Learning from data Coursework

December 2021

Contents

1 Introduction

The main objective of this work is to produce models that can predict the quality of a wine based on its chemical composition. This prediction will take the form of the models assigning a wine with a number for 0 to 10 as a ranking, therefore making this a classification problem then is further analysed to provide predictive insite. As the problem is classification in nature the two models chosen are multilayer perceptron (MLP) and convolution neural networks (CNN).

To gain a full understanding of the data and its predictions the analysis will be split into multiple subtasks: An outline of the data is required, what are the possible relations between different chemical components? Furthermore is the relation between each individual chemical component and the related score, this is to identify if any particular component is more influential than the others and will help in the generation of chemically superior wines that can then be assessed with the models. The data as a whole must be analysed in comparison with the ranking as this may produce further interactions that are not seen individually between components. These steps should provide insite into the components of the wine in order to predict what levels of each are required to create superior wines and produce models that are able to predict a wines quality.

There is one significant potential issue with the data and that is the nature of the quality statistic. Due to this having been defined subjectively by one or more people it. As different people have different tastes, the data may not follow a perfectly logical pattern and may not produce the best results. Therefore the models may not produce accurate predictions.

2 Data Set

The data set [1] comprises of a [12, 1600] arrangement of data that are all numerical, apart from the titles of course. There are 11 features to the data set, these are; fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates and alcohol. There is also one outcome variable that is the quality. All values are positive numbers without units as for this purposes the units are not required, and the quality is between 0 and 10. However when it comes to quality the data is not balanced as there are only 6 of the 10 possible classifications present and they are all in the middle. See Figure 1.

This dataset is one of the few cases where there is complete data, without any missing values and complete, unified data. Therefore very little feature cleaning or engineering is required. However in the future as more data may be collected complete data coverage cannot

be guaranteed and therefore must be guarded against. If the data was just set to 0 it would drastically effect the models so another option is needed. If the data is received with a quality value then all of the wines in the same quality category can have their features averaged and the average is then used to fill the missing data. This will slow down the pre-processing however it will provide a much more reliable standard for what the value should be improving the accuracy of the model.

Further to this, analysis of how the relationship between the features and the outcome variable shows there are some features that effect the outcome variable more than others. For each feature I have normalised its data to produce a standardised gradient, allowing me to accurately analyse the data produced, the gradients are all to three significant figures. See appendix 2 for all of the graphs produce. The first feature is fixed acidity which produces a noticeable, yet small positive gradient of 0.017 showing that it does have an impact on the result however it is not huge. Volatile acidity and citric acid have similar, yet opposite effects on the quality of the wine with gradients of -0.055 and 0.055 respectively. These values are relatively large compared to the other features showing they are areas of interest in predicting the quality of a wine. However some areas appear to not hold much value in predicting the quality, this is the case for residual sugar with a very small positive correlation of 0.002 showing it has some effect but very small. The chlorides have a similar, yet negative effect to the fixed acidity having a gradient of -0.012, therefore these should be reduced when creating a quality wine. There is an interesting difference between two closely related features. The levels of free sulfur dioxide seem to produce a small negative effect with a gradient of -0.009 and very little effect on the quality, however the total amount of sulfur dioxide produces a large effect on the quality with a gradient of -0.026. With both of these values being negative and free sulphur being derived from the total sulphur, a simple area to work on to improve the quality is to reduce the total sulphur levels in the wine. All of the features have some effect on the quality except for the density that produces a gradient of 0.0 with three significant figures and therefore does not seem to matter when creating wine. The pH is another feature with very little effect on the quality of the wine, with a negative gradient of -0.003 and therefore is not a vital feature to check. The final two features are much more important to the quality of the wine, with the amount of sulphates having a positive 0.026 gradient and the alcohol level having a positive 0.042 and therefore are vital when creating a quality wine.

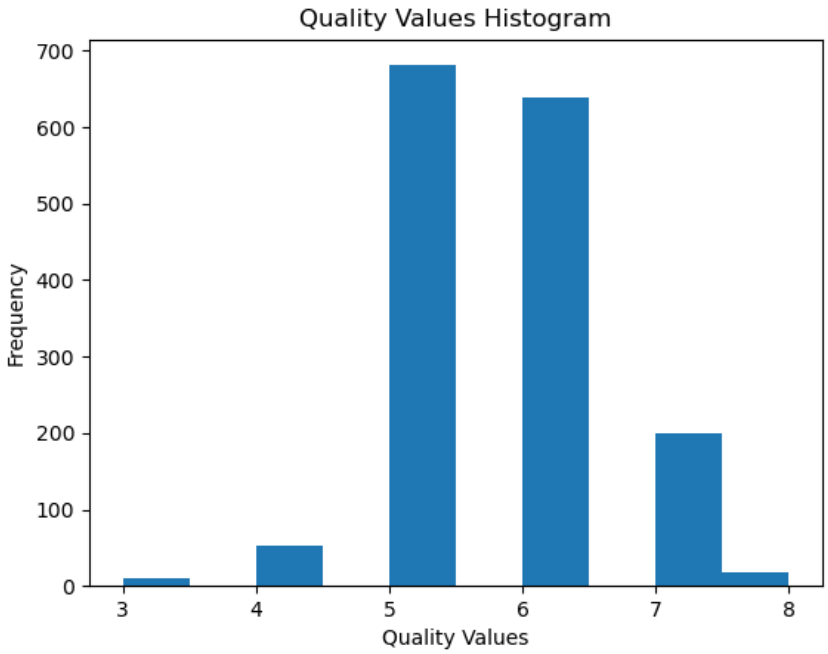

Figure 1: A histogram showing the distribution of quality data in the data. This shows a large tendency towards the middle numbers and very few highs and lows.

