Practical Activity 4.1 Regression using kNN

\

1 Week 4 Hands-on Task 4.1: Regression with kNN

This notebook is an excercise for developing a k Nearest Neighbour (kNN) regression model for house price prediction. We apply the concepts discussed in - Concept 4.1: Introduction to regression analysis - Concept 4.2: Regression using kNN

We will use the following python libraries for this practical. - Pandas: https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html - scikit-learn: https://scikit-learn.org/stable/index.html

Note: this is assessment is not marked. Please check your work with the provided solution.

2 The Housing dataset

It contains information about houses in the suburbs of Boston collected by D. Harrison and D.L. Rubinfeld in 1978. The Housing dataset is available online at - https://raw.githubusercontent.com/rasbt/python-machine-learning-book-3rd-edition/master/ch10/housing.data.txt or - scikit-learn (https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/datasets/data/boston_house_prices.csv)

The dateset has 506 instances and each instance has the following features or attributes: - CRIM: Per capita crime rate by town - ZN: Proportion of residential land zoned for lots over 25,000 sq. ft. - INDUS: Proportion of non-retail business acres per town - CHAS: Charles River dummy variable (= 1 if tract bounds river and 0 otherwise) - NOX: Nitric oxide concentration (parts per 10 million) - RM: Average number of rooms per dwelling - AGE: Proportion of owner-occupied units built prior to 1940 - DIS: Weighted distances to five Boston employment centers - RAD: Index of accessibility to radial highways - TAX: Full-value property tax rate per $\$10,000$ - PTRATIO: Pupil-teacher ratio by town - B: 1000(Bk – 0.63)2, where Bk is the proportion of [people of African American descent] by town - LSTAT: Percentage of lower status of the population - MEDV: Median value of owner-occupied homes in $1000s

Goal: our goal for this practical activity is to develop a kNN regerssion model to predict the value of a house given the other attributes i.e., our target is MEDV.

3 Loading the dataset

We can use either of the sources mentioned above to load the dataset.

[1]:
```
#laoding from github source

import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
                 'python-machine-learning-book-3rd-edition'
                 '/master/ch10/housing.data.txt',
                  header=None,
                  sep='\s+')
df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
               'NOX', 'RM', 'AGE', 'DIS', 'RAD',
               'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

df.head()
```

[1]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

[2]:
```
#laoding from sklearn
from sklearn.datasets import load_boston
data = load_boston()
```

[3]:
```
data.feature_names
```

[3]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

Note: the sklearn dataset comes as ndarray. We can convert this array to pandas dataframe and continue rest of the procedure or we can use the arrays to build the model. Below is the process to convert the array to a panadas DF.

[4]:
```
# Read the DataFrame, first using the feature data
df_sklearn = pd.DataFrame(data.data, columns = data.feature_names)

# Add a target column, and fill it with the target data
df_sklearn['target'] = data.target# Show the first five rows
df_sklearn.head()
```

[4]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  target
0     15.3  396.90   4.98    24.0
1     17.8  396.90   9.14    21.6
2     17.8  392.83   4.03    34.7
3     18.7  394.63   2.94    33.4
4     18.7  396.90   5.33    36.2
```

4 Preprocessing

For simplicity we will consider the numeric features only. So that we can apply simpler distance metric. Let's check the types of the features.

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

We observer that pandas has not detected the data types correctly. From the descriptions of the features, we know that - CHAS: Charles River dummy variable (= 1 if tract bounds river and 0 otherwise) - RAD: Index of accessibility to radial highways are categorical variables. We remove them from our dataset.

[6]:
```
df.drop(['CHAS', 'RAD'], axis=1, inplace=True)
```

4.1 Create train and test set

For this excercise, we will use 70/30 split.

[9]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size = 0.3)

X_train = train.drop('MEDV', axis=1)
y_train = train['MEDV']

X_test = test.drop('MEDV', axis = 1)
y_test = test['MEDV']
```

4.2 Preprocessing – Scaling the features

We have seen that we feature scaling provides better models. So, we scale our feautres. Note: we will not scale our target.

