COMS 4771
Nearest Neighbors and Decision Trees

Nakul Verma

Last time…

- Why machine learning
- Basics of Supervised Learning
- Maximum Likelihood Estimation
- Learning a classifier via probabilistic modelling
- Optimality of Bayes classifier
- Naïve Bayes classifier
- How to evaluate the quality of a classifier

Classifier via Probabilistic Model

$$\hat{f}(\vec{x}) = \arg\max_{y \in \mathcal{Y}} \; P[Y = y | X = \vec{x}]$$

Bayes optimal

$$= \arg\max_{y \in \mathcal{Y}} \; P[X = \vec{x} | Y = y] \cdot P[Y = y]$$

Class conditional

Class prior

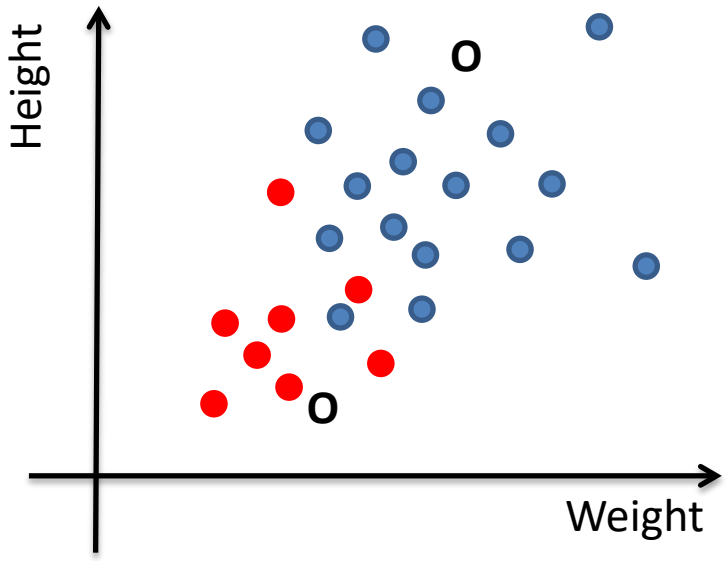
Assume a probabilistic model for each class (say Gaussian) and learn the best parameters via MLE using the training data

Estimate via fraction of examples belonging to that class in the training data (MLE)

- Unclear how to **correctly** model P[X|Y]
- Probability density estimation from samples degrades with representation dimension!

Let's look back at geometry

Data geometry:



$$\mathcal{X} = \mathbf{R}^2 = \text{Weight} \times \text{Height}$$

male data

female data

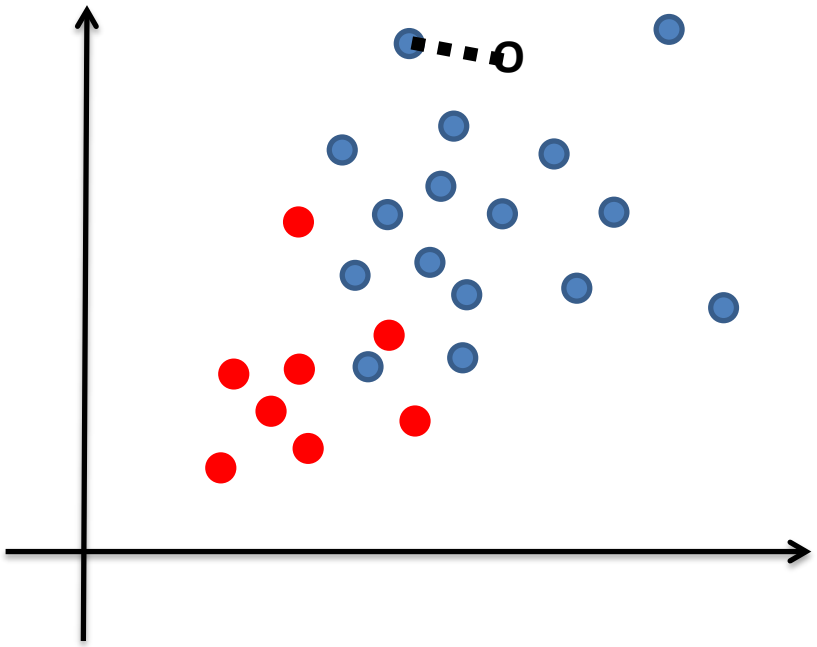
What do we want ultimately?

