ANN Classifier Assignment Assignment 3
Xiao(Megan)Ling

1. **Object**
   We are going to use UCI red wine dataset predict white wine quality using an Artificial Neural Network. With given data points about Portuguese “Vinho Verde” wine, such as fixed acidity, residual sugar, etc., we will be able to determine wine qualities based on a model designed.

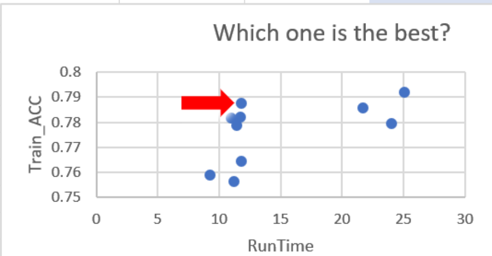
2. **The Final Model & Training Algorithm**

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Final Model Application
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
model.add(Dense(12, activation='relu'))
model.add(Dense(1,activation = 'sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Nadam', metrics=['accuracy'])
new = model.fit(x_test, y_test, epochs=80, batch_size=16, verbose=0)
train_acc_new = new.history["acc"]
print(max(train_acc_new))
```

The Final Model to predict the wine quality is a binary classification model, whih has 2 hidden layers using Nadam as the model optimizer, binary_corssentropy as loss function and Accuracy as metrics. The training algorithm here is Artificial Neural Network Algorithms which consists of different layers with weighted connections for analyzing and learning data.

3. **The experimental plan for arriving at the final model and How long it took to run all the models in your experimental plan**

Layers	Optimizer	Epochs	Run Time	train_acc	Validate_acc
3	Adagrad	200	11.81836581	0.7644817	0.659056323
3	adam	500	25.09286237	0.791920728	0.659056323
2	nadam	200	11.76551247	0.787728657	0.659056323
2	RMSPROP	500	21.65629768	0.785823168	0.659056323
3	nadam	200	11.73660517	0.782012195	0.659056323
2	adam	200	10.96820211	0.781631095	0.659056323
2	adam	500	23.9991014	0.77934451	0.659056323
3	adam	200	11.44937444	0.778582317	0.659056323
2	RMSPROP	200	9.263513803	0.758765242	0.659056323
2	Adagrad	200	11.16411638	0.756478657	0.659056323



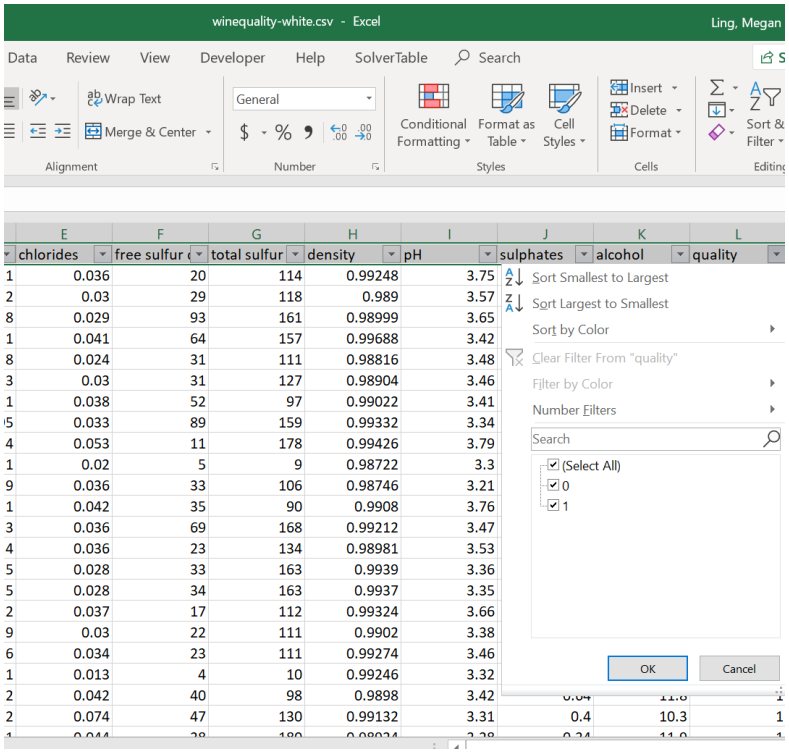
Select Nadam Model
2 hidden layers
200 Epochs
With 0.7699 Train Accuracy

To arrive at the final model, I changed parameters such as the number of Epochs, the size of the hidden layers, and the model compiler optimizers.

First, I designed model with RMSprop as compile optimizers with Epochs = 500 and 2 hidden layers. The RMSprop is suitable selection since it divides the learning rate by exponential decaying average of squared gradients. I applied the same parameters on different model optimizers, Nadam and Adam. I chose Adam because it quires low memory and Nadam is Adam RMSprop with Nesterov

momentum. Also, I include Adagard. It adapts the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features.
The considered time factor, generally, models with 500 epochs run slower than models with 200 epochs but with higher training accuracy. However, some models with 200 epochs and additional hidden layers can reach similar level training accuracy and save time. As a result, I selected the Nadam Model with 2 hidden layers and 200 epochs as my final model.

4. An explanation of the input variables and any preprocessing steps you took



To figure out the propitiate separate point for binary classification, I calculated the mean of the qualities, and it returns around 5.6. Consequently, I set 0 – 5 wine quality as low quality and 6 – 10 as high wine quality.

The challenge of binary classification is to put data into right categories. I tried to use pd.replace to change target variables Y into two categories 0 and 1, which means low and high qualities. But it costs me a long time to figure out the Python function. Then I realized since the white wine dataset is a relatively small data set, it would nice and quick to use Excel manually categorize qualities into 0 and 1.

