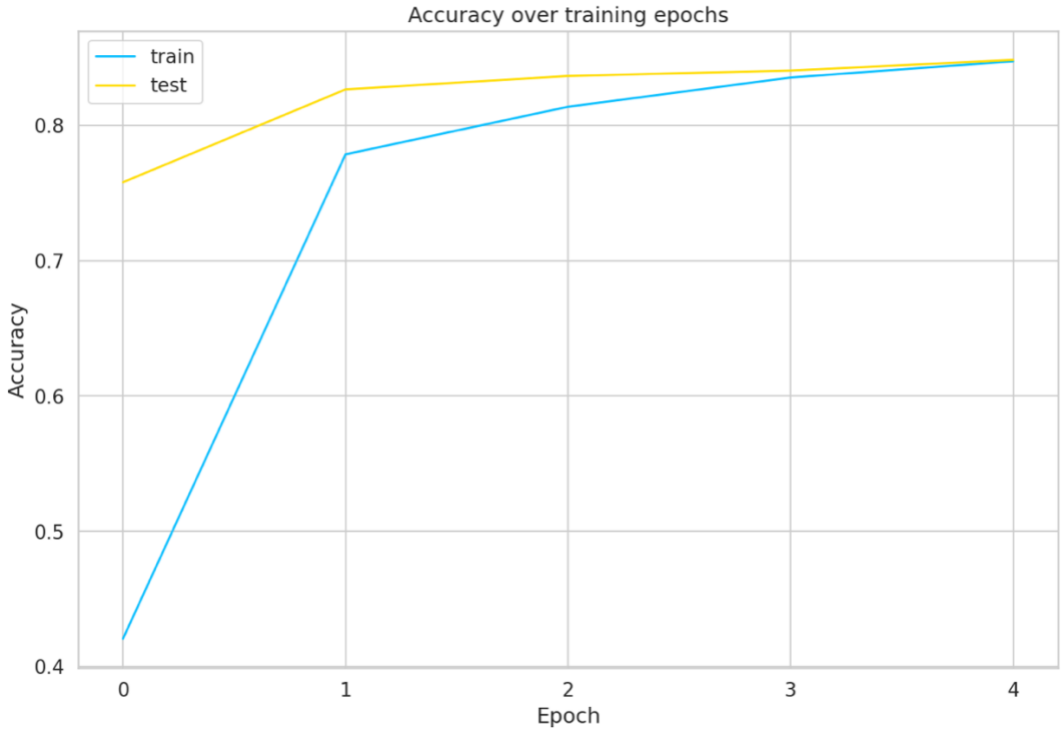

Figure 20: Accuracy over Training Epochs (batch size = 32 and epochs = 5)

```
#Calculating training accuracy and test accuracy
_, train_acc = model.evaluate(data.train_x, data.train_y)
_, test_acc = model.evaluate(data.test_x, data.test_y)

print("train acc", train_acc)
print("test acc", test_acc)
```

```
788/788 [==============================] - 139s 176ms/step - loss: 1.2884 - acc: 0.
8764
351/351 [==============================] - 61s 174ms/step - loss: 1.3418 - acc: 0.8
226
train acc 0.8763889074325562
test acc 0.8225691318511963
```

Figure 21: Training accuracy of 87.63% and Test accuracy of 82.25% (batch size = 32 and epochs = 5)

```
#Classification report
y_pred = model.predict(data.test_x).argmax(axis=-1)
print(classification_report(data.test_y, y_pred, target_names=classes))
```

	precision	recall	f1-score	support
Pinot Noir	0.90	0.79	0.84	2461
Red Blend	0.84	0.79	0.81	1661
Chardonnay	0.92	0.81	0.86	2155
Cabernet Sauvignon	0.73	0.83	0.78	1717
Riesling	0.83	0.92	0.87	941
Sauvignon Blanc	0.70	0.85	0.77	975
Bordeaux-style Red Blend	0.77	0.85	0.81	1300
accuracy			0.82	11210
macro avg	0.81	0.83	0.82	11210
weighted avg	0.83	0.82	0.82	11210