Make the right prediction on a new test data !

What should be the predicted labels for the test data **o** above?

Nearest Neighbor (NN) classification

Idea: For new test example, assign the label to the same label as its 'closest neighbor'!



How to measure 'closeness' in $\mathcal{X}$?

How to Measure Closeness in Feature Space?

We can measure closeness between two examples x_1, x_2 in many ways!

- Compute some sort of distance (smaller the distance, closer the examples)
- Compute some sort of similarity (higher the similarity, closer the examples)
- Can use domain expertise to measure closeness

Typically our measurements are real numbers, that is $\mathcal{X} = \mathbf{R}^d$

Computing Distance Example

If $\mathcal{X} = \mathbf{R}^d$ there a few natural ways of computing distance:

Euclidean distance:

$$\rho(\vec{x}_1, \vec{x}_2) = \left[\left(x_1^{(1)} - x_2^{(1)}\right)^2 + \ldots + \left(x_1^{(d)} - x_2^{(d)}\right)^2 \right]^{1/2}$$
$$= \left[(\vec{x}_1 - \vec{x}_2)^\mathsf{T} (\vec{x}_1 - \vec{x}_2) \right]^{1/2} = \left\| \vec{x}_1 - \vec{x}_2 \right\|_2$$

Other 'normed' distances:

$$\rho(\vec{x}_1, \vec{x}_2) = \left[\left|x_1^{(1)} - x_2^{(1)}\right|^p + \ldots + \left|x_1^{(d)} - x_2^{(d)}\right|^p \right]^{1/p} = \left\| \vec{x}_1 - \vec{x}_2 \right\|_p$$

- $p = 2$ Euclidean distance
- $p = 1$ Manhattan distance or cityblock distance
- $p = \infty$ Max distance
- $p = 0$ count 'non-zero' distance

Computing Similarity Example

If $\mathcal{X} = \mathbf{R}^d$ there a few natural ways of computing similarity:

Typical ways:

$$\rho(\vec{x}_1, \vec{x}_2) = \frac{1}{1 + \|\vec{x}_1 - \vec{x}_2\|_2} \qquad \rho(\vec{x}_1, \vec{x}_2) = \sum_{i=1}^{d} \frac{1}{1 + \left|x_1^{(i)} - x_2^{(i)}\right|}$$

Cosine similarity:

$$\rho(\vec{x}_1, \vec{x}_2) = \cos(\angle(\vec{x}_1, \vec{x}_2)) = \frac{\vec{x}_1 \cdot \vec{x}_2}{\|\vec{x}_1\|_2 \ \|\vec{x}_2\|_2}$$

Closeness Using Domain Expertise Example

Use information about the particular domain:

Edit distance (to compare e.g. genome sequences)

x_1 = AAATCCCGTAA

x_2 = AATCGCGTAA

Minimum number of insertions, deletions and mutations needed

$$\rho(x_1, x_2) = 2$$

Kendell-Tau distance (to compare e.g. rankings)

x_1 = [o1 o2 o3 o4 o5]

x_2 = [o2 o1 o3 o4 o5]

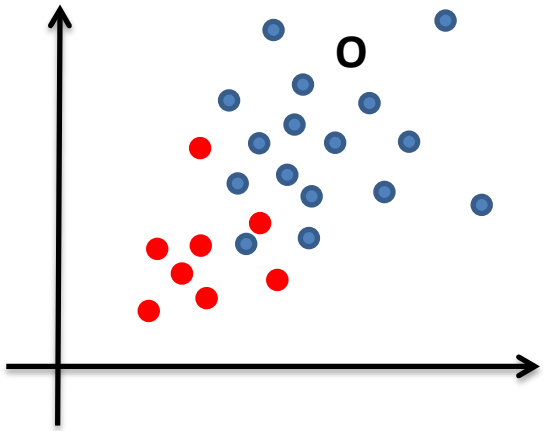
Bubble sort distance to make one ranking order same as the other

$$\rho(x_1, x_2) = 1$$

Nearest Neighbor (NN) classification

For any test example:

assign the label to the same label as its ‘closest neighbor’!