```
"""
Pretreat Data Section
"""

x_data = data[x_vars]
y_data = data[y_vars]

#print (len(x_data))
#print (len(y_data))

x_data_MinMax = preprocessing.MinMaxScaler()

x_data.values
x_data = np.array(x_data).reshape((len(x_data), 11))

y_data.values
y_data = np.array(y_data).reshape((len(y_data), 1))

x_data = x_data_MinMax.fit_transform(x_data)

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.33, random
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_

x_train.mean(axis=0)
y_train.mean(axis=0)
x_test.mean(axis=0)
y_test.mean(axis=0)
"""
```

I also scaled variables x_data, which contains 11 variables such as chlorides, free sulfur dioxide, total sulfur dioxide etc. to the same measures. For target variable y_data set, I already changed them into binary values. So, there is no need to scale target variable dataset. Then splitting the dataset into train, test and validate datasets through train_test_split function.

6. An explanation of your model, metrix and justification for your choice

```
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
model.add(Dense(12, activation='relu'))
model.add(Dense(1,activation = 'sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Adagrad', metrics=['accuracy'])
```

To prevent overfitting, I selected a very small network with three hidden layers, each with 12 units. The network ends with a single unit and no activation. Since we are building up a binary classifier model, binary_crossentroy loss function is wildly applied when compiling a binary classifier network. Also, I add activation=’sigmoid’ in to the last model layer since ‘sigmoid’ indicates binary results. To monitor the training result, accuracy metric is introduced to show the difference between the predictions and the targets.

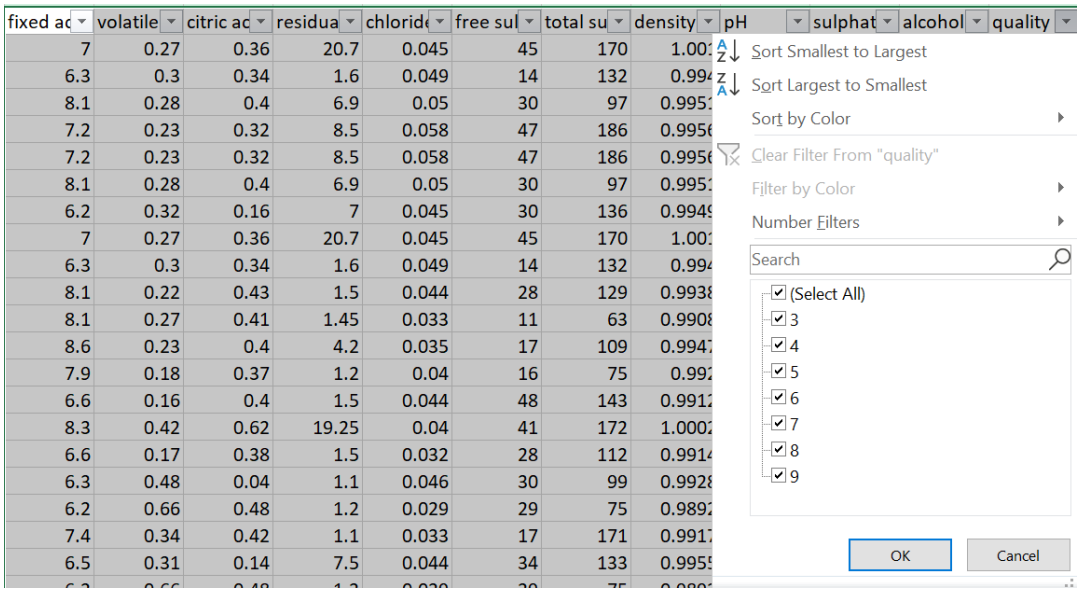
```
history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    epochs=200, batch_size=50, verbose=0)

