Scikit-learn and Tour of Classifiers

K-Nearest Neighbors

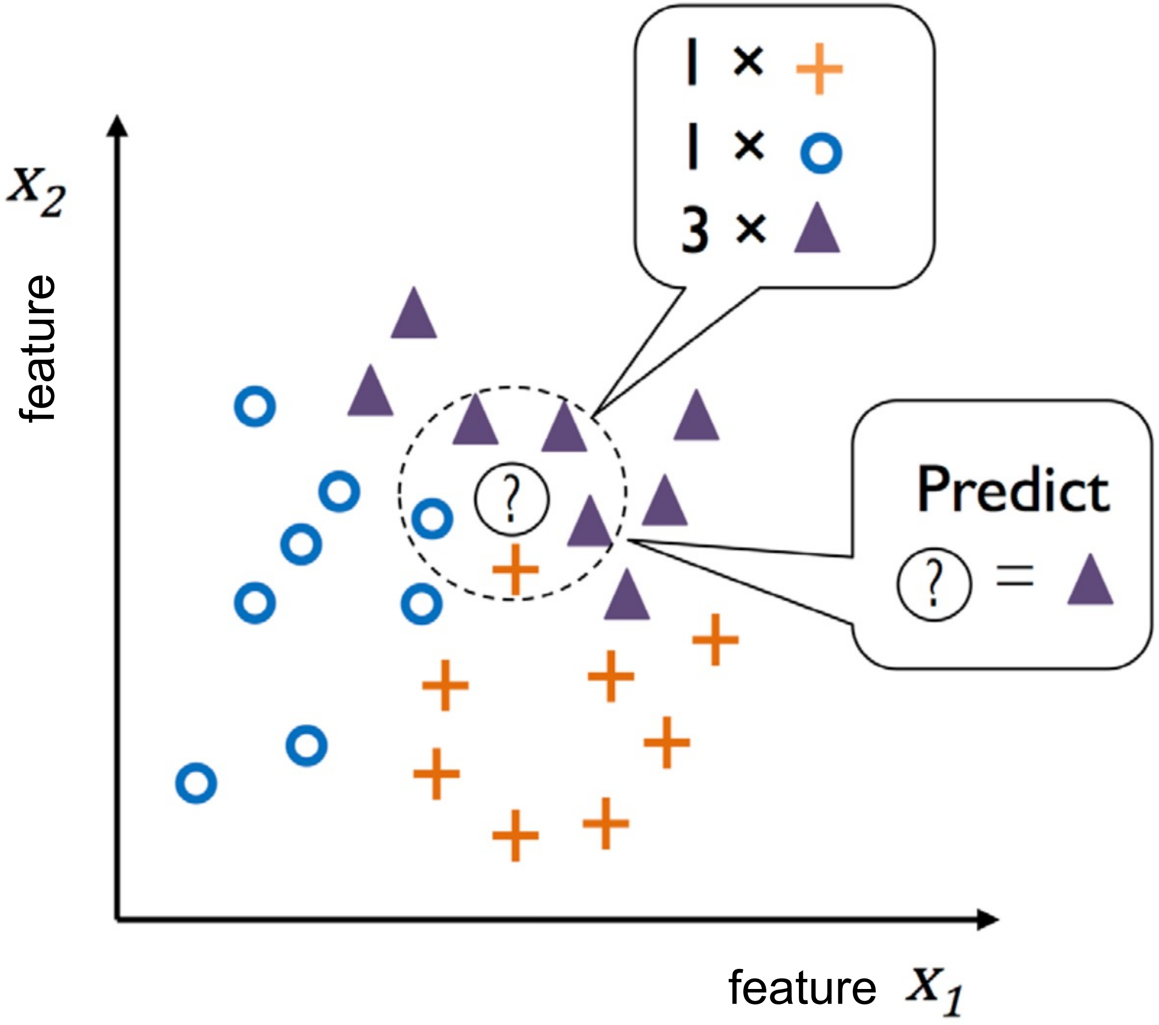
K-Nearest Neighbors

- Fundamentally different algorithm from what we learned so far
- Does not learn a discriminative function
- Memorizes the training data set and makes prediction based on that

