WINE QUALITY ANALYSIS

An Artificial Neural Network Approach

Michelle Masci
Kyle Lavorato

Table of Contents

1.0 Introduction

Importance of Study

The global wine market is a sector that experiences constant high value and sales, contributing a fair amount of revenue and taxes into the global economy. In the US alone, the wine market experiences total annual sales of $55.8 billion [1]. This is not even including cultures that are based around daily wine consumption, such as Italy. Additionally, wine appeals to many different people, making sales in both low and high class income regions. By nature, a slight difference in the quality of the wine can be the difference between a $25 or $3,000 bottle. Therefore, it is important to have ways to be able to easily classify wine for pricing and find ways to improve quality.

With taste being the least understood of the human senses, this makes wine classification and assessment a difficult task. Currently in the world, quality of wine is assessed by the inspection of chemical characteristics at a lab. These chemical details are gathered using an electronic tongue, which is an instrument that uses seven sensors to measure and compare tastes, similar to the human tongue [4].

Classifying wines using an ANN has been done in the past, with excellent results. A report from The Journal of Food completed testing on a multilayer perceptron with a root mean square error index between calculated and real data of less of than 0.14 [2]. However, the group had to go through several models, altering their choice of inputs until they reached this level. Using neural networks to improve wine quality classification can make wine production more efficient and help target marketing [3].

Purpose

The neural network that was constructed will classify wines by quality using the chemical details form the electronic tongue as input. The intended application of the artificial neural network is to assist in the assessment of wine quality. This ANN will lend to the increased transparency, speed, and further regulation of this important process.

Moreover, through the analysis of the chemical characteristics that compose high quality wine, this can assist in the production as well as the selling of wine. Companies will be able to learn about trends in the chemical composition that cause a wine to have higher quality. This project will help to eliminate the more fickle, human component of quality certification. With this increasing industry, the new technologies can have a large impact on the companies producing and selling wine.

2.0 ANN Architecture

Dataset

The data set (including both red and white variants) of “Vinho Verde” wine was obtained from the machine learning repository of UCI [1]. The data contains 6,497 points (1,599 red wine and 4,898 white wine), each with 11 input parameters (fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sugar dioxide, total sulfur dioxide, density, pH, sulfates and alcohol) in decimals and 1 output parameter (wine quality) represented by an integer between 0 (very bad) and 10 (premium). Additionally, the dataset contains many more normal quality wines than poor or excellent.

Data Preprocessing

The data has been preprocessed with a custom script before being used for neural network training. By examining the wine quality (output parameter) of each pattern, several outliers were discovered and removed from the data set to achieve better generalization. The outliers were pruned by calculating the frequency of occurrence of each value of wine quality. Then if a quality value's frequency was less than or equal to 1% of total occurrences, it was pruned. In addition, the 11 input parameters of each pattern have different value ranges, which will cause parameters with large values have larger effects in updating connection weights during training. Therefore, the data set has been normalized using a magnitude normalisation, such that all the input parameters of each pattern retain their original magnitude but are within the range [0,1]. The equation used for normalization is shown in equation 1.1

$$X' = (X - X_{min}) / (X_{max} - X_{min}) \qquad \text{Equation 1.1}$$

In which X is the value of a data pattern in one of 11 input parameters before normalization and X_{min} and X_{max} are the minimum and maximum among all data patterns in the same input parameter. Equation 1.1 has been applied to all data patterns for all 11 input parameters. Additionally, Gaussian Normalisation was also used on the input data initially, but the Magnitude Normalisation proved to generate better results. The Gaussian Normalisation option is still coded into the preprocessing script and can be reactivated in the future to try different methods to improve the network. Finally, 20% of the data samples has been randomly selected for the testing set while the remaining data points form the training set. Notice the data preprocessing procedure has been applied to the red wine samples and white wine samples independently and the testing set and the training set are the combination of the preprocessed results of the two species. A flow diagram representing the detailed data preprocessing is shown in figure 1.1.

