Neural Network Classifier

White Wine Quality

Part 1: Overall Objective

My objective is to use the 11 variables to predict the quality of wine. The data type of quality of wine is a integer (0-10). I choose to use a multi-class classification to predict the 11 categories of quality. My goal is to build a neural network with maximum accuracy without over-fitting.

Part 2: Final ANN model in code

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from keras import models
from keras import layers
import datetime
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
from sklearn.metrics import confusion_matrix
import pandas as pd
from sklearn import preprocessing,model_selection
from keras.datasets import imdb
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from keras import backend as K
from keras.utils.np_utils import to_categorical


""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
data1= pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-
white.csv",delimiter=";",header = 0)
data2=data1.values

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
start_time=datetime.datetime.now()
X = data2[:,0:11]
Y = to_categorical(data2[:,11].astype(np.int))
X=preprocessing.StandardScaler().fit_transform(X)
X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.2)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Parameters Section
```

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
np.random.seed(97)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model = Sequential()
model.add(Dense(30, input_dim=11, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Train Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model.fit(X_train,Y_train, epochs=160, batch_size=50, verbose=2)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
scores = model.evaluate(X_test, Y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Part 3: Final ANN model and training algorithm in words

My final model is a simple stack of fully connected four layers (Dense layer) with "relu" activations. The input should have 11 dimensions, corresponding to the 11 predictors.

There are three intermediate layers with 30, 64, 500 hidden units respectively.

The last layer outputs the categorical prediction of the quality of wine. Since the output response has 11 categories, from 0 to 11, there are 10 (11-1) dummy variables created for it. As a result, each response has 10 columns, and therefore the output dimension is 10. Moreover, the last layer uses "softmax" as its activation function. It means the network will output a probability distribution over the 11 different output classes.

The learning process is configured in the compilation step. The optimizer, which is the learning method, is set to be "rmsprop". The loss function, which is the feedback signal used for learning, is set to be "categorical_crossentropy" because this is a multi-class classification. The metrics is also set to be "categorical_accuracy", which is the correct number of prediction over total number of prediction.

The learning process/training process is done by "fit()" function. I already divided dataset into training data and testing data. Here, I just use training data to train the neural network because I need to use testing data to measure the performance of this algorithm later.

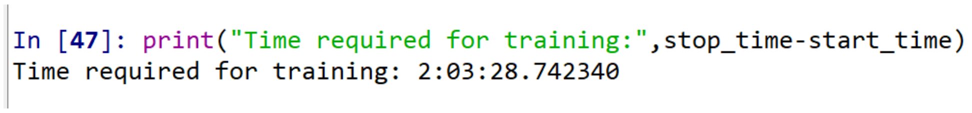
The number of epochs is set to be 160, and the batch size is set to be 50, which means that the model will go over the whole dataset for 160 times while it only learns and gains feedback from 50 data points rather than the whole dataset each time.

Part 4: Experimental plan for arriving at the final model

1. I tried the different layer numbers and the result shows that two or three hidden layers is enough for this dataset.
2. To make sure whether it is two or three layers and what the node numbers should be, I further divide training data into training data and validation data. Because what I care about is not the training accuracy but the validation accuracy. The training accuracy could be easily higher than 99%.
3. Therefore I wrote a loop as follows to test which combination of (1)number of nodes in first layer, (2)number of nodes in second layer, (3) number of nodes in third layer, determining whether the number of layers is two or three and (4)the choice for optimizer is best.

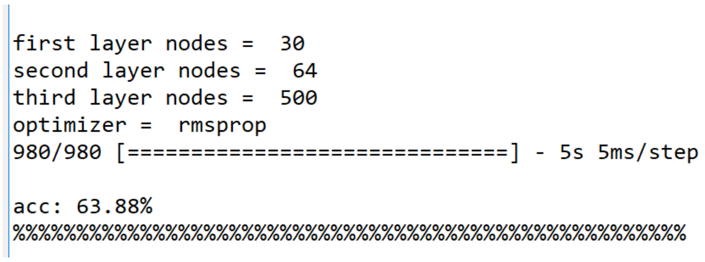
```
FL_Nodes = [11, 24, 30, 64, 100,500,1000]
SL_Nodes = [0, 32, 64, 100, 500,1000]
TL_Nodes = [0, 32, 64, 100, 500,1000]
optimizer_choice = ['rmsprop', 'adam']
for Num_Nodes_FL in FL_Nodes:
    for Num_Nodes_SL in SL_Nodes:
        for Num_Nodes_TL in TL_Nodes:
            for optimizer_choice1 in optimizer_choice:
                optimizer_choice1 = 'rmsprop'
                print("first layer nodes = ", Num_Nodes_FL)
                print("second layer nodes = ", Num_Nodes_SL)
                model = Sequential()
                model.add(Dense(Num_Nodes_FL, input_dim=11, activation='relu'))
                if Num_Nodes_SL > 0:
                    model.add(Dense(Num_Nodes_SL, activation='relu'))
                    if Num_Nodes_TL > 0:
                        model.add(Dense(Num_Nodes_TL, activation='relu'))
                        print("third layer nodes = ", Num_Nodes_TL)
                    else:
                        print("third layer nodes = ", 0)
                else:
                    print("third layer nodes = ", 0)
                print("optimizer = ", optimizer_choice1)
                model.add(Dense(10, activation='softmax'))
                model.compile(optimizer='rmsprop',
                              loss='categorical_crossentropy',
                              metrics=['accuracy'])
                model.fit(X_train, Y_train, epochs=200, batch_size=200, verbose=0)
                scores = model.evaluate(X_val, Y_val)
                print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
                print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
                print()
```

4. However, I met a challenge that it takes more than 2 hours to run only part of my code. The expected time of finishing all of them would be 12 hours. Therefore, I changed my strategy to narrow down the choices.