Some issues:

Sensitive to noise in data, so labelling is unstable

Can make it stable: by **taking majority** among *k*-nearest neighbors!

Approaches to Classification

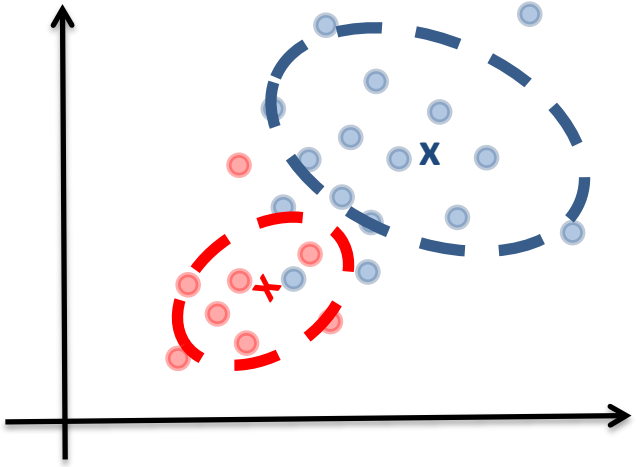
Generative approach:

Advantages:

- A probability model gives **interpretation** of how data gets **generated** from population

Disadvantages:

- Need to pick a probability model
- Doing more work then required to do classification so prone to errors!



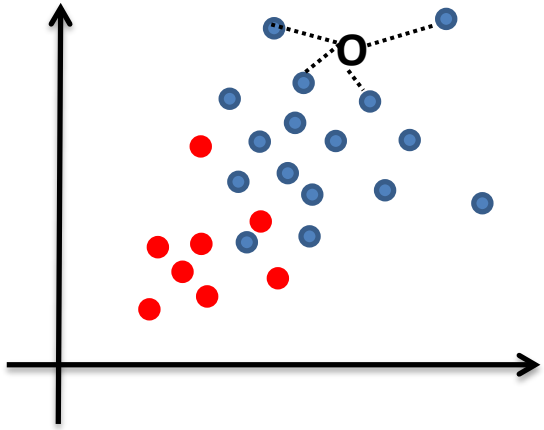
Discriminative approach:

Advantages:

- Typically **better** classification accuracies

Disadvantages:

- Gives no understanding of the population.



What about k-NN optimality?

Theorem 1:

For fixed k, as and $n \to \infty$, k-NN classifier error converges to no more than twice Bayes classifier error.

Theorem 2:

If $k \to \infty$, $k/n \to 0$, and $n \to \infty$, k-NN classifier converges to Bayes classifier.

The proof is quite technical, we shall provide a simple sketch for 1-NN for Theorem 1

Proof Sketch (Thm. 1)

1-NN case

Let's look at error at a fixed test point x_t

Notation:

P[e] = NN error rate

$D_n = (X_n, Y_n)$ = labeled training data (size n)

x_n = nearest neighbor of x_t in D_n

y_n = label of the nearest neighbor

$$\lim_{n\to\infty} P_{y_t,D_n}\left[e|x_t\right]$$

$$= \lim_{n\to\infty} \int P_{y_t,Y_n}\left[e|x_t X_n\right]\, P[X_n|x_t]\, dX_n$$

$$= \lim_{n\to\infty} \int P_{y_t,y_n}\left[e|x_t, x_n\right]\, P[x_n|x_t]\, dx_n$$

For NN classifier

$$= \lim_{n\to\infty} \int \left[1 - \sum_{y\in\mathcal{Y}} P(y_t = y, y_n = y|x_t, x_n)\right]\, P[x_n|x_t]\, dx_n$$

1-NN

$$= \lim_{n\to\infty} \int \left[1 - \sum_{y\in\mathcal{Y}} P(y_t = y|x_t) P(y_n = y|x_n)\right]\, P[x_n|x_t]\, dx_n$$

i.i.d

$$= 1 - \sum_{y\in\mathcal{Y}} P^2(y_t = y|x_t)$$

$x_n \to x_t$

Proof Sketch (Thm. 1) contd.