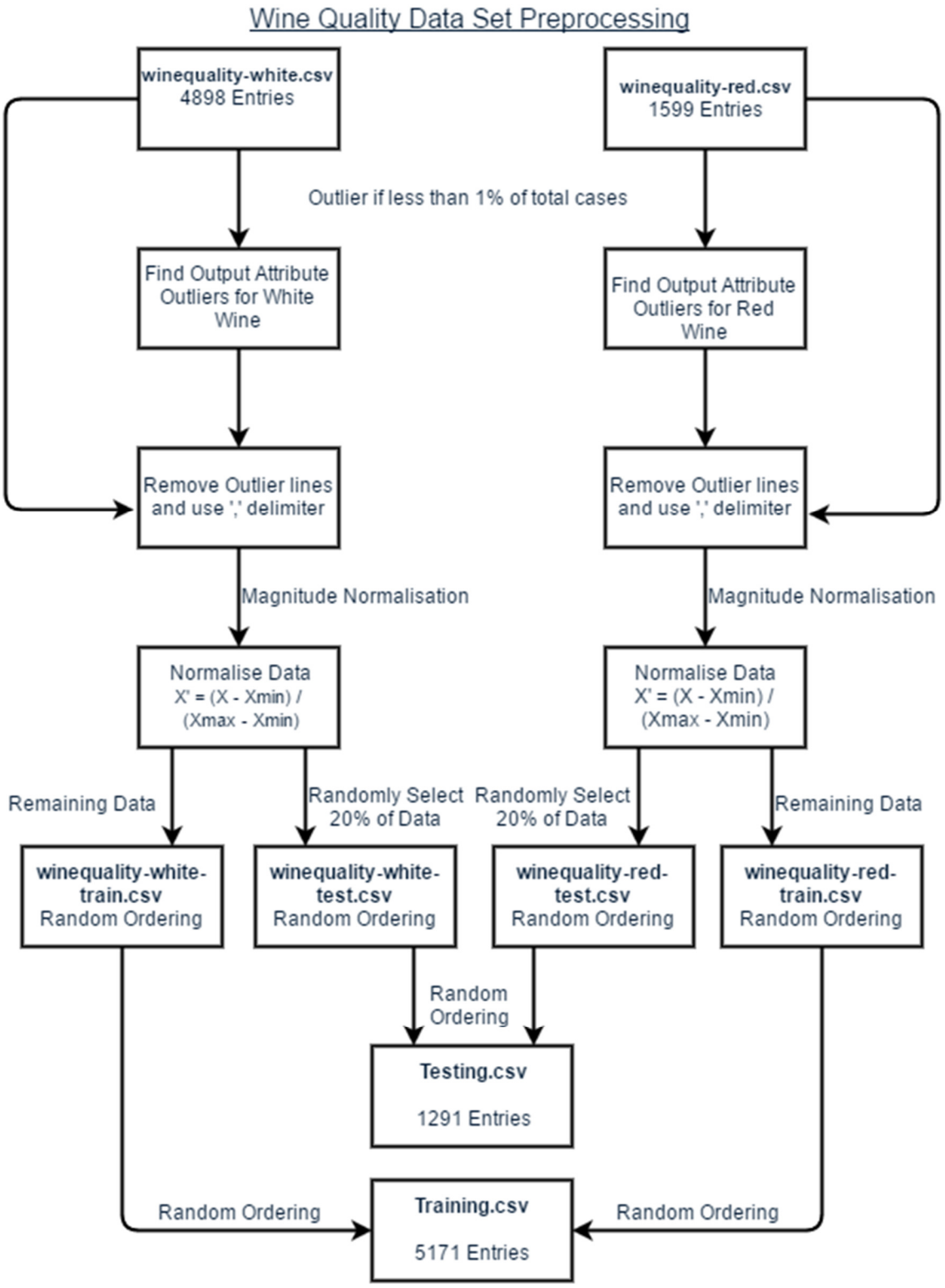

Figure 1.1: Flow Chart showing the detail of data preprocessing. Created by Kyle Lavorato. All rights reserved.

