Instanced-based learning

Nishant Mehta

Lectures 22 and 23

Instance-based learning

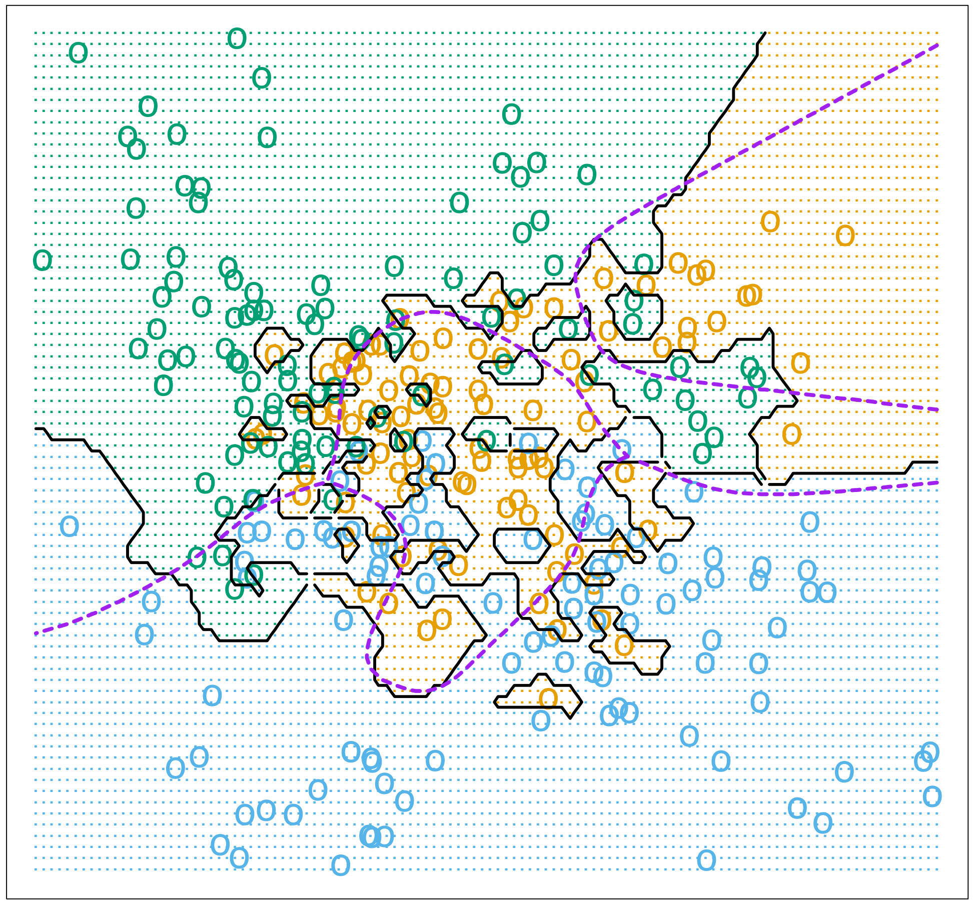
Can be used for either classification or regression

At training time: simply store the training data

At test time:

- For a test example, find neighboring examples in the training set
- Predict label by combining the labels of these neighboring examples

1-Nearest Neighbor



(Hastie et al., 2009), Figure 13.3

1-nearest neighbor classifier

Input examples in $\mathbb{R}^d$

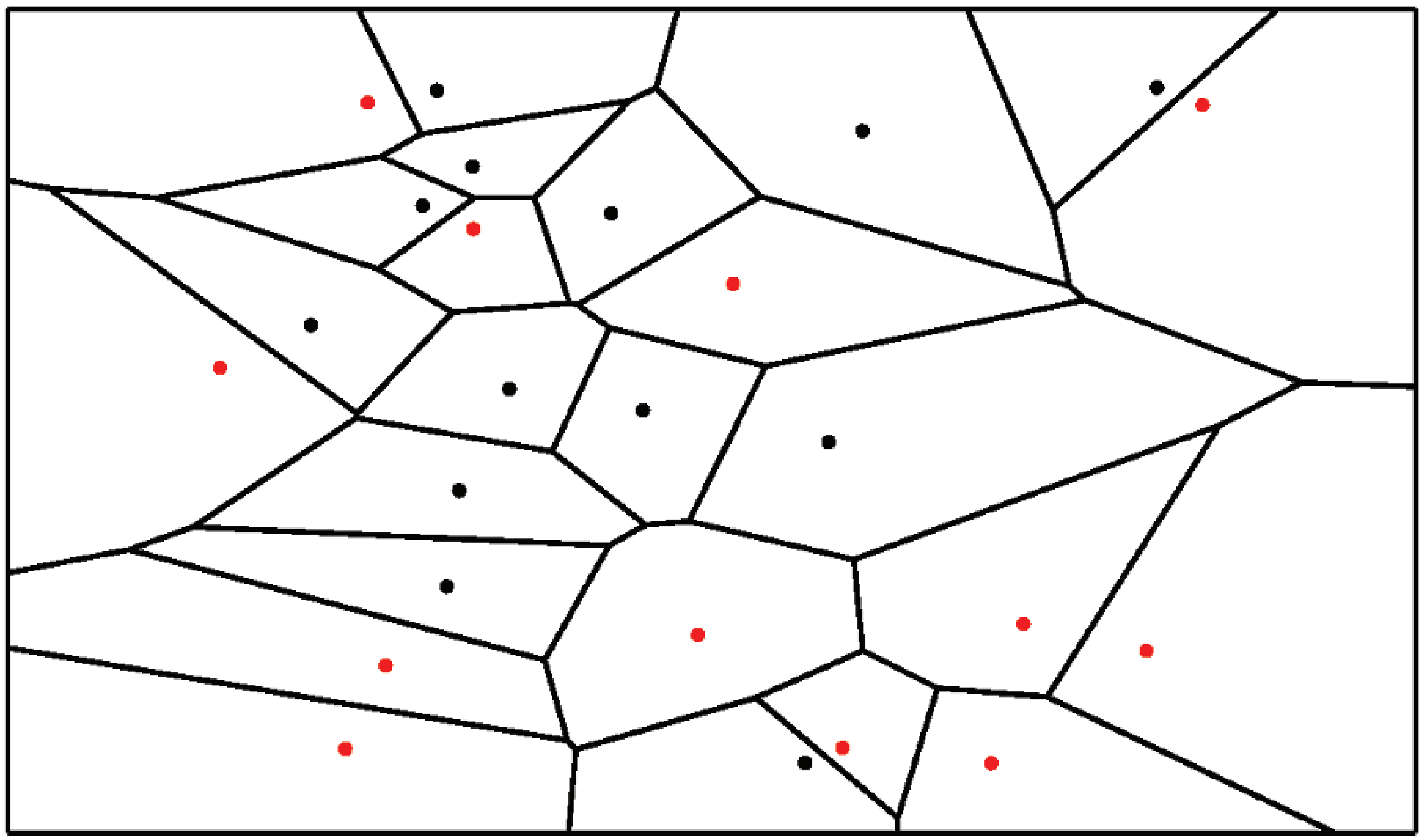
Euclidean distance $\|x_{\text{test}} - x\|$

