Deep Learning
Wine Quality Assessment – Regression & Classification Tasks

INTRODUCTION

This project provides a step-by-step guide to preparing a dataset of wine quality measurements for use in a classification model and regression model. The dataset is first split into training and validation sets, and then the type information (red = 1, white = 0) is added as the 13th column in both Dataframes. The red DataFrame is then appended to the white DataFrame to create a single DataFrame with 6497 samples. The data is then split further into a training set (75% of the data) and a validation set (25% of the data), with 10 samples from the validation set being taken out for “testing”. Finally, the .ipynb notebook uses the target (rank-1) vectors to create three additional target arrays, where the quality levels have been mapped and one-hot-encoded. These additional target arrays are used for the classification model.

Technology Used:

- Python

Library Used:

- Pandas
- Numpy
- Matplotlib
- XG Bost
- Scikit-learn

[PART I]: Getting the data as Numpy arrays

a) Adds the 'type' information (red = 1, white = 0) as the 13th column in both Dataframes

1.We use the Pandas library to load two datasets containing information on red and white wines. The datasets are stored in CSV format and are loaded into Pandas dataframes using the read_csv() function. The sep parameter is used to specify that the separator in the CSV file is a semicolon.

2. After loading the datasets, the code adds a new column to each data frame called 'type' to distinguish between red and white wines. A value of 1 is assigned to the 'type' column in the red wine dataframe, and a value of 0 is assigned to the 'type' column in the white wine dataframe.

3.Finally, the code prints the first 5 rows of the white wine dataframe using the head() function. The commented-out lines of code can be uncommented to print the first 5 rows of the red wine dataframe or to print the first 5 rows of the white wine dataframe after adding the 'type' column.

b) Appends the red DataFrame to the white DataFrame, yielding a single DataFrame, wines, with 6497 samples

1. This code appends the red wine dataframe to the bottom of the white wine dataframe using the append() function. The resulting combined data frame is stored in a new variable called 'wines'.

2. The ignore_index=True parameter is used to reset the index of the combined dataframe so that it is sequential starting from 0.

3. After combining the dataframes, the code prints the first 5 rows of the combined dataframe using the head() function.

4. Finally, the code verifies that the combined data frame contains 6497 samples using an assert statement. If the number of samples is not equal to 6497, an error will be raised.

5. Overall, we are combining the red and white wine datasets into a single dataframe called 'wines' , which will be used for further analysis.

C) Uses sklearn's train_test_split() to create TRAINING [TR] and VALIDATION [TT] SPLITS – we will be using the results for training [TR], AND VALIDATION [TT].

1. This code uses the train_test_split() function from the sci-kit-learn library to split the combined wine dataframe into training, validation, and test sets.

2. First, the code sets aside 10 samples for the test set using the test_size parameter. The random_state parameter is used to ensure that the random splitting is reproducible.

3. Next, the remaining data is split into training and validation sets using the test_size parameter set to 0.25, which means that 25% of the remaining data will be used for validation. Again, the random_state parameter is used to ensure reproducibility.

4. Finally, the code verifies that the resulting training, validation, and test sets have the expected sizes using the len() function and == operator in the assert statements.

5. Overall, we are splitting the combined wine dataset into three subsets - training, validation, and test - which will be used to train, tune, and evaluate machine learning models for predicting wine quality.

D) The training and validation input sets will be returned from train_test_split() as Pandas DataFrames. The notebook converts them to NumPy arrays (to be used in Keras).

1. This code extracts the input features (X) and target variable (y) from t

   the training and validation sets.

2. The input features are obtained by dropping the 'quality' column from the data frames using the drop() function with the axis=1 parameter to indicate that the column is a dimension to be dropped. The resulting data frames are then converted to numpy arrays using the values attribute.

3. The target variable is obtained by selecting only the 'quality' column from the data frames using square bracket notation.

4. The resulting input features and target variables are then converted to NumPy arrays of type float32, which is a data type that is commonly used for deep learning models in the Keras library.

5. Overall, we are preparing the training and validation data for use in training and evaluating a Keras deep learning model for predicting wine quality.

E) Finally, where the quality LEVELS have been mapped as indicated in the table above (1 = BAD, 2 = MEDIUM, 3 = GOOD, 4 = EXCELLENT) and one-hot-encoded, so there is a 4-value target for each pattern. – These additional target arrays are used for the classification model.