Solution Approach

Since each pattern has an expected output parameter (wine quality), supervised networks are preferred over unsupervised networks. Also, since it is unclear whether the data is linearly separable, a multilayer network is more plausible for this study. The network chosen is a backpropagation network with a total of 1 hidden layer. The wine quality parameter for each pattern is in the range between 3 and 8 after the removal of outliers. The input layer uses 11 nodes (one for each input parameter), and the output layer uses 2 nodes to indicate whether it is of average quality wine (quality rating from 3-5) or good quality (quality rating from 6-8). Finally, it has been determined that the hidden layer will use a total of 8 hidden nodes to further decrease the crosstalk effect. If each hidden node represents a hyperplane, the total number of hyperplanes required to separate these data points is 5 (equals 6 – 1) in the worst case. A Sigmoid function has been used at each processing node because it is continuous, invertible, monotonically increasing, differentiable everywhere, and asymptotically approaching to its saturation values as net approaches infinity. The detailed architecture of the network is shown in Figure 1.2:

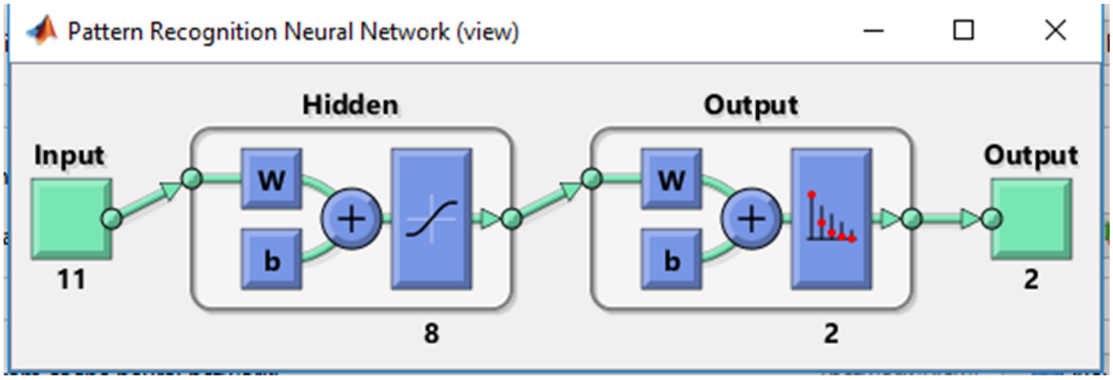

Figure 1.2: Matlab software showing the architecture of the backpropagation network.1 input layer of 11 nodes, 1 hidden layer of 8 nodes, and one output layer with 2 nodes. Created by Michelle Masci. All rights reserved.

3.0 Results and Analysis

Validation Criteria

To evaluate whether the ANN can accurately classify the data by wine quality, the expected accuracy for the test dataset will be approximately 98%. This was determined by the following equation:

$$(\#\ of\ patterns)\ >\ (\#\ of\ weights)\ /\ (1\ -\ expected\ accuracy\ on\ test)$$

where the number of patterns was (6497*0.8) and the number of weights is 11.

Results

The training and testing results were generated using Matlab. The confusion matrices are shown in figure 2.1

