k-Nearest Neighbors

CSC 461: Machine Learning

Fall 2021

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** is trivial (just store instances)
- **Predicting** new labels is where computation happens

what is the computational complexity of training?

Nearest Neighbor Classification

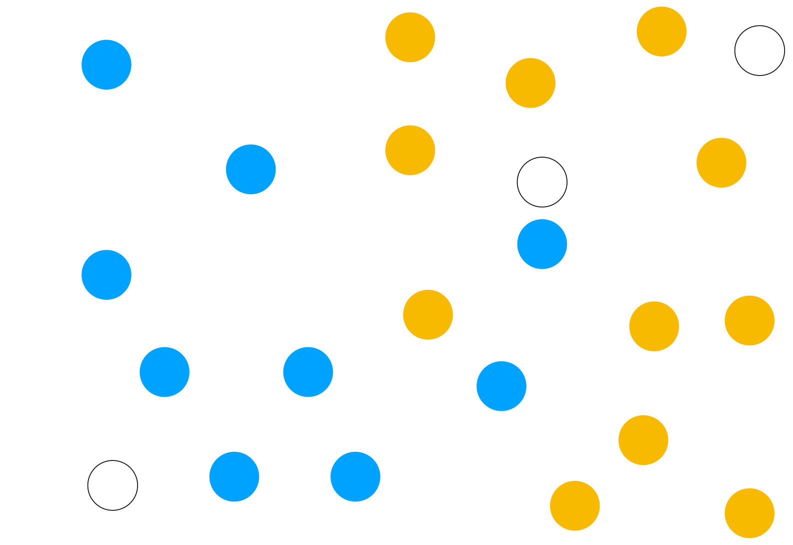
Nearest neighbor classification

- Training examples are vectors with a class label

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

- 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 test point x
  - ✓ recover a subset Sx (k nearest neighbors to x)

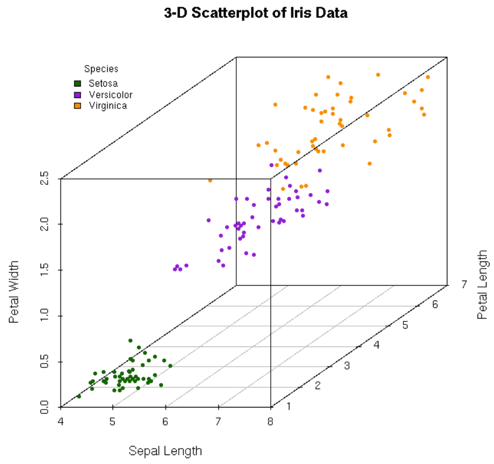
$$S_x \subseteq \mathcal{D} \text{ s.t. } |S_x| = k$$

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

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

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

Classification example



https://spin.atomicobject.com/2013/05/06/k-nearest-neighbor-racket/

Distance

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

minkowski

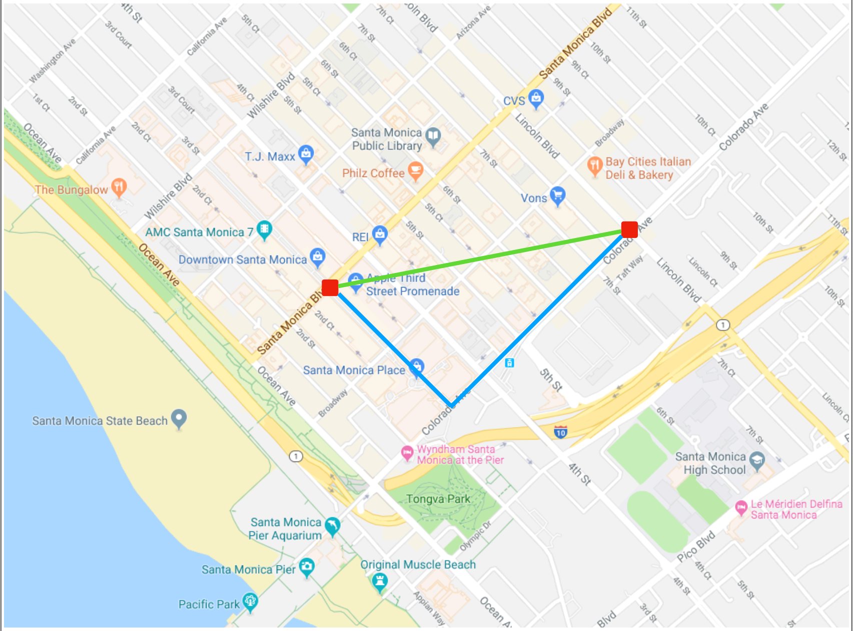
$$a \in \mathbb{R}^d, b \in \mathbb{R}^d$$

