Data Analysis and Machine-Learning

Chapter 7:

KNN (K-Nearest Neighbor)

POLITICAL AND ECONOMIC RESEARCH INSTITUTE

1. Introduction

A Model without a Model.

So far we have investigated on model-based learnings, including linear regression, generalized linear models, logistic regression, support vector machine etc. More specifically, model-based learnings refer to the process of generating a model based on a dataset to make classifications or predictions. In other words, the procedure would be:

When X is given,

$$X \rightarrow f(X) \rightarrow \hat{Y}$$

Not all machine-learning algorithms, however, are necessarily based on this procedure. Instance-based learning, such as locally weighted regression or K-nearest neighbor that we will cover in this chapter, directly utilizes the ***adjacent data*** to make classifications and predictions without actually generating a model. The procedure for instance-based learning would be:

When X is given,

$$X \rightarrow Search\ for\ Nearest\ Xs \rightarrow \hat{Y}$$

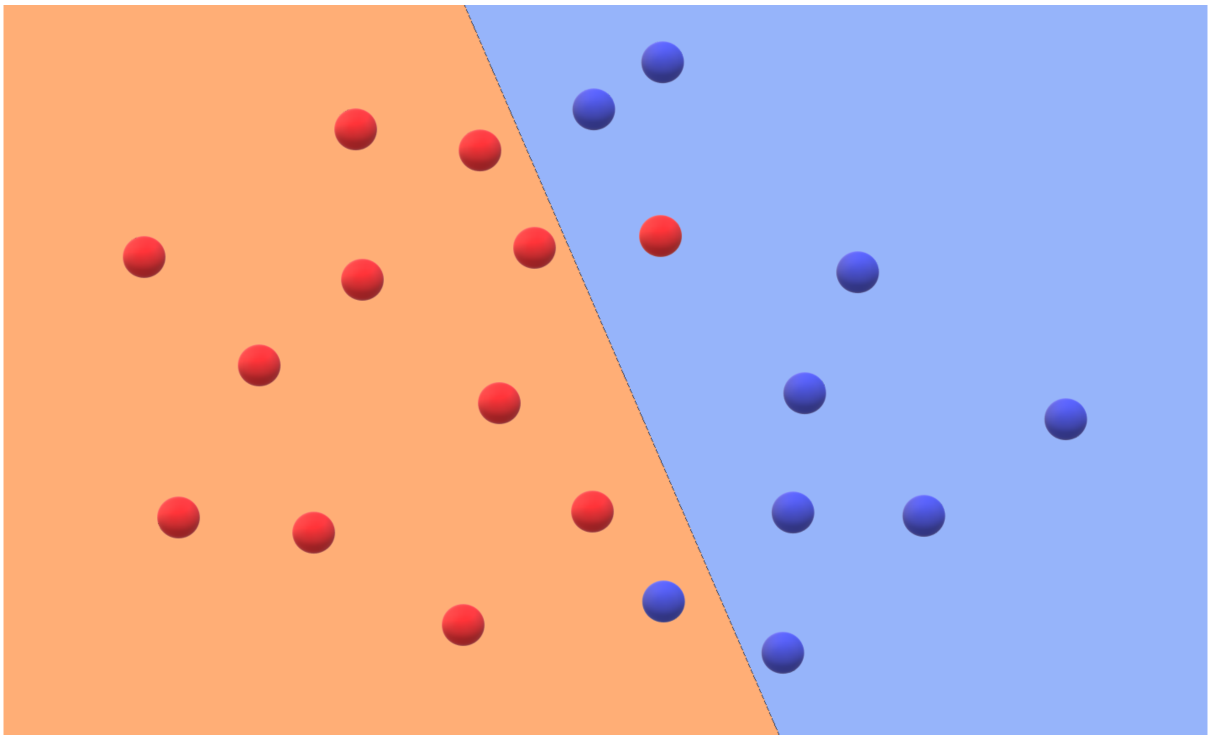
Despite of the intuitiveness of the logic, it might be difficult to grasp the idea that prediction can be accurate without actually going through a procedure of modelling when reflecting on what we have covered so far. So how does this actually work? I will illustrate the concept step by step with concepts and calculation procedures.