K-Nearest Neighbors – Algorithm

1. Choose parameter ”**k**” and a **distance metric**
2. Find the k-nearest neighbors of a data record that we want to classify
3. Assign the class label by majority vote
    - *Tie break*: take the class of the closest neigbor and if still tied use lowest label

K-Nearest Neighbors – Algorithm

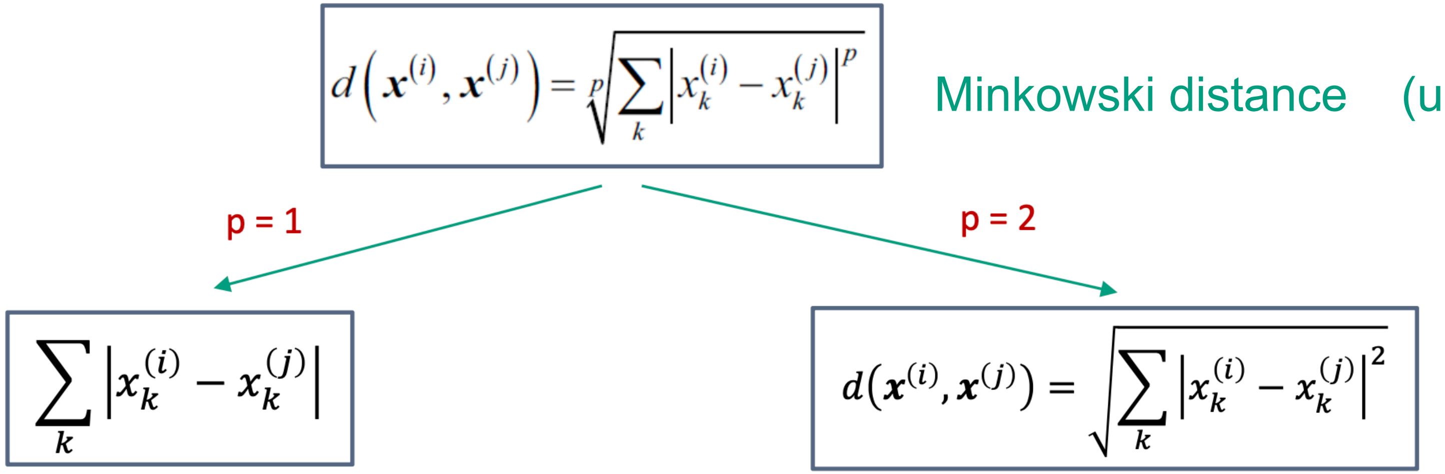


Example: 5-nearest neighbors of a new data record *(?)* that we want to classify

Figure from Python Machine Learning (Raschka & Mirjalili)

K-Nearest Neighbors – Distance metric

- A **distance metric** measures **distance** between two samples
- **Should fit the type of data**
- Common for real numbers: Euclidean distance



$$d\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \sqrt[p]{\sum_k \left|x_k^{(i)} - x_k^{(j)}\right|^p}$$

Minkowski distance (used in scikit-learn K-NN)

