Machine Learning

FIB, Master in Data Science

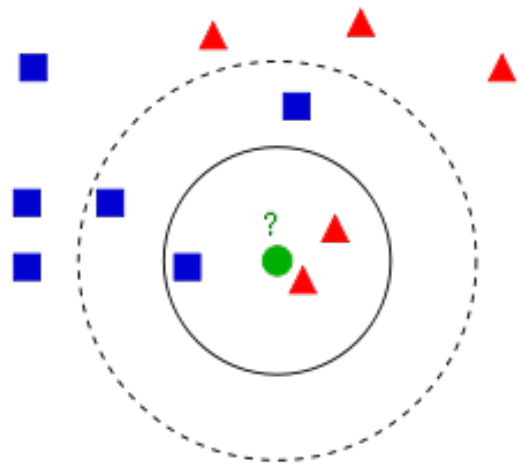
Marta Arias, Computer Science @ UPC

Nearest neighbors

Nearest neighbor prediction

The **nearest neighbor** predictor uses the **local** neighborhood to compute a prediction

- 1-NN predicts using same target as the **closest** training example
- k-NN predicts using combination of targets from k **closest** training examples



Nearest neighbor pseudocode

- **Training:**
  1. store all examples

- **Prediction:** given new example $\mathbf{x}$:
  1. Compute distance/similarity with all examples in training set
  2. Locate k closest
  3. Emit prediction by combining labels of k closest

There are three things to decide when applying k-NN:

1. Distance/similarity function
2. How many neighbors k to choose
3. How to combine labels of k neighbors to emit final prediction

Locality: similarities and distances

What does "closest" neighbor mean?

A crucial aspect of nearest neighbor prediction is the notion of **distance/similarity** since it affects very directly the predictions made

Let a, b, c be examples in some feature space. Then a **distance function** should satisfy:

1. $d(a, b) \geq 0$
2. $d(a, b) = 0$ iff $a = b$
3. $d(a, b) = d(b, a)$ (symmetry)
4. $d(a, b) \leq d(a, c) + d(c, b)$ (triangle inequality)

If using a **similarty function**, it should satisfy:

1. $-1 \geq s(a, b) \geq 1$ or $0 \geq s(a, b) \geq 1$
2. $s(a, b) = 1$ iff $a = b$
3. $s(a, b) = s(b, a)$ (symmetry)

Examples of distance functions for continuous data in $\mathbb{R}^d$

- The most common is the **Minkownski distance** family; typically Euclidean ($p = 2$) or Manhattan ($p = 1$):

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_p = \left(\sum_{j=1}^{d} |a_j - b_j|^p \right)^{\frac{1}{p}}$$

- Another choice is to use the **Mahalanobis distance** if we want to take into account covariances Σ among features:

$$d(\mathbf{a}, \mathbf{b}) = (\mathbf{a} - \mathbf{b})^T \Sigma^{-1} (\mathbf{a} - \mathbf{b})$$

for high dimension

Examples of similarity functions for continuous data in $\mathbb{R}^d$

- **Cosine similarity**

$$s(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2}$$

- The **Pearson correlation measure** is similar to cosine similarity but centers data (μ is the **mean** of all training points):

$$s(\mathbf{a}, \mathbf{b}) = \frac{(\mathbf{a} - \mu)^T (\mathbf{b} - \mu)}{\|\mathbf{a} - \mu\|_2 \|\mathbf{b} - \mu\|_2}$$

Examples of similarities for binary data

- **Hamming distance**: proportion of common "bits"
- **Jaccard coefficient**

$$s(\mathbf{a}, \mathbf{b}) = \frac{\sum_j [a_j = b_j = 1]}{\sum_j [a_j = 1 \text{ or } b_j = 1]}$$

Example:

$$\mathbf{a} = + - - + - - - - + + - - - -$$
$$\mathbf{b} = + + - - - - - + + - - - - +$$

Here, $s_{Jaccard}(\mathbf{a}, \mathbf{b}) = \frac{2}{7} \approx 0.3$ and $s_{Hamming}(\mathbf{a}, \mathbf{b}) = \frac{9}{14} \approx 0.6$.

Choosing k hyperparameter

Nearest neighbor is very sensitive to different values of k

- if k is too **low**, then we can easilily overfit
- if k is too **large**, then we can underfit

So, it is important to select an appropriate k, which will depend on the dataset. Typically we use **cross-validation** or other resampling methods.

k can be seen as a hyper-parameter that trades-off bias and variance of the resulting classifier.