For test example x_{test}, find nearest neighbor $x^{(1)} = \underset{x \in D}{\operatorname{argmin}}\, d(x_{\text{test}}, x)$

Input points in training set

Prediction? Nearest neighbor's label $\hat{h}(x) = y^{(1)}$

Voronoi diagram



(Murphy, 2012), Figure 1.14

Properties of the 1-NN classifier

Consider the setting of binary classification, where examples (X, Y) are jointly drawn from probability distribution P

Let R^* be the risk of the optimal rule:

$$R^* = \min_{h} \Pr_{(X,Y)\sim P} (h(X) \neq Y)$$

Bayes risk

min is taken over all possible rules mapping from input space to {0,1}

Asymptotically (as number of training examples $n \to \infty$), expected risk of 1-NN rule is at most $2R^*$.
So, expected risk of 1-NN rule is at most twice the Bayes risk.

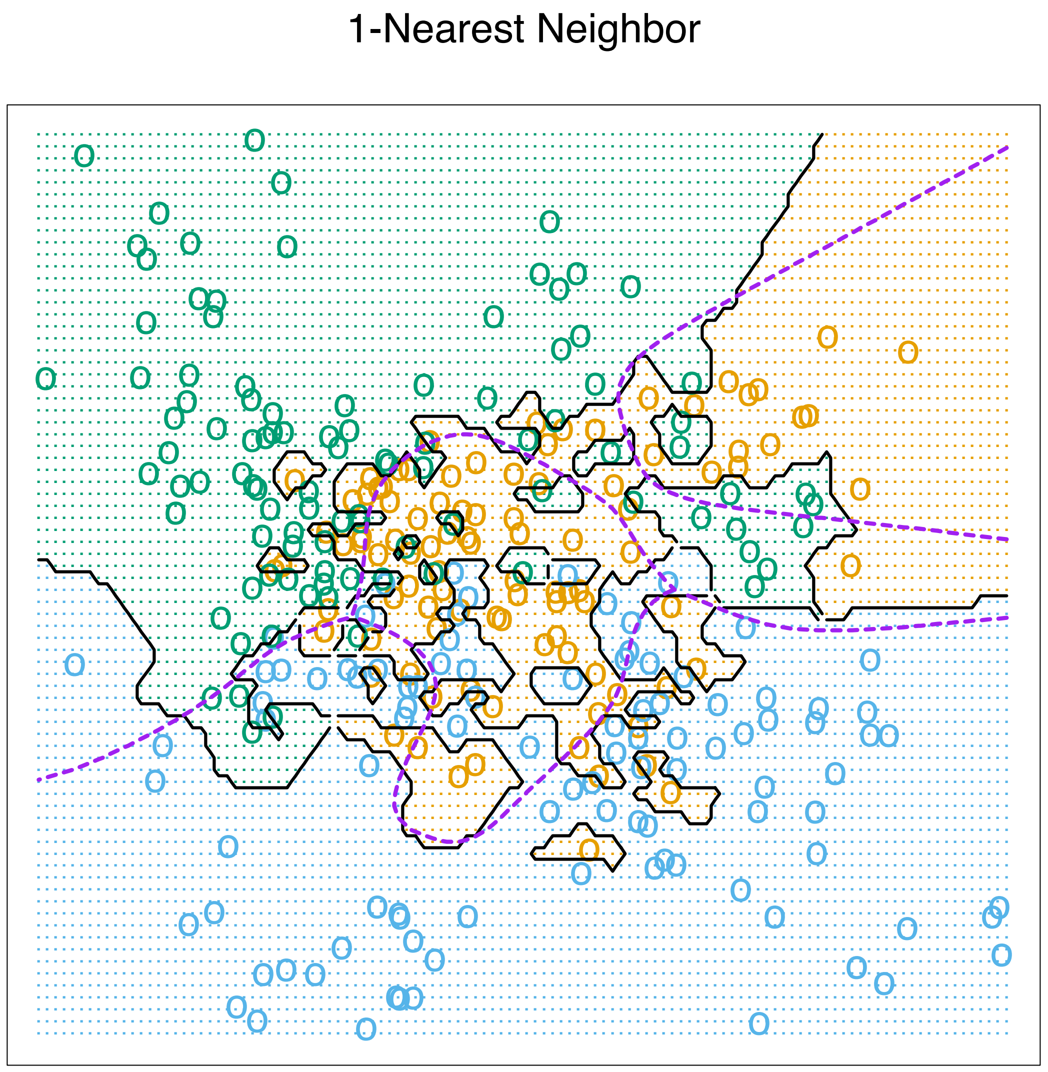
1-NN is "2-approx"