$p = 1$? manhattan

$p = 2$? euclidean

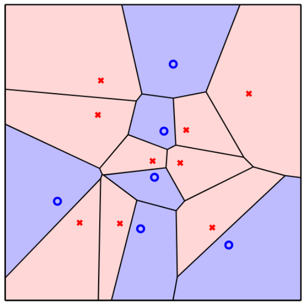
$p = \infty$? chebyshev

could also use other distances (for different input spaces)



What is the decision boundary?

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

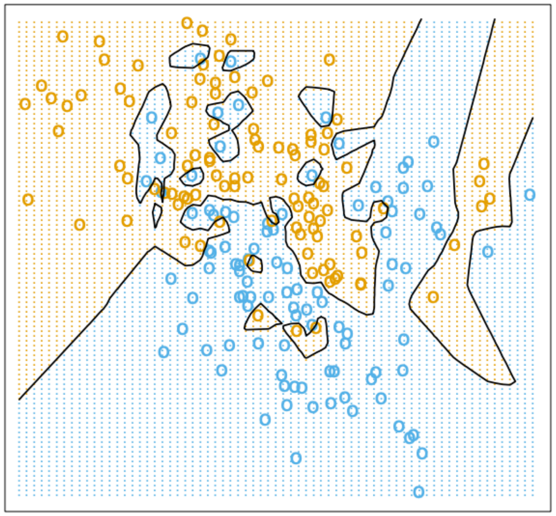


Nearest neighbor Voronoi tesselation

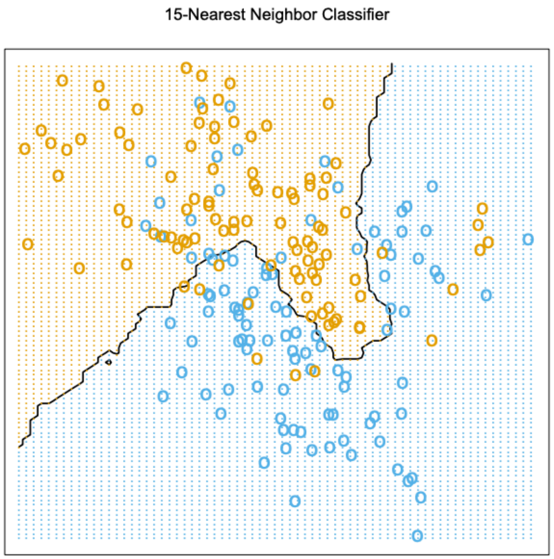
is the diagram sensitive to k? what about the distance function?

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

1-Nearest Neighbor Classifier

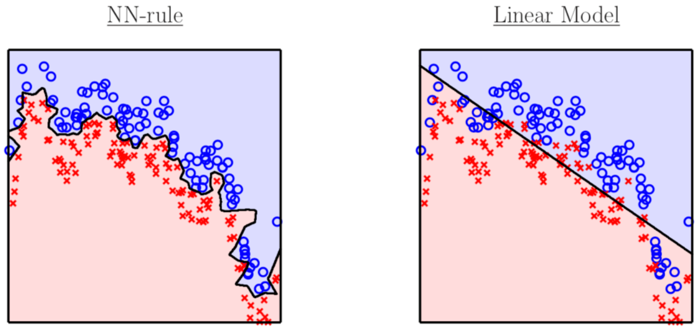


Elements of Statistical Learning (2nd Ed.) c Hastie, Tibshirani & Friedman 2009 Chap 2