3 Training Summary

As stated previously the two models being used to analysis the data are Multilayer Perceptron models and Convolution Neural Networks, all of these tests are run with the full values of the wine at once as the model may provide some insite or find some connections that doing it individually would not find. The implementation of the MLP provides many different variables to change in an attempt to provide the best results, these are the size of the hidden layers, maximum iterations, the activation function, the solver and the learning rate. The initial implementation has 5 hidden layers, 100 max iterations, the relu activation function, a Stochastic Gradient Descent solver and inverse scaling learning rate. When run 50 times with a test/train split of 0.2 it produces an average mean accuracy of 0.34 which is quite low and therefore requires a change in variables. Increasing the size of the hidden layer to 10 and the max iterations to 1000 improves the mean accuracy to 0.41 with yet another increase of the hidden layer size to 12 providing no additional benefit with

a mean accuracy of 0.412 again. therefore the previous noticeable increase may be due to the max iterations, however increasing this to 1500 provides very little improvement, with a mean average of 0.43, this appears to be because the training loss does not improve and the model stops its self after 10 consecutive epochs if it does not improve by more than 0.0001. This leads me to believe the learning rate needs to be changed in order to improve the model, so changing the learning rate to constant provides a drastic benefit, raising the average mean to 0.51 with a adaptive learning rate improving on this very slightly at 0.52. The activation function is a crucial part of any model and therefore selecting the correct one is vital for performance, previously the relu function has been used as it is the default for sklearn however changing this to identity provides a range of means, ranging from 0.55, a marked improvement, to 0.47. Equally a change to a logistic and tanh activation functions provides a similar range of results, this shows that for this dataset the activation function provides very little change in mean accuracy and therefore improvements must be made elsewhere. This improvement can be seen with the solver functions, where a switch from sgd to ldfgs can provide an improvement to 0.6 with 3000 iterations and a switch to adam can give improvements of up to 0.62 with only 1000 iterations. Finally the teas/train split must be taken into consideration, a change from 0.2 to 0.3 provides a slight decrease in performance, dropping the average mean to 0.56, equally a change to 0.1 also decreases the models accuracy, therefore the originally split of 0.2 is the best. This testing provides the best arrangement of variables for this model on this data set, that arrangement being a hidden layer size of 12, 100 max iterations, the relu activation function, the adam solver, an adaptive learning rate and a test/train split of 0.2 producing an average mean of 0.62, an improvement of 0.28 over the original settings.

Further to the MLP I have implemented a Convolution Neural Network as the second model. This model has some expected similar categories as the MLP, such as the activation functions and test/-train ratios. The CNN also includes a batch size, number of epochs, a softmax function, an optimizer and a loss function. To start with these values are set at using the relu activation function, a 0.2 test/-train split, a 256 batch size, 100 epochs, the rmsprop optimizer,and the sparse categorical crossentropy. This gives a much better accuracy than the original accuracy of the MLP at 0.57. With changes for the activation functions producing no decidable effect on the accuracy with all options producing the same or lower mean accuracy. Next was to test how changing the number of epochs would effect the model, increasing the number from 10 to 1000 did improve the accuracy to 0.65 however the testing accuracy was 0.94, this is indicative of over fitting the model and would make the model less

reliable when interacting with new data, see figure 2. Therefore to make an effective model a balance must be found where the accuracy is high, yet over fitting has not taken place. A value of 250 provides this balance where there is little overfitting and the accuracy has started to plateau, see figure 3. This produced an accuracy of 0.58, which although less than 1000 ephocs, is more useful due to the more correct fitting. Attempting to reduce the batch size down to its default of 32 simply shows an increase of overfitting with no accuracy gain showing the 512 batch size is still better, equally the same effect happens when the batch size is doubled to 1024.

For this problem I would recommend the MLP as it consistently provides higher accuracy than the CNN and shows the distinct possibility to be optimized and improved whereas the CNN did not change much when its variables were changed. The low accuracy rates of both these models may be attributed to the problem I raised in the Introduction where the quality measurement is subjective and therefore a clear, generalised pattern may not be possible to find.

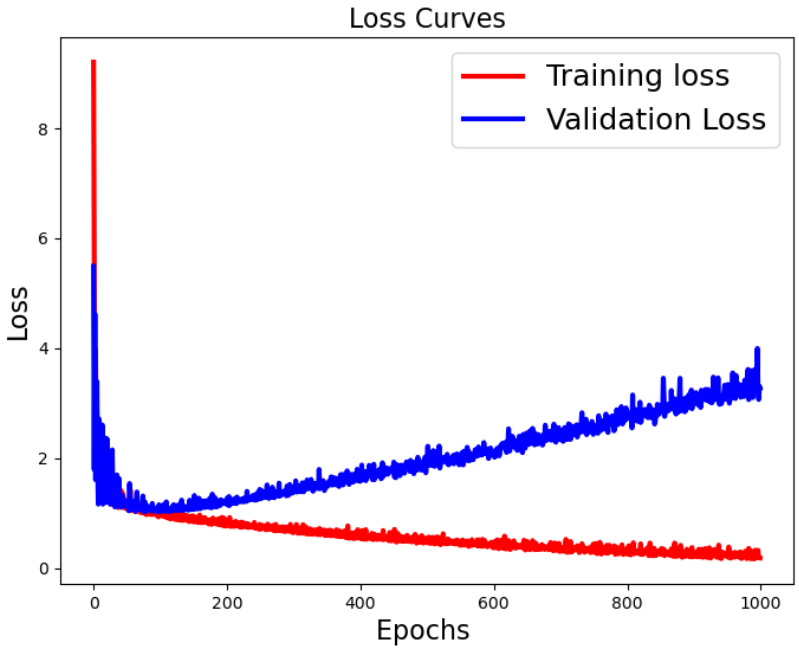

Figure 2: A graph detailing the training and validation loss for the CNN with 1000 ephoc's.