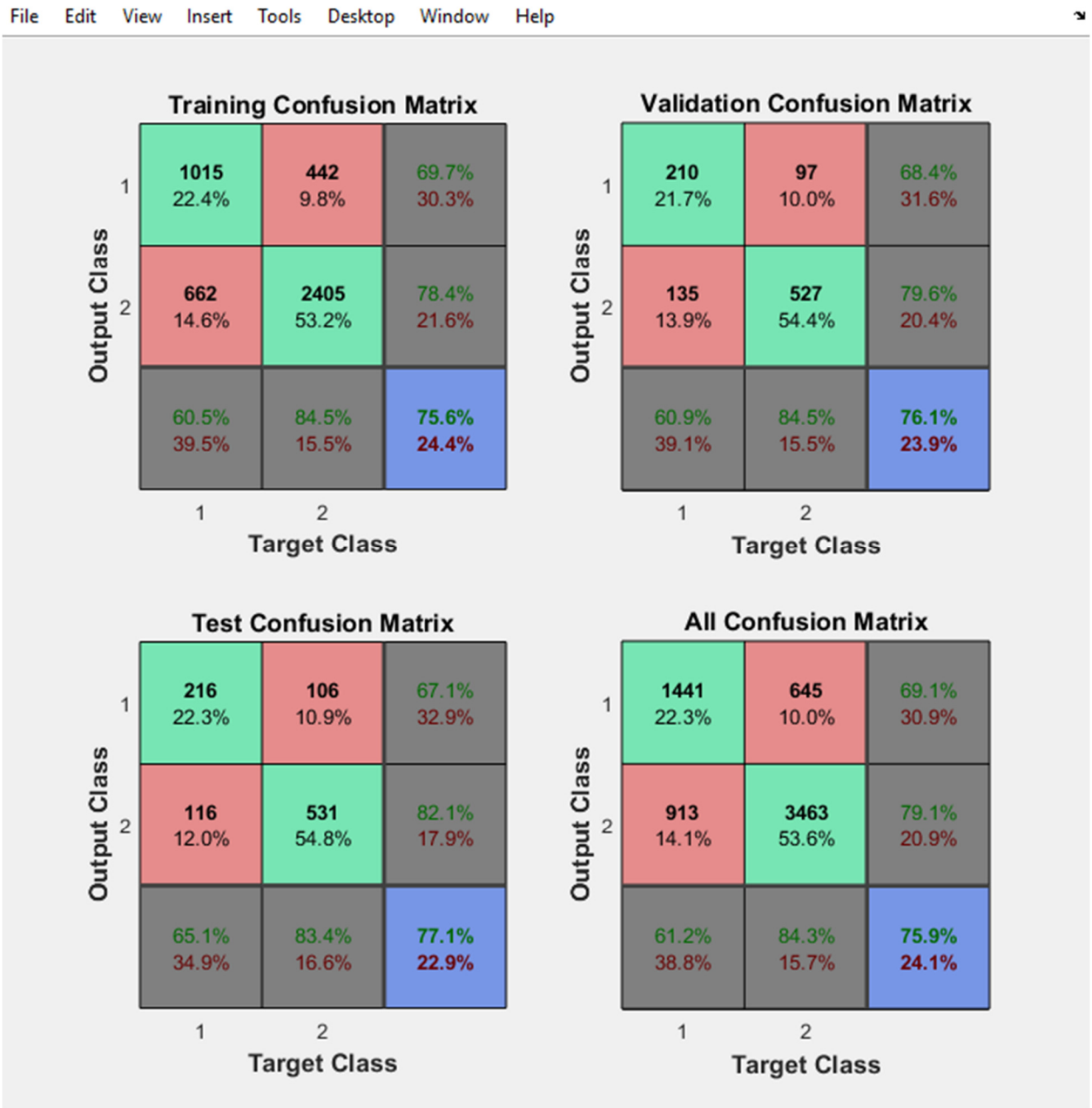

Figure 2.1: The confusion matrices generated by the Matlab software. Created by Michelle Masci. All rights reserved.

The classify accuracy is 75.6% for the training set and 77.1% for the testing set. Both are way below the expected accuracy, which is 98%.

This discrepancy is likely caused by improper choice of the number of hidden layers. It is also possible that the weights updating directions were stuck inside some local minima and/or the choice of momentum parameter (alpha) is also inappropriate.

As shown in Figure 2.1, the precision (positive predictive value) of the training set and testing set are 69.7% and 67.1%, respectively. The recall (true positive rate) is 60.5% for the training data and 65.1% for the testing data.

Class 1 (average wine):

$$Precision = (True\ Positives) / (True\ Positives + False\ Positives)$$

$$Precision_{training} = 1015 / (1015 + 442) = 0.6966$$

$$Precision_{testing} = 216 / (216 + 106) = 0.6708$$

$$Recall = (True\ Positives) / (True\ Positives + False\ Negatives)$$

$$Recall_{training} = 1015/ (1015 + 662) = 0.6052$$

$$Recall_{testing} = 216/ (216 + 116) = 0.6506$$

Figure 2.1 also displays the negative predictive value and the true negative rate of Class 2.

Class 2 (good wine):

$$Precision = (True\ Negatives) / (True\ Negatives + False\ Negatives)$$

$$Precision_{training} = 2405/ (2405 + 662) = 0.7841$$

$$Precision_{testing} = 531/ (531 + 116) = 0.8207$$

$$Recall = (True\ Negatives) / (True\ Negatives + False\ Positives)$$

$$Recall_{training} = 2405/ (2405 + 442) = 0.8447$$

$$Recall_{testing} = 531/ (531 + 106) = 0.8335$$

The reason that the "good" wine class has a better predictive value and true rate is because there were more samples that were classified as "good" (quality rating 6-8) than samples that were classified as "average" (quality rating 3-5). In fact, the ratio was almost 2:1, so the balance of our samples between the two classes was not ideal. Therefore, a possible way to have improved our neural network's accuracy was to balance the sample classes. This could have been done by either oversampling the minority class (in our case, the "average" wine) or adjusting our algorithm to include penalization, which would bias our model to pay more attention to the minority class.

