9. K-Nearest Neighbors (K-NN) algorithm

- **Parametric vs. Non-parametric ML Algorithms**

  A model *parameter* is a variable that is internal to the model and whose value is learned from the training data.

  - parameters are often saved as part of the learned model
  - parameters are required by the model when making predictions
  - examples of model parameters are:
    - weights and biases in perceptron, linear and logistic regressions, neural networks

  Machine learning algorithms are classified into two distinct groups:

  - *parametric*: perceptron, linear regression, logistic regression, neural networks, etc.
  - *non-parametric*: K-nearest neighbors, decision trees, etc.

  Feature engineering is important in parametric algorithms because you can poison the model if you feed a lot of unrelated features. Non-parametric algorithms handle feature engineering easily – we can feed all the features we have and the algorithm will ignore unimportant features.

  Non-parametric models are sometimes slower and require large amounts of data, but they are rather flexible as they don't need many assumptions about the data.

- **Eager vs. Lazy Learners**

  - *Eager learners* are those when given the training data, a generalized model will be constructed before performing prediction on a new data instance. You can think of such learners as being ready, active, and eager to make predictions on new data points. Examples are: perceptron, linear and logistic regressions, neural networks, decision trees, support vector machines, etc.

  - *Lazy learners* means that there is no need for training of the model and these models wait until the last minute before making a prediction on a new data point. Lazy learner just remembers the training data set and waits until the prediction needs to be made. Unlike eager learning methods, lazy learners do less work in the training phase and more work in the testing phase.

- **K-Nearest Neighbors (K-NN) Algorithm**

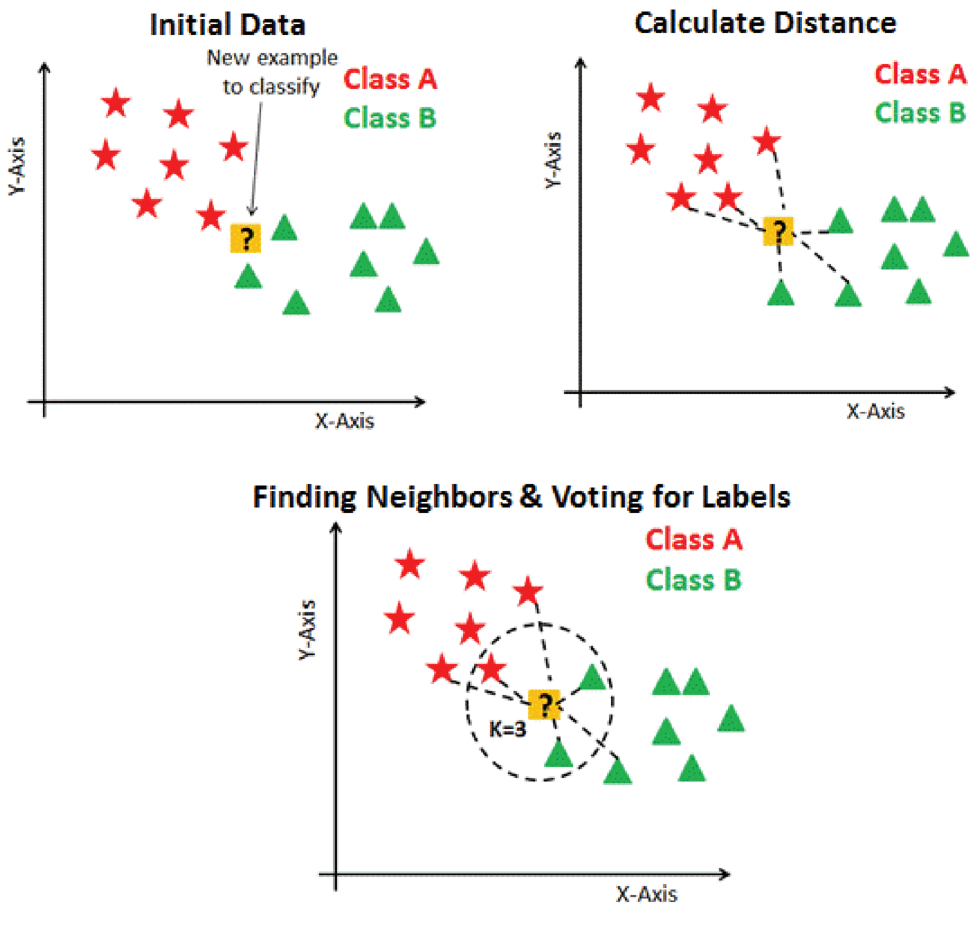
  - It is a supervised machine learning algorithm used for both classification and regression problems. K-NN is extremely easy to implement in its most basic form and yet can perform very well on quite complex tasks.

