regression model using the deep learning Keras library

April 28, 2020

1 Regression model using the deep learning Keras library

Objective: I am going to build a regression model using the Keras library to model the data about concrete compressive strength. -The predictors in the data of concrete strength include:

1. Cement
2. Blast Furnace Slag
3. Fly Ash
4. Water
5. Superplasticizer
6. Coarse Aggregate
7. Fine Aggregate

2 Download and Clean Dataset

Let's start by importing the pandas and the Numpy libraries.

[4]:
```
import pandas as pd
import numpy as np
```

[5]:
```
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()
```

[5]:
```
   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
```

[6]:
```
concrete_data.shape #data points
```

[6]: (1030, 9)

[7]:
```
concrete_data.describe()
```

[7]:
```
            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
```

[8]:
```
concrete_data.isnull().sum() # CLEAN THE DATA
```

[8]:
```
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
```

2.0.1 Split data into predictors and target

[9]:
```
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
```

[10]:
```
predictors.head()
```

[10]:
```
   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
```

[11]:
```
target.head()
```

[11]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

normalize the data by substracting the mean and dividing by the standard deviation.

[12]:
```
predictors_norm = (predictors - predictors.mean()) / predictors.std()
predictors_norm.head()
```

[12]:
```
     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
```

2.0.2 save the number of predictors to n_cols

[13]:
```
n_cols = predictors_norm.shape[1] # number of predictors
```

3 Import the Keras library

[14]:
```
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)])
```

[15]:
```
from keras.models import Sequential
from keras.layers import Dense
```

4 Build a Neural Network

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.

[16]:
```
# define regression model
def regression_model():
    # create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))

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

4.0.1 import scikit-learn in order to randomly split the data into a training and test sets

[17]:
```
from sklearn.model_selection import train_test_split
```

4.0.2 Splitting the data into a training and test sets by holding 30% of the data for testing

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(predictors, target,␣
↪test_size=0.3, random_state=42)
```

5 Train and Test the Network

[19]:
```
# build the model
model = regression_model()
```

5.0.1 Next, we will train and test the model at the same time using the fit method. We will leave out 30% of the data for validation and we will train the model for 50 epochs.