[10]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled = scaler.fit_transform(X_train)
#reverting back to df
X_train = pd.DataFrame(x_train_scaled)

x_test_scaled = scaler.fit_transform(X_test)
X_test = pd.DataFrame(x_test_scaled)
```

5 Traininig kNN regression model

[11]:
```
#import required packages
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
%matplotlib inline
```

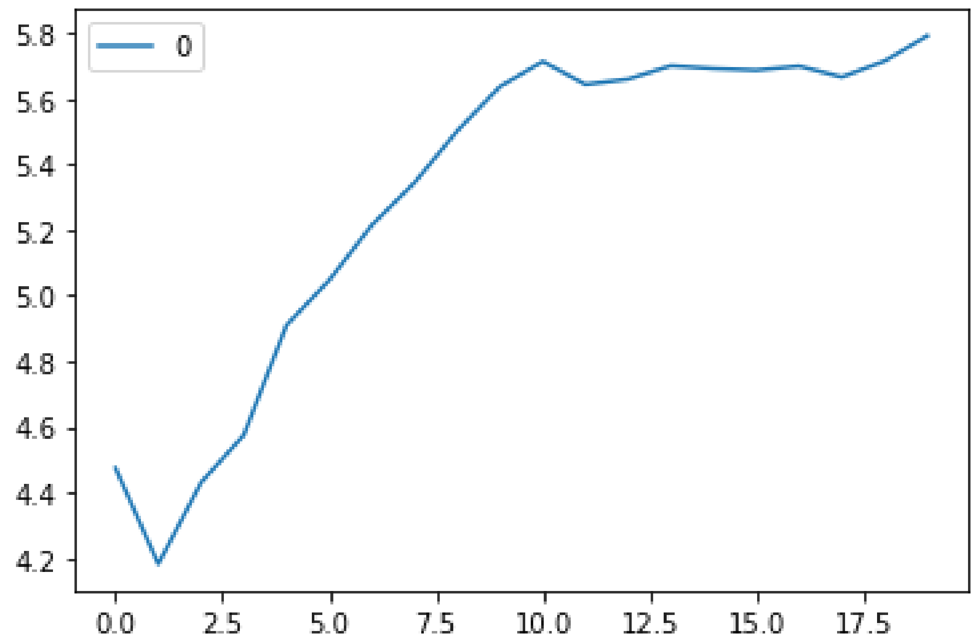
[13]:
```
rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

    model.fit(X_train, y_train)  #fit the model
```

```
    pred  = model.predict(X_test) #make prediction on test set
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
    print('RMSE value for k= ' , K , 'is:', error)
```

```
RMSE value for k=  1 is: 4.473893570952546
RMSE value for k=  2 is: 4.181632771009304
RMSE value for k=  3 is: 4.4300739898511345
RMSE value for k=  4 is: 4.5747492381665325
RMSE value for k=  5 is: 4.909446586884685
RMSE value for k=  6 is: 5.046892318647525
RMSE value for k=  7 is: 5.214636922974914
RMSE value for k=  8 is: 5.344373520871469
RMSE value for k=  9 is: 5.502161176786321
RMSE value for k=  10 is: 5.637782493797399
RMSE value for k=  11 is: 5.713593112853167
RMSE value for k=  12 is: 5.64382870477395
RMSE value for k=  13 is: 5.660183213572813
RMSE value for k=  14 is: 5.70048880684363
RMSE value for k=  15 is: 5.692019086980445
RMSE value for k=  16 is: 5.687010308227703
RMSE value for k=  17 is: 5.699462472298406
RMSE value for k=  18 is: 5.666462575874093
RMSE value for k=  19 is: 5.714617600110954
RMSE value for k=  20 is: 5.7917125313721005
```

[14]:
```
#plotting the error
#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fac72ead88>
```


0
5.8
5.6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5