Thomas Courtney

Artificial Intelligence

Professor Blossom

19 September 2021

Artificial Neural Network Regressor Assignment

Summary

The assignment called to produce a classifier neural network that would successfully predict wine quality. In this assignment the two datasets, red and white wine, were trained separately and got different results. The following document explores and answers questions regarding the models created to classify the two datasets.

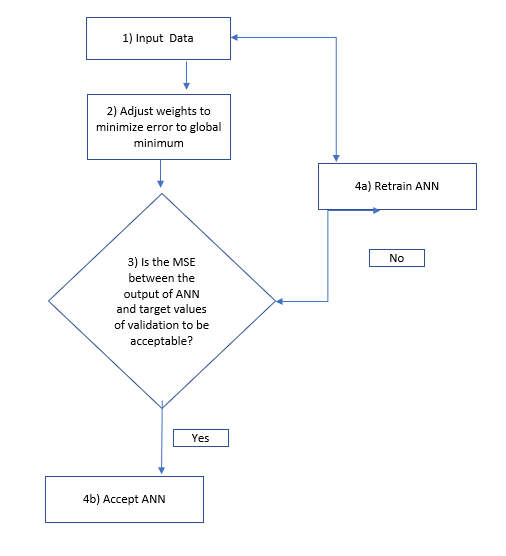
Introduction

Artificial Neural Network (ANN) is one of the most common artificial intelligence coding designs in the world. Regression ANNs predict output variables by the function of the inputs. ANNs are used throughout many applications, mostly involving predictive classification. Some examples of how to use predictive classification to detect faces or image recognition or voice recognition.

The code attached predicts the wine quality of the different wines. The data set had multiple columns in the data set to help us create the prediction. The different columns contain the following data: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol. The values of these numbers are vital in determining wine quality.

Body

Artificial Neural Network Flowchart



# -\*- coding: utf-8 -\*-

#"""

#Created on Tue Sep 14 20:47:42 2021

#@author: thoma

#"""

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

from matplotlib import pyplot

import datetime

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Load Data Section

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" #load dataset

red\_df = pd.read\_csv("C:/Users/thoma/OneDrive/Documents/AI/M3/winequality-red.csv", delimiter = ";", header=0)

white\_df = pd.read\_csv("C:/Users/thoma/OneDrive/Documents/AI/M3/winequality-white.csv", delimiter = ";", header=0)

# drop any missing values from the datasets

red\_df= red\_df.dropna()

red\_df=red\_df.values

white\_df = white\_df.dropna()

white\_df=white\_df.values

#identify independent and dependent variables

red\_x = red\_df[:,0:11]

red\_y = red\_df[:,11]

white\_x = white\_df[:,0:11]

white\_y = white\_df[:,11]

# Start the timer for run time

start\_time = datetime.datetime.now()

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

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" #Scale the x variables for the red data set and white data set red\_x\_MinMax = preprocessing.MinMaxScaler()

red\_x\_MinMax = preprocessing.MinMaxScaler()

red\_x = np.array(red\_x).reshape((len(red\_x), 11))

red\_y = np.array(red\_y).reshape((len(red\_y), 1))

white\_x\_MinMax = preprocessing.MinMaxScaler()

white\_x = np.array(white\_x).reshape((len(white\_x), 11))

white\_y = np.array(white\_y).reshape((len(white\_y), 1))

#transform the x variables into the MinMax function

red\_x = red\_x\_MinMax.fit\_transform(red\_x)

red\_x = red\_x\_MinMax.fit\_transform(red\_x)

white\_x = white\_x\_MinMax.fit\_transform(white\_x)

white\_x = white\_x\_MinMax.fit\_transform(white\_x)

#assign the training and test data sources

red\_x\_train, red\_x\_test, red\_y\_train, red\_y\_test = train\_test\_split(red\_x,red\_y, test\_size = 0.10)

white\_x\_train, white\_x\_test, white\_y\_train, white\_y\_test = train\_test\_split(white\_x,white\_y, test\_size = 0.25)