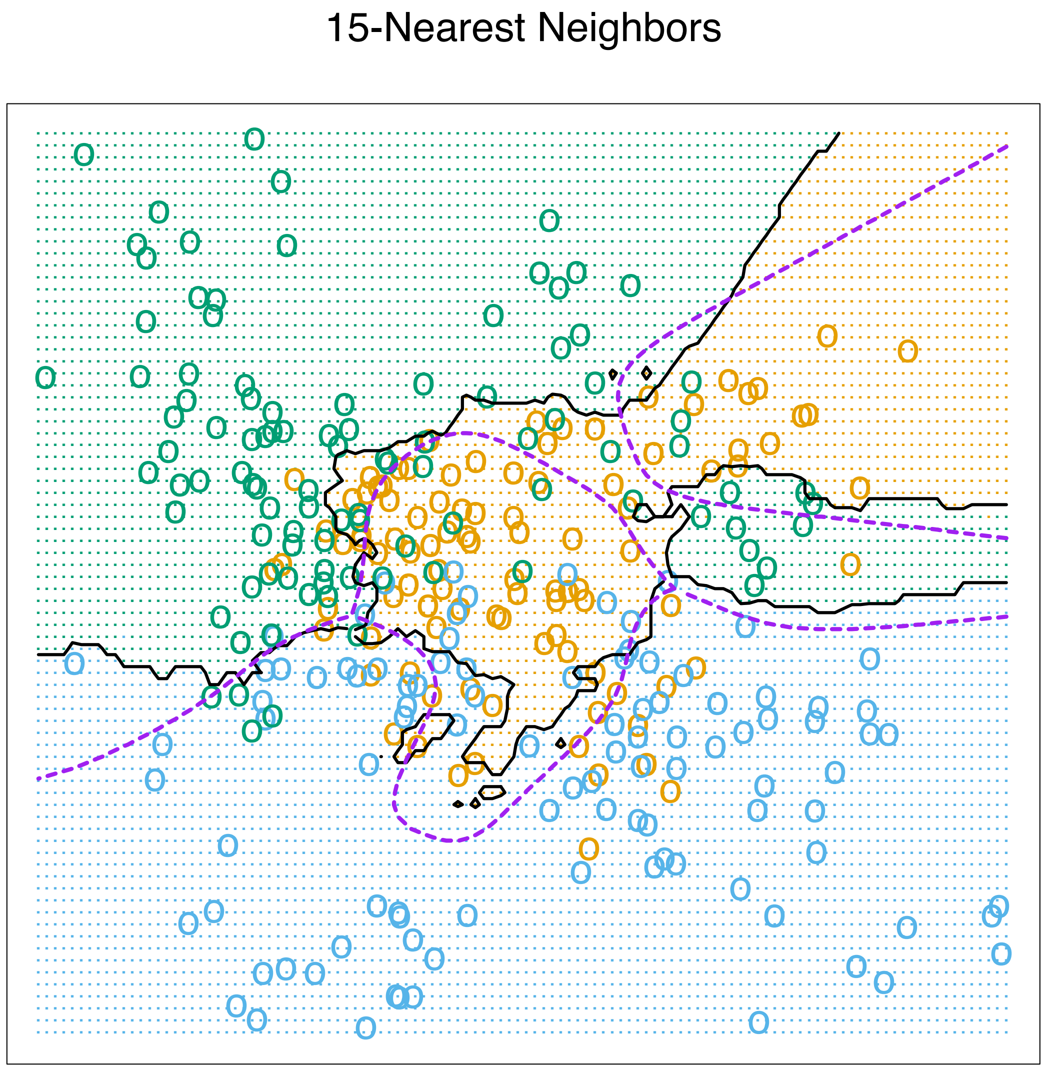
k-nearest neighbor classifier

For test example x_{test}, find k nearest neighbors $x^{(1)}, \ldots, x^{(k)}$

Prediction? Majority vote among the k nearest neighbors:

$$\hat{h}(x) = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{k} 1\left[y^{(i)} = y\right]$$





Computational concerns

For each test example, need to find the *k* nearest neighbors among the *n* training examples