Analysis with Previous Work

Previous work on the data set was done by Cortez, Paulo et al. in their paper *Modeling Wine Preferences By Data Mining From Physicochemical Properties*[3]. In their case, they compared the results and accuracy of a multiple regression model, a neural network (multi-layer perceptron), and a SVM (support vector machine)[3]. According to their findings, the SVM outperformed the multiple regression model and neural network[3]. The neural network for $Accuracy_{T=1.00}$ was between 84.7% and 88.8% and the accuracy for the SVM was between 86.8% and 89.0% [3]. For our purposes, we are interested in considering why their neural network performed significantly better than ours. Per their paper, the activation function used was:

$$\hat{y} = w_{o,0} + \sum_{j=I+1}^{o-1} \frac{1}{1 + \exp(-\sum_{i=1}^{I} x_i w_{j,i} - w_{j,0})} \cdot w_{o,i}$$

They also performed principle component analysis by performing a sensitivity analysis procedure to the model, after the training phase, to determine which input variables produced a high variance. Both these factors likely explained why their neural network outperformed ours.

4.0 Contributions and Future Work

Contributions

Through the efforts of this project, some advancements have been made past the work done by the authors of the dataset. The original authors did not do any preprocessing on the data before they sent it though a multilayer perceptron network. Through analysis, it has been determined that preprocessing can advance the quality of a network's results. Since preprocessing here was uncharted waters, many different levels of preprocessing were used before it was settled that the arrangement discussed in the preprocessing section above was the optimal level.

Multiple different bounds on outlier removal were experimented with, until finally it was discovered that removing data that occurs less than 1% of all cases optimizes results. This is because the red wine samples of quality 8 were being removed while white of the same quality were not, giving too few of that quality, reducing accuracy.

Next the normalization method needed to be chosen, and two were experimented with. Gaussian normalization was used first, which reduces values into a range of [-10…10]. This method while still improving over the non-normalized data, had issues due to leaving values in too large of a range. Magnitude normalization was settled on, which reduces values to the range [0…1]. This ended up improving accuracy more due to the more concise range, allowing the weights to not change as drastically. Regardless of the method, normalization was a huge improvement over the original data values. This is because the original data uses many different units, which made values be magnitudes apart from others. For example, some values were in the range of 10^{-3} and other were in 10^{3}.

The final contribution made to this study originated from a comment made from the authors. They stated that some input features are likely correlated so dimensionality reduction may prove useful. Therefore, once the other preprocessing was completed, an initial correlation analysis was conducted. This analysis was conducted through the custom MATLAB script, DataAnalysis.m. This script creates the graph shown in Figure 4.1. To get the most accurate view of the data values, the script uses all wine samples of quality 7 as input as the values change significantly in different qualities. The graph then indicates which input attributes are correlated, based on their distance difference on the graph. As highlighted, free sulfur dioxide and citric acid, as well as total sulfur dioxide and density are very closely correlated. This indicates that the dimensionality of the data set can at a minimum be reduced by two. With this information determined, it is highly suggested to conduct a more robust analysis on the data to determine any other potential reductions as there was not time in this study. Additionally, the data should then be run through a PCA network to eliminate the desired number of dimensions. This in turn will greatly reduce the training time of the network and will result in higher accuracy of the network.