[20]:
```
# fit the model
model.fit(predictors_norm, target, validation_split=0.3, epochs=50, verbose=1)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
721/721 [==============================] - 1s 1ms/step - loss: 1726.3008 -
```

```
val_loss: 1236.9262
Epoch 2/50
721/721 [==============================] - 0s 387us/step - loss: 1693.4964 -
val_loss: 1214.2847
Epoch 3/50
721/721 [==============================] - 0s 439us/step - loss: 1662.3141 -
val_loss: 1191.7048
Epoch 4/50
721/721 [==============================] - 0s 436us/step - loss: 1629.5006 -
val_loss: 1167.0351
Epoch 5/50
721/721 [==============================] - 0s 389us/step - loss: 1589.3327 -
val_loss: 1138.3053
Epoch 6/50
721/721 [==============================] - 0s 411us/step - loss: 1539.1931 -
val_loss: 1102.9854
Epoch 7/50
721/721 [==============================] - 0s 365us/step - loss: 1476.7772 -
val_loss: 1059.5614
Epoch 8/50
721/721 [==============================] - 0s 411us/step - loss: 1397.8101 -
val_loss: 1005.5447
Epoch 9/50
721/721 [==============================] - 0s 442us/step - loss: 1301.9842 -
val_loss: 942.7596
Epoch 10/50
721/721 [==============================] - 0s 384us/step - loss: 1190.5798 -
val_loss: 873.6619
Epoch 11/50
721/721 [==============================] - 0s 393us/step - loss: 1070.5728 -
val_loss: 796.3750
Epoch 12/50
721/721 [==============================] - 0s 341us/step - loss: 937.8116 -
val_loss: 718.0617
Epoch 13/50
721/721 [==============================] - 0s 382us/step - loss: 806.5951 -
val_loss: 637.1517
Epoch 14/50
721/721 [==============================] - 0s 442us/step - loss: 680.2037 -
val_loss: 559.3722
Epoch 15/50
721/721 [==============================] - 0s 395us/step - loss: 567.0572 -
val_loss: 487.0216
Epoch 16/50
721/721 [==============================] - 0s 433us/step - loss: 472.4213 -
val_loss: 424.3209
Epoch 17/50
721/721 [==============================] - 0s 446us/step - loss: 399.7081 -
```

```
val_loss: 373.0173
Epoch 18/50
721/721 [==============================] - 0s 391us/step - loss: 347.5732 -
val_loss: 329.4477
Epoch 19/50
721/721 [==============================] - 0s 397us/step - loss: 311.3485 -
val_loss: 295.9437
Epoch 20/50
721/721 [==============================] - 0s 385us/step - loss: 287.4794 -
val_loss: 270.8085
Epoch 21/50
721/721 [==============================] - 0s 380us/step - loss: 271.5739 -
val_loss: 251.2509
Epoch 22/50
721/721 [==============================] - 0s 387us/step - loss: 260.9291 -
val_loss: 236.1270
Epoch 23/50
721/721 [==============================] - 0s 365us/step - loss: 251.8219 -
val_loss: 225.3402
Epoch 24/50
721/721 [==============================] - 0s 421us/step - loss: 244.8533 -
val_loss: 216.1450
Epoch 25/50
721/721 [==============================] - 0s 446us/step - loss: 238.7277 -
val_loss: 208.2911
Epoch 26/50
721/721 [==============================] - 0s 444us/step - loss: 233.4298 -
val_loss: 201.6223
Epoch 27/50
721/721 [==============================] - 0s 367us/step - loss: 228.8940 -
val_loss: 195.9712
Epoch 28/50
721/721 [==============================] - 0s 415us/step - loss: 224.8551 -
val_loss: 191.7873
Epoch 29/50
721/721 [==============================] - 0s 388us/step - loss: 220.9874 -
val_loss: 187.6183
Epoch 30/50
721/721 [==============================] - 0s 412us/step - loss: 217.3150 -
val_loss: 184.3843
Epoch 31/50
721/721 [==============================] - 0s 466us/step - loss: 213.8209 -
val_loss: 181.7941
Epoch 32/50
721/721 [==============================] - 0s 412us/step - loss: 210.7043 -
val_loss: 178.8490
Epoch 33/50
721/721 [==============================] - 0s 441us/step - loss: 207.5997 -
```

```
val_loss: 174.9418
Epoch 34/50
721/721 [==============================] - 0s 368us/step - loss: 204.7411 -
val_loss: 172.2746
Epoch 35/50
721/721 [==============================] - 0s 445us/step - loss: 202.3195 -
val_loss: 169.3990
Epoch 36/50
721/721 [==============================] - 0s 417us/step - loss: 199.6437 -
val_loss: 167.7714
Epoch 37/50
721/721 [==============================] - 0s 437us/step - loss: 197.7970 -
val_loss: 166.5286
Epoch 38/50
721/721 [==============================] - 0s 395us/step - loss: 195.4487 -
val_loss: 163.8821
Epoch 39/50
721/721 [==============================] - 0s 412us/step - loss: 193.2727 -
val_loss: 160.8351
Epoch 40/50
721/721 [==============================] - 0s 363us/step - loss: 191.1801 -
val_loss: 160.4910
Epoch 41/50
721/721 [==============================] - 0s 470us/step - loss: 188.9282 -
val_loss: 157.8214
Epoch 42/50
721/721 [==============================] - 0s 415us/step - loss: 186.9619 -
val_loss: 156.2641
Epoch 43/50
721/721 [==============================] - 0s 393us/step - loss: 185.1438 -
val_loss: 155.3652
Epoch 44/50
721/721 [==============================] - 0s 439us/step - loss: 183.2322 -
val_loss: 154.3583
Epoch 45/50
721/721 [==============================] - 0s 446us/step - loss: 181.7427 -
val_loss: 153.6083
Epoch 46/50
721/721 [==============================] - 0s 393us/step - loss: 180.0934 -
val_loss: 152.6627
Epoch 47/50
721/721 [==============================] - 0s 411us/step - loss: 178.4411 -
val_loss: 151.0056
Epoch 48/50
721/721 [==============================] - 0s 475us/step - loss: 177.0964 -
val_loss: 150.1668
Epoch 49/50
721/721 [==============================] - 0s 390us/step - loss: 175.5285 -
```

```
val_loss: 149.6706
Epoch 50/50
721/721 [==============================] - 0s 493us/step - loss: 173.9959 -
val_loss: 149.3502
```