#fit the scaled data

sc = StandardScaler(with\_mean=True, with\_std = True)

sc.fit(red\_x\_train)

sc.fit(white\_x\_train)

red\_x\_train\_std = sc.transform(red\_x\_train)

red\_x\_test\_std = sc.transform(red\_x\_test)

white\_x\_train\_std = sc.transform(white\_x\_train)

white\_x\_test\_std = sc.transform(white\_x\_test)

#make the y variables categorical for our classification problem red\_y\_train = to\_categorical(red\_y\_train)

red\_y\_test = to\_categorical(red\_y\_test)

white\_y\_train = to\_categorical(white\_y\_train)

white\_y\_test = to\_categorical(white\_y\_test)

print("red\_x\_train.shape", red\_x\_train.shape)

print("len(red\_y\_train)", len(red\_y\_train))

#print("train\_lebels", red\_y\_train)

print("white\_x\_train.shape", white\_x\_train.shape)

print("len(white\_y\_train)", len(white\_y\_train))

#print("white\_y\_train", white\_y\_train)

print("red\_x\_test.shape", red\_x\_test.shape)

print("len(red\_y\_test)", len(red\_y\_test))

#print("red\_y\_test", red\_y\_test)

print("white\_x\_test.shape", white\_x\_test.shape)

print("len(white\_y\_test)", len(white\_y\_test))

print("white\_y\_test", white\_y\_test)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Define Model Section

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" #initialize the model for the red wine dataset

network\_red = tf.keras.models.Sequential()

#Add the first hidden layer specifying the input shape

network\_red.add(tf.keras.layers.Dense(200, activation='relu')) #Add a second hidden layer

network\_red.add(tf.keras.layers.Dense(200, activation='relu'))

network\_red.add(tf.keras.layers.Dense(200, activation='softmax'))

network\_red.add(tf.keras.layers.Dense(200, activation='sigmoid'))

network\_red.add(tf.keras.layers.Dense(120, activation='relu')) #Add the output layer

network\_red.add(tf.keras.layers.Dense(9, activation='softmax')) #compile the model

network\_red.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

#initialize the model for the white wine dataset

network\_white = tf.keras.models.Sequential()

#Add the first hidden layer specifying the input shape

network\_white.add(tf.keras.layers.Dense(300, activation='relu'))

network\_white.add(tf.keras.layers.Dense(300, activation='relu'))

network\_white.add(tf.keras.layers.Dense(300, activation='relu'))

network\_white.add(tf.keras.layers.Dense(150, activation='relu')) #Add the output layer

network\_white.add(tf.keras.layers.Dense(10, activation='softmax')) #compile the model

network\_white.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Fit Model Section

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" #Train the models

red\_train = network\_red.fit(red\_x\_train\_std, red\_y\_train, epochs=750, batch\_size=1000, verbose=2, validation\_split = 0.6)

white\_train = network\_white.fit(white\_x\_train\_std, white\_y\_train, epochs=750, batch\_size=2000, verbose=2, validation\_split = 0.33)

network\_red.summary()

network\_white.summary()

#pyplot.plot(network\_red['accuracy'], color = 'green', label = "red wine")

#pyplot.plot(network\_white['accuracy'], color = 'red', label = "white wine")

# =============================================================================

#plt.xlabel("EPOCH'S")

#plt.ylabel("Accuracy")

#plt.title("Accuracy Graph")

#plt.legend()

#pyplot.show()

# """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Show output Section

# """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" #output the scores

scores\_red = network\_red.evaluate(red\_x\_train\_std, red\_y\_train, verbose=1)

scores\_white = network\_white.evaluate(white\_x\_train\_std, white\_y\_train, verbose=1)

red\_test\_loss, red\_test\_acc = network\_red.evaluate(red\_x\_test\_std, red\_y\_test)

print('red\_test\_loss:', red\_test\_loss)

print('red\_test\_acc:', red\_test\_acc)

white\_test\_loss, white\_test\_acc = network\_white.evaluate(white\_x\_test\_std, white\_y\_test)

print('white\_test\_loss:', white\_test\_loss)

print('white\_test\_acc:', white\_test\_acc)

stop\_time = datetime.datetime.now()

print ("Time required for training:",stop\_time - start\_time)

# # fix random seed for reproducibility

seed = 7

np.random.seed(seed)

estimator\_red = network\_red.fit(red\_x\_train\_std, red\_y\_train, epochs=8000, verbose=2)

estimator\_white = network\_white.fit(white\_x\_train\_std, white\_y\_train, epochs=150, verbose=1)

print(estimator\_red)

print(estimator\_white)