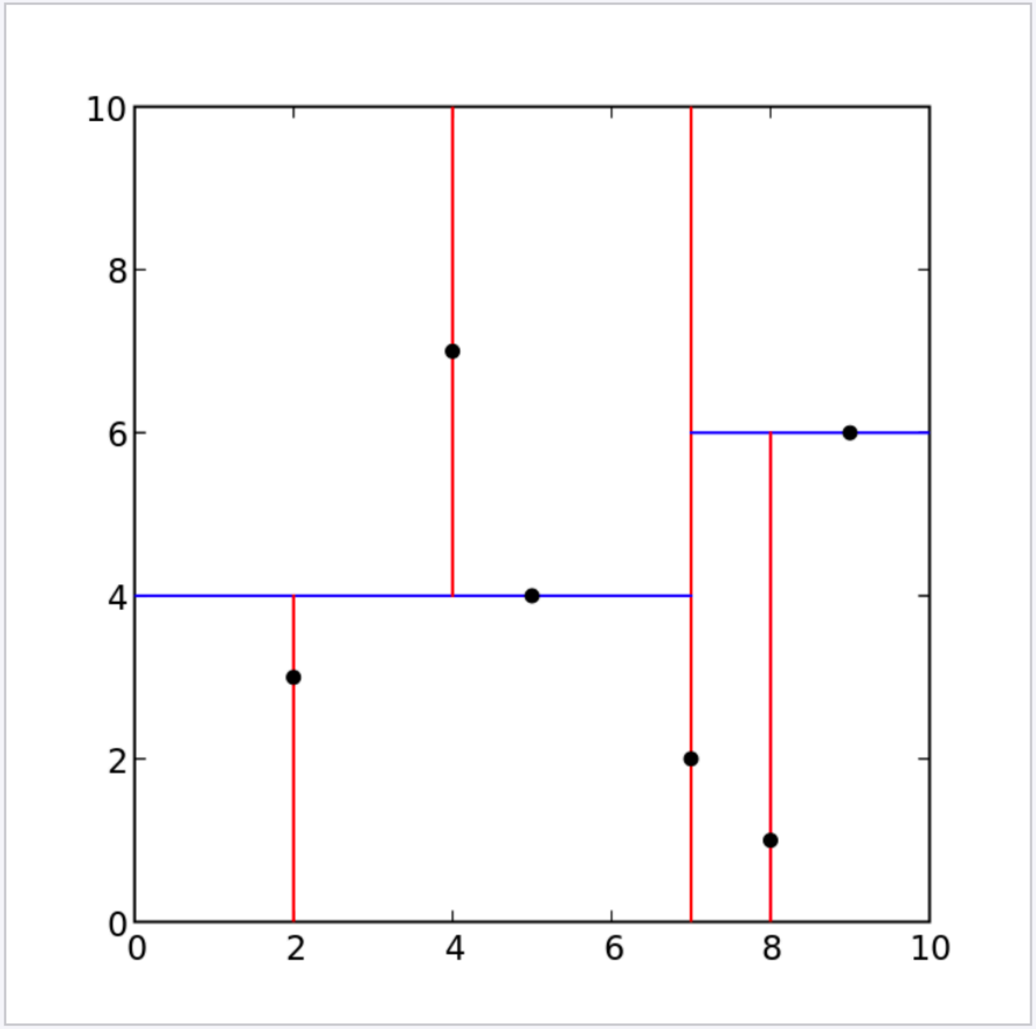
- Cost per test example, if naively implemented?
- If we have *n* test examples, total (naive) cost?

Computational concerns

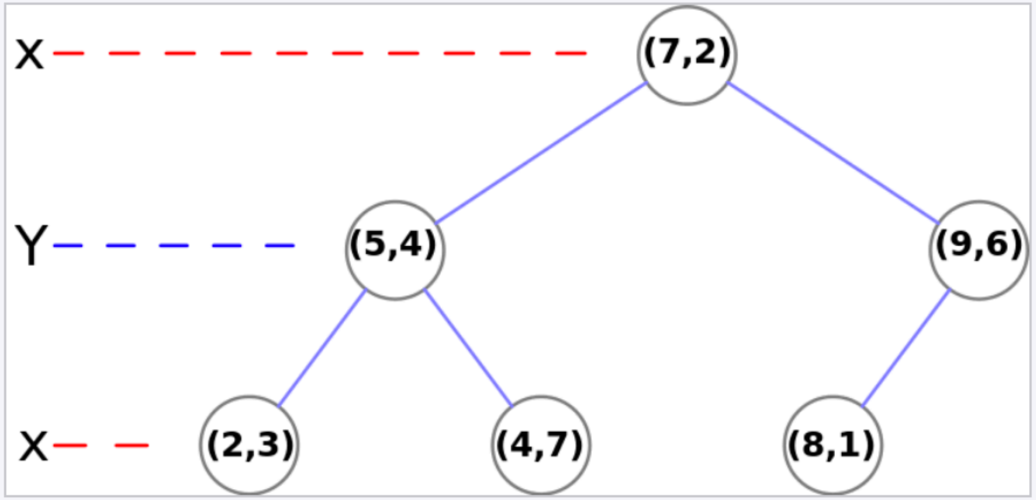
For each test example, need to find the k nearest neighbors among the n training examples

- Cost per test example, if naively implemented?
- If we have n test examples, total (naive) cost?
- At least in low dimensions (small d), there is hope by using good data structures. In low dimensions, can typically label the entire test set at a cost of

kd-tree



k-d tree decomposition for the point set `(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)`.



The resulting *k*-d tree.