1. This code is performing one-hot encoding of the target variable in the training and validation sets.

2. First, the code creates numpy arrays of zeros with shape (n_samples, 4), where n_samples is the number of samples in the training or validation set. The number 4 represents the number of classes for the target variable, which are wines with a quality score of 3 or lower, wines with a quality score between 4 and 6, wines with quality scores between 7 and 8, and wines with a quality score of 9 or higher.

3. Next, the code loops through each sample in the training or validation set and sets the corresponding element in the one-hot encoded array to 1 based on the quality score of the sample. For example, if the quality score is between 4 and 6, the second element in the one-hot encoded array is set to 1, indicating that the sample belongs to the second class.

4. Overall, we are preparing the target variable for use in a Keras deep learning model by converting it to a one-hot encoded array, which is a common technique for multi-class classification problems.

[PART II]: The REGRESSION MODEL

Adapting the data for a REGRESSION model (red & white together), target = quality (0 to 10)

The two files will be considered together (red & white wines) for a total of 4898 + 1599 = 6497 samples

and one more attribute ('type') will be added to each sample, to identify if the sample is from a red (type = 1) or a white (type = 0) wine.

Therefore, the neural network for solving this regression problem, will have:

• 12 inputs in the first layer (the 11 original attributes AND the new 'type' attribute)

• 1 output processing element in the output layer, since the result will be a single number (0 to 10)

II.1 Develop a (very simple) model, (regmodl1) – This model must only have 1 hidden layer and no more than 8 processing elements in that layer.

1. First, the training data is normalized by subtracting the mean and dividing it by the standard deviation. This is done using the mean and std functions of the NumPy array.

2. Next, a simple neural network model is defined using the Sequential API in TensorFlow. The model consists of an input layer with 8 nodes, an activation function of 'relu', and an output layer with a single node.

3. The model is then compiled using the 'adam' optimizer and the 'mae' loss function, which stands for mean absolute error. The mean absolute error is a common loss function used for regression problems and measures the average absolute difference between the predicted and actual values.

4. The model is then trained on the training set for 50 epochs with a batch size of 64, and the training history is stored in the history variable.

5. After training, the model is evaluated on the training set using the evaluate method, which returns the loss and means absolute error for the training set. These values are printed on the console.

6. The model is also evaluated on the validation set using the evaluate method, and the loss and mean absolute error for the validation set are printed to the console.

Interpretation:

1. The plot shows the training and validation losses per epoch for the model. The training loss is the error of the model on the training data, while the validation loss is the error of the model on the validation data.

2. The plot can be used to evaluate the performance of the model. Ideally, we want the training loss and validation loss to decrease together and stabilize at a low value. If the training loss decreases but the validation loss does not, it is an indication of overfitting, where the model is memorizing the training data and not generalizing well to new data.

3. In this plot, we see that the training loss and validation loss both decrease and stabilize around 0.67 after around 20 epochs. The two lines are very close together, indicating that the model is not overfitting the training data. Overall, this suggests that the model is performing well on the data.

II.2 Developing a (better) model that actually would be capable of overfitting (regmodel 2)– There must be at least 2 hidden layers in this model.

1. This code trains a neural network model for regression using the Keras API in TensorFlow.

2. The model is a feedforward neural network with three dense layers, the first two layers having 32 neurons and a linear activation function, and the output layer having a single neuron with a linear activation function.

3. The optimizer used to minimize the mean absolute error (MAE) loss function is Adam with a learning rate of 0.01. The model is trained for 50 epochs on the training set (X_tr_norm and y_train) with a batch size of 32 and the validation set (X_val_norm and y_val) is used for validation during training.

4. After training the model, the code evaluates its performance on both the training and validation sets using the MAE metric. The evaluation metrics are printed on the console. Finally, the validation loss and MAE are also printed in a formatted string.

Interpretation:

1. This code is using matplotlib to plot the training and validation losses per epoch for the second regression model regmodl2.

2. First, a figure with size (10,6) is created using plt.figure(). Then, the training and validation loss values for each epoch are extracted from history2.history and plotted using plt.plot().

3. The label for each line is specified using the label, and a legend is created with plt.legend() to differentiate the two lines.

4. Finally, the title, x-axis label, and y-axis label are set using plt. title(), plt.xlabel(), and plt.ylabel(), respectively. The plot is displayed using plt.show().

II.3 Use your iterative observations of the performance of the model in II.2 to modify/tune hyperparameters (for example deciding how many epochs of training to allow) to arrive at a "Final Regression Model" (regmodl3), which will be considered to be the "best model" you could develop to solve this regression task.- There must be at least 2 hidden layers in this model.