Figure 22: Classification report when a model is trained with (batch size = 32 and epochs = 5)

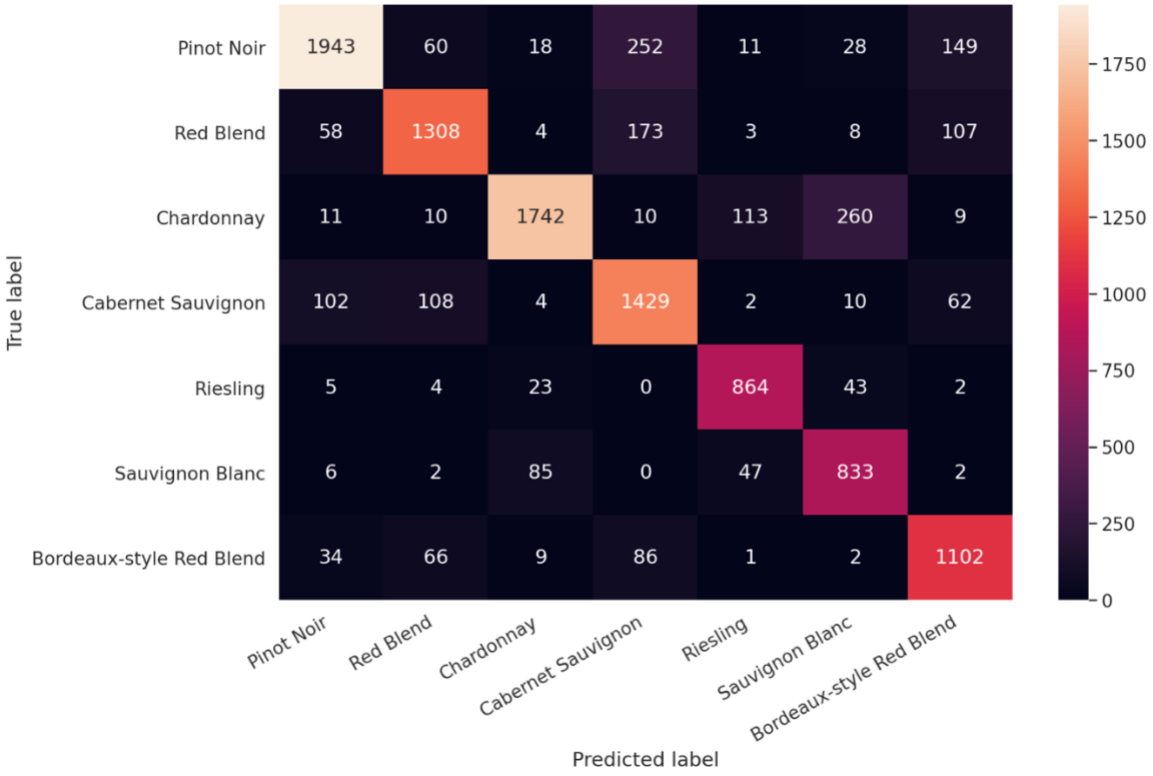

Figure 23: Confusion matrix (batch size = 32 and epochs = 5)

```
text: Strong wine made of red grapes
intent: Pinot Noir

text: Grapy plummy and juicy taste
intent: Cabernet Sauvignon
```

Figure 24: Model Testing with batch size =32 and epochs = 5

Model Training with batch size =32 and epochs = 10

```
model.compile(
  optimizer=keras.optimizers.Adam(1e-5),
  loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")]
)
```

```
#Try with batch size = 32 and epochs = 10
log_dir = "log/intent_detection/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%s")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=log_dir)

history3 = model.fit(
  x=data.train_x,
  y=data.train_y,
  validation_split=0.1,
  batch_size=32,
  shuffle=True,
  epochs=10,
  callbacks=[tensorboard_callback]
)
```

