Aniket Amar Thopte

Electric Load Forecasting using different techniques

K-Nearest Neighbor Regressor

Introduction:

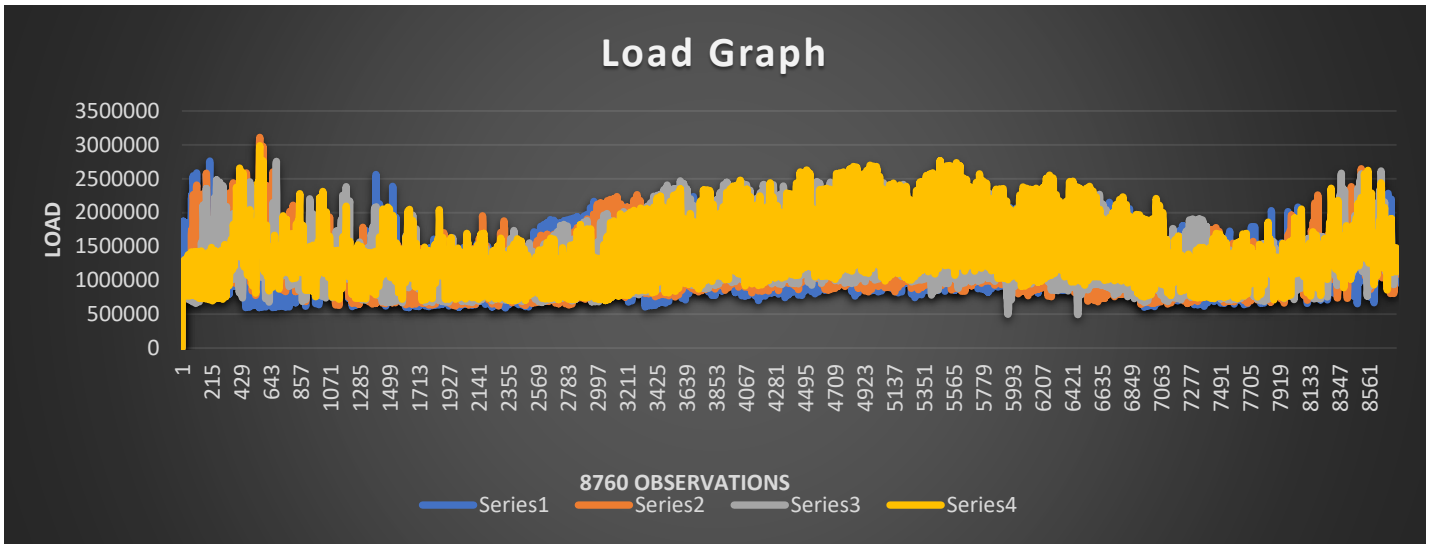
The problem statement deals with hourly load forecasting. We have been given dataset from 2002 to 2005 (i.e. 4 years data) on hourly basis. I have used **K-Nearest Neighbor Regressor** technique for forecasting the load for 2006 year on hourly basis.

Dataset:

The Dataset file is named **'history.csv**'. The dataset provided has 4 variable columns namely 'Date' in dd/mm/yy (as per python date stamp), 'Hour' ranging from 1 to 24, 'Temperature' & 'Load'. The total **number of observations given is 35064**. Another excel file containing the 2006 data is provided named '**fcst.csv**' in which load column is kept blank to be forecasted.

Data-Preprocessing:

In order to check if the data has any seasonality, trend or just in simple terms is following the same pattern, we have plotted a line graph in Excel for years 2002(Series-1) to 2005(Series-4).