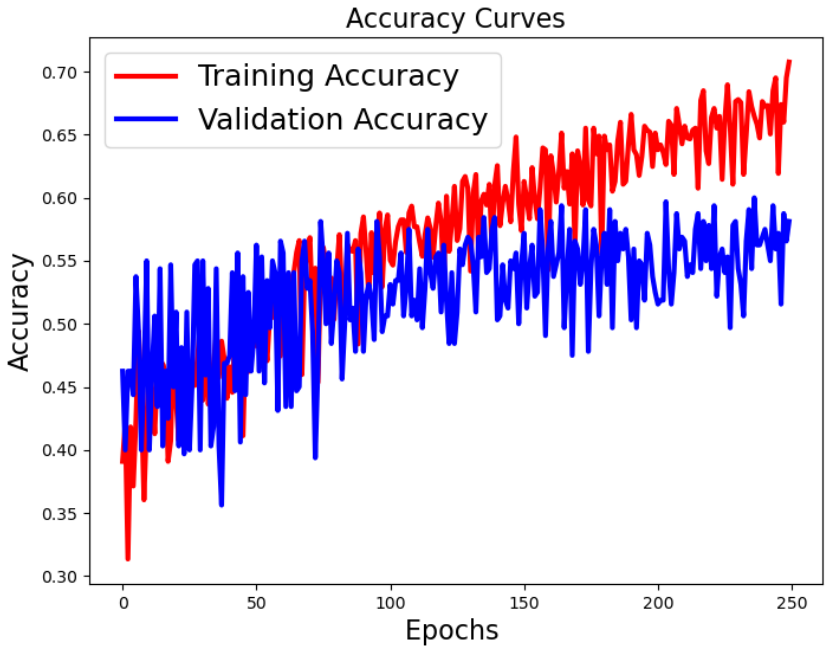

Figure 3: A graph detailing the training and validation accuracy of the CNN with 250 ephoc's

4 Key Findings and Insights

Having implemented a way to generate randomised wines and pass them to the model I have decided to attempt to predict what the quality of these wines will be and to test if they follow a similar pattern to the previous database and if certain outliers can be found. As can be seen in figure 4 with a small sample of only 100 generated wines, the model greatly favours predicting the absolute average of the classes with 5 taking up the majority of the predictions, what can also be seen is the model will also predict down to 3 and up to 8, however these are in small quantities. This is representative of the original data set as in that set there are no wines that are graded below a 3 or above an 8. From this we can either take away that from a small sample of wines most are very average or that the model is exaggerating its training data and is 'playing it safe'. However when using a much larger data set of 10000 new wines the original trend of favouring a quality of 5, a quality 7 of is relatively high as well, see figure 5. What this might tell us is that even through random generation some variables hold more sway over the

quality and therefore should be more closely watched when creating a wine.

Further to the last two points the problem set out earlier in the introduction has yet been proven further, with that the quality feature for the wine is a subjective measurements in the tester and does not come from a place of true objective fact it makes it hard for the models to truly predict what wine will be good quality as that fact is different fro everybody, although the insite can be used to further improve the wine.

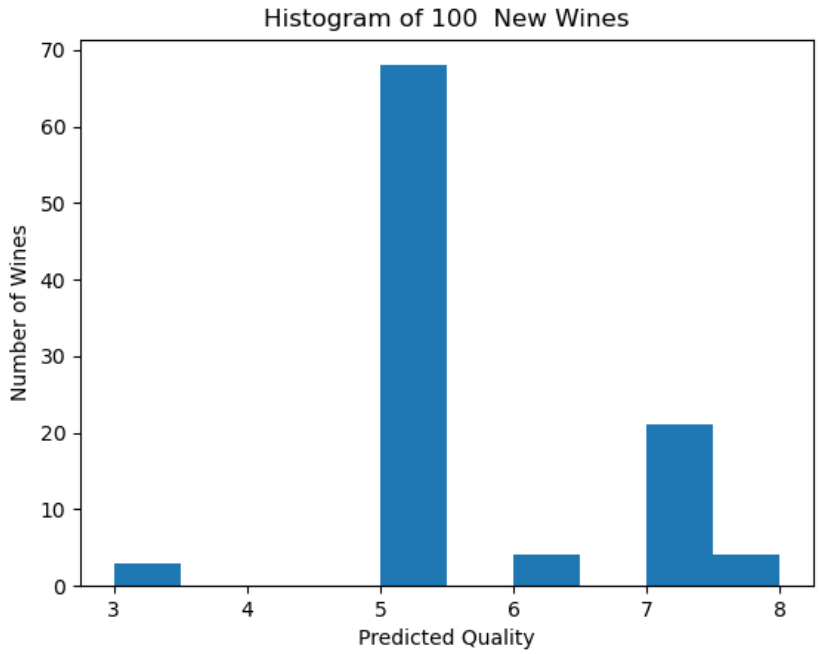

Figure 4: A histogram showing the distribution of predicted qualities for 100 randomly generated wines

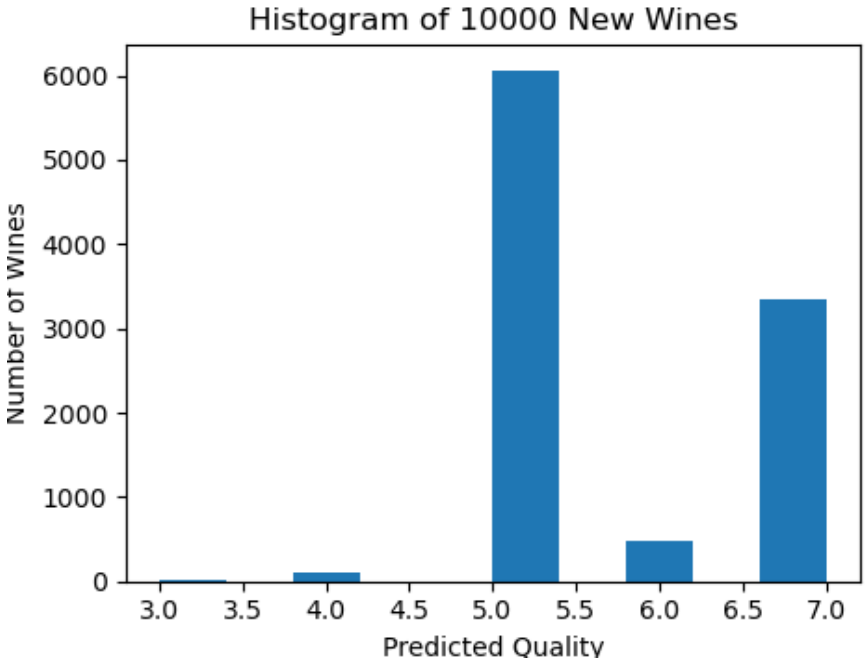

Figure 5: A histogram showing the distribution of predicted qualities for 10000 randomly generated wines

