k-NN

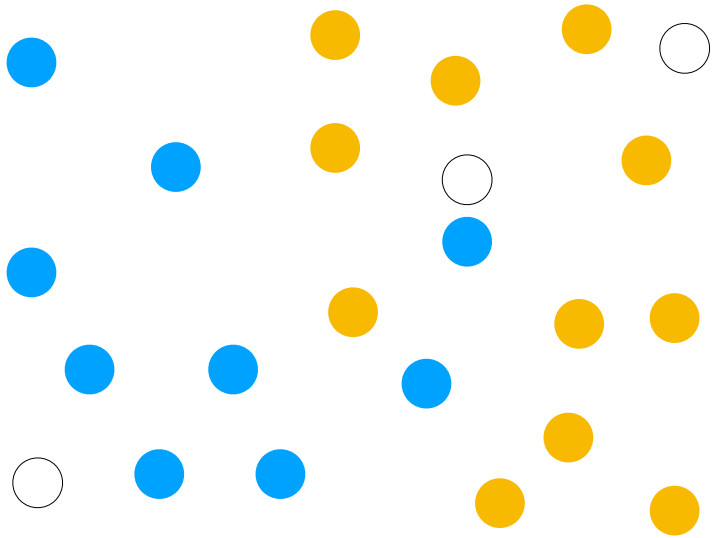
CSC 461: Machine Learning

Fall 2022

Prof. Marco Alvarez
University of Rhode Island

Instance-based learning

- Class of learning methods
  - ✓ also called **lazy learning**
- No need to learn any explicit hypothesis
- **Training**
  - ✓ trivial, just need to store instances
- **Inference**
  - ✓ time consuming, this is where computation happens



Nearest neighbor classification

- Training examples are vectors with a class label

$$\mathbf{x_i} \in \mathbb{R}^d \qquad y_i \in \{1, \ldots, C\}$$

- Learning
  - ✓ **store** all training examples
- Prediction
  - ✓ predict the label of the new example as the label of its **closest point** in the training set

what is the computational complexity of predicting a new label?

k-Nearest Neighbors

k-nearest neighbors

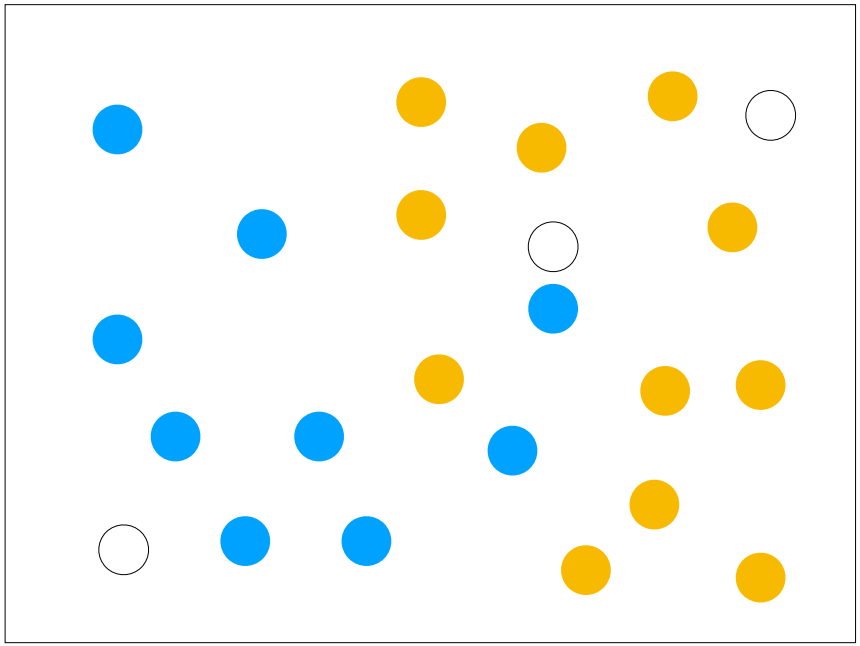
- Prediction for a new point x
  - ✓ recover a subset S_x (k nearest neighbors to x)

$$S_x \subseteq \mathcal{D} \text{ s.t. } |S_x| = k$$

$$\forall (\mathbf{x}', y') \in \mathcal{D} \setminus S_x$$

$$D(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_x} D(\mathbf{x}, \mathbf{x}'')$$

  - ✓ take a **majority vote (mode)** (classification)
  - ✓ calculate the **average** (regression)



Distance metrics

$$D(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{d} |a_i - b_i|^p \right)^{1/p}$$

minkowski

$$\mathbf{a} \in \mathbb{R}^d, \mathbf{b} \in \mathbb{R}^d$$

$p = 1$? manhattan

$p = 2$? euclidean

$p = \infty$? chebyshev $\max_i(|a_i - b_i|)$

could also use other distances (for non-euclidean spaces)