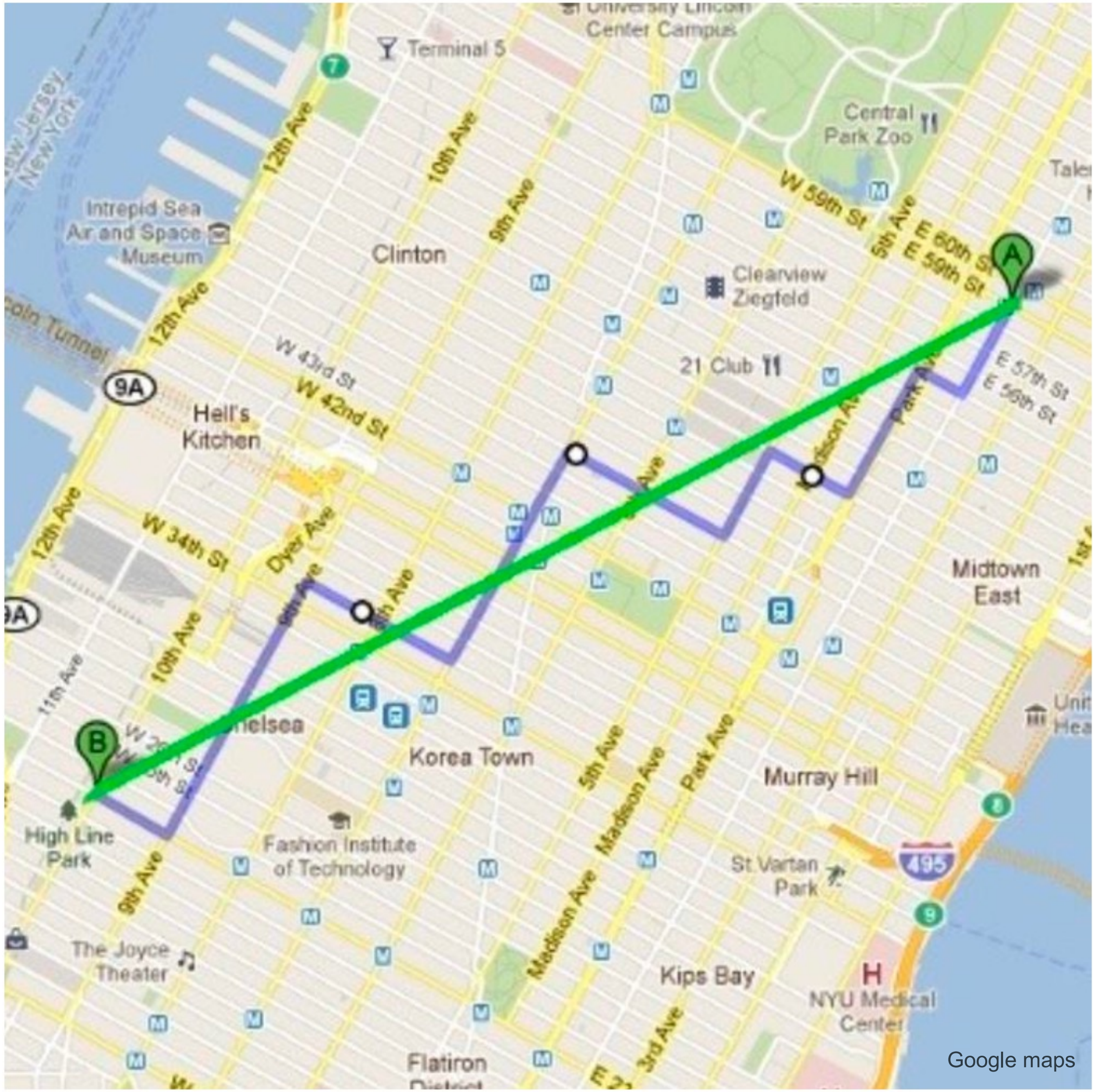
$$\sum_k \left|x_k^{(i)} - x_k^{(j)}\right|$$

Manhattan distance

$$d\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \sqrt{\sum_k \left|x_k^{(i)} - x_k^{(j)}\right|^2}$$