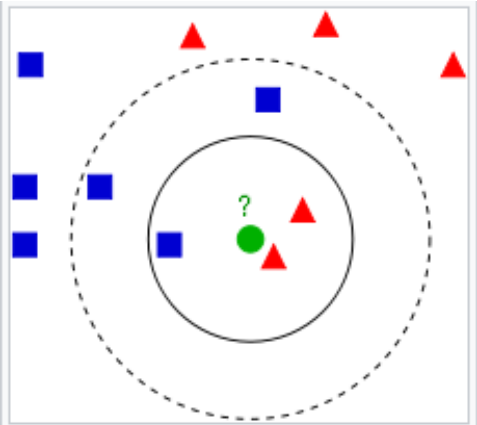
Hence, we can clearly see that the entire years load majorly (approximately) repeats itself. For our convenience we have dropped the date column. The cluster has been formed on basis of similar hour and temperature features.

Methodology:

The coding was done on Python (Jupyter Notebook). The KNN algorithm works on simple logic that the forecast point is obtained using similar historical data of the previous similar features. In our case we have fed the entire data from excel to the KNN Regressor. That is for forecasting the load at given hour at given temperature the KNN algorithm will use similar hour and temperature feature data. Other similar data will not be used and will not affect the forecast in any way. Some brief information on the ***KNN algorithm*** used.

- KNN algorithm can be used for both classification and regression problems. The KNN algorithm uses '**feature similarity**' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

- k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until function evaluation.
- The best choice of k depends upon the data; generally, larger values of k reduces effect of the noise on the classification but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques (see hyperparameter optimization). The special case where the class is predicted to be the class of the closest training sample (i.e. when k = 1) is called the nearest neighbor algorithm.
- The accuracy of the k-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance.



Example of *k*-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If *k* = *3* (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If *k* = *5* (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

In our case the target variable is Load and the feature variables are Temperature & Hour. The logic behind the code runs like this; the forecast for the query point is the average of the loads corresponding to the similar hour and temperature loads.

Code and Theory:

The following libraries have been used:-

NumPy:
It is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Pandas:
In computer programming, pandas are a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Matplotlib:
Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits.

Seaborn:
Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Scikit-learn:
Scikit-learn (also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The below mentioned code is for the month of January and it has been run 11 more times for rest months to get the forecast.

1- Read the file

In [7]:
```
import numpy as np
import pandas as pd

df=pd.read_csv("history.csv")
```

In [8]:
```
df.head(5)
```

Out[8]:

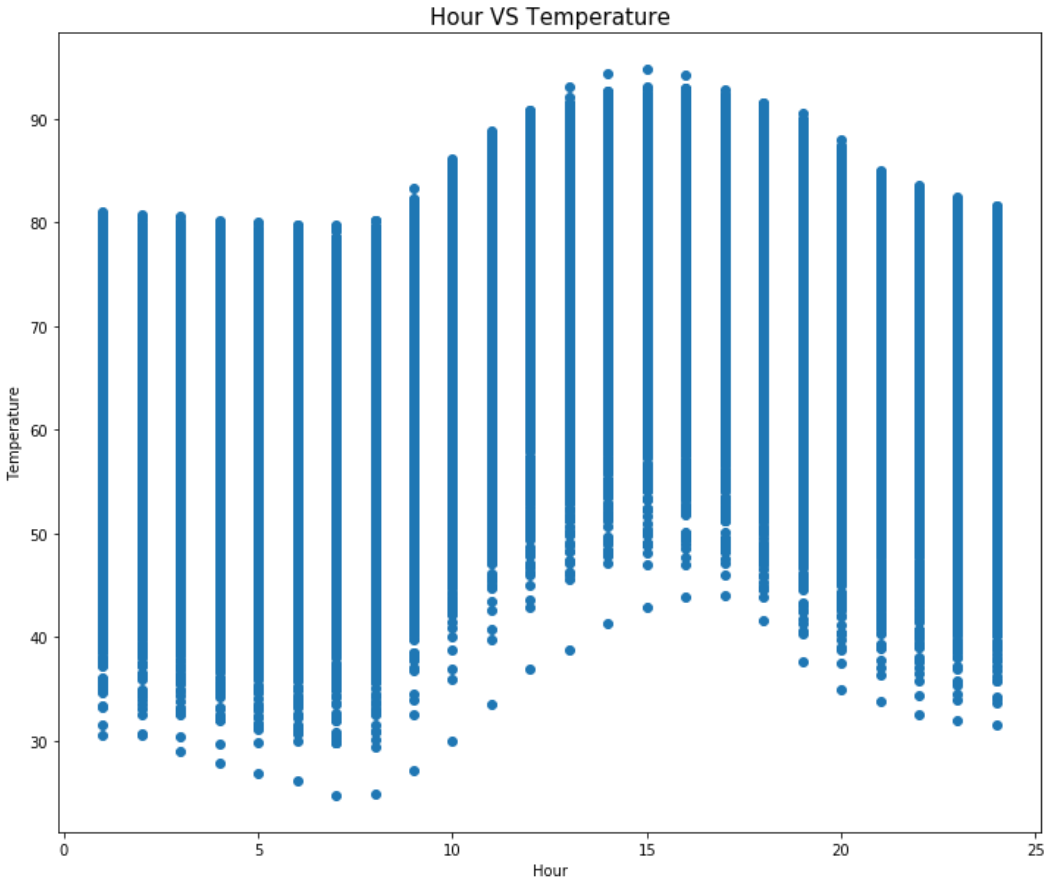
	Hour	Temperature	Load
0	1	43.72	1384494
1	2	42.72	1392822
2	3	41.84	1407887
3	4	41.04	1438658
4	5	40.56	1484046