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

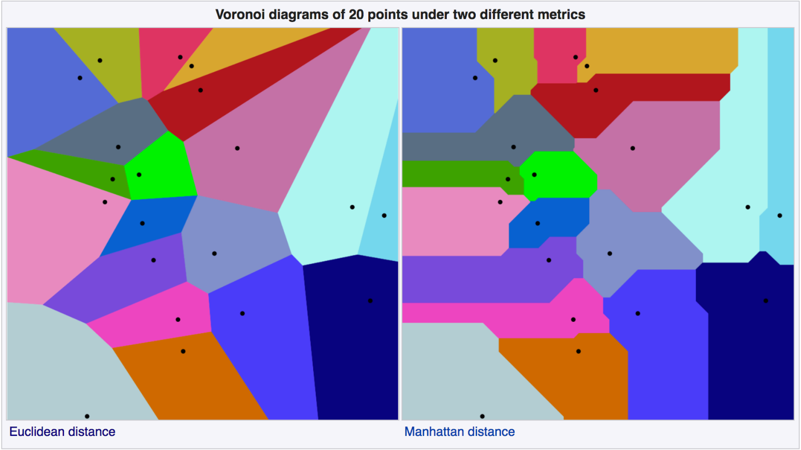
kNN vs linear models



NN-rule	Linear Model
no parameters	$(d+1)$ parameters
expressive/flexible	rigid, always linear
$g(\mathbf{x})$ needs data	$g(\mathbf{x})$ needs only weights
generic, can model anything	specialized

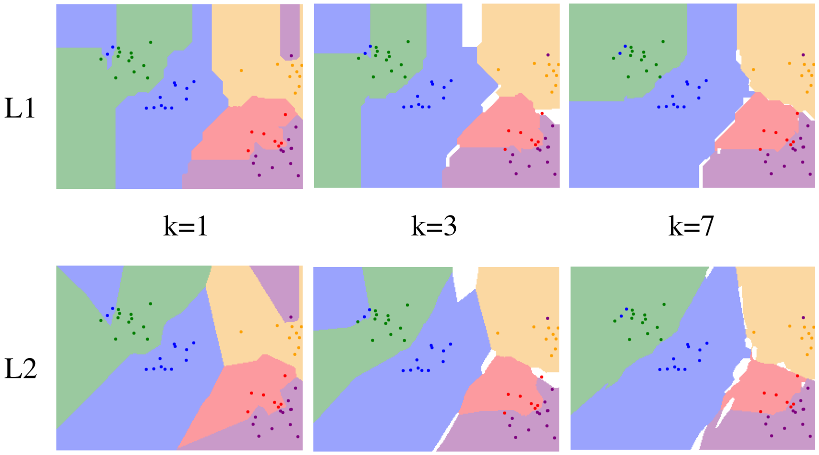
http://www.cs.rpi.edu/~magdon/courses/LFD-Slides/SlidesLect16.pdf

Euclidean vs Manhattan



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

Hyperparameters



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

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

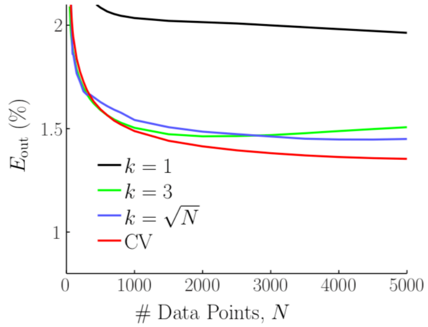
Choosing k

1. $k = 3$.

2. $k = \left\lceil \sqrt{N} \right\rceil$.

3. Validation or cross validation:

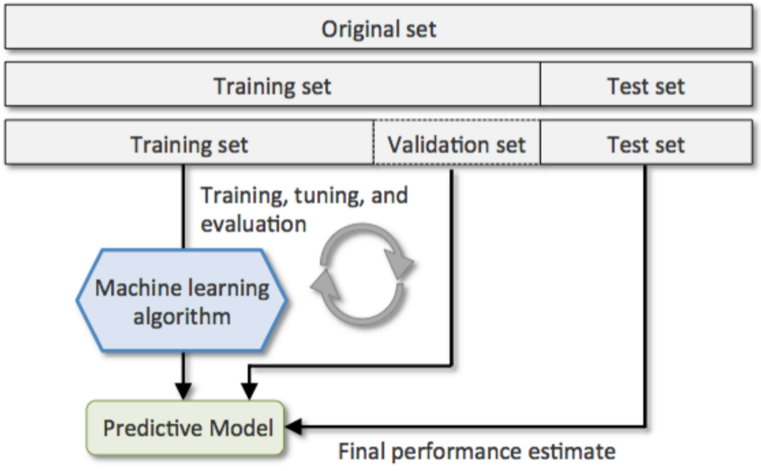
k-NN rule hypotheses $g_{\bar{k}}$ constructed on training set, tested on validation set, and best k is picked.