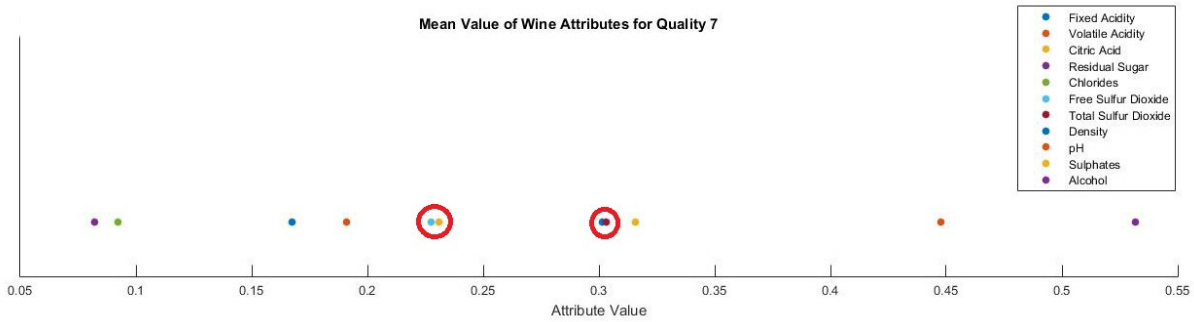

Figure 4.1: The correlation graph generated by the Matlab software. Created by Kyle Lavorato. All rights reserved.

Future Work

Future work can be done to both data and neural network architecture to improve classifying performance. Several possibilities are discussed.

Principal Component Analysis

One issue with the current training data is that there are too many input parameters or in other words the input vectors have too many components. This results in the developed neural network having many input nodes and many connection weights.

To construct a less computationally expensive network, it would be helpful to remove the least important input parameters from training and testing data. Namely, these features with the least wine quality variance can be removed since it simply means wine quality barely changes due to these features.

One option to achieve this is to use the Principal Component Analysis (PCA), which computes the n most important features by using a transformation matrix constructed from n unit eigenvectors correspond to the n largest eigenvalues of the sample covariance matrix.

Pre-clustering by Unsupervised Network

Another potential issue with the training data is the human rated wine quality values (between 0 and 10). Like everything else, human rated values could be subjective and suffering bias. For example, a Portuguese wine expert may tend to give high values to the wine quality of "Vinho Verde" while a different value may be given by a person from a different country. This will result in "cross talk" during training process. Namely, different training samples may require conflicting changes in the same weight. This will weaken the ability of the neural network to generalize and required to have more hidden nodes.

Unsupervised networks such as "k-means clustering" can be used to separate the training data into different clusters, which may be used for training one at a time.

Other Optimization Methods

Architecture of the neural network could be further optimized with the concept of "adaptive multilayer network" which allows pruning of unimportant nodes (these with small weights magnitudes and whose existences do not have significant effects at network outputs) and reintroduction of new nodes with random weights. Networks optimized in the same way will likely require fewer training samples with fewer input parameters, yet have faster training and better generalization abilities.

Finally, other optimization methods such as "simulated annealing" could also be considered with regard to escaping local minima.

Appendix A: Citations

[1] https://www.statista.com/topics/1541/wine-market/

Astray G, Castillo JX, Ferreiro-Lage JA, Gálvez JF, Mejuto JC. Artificial neural networks: a promising tool to evaluate the authenticity of wine redes neuronales: una herramienta prometedora para evaluar la autenticidad del vino. CyTA – Journal of Food 2010 May 2010;8(1):79-86

[2]http://journals2.scholarsportal.info.proxy.queensu.ca/details/19476337/v08i0001/79_annaptpeladv.xml

[3]http://www.sciencedirect.com/science/article/pii/S0167923609001377

Cortez, Paulo et al. "Modeling Wine Preferences By Data Mining From Physicochemical Properties". *Decision Support Systems* 47.4 (2009): 547-553. Web.

[4]https://www.thermofisher.com/ca/en/home/industrial/food-beverage/food-beverage-learning-center/beverage-testing-information/wine-testing-information/wine-quality-testing.html

[5] https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Appendix B: Program Code

- Code generated by Matlab, see attached: *BP_Neural_Network.m*
- Feature dimension analysis, see attached: DataAnalysis.m
- The code written to preprocess the data, see attached: Preprocess.py

