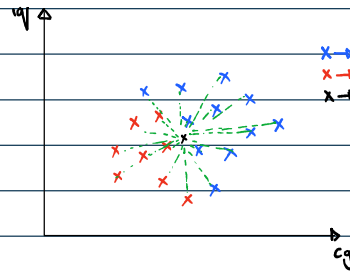
Algorithm Working

Sunday 14 April 2024 3:33 AM



X → 1
X → 0
X → ?

1) Choose a K value
Assume K = 3 → Hyperparameter
2) calculate distance w.r.t each neighbour
3) sort distances in ascending order
4) majority count i.e. out of the 3 closest point which class dominates?

e.g. 1 1 0
then X → X(1) [class label of the query point]

→ This logic is applicable in n-D also

→ Non-parametric ML Algo

→ evaluation metric : accuracy score

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

* bring all your data on the same scale [KNN relies solely on distances]

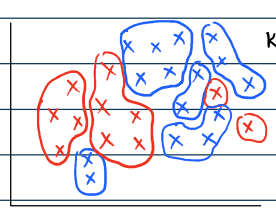
→ How to choose K ?

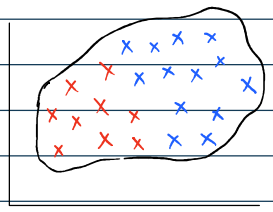
- avoid even numbers
- experiment / hyperparameter tuning

→ Decision Surface / Boundary

- visually divides the coordinate space into 'n' regions, where n is the # of classes in the target variable.

→ Overfitting & Underfitting in KNN





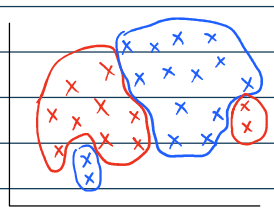
- max class always dominates

→ Limitations of KNN

- large datasets, as KNN is a lazy learner i.e. ND work during the training stage, all work is done when we have a query point.
  need to calculate distances → sort → majority [slow / Latency]

- High dimensional data → Curse of dimensionality
  The concept of calculating distances in higher dimensions is not reliable, but KNN totally depends on distances. If the distances are distorted so are our results.

- Outliers



need to choose K wisely else we risk overfitting

- Non homogeneous scales [features with higher values dominate & calculated distances are not reliable]

- Imbalanced dataset

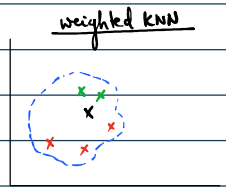
- good for prediction but not for inference [acts as a black box model]
  i.e. How much does feature 1 contribute to the prediction compared to feature 2, unable to deduce feature importance

→ KNN can be used for regression

- decide a K value
- find K nearest nearest neighbors
- Average/Median of K nearest neighbors

→ Explore Hyperparameters
- n_neighbors
- weights → weighted KNN
- metric → minkowski → p
- n_jobs
- algorithm
  - leaf_size
  - KD-Tree
  - ball-tree
  - brute

[Research]



uniform (majority count) x → x

distance → closer the point ↑ weight = $\frac{1}{\text{distance}}$ [add weights, pick class with highest weight]

→ Types of Distances
- Euclidean → $\left(\sum_{i=1}^{d}(X_{2i}-X_{1i})^2\right)^{1/2}$ → L2 norm
  - * Scale is crucial
  - * not reliable in higher dimensions
- Manhattan distance → $\left(\sum_{i=1}^{d}|X_{2i}-X_{1i}|^1\right)^{1/1}$ → L1 norm
  - * works better in higher dimensions
  - Lower vector norms work better in higher dimensions

→ P → hyperparameter

general eq = $\left(\sum_{i=1}^{d}(|X_{2i}-X_{1i}|)^p\right)^{1/p}$

→ Time / space complexity O(nd)

After some digging, I have some good answers. First let me tell you that as mentioned by some users like @anasvaf, you should only use odd number for binary classification. This is absolutely untrue. Firstly, when we use majority voting for binary classification, on some tie, it is entirely dependent on the actual library to choose the action. For example, in scikit-learn, it takes the mode of the variable. This means if in the training dataset, number of datapoints for class 1 is more, then 1 would be used on the tie. But there is a better solution.

What we can use weighted-KNN instead of normal KNN. In the weighed KNN, if there is a tie, we can see total distance for 1 labeled datapoints from k points and 0 labelled points. If total distance for 1 is more, then class would be 0 and vice-versa.

There are other good techniques too to handle tie in KNN but to be very honest, KNN is not a good learning algorithm specially due to its space of time complexity on large datasets.