1-NN case, for a fixed test point x_t

$$\lim_{n\to\infty} P_{y_t,D_n}\left[e|x_t\right] = 1 - \sum_{y\in\mathcal{Y}} P^2(y_t = y|x_t)$$

Notation:

P[e] = NN error rate

X_n = training data (size n)

P^*[e] = Bayes error rate

If Bayes classifier returns y* at point x_t, then

$$\begin{aligned} 1 - \sum_{y\in\mathcal{Y}} P^2(y_t = y|x_t) &\leq 1 - P^2(y_t = y^*|x_t) \\ &\leq 2\Big(1 - P(y_t = y^*|x_t)\Big) \\ &= 2P^*\left[e|x_t\right] \end{aligned}$$

Finally, integrate over the fixed x_t for the final result.

■

A Closer Look at *k*-NN Classification

- Finding the *k* closest neighbor takes time!
- Most times the 'closeness' in raw measurement space is not good!
- Need to keep all the training data around during test time!

Issues with *k*-NN Classification

- Finding the *k* closest neighbor takes time!
- Most times the 'closeness' in raw measurement space is not good!
- Need to keep all the training data around during test time!

Speed Issues with *k*-NN

Given a test example $\vec{x}_t$

What is computational cost of finding the closest neighbor?

$O(nd)$

n = # of training data

d = representation dimension

Modern applications of machine learning

n = millions

d = thousands

How can we find the neighbor faster?

Finding the Neighbor Quickly

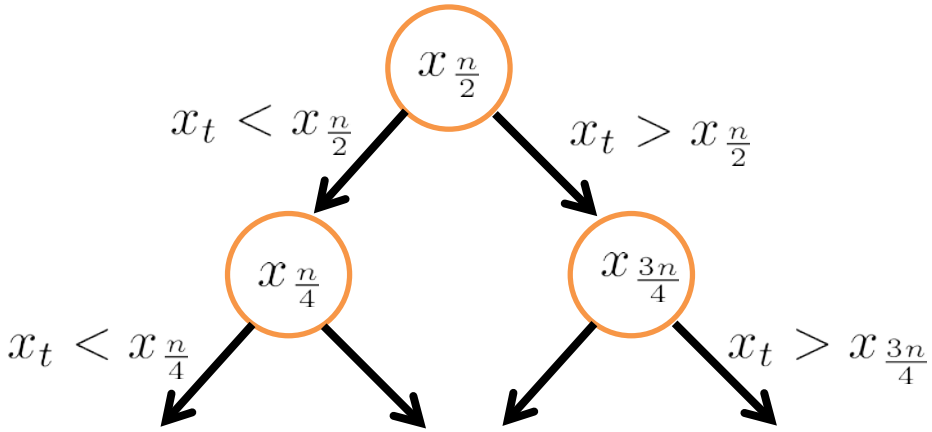
Let's simplify to **R**

How do you find an element x_t from a pool $x_1, x_2, \ldots, x_n$ of examples?

Naïve approach $O(n)$

How can we do the search more quickly?

Say the pool of examples is **sorted**:



Can significantly **improve** the search time

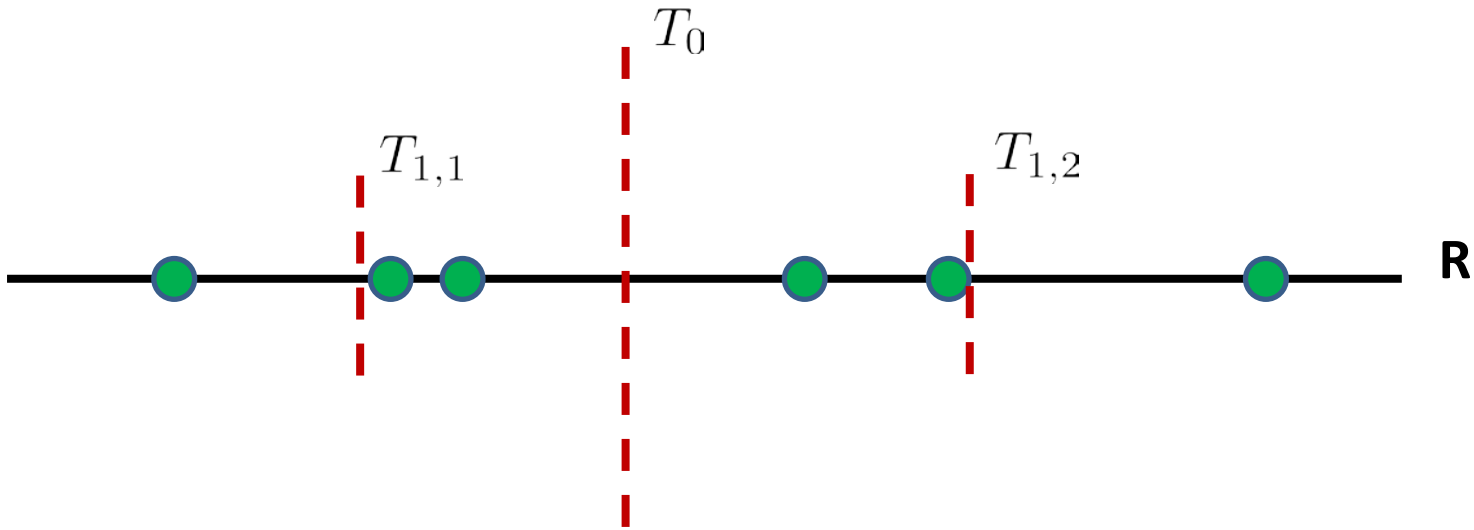
$O(\log n)$

Preprocessing overhead (**sorting**)

$O(n \log n)$

Finding the Neighbor Quickly (contd.)

What if x_t is not in the pool?

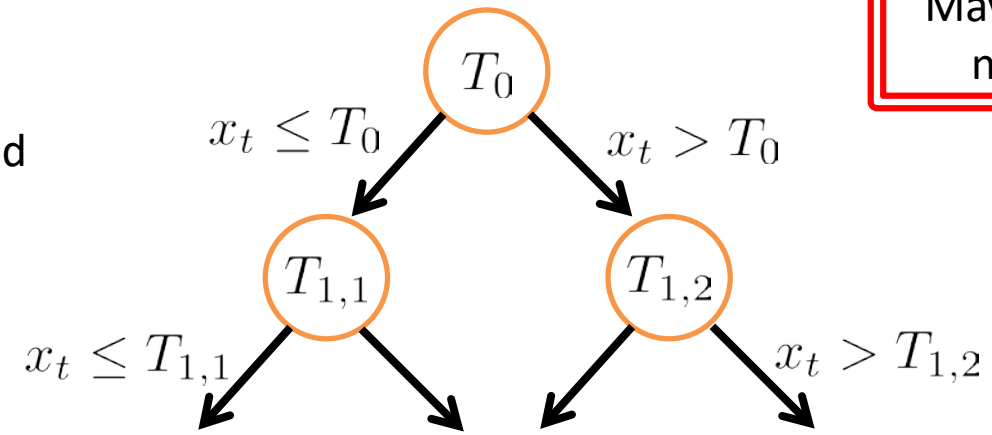


the search time

$O(\log n)$

Preprocessing overhead (finding **medians**)

$O(n \log n)$



May not give the exact nearest neighbor!