2. Basic Concepts of KNN

There are few unique features attached to the KNN algorithm. First, as briefly introduced above, KNN can be classified as an instance-based learning, which utilizes each ***instance*** of observation as a means of making a new prediction rather than generating a model with given dataset. Without instance, prediction cannot be validated. Second, ***all*** learning data are saved on memory slots and used for prediction, which is why KNN is also classified as a memory-based learning method. Third, KNN is also referred to as *lazy* learning, because the algorithm does not operate until the testing data are actually presented. This is because there is no priorly learned model in KNN.

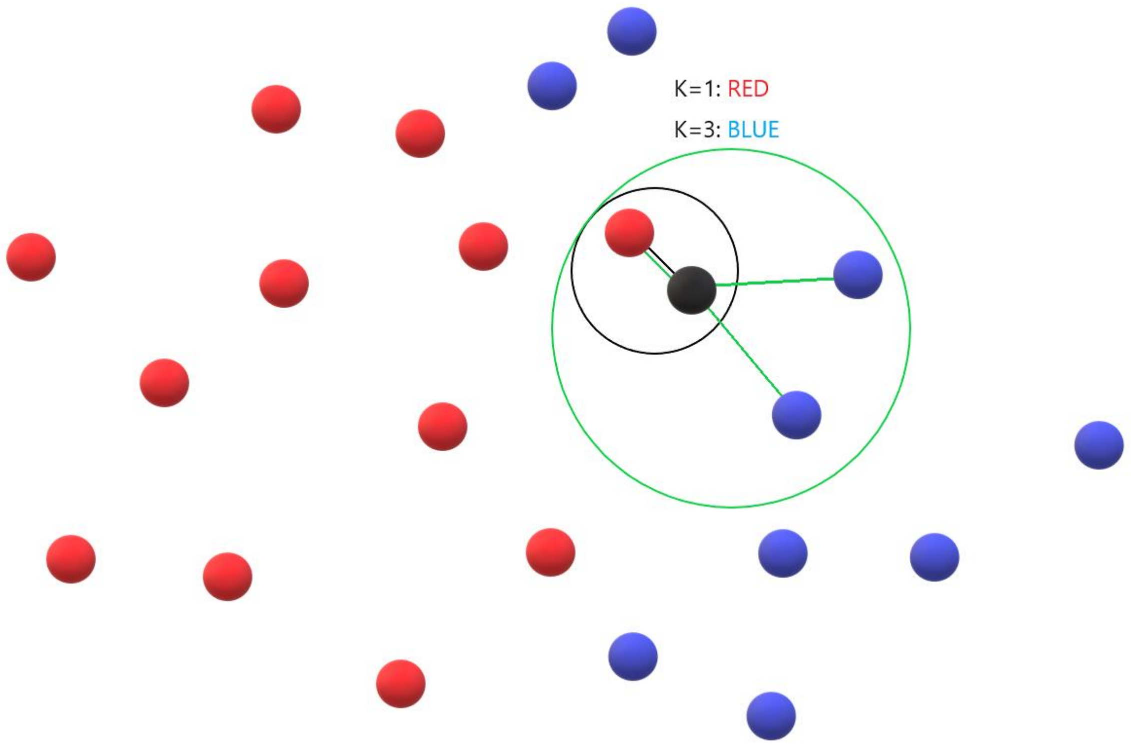
The methodological approach used in KNN is rather intuitive and simple, yet powerful in terms of classification accuracy. Consider following figure for understanding.

For comparison, let us first review linear models that we have covered earlier:



As the figure illustrates, linear models generate a linear boundary that divides classification with lines. The model thus typically comes with coefficients and bias (intercept), to figure out weights of each variable.

