Regression Models In Keras

Paul Tuyikunde

july 3,2021

Objectives

Download and Clean Dataset

let's start by importing libraries

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

from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```
```
Using TensorFlow backend.
```

Download the dataset and do some Exploratory data analysis

Data description:
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.

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

In [2]:
```
file = 'concrete_data.csv' #file
url = 'https://cocl.us/concrete_data' #url for the file.
```

In [3]:
```
concrete_df = pd.read_csv(file)
concrete_df.head()
```

Out[3]:

	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 [4]:
```
#let's check the shape of data
print(concrete_df.shape)
concrete_df.describe()
```
```
(1030, 9)
```

Out[4]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

In [5]:
```
#let's check for missing data
concrete_df.isnull().sum()
```

Out[5]:
```
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
```

we have no missing data in our dataframe

we will separate the predictors from the target

In [6]:
```
cols = concrete_df.columns
predictors = concrete_df[cols[cols != 'Strength']]
predictors.head()
```

Out[6]:

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

In [7]:
```
target = concrete_df['Strength']
target.head()
```

Out[7]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

In [8]:
```
ncols = predictors.shape[1]
```

Build a baseline Model

In [9]:
```
#Create a model
model_A = Sequential()
model_A.add(Dense(10, activation = 'relu', input_shape = (ncols,)))
model_A.add(Dense(10, activation = 'relu'))
model_A.add(Dense(1))
#compile model
model_A.compile(optimizer = 'adam', loss = 'mean_squared_error')
model_A.summary()
```
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                90
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 211
Trainable params: 211
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
def train_evaluate(model, x, y, epochs, iterations = 50):
    list_of_mse=[]
    print('===> Fitting the model...')
    for i in range(iterations):
        #train test split
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                            random_state = 42)
        #fit the model

        model.fit(X_train, y_train, epochs = epochs, verbose = 0)

        #Evaluate the model
        y_hat = model.predict(X_test)
        mse = mean_squared_error(y_test, y_hat)
        list_of_mse.append(mse)
    return list_of_mse
```

In [11]:
```
def mse_std(list_of_mse):
    mse = round(np.mean(list_of_mse),3)
    std = round(np.std(list_of_mse),3)
    return print('Mse : {}\nSTD : {}'.format(mse,std))
```

In [12]:
```
list_A = train_evaluate(model_A, predictors, target, 50, 50)
```
```
===> Fitting the model...
```

In [13]:
```
mse_std(list_A)
```
```
Mse : 53.701
STD : 18.499
```

Normalize data

Let's normalize our predictors

In [14]:
```
norm_predictors = (predictors - predictors.mean()) / predictors.std()
norm_predictors.head()
```

Out[14]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	0.862735	-1.217079	-0.279597
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079	-0.279597
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	3.551340
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	5.055221
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569	4.976069

In [15]:
```
list_B = train_evaluate(model_A, norm_predictors, target, 50, 50)
```
```
===> Fitting the model...
```

In [16]:
```
mse_std(list_B)
```
```
Mse : 34.232
STD : 14.755
```

Observation: The mean squared error and the standard deviation of the normalized data are both less than the MSE and STD of the baseline data.

Increase Epochs

lets increase the number of epochs to 100.

In [19]:
```
list_C = train_evaluate(model_A, norm_predictors, target, epochs=100, iterations=50)
```
```
===> Fitting the model...
```

In [20]:
```
mse_std(list_C)
```
```
Mse : 29.752
STD : 0.734
```

Observation: The mean squared error and the standard deviation of the **increased epochs** model are both less than the MSE and STD of the normalized data.

Increase Hidden layers

let's increase the number of hidden layers to 3 hidden layers, each of 10 nodes and ReLU activation function.

In [21]:
```
#Create a model
model_D = Sequential()
model_D.add(Dense(10, activation = 'relu', input_shape = (ncols,)))
model_D.add(Dense(10, activation = 'relu'))
model_D.add(Dense(10, activation = 'relu'))
model_D.add(Dense(10, activation = 'relu'))
model_D.add(Dense(1))
#compile model
model_D.compile(optimizer = 'adam', loss = 'mean_squared_error')
model_D.summary()
```
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 10)                90
_________________________________________________________________
dense_5 (Dense)              (None, 10)                110
_________________________________________________________________
dense_6 (Dense)              (None, 10)                110
_________________________________________________________________
dense_7 (Dense)              (None, 10)                110
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 11
=================================================================
Total params: 431
Trainable params: 431
Non-trainable params: 0
_________________________________________________________________
```

With this model i will use the *normalized data* and keep *50 epochs*.

In [22]:
```
list_D = train_evaluate(model_D, norm_predictors, target, 50, 50)
```
```
===> Fitting the model...
```

In [23]:
```
mse_std(list_D)
```
```
Mse : 37.972
STD : 11.514
```

Observation: The mean squared error and the standard deviation of the **increased epochs** model are both less than the MSE and STD of the model with 3 hidden layers with 10 nodes each.

Discussion

Comparing models with their mean squared error `mse` and standard deviation `std`. The baseline model performance:

Mse	Std
53.701	18.499

The second scenario involved **normalizing data** by subtracting the mean from the individual predictors and dividing by the standard deviation. the model seems to performe best than before with the following numbers.

Mse	Std
34.232	14.755

The third model with increased epochs performed great where mse was reduced compared to the previous model results.

Mse	Std
29.752	0.734

The last one where the number of hidden layers was increased from 1 to 3 with 10 nodes each, the model didn't out perform the previous results but it peformed better than the baseline model.

Mse	Std
37.972	11.514

To conclude, to get the best model one can keep tweaking and tuning the model and hyperparamaters such as *increasing number of epochs, data normalization and model layers.*