train_loss = history.history["loss"]
validate_loss = history.history["val_loss"]
train_acc = history.history["acc"]
validate_acc = history.history["val_acc"]
```

For binary classification, it is prevalent to monitor binary_crossentroy loss and accuracy. As expected, the binary classification model accuracy is not as high as categorical_crossentrop. Binary_crossentroy applied when the data has two target classes, in this case are 0 and 1. The accuracy monitors if the model predict accurate results as the y from the test.set.

With the best model selected, nearly 80% of model prediction are correct. It is not a bad result compared with the complexity of building a 10 class multi-classification and bearing the risks of

overfitting. Plus, when we digging into the Excel data, we found out the lowest quality level of the Portuguese white wine is 3, which means it is unnessarry to build a 10 class model which has 3 empty class to fit. The multi-classification model prediction result would probably be a overfitting result.



7. Business Insights

Even though binary classification model sacrificed some accuracy in prediction, from a practical perspective, I believe a binary classification model is a good fit for people who wants to invest in the Portuguese “Vinho Verde” wine business. As mentioned before, the white wine is seldomly categorized as a quality 1 level wine, which means the white wine producer would not put the actual low quality wine into the market. Plus, the wine quatliy system within EU indicates that tasting wine quality should within the “high” level boundy. So, the white wine quality around 5-6 will probably sale in open market instead of in auction or private sales. Plus, based on demographic data and common sense, people who consume white wine has higher education, better knowledge of tasting, and higher income. Consumers and investors would care more about those wines who graded as good quality. A quick model would help them know their wine qualities within seconds.

CODE

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from keras.models import Sequential
from keras.layers import Dense
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import time

import sys
if "C:\\My_Python_Lib" not in sys.path:
    sys.path.append("C:\\My_Python_Lib")

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
start = time.time()

data = pd.read_csv("winequality-white.csv",
                   header=0,
                  names = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
                           'chlorides', 'free sulfur dioxide',
                           'total sulfur dioxide', 'density', 'pH',
                           'sulphates', 'alcohol', 'quality'])

x_vars = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
                           'chlorides', 'free sulfur dioxide',
                           'total sulfur dioxide', 'density', 'pH',
                           'sulphates', 'alcohol',]

y_vars = ['quality']
```

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""


x_data = data[x_vars]
y_data = data[y_vars]

#print (len(x_data))
#print (len(y_data))

x_data_MinMax = preprocessing.MinMaxScaler()

x_data.values
x_data = np.array(x_data).reshape((len(x_data), 11))

y_data.values
y_data = np.array(y_data).reshape((len(y_data), 1))

x_data = x_data_MinMax.fit_transform(x_data)

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.33, random_state=7)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=7)

x_train.mean(axis=0)
y_train.mean(axis=0)
x_test.mean(axis=0)
y_test.mean(axis=0)
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section - RMSPROP
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""


model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
model.add(Dense(12, activation='relu'))
```

```
model.add(Dense(1,activation = 'sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Adagrad', metrics=['accuracy'])
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
ACC & LOSS validation
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    epochs=200, batch_size=50, verbose=0)
train_loss = history.history["loss"]
validate_loss = history.history["val_loss"]
train_acc = history.history["acc"]
validate_acc = history.history["val_acc"]
end = time.time()
print(end - start)
print(max(train_acc))
print(min(validate_acc))
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Final Model Application
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
model.add(Dense(12, activation='relu'))
model.add(Dense(1,activation = 'sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Nadam', metrics=['accuracy'])
new = model.fit(x_test, y_test, epochs=80, batch_size=16, verbose=0)
train_acc_new = new.history["acc"]
print(max(train_acc_new))
```