5 Next Steps

The next steps for this analysis is to attempt to produce a more reliable quality measurement perhaps through large scale testing so the model is able to find more of a pattern. Equally a larger data set of wine would help refine the models as 1599 wines may not be enough to distinguish all of the relationships between the compounds. before these issues are fixed I believe different models would not produce much improved results due to the fundamentally subjective nature of the quality. A further issue of the model is that it focuses on the middle ground a lot more and will not produce outputs outside of the set 3-8 range provided by the data set therefore more examples of these sorts of wines, even if there are not many of them would in turn improve the predictive capabilities of the model due to it being able to learn what the qualities of these wines are, potentially being able to recommend near-perfect wines. If this data is collected on one person and the model is used to predict only for that person then the subjective nature of the quality is removed, therefore I can see this being deployed for wine personalisation, where a the model

learns a singular persons tastes and is able to recommend wine types and certain features, although a lot of data would need to be collected on that one person, potentially restricting its effectiveness for a while.

Overall the findings of this project conclude that the MLP model is more consistent with this data than the CNN and that wine quality can be predicted from its chemical composition. However more data and a more objective quality check is required to optimize the models for prediction.

References

[1] P. Cortez, "Wine quality data set," 2009. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Wine+Quality

6 Appendix 1

```
import pandas as pd
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from random import randint, uniform
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Read in csv file and slpit by ;
wine_df = pd.read_csv('winequality-red.csv', sep=";")

# Create target dataframe of the quality column
target_data = wine_df['quality']
del wine_df['quality']
# Create dataframe of all other values
values = wine_df

train_features, test_features, train_targets, test_targets =
    train_test_split(values, target_data, test_size=0.2)


def quality_values(target_data, values):
    '''
    This function takes in the target data amd tjhe values and
    produces a scatter graph comparing the two,
    change the y value for different values
    :param target_data:
    :param values:
    '''

    x = target_data
    y = values['alcohol']

    # Normalized data to allow for more accurate gradients
    y_max = y.max()
    normalized_y = y / y_max

    plt.scatter(x, normalized_y)

    # Create a trend-line
    z = np.polyfit(x, normalized_y, 1)
    p = np.poly1d(z)
    plt.plot(x, p(x), "r--")

    print(z)

    plt.title("Alcohol: Gradient = " + str(round(z[0], 3)))
    plt.ylabel("Normalised Alcohol")
    plt.xlabel("Quality")
    plt.show()


# fixed acidity = small positive correlation = 0.017
```

```
# volatile acidity = negative correlation = -0.055
# citric acid = positive correlation = 0.055
# residual sugar = very small positive correlation, outliers = 0.002
# chlorides = very small negative correlation, outliers = -0.012
# free sulfur dioxide = very large negative correlation = -0.009
# total sulfur dioxide = very large negative correlation, notable outliers = -0.026
# density = small negative correlation = 0.0
# pH = very small negative correlation, some outliers = -0.003
# sulphates = positive correlation, many outliers = 0.026
# alcohol = strong positive correlation = 0.042

def CNN(train_features, test_features, train_targets, test_targets):
    '''
    Implements a CNN, trains it and returns it
    :param train_features:
    :param test_features:
    :param train_targets:
    :param test_targets:
    :return: model: A Tensorflow Sequential Model of the CNN
    '''
    classes_num = 10

    activation = 'relu'

    model = Sequential()
    model.add(Dense(512, activation=activation, input_shape=(np.prod(train_features.shape[1:]),)))
    model.add(Dense(512, activation=activation))
    model.add(Dense(classes_num, activation='softmax'))

    # rmsprop
    model.compile(optimizer='rmsprop',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(train_features, train_targets, batch_size=512, epochs=250,
                        verbose=1, validation_data=(test_features, test_targets))

    [test_loss, test_acc] = model.evaluate(test_features, test_targets)
    print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

    # Loss and accuracy graphs, uncomment to generate

    # # Plot the Loss Curves
    # plt.figure(figsize=[8, 6])
    # plt.plot(history.history['loss'], 'r', linewidth=3.0)
    # plt.plot(history.history['val_loss'], 'b', linewidth=3.0)
    # plt.legend(['Training loss', 'Validation Loss'], fontsize=18)
    # plt.xlabel('Epochs ', fontsize=16)
    # plt.ylabel('Loss', fontsize=16)
```

```
    # plt.title('Loss Curves', fontsize=16)
    #
    # plt.show()
    #
    # # Plot the Accuracy Curves
    # plt.figure(figsize=[8, 6])
    # plt.plot(history.history['accuracy'], 'r', linewidth=3.0)
    # plt.plot(history.history['val_accuracy'], 'b', linewidth=3.0)
    # plt.legend(['Training Accuracy', 'Validation Accuracy'],
    fontsize=18)
    # plt.xlabel('Epochs ', fontsize=16)
    # plt.ylabel('Accuracy', fontsize=16)
    # plt.title('Accuracy Curves', fontsize=16)
    #
    # plt.show()

    return model


def MLPModel(train_features, test_features, train_targets,
    test_targets):
    """
    Implements a MLP, trains it then returns it
    :param train_features:
    :param test_features:
    :param train_targets:
    :param test_targets:
    :return: model: The MLP model
    """