Figure 25: Model Training Model Training with batch size = 32 and epochs = 10

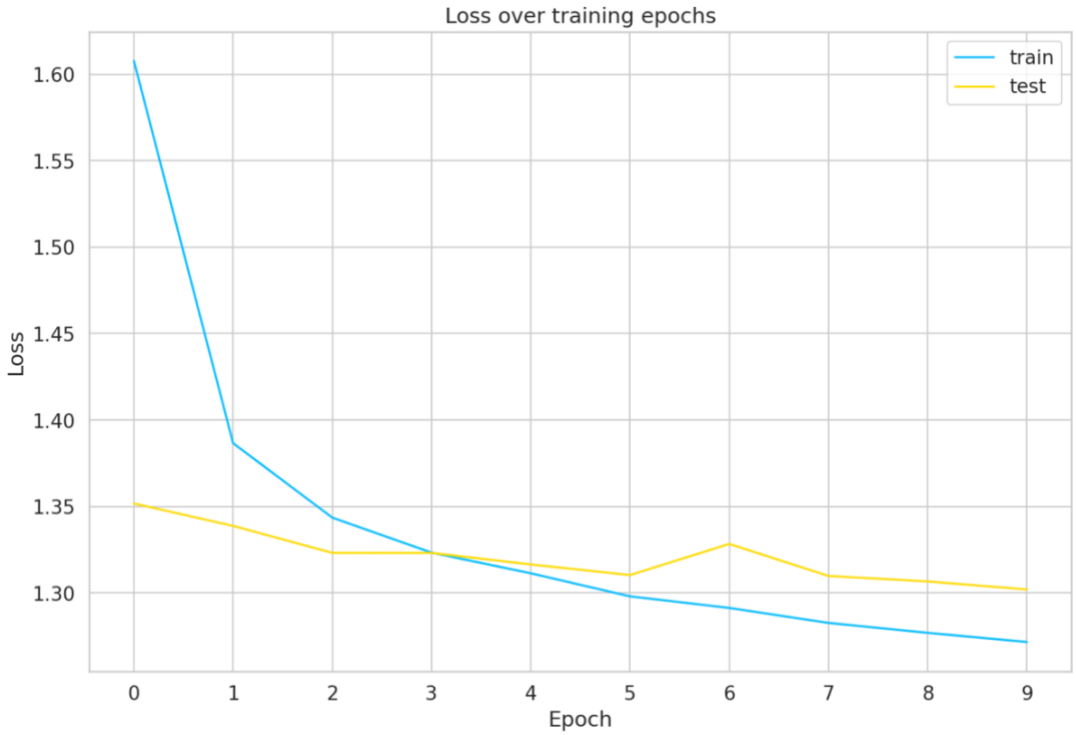

Figure 26: Loss over Training Epochs with (batch size = 32 and epochs = 10)