In [3]:
```
df.tail(5)
```

Out[3]:

	Hour	Temperature	Load
35059	20	64.12	1409813.0
35060	21	62.88	1309493.0
35061	22	61.80	1217706.0
35062	23	61.16	1127940.0
35063	24	60.80	1047116.0

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sb
plt.figure(figsize=(12,10))
plt.scatter(df['Hour'],df['Temperature'])
plt.title('Hour VS Temperature',fontsize=15)
plt.ylabel('Temperature')
plt.xlabel('Hour')
plt.show()
```



2 - Create train and test set

In [12]:
```
X=df.iloc[:,0:2].values
Y=df.iloc[:,2].values
```

In [13]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=1/3,random_state=0)
```

We have divided the data in 70:30 % for training & testing purpose.

Parameter Selection:

The best choice of k depends upon the data; generally, larger values of k reduces effect of the noise on the classification but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques (see hyperparameter optimization). The special case where the class is predicted to be the class of the closest training sample (i.e. when k = 1) is called the nearest neighbor algorithm.

3 - Error rate for different k values

In [14]:
```
# 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
```

In [8]:
```
rmse_val = [] # to store rmse values for different k
for K in range(100):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

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

```
RMSE value for k=  1 is: 176930.55780715542
RMSE value for k=  2 is: 154649.52453350995
RMSE value for k=  3 is: 145452.31486904953
RMSE value for k=  4 is: 140738.43918351285
RMSE value for k=  5 is: 137761.08952254642
RMSE value for k=  6 is: 135807.78784339133
RMSE value for k=  7 is: 134538.5126487041
RMSE value for k=  8 is: 133490.6458304449
RMSE value for k=  9 is: 132387.30476327537
RMSE value for k=  10 is: 131748.23270456272
RMSE value for k=  11 is: 131476.919552738
RMSE value for k=  12 is: 130848.63630795322
RMSE value for k=  13 is: 130315.91000882782
RMSE value for k=  14 is: 129966.41437764531
RMSE value for k=  15 is: 129517.80237415918
RMSE value for k=  16 is: 129234.90971591802
RMSE value for k=  17 is: 129004.49398016457
RMSE value for k=  18 is: 128884.85599026046
RMSE value for k=  19 is: 128605.54104102725
RMSE value for k=  20 is: 128456.78109217172
.
.
.
```

.
.

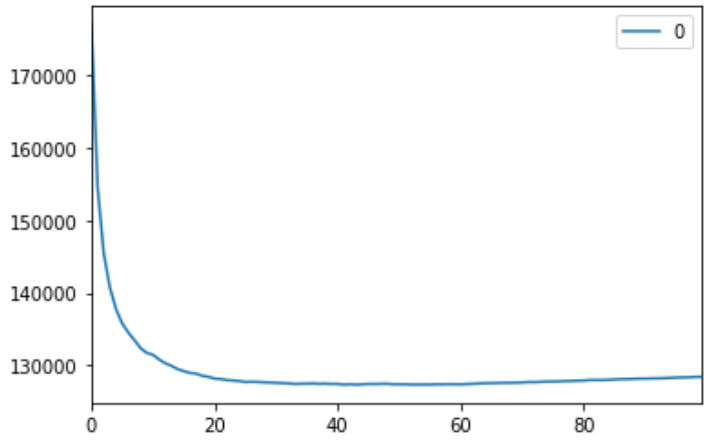
```
RMSE value for k=  81 is: 127956.92547202729
RMSE value for k=  82 is: 128018.07123973234
RMSE value for k=  83 is: 128019.16816008539
RMSE value for k=  84 is: 128021.02343936227
RMSE value for k=  85 is: 128044.51197966527
RMSE value for k=  86 is: 128097.79800724279
RMSE value for k=  87 is: 128136.74268271243
