

k-Nearest Neighbors

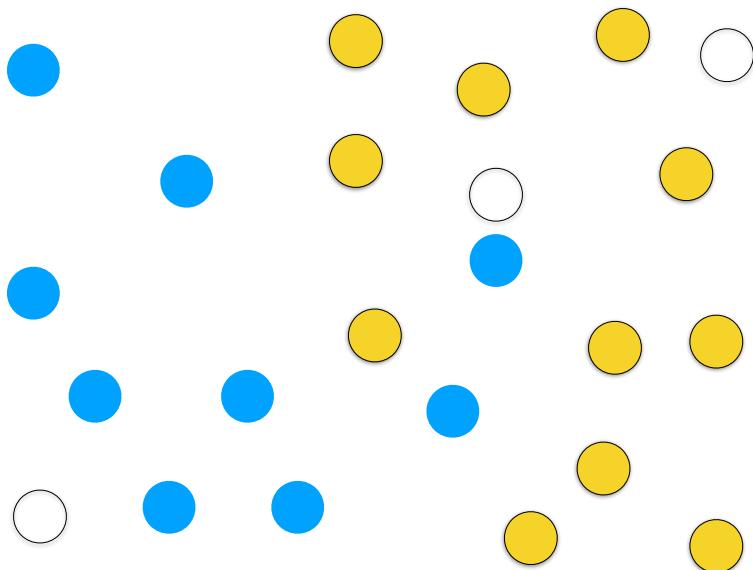
Prof. Marco Alvarez, Computer Science
University of Rhode Island

Instance-based learning

- ▶ Class of learning methods
 - also called **lazy learning**
- ▶ No **explicit hypothesis** is learned during “training”
- ▶ Training/learning phase
 - store instances
- ▶ Inference phase
 - computationally intensive

Nearest neighbor classification

- ▶ Training examples
 - $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$
 - typically $x_i \in \mathbb{R}^d$ but can work with any data type, given an appropriate distance function
- ▶ Learning
 - **store** all training examples, no explicit model is built
- ▶ Prediction
 - for a new example, find the closest point in the training set
 - **predict** the label of the new example as the label of the closest point



k-Nearest Neighbors

k-Nearest Neighbors

► Prediction for a new example x

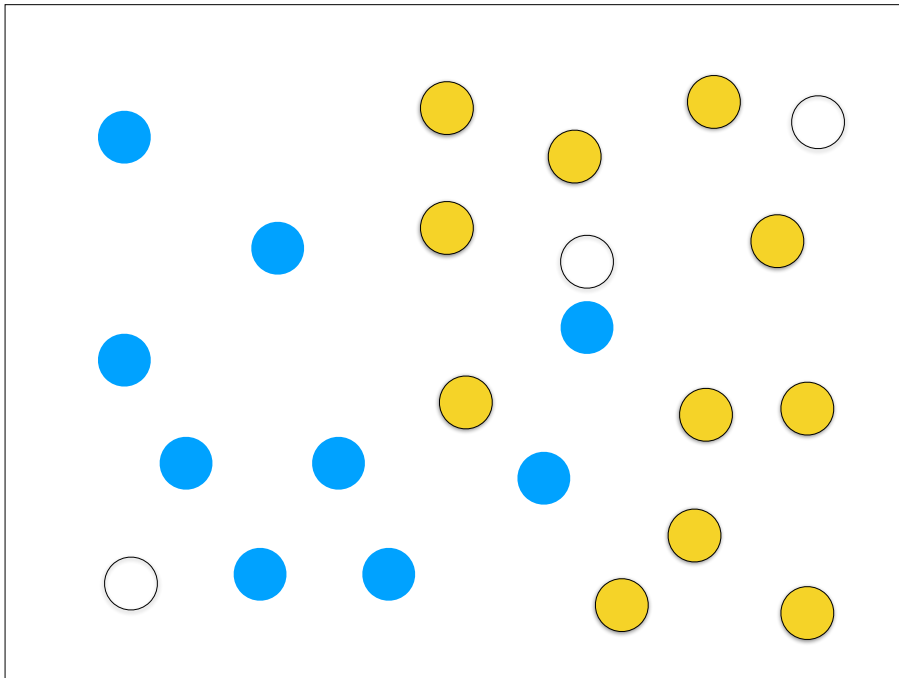
- recover a subset S_x (**k nearest neighbors to x**)

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

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

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

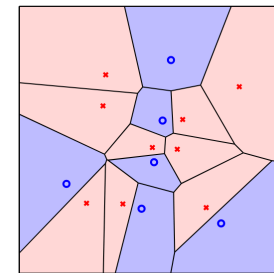
- **classification**: predict the majority label in S_x
- **regression**: predict the average of labels in S_x