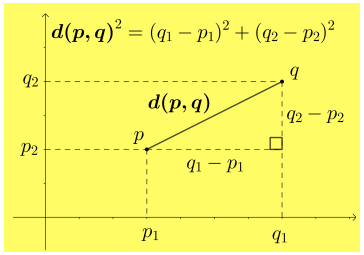
More on the euclidean distance

$$d(\mathbf{p},\mathbf{q}) = \left(\sum_{i=1}^{d} |p_i - q_i|^2\right)^{1/2}$$

$$= \sqrt{\sum_{i=1}^{d} |p_i - q_i|^2}$$

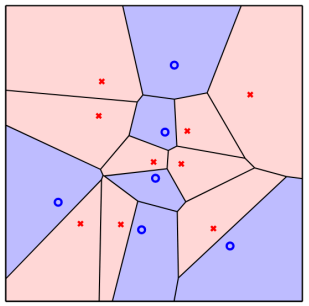
$$= \|\mathbf{p}-\mathbf{q}\|_2$$

$$= \sqrt{(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q})}$$



What is the decision boundary?

- Is k-NN building an explicit decision boundary?
  - ✓ not really, but it can be inferred



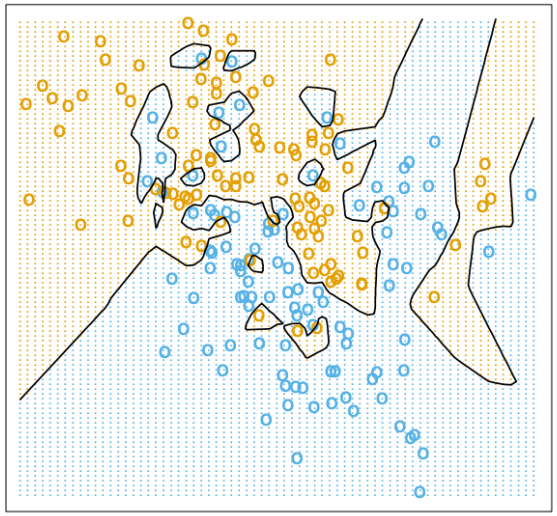
Nearest neighbor Voronoi tesselation

is the diagram sensitive to k?

is the diagram sensitive to the distance function?

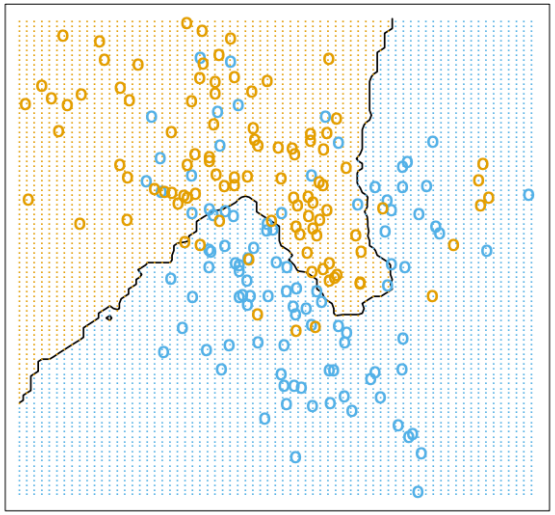
http://www.cs.rpi.edu/~magdon/courses/LFD-Slides/SlidesLect16.pdf

1-Nearest Neighbor Classifier



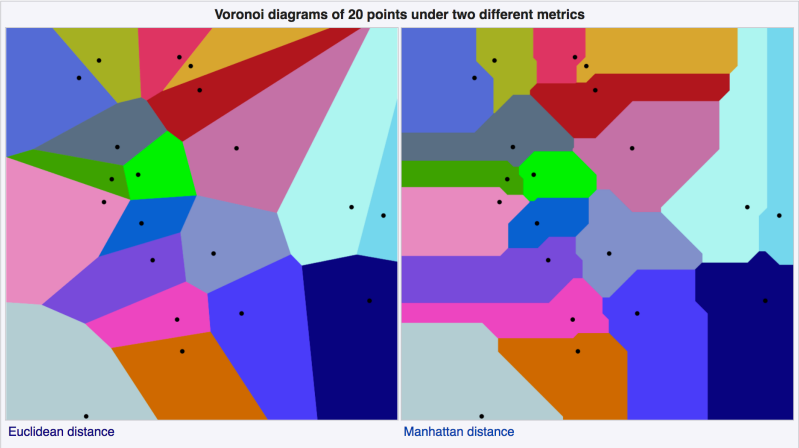
Elements of Statistical Learning (2nd Ed.) c Hastie, Tibshirani & Friedman 2009 Chap 2

