Part-A

June 20, 2020

1 Peer-graded Assignment: Build a Regression Model in Keras

1.0.1 Date: 20-June-2020

1.1 Download and Clean Dataset

[1]:
```
import pandas as pd
import numpy as np
```

We will be playing around with the same dataset that we used in the videos.

The dataset is about the compressive strength of different samples of concrete based on the volumes of the different ingredients that were used to make them. Ingredients include:

1.1.1 1. Cemen

1.1.2 2. Blast Furnace Slag

1.1.3 3. Fly Ash

1.1.4 4. Water

1.1.5 5. Superplasticizer

1.1.6 6. Coarse Aggregate

1.1.7 7. Fine Aggregate

Let's download the data and read it into a pandas dataframe.

[4]:
```
concrete_data = pd.read_csv('https://s3-api.us-geo.objectstorage.softlayer.net/
↪cf-courses-data/CognitiveClass/DL0101EN/labs/data/concrete_data.csv')
concrete_data.head()
```

[4]:
```
   Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse Aggregate  Fine Aggregate  Age  Strength
```

```
0           1040.0         676.0   28     79.99
1           1055.0         676.0   28     61.89
2            932.0         594.0  270     40.27
3            932.0         594.0  365     41.05
4            978.4         825.5  360     44.30
```

Let's check how many data points we have.

[5]:
```
concrete_data.shape
```

[5]:
```
(1030, 9)
```

Let's check the dataset for any missing values.

[6]:
```
concrete_data.describe()
```

[6]:
```
            Cement  Blast Furnace Slag      Fly Ash        Water  \
count  1030.000000         1030.000000  1030.000000  1030.000000
mean    281.167864           73.895825    54.188350   181.567282
std     104.506364           86.279342    63.997004    21.354219
min     102.000000            0.000000     0.000000   121.800000
25%     192.375000            0.000000     0.000000   164.900000
50%     272.900000           22.000000     0.000000   185.000000
75%     350.000000          142.950000   118.300000   192.000000
max     540.000000          359.400000   200.100000   247.000000

       Superplasticizer  Coarse Aggregate  Fine Aggregate          Age  \
count       1030.000000       1030.000000     1030.000000  1030.000000
mean           6.204660        972.918932      773.580485    45.662136
std            5.973841         77.753954       80.175980    63.169912
min            0.000000        801.000000      594.000000     1.000000
25%            0.000000        932.000000      730.950000     7.000000
50%            6.400000        968.000000      779.500000    28.000000
75%           10.200000       1029.400000      824.000000    56.000000
max           32.200000       1145.000000      992.600000   365.000000

          Strength
count  1030.000000
mean     35.817961
std      16.705742
min       2.330000
25%      23.710000
50%      34.445000
75%      46.135000
max      82.600000
```

[7]:
```
concrete_data.isnull().sum()
```

[7]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

The data looks very clean and is ready to be used to build our model.

Split data into predictors and target

[8]:
```
concrete_data_columns = concrete_data.columns

predictors = concrete_data[concrete_data_columns[concrete_data_columns !=␣
↪'Strength']] # all columns except Strength
target = concrete_data['Strength'] # Strength column
```

Let's do a quick sanity check of the predictors and the target dataframes.

[27]:
```
predictors.head()
```

[27]:
```
   Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse Aggregate  Fine Aggregate  Age
0            1040.0           676.0   28
1            1055.0           676.0   28
2             932.0           594.0  270
3             932.0           594.0  365
4             978.4           825.5  360
```

[28]:
```
target.head()
```

[28]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

[29]:
```python
predictors_norm = (predictors - predictors.mean()) / predictors.std()
predictors_norm.head()
```

[29]:
```
     Cement  Blast Furnace Slag   Fly Ash     Water  Superplasticizer  \
0  2.476712           -0.856472 -0.846733 -0.916319         -0.620147
1  2.476712           -0.856472 -0.846733 -0.916319         -0.620147
2  0.491187            0.795140 -0.846733  2.174405         -1.038638
3  0.491187            0.795140 -0.846733  2.174405         -1.038638
4 -0.790075            0.678079 -0.846733  0.488555         -1.038638

   Coarse Aggregate  Fine Aggregate       Age
0          0.862735       -1.217079 -0.279597
1          1.055651       -1.217079 -0.279597
2         -0.526262       -2.239829  3.551340
3         -0.526262       -2.239829  5.055221
4          0.070492        0.647569  4.976069
```

Let's save the number of predictors to n_cols since we will need this number when building our network.

