

K Nearest Neighbors

video

github

can be applicable for nd data

intuition

Working

- for a point P, get all the distance from p and rest of the point
- get the k closest points from p
- in the end do prediction on the basis of majority count (because we are using majority count -> we avoid even k values)
- For example -> if majority of k points are 1 -> then the output will be 1

How to select k= the number of neighbors

approch 1

- \sqrt{n}
- n= number of rows
- Not very much suggested

approch 2

- make a multiple knn model with different values of k(1,2,..25,26,etc)
- use cross validation and choose best accurate model
- Better approach

Note

- It is adviced to standerize your data if you are using knn
- It is lazy learning technique

- Because we just store points while training and most of work done while prediction
- prediction become slow

over fitting and under fitting in knn

- for very high value of k lead to under fitting
- for very low value of k lead to over fitting

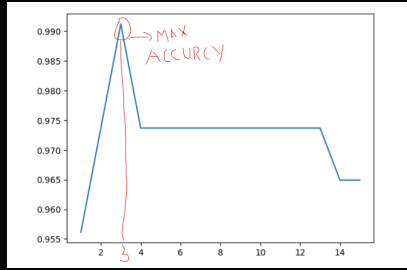
Limitation

- Not good for large data because it is a lazy learning technique
- Not good for high dimentional data because of curse of dimensionality
 - for very high dimensional data(like 500 features) , distance concept becomes quite un reliable
- Not work good for data with outliers
- Not work good for Imbalanced data set
 - in output column
 - 900 yes points
 - 100 no points
- Not work good for data set that have features with very large scale difference
 - that's why standardization is suggested

code

choosing the best model

```
for i in range(1,16):  
    knn = KNeighborsClassifier(n_neighbors=i)  
    knn.fit(X_train,y_train)  
    y_pred = knn.predict(X_test)  
    scores.append(accuracy_score(y_test, y_pred))
```



```
import matplotlib.pyplot as plt  
plt.plot(range(1,16),scores)
```

single model

```
from sklearn.neighbors import  
KNeighborsClassifier  
  
knn = KNeighborsClassifier(n_neighbors=3)  
knn.fit(X_train,y_train)  
y_pred = knn.predict(X_test)
```