  - K-NN is a lazy learning algorithm since it doesn't have a specialized training phase.

  - K-NN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data. This is very useful since most of the real world data doesn't really follow any theoretical assumptions such as linear-separability, uniform distribution, etc.

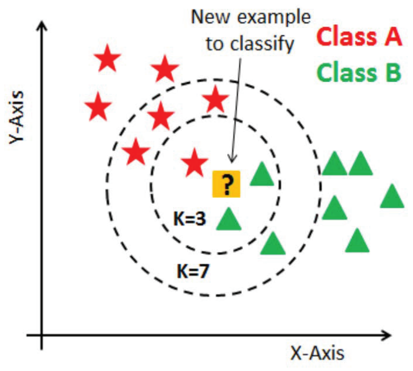
- K-NN uses a very simple approach to perform classification or regression. When tested with a new data point, it looks through the training data and finds K training data points that are closest to the new data point. From these neighbors, a summarized prediction is made:
  * in K-NN classification, the output is the class most common among the K nearest neighbors.
  * in K-NN regression, the output is the average of the target values of the K nearest neighbors.

  (As an intuitive example of why this works, think of your neighbors. Your neighbors are often relatively similar to you. They are probably in the same socioeconomic class as you. Maybe they have the same type of work as you, maybe their children go to the same school as yours, and so on. But for some tasks, this kind of approach is not as useful. For instance, it would not make any sense to look at your neighbor's favorite color to predict yours.)



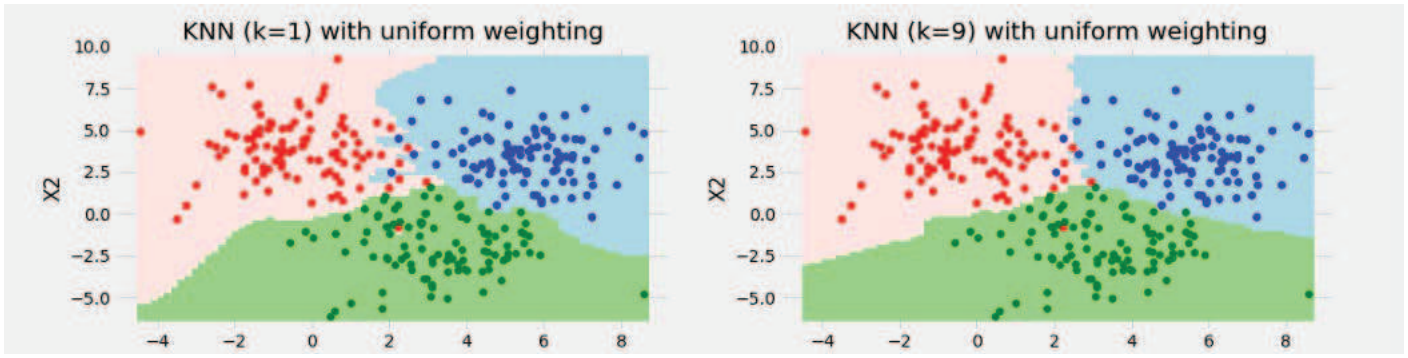
https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