1. This code performs a hyperparameter tuning using GridSearchCV to fnd the best combination of hyperparameters for a neural network model.

2. First, the create_model function is defined, which creates a neural network model with the specified number of hidden layers, processing elements, learning rate, and activation function.

3. Next, a KerasRegressor object is created using create_model as the build function. This allows the neural network model to be used within sci-kit-learn's GridSearchCV framework.

4. The hyperparameter ranges to search over are then defined using lists of possible values for hidden_layers, processing_elements, learning_rate, and activation.

5. A dictionary of param_grid is then created, which maps the hyperparameters to their possible values.

6. A GridSearchCV object is created using the model and param_grid objects, with scoring set to 'neg_mean_absolute_error', cv set to 3 (for 3-fold cross-validation), and n_jobs set to -1 (to use all available CPUs).

7. Finally, the ft method of the grid object is called with the training data (X_tr_norm and y_train) and validation data (X_val_norm and y_val), as well as the number of epochs and batch size to use during training.

8. After the grid search is complete, the best mean absolute error and corresponding hyperparameters are printed using grid_result.best_score_ and grid_result.best_params_, respectively.

Interpretation & Comparison:

1**.** Since the hyperparameters are optimized using GridSearchCV and the best estimator is already selected, we can access the training history of the best estimator using grid_result.best_estimator_.history.history.

2. Here, we are getting the mean absolute error on the training and validation sets per epoch from the training history of the best estimator using grid_result.best_estimator_.history.history['loss'] and grid_result.best_estimator_.history.history['val_loss'], respectively.

3. Finally, we are plotting the training and validation losses over the epochs using plt.plot(train_mae, label='Training loss') and plt.plot(val_mae, label='Validation loss'), respectively. The x-axis represents the epochs, and the y-axis represents the loss. We are also adding labels to the x-axis and y-axis using plt.xlabel('Epoch') and plt.ylabel('loss'), respectively. Lastly, we are adding a legend to the plot using plt.legend().

4.This code is for visualizing the mean absolute error (MAE) of a neural network model durin g training and validation over a number of epochs. Here are the steps taken in the code:

i)The train_mae and val_mae variables are set to the MAE values for the training and validati on sets, respectively. These values are obtained from the history object of the neural network model that resulted from the grid search. history contains information about the performance of the model on the training and validation sets over each epoch during training.

ii) The plt.plot() function is used to plot the train_mae values and val_mae values on the same graph. The label parameter is used to specify the labels for each line on the graph. The xlabel () and ylabel() functions are used to add labels to the x-axis and y-
axis of the graph, respectively. The legend() function is used to show the labels for each line on the graph.

iii) Finally, the plt.show() function is called to display the graph.

iv) This code provides a way to visualize the training and validation performance of a neural n etwork model over a number of epochs, which can help to identify overfitting or underfitting and to optimize hyperparameters for the model.

CONCLUSIONS

Using the GridSearchCV helped me in achieving to finalize the best parameter values from a given set of parameters in a grid in the final model that is best model.

Statistics I achieved from the 3 Regression models:

1) For Regmodl1 that is basic regression model:

```
Training Loss: 0.5571, Training MAE: 0.5571
Validation Loss: 0.5562, Validation MAE: 0.5562
```

2) For Regmodl2 that is second regression model:

```
Training Loss: 0.5754, Training MAE: 0.5754
Validation Loss: 0.5632, Validation MAE: 0.5632
```

3) For Regmodl3 that is final regression model:

```
Validation Loss: 0.4921, Validation MAE: 0.5393
```

Using the Best Model and Predicting the 10 test patterns and comparing them to the 10 target values of the test set and recorded in the below table

Pattern #	Features														
	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	Activation	Target	Error
1	6.	0.	0.	10.	0.	26.	108.	1.	3.	0.	11.	0.	1.707	2	0.293
2	10.	0.	0.	2.	0.	7.	20.	1.	3.	1.	11.	1.	-1.906	3	1.094
3	13.	0.	1.	3.	0.	5.	16.	1.	3.	1.	9.	1.	-3.205	2	1.205
4	8.	0.	0.	7.	0.	22.	64.	1.	3.	0.	11.	0.	0.412	2	1.588
5	7.	0.	0.	3.	0.	40.	122.	1.	3.	1.	11.	0.	1.651	3	1.349
6	8.	1.	0.	2.	0.	13.	151	.1.	3.	0.	10.	0.	6.873	2	4.873
7	7.	0.	0.	14.	0.	40.	183.	1.	3.	0.	9.	0.	3.751	2	1.751
8	6.	1.	0.	2.	0.	15.	26.	1.	4.	1.	12.	1.	-0.188	2	1.812
9	6.	0.	0.	6.	0.	56.	158.	1.	4.	0.	10.	0.	-1.534	2	0.466
10	7.	0.	0.	13.	0.	46.	120.	1.	3.	0.	10.	0.	-2.187	3	0.813