Preprocess.py

```
import random, math



def read_freq(filename):
    """Read the occurrences of each quality value of wine; Return a list of outliers"""
    n = 0  # Total number of data entries
    data = {}  # Dictionary to hold occurrences of each quality
    outliers = []  # Any ratings that are outliers
    with open(filename, 'r') as f:
        next(f)  # Skip format line
        for line in f:
            n += 1
            key = int(str.strip(line)[-1])  # The quality value is the last character
            if key not in data:
                data[key] = 1  # First time that quality is found
            else:
                data[key] += 1  # Increase the quality count


    bound = int(n * 0.01)  # Data is an outlier if it is less than 3% of total cases
    for key in data:
        if data[key] < bound:
            outliers.append(key)


    # Output the frequencies of each type for reference
    for i in range(0, len(data)):
        low = 10
        for key in data:
            if key < low:
                low = key
```

```
        print(low, ":", data[low])
        del data[low]

    print("Pruned outliers:", outliers)
    return outliers
    # End read_freq()



def combineFile(file1, file2, output):
    """Take two files and combine the data into one file"""
    data = []
    n = 0
    # Read all the data from both files
    with open(file1, "r") as f:
        for line in f:
            data.append(line)
    with open(file2, "r") as f:
        for line in f:
            data.append(line)
    print("\nCombining:", file1, "and", file2)
    print(len(data), "data points detected")
    snOut = open(output, "w")
    for i in range(0, len(data)):
        index = random.randint(0, len(data) - 1)  # Write the elements to file in a
random order
        snOut.write(data.pop(index))
        n += 1
    snOut.close()
    print(n, "data points written randomly to", output)  # Print how many data points
were written to file
    # End combineFile()



def generateFile(data, filename):
```

```
    """Write all the passed in data to a file"""
    snOut = open(filename, "w")
    for i in range(0, len(data)):
        snOut.write(data[i])
    snOut.close()
    # End generateFile()


def floatTransform(data):
    """Transform data from ',' delimited string to floats"""
    mathData = []  # Hold numeric data version of the passed in data
    for line in data:
        nums = []
        values = line.split(',')  # Split line by ',' delimiter
        for v in values:
            nums.append(float(v))  # Convert to float before saving
        mathData.append(nums)  # List of float version of the line
    return mathData
    # End floatTransform()



def stringTransform(mathData):
    """Transform data back into string form with ',' delimiter"""
    stringData = []
    for i in range(0, len(mathData)):
        line = ""
        for j in range(0, len(mathData[i])):
            line += str(mathData[i][j]) + ","  # Add each float term to the string
with the delimiter
        line = line[:-1] + "\n"  # Remove extra ',' at the end and add newline
        stringData.append(line)
    return stringData
    # End stringTransform()
```

```
def normal(data):
  """Apply a Max/Min normalisation to the data"""
  mathData = floatTransform(data)  # Hold numeric data version of the passed in data

  maximum = [0.0] * (len(mathData[0]) - 1)  # The max value for each of the data values
  minimum = [99999.9] * (len(mathData[0]) - 1)  # The min value for each of the data values

  # Find the max and min for each data value
  for i in range(0, len(mathData)):
      for j in range(0, len(mathData[i]) - 1):
          if mathData[i][j] > maximum[j]:
              maximum[j] = mathData[i][j]
          if mathData[i][j] < minimum[j]:
              minimum[j] = mathData[i][j]

  # X' = (X - Xmin) / (Xmax - Xmin)
  for i in range(0, len(mathData)):
      for j in range(0, len(mathData[i]) - 1):
          mathData[i][j] = (mathData[i][j] - minimum[j]) / (maximum[j] - minimum[j])

  return stringTransform(mathData)
  # End Normal()


def gaussianNormal(data):
  """Apply a Gaussian Normalisation to the data"""
  mathData = floatTransform(data)  # Hold numeric data version of the passed in data

  mean = [0.0] * (len(mathData[0]) - 1)  # The mean for each of the data values
  stdDev = [0.0] * (len(mathData[0]) - 1)  # The standard deviation for each of the data values
```

```
    # Calculate the mean for each data value
    for i in range(0, len(mathData)):
        for j in range(0, len(mathData[i]) - 1):
            mean[j] += mathData[i][j]  # Running total sum
    for i in range(0, len(mean)):
        mean[i] = mean[i] / len(mathData)  # Calcuate final mean


    # Calculate the standard deviation for each data value
    for i in range(0, len(mathData)):
        for j in range(0, len(mathData[i]) - 1):
            stdDev[j] += (mathData[i][j] - mean[j])**2  # Running total sum of Std
Dev terms
    for i in range(0, len(stdDev)):
        stdDev[i] = math.sqrt(stdDev[i] / len(mathData))  # Calculate final Std Dev


    # Normalize the data: v' = (v - mean) / StdDev
    for i in range(0, len(mathData)):
        for j in range(0, len(mathData[i]) - 1):  # Start at one as first term cannot
be normalised
            mathData[i][j] = (mathData[i][j] - mean[j]) / stdDev[j]


    return stringTransform(mathData)
    # End gaussianNormal()



def preprocess(outliers, filename):
    """Preprocess the data set to remove outliers and divide to test and train set"""
    print("\nCreated test and train set for:", filename)
    data = []  # List to hold the data strings
    n = 0  # Counter variable
    with open(filename, 'r') as f:
        next(f)  # Skip format line
        for line in f:
```

```
            outlier = False  # Boolean to test if it is an outlier
            string = line.replace(";",",")  # Network is designed to accept comma
seperated values
            for o in outliers:
                if int(str.strip(string)[-1]) == o:  # Test if the data string is an
outlier
                    outlier = True
                    n += 1
            if not outlier:  # Prune line if it is an outlier
                data.append(string)  # Save all non-outlier lines
    print("Removed", n, "outliers")  # Print the number of outliers removed

    data = normal(data)  # Apply a Magnitude Normalisation to the data
    #data = gaussianNormal(data)  # Apply a Gaussian Normalisation to the data

    # Create filenames for output files
    outFileTrain = filename.replace(".csv", "-train.csv")
    outFileTest = filename.replace(".csv", "-test.csv")

    # Randomly select 20% of the data points to be the test data
    testData = []
    t = int(len(data) * 0.20)
    for i in range(0, t):
        index = random.randint(0, len(data) - 1)
        testData.append(data.pop(index))

    # Create the train and test files
    generateFile(data, outFileTrain)
    generateFile(testData, outFileTest)

    # Print the size of the files
    print("Traning set size:", len(data))
    print("Testing set size:", len(testData))
    # End preprocess()
```

```
## MAINLINE ##

print("Original Wine Quality v. Frequency")
print("[Quality] : [Frequency]\n")

print("White Wine\n")
whiteOutliers = read_freq("winequality-white.csv")

print("\nRed Wine\n")
redOutliers = read_freq("winequality-red.csv")

preprocess(whiteOutliers, "winequality-white.csv")
preprocess(redOutliers, "winequality-red.csv")

combineFile("winequality-white-train.csv", "winequality-red-train.csv", "training.csv")
combineFile("winequality-white-test.csv", "winequality-red-test.csv", "testing.csv")

combineFile("Testing.csv", "Training.csv", "Samples.csv")
```