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

Additional Remarks

Train, Validation, and Test Sets



Original set

Training set | Test set

Training set | Validation set | Test set

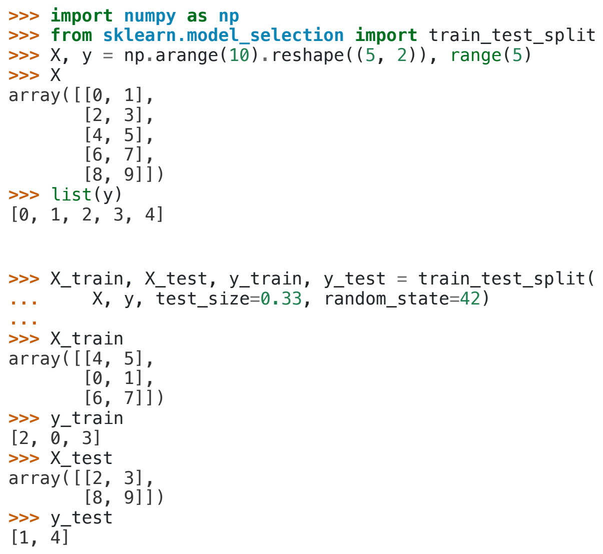
Training, tuning, and evaluation

Machine learning algorithm

Predictive Model

Final performance estimate

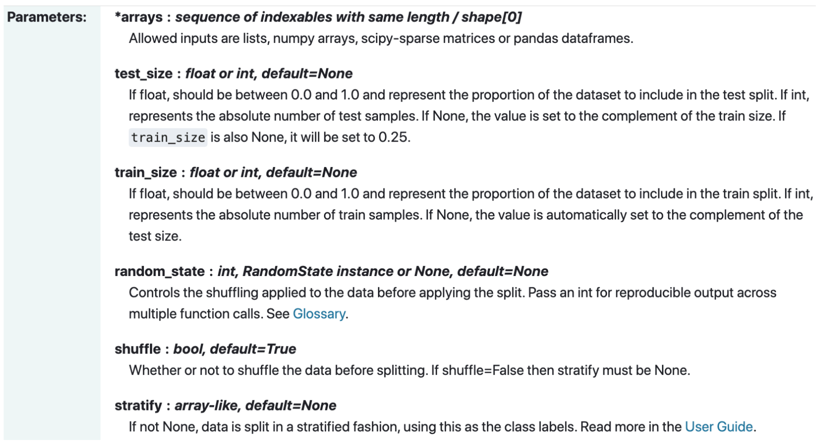
from INFO-4604: Applied Machine Learning, Fall 2017, Michael Paul, Univ. of Colorado



```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]


>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html



Parameters:

arrays** : ***sequence of indexables with same length / shape[0]
Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : ***float or int, default=None***
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : ***float or int, default=None***
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : ***int, RandomState instance or None, default=None***
Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : ***bool, default=True***
Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratify : ***array-like, default=None***
If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Normalization

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

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

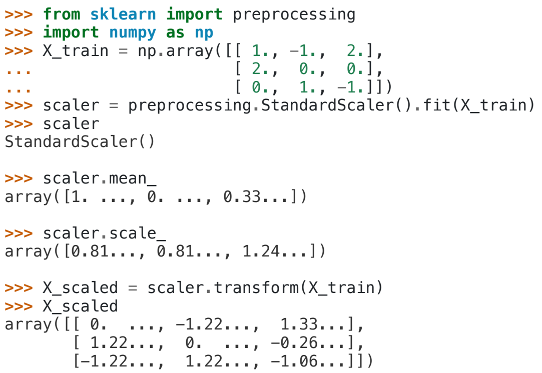
- Features can be preprocessed
  - ✓ zero mean and unit variance

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

For certain datasets, the scale may be important

Normalization

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

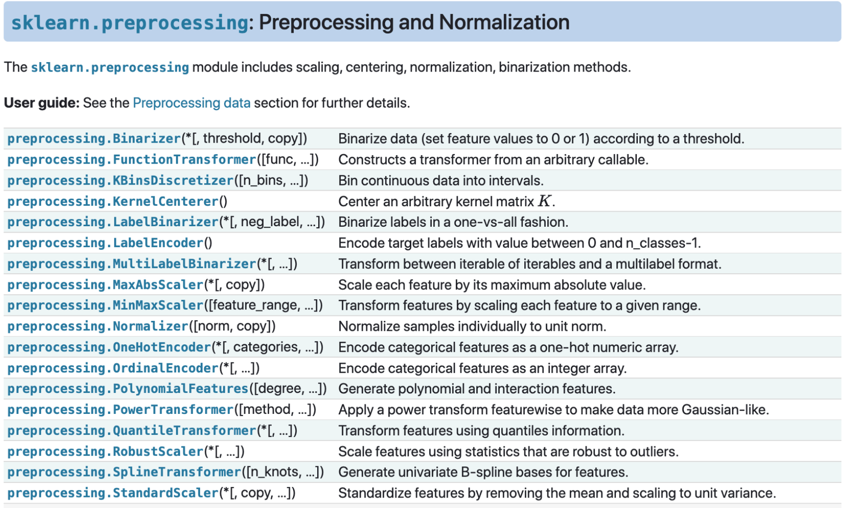