Curse of dimensionality

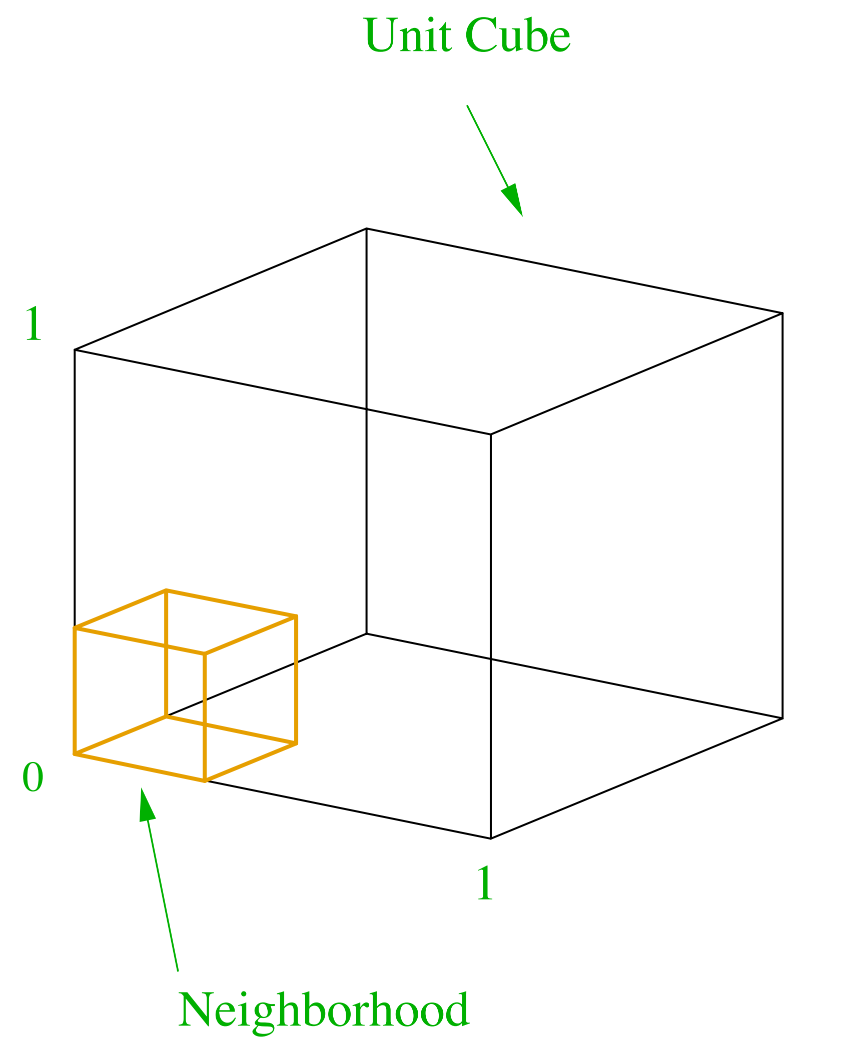
Dimension of problem increases (# features goes up)

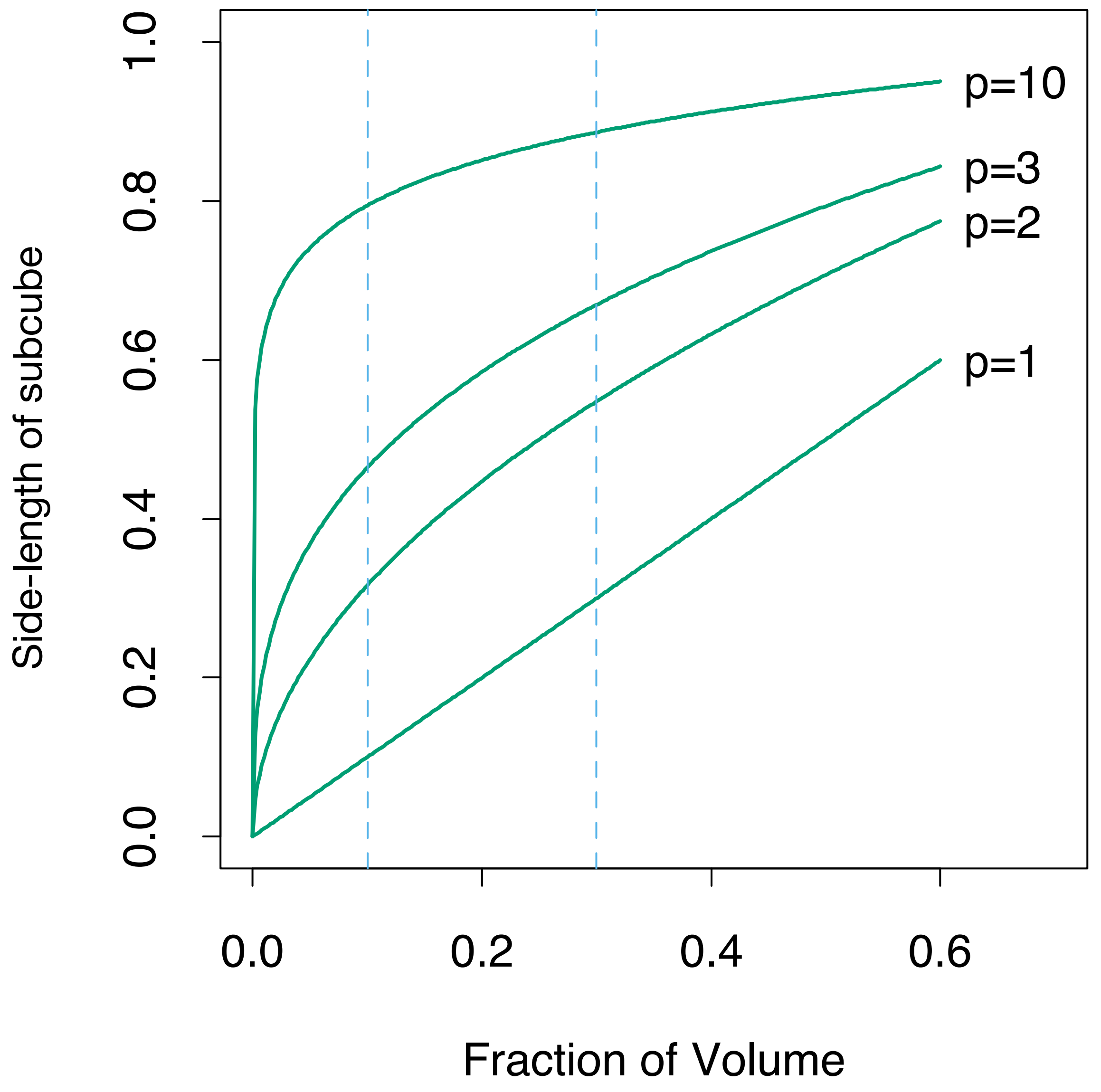
$\Rightarrow$ need exponentially more training examples to ensure each test point is close to some training example