    all_means = 0
    iterations = 50
    for i in range(iterations):
        model = MLPClassifier(hidden_layer_sizes=12, max_iter=1000,
     activation='relu', solver='adam', verbose=10,
                              learning_rate='adaptive')
        model.fit(train_features, train_targets)

        predictions = model.predict(test_features)
        score = np.round(metrics.accuracy_score(test_targets,
    predictions), 2)
        print("Mean accuracy of predictions: " + str(score))
        all_means += score
    print("Average means = " + str(all_means / iterations))

    return model


def generateWines(values):
    """
    This function generates a dataframe of randomly generated wines
    , without quality measures. The range of each
    feature is generated by taking the difference between max and
    min, then subtracting it from the min, to a lower
    limit of 0 and adding it to the max.
    :param values:
    :return: newWines:
    """
```

```
    numOfWinesToGenerate = 100
    newWines = pd.DataFrame()

    minFixedAcidity = values['fixed acidity'].min()
    maxFixedAcidity = values['fixed acidity'].max()
    fixedAcidityDifference = maxFixedAcidity - minFixedAcidity
    fixedAcidityRange = [
        minFixedAcidity - fixedAcidityDifference if minFixedAcidity
     - fixedAcidityDifference > 0 else 0,
        maxFixedAcidity + fixedAcidityDifference]

    minVolatileAcidity = values['volatile acidity'].min()
    maxVolatileAcidity = values['volatile acidity'].max()
    volatileAcidityDifference = maxVolatileAcidity -
    minVolatileAcidity
    volatileAcidityRange = [
        minVolatileAcidity - volatileAcidityDifference if
    minVolatileAcidity - minVolatileAcidity > 0 else 0,
        maxVolatileAcidity + volatileAcidityDifference]

    minCitricAcid = values['citric acid'].min()
    maxCitricAcid = values['citric acid'].max()
    citricAcidDifference = maxCitricAcid - minCitricAcid
    citricAcidRange = [minCitricAcid - citricAcidDifference if
    minCitricAcid - citricAcidDifference > 0 else 0,
                       maxCitricAcid + citricAcidDifference]

    minResidualSugar = values['residual sugar'].min()
    maxResidualSugar = values['residual sugar'].max()
    residualSugarDifference = maxResidualSugar - minResidualSugar
    residualSugarRange = [
        minResidualSugar - residualSugarDifference if
    minResidualSugar - residualSugarDifference > 0 else 0,
        maxResidualSugar + residualSugarDifference]

    minChlorides = values['chlorides'].min()
    maxChlorides = values['chlorides'].max()
    chloridesDifference = maxChlorides - minChlorides
    chloridesRange = [minChlorides - chloridesDifference if
    minChlorides - chloridesDifference > 0 else 0,
                      maxChlorides + chloridesDifference]

    minFreeSulphurDioxide = values['free sulfur dioxide'].min()
    maxFreeSulphurDioxide = values['free sulfur dioxide'].max()
    freeSulphurDioxideDifference = maxFreeSulphurDioxide -
    minFreeSulphurDioxide
    freeSulphurDioxideRange = [
        minFreeSulphurDioxide - freeSulphurDioxideDifference if
        minFreeSulphurDioxide - freeSulphurDioxideDifference > 0
    else 0,
        maxFreeSulphurDioxide + freeSulphurDioxideDifference]

    minTotalSulphurDioxide = values['total sulfur dioxide'].min()
    maxTotalSulphurDioxide = values['total sulfur dioxide'].max()
    totalSulphurDioxideDifference = maxTotalSulphurDioxide -
    minTotalSulphurDioxide
    totalSulphurDioxideRange = [
```

```
        minTotalSulphurDioxide - totalSulphurDioxideDifference if
        minTotalSulphurDioxide - totalSulphurDioxideDifference > 0
    else 0,
        maxTotalSulphurDioxide + totalSulphurDioxideDifference]

    minDensity = values['density'].min()
    maxDensity = values['density'].max()
    densityDifference = maxDensity - minDensity
    densityRange = [minDensity - densityDifference if minDensity -
    densityDifference > 0 else 0,
                    maxDensity + densityDifference]

    minpH = values['pH'].min()
    maxpH = values['pH'].max()
    phDifference = maxpH - minpH
    pHRange = [minpH - phDifference if minpH - phDifference > 0
    else 0, maxpH + phDifference]

    minSulphates = values['sulphates'].min()
    maxSulphates = values['sulphates'].max()
    sulphatesDifference = maxSulphates - minSulphates
    sulphatesRange = [minSulphates - sulphatesDifference if
    minSulphates - sulphatesDifference > 0 else 0,
                      maxSulphates + sulphatesDifference]

    minAlcohol = values['alcohol'].min()
    maxAlcohol = values['alcohol'].max()
    alcoholDifference = maxAlcohol - minAlcohol
    alcoholRange = [minAlcohol - alcoholDifference if minAlcohol -
    alcoholDifference > 0 else 0,
                    maxAlcohol + alcoholDifference]

    for i in range(numOfWinesToGenerate):
        newWines = newWines.append([[uniform(fixedAcidityRange[0],
    fixedAcidityRange[1]),
                                     uniform(volatileAcidityRange
    [0], volatileAcidityRange[1]),
                                     uniform(citricAcidRange[0],
    citricAcidRange[1]),
                                     uniform(residualSugarRange[0],
     residualSugarRange[1]),
                                     uniform(chloridesRange[0],
    chloridesRange[1]),
                                     randint(
    freeSulphurDioxideRange[0], freeSulphurDioxideRange[1]),
                                     randint(
    totalSulphurDioxideRange[0], totalSulphurDioxideRange[1]),
                                     uniform(densityRange[0],
    densityRange[1]),
                                     uniform(pHRange[0], pHRange
    [1]),
                                     uniform(sulphatesRange[0],
    sulphatesRange[1]),
                                     (uniform(alcoholRange[0],
    alcoholRange[1]))]], ignore_index=True)
    return newWines