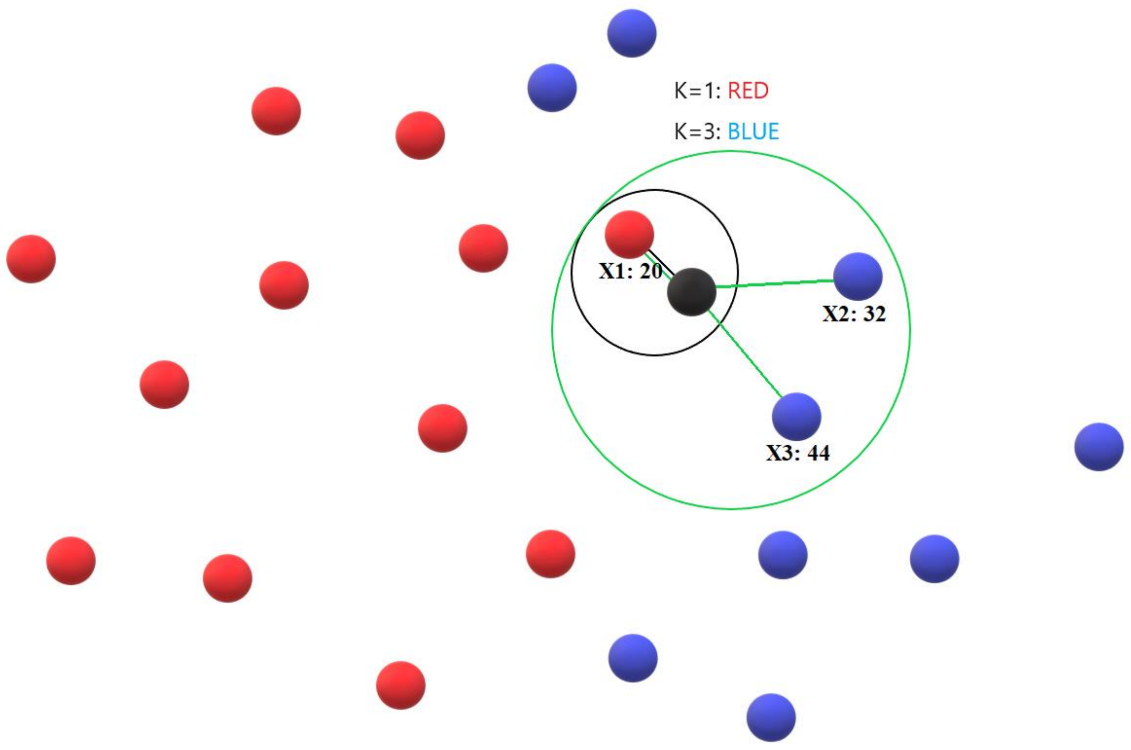
KNN, on the other hand, is basically non-linear and can be used for both continuous and classification data. More specifically, the algorithm adopts the concept of "distance" for predictions and classifications. Consider following figure for better understanding.



The K in KNN refers to the number of nearest neighbors (and thus the concept of ***distance,*** which we will cover in depth, becomes significant). Consider the case in the figure above. Suppose the black dot is the new data for classification. If the parameter K is set as 1, KNN algorithm will classify this datum as red, because the nearest neighboring datum in terms of "*distance*" is the red dot. If K is set as 3, on the other hand, the range of exploration circle becomes larger to incorporate three data that are relatively close to the presented black dot. Since there are two blue dots and one red dot in K=3 boundary, the KNN algorithm will determine the new datum as blue, accordingly to the decision by the *majority voting method*.

The classification algorithm for KNN thus works in the following sequence. First, new observation X to be classified is selected. Second, the algorithm explores a K number of learning data surrounding the observation X. Third, the Majority Class C is defined based on the explored K number of learning data. And finally, C is returned as the classification result of the observation X.

This basic concept remains the same for when KNN is used for continuous data. Consider the following figure for understanding.