[30]:
```python
n_cols = predictors_norm.shape[1] # number of predictors
n_cols
```

[30]: 8

1.2 Import Keras

Let's go ahead and import the Keras library

[14]:
```python
import keras
```

```
Using TensorFlow backend.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:521: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
```

```
packages/tensorflow/python/framework/dtypes.py:522: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:528: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Let's import the rest of the packages from the Keras library that we will need to build our regressoin model.

[31]:
```
from keras.models import Sequential
from keras.layers import Dense
```

Build a Neural Network

[32]:
```
# define regression model
def regression_model():
    # create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(1))

    # compile model
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

The above function creates a model that has one hidden layer with 10 neurons and a ReLU activation function. It uses the adam optimizer and the mean squared error as the loss function

1.3 1. Randomly split the data into a training and test sets by holding 30% of the data for testing.

[33]:
```
from sklearn.model_selection import train_test_split
```

By using the train_test_split helper function from Scikit-learn.

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(predictors, target,␣
↪test_size=0.3, random_state=42)
```

1.4 Train and Test the Network

Let's call the function now to create our model.

[35]:
```
# build the model
model = regression_model()
```

1.5 2. Train the model on the training data using 50 epochs.

[47]:
```
# fit the model
model.fit(X_train, y_train, epochs=50, verbose=2)
```

```
Epoch 1/50
 - 0s - loss: 133.6111
Epoch 2/50
 - 0s - loss: 131.2903
Epoch 3/50
 - 0s - loss: 129.9967
Epoch 4/50
 - 0s - loss: 128.3137
Epoch 5/50
 - 0s - loss: 126.2649
Epoch 6/50
 - 0s - loss: 124.3139
Epoch 7/50
 - 0s - loss: 125.2411
Epoch 8/50
 - 0s - loss: 121.5747
Epoch 9/50
 - 0s - loss: 120.2494
Epoch 10/50
 - 0s - loss: 119.2834
Epoch 11/50
 - 0s - loss: 117.4384
Epoch 12/50
 - 0s - loss: 116.8143
Epoch 13/50
 - 0s - loss: 114.3147
Epoch 14/50
 - 0s - loss: 113.0517
Epoch 15/50
 - 0s - loss: 112.6148
Epoch 16/50
 - 0s - loss: 110.2314
Epoch 17/50
 - 0s - loss: 110.4533
Epoch 18/50
 - 0s - loss: 108.7342
```

```
Epoch 19/50
 - 0s - loss: 108.7863
Epoch 20/50
 - 0s - loss: 105.2125
Epoch 21/50
 - 0s - loss: 104.3458
Epoch 22/50
 - 0s - loss: 103.4102
Epoch 23/50
 - 0s - loss: 101.2493
Epoch 24/50
 - 0s - loss: 101.2641
Epoch 25/50
 - 0s - loss: 99.6868
Epoch 26/50
 - 0s - loss: 98.1105
Epoch 27/50
 - 0s - loss: 97.9111
Epoch 28/50
 - 0s - loss: 96.9896
Epoch 29/50
 - 0s - loss: 95.1390
Epoch 30/50
 - 0s - loss: 93.9108
Epoch 31/50
 - 0s - loss: 92.8296
Epoch 32/50
 - 0s - loss: 91.8927
Epoch 33/50
 - 0s - loss: 90.8733
Epoch 34/50
 - 0s - loss: 90.0945
Epoch 35/50
 - 0s - loss: 88.5296
Epoch 36/50
 - 0s - loss: 87.5171
Epoch 37/50
 - 0s - loss: 86.9160
Epoch 38/50
 - 0s - loss: 86.7338
Epoch 39/50
 - 0s - loss: 85.9635
Epoch 40/50
 - 0s - loss: 84.2079
Epoch 41/50
 - 0s - loss: 82.6882
Epoch 42/50
 - 0s - loss: 82.9486
```

```
Epoch 43/50
 - 0s - loss: 82.0680
Epoch 44/50
 - 0s - loss: 80.8430
Epoch 45/50
 - 0s - loss: 79.8103
Epoch 46/50
 - 0s - loss: 78.8370
Epoch 47/50
 - 0s - loss: 79.4206
Epoch 48/50
 - 0s - loss: 77.9905
Epoch 49/50
 - 0s - loss: 77.8032
Epoch 50/50
 - 0s - loss: 77.8094
```

[47]: <keras.callbacks.History at 0x7f2c9da53d30>

1.6 3a. Evaluate the model on the test data.

[61]:
```
loss_val = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
loss_val
```