Common problem in machine learning - in high dimensions, without any special structure, learning is fundamentally hard.

1-NN is still 2-approx, but rate of convergence with respect to n depends strongly on d

Curse of dimensionality

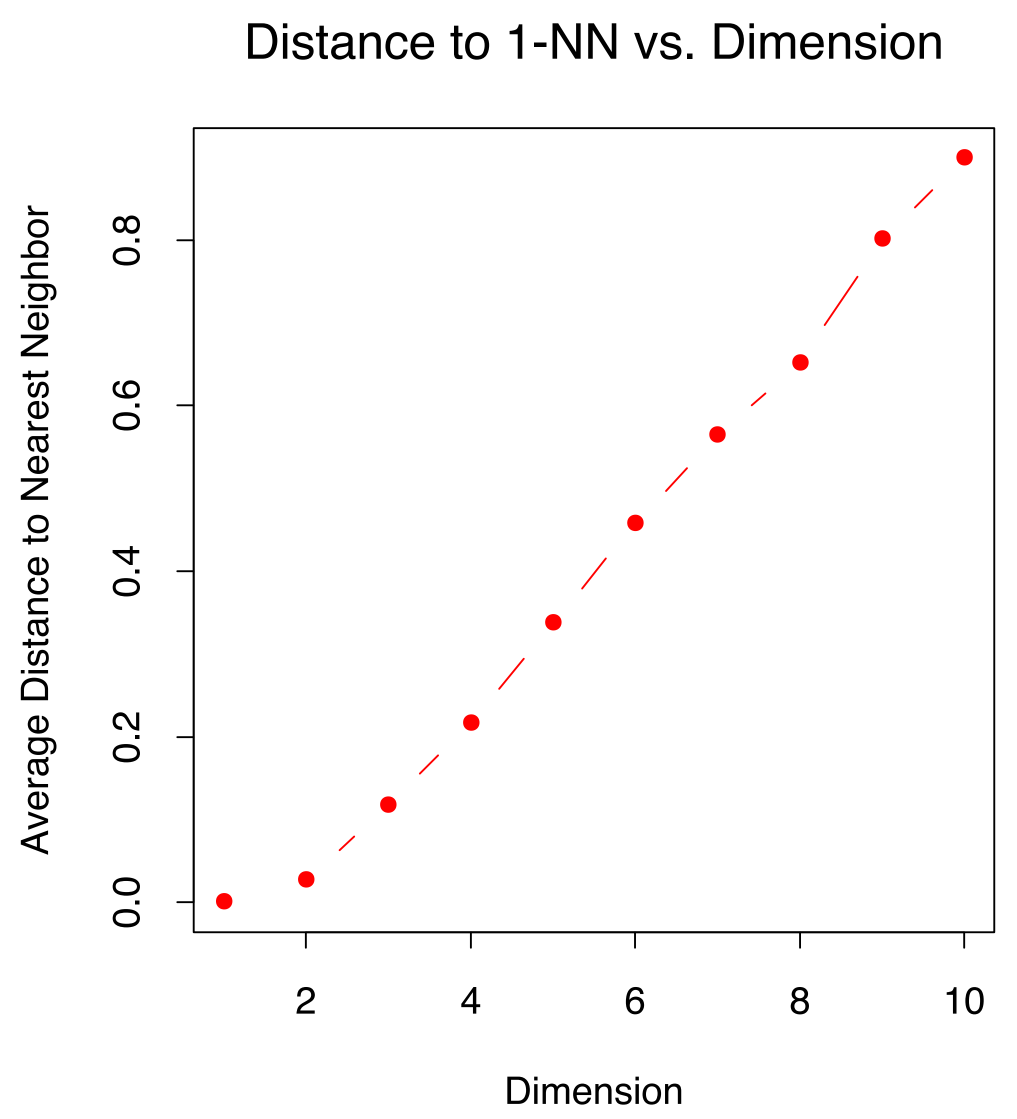


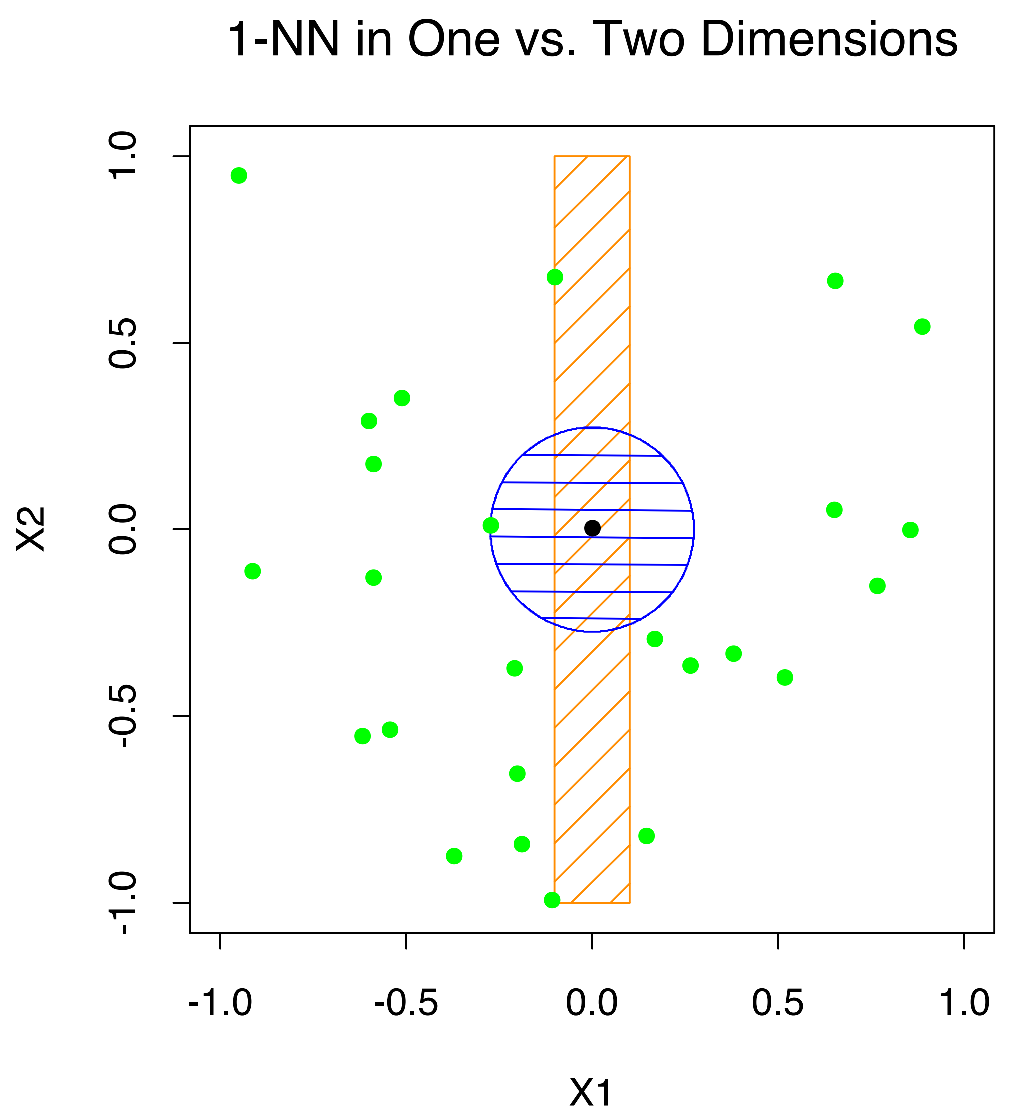


In 10 dimensions, the side-length of a subcube must be ≈ 0.8
(covering 80% of the range of each coordinate!)
in order to cover 10% of the data.

(Hastie et al., 2009), Figure 2.7

Curse of dimensionality





Data drawn uniformly at random from the hypercube $[-1, 1]^d$

In high dimensions, the nearest neighbor is very far away!

(Hastie et al., 2009), Figure 2.7

k-nearest neighbor regression

Input examples in $\mathbb{R}^d$

still based on Euclidean distance

For test example x_{test}, find k nearest neighbors $x^{(1)}, \ldots, x^{(k)}$

Prediction? Simple average of labels of k nearest neighbors:

$$\hat{h}(x) = \frac{1}{k} \sum_{i=1}^{k} y^{(i)}$$

k-nearest neighbor regression

Is regression using the k-nearest neighbor rule a good idea?

If the underlying function we are trying to predict does not change quickly, then it is sensible to take a local average of the targets we have observed in the vicinity of a test example. This is what k-NN regression does.

If the targets all have independent, zero-mean noise, the average can serve to dampen the noise.

Distance-weighted nearest-neighbor prediction

Rather than using the simple average (for regression) or the simple majority vote (for classification), we can also give weights to the *k*-nearest neighbors according to their distance to the test example.