When the continuous data are presented for KNN analysis, the arithmetic method (usually the geometric mean) is used for prediction instead of majority voting rule. For example, for the case above, the new Y for K=1 would be 20 (as X1 is 20). For K=3, the new Y for K=3 would be (20+32+44)/3 = 32.

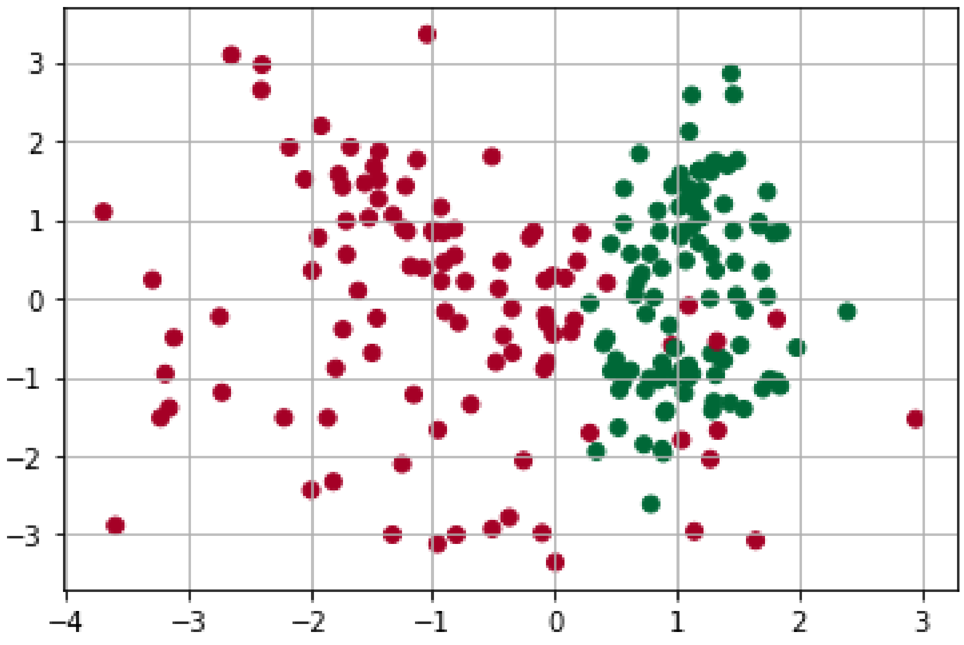
Why average instead of mode, median, min or max? The fact that default method for calculation is average is purely based on experiential accuracy. That is, empirically speaking, average method usually turned out to be the most accurate. Indeed, if considered necessary depending on the cases or the structure of data presented, mode, median, min or max method can be used, but the basic calculation concept of the algorithm would be the same.

The KNN prediction algorithm for continuous data is thus structured in the following sequence. First, observation X (testing data) to be predicted is selected. Second, K number of learning data surrounding X are selected. Third, the algorithm takes average on the selected K number of learning data, and returns the value as the prediction value for observation X.