15-Nearest Neighbor Classifier



Elements of Statistical Learning (2nd Ed.) c Hastie, Tibshirani & Friedman 2009 Chap 2

Euclidean vs Manhattan



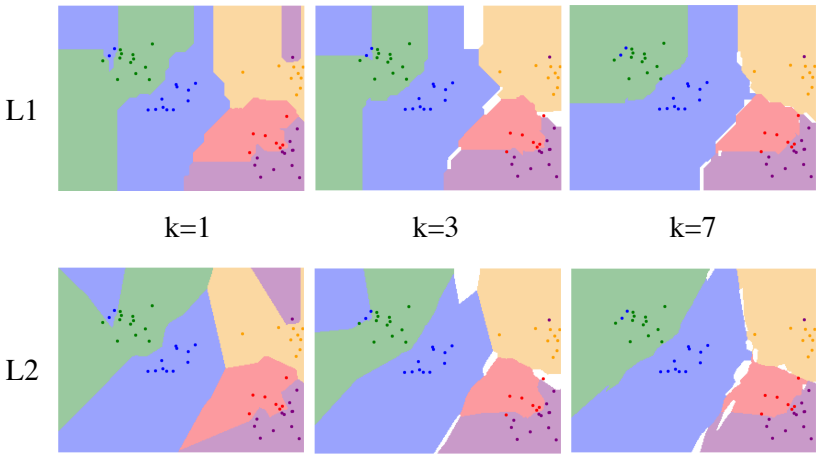
https://en.wikipedia.org/wiki/Voronoi_diagram

Hyperparameters

- The number of neighbors **k**
  - ✓ too small, sensitive to noise
  - ✓ too large, neighborhood includes points from other classes
- **Distance** function
- How to find a value that may generalize better?

  use Cross-Validation for parameter tuning

Hyperparameters

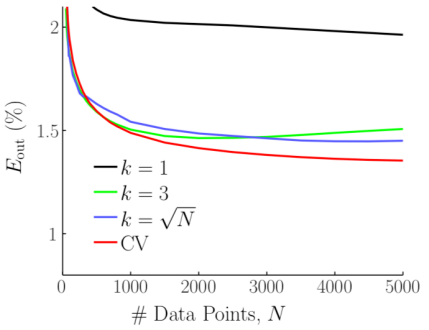


http://vision.stanford.edu/teaching/cs231n-demos/knn

Choosing k

1. $k = 3$.
2. $k = \lceil \sqrt{N} \rceil$.
3. Validation or cross validation:

   k-NN rule hypotheses g_k^- constructed on training set, tested on validation set, and best k is picked.

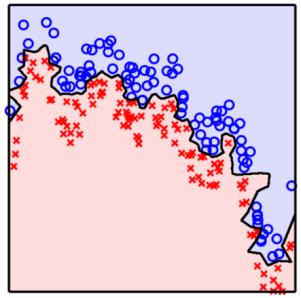


http://www.cs.rpi.edu/~magdon/courses/LFD-Slides/SlidesLect16.pdf

Additional Remarks

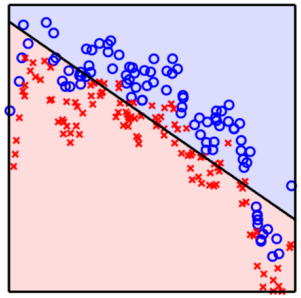
kNN vs linear models

NN-rule



no parameters
expressive/flexible
$g(\mathbf{x})$ needs data
generic, can model anything

Linear Model



$(d+1)$ parameters
rigid, always linear
$g(\mathbf{x})$ needs only weights
specialized

http://www.cs.rpi.edu/~magdon/courses/LFD-Slides/SlidesLect16.pdf

Normalization

- k-NN can be sensitive to feature ranges
  - ✓ e.g., euclidean distance

$$D(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{i=1}^{d}(a_i - b_i)^2}$$