```
In [47]: print("Time required for training:",stop_time-start_time)
Time required for training: 2:03:28.742340
```

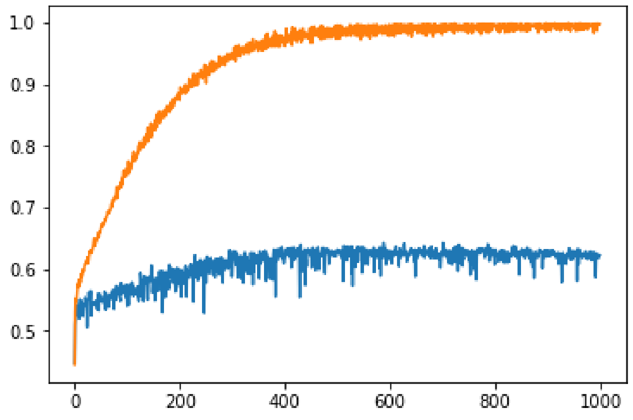
5. The best available result shows as follows, using 30, 64, and 500 nodes in the three layers and using “rmsprop” as optimizer method.



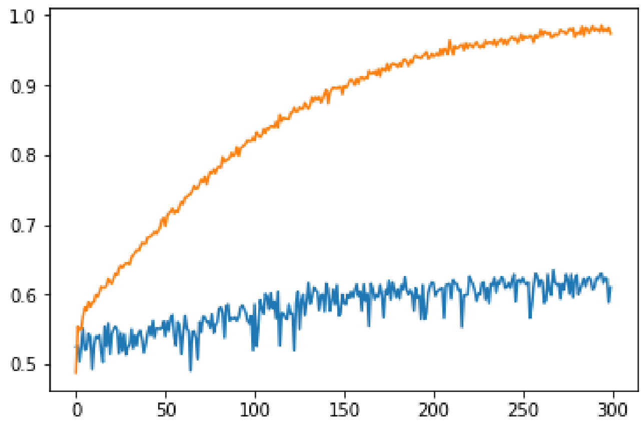
```
first layer nodes =  30
second layer nodes =  64
third layer nodes =  500
optimizer =  rmsprop
980/980 [==============================] - 5s 5ms/step

acc: 63.88%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6. I used validation set to see what’s the best number of epochs and batches. Since I can only get the “categorical_accuracy” for each epoch on a single batch size. I tried different batch size to see which combination of epoch number and batch size is great. I tried 200, 100, 50, 25 as batch size and I listed the result below. As a result, the best combination would be using 160 as epoch number and 50 as batch size.
   a. When batch size is 200, the result is as follows. As shown in the graph, the best epoch would be 440 while the validation accuracy is around 62%.

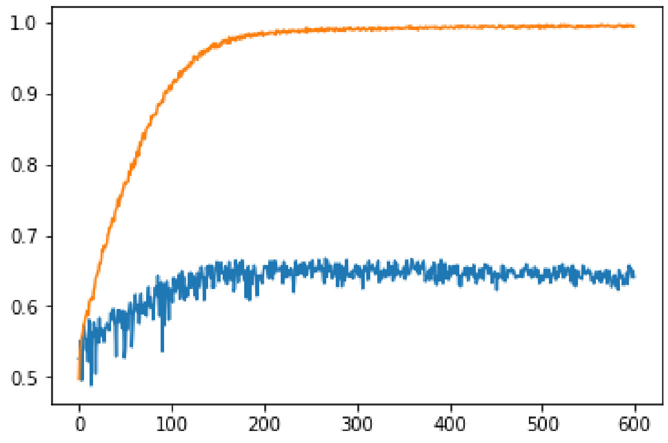
```
X_train, X_val, Y_train, Y_val = model_selection.train_test_split(X, Y, test_size=0.2)
history = model.fit(X_train,
                    Y_train,
                    epochs=1000,
                    batch_size=200,
                    validation_data=(X_val, Y_val))
plt.plot(history.history['val_categorical_accuracy'])
```