RMSE value for k=  88 is: 128149.05930329795
RMSE value for k=  89 is: 128177.90939823964
RMSE value for k=  90 is: 128211.44721302016
RMSE value for k=  91 is: 128209.5907427358
RMSE value for k=  92 is: 128246.41480487917
RMSE value for k=  93 is: 128248.32830140869
RMSE value for k=  94 is: 128275.24699386864
RMSE value for k=  95 is: 128312.2963884713
RMSE value for k=  96 is: 128351.05395809242
RMSE value for k=  97 is: 128381.10838759641
RMSE value for k=  98 is: 128392.20938908235
RMSE value for k=  99 is: 128446.01344268025
RMSE value for k=  100 is: 128458.04969091223
```

As we can see we have run the code to find the optimum value for K in the range 1-100. We check for the RMSE value for different K values and compare them. We can see that the RMSE value is 127380.64 for K= 44.

We have also plotted a graph for K Value vs RMSE value.
For a very low value of k (suppose k=1), the model overfits on the training data, which leads to a high error rate on the validation set. On the other hand, for a high value of k, the model performs poorly on both train and validation set. If you observe closely, the validation error curve reaches a minima at a value of k = 44. This value of k is the optimum value of the model (it will vary for different datasets). This curve is known as an '**elbow curve**' (because it has a shape like an elbow) and is usually used to determine the k value.

In [9]:
```
#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17682477948>
```



In [10]:
```
np.where(curve[0]==min(curve[0])) # Find the lowest value for K from the plot above
```

Out[10]:
```
(array([43], dtype=int64),)
```

4 - Predictions on the test dataset

In [11]:
```
from sklearn.neighbors import KNeighborsRegressor
KNR = KNeighborsRegressor(44)
KNR.fit(xtrain, ytrain)
```

Out[11]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=44, p=2,
                    weights='uniform')
```

In [12]:
```
predicted=KNR.predict(xtest)
```

In [13]:
```
predicted
```

In [24]:
```
ytest.shape
```

Out[24]:
```
(992,)
```

In [25]:
```
X.shape
```

Out[25]:
```
(2976, 2)
```

In [26]:
```
len(predicted)
```

Out[26]:
```
992
```

In [27]:
```
ytest
```

5- Forecasting

```
In [33]: df1=pd.read_csv("fcst.csv")

In [34]: dfnew=df1.iloc[:,0:2]

In [35]: newpred=KNR.predict(dfnew)

In [36]: newpred=newpred.astype(int)

In [37]: np.savetxt('array.csv', newpred, delimiter=',', fmt='%d')
```

The forecasted file gets saved in array.csv file for temporary purpose.

Conclusion:

Hence, we have used the simple K-nearest neighbor model for forecasting year ahead hourly based load. Our model is working.

*The Mean Average Percentage Error is **10.56%.***