Neural Network Regressor

Red Wine Quality Prediction

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 numeric number (0-10). Therefore, I choose to use regression method. My goal is to build a neural network with minimum mean squared error.

Part 2: Final ANN model in code

```
################################################################################
Import Libraries Section
################################################################################
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
import datetime
from keras import backend as K
import matplotlib.pyplot as plt



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

################################################################################
Pretreat Data Section
################################################################################
start_time=datetime.datetime.now()
X = data2[:,0:11]
Y = data2[:,11].astype(np.float)
X = preprocessing.normalize(X, axis = 0)
X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.2)

################################################################################
Parameters Section
################################################################################
np.random.seed(1995)

################################################################################
```

```
Define Model Section
######################################################################################
model = Sequential()
model.add(Dense(150, input_dim=11, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

######################################################################################
Train Model Section
######################################################################################
model.fit(X_train,Y_train, epochs=1000, batch_size=32, verbose=0)

######################################################################################
Show output Section
######################################################################################
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
predictions = model.predict(X_test)
rounded_p = [round(x[0]) for x in predictions]
print("\n%s: %.2f%%" % ("accuracy", accuracy_score(Y_test, rounded_p) * 100))
```

Part 3: Final ANN model and training algorithm in words

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

There are two intermediate layers with 150 and 32 hidden units respectively. The third layer outputs the scalar prediction of the quality of wine, and therefore the output dimension is one.

The learning process is configured in the compilation step. The optimizer, which is the learning method, is set to be “adam”. The loss function, which is the feedback signal used for learning, is set to be “mean squared error”. The metrics is also set to be “mean squared error”.

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 1000, and the batch size is set to be 32, which means that the model will go over the whole dataset for 1000 times while it only learns and gains feedback from 32 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 hidden layers is enough for this dataset.

2. For the two hidden layers, I need to figure out how many nodes should be set and what is the optimizer. 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, and (3)the choice for optimizer is best.

```
FL_Nodes = [5, 10, 11, 30, 50, 100, 150]
SL_Nodes = [0, 2, 4, 6, 8, 16, 32, 64, 100]
optimizer_choice = ['rmsprop', 'adam']
for Num_Nodes_FL in FL_Nodes:
    for Num_Nodes_SL in SL_Nodes:
        for optimizer_choice1 in optimizer_choice:
            print("first layer nodes = ", Num_Nodes_FL)
            print("second layer nodes = ", Num_Nodes_SL)
            print("optimizer = ", optimizer_choice1)
            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'))
            model.add(Dense(1))
            model.compile(optimizer=optimizer_choice1, loss='mse', metrics=['mse'])
            model.fit(X, Y, epochs=1000, batch_size=32, verbose=0)
            scores = model.evaluate(X, Y)
            print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
            predictions = model.predict(X)
            rounded_p = [round(x[0]) for x in predictions]
            print("\n%s: %.2f%%" % ("accuracy", accuracy_score(Y, rounded_p) * 100))
            print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
            print()
```

3. The best result shows as follows, using 150 and 32 nodes in the two layers and using “adam” as optimizer method.

```
first layer nodes =  150
second layer nodes =  32
optimizer =  adam
1599/1599 [==============================] - 1s 881us/step
mean_squared_error: 39.86%
accuracy: 60.60%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

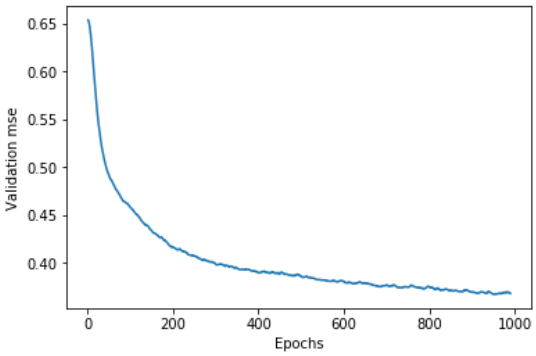
4. I used cross validation to see what’s the best number of epochs. After I plot cross validated “mean squared error” according to different epoch number, I found that 1000 is a good epoch number. And I tried some batch size and found that 32 is a good number.

