
KN
N

KNN – k Nearest Neighbours

KNN is a simple non-parametric supervised machine learning algorithm used for classification and regression tasks. It operates by storing all labeled training examples and classifying new data points based on the majority class among their k closest neighbours, where "k" is a user-defined parameter representing the number of neighbours considered.

It is always advised to apply standard scaling on the data.

Code:

```
Scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

knn = KNeighboursClassifier(n_neighbours = 4)
knn.fit(data_scaled, classes)
```

There are two ways to find the best value of k.

First is by counting data points manually which is not recommended.

Second is experimental, where you run a loop with different k values to find the best one.

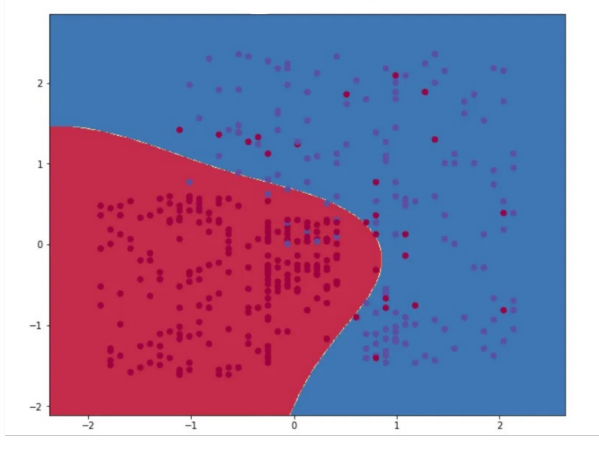
```
scores = []

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))
```

→ Cross Validation Method

KNN Regressor is almost similar. The only difference is that it's for regression meaning for numerical values. So instead of extracting the final predicted class, we simply extract the output numerical value from training data.

Descision Surface / Boundary

A descision Boundary is a visualization tool used in classification tasks to show how a machine learning model divides the input feature space into regions associated with different class labels. It helps to understand how the model "sees" the prediction problem by illustrating boundaries where the model switches from predicting one class to another.



Overfitting and Underfitting.

If the value of K is very small → Overfitting

If the value of K is very big → Underfitting

KNN is a predictive model and not an inference model. Therefore, its a blackbox model.

Because KNN only gives the output but does not tell the strength of the input cols that might influence output

Limitations of KNN

1. **Computationally Expensive and Slow with Large Datasets**
   kNN must compute the distance between the query point and every training sample for each prediction, making it very slow and resource-intensive as the dataset grows.

2. **High Memory Usage**
   The algorithm stores the entire training dataset in memory, which becomes impractical for large datasets.

3. **Sensitive to Feature Scale**
   Features with larger scales can dominate distance calculations, leading to biased or inaccurate results unless the data is properly scaled.

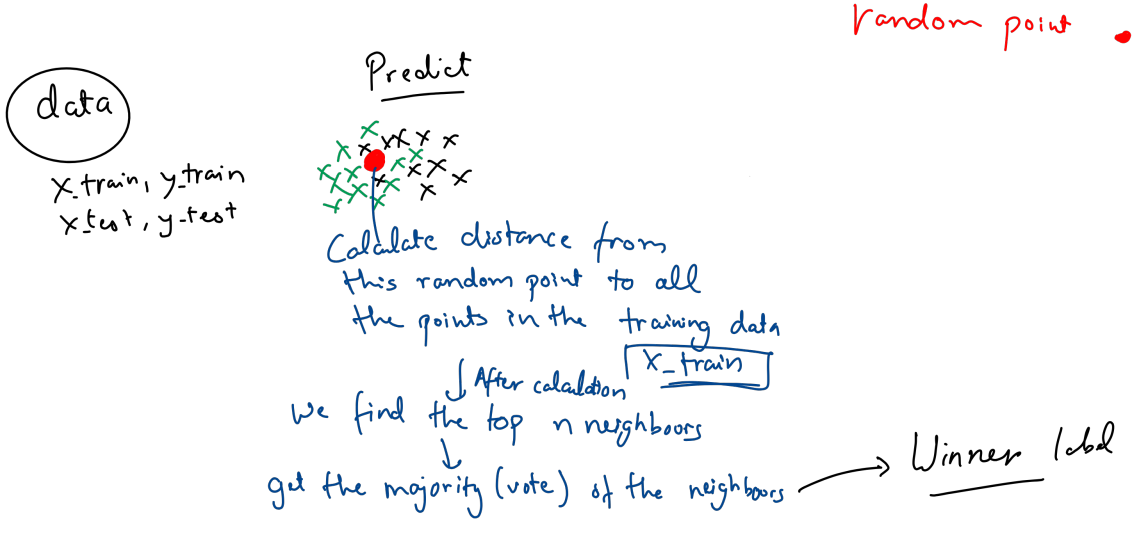
4. **Curse of Dimensionality**
   As the number of features increases, the distance between points becomes less meaningful, degrading kNN's performance in high-dimensional spaces.