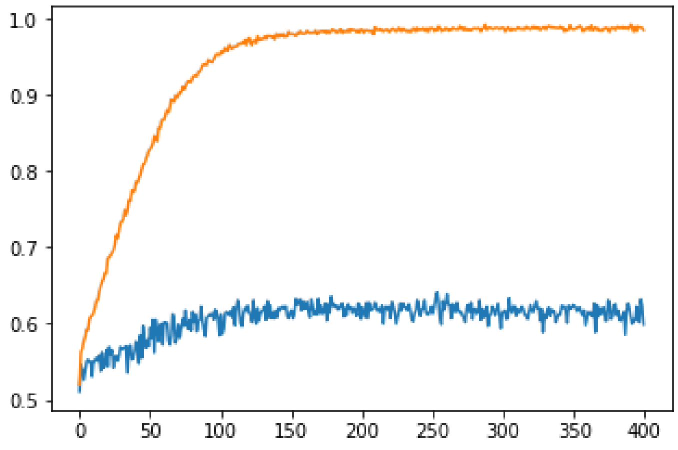
b. When batch size is 100, the result is as follows. As shown in the graph, the best epoch would be 280 while the validation accuracy is around 62%.



c. When batch size is 50, the result is as follows. As shown in the graph, the best epoch would be 160 while the validation accuracy is around 66%.

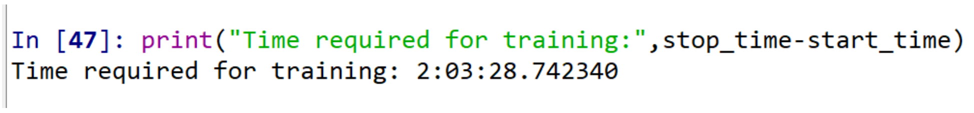


d. When batch size is 25, the result is as follows. As shown in the graph, the best epoch would be 170 while the validation accuracy is around 63%.



Part 5: How long it took to run all the models in your experimental plan

On my original plan, it takes more than 12 hours to run all my code. (The picture shown below is the time to run 1/6 of my codes).



```
In [47]: print("Time required for training:",stop_time-start_time)
Time required for training: 2:03:28.742340
```

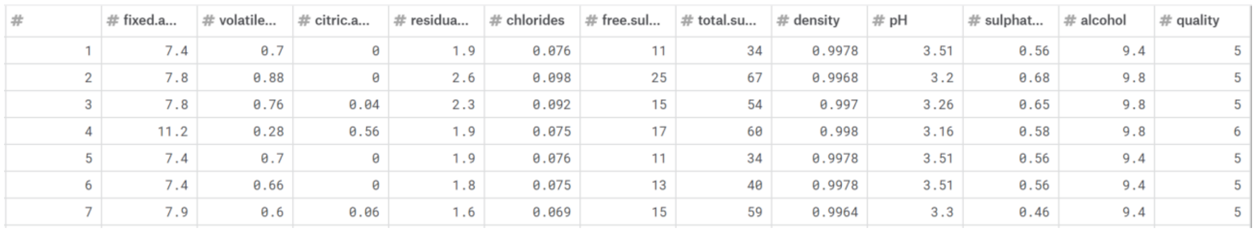
My adjusted experimental plan takes 5 hours to run.

```
start_time=datetime.datetime.now()
stop_time=datetime.datetime.now()
print("Time required for training:",stop_time-start_time)
```

Part 6: An explanation of the input variables and preprocessing steps

Predictors:

1. Fixed acidity (numeric): most acids involved with wine or fixed or nonvolatile (do not evaporate readily)
2. Volatile acidity (numeric): the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
3. Citric acid (numeric): Found in small quantities, citric acid can add 'freshness' and flavor to wines
4. Residual sugar (numeric): The amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet
5. Chlorides (numeric): The amount of salt in the wine
6. Free sulfur dioxide (numeric): The free form of so2 exists in equilibrium between molecular so2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine
7. Total sulfur dioxide (numeric): Amount of free and bound forms of s02; in low concentrations, so2 is mostly undetectable in wine, but at free so2 concentrations over 50 ppm, so2 becomes evident in the nose and taste of wine
8. Density (numeric): The density of water is close to that of water depending on the percent alcohol and sugar content
9. Ph (numeric): Describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the ph scale
10. Sulphates (numeric): A wine additive which can contribute to sulfur dioxide gas (s02) levels, wich acts as an antimicrobial and antioxidant
11. Alcohol (integer): The percent alcohol content of the wine



#	# fixed.a...	# volatile...	# citric.a...	# residua...	# chlorides	# free.sul...	# total.su...	# density	# pH	# sulphat...	# alcohol	# quality
1	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
2	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5
3	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5
4	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6
5	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
6	7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
7	7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5

Response/output variable:

Quality (int): Output variable (based on sensory data, score between 0 and 10)

Preprocessing:

After using “pd.read_csv” to get the dataset, I use “.values” to change the data format.

