# Section C:

K-Nearest Neighbour Algorithms is easy to understand for freshers of machine learning. However, it still contains many problems which deserve to study and learn. This section will introduce how KNN works and explore some performance and efficiency problems when implementing the KNN algorithm and corresponding optimizations. Then,

## 3.1 Introduction for K-Nearest Neighbour algorithms

K-Nearest Neighbour is abbreviated as KNN, which can be used to solve both regression and classification problems. The core concept of KNN can be summarised as that if things are similar, they are closed to each other. Briefly speaking, in KNN, the class or the regression predicting the value of the test sample is determined by the “votes” of the K nearest neighbours. The following is the specific steps of using basic KNN.

1. Map the training data to n dimension space. (n is the number of features in a dataset)

2. Choose a K as the number of neighbours (if classification, K should be an odd number in most cases)

3. Computing the distance between the test data and each training data according to the feature value.

4. Ordering the distances from smallest to largest (ascending order)

5. Retrieve the first K in this order

6. If regression, the mean of the targeted labels of these K neighbours need to be calculated. If classification, the mode of the classes of these K neighbours need to be calculated.

The suitable scenarios for KNN are to deal with the low-dimension dataset with a small distance between classes (but it is sensitive for unbalanced datasets). The following will describe in more detail.

## 3.2 Further Exploration and Optimization

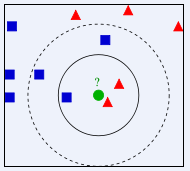
1. **Time Complexity**: KNN is the representative applying for “Lazy Learning”, which means that some algorithms do not need data training and model parameters before testing. However, to some degree, these algorithms just delay the time consumption of data training to the data test stage. Lack of generalization requires KNN to utilize all training data in the test stage. Based on the time complexity in the test stage, O(N\*D), where N is the number of a sample set, and D is the number of dimensions, a conclusion can be made that if N or D becomes enormous, the time complexity and computational volume becomes large. It also means KNN is not suitable for a high-dimensions sample set.

**KD Tree**: This kind of tree data structure stores instance points in multidimensional space for rapid retrieval. If the dataset for training is stored in the KD tree in advance, part of the time complexity of KNN can be transferred to the space complexity, and the efficiency can be improved in the data test stage.

1. **Wild Difference between Features:** Features in real-world datasets often vary wildly in range, unit. And distance-based algorithms like KNN are the most vulnerable to unscaled data. The model will heavily bias towards features with a greater magnitude and ignore the influence of the other features if the model assigns features with a greater magnitude higher weight.

**Feature Scaling:** For avoiding such a consequence, feature scaling needs to be executed before pouring datasets into the model. Z-score standardization, , will be applied in the following exercise. Using standardization, each dimension in datasets has **zero mean and unit variance**.

1. **Hyperparameters Tuning:** In KNN, the hyperparameters can be K and distance metric. Hyperparameters also affects the algorithm performance. For instance, as can be seen from the chart below, K takes different values, the green points will classify into different groups. More specifically, choosing optimal hyperparameters is a good way to balance the train error and variance.

In most cases, Euclidean distance is selected as the distance metric by default. However, many other distance measures also can be used, such as cosine distance, Mahalanobis distance, etc. Each specific dataset has a different optimal distance measurement for classification or regression. This report will explore the generalization of Euclidean distance, Minkowski distance as distance metric where p is a hyperparameter.

**RandomizedSearchCV:** Compared with GridsearchCV, this method to find optimal hyperparameter does not need to try all possible combinations but obtain the random values of each hyperparameter range based on their statistic distribution and then combine them to do the crossing validation.

1. **Unbalanced Classes:** Real-world datasets have different data volumes for different classes, leading to a situation where there are more points belonging to the class with more data volume in the first K neighbors. This problem especially influences the model performance for datasets with a small distance between classes.

**Add Different Weights to K neighbors:** The solution is to give more weight to the points which are nearby and less weight to the points which are farther away when computing the distance. Typically, the Gaussian kernel can be used to optimize the weight of samples with different distances.

# 3.3 Applying for KNN to Real World

**1. The choice of dataset:** This retrieved dataset from UCL Machine Learning repository (<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>) is about breast cancer patients in Wisconsin with malignant and benign tumors. Excluding “id” and “diagnosis” (M = malignant, B = benign), this dataset has 30 dimensions (features) and 569 samples with 357 benign and 212 malignant. It does not have any missing attribute values. Meanwhile, all the feature types are continuous while “diagnosis” is categorical variable. This dataset can be trained to predict the patient's breast cancer tumor is malignant or benign.

2.