- **What is "K" in the K-NN Algorithm?**
  - K in the K-NN algorithm is the number of nearest neighbor points which are used for the new data's prediction.



https://ai.plainenglish.io/introduction-to-k-nearest-neighbors-knn-algorithm-e8617a448fa8

  - In general the training accuracy of the model rises as the model complexity increases. For K-NN the model complexity is determined by the value of K. Larger K leads to a less complex model and a smoother decision boundary. Smaller K leads to a more complex model and may lead to over-fitting.



https://towardsdatascience.com/k-nearest-neighbors-k-nn-explained-8959f97a8632

- **Distance between two data points**

  There are many distance functions one can use, depending on the actual problem.

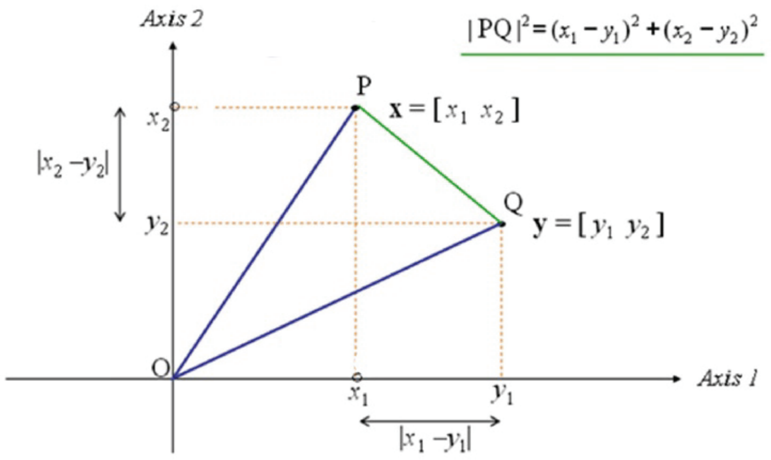
  Distance function (or metric) is any function $d : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$ such that for all $x, y, z \in \mathbb{R}^d$

$$d(x,x) = 0$$
$$d(x,y) = d(y,x) \qquad \text{symmetry}$$
$$d(x,y) \le d(x,z) + d(z,y) \qquad \text{triangle inequality}$$

Some commonly used distances are:

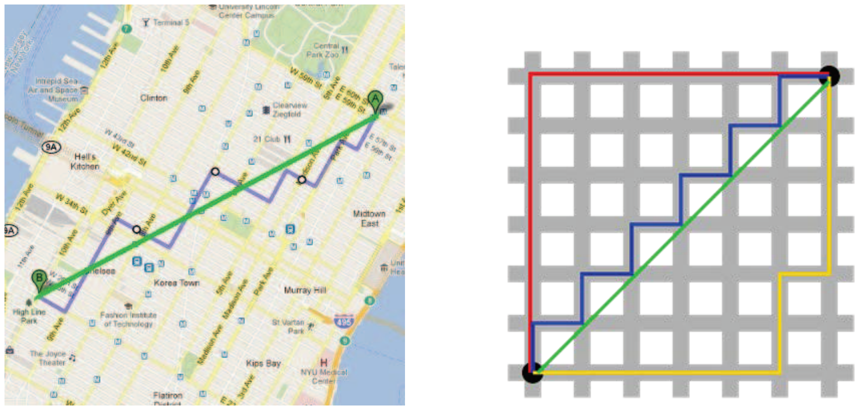
- *Euclidean distance* (or L^2-norm)

$$\begin{aligned} d(x,y) &= \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\ldots+(x_d-y_d)^2} \\ &= \sqrt{(x-y)^T(x-y)} \\ &= \|x-y\|_2 \end{aligned}$$



https://www.researchgate.net/figure/Representation-of-Euclidean-distance_fig1_307173163

