Build a Regression Model

by Mei Chiao Lin

Jun/3rd/2020

Part A -- *One* hidden layer, *non normalized data*, *50* epochs

In [1]:

```
import keras
```

```
Using TensorFlow backend.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, 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/tensorfl
ow/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, 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/tensorfl
ow/python/framework/dtypes.py:521: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, 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/tensorfl
ow/python/framework/dtypes.py:522: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, 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/tensorfl
ow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, 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/tensorfl
ow/python/framework/dtypes.py:528: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [2]:

```
import pandas as pd
import numpy as np
```

In [3]:

```
from sklearn.model_selection import train_test_split
```

In [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()
```

Out[4]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

In [5]:

```
concrete_data.shape
```

Out[5]:

```
(1030, 9)
```

In [6]:

```
concrete_data.describe()
```

Out[6]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Aggı
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.0
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.5
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.1
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.0
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.9
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.5
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.0
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.6

In [7]:

```
concrete_data.isnull().sum()
```

Out[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
```

Splitting the data

In [8]:

```
concrete_data_columns = concrete_data.columns
X = concrete_data[concrete_data_columns[concrete_data_columns != 'Strength']] # all columns except Strength
y = concrete_data['Strength'] # Strength column
n_cols=X.shape[1]
```

In [9]:

```
#Split the data into training dataset and testing dataset with 30% test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Building the model

In [10]:

```
from keras.models import Sequential
from keras.layers import Dense
```

In [11]:

```
#One hidden layer with 10 nodes and relu function
#adam optimizer and mean_squared_error as loss function
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
```

Training

In [12]:

```
#build the model
model = regression_model()
```

In [13]:

```
#fitting data to the model with 50 epoch
model.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
721/721 [==============================] - 1s 1ms/step - loss: 25892.42
30
Epoch 2/50
721/721 [==============================] - 0s 414us/step - loss: 3221.2
409
Epoch 3/50
721/721 [==============================] - 0s 407us/step - loss: 2033.3
941
Epoch 4/50
721/721 [==============================] - 0s 359us/step - loss: 1701.5
336
Epoch 5/50
721/721 [==============================] - 0s 416us/step - loss: 1456.9
5960s - loss: 1
Epoch 6/50
721/721 [==============================] - 0s 494us/step - loss: 1223.0
491
Epoch 7/50
721/721 [==============================] - 0s 447us/step - loss: 1017.8
724
Epoch 8/50
721/721 [==============================] - 0s 418us/step - loss: 852.52
06
Epoch 9/50
721/721 [==============================] - 0s 339us/step - loss: 731.99
66
Epoch 10/50
721/721 [==============================] - 0s 389us/step - loss: 644.29
47
Epoch 11/50
721/721 [==============================] - 0s 420us/step - loss: 571.26
89
Epoch 12/50
721/721 [==============================] - 0s 445us/step - loss: 512.60
51
Epoch 13/50
721/721 [==============================] - 0s 395us/step - loss: 460.12
32
Epoch 14/50
721/721 [==============================] - 0s 405us/step - loss: 418.62
10
Epoch 15/50
721/721 [==============================] - 0s 360us/step - loss: 382.94
67
Epoch 16/50
721/721 [==============================] - 0s 358us/step - loss: 350.50
77
Epoch 17/50
721/721 [==============================] - 0s 418us/step - loss: 325.23
41
Epoch 18/50
721/721 [==============================] - 0s 416us/step - loss: 302.82
44
Epoch 19/50
721/721 [==============================] - 0s 526us/step - loss: 284.67
79
```

```
Epoch 20/50
721/721 [==============================] - 0s 468us/step - loss: 270.24
86
Epoch 21/50
721/721 [==============================] - 0s 525us/step - loss: 256.52
57
Epoch 22/50
721/721 [==============================] - 0s 441us/step - loss: 244.63
43
Epoch 23/50
721/721 [==============================] - 0s 445us/step - loss: 234.74
50
Epoch 24/50
721/721 [==============================] - 0s 468us/step - loss: 225.39
37
Epoch 25/50
721/721 [==============================] - 0s 524us/step - loss: 217.31
46
Epoch 26/50
721/721 [==============================] - 0s 389us/step - loss: 207.79
42
Epoch 27/50
721/721 [==============================] - 0s 411us/step - loss: 199.09
50
Epoch 28/50
721/721 [==============================] - 0s 415us/step - loss: 192.45
92
Epoch 29/50
721/721 [==============================] - 0s 418us/step - loss: 186.27
34
Epoch 30/50
721/721 [==============================] - 0s 464us/step - loss: 181.27
83
Epoch 31/50
721/721 [==============================] - 0s 474us/step - loss: 172.88
94
Epoch 32/50
721/721 [==============================] - 0s 528us/step - loss: 169.18
86
Epoch 33/50
721/721 [==============================] - 0s 413us/step - loss: 162.88
49
Epoch 34/50
721/721 [==============================] - 0s 450us/step - loss: 157.90
45
Epoch 35/50
721/721 [==============================] - 0s 490us/step - loss: 156.52
91
Epoch 36/50
721/721 [==============================] - 0s 443us/step - loss: 151.00
12
Epoch 37/50
721/721 [==============================] - 0s 383us/step - loss: 146.85
84
Epoch 38/50
721/721 [==============================] - 0s 665us/step - loss: 144.02
53
```

```
Epoch 39/50
721/721 [==============================] - 0s 665us/step - loss: 142.30
53
Epoch 40/50
721/721 [==============================] - 1s 721us/step - loss: 139.40
21
Epoch 41/50
721/721 [==============================] - 1s 748us/step - loss: 135.75
19
Epoch 42/50
721/721 [==============================] - 0s 692us/step - loss: 134.43
32
Epoch 43/50
721/721 [==============================] - 0s 636us/step - loss: 131.61
08
Epoch 44/50
721/721 [==============================] - 0s 556us/step - loss: 129.09
26
Epoch 45/50
721/721 [==============================] - 0s 492us/step - loss: 127.96
97
Epoch 46/50
721/721 [==============================] - 0s 362us/step - loss: 126.72
09
Epoch 47/50
721/721 [==============================] - 0s 417us/step - loss: 127.65
54
Epoch 48/50
721/721 [==============================] - 0s 416us/step - loss: 123.89
20
Epoch 49/50
721/721 [==============================] - 0s 418us/step - loss: 125.03
49
Epoch 50/50
721/721 [==============================] - 0s 480us/step - loss: 121.52
10
```

Out[13]:

```
<keras.callbacks.History at 0x7f49256511d0>
```

In [14]:

```
#evaluate the model
evaluated_score = model.evaluate(X_test, y_test, verbose=1)
```

```
309/309 [==============================] - 0s 315us/step
```

In [15]:

```
y_predict=model.predict(X_test)
```

In [16]:

```
from sklearn.metrics import mean_squared_error
```

In [17]:

```
squared_error_score = mean_squared_error(y_test, y_predict)
```

Repeat 50 times

In [18]:

```
#Repeat 50 times
error_score=[]
for i in range(50):
    model.fit(X_train, y_train, epochs=50, verbose=0)
    y_predict=model.predict(X_test)
    error_score.append(mean_squared_error(y_test,y_predict))
```

In [19]:

```
Mean = np.mean(error_score)
Std = np.std(error_score)
print('The mean of mean_squared_error is : {:.3f}\nThe standard deviation of mean_s
quared_error is : {:.3f}'.format(Mean, Std))
```

```
The mean of mean_squared_error is : 64.019
The standard deviation of mean_squared_error is : 21.461
```

In [ ]:

```

```