We introduce a weight function, or similarity function
(often called a **kernel function** (not to be confused with kernels from SVMs!))

Distance-weighted nearest-neighbor prediction

We introduce a weight function, or similarity function
(often called a **kernel function** (not to be confused with kernels from SVMs!))

Let $x^{(1)}, \ldots, x^{(k)}$ be the k nearest neighbors of test example x

Classification

$$\hat{h}(x) = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{k} S(x, x^{(i)}) 1\left[y^{(i)} = y\right]$$

Regression

$$\hat{h}(x) = \frac{\sum_{i=1}^{k} S(x, x^{(i)}) y^{(i)}}{\sum_{i=1}^{k} S(x, x^{(i)})}$$

Examples of similarity functions:

$$S(x, x') = \frac{1}{d(x, x')^2}$$

$$S(x, x') = e^{-\gamma d(x, x')^2}$$

Distance-weighted nearest-neighbor prediction

Since we are weighting examples now, we can just as well use *all* of the training examples

Classification

$$\hat{h}(x) = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{n} S(x, x^{(i)}) 1 \left[y^{(i)} = y \right]$$

Regression

$$\hat{h}(x) = \frac{\sum_{i=1}^{n} S(x, x^{(i)}) y^{(i)}}{\sum_{i=1}^{n} S(x, x^{(i)})}$$

Examples of similarity functions:

$$S(x, x') = \frac{1}{d(x, x')^2}$$

$$S(x, x') = e^{-\gamma d(x, x')^2}$$

Challenge: Using the right distance metric

What if not all the features are relevant?
Maybe we want to figure out (using only the training set!) which features to include in our distance calculations

Example:

Predicting preference for dark chocolate based on features:

{ likes-coffee, likes-tea, height, likes-milk }

x_1 x_2 x_3 x_4

Probably not relevant!

