Contents

[What is KNN 1](#_Toc520978190)

[How Does KNN works 1](#_Toc520978191)

[Value K 1](#_Toc520978192)

[Pros 3](#_Toc520978193)

[Cons 3](#_Toc520978194)

[Interview Questions 4](#_Toc520978195)

Revision notes

# What is KNN

KNN is non-parametric, instance-based and used in a supervised learning setting.

It is worth noting that the minimal training phase of KNN comes both at a memory cost since we must store a potentially huge data set, as well as a computational cost during test time since classifying a given observation requires a rundown of the whole data set. Practically speaking, this is undesirable since we usually want fast responses.

# How Does KNN works

1. Choose the number of K
2. Take the K nearest neighbor of new data points , according to Euclidean distance
3. Among these K neighbors, count the number of data points in each category .
4. Assign the new data point to the category where you counted the most neighbor.

**Step2 – calculating K nearest neighbors** - It runs through the whole dataset computing d between x and each training observation. We’ll call the K points in the training data that are closest to x the set A. Note that K is usually odd to prevent tie situations.

**Step 3** - It then estimates the conditional probability for each class, that is, the fraction of points in A with that given class label. (Note I(x) is the indicator function which evaluates to 1 when the argument x is true and 0otherwise)

**Step** **4** Finally, our input x gets assigned to the class with the largest probability

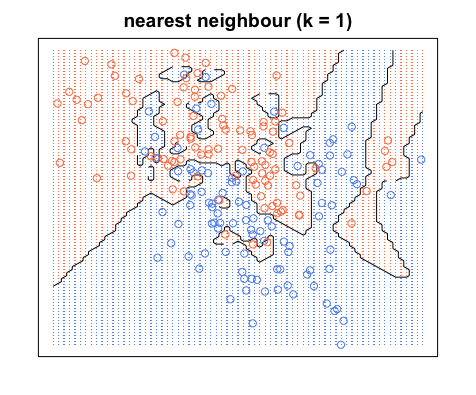
# Value K

Small k = small training error (small bias high variance)

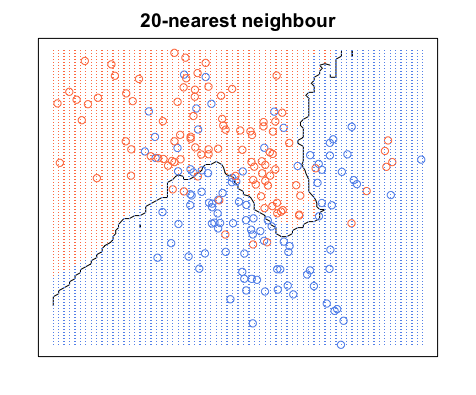
Large k =smooth decision boundary - prediction are more stable due to voting ( low variance bit more bias )

At this point, you’re probably wondering how to pick the variable K and what its effects are on your classifier. Well, like most machine learning algorithms, the K in KNN is a hyperparameter that you, as a designer, must pick in order to get the best possible fit for the data set. Intuitively, you can think of K as controlling the shape of the decision boundary we talked about earlier.

When K is small, we are restraining the region of a given prediction and forcing our classifier to be “blinded” to the overall distribution. A small value for K provides the most flexible fit, which will have low bias but high variance. Graphically, our decision boundary will be more jagged.



On the other hand, a higher K averages more voters in each prediction and hence is more resilient to outliers. Larger values of K will have smoother decision boundaries which mean lower variance but increased bias.



(If you want to learn more about the bias-variance tradeoff, check out [Scott Roe’s Blog post](http://scott.fortmann-roe.com/docs/BiasVariance.html). You can mess around with the value of K and watch the decision boundary change!)

# Pros

* No assumptions about data — useful, for example, for nonlinear data
* Simple algorithm — to explain and understand/interpret
* High accuracy (relatively) — it is pretty high but not competitive in comparison to better supervised learning models
* Versatile — useful for classification or regression

As you can already tell from the previous section, one of the most attractive features of the K-nearest neighbor algorithm is that is simple to understand and easy to implement. With zero to little training time, it can be a useful tool for off-the-bat analysis of some data set you are planning to run more complex algorithms on. Furthermore, KNN works just as easily with multiclass data sets whereas other algorithms are hardcoded for the binary setting. Finally, as we mentioned earlier, the non-parametric nature of KNN gives it an edge in certain settings where the data may be highly “unusual”.

# Cons

Computationally expensive — because the algorithm stores all of the training data

High memory requirement

Stores all (or almost all) of the training data

Prediction stage might be slow (with big N)

Sensitive to irrelevant features and the scale of the data

One of the obvious drawbacks of the KNN algorithm is the computationally expensive testing phase which is impractical in industry settings. Note the rigid dichotomy between KNN and the more sophisticated Neural Network which has a lengthy training phase albeit a **very fast** testing phase.

Furthermore, KNN can suffer from skewed class distributions. For example, if a certain class is very frequent in the training set, it will tend to dominate the majority voting of the new example (large number = more common).

Finally, the accuracy of KNN can be severely degraded with high-dimension data because there is little difference between the nearest and farthest neighbor.

# Interview Questions

**How is kNN different from k-means clustering?**

kNN, or k-nearest neighbors is a classification algorithm, where the k is an integer describing the the number of neighboring data points that influence the classification of a given observation.

K-means is a clustering algorithm, where the k is an integer describing the number of clusters to be created from the given data. Both accomplish different tasks.

https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/

**Q33.** **In k-means or kNN, we use euclidean distance to calculate the distance between nearest neighbors. Why not manhattan distance ?**

**Answer:** We don’t use manhattan distance because it calculates distance horizontally or vertically only. It has dimension restrictions. On the other hand, euclidean metric can be used in any space to calculate distance. Since, the data points can be present in any dimension, euclidean distance is a more viable option.

Example: Think of a chess board, the movement made by a bishop or a rook is calculated by manhattan distance because of their respective vertical & horizontal movements.