[PART III]: THE CLASSIFICATION MODEL

III.1 Develop a (very simple) model , (clasmodl1) –This model must only have 1 hidden layer, and no more than 8 processing elements in that layer. – Is this model achieving better accuracy than a "random classifer"

1. This code is implementing a neural network for a classifcation problem. The neural network has two layers: a hidden layer with 8 neurons and ReLU activation function, and an output layer with 4 neurons and softmax activation function.

2. The model is compiled with categorical_crossentropy as the loss function, adam as the optimizer, and accuracy as the evaluation metric.

3. The model is then trained using the ft() method, with the training data X_tr_norm and y_train_categorical as inputs, and validation data X_val_norm and y_val_categorical. The number of epochs for training is set to 100.

4. After training, the model is evaluated on both the training and validation sets using the evaluate() method, and the training loss, training accuracy, validation loss, and validation accuracy are printed to the console.

Interpretation:

1. This code plots the training and validation loss (categorical cross-entropy) of a neural network classifcation model over the epochs of training.

2. The loss measures how well the model is performing in terms of predicting the correct class for each input sample. The training loss is calculated using the training set, while the validation loss is calculated using a separate validation set that is not used during training.

3. The history_clasmodl1 object is the training history of the model, which is generated by calling the ft() method of the model. The history object contains the loss and accuracy values of the model on the training and validation sets at each epoch of training.

4. The plt.plot() function is used to plot the training and validation loss over the epochs. The history_clasmodl1.history['loss'] and history_clasmodl1.history['val_loss'] attributes are used to access the training and validation loss values from the history object, respectively. The resulting plot shows the training and validation loss as a function of the number of epochs of training.

III.2 Develop a (better) model that overfits (clasmodl2)– There must be at least 2 hidden layers.

1. This code defines and trains a more complex neural network model, called clasmodl2, using Keras. It consists of six dense layers with an increasing number of neurons and uses the ReLU activation function for each hidden layer and the softmax activation function for the output layer, which produces four class probabilities as the final output.

2. The model is compiled with the categorical cross-entropy loss function and the Adam optimizer. During training, it uses the same training and validation sets as before (X_tr_norm and y_train_categorical for training, and X_val_norm and y_val_categorical for validation), and trains for 100 epochs.

3. After training, the accuracy of the validation set is printed.

Interpretation:

This code creates a plot of the training and validation loss over the epochs for the clasmodl2 model trained in the previous code block. history_clasmodl2.history is a dictionary that contains the training and validation loss (and accuracy, if specified) values at each epoch during training. The code uses the plot function from the matplotlib library to plot the training and validation loss values, with the training loss in blue and the validation loss in orange. The title, xlabel, ylabel, and legend functions are used to add labels and a legend to the plot. Finally, the show is called to display the plot.

III.3 Use your iterative observations of the performance of the model in III.2 to modify/tune hyperparameters (for example deciding how many epochs of training to allow) to arrive at a "Final Classification Model" (clasmodl3), which will be considered to be the "best model" you could develop to solve this classification task.- There must be at least 2 hidden layers in this model.

1. This code performs hyperparameter tuning using grid search on a Keras neural network model.

2. The first step is to import the necessary libraries, including the GridSearchCV class from sci-kit-learn, and the Sequential, Dense, Dropout, and Adam classes from Keras. The KerasClassifer class is also imported from keras.wrappers.scikit_learn, which allows the Keras model to be used as an estimator in sci-kit-learn's GridSearchCV.

3. Next, a function called create_model is defined. This function returns a sequential model with dense layers and dropout layers, with the specified number of neurons, activation function, dropout rate, and learning rate. The model architecture includes an input layer with the same number of neurons as the number of features in the input data, followed by two hidden layers with the specified number of neurons and activation function, and a dropout layer with the specifed dropout rate after each hidden layer. Finally, the model has an output layer with 4 neurons and a softmax activation function. The model is compiled using the Adam optimizer with the specifed learning rate and 'categorical_crossentropy' as the loss function, and 'accuracy' as the metric to optimize.