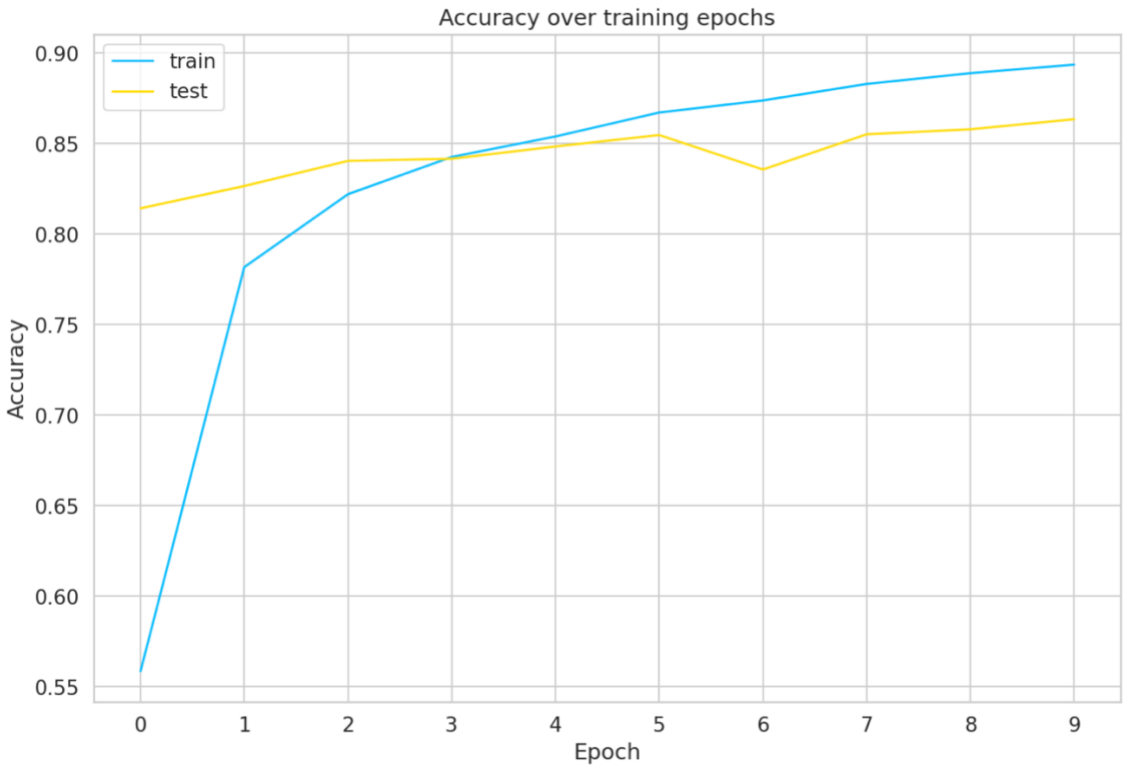

Figure 27: Accuracy over training epochs (batch size = 32 and epochs = 10)

```
#Training and Testing accuracy
_, train_acc = model.evaluate(data.train_x, data.train_y)
_, test_acc = model.evaluate(data.test_x, data.test_y)

print("train acc", train_acc)
print("test acc", test_acc)
```

```
788/788 [==============================] - 139s 176ms/step - loss: 1.2575 - acc: 0.
9077
351/351 [==============================] - 61s 174ms/step - loss: 1.3332 - acc: 0.8
301
train acc 0.9077380895614624
test acc 0.8300624489784241
```

Figure 28: Training accuracy of 90.77 % and Testing accuracy of 83% (batch size = 32 and epochs = 10)

When a model is trained with a batch size of 32 with 10 epochs, I got a training accuracy of 90.77% and a test accuracy of 83.00%. As observing the training and testing loss with the number of increasing epochs, losses are converging.

```
#Classification Report
y_pred = model.predict(data.test_x).argmax(axis=-1)
print(classification_report(data.test_y, y_pred, target_names=classes))
```

```
                        precision    recall  f1-score   support

           Pinot Noir       0.90      0.82      0.85      2461
          Red Blend         0.89      0.78      0.83      1661
          Chardonnay        0.94      0.78      0.85      2155
  Cabernet Sauvignon        0.75      0.85      0.80      1717
            Riesling        0.76      0.95      0.84       941
    Sauvignon Blanc         0.72      0.86      0.78       975
Bordeaux-style Red Blend    0.78      0.87      0.82      1300

            accuracy                            0.83     11210
           macro avg        0.82      0.84      0.83     11210
        weighted avg        0.84      0.83      0.83     11210
```

Figure 29: Classification report when a model is trained with (batch size = 32 and epochs = 10)