Finding the Neighbor Quickly (contd. 2)

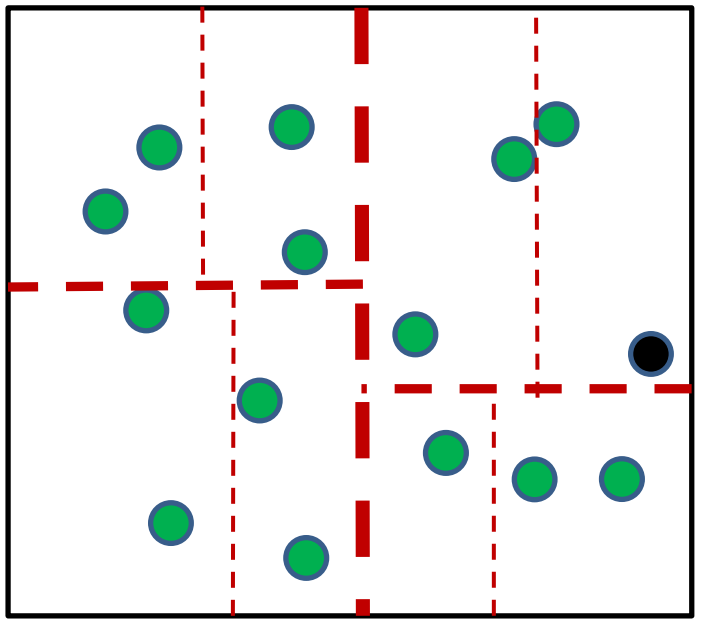
Generalization to $\mathbf{R}^d$

the search time

$O(\log n)$

Preprocessing overhead
(finding **medians**)

$O(n \log n)$



This datastructure is called *k*-d trees

Several Ways to Find Neighbors Fast

Tree-based methods:

- *k*-d trees
- Cover trees
- Navigation nets
- Ball trees
- Spill trees
- …

Compression-based methods:

- Locality Sensitive Hashing (LSH)
- Vector Quantization (VQ) methods
- Clustering methods
- …

Issues with *k*-NN Classification

- Finding the *k* closest neighbor takes time!

- Most times the 'closeness' in raw measurement space is not good!

- Need to keep all the training data around during test time!

Issues with *k*-NN in Raw Measurement Space

Often times we don't know what measurements are helpful for classification a priori.

Recall the old task: learn a classifier to distinguish **`males`** from **`females`**

But say we don't know which measurements would be helpful, so we measure a whole bunch:

height	Income
weight	Number of friends
blood type	Blood sugar level
eye color	…

What happens to the k-NN distance computations?

Improving the Measurement Space

Observation:

- Feature measurements not-relevant (noisy) for the classification task simply distorts NN distance computations
- Even highly correlated relevant measurements (signal) distorts the distance comparisons

How can we make our distance measurement robust?

Idea:

Re-weight the contribution of each feature to the distance computation!

$$\rho(\vec{x}_1, \vec{x}_2; \vec{w}) = \left[w_1 \cdot \left(x_1^{(1)} - x_2^{(1)}\right)^2 + \ldots + w_d \cdot \left(x_1^{(d)} - x_2^{(d)}\right)^2 \right]^{1/2}$$

$$= \left[(\vec{x}_1 - \vec{x}_2)^\top W (\vec{x}_1 - \vec{x}_2) \right]^{1/2}$$

$$W = \begin{pmatrix} w_1 & & 0 \\ & w_i & \\ 0 & & w_d \end{pmatrix}$$

Learn the **optimal weights** from data!

How to Learn Optimal Weighting?

Want:

Distance metric: $\rho(\vec{x}_1, \vec{x}_2; \vec{w})$

such that: data samples from **same class** yield **small values**

data samples from different class yield large values

One way to solve it mathematically:

Create **two** sets: Similar set $S := \{(\vec{x}_i, \vec{x}_j) \mid y_i = y_j\}$

Dissimilar set $D := \{(\vec{x}_i, \vec{x}_j) \mid y_i \neq y_j\}$

i, j = 1,…, *n*

Define a cost function:

$$\Psi(\vec{w}) := \lambda \sum_{(\vec{x}_i, \vec{x}_j) \in S} \rho(\vec{x}_i, \vec{x}_j; \vec{w}) - (1-\lambda) \sum_{(\vec{x}_i, \vec{x}_j) \in D} \rho(\vec{x}_i, \vec{x}_j; \vec{w})$$