Euclidean distance

- Euclidean distance
- Manhattan distance



Google maps

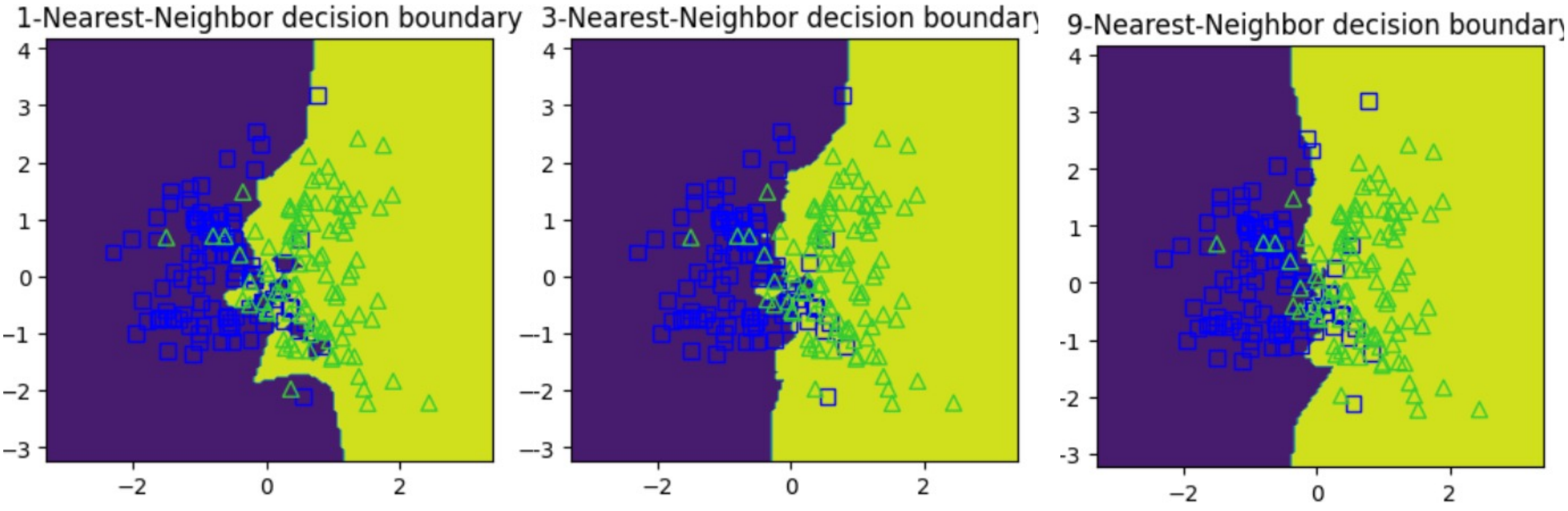
K-Nearest Neighbors

- Tuning **k** (and distance metric) is crucial to find a good balance between overfitting and underfitting
  - Lower **k** means that the model is more complex and has higher variance
  - Higher **k** means less variance (consulting more neighbors) but higher bias (far away neighbors might be consulted)
  - Higher **k** means that prediction becomes more costly

K-Nearest Neighbors

03_knn.ipynb

- Code example, testing accuracy on test/train set for different **k**



Parametric versus non-parametric models

Parametric versus non-parametric models

- **Parametric models**
  - Learn **parametrized** discriminative functional (fixed set of parameters)
  - Predict/classify new points **without need for** the training data
  - Examples: Perceptron, Adaline, Linear Regression, Logistic Regression, (*linear* SVM)
- **Non-parametric models**
  - No fixed set of params, no. of parameter **changes with amount of training data**
  - Training data (at least a subset) is needed for prediction
  - Examples: Decision trees, most kernel machines (Kernel-SVM, Kernel-Perceptron, …)

Some advantages and disadvantages of a memory-based non-parametric approach like K-NN

- Advantages:
  - Low (or zero) cost adaption to new training samples
  - Good predictive vs computational performance for small to medium-sized data sets
- Disadvantages:
  - Storage and prediction cost grows with the number of samples (prediction cost can be lowered by using efficient data structures, e.g. k-d trees for K-NN)
  - Curse of dimensionality → for a high number of features, prone to overfitting, no other training sample may be informative due to increasingly sparse feature space population with increasing dimension (number of features)