```

```
243
244 def predictQuality(newWines, MLPModel, CNNModel):
245     """
246     A simple function that just decides which model to use, it
    returns an array of the predictions
247     :param newWines:
248     :param MLPModel:
249     :param CNNModel:
250     :return: predictions: Array of the predictions
251     """
252     if MLPModel is not None:
253         predictions = MLPModel.predict(newWines)
254     else:
255         predictions = CNNModel.predict_on_batch(newWines)
256
257     return predictions
258
259
260 mode = 0
261 if mode == 0:
262     prediction = predictQuality(generateWines(values),
263                                 MLPModel(train_features,
    test_features, train_targets, test_targets),
264                                 None)
265 else:
266     prediction = predictQuality(generateWines(values),
267                                 None,
268                                 CNN(train_features, test_features,
    train_targets, test_targets))
269
270 # A histogram of the predictions generated by the model
271 plt.hist(prediction, bins=10, histtype='stepfilled')
272 plt.title('Histogram of 100  New Wines')
273 plt.xlabel('Predicted Quality')
274 plt.ylabel('Number of Wines')
275 plt.show()
```

7 Appendix 2

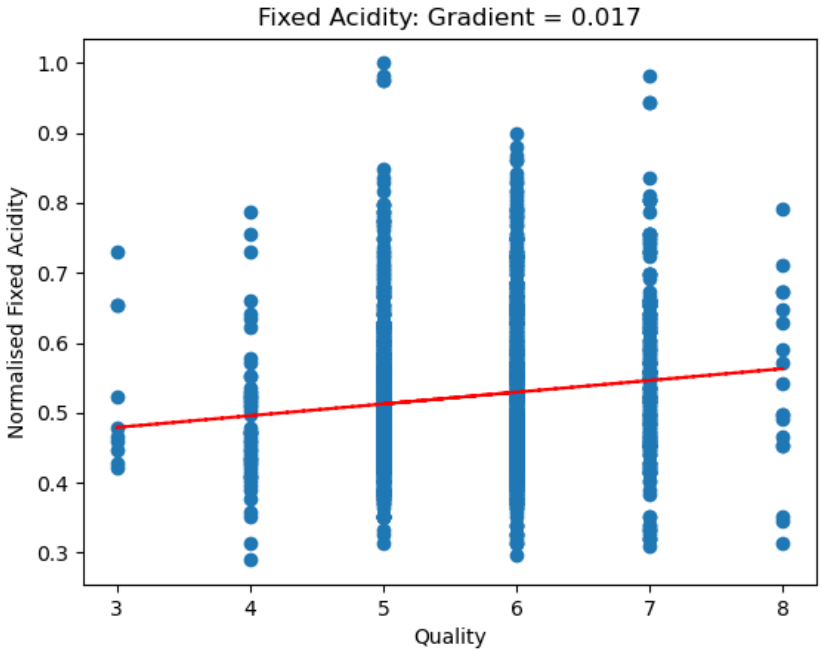

Figure 6: A graph showing the relationship between fixed acidity and quality

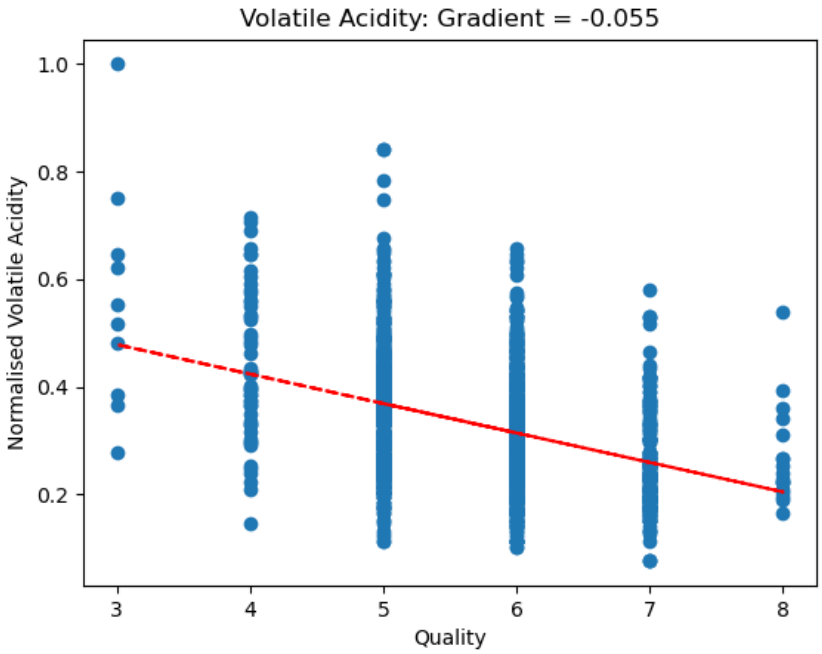

Figure 7: A graph showing the relationship between volatile acidity and quality

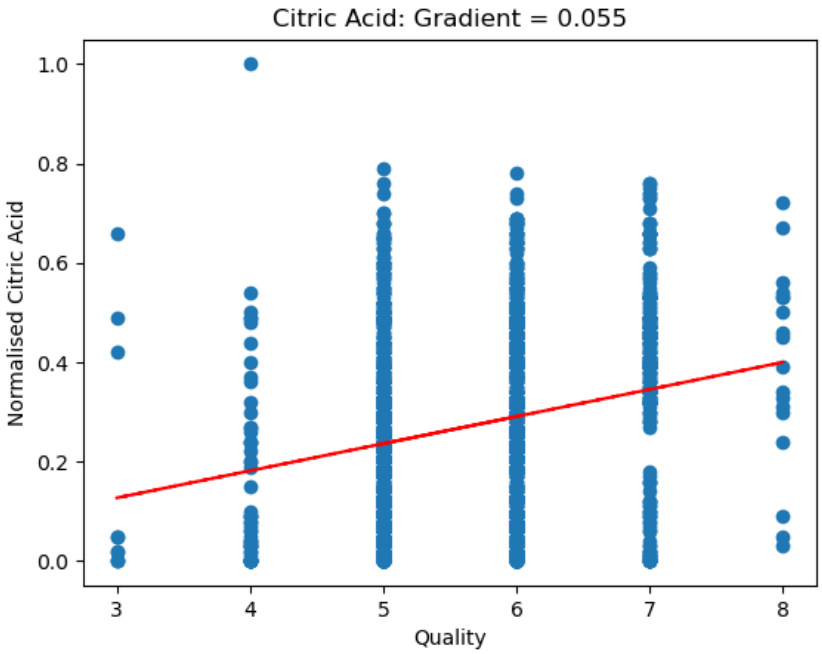

Figure 8: A graph showing the relationship between citric acid and quality

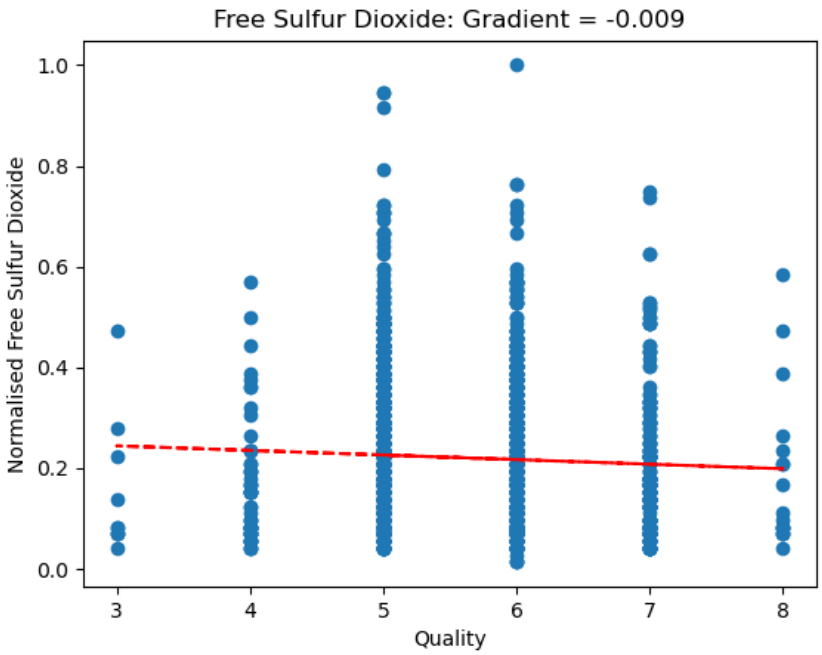

Figure 9: A graph showing the relationship between free sulphur dioxide and quality

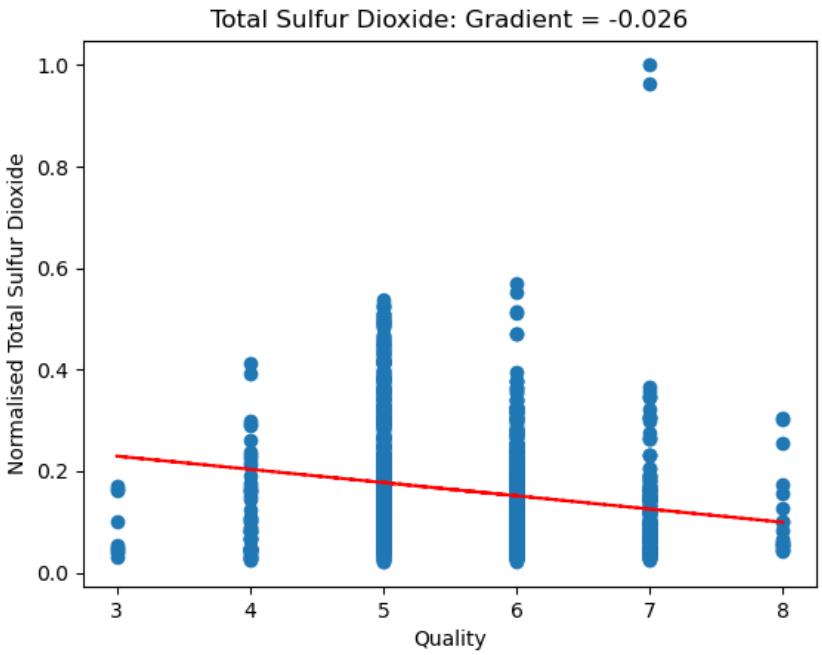

Figure 10: A graph showing the relationship between total sulphur dioxide and quality