```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler()

>>> scaler.mean_
array([1. ..., 0. ..., 0.33...])

>>> scaler.scale_
array([0.81..., 0.81..., 1.24...])

>>> X_scaled = scaler.transform(X_train)
>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```



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

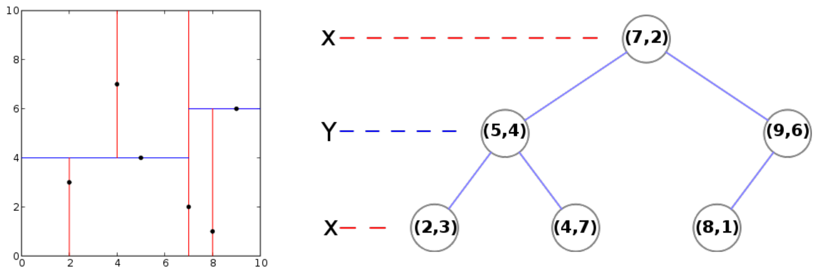
Irrelevant features

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



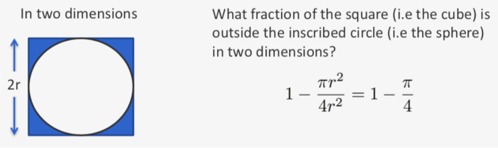
Weighted k-NN

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

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

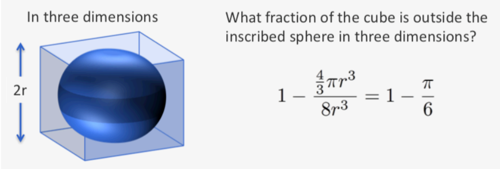
Curse of dimensionality

- What fraction of the points lie outside the sphere?


In two dimensions

What fraction of the square (i.e the cube) is outside the inscribed circle (i.e the sphere) in two dimensions?

$$1 - \frac{\pi r^2}{4r^2} = 1 - \frac{\pi}{4}$$


In three dimensions

What fraction of the cube is outside the inscribed sphere in three dimensions?

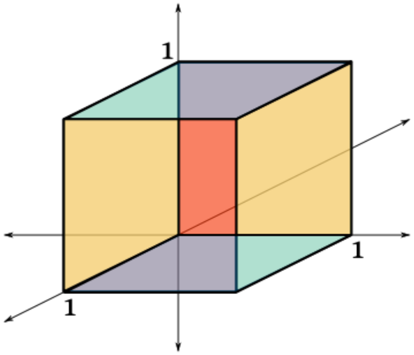
$$1 - \frac{\frac{4}{3}\pi r^3}{8r^3} = 1 - \frac{\pi}{6}$$

as dimensionality increases, this fraction approaches 1 !

distances do not behave the same way in high dimensions

https://svivek.com/teaching/machine-learning/lectures/slides/nearest-neighbors/nearest-neighbors.pdf

Curse of dimensionality



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

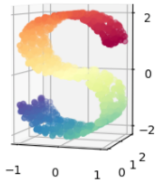
Now think about the volume of the minimal enclosing box for the set of **k** nearest neighbors

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

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

Why k-nn might work?

- Data is <u>not always uniformly distributed</u> 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