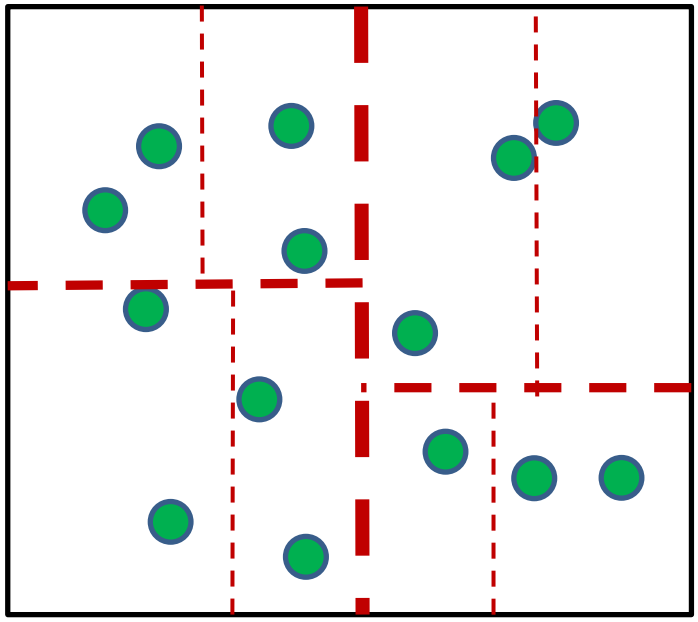
Minimize Ψ w.r.t. $\vec{w}$!

Issues with *k*-NN Classification

- Finding the *k* closest neighbor takes time!
- Most times the 'closeness' in raw measurement space is not good!
- Need to keep all the training data around during test time!

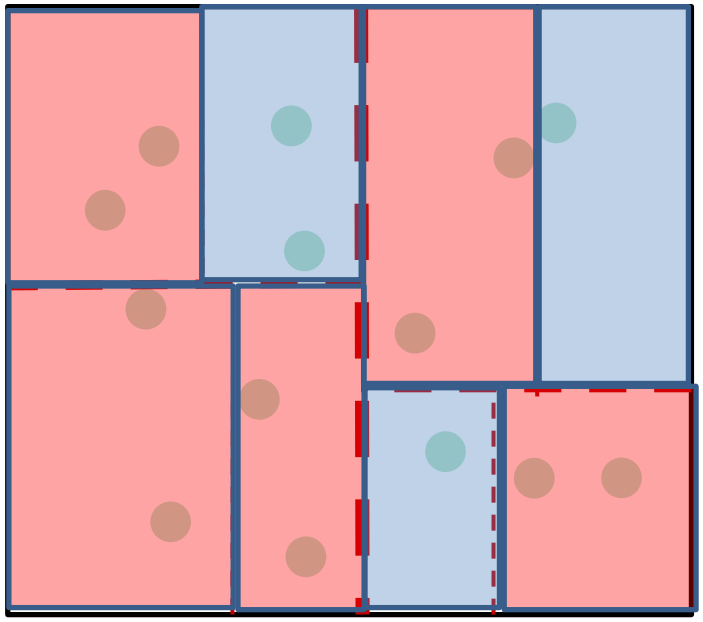
Space issues with *k*-NN

Seems like we need to keep all the training data around during test time



Space issues with *k*-NN

Seems like we need to keep all the training data around during test time



We can **label each cell** instead and discard the training data?

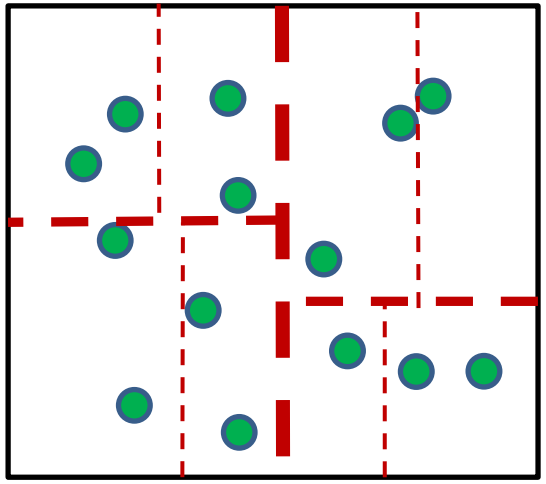
What's the space requirement then? # cells (of width *r*) = $\min\{n, \approx (1/r)^d\}$