3. Hyperparameter K

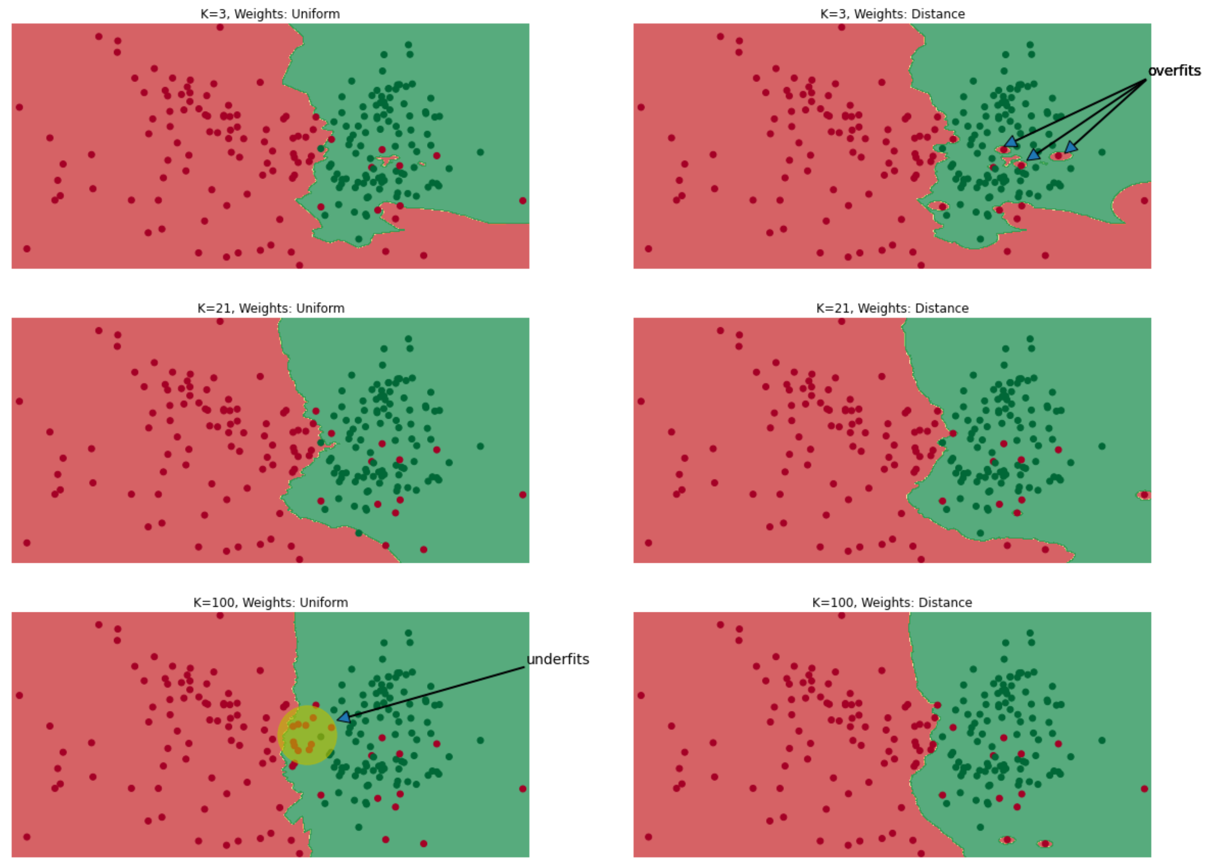
Then, the next important question is how to set appropriate hyperparameter K for best prediction results. Let us deduce the answer for this question using actual samples.

```
#Import Essential Libararies
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
#Generate Sample Data
x, y = datasets.make_classification(
    n_samples=200, n_features=2, n_informative=2, n_classes=2, n_redundant=0,
    n_clusters_per_class=2, random_state=123
)
np.c_[y, x]
x1, x2 = x[:,0], x[:,1]
#Visualization of the scatterplot
plt.scatter(x1,x2, c=y, cmap='RdYlGn')
plt.grid()
plt.show()
```



```
#Hyperparameter K and Distance Penalties
plt.figure(figsize=(20,15))
plt.subplot(3,2,1)
knn_model = KNeighborsClassifier(n_neighbors=3, weights='uniform').fit(x,y)
dt.dimensionchange(knn_model, x1, x2, cmap='RdYlGn', alpha=0.7)
plt.scatter(x1, x2, c=y, cmap='RdYlGn')
plt.title('K=3, Weights: Uniform')
plt.axis('off')

plt.subplot(3,2,2)
knn_model = KNeighborsClassifier(n_neighbors=3, weights='distance').fit(x,y)
dt.dimensionchange(knn_model, x1, x2, cmap='RdYlGn', alpha=0.7)
plt.scatter(x1, x2, c=y, cmap='RdYlGn')
plt.annotate('overfits', (1.9,-
0.2), (3,2), arrowprops={'width': 1}, fontsize=14)
plt.annotate('overfits', (1.4,-
0.4), (3,2), arrowprops={'width': 1}, fontsize=14)
plt.annotate('overfits', (1.1,0), (3,2), arrowprops={'width': 1}, fontsize=14)
plt.title('K=3, Weights: Distance')
plt.axis('off')

plt.subplot(3,2,3)
```

```
knn_model = KNeighborsClassifier(n_neighbors=21, weights='uniform').fit(x,y)
dt.dimensionchange(knn_model, x1, x2, cmap='RdYlGn', alpha=0.7)
plt.scatter(x1, x2, c=y, cmap='RdYlGn')
plt.title('K=21, Weights: Uniform')
plt.axis('off')

plt.subplot(3,2,4)
knn_model = KNeighborsClassifier(n_neighbors=21, weights='distance').fit(x,y)
dt.dimensionchange(knn_model, x1, x2, cmap='RdYlGn', alpha=0.7)
plt.scatter(x1, x2, c=y, cmap='RdYlGn')
plt.title('K=21, Weights: Distance')
plt.axis('off')

plt.subplot(3,2,5)
knn_model = KNeighborsClassifier(n_neighbors=100, weights='uniform').fit(x, y)
dt.dimensionchange(knn_model, x1, x2, cmap='RdYlGn', alpha=0.7)
plt.scatter(x1, x2, c=y, cmap='RdYlGn')
plt.scatter(0.1,0, s=3300, c='y', alpha=0.6)
plt.annotate('underfits', (0.5,0.4), (3,2), arrowprops={'width': 1}, fontsize=
14)
plt.title('K=100, Weights: Uniform')
plt.axis('off')

plt.subplot(3,2,6)
knn_model = KNeighborsClassifier(n_neighbors=100, weights='distance').fit(x,y)
dt.dimensionchange(knn_model, x1, x2, cmap='RdYlGn', alpha=0.7)
plt.scatter(x1, x2, c=y, cmap='RdYlGn')
plt.title('K=100, Weights: Distance')
plt.axis('off')
```