Console Output:

```
Original Wine Quality v. Frequency
[Quality] : [Frequency]

White Wine

3 : 20
4 : 163
5 : 1457
6 : 2198
7 : 880
```

8 : 175

9 : 5

Pruned outliers: [3, 9]

Red Wine

3 : 10

4 : 53

5 : 681

6 : 638

7 : 199

8 : 18

Pruned outliers: [3]

Created test and train set for: winequality-white.csv

Removed 25 outliers

Traning set size: 3899

Testing set size: 974

Created test and train set for: winequality-red.csv

Removed 10 outliers

Traning set size: 1272

Testing set size: 317

Combining: winequality-white-train.csv and winequality-red-train.csv

5171 data points detected

5171 data points written randomly to training.csv

Combining: winequality-white-test.csv and winequality-red-test.csv

1291 data points detected

1291 data points written randomly to testing.csv

Combining: Testing.csv and Training.csv

6462 data points detected

6462 data points written randomly to Samples.csv

Process finished with exit code 0

DataAnalysis.m

```
data = csvread('resultsSeven.csv');  % Get the data from file


d = length(data(1,:)) - 1;  % Find index to ignore the quality output

for i = 1:11

   attribute = data(:,i);  % Get the column for each attribute

   attributeMean = mean(attribute);  % Find the mean of the attribute columns

   scatter(attributeMean,0.5,'filled')  % Plot the mean point

   hold on  % Prepare to plot more points

end


title('Mean Value of Wine Attributes for Quality 7')

xlabel('Attribute Value');

legend('Fixed Acidity','Volatile Acidity','Citric Acid','Residual Sugar','Chlorides','Free Sulfur Dioxide','Total Sulfur Dioxide','Density','pH','Sulphates','Alcohol','Location','NorthEast')

axis([0.05, 0.55, 0, 1])  % Set axis regions

set(gca,'ytick',[])  % Remove yAxis values

set(gca,'yticklabel',[])
```