- *Manhattan (or city-block) distance* (or L^1-norm)



https://www.quora.com/What-is-common-between-the-Euclidean-distance-and-the-Manhattan-distance

$$\begin{aligned} d(x,y) &= |x_1-y_1|+|x_2-y_2|+\ldots+|x_d-y_d| \\ &= \|x-y\|_1 \end{aligned}$$

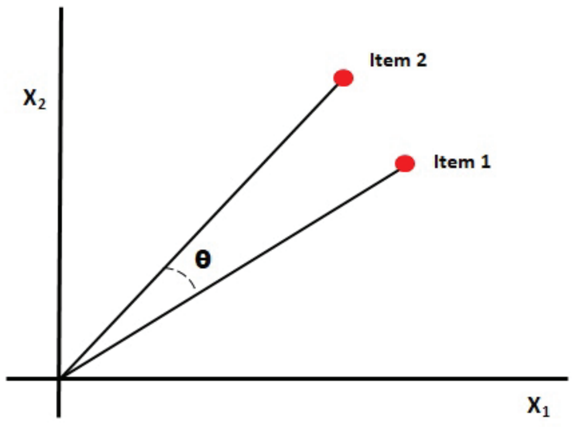
– *maximum distance* (or L^∞ norm)

$$\begin{aligned} d(x,y) &= \max_{1\le i\le d} |x_i - y_i| \\ &= \|x-y\|_\infty \end{aligned}$$

Remark: Dissimilarity (or proximity) function is any function $d : \mathbb{R}^d \times \mathbb{R}^d \to [0,\infty)$ such that for all $x, y \in \mathbb{R}^d$

$$\begin{aligned} &d(x,x) = 0 \\ &d(x,y) = d(y,x) \quad \text{symmetry} \end{aligned}$$

One of the most useful dissimilarity functions in Natural Language Processing is *cosine dissimilarity*



https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/eb9cd609-e44a-40a2-9c3a-f16fc4f5289a.xhtml

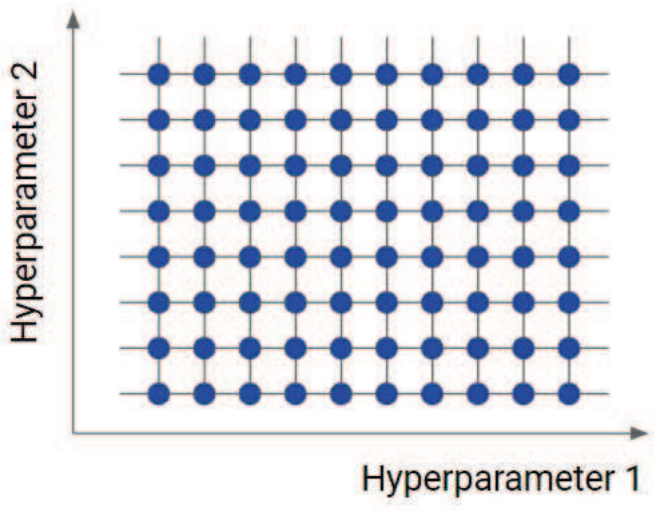
$$\cos(\theta) = \frac{x^T y}{\|x\|_2 \cdot \|y\|_2}$$