As it can be inferred from the illustration,

If K is too small (e.g., K=3 for N=200), the algorithm may overly reflect regional characteristics, resulting in overfitting problem (consider the first and second figure, with K=3).

If K is too large (e.g., K=100 for N=200), the algorithm may include too many unities of opposing classification data, which may result in misclassification (underfitting problems) (consider the fifth and sixth figure, with K=100).

4. Selection Method for hyperparameter K

Although there is no absolute principle in terms of setting the exact number of K, it is possible to find regular scope of optimal parameter by using algorithm performance evaluation and cost functions (i.e., gridsearching).

For classifications, misclassification error E for indicator function I can be defined as:

$$E_k = \frac{1}{k}\sum_{i=1}^{k} I(c_i \neq \hat{c}_i) for\ k = 1,2,3,\dots,k^*$$

where indicator function presents either 0 (false) or 1 (true)

And for continuous data, ordinary least squared method can be applied as:

$$SSE_k = \sum_{i=1}^{k} (y_i \neq \hat{y}_i)^2 for\ k = 1,2,3,\dots,k^*$$

Based on such error function, the best K can be located in the error function graph for training data and testing data where:

$$K_{Best} = min\ |E_{test} - E_{train}|$$

5. Distance

In prior to mentioning about the concept of distance in KNN, it is foremost important to discuss the significance of standard scaling prior to adopting distance parameter. For instance, distance can be either inflated or deflated for each scale (e.g., \$100,000,000~\$200,000,000 vs. 150cm~190cm). Regularization or standardization is necessary in prior to adopting the concept of distance in such cases.

Now going into the concept of distance, there are several methodologies in terms of measuring distances, including the most common Euclidean distance, Manhattan distance, correlation distance, and Mahalanobis distance.