```
k = 5
num_val_samples = len(X_train) // k
K.clear_session()
num_epochs = 1000
all_mse_histories = []
for i in range(k):
    print('processing fold #', i)
    val_data = X_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = Y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [X_train[:i * num_val_samples],
         X_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [Y_train[:i * num_val_samples],
         Y_train[(i + 1) * num_val_samples:]],
        axis=0)

    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=32, verbose=0)
    mse_history = history.history['mean_squared_error']
    all_mse_histories.append(mse_history)

average_mse_history = [
    np.mean([x[i] for x in all_mse_histories]) for i in range(num_epochs)]
import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mse_history) + 1), average_mse_history)
plt.xlabel('Epochs')
plt.ylabel('Validation mse')
plt.show()

def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points
smooth_mse_history = smooth_curve(average_mse_history[10:])
plt.plot(range(1, len(smooth_mse_history) + 1), smooth_mse_history)
plt.xlabel('Epochs')
plt.ylabel('Validation mse')
plt.show()
```



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

It takes 4 hours to run the codes alone. I calculated it by relevant functions as follows. However, it took me more than 10 hours to complete the whole assignment.

```
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 (numeric): The percent alcohol content of the wine

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Response/output variable:

Quality (numeric): 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” 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.normalize” function to do the feature-wise normalization, centering them around 0 and having a unit standard deviation.

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 “mean squared error” to be my metric. Since mean squared error is a common metric for regressor and it is sensitive if the difference between target response and prediction is large.

Part 8: An explanation of method to validate the model

Cross validation

To evaluate my neural network, I used k-fold validation to validate my approach.

To do this, I firstly split the data into a training set and a validation set in the preprocessing step. Then I set K equals to 5, which means the available data is split into 5 partitions. In addition, I set "num_epochs" to be 1000 because I want to see what number of epochs produces the lowest "mean squared error". I then builds K models, training each one on K-1 partitions while evaluating one the remaining partition. Next, I used the average "mean squared error" of those 5 models to evaluate the overall performance of my model. The result averaged mse is 0.4.

```
k = 5
num_val_samples = len(X_train) // k

num_epochs = 1000
all_scores = []
for i in range(k):
    print('processing fold #', i)
    val_data = X_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = Y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [X_train[:i * num_val_samples],
         X_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [Y_train[:i * num_val_samples],
         Y_train[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    history= model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs)
    val_mse = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mse)


all_scores
np.mean(all_scores)
```

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 mse is also 0.40.

```
....
...: model.fit(X_train,Y_train, epochs=1000, batch_size=32, verbose=0)
...:
...: """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
...: Show output Section
...: """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
...: print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
...: predictions = model.predict(X_test)
...: rounded_p = [round(x[0]) for x in predictions]
...: print("\n%s: %.2f%%" % ("accuracy", accuracy_score(Y_test, rounded_p) * 100))

mean_squared_error: 40.21%

accuracy: 60.62%
```

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

The cross validated and test mean squared error are both 0.40. But in the process, the training mean squared error could be low at 0.34.

In addition, although the original “accuracy” is not appropriate for regressor, I manually rounded my results to integers to make it meaningful. The manipulated accuracy is above 60%。

```
Epoch 997/1000
 - 0s - loss: 0.3583 - mean_squared_error: 0.3583
Epoch 998/1000
 - 0s - loss: 0.3694 - mean_squared_error: 0.3694
Epoch 999/1000
 - 0s - loss: 0.3621 - mean_squared_error: 0.3621
Epoch 1000/1000
 - 0s - loss: 0.3485 - mean_squared_error: 0.3485
Out[153]: <keras.callbacks.History at 0x21e96ff7e48>
```

Full data set:

```
mean_squared_error: 0.40
accuracy: 63.35%
```

Test data set:

```
mean_squared_error: 0.40
accuracy: 60.23%
```