A better distance metric:

$$d(x, x') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + (x_4 - x_4')^2}$$

Challenge: Using the right distance metric

What if some features should have more weight than others? We could consider a modified distance metric that weights each feature differently:

$$d(x, x') = \sqrt{\sum_{j=1}^{d} a_j (x_j - x'_j)^2}$$

some nonnegative weight given to feature j

Learning the feature weights $a_1, \dots, a_d$ is one example of a general problem called *metric learning*.

How to learn good feature weights? Idea: Use cross-validation to learn weights that make a k-NN learner predict the best.

Recommender systems

Suppose that we have n users in a system that have rated m items (like movies)

Formally, we are given a sparse matrix (mostly empty entries) with user ratings of items

$$
\begin{array}{c|c|c|c|c|c|}
 & i_1 & i_2 & i_3 & \cdots & i_m \\
\hline
u_1 & 4 & ? & 2 & & 5 \\
\hline
u_2 & ? & 2 & ? & & 4 \\
\hline
u_3 & 3 & ? & 5 & & ? \\
\hline
\vdots & & & & & \\
\hline
u_n & ? & 1 & ? & & 3 \\
\hline
\end{array}
$$

Recommender systems

We want to recommend some item to a user.

Key subproblem:

- Predict how the user would rate each item they haven't yet rated (afterwards, recommend the item with the highest predicted rating)

Key sub-subproblem:

- Predict how the user would rate a single item.

	i_1	i_2	i_3	$\cdots$	i_m
u_1	4	?	2		5
u_2	?	2	?		4
u_3	3	?	5		?
$\vdots$					
u_n	?	1	?		3

Recommender systems

Key sub-subproblem: Predict how user u would rate item i.

How? View this as a regression problem with missing data:

- Training examples: Each user that has rated item i
- Label: the rating of item i

	i_1	i	i_3	$\cdots$	i_m
u_1	4	?	2		5
u_2	?	2	?		4
u	3	?	5		?
$\vdots$					
u_n	?	1	?		3

Using our predictor based on a similarity measure

Suppose that we have similarity measure between users:

- For users u and u', we have a nonnegative similarity $S(u, u')$
- Then we can predict how user u rates item i as

Rating user v gave to item i

$$\hat{r}_{u,i} = \frac{\sum_{v \in U_i} S(u, v) r_{v,i}}{\sum_{v \in U_i} S(u, v)}$$

Set of users that rated item i

What is a good similarity measure?

First, for two users u and v, how can we compare them?

- We can look at the items that they both rated, and see how similar their ratings are on this set

Suppose there are 5 items in common
u rated them as $x = (4.0, 2.0, 4.0, 3.0, 5.0)$
v rated them as $y = (2.0, 1.0, 2.0, 1.5, 2.5)$

One similarity measure is inverse Euclidean distance. Seems bad here. Why?

- Users have same relative ratings ($x = 2\ y$),
  but Euclidean distance is high: $||x - y|| = 4.18$

What is a good similarity measure?

First, for two users u and v, how can we compare them?

- We can look at the items that they both rated, and see how similar their ratings are on this set

Suppose there are 5 items in common
u rated them as $x = (4.0, 2.0, 4.0, 3.0, 5.0)$
v rated them as $y = (2.0, 1.0, 2.0, 1.5, 2.5)$

A natural similarity measure is the *cosine similarity*: the angle between these vectors (i.e., the inner product of the normalized vectors)

- More sensible. Ratings vectors with the same direction will be maximally similar (regardless of length)

$$S(u, v) = \underset{\text{Natural overloading of notation}}{S}(x, y) = \frac{\langle x, y \rangle}{\|x\| \|y\|}$$ = 1 in this case (the maximum possible value)

Pearson correlation

An issue with cosine similarity

- If all the entries of a ratings vector are shifted by the same amount (Eva decides to increase her rating of each movie by 1), then cosine similarity changes

One solution to this problem is the *Pearson correlation*

- First, center each ratings vector (subtract off the mean of its entries)
- Then compute cosine similarity

Let $\bar{x}$ and $\bar{y}$ be means of entries of x and y respectively

$$S(u,v) = S(x,y) = \frac{\langle x-\bar{x}, y-\bar{y}\rangle}{\|x-\bar{x}\|\|y-\bar{y}\|}$$

Pearson correlation

One solution to this problem is the *Pearson correlation*

- First, center each ratings vector (subtract off the mean of its entries)
- Then compute cosine similarity

Let $\bar{x}$ and $\bar{y}$ be means of entries of x and y respectively

$$S(u,v) = S(x,y) = \frac{\langle x - \bar{x}, y - \bar{y} \rangle}{\|x - \bar{x}\| \|y - \bar{y}\|}$$

Careful: similarity is now between -1 and 1. So, when we predict, we need to divide by the sum of the *absolute value* of the similarities

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in U_i} S(u,v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in U_i} |S(u,v)|}$$