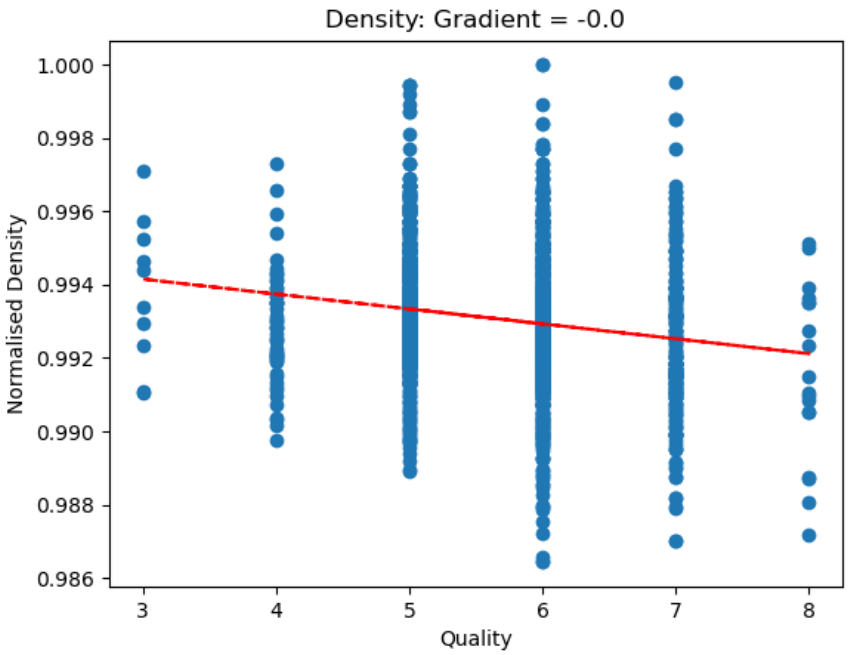

Figure 11: A graph showing the relationship between density and quality

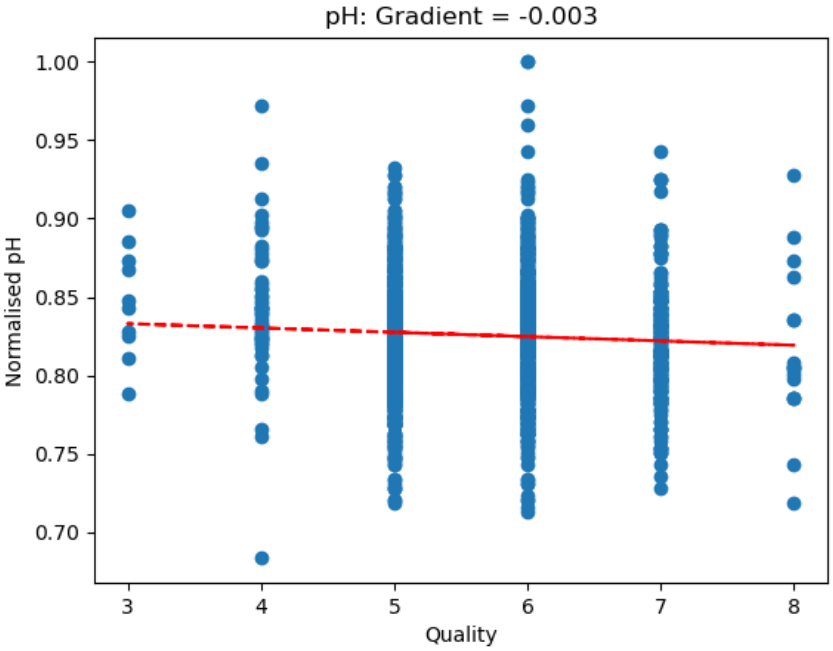

Figure 12: A graph showing the relationship between pH and quality

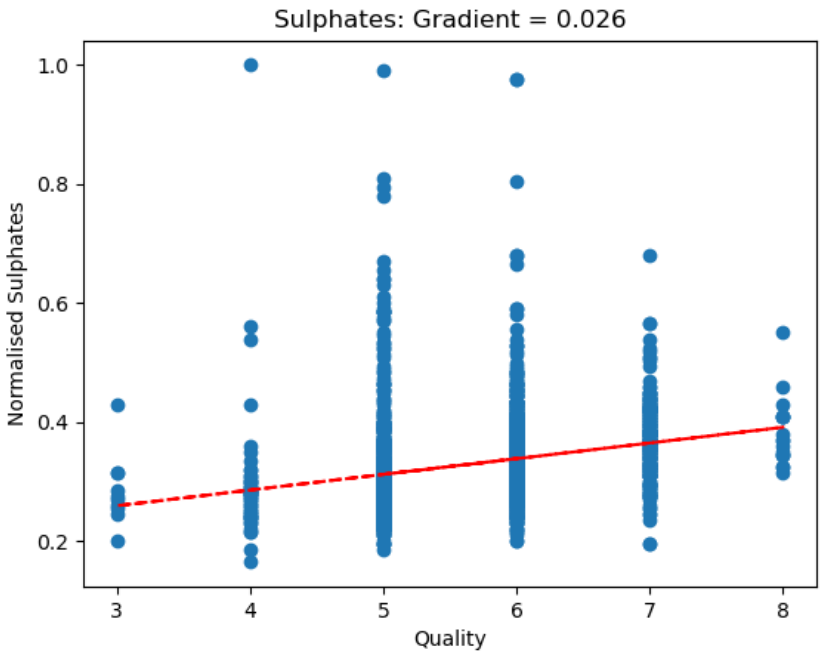

Figure 13: A graph showing the relationship between sulphates and quality

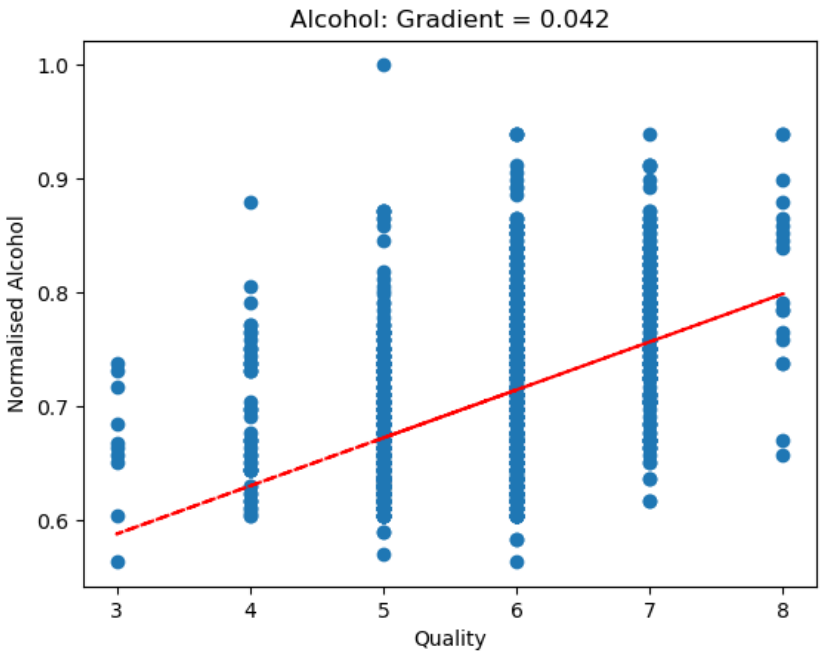

Figure 14: A graph showing the relationship between alcohol and quality