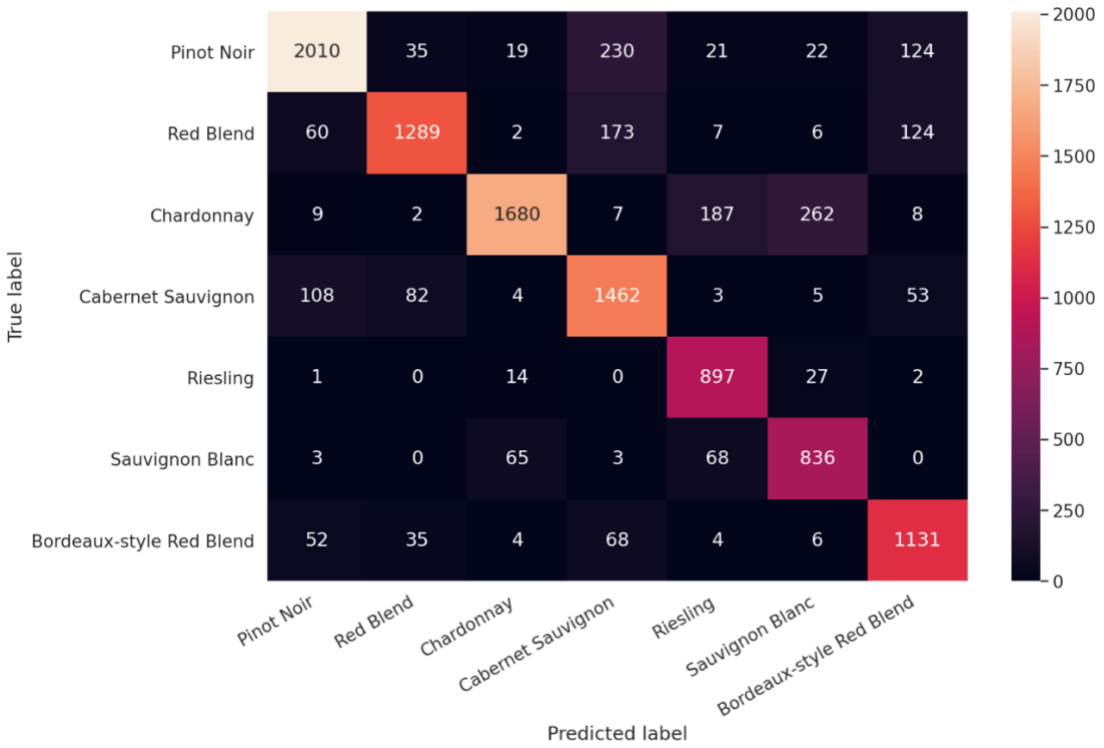

Figure 30: Confusion matrix (batch size = 32 and epochs = 10)

```
text: Strong wine made of red grapes
intent: Red Blend

text: Grapy plummy and juicy taste
intent: Cabernet Sauvignon
```

Figure 31: Model Testing (batch size = 32 and epochs = 10)

From the above experiments, I come up with the insight that the model training with different hyperparameters will provide different results and model performance in different classes. For instance, in some classes models perform well and in certain classes not, this can be visualized with the confusion matrix. Here in all these experiments, the model has been trained in Kaggle GPU P100 up to 6 versions for the visualization of results in different approaches. The working Jupyter Notebook based on this report has been submitted along with it.

CONCLUSION:

The presented work demonstrates the successful implementation of a multi-level classification model utilizing the powerful BERT pre-trained model. The results showcase state-of-the-art performance on a widely recognized public dataset, with an overall accuracy of 84% achieved through training the model with a batch size of 16 and 10 epochs. The model was trained using a learning rate of 0.00001, an Adam optimizer, and a loss function employing sparse categorical cross-entropy. Fine-tuning of the pre-trained BERT model was conducted using the wine dataset in four different configurations: 16 batch size with 5 epochs, 16 batch size with 10 epochs, 32 batch size with 5 epochs, and 32 batch size with 10 epochs. The best results were obtained when training the model with a batch size of 16 and 10 epochs, demonstrating improved convergence of train and test loss curves and increased overall accuracy. This model holds significant potential for various applications, particularly in industries such as customer service, where accurate prediction of customer needs and behaviors is vital for success. Overall, this work establishes a solid foundation for leveraging state-of-the-art natural language processing techniques to address complex classification problems.

REFERENCE

[1] https://www.kaggle.com/datasets/zynicide/wine-reviews (Datasets)

[2] https://www.kaggle.com/code/kshitijmohan/wine-recommendation-system-based-on-bert (Model Accuracy: 82%)

[3] https://www.kaggle.com/code/mariapushkareva/wine-reviews-spacy-bert/notebook (Model Accuracy: 81%)