On combining labels to make predictions

For **classification**:

- Majority vote (ties broken randomly)
- Distance-weighted vote: choose weights that are higher for closer points
  - e.g. inverse of the distance or inverse of the squared distance

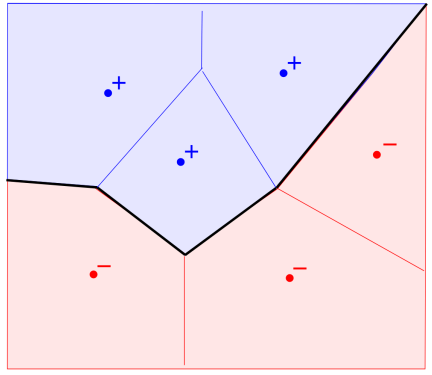
For **regression**:

- Use the **average** of targets of k nearest neighbors
- Use **weighted average** of targets of k nearest neighbors
  - e.g. use inverse distance as weight, or squared of inverse distance
  - *kernel smoothing*

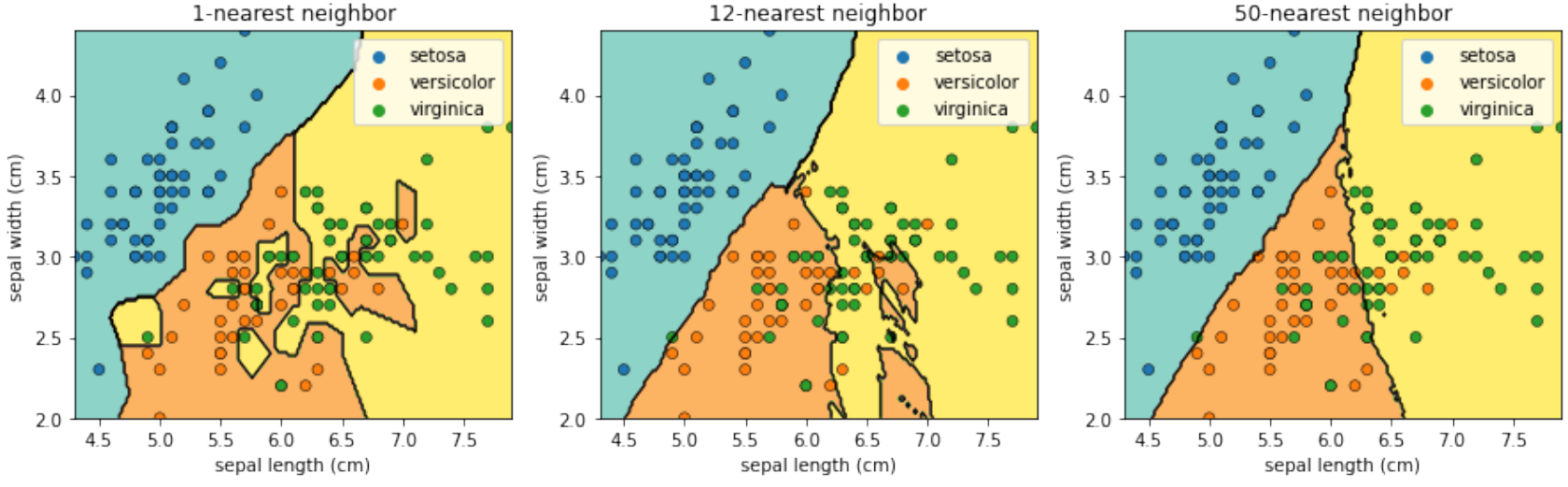
Once we use weights for the prediction, we can relax the constraint of using only k examples and use the whole training set instead.

Decision boundaries for nearest neighbors classifier

In 1-nearest neighbor, the decision regions correspond to the union of each example's *Voronoi* cell, with appropriate "class"



The decision boundaries and regions are highly non-linear, but get smoother as we increase k



Considerations with k-NN

- Making prediction can be **slow** especially if we have a large training set
  - data structures like *kd* trees used to speed up neighbor retrieval, only good for moderate nr. of features
  - use *prototypes*
  - use approximate neighbors; e.g. with *locality sensitive hashing*
- Prone to **overfitting**
  - remove "noisy" examples from training (e.g. examples with nearest neighbor of different class)
  - use *prototypes*
  - set k to appropriate value
- Suffers from **curse of dimensionality**: as dimensionality increases, everything seems to be close
- Suffers from presence of **irrelevant** features
- **Standarization** of features is crucial to avoid domination of features with larger (absolute) values

Comparison with toy datasets