$$d(x,y) = 1 - \cos(\theta)$$

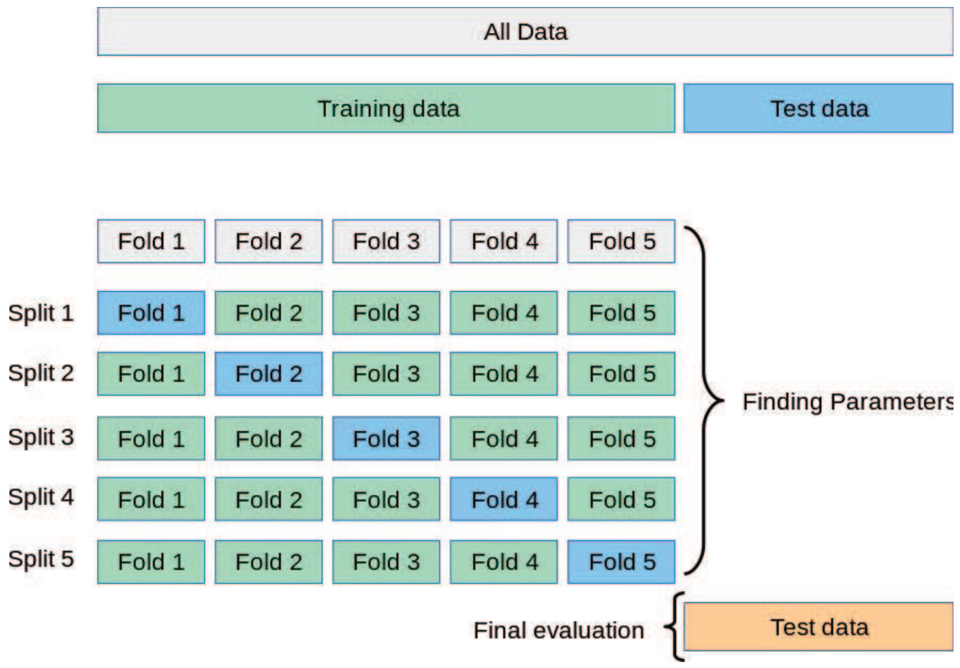
- **Implementation of the K-NN Algorithm**
  1. If the features have different scales, scale the features so that variance in each direction is 1
  2. Given a new data point, calculate the distances between the new data point and all the points in the training data
  3. Find K nearest training data points
  4. Make the prediction on the new data point:
     - classification: majority vote, weighted majority vote
     - regression: average, weighted average
- **Advantages and Disadvantages of K-NN**
  - Advantages:
    * No assumptions about data
    * Simple algorithm: easy to implement and easy to understand
    * Can be used for classification and regression
    * It is a lazy learning algorithm and therefore requires no training prior to making predictions
    * There are only several hyper-parameters required to implement K-NN such as the value of K, distance function, and whether we want to use straightforward predictions or weighted predictions
  - Disadvantages:
    * High memory requirement – all of the training data must be present in memory in order to find the closest K neighbors
    * Sensitive to the scale of the data since we are computing distances
    * Does not work well with high dimensional data (becomes very costly) because with large number of dimensions, it becomes difficult for the algorithm to calculate distances
    * Does not work always well with categorical features since it is difficult to find the appropriate distance function

Python code: Lecture_8_KNN.ipynb

`sklearn GridSearchCV` and `RandomizedSearchCV` are used to select optimal hyper-parameters.



https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random-search/



https://scikit-learn.org/stable/modules/cross_validation.html

Homework 5:

- Consider the Abalone data set available at UCI ML Repository

  https://archive.ics.uci.edu/ml/index.php

- You can upload this data set into your jupyter notebook using the following code

  ```
  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
  df = pd.read_csv(url)
  ```

- Change the names of the columns using the following code

  ```
  df.columns = [ "Sex", "Length", "Diameter", "Height", "Whole weight", "Shucked
  weight", "Viscera weight", "Shell weight", "Rings"]
  ```

- The goal of this homework is to use K-Nearest Neighbors Regressor to predict the number of rings `df["Rings"]` (target variable) using the features given. Note that one of the features is categorical and you can choose to drop it or to encode it numerically.

- You can either use the code we created from scratch and modify it for this problem or you can use `sklearn KNeighborsRegressor`

  https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

- Divide the data set into subsets for training and testing.

- Since this is a regression problem, the appropriate error functions you may use are MSE, MAE, and RMSE. Consider several different choices of the hyper-parameter K and see how the error changes with respect to K.

- Use `GridSearchCV` to select the optimal values of K (and maybe other hyper-parameters) and report the model performance.

References:

[1] Youtube video on MIT Artificial Intelligence course 6.034
Prof. Patrick Winston's lecture 10 on Nearest Neighbors