Since the data all have the same length and all are numeric numbers. I directly use slice to divide the dataset into two parts, predictors and response: the first 11 variables are predictors while the last variable is response.

Since “quality of wine” has 11 categories, I use “to_categorical” function from “keras.utils.np_utils” package to change it to 10-column dummy variables.

Since “quality of wine” shows like integers, I change it into float.

Moreover, It would be problematic to use variables in widely different ranges in neural network. Therefore, I use the “preprocessing.StandardScaler().fit_transform” function to do the feature-wise normalization, centering them around 0 and having a unit standard deviation. I also tried “preprocessing.normalize” function to do it, however, the result is not that good.

Next, I divide those data into training data set, and test data set.

Part 7: An explanation of metrics and justification for this choice

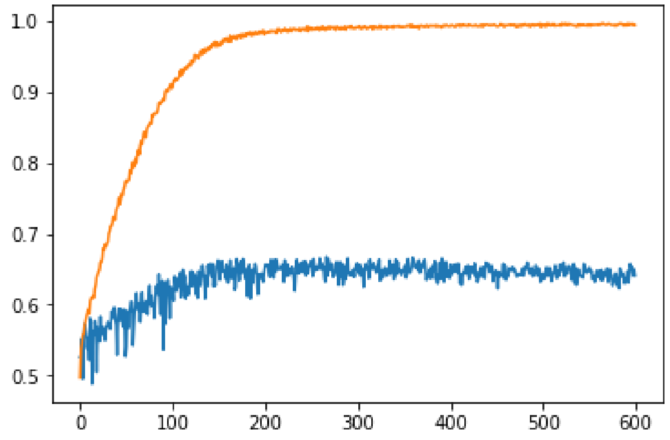
I choose “categorical_accuracy” to be my metric because this is a multi-class classification. This measure the number of correct predictions over the number of total predictions.

Part 8: An explanation of method to validate the model

Cross validation

To evaluate my neural network, I used validation set to validate my approach. When tuning the parameters of the model, such as batch size, epoch number, optimizer type, layer number, node number, I used the accuracy of validation set as the performance indicator because it is not reasonable to rely on training data for we can always overfitting and get an almost 100% accuracy of training set. For each training set, I set epoch parameter to a large number. Therefore, I can see the overall pattern across different epoch number rather than a variable single trial.

According to my best model, as shown below, the validation accuracy is 66%.

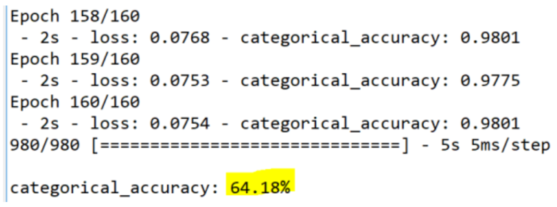


Training set & test set

Moreover, I divided the data into training set and test set at very beginning. I just used training data to train the neural network and used the test set to evaluate its performance. The resulting test "categorical_accuracy" is 64.8%.

Part 9: Your results in terms of appropriate metrics for the objective and problem

The training "categorical_accuracy" is 97.6%.The cross validated "categorical_accuracy" is 66%. The testing "categorical_accuracy" is 64.18%.



```
Epoch 158/160
 - 2s - loss: 0.0768 - categorical_accuracy: 0.9801
Epoch 159/160
 - 2s - loss: 0.0753 - categorical_accuracy: 0.9775
Epoch 160/160
 - 2s - loss: 0.0754 - categorical_accuracy: 0.9801
980/980 [==============================] - 5s 5ms/step

categorical_accuracy: 64.18%
```

Part 10: Limitations and Considerations

A important thing I learned in this model is that, to avoid overfitting, model must be trained by training data, be tuned by a clean validation set, and strictly use testing data to see the result. Nothing can be deteriorated.

In this classification model, I tried to adjust the following parameters or settings:

1. number of layers
2. number of nodes in each layer
3. optimizer type
4. epoch number
5. batch size
6. methods of data preprocessing: “preprocessing.StandardScaler().fit_transform” or “preprocessing.normalize”

I did not have time to go through other parameters such as learning rate.

Due to the speed limit, I did not have enough time to try more layers (>4) and loop through more node number (>1000). Similarly, I can only use a validation set to tune the parameters rather than use k-fold cross validation because of the low speed.

The next step would be learning more functions, packages, and theories about how to explore more possibilities efficiently, and how to design everything more scientifically.