5. **No Model Generalization**
   kNN does not learn a general model or summary of the data; it simply memorizes the training set and provides no insight into feature importance or relationships.

6. **Poor Performance with Imbalanced Data**
   If one class is much more frequent, kNN may misclassify minority class samples because the majority class dominates the neighborhood.

7. **Sensitive to Noise and Outliers**
   Outliers and noisy data can significantly affect predictions, especially with small k values.

8. **Choice of k is Non-Trivial**
   Finding the optimal number of neighbors (k) is challenging; too small a k can lead to overfitting, while too large a k can cause underfitting



Hyper parameters.

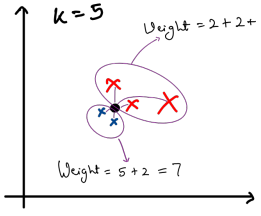
1) Weights : Uniform and distance

Normal KNN

where weights of neighbours is considered as 1 (uniform).

Weighted KNN : Weighted KNN is simply adding weights across the n nearest neighbours.

$$W = \frac{1}{\text{distance}}$$



Here each neighbour is given a weight based on how near the point is to the target point.

The weight of Red points is 5 and blue is 7 Therefore, even though the occurence of blue is less, due to more weights, we will tag the new target point as blue.

In Normal (Uniform) KNN, the target value is decided by the majority occurence points of any class, while in Weighted KNN, the final target value is decided based on weights.

It's recommended to try both parameters and chose one depending on the accuracy score.

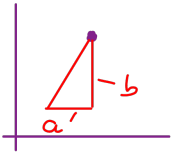
Weighted KNN is good in dealing with Outliers.

Types of Distances (hyperparameter = "metric")

Default distance used is Euclidian (Minkowski $p=2$)

Manhattan Distance : Works as pythagoras thm. (a+b)

$$m = \sum_{i=1}^{d} | x_{2i} - x_{1i} |$$



Euclidian distance problems:

1. Same scale : different cols have different scale.
2. Curse of dimensionality

↓

you can use **Manhattan** to improve a bit.

$p=1 \rightarrow$ Manhattan
$p=2 \rightarrow$ Euclidian

Euclidian : $\left(\sum_{i=1}^{d} (x_{2i} - x_{1i})^2 \right)^{1/2}$ → L2 Norm

Manhattan : $\left(\sum_{i=1}^{d} | x_{2i} - x_{1i} |^1 \right)^{1/1}$ → L1 Norm

→ P value

P ≥ 0 always

Minkowski → To take different p values

Minkowski eq : $\left(\sum_{i=1}^{d} (| x_{2i} - x_{1i} |)^p \right)^{1/p}$ → LP Norm

Time Complexity for KNN

$O(ND)$ → N → # of rows
D → # of columns

Space Complexity

$O(ND)$

Due to this high Time and space Complexity, we use hypermeter "algorithm".

Default : brute (Very expensive in terms of time and space)