Decision boundary

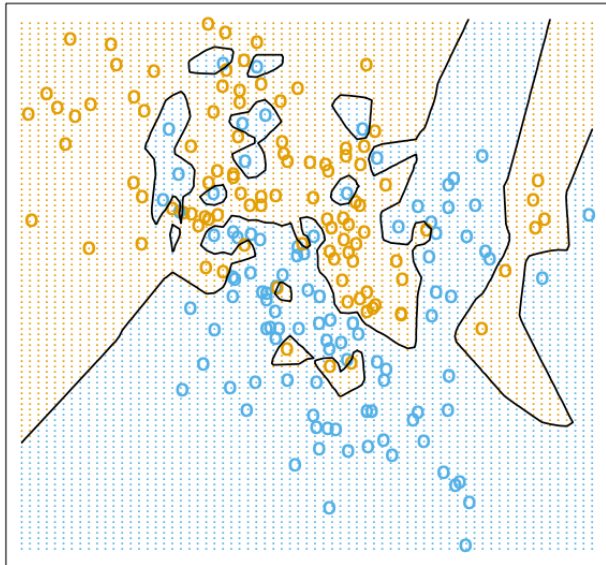
► Is k-NN building an explicit decision boundary?

- k-NN does not explicitly construct a decision boundary
- decision boundary is implicitly defined by the training data



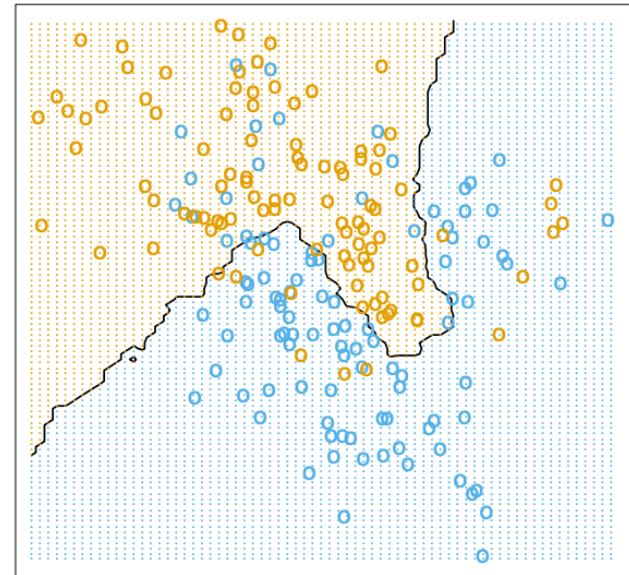
Nearest neighbor Voronoi tessellation

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

Iris dataset (example)

```
In [25]: from sklearn.datasets import load_iris
```

```
In [26]: data = load_iris()
```

```
In [30]: data.data.shape
```

```
Out[30]: (150, 4)
```

sepal length, sepal width,
petal length, petal width

```
In [27]: idx = np.random.choice(np.arange(len(data.target)), 10)
```

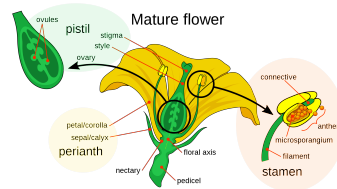
```
In [28]: data.target[idx]
```

```
Out[28]: array([1, 0, 0, 1, 1, 1, 2, 0, 0, 2])
```

```
In [29]: data.data[idx]
```

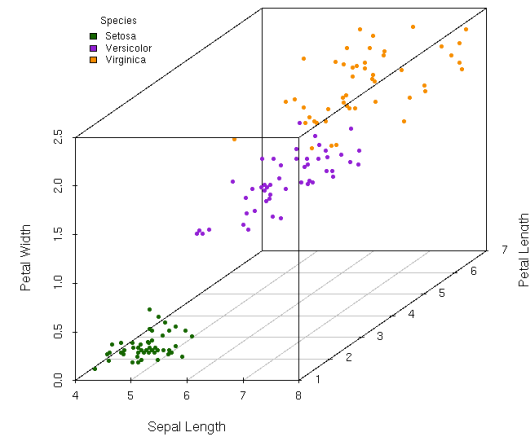
```
Out[29]:
```

```
array([[6.6, 3. , 4.4, 1.4],
       [5. , 3.3, 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [5.7, 2.8, 4.1, 1.3],
       [6.1, 2.9, 4.7, 1.4],
       [6.3, 2.5, 4.9, 1.5],
       [6.3, 2.7, 4.9, 1.8],
       [5.4, 3.4, 1.7, 0.2],
       [4.6, 3.6, 1. , 0.2],
       [6.3, 2.9, 5.6, 1.8]])
```



Iris dataset (example)