- Features can be preprocessed
  - ✓ e.g., zero mean and unit variance

$$x_j' = \frac{x_j - \mu_j}{\sigma_j}$$

For certain datasets, the scale may be important

Normalization

- Must **calculate parameters using training data** then **transform** the test data

```
from sklearn import preprocessing
import numpy as np

# define the scaler object
scaler = preprocessing.StandardScaler()

# fit training data
scaler.fit(X_train)

# apply to training and test data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

`sklearn.preprocessing`: Preprocessing and Normalization

The `sklearn.preprocessing` module includes scaling, centering, normalization, binarization methods.

User guide: See the Preprocessing data section for further details.

`preprocessing.Binarizer`(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
`preprocessing.FunctionTransformer`([func, ...])	Constructs a transformer from an arbitrary callable.
`preprocessing.KBinsDiscretizer`([n_bins, ...])	Bin continuous data into intervals.
`preprocessing.KernelCenterer`()	Center an arbitrary kernel matrix K.
`preprocessing.LabelBinarizer`(*[, neg_label, ...])	Binarize labels in a one-vs-all fashion.
`preprocessing.LabelEncoder`()	Encode target labels with value between 0 and n_classes-1.
`preprocessing.MultiLabelBinarizer`(*[, ...])	Transform between iterable of iterables and a multilabel format.
`preprocessing.MaxAbsScaler`(*[, copy])	Scale each feature by its maximum absolute value.
`preprocessing.MinMaxScaler`([feature_range, ...])	Transform features by scaling each feature to a given range.
`preprocessing.Normalizer`([norm, copy])	Normalize samples individually to unit norm.
`preprocessing.OneHotEncoder`(*[, categories, ...])	Encode categorical features as a one-hot numeric array.
`preprocessing.OrdinalEncoder`(*[, ...])	Encode categorical features as an integer array.
`preprocessing.PolynomialFeatures`([degree, ...])	Generate polynomial and interaction features.
`preprocessing.PowerTransformer`([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.
`preprocessing.QuantileTransformer`(*[, ...])	Transform features using quantiles information.
`preprocessing.RobustScaler`(*[, ...])	Scale features using statistics that are robust to outliers.
`preprocessing.SplineTransformer`([n_knots, ...])	Generate univariate B-spline bases for features.
`preprocessing.StandardScaler`(*[, copy, ...])	Standardize features by removing the mean and scaling to unit variance.

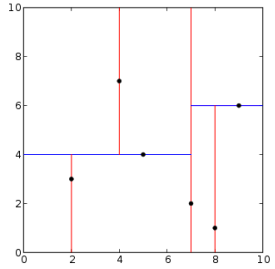
https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

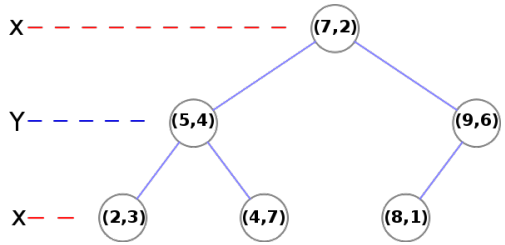
k-NN Regression

- Prediction
  - ✓ instead of taking a majority vote (as in classification)
  - ✓ return the **average output of the k nearest neighbors**

Computational cost

- Can use advanced algorithms and data structures
  - ✓ e.g., kd-trees





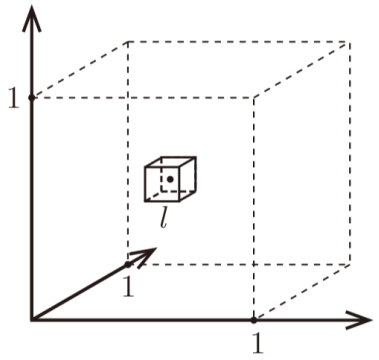
Weighted k-NN

- By default (vanilla k-NN) all neighbors have equal weight

- Can weight the votes according to their distance
  - ✓ for example:

$$w = \frac{1}{d^2}$$

Curse of dimensionality



Assume **n** points are **uniformly distributed** and we are looking for the **k** nearest neighbors in **d** dimensions

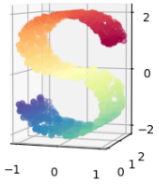
Now think about the volume of the minimal enclosing box for the set of **k** nearest neighbors

$$l^d \approx \frac{k}{n}$$

Solve for **l** and play with different values for **d**

Why k-nn might work?

- Data is not always uniformly distributed over **d** dimensions
  - ✓ **P** may be lying on a low-dimensional subspace (low intrinsic dimensionality)
  - ✓ **P** may be on an underlying manifold
    - ✓ local distances (such as nearest neighbors) work better than global distances



Summary

- No assumptions about **P**
  - ✓ adapts to data density
- Cost of learning is zero
  - ✓ unless a **kd-tree** or other data structures are used
- Need to normalize/scale the data
  - ✓ features with larger ranges dominate distances (automatically becoming more important)
  - ✓ be careful: sometimes range matters

Summary

- Irrelevant or correlated attributes add noise to distance
  - ✓ may want to drop them
- Prediction is computationally expensive
  - ✓ can use **kd-trees** or **hashing techniques** like Locality Sensitive Hashing (LSH)
- Curse of dimensionality
  - ✓ data required to generalize grows exponentially with dimensionality
  - ✓ distances less meaningful in higher dimensions