[20]:
```
<keras.callbacks.History at 0x7f13ad551e80>
```

6 Evaluate the model on the test data

[21]:
```
loss_val = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
loss_val
```

```
309/309 [==============================] - 0s 108us/step
```

[21]:
```
75354250.45954692
```

6.0.1 compute the mean squared error between the predicted concrete strength and the actual concrete strength.

Let's import the mean_squared_error function from Scikit-learn.

[22]:
```
from sklearn.metrics import mean_squared_error
```

[23]:
```
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)
```

```
75354248.91785064 0.0
```

7 Build a Neural Network

[24]:
```
total_mean_squared_errors = 50
epochs = 100
mean_squared_errors = []
for i in range(0, total_mean_squared_errors):
    X_train, X_test, y_train, y_test = train_test_split(predictors_norm, target, test_size=0.3, random_state=i)
    model.fit(X_train, y_train, epochs=epochs, verbose=0)
    MSE = model.evaluate(X_test, y_test, verbose=0)
    print("MSE "+str(i+1)+": "+str(MSE))
    y_pred = model.predict(X_test)
    mean_square_error = mean_squared_error(y_test, y_pred)
    mean_squared_errors.append(mean_square_error)

mean_squared_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_errors) + " mean squared errors with normalized data.
 ↪ Total number of epochs for each training is: " +str(epochs) + "\n")
print("Mean: "+str(mean))
print("Standard Deviation: "+str(standard_deviation))
```

```
MSE 1: 94.0861078898112
MSE 2: 69.73465504075331
MSE 3: 41.54775376921719
MSE 4: 42.71618091867194
MSE 5: 37.36165171996675
MSE 6: 35.69694925203292
MSE 7: 38.464529660913165
MSE 8: 29.792352201097607
MSE 9: 31.08339798566207
MSE 10: 32.261866942964325
MSE 11: 28.205467051286913
MSE 12: 24.49172276740707
MSE 13: 32.455063603842525
MSE 14: 32.137673263796714
MSE 15: 30.564119271090117
MSE 16: 23.42215183097568
MSE 17: 28.91355815751653
MSE 18: 27.627063269754057
MSE 19: 23.75466105390135
MSE 20: 26.9092894742404
MSE 21: 25.579971757907312
MSE 22: 27.248190574275636
MSE 23: 19.101532939182515
MSE 24: 24.28075494735372
MSE 25: 26.657361755864905
MSE 26: 27.359636991926767
MSE 27: 22.512651622488274
MSE 28: 23.38672015119139
MSE 29: 27.738358229109384
MSE 30: 23.105451330783684
MSE 31: 22.600225936247693
MSE 32: 22.59083691692661
MSE 33: 20.17199847150389
MSE 34: 22.966079261310664
MSE 35: 22.310424730615708
MSE 36: 25.68064350757784
MSE 37: 19.228271114016042
```

```
MSE 38: 22.293625840862976
MSE 39: 25.54287717257503
MSE 40: 19.012725265280714
MSE 41: 22.582373813518043
MSE 42: 20.411192156350342
MSE 43: 23.11588991961433
MSE 44: 25.820463137333448
MSE 45: 26.075211719401832
MSE 46: 21.465590862780328
MSE 47: 21.12123419011681
MSE 48: 21.25171399656623
MSE 49: 21.18800794731066
MSE 50: 24.3175544923949


Below is the mean and standard deviation of 50 mean squared errors with
normalized data. Total number of epochs for each training is: 100

Mean: 28.558875355673162
Standard Deviation: 12.449009876691735
```

[ ]:

```

```