4. After defning the model architecture, a parameter grid is defned, which contains different combinations of hyperparameters to test. The hyperparameters that are tested include the number of neurons in each hidden layer, the activation function used in each layer, the dropout rate, and the learning rate of the optimizer.

5. A KerasClassifer object is created using the create_model function and is passed to the GridSearchCV object. The estimator parameter of the GridSearchCV object is set to Kmodel, which is the KerasClassifer object, and the param_grid parameter is set to the parameter grid that was defned earlier. The cv parameter specifes the number of cross-validation folds, which is set to 3 this case, and the scoring parameter specifies the metric to use for evaluation, which is 'accuracy'.

6. The ft method of the GridSearchCV object is called on the training data, which performs a grid search over the specifed parameter grid and fnds the best hyperparameters. The training data is passed as X_tr_norm and y_train_categorical. X_tr_norm is the normalized training data, and y_train_categorical is the training data with one-hot encoded labels. The grid search is performed using 3-fold cross-validation.

7.Finally, the best parameters and corresponding accuracy score are printed out. The best_params_ attribute of the grid_result object contains the best hyperparameters, and the best_score_ attribute contains the corresponding accuracy score. The printed output displays the best parameters and the corresponding accuracy score.

Interpretation & Comparison

1. This code creates the fnal model using the best hyperparameters obtained from the grid search. Then it trains the fnal model on the entire training set and evaluates its performance on the test set.

2. First, the create_model function is called with the best hyperparameters obtained from the grid search to create the fnal model.

3. Then the fnal model is trained using the ft method with the entire training set. The number of epochs is set to 100, batch size is set to 32 and validation data is provided. The training data is used to update the weights of the model and the validation data is used to

evaluate the model's performance at the end of each epoch. The history object is created to store the loss and accuracy values of the model during training.

4. Finally, the evaluate method is called on the fnal model to obtain its performance on the test set. The test loss and accuracy are printed out.

5. Overall, this code performs the fnal step of the machine learning pipeline, which is to train the fnal model with the best hyperparameters obtained from hyperparameter tuning and evaluate its performance on the test set. This step helps to estimate the generalization performance of the model and assess its suitability for deployment in a real-world scenario.

<u>CONCLUSIONS</u>

To conclude that the minority classes of a dataset will not get a good representation on a classifier and representation for each class can be solved by oversampling and undersampling to balance the representation classes over datasets.

Using the GridSearchCV helped me in achieving to finalize the best parameter values from a given set of parameters in a grid in the final model that is best model.

Statistics I achieved from the 3 Classification models:

1) For **clasmodl1** that is basic classification model:

```
Accuracy: 82.37

Training accuracy: 0.8249
Validation accuracy: 0.8237
```

2) For **clasmodl2** that is second classification model:

```
Accuracy: 79.35
Training accuracy: 0.7807
Validation accuracy: 0.7935
```

3) For **clasmodl3** that is final classification model:

```
Accuracy: 81.81257843971252
```

Using the Best Model and Predicting the 10 test patterns and comparing them to the 10 target values of the test set and recorded in the below table

Pattern #	Features														
	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	Activation	Target	Hit?(Y/N)
1	6.	0.	0.	10.	0.	26.	108.	1.	3.	0.	11.	0.	2	2	Y
2	10.	0.	0.	2.	0.	7.	20.	1.	3.	1.	11.	1.	2	3	N
3	13.	0.	1.	3.	0.	5.	16.	1.	3.	1.	9.	1.	2	2	Y
4	8.	0.	0.	7.	0.	22.	64.	1.	3.	0.	11.	0.	2	2	Y
5	7.	0.	0.	3.	0.	40.	122.	1.	3.	1.	11.	0.	3	3	Y
6	8.	1.	0.	2.	0.	13.	151	.1.	3.	0.	10.	0.	2	2	Y
7	7.	0.	0.	14.	0.	40.	183.	1.	3.	0.	9.	0.	2	2	Y
8	6.	1.	0.	2.	0.	15.	26.	1.	4.	1.	12.	1.	3	2	N
9	6.	0.	0.	6.	0.	56.	158.	1.	4.	0.	10.	0.	3	2	N
10	7.	0.	0.	13.	0.	46.	120.	1.	3.	0.	10.	0.	3	3	Y