# """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Show output Section

# """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" # plot metrics

# =============================================================================

pyplot.plot(estimator\_red.history['acc'], color = 'green', label = "red wine")

pyplot.plot(estimator\_white.history['acc'], color = 'blue', label = "white wine")

plt.xlabel("EPOCH'S")

plt.ylabel("Accuracy")

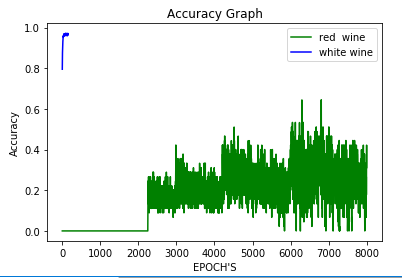
plt.title("Accuracy Graph")

plt.legend()

pyplot.show()

To help understand the Artificial Neural Network process, the flow chart is above. In the code attached, you will see this process come to life. The first step is to input the data. You continue running the code and adjusting your code to ultimately minimize the error. You then confirm that the data meets your target values to be acceptable. In the event that it does not, you will change these values. Both the training and final model are the same, as done in the examples.

The plan for arriving at the final models involved changing the input variables to ultimately arrive at a strong answer with high accuracy. This specifically involves changing the Keras Parameters. Epochs is the number of epochs to train each model. This is an iteration over the entire amount of data provided. The number the user selects is the number of times the data is trained data. Data scientists try to avoid overfitting by not over training, so they do this by training with the right number of epochs. In the code provided, they train with 8000 epochs for red and 150 epochs for white. While creating the code, the user had to change the amount of epochs to create a high amount of accuracy without concern of overfitting. This provided a high level of accuracy without fear of overfitting

Batch size is another one. The user can process the number of training samples. You can set the batch size to any number, but if it is unspecified, it defaults to 32. At the end of the batch, it updates the training set. The batch size ultimately speeds up the training. The disadvantage of this, a small batch size makes the estimate less accurate. For our example, the user set it at 1000 for red and 2000 for white based on the size of the training dataset while providing a highly accurate estimate. Since there are more 4899 white values and 1599 for red, it makes sense to have different values.

Another way you can improve your model is by adding additional layers. Adding additional layers can extract additional features of the model. Again, this can cause overfitting, so be careful not to add to many layers. The user here added 3 extra layers for both red and white. For red, the user used relu, sigmoid and softmax activation layers. For white, the user only used relu in order to ensure accuracy without risking overfitting.

Overall, the user played with a different number of inputs to provide the most accurate model to ensure there was no overfitting. This took almost four hours of tuning the model to ensure that accuracy is correct for both datasets. It took our model around 28 minutes to train.

The user was able to get to 99% accuracy for white, but struggled to improve accuracy for red, and sits at around 50% accuracy. The model took 28 minutes to complete, and this is mostly due to the amount of training for the red model.

Tuning these models was rather difficult, especially because of the amount of epochs that the red model needed to run to improve accuracy. If there were more data points contained in the red data set, it would be easier to train the model. The user spent hours improving the model to see improved accuracy, when in reality the greatest improvements came by running more iterations. The user changed the inputs identified above such as adding layers, changing batch size, the percentage of values in the training set and changing the Keras parameters.

The user validated the model by using visual graphs to confirm the accuracy.

Conclusion

Overall, artificial neural networks help individuals determine the quality of their processes and evaluations. In the example, wine quality was in question as the code classified the wine as best that it could. The disadvantage to this is that adjusting models and tuning them takes time to be able to correctly predict and classify. In the example here, wine quality is not a dangerous evaluation. As the world continues to see these forms of artificial intelligence to prove identity, the model has to be 100% perfect, or else there could be concern. There would also be implications of racial profiling in this example. There is no way to ensure that a model will be completely accurate, and this should be a consideration when making decisions using artificial neural networks.

Recommendations

Be cautious when using neural networks in predictions. No artificial intelligence platform will ever be completely accurate, so please consider before making decisions.

Appendix

*The flow chart was created using excel and then copied and pasted into this document.*