```
309/309 [==============================] - 0s 71us/step
```

[61]: 63.92306185231625

1.7 3b. And now we compute the mean squared error between the predicted concrete strength and the actual concrete strength.

You can use the mean_squared_error function from Scikit-learn.

[62]:
```
from sklearn.metrics import mean_squared_error
```

[63]:
```
mean_square_error = mean_squared_error(y_test, y_pred)
mean = np.mean(mean_square_error)

standard_deviation = np.std(mean_square_error)

print (mean, standard_deviation)
```

```
63.92306280280422 0.0
```

1.8 4. Repeat steps 1 - 3, 50 times, i.e., create a list of 50 mean squared errors.

[66]:
```
# To Repeat 50 Times
total_mean_squared_error = 50

mean_squared_errors = []

for i in range(0, total_mean_squared_error):
    X_train, X_test, y_train, y_test = train_test_split(predictors,
                                                        target,
                                                        test_size=0.3,
                                                        random_state=i)
    model.fit(X_train, y_train, epochs = 50, verbose = 0)
    MSE = model.evaluate(X_test, y_test, verbose = 0)
    print ("MSE " + str(i + 1)+" : " + str(MSE))


    mean_square_error = mean_squared_error(y_test, y_pred)
    mean_squared_errors.append(mean_square_error)
```

```
MSE 1 : 44.663418927238986
MSE 2 : 47.87022685078741
MSE 3 : 34.688665402359945
MSE 4 : 39.38769584334784
MSE 5 : 43.51740380938385
MSE 6 : 41.13842177159578
MSE 7 : 45.797947164492314
MSE 8 : 35.244783765675564
MSE 9 : 37.12056314906642
MSE 10 : 44.94107544537887
MSE 11 : 36.43897486813246
MSE 12 : 36.2783475462287
MSE 13 : 42.39349413381039
MSE 14 : 45.263144928274805
MSE 15 : 38.88823151819914
MSE 16 : 35.637282343744076
MSE 17 : 38.77879111280719
MSE 18 : 41.202748554809965
MSE 19 : 40.07832917889345
MSE 20 : 39.310139764088255
MSE 21 : 41.72194360529335
MSE 22 : 38.30468684569917
MSE 23 : 40.333286273055094
MSE 24 : 38.25562323412849
MSE 25 : 41.929700079859266
MSE 26 : 42.86127701546382
MSE 27 : 37.52999542137566
```

```
MSE 28 : 44.77288143071542
MSE 29 : 46.095086637824096
MSE 30 : 40.837308661451615
MSE 31 : 40.69298186811429
MSE 32 : 33.711160832624216
MSE 33 : 35.29083591448836
MSE 34 : 44.17205175998527
MSE 35 : 39.218183301413326
MSE 36 : 40.39746430776652
MSE 37 : 39.447557430822876
MSE 38 : 39.89283404303986
MSE 39 : 37.8940974115168
MSE 40 : 35.26393770014198
MSE 41 : 42.493357075070875
MSE 42 : 36.6702301371059
MSE 43 : 41.08448649532973
MSE 44 : 47.43366435276266
MSE 45 : 42.64236313162498
MSE 46 : 51.1311939572825
MSE 47 : 39.31581337938031
MSE 48 : 42.42900095325458
MSE 49 : 44.82759920756022
MSE 50 : 42.67289703023472
```

[67]:

```
mean_quared_errors = np.array(mean_squared_errors)

mean = np.mean(mean_squared_errors)

standard_deviation = np.std(mean_squared_errors)


print('\n')
print("Below is the mean and standard deviation of "␣
 ↪+str(total_mean_squared_error) + " mean squared errors without normalized␣
 ↪data. Total number of epochs used for each training is: 50" + "\n")
print("Mean: "+str(mean))
print("Standard Deviation: "+str(standard_deviation))
```

```
Below is the mean and standard deviation of 50 mean squared errors without
normalized data. Total number of epochs used for each training is: 50

Mean: 501.22813341779806
Standard Deviation: 68.83105067655065
```

```
[ ]:
```