Euclidean distance, which is most commonly used also in terms of KNN, is defined as,

$$Distance_{(X,Y)} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

which refers to the straight linear distance between two observations. As it can be inferred from the equation, the main benefit of the Euclidean distance is that the resulting value is always a single value (1 by 1) regardless of the number of dimensions.

Manhattan distance, on the other hand, can be defined as:

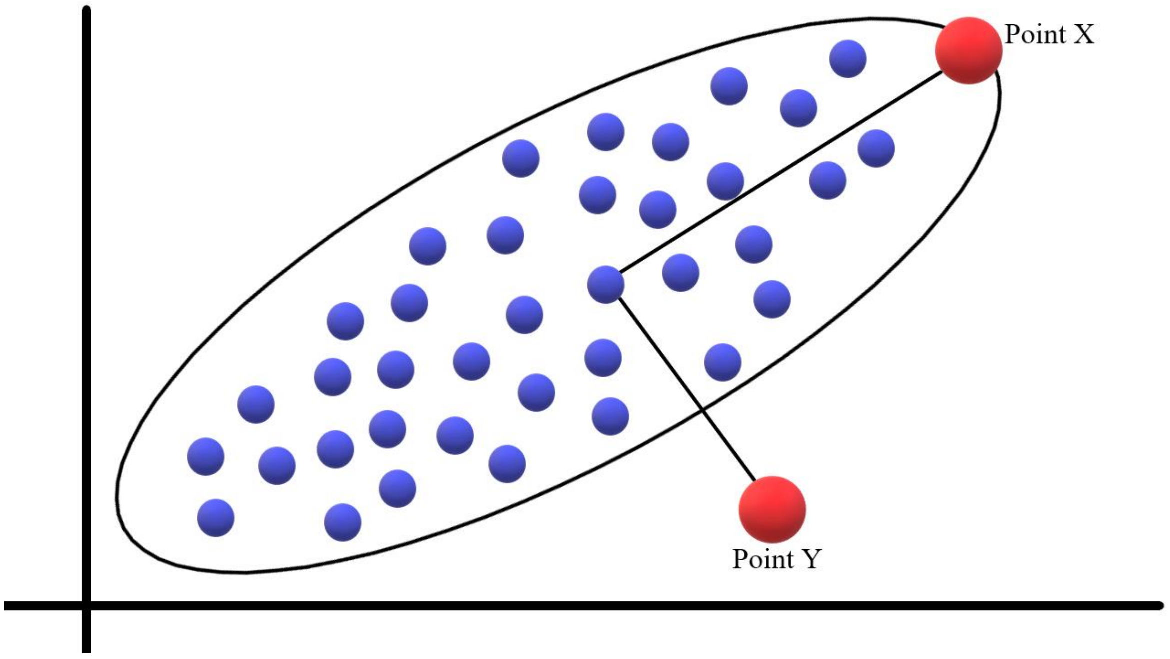
$$Distance_{Manhattan(X,Y)} = \sum_{i=1}^{n}|x_i - y_i|$$

Manhattan distance supposes a grid between the two points, and the distance is calculated via calculating the shortest grid course from point A to point B.

Lastly, Mahalanobis distance, a statistically powerful and accurate method of distancing in terms of KNN application, is defined as:

$$Distance_{Mahalanobis(X,Y)} = \sqrt{(X - Y)^T \sum^{-1} (X - Y)}$$

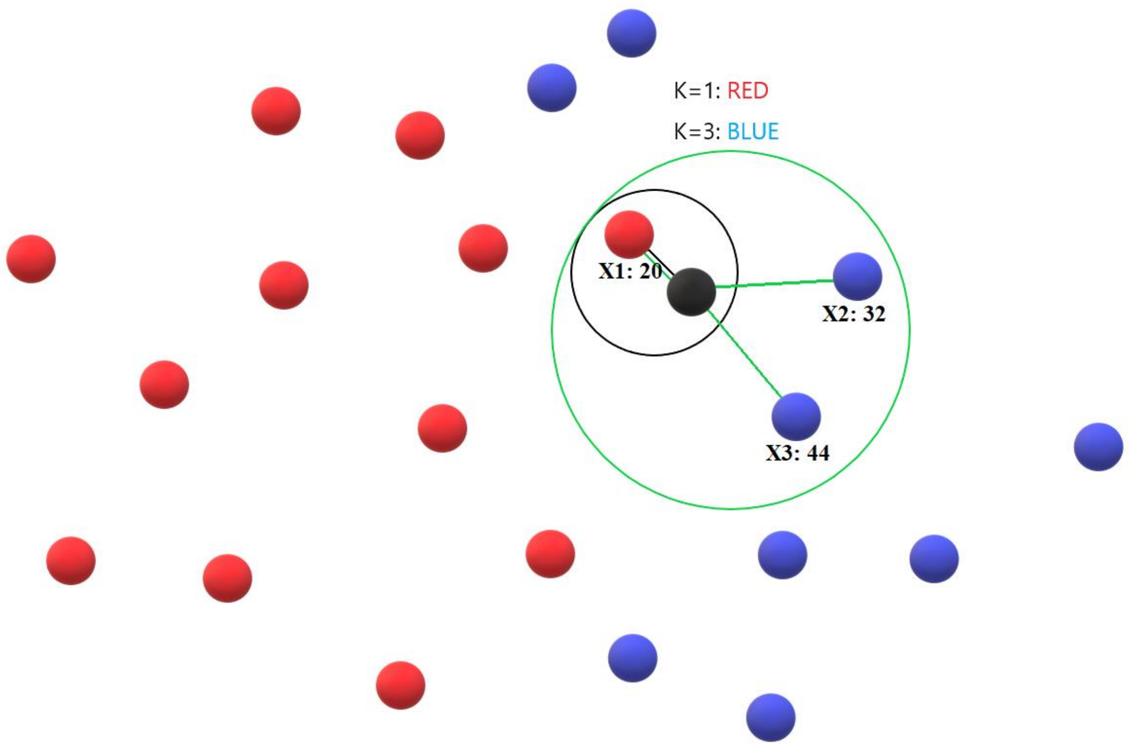
Notice that the equation is basically the same with the Euclidean distance equation except for the inverse of covariance matrix function in the middle. The peculiar feature of Mahalanobis distance is that variance and covariance are both reflected in the calculation process of the distance. Thus, Mahalanobis distance method can be particularly useful in terms of statistical accuracy and machine-learning predictions because it accounts for the dispersity of data points. For better understanding, consider the following figure.



In terms of Euclidean distance, point X is obviously farther away from point Y. This, however, may not be statistically accurate when considering the variance structure of the data points. Mahalanobis distance calculation, on the other hand, accounts for the variance and covariance in the calculation process, as illustrated in equation earlier, and the adjusted result indeed implies that point Y is farther away from point X. In a sense, therefore, Mahalanobis distance provides weights (or penalties) accordingly to the shape of the data dispersion. In this context, the main advantage of KNN with mahalanobis distancing method is that it is relatively immune to data noises.

6. Weighted KNN

One significant limitation of the average method that we have covered earlier is that the effects of distances are not considered in terms of calculation. Consider the following figure again.



X2 and X3 are obviously farther away from the black dot compared to the distance to the red dot. The average method, however, does not incorporate such problem, which may result in inaccurate estimation.

Weighted KNN solves such problem by attributing weights accordingly to the distance between the new data and the learning data.

For classification model, the equation can be defined as:

$$\hat{c}_{new} = argmax_c \sum_{i=1}^{k} w_i I(w_i \in c)$$

And for continuous data:

$$\hat{y}_{new} = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i} \ where \ w_i = \frac{1}{d_{(new,x_i)}{}^2}$$

As weight is defined as the inverse proportional value to the distance squared, little weight is attributed to longer distances, which makes more sense in terms of algorithmic accuracy.