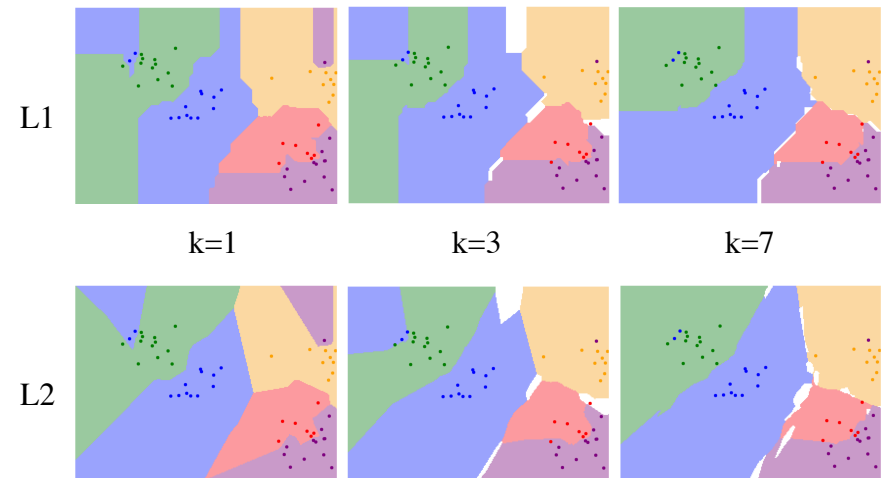
3-D Scatterplot of Iris Data



Hyperparameters

- ▶ The number of neighbors (K)
 - if too small, sensitive to noise
 - if too large, may include points from other classes
- ▶ Distance function
 - e.g., euclidean, manhattan, etc.
- ▶ How to find a value that may **generalize** better?
 - try multiple values and evaluate performance using an evaluation metric

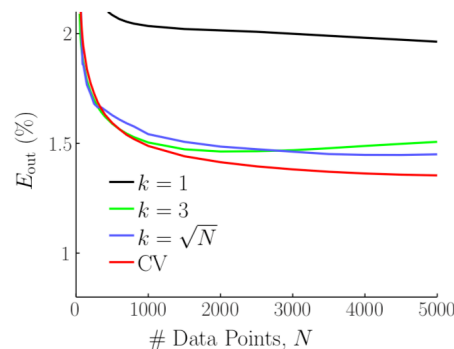
Different hyperparameters



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

Using a validation set

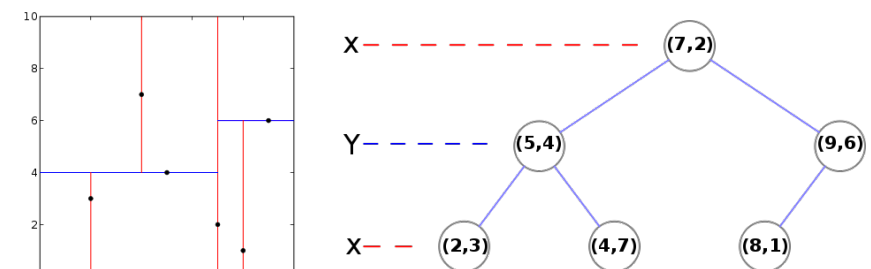
1. $k = 3$.
2. $k = \lceil \sqrt{N} \rceil$.
3. Validation or cross validation:
 k -NN rule hypotheses g_k constructed on training set, tested on validation set, and best k is picked.



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

Computational cost

- ▶ Calculating the nearest neighbors is expensive
 - does not scale well to large and/or high-dimensional datasets
- ▶ Can use advanced data structures and algorithms
 - e.g., kd-trees, ball trees



Additional remarks

Data normalization/scaling

► k-NN is sensitive to feature scales

- e.g., using the euclidean distance

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

► Solution: preprocess the data

- e.g., standardize to zero mean and unit variance

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

apply this formula
to all columns
(features)

For certain datasets, the scale may be important

Data normalization

► Be careful ...

- must **calculate normalization parameters using training data** then apply same transformation to validation/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.

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

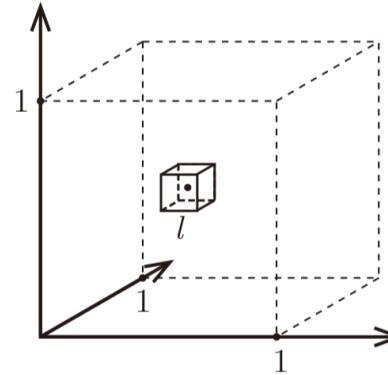
<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing>

kNN regression

► Prediction

- find the k nearest neighbors
- return the average label of the k nearest neighbors

Curse of dimensionality



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

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

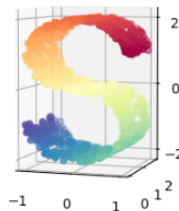
Assume n points are **uniformly distributed** and we are looking for the k nearest neighbors in d dimensions

Solve for l and play with different values for d

Why k-NN might work?

► Data is not always uniformly distributed over d dimensions

- data distribution P may be lying on a low-dimensional subspace (low intrinsic dimensionality) or on an underlying manifold
- local distances (such as nearest neighbors) work better than global distances



Summary

- No assumptions about data distribution P
 - adapts to data density
- Cost of training/learning is zero
 - unless a **kd-tree** or other data structures are used
- Recommended 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 or use dimensionality reduction
- 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