NN Summary

- A **simple** and **intuitive** way to do classification
- Don’t need to deal with probability modeling
- Care needs to be taken to select the distance metric
- **Can improve** the basic speed and space requirements for NN

Classification with Trees (Directly)

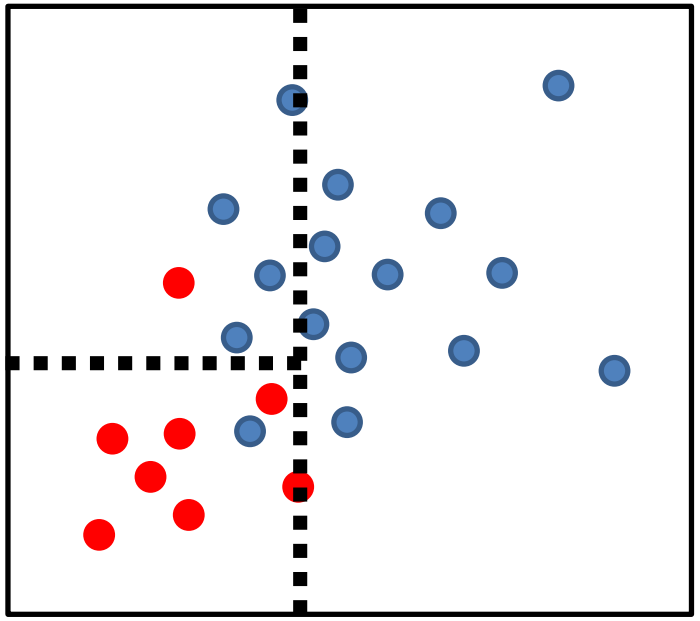
k-d tree construction does not optimize for classification accuracy. Why?



idea: we should choose the features and the thresholds that directly optimize for classification accuracy!

Decision Trees Classifier

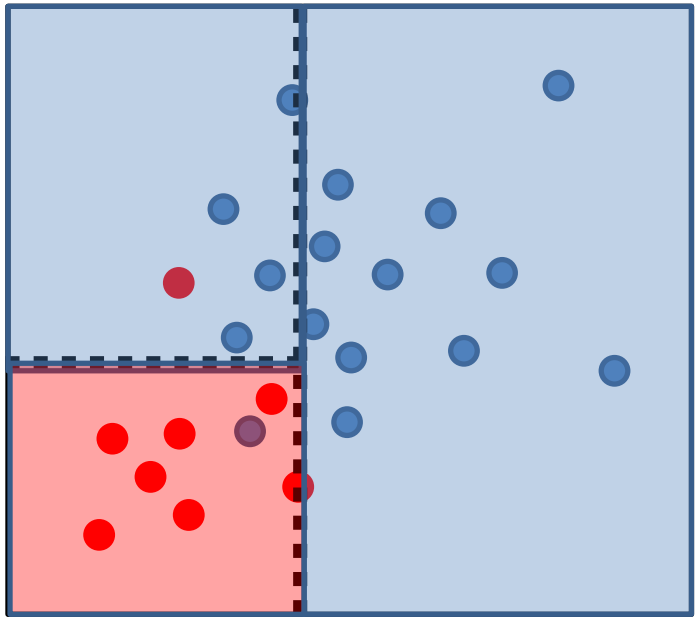
Rather than selecting arbitrary feature and splitting at the median,
select the feature and threshold that **maximally reduces label uncertainty**!



done!

Decision Trees Classifier

Rather than selecting arbitrary feature and splitting at the median,
select the feature and threshold that **maximally reduces label uncertainty**!



done!

How do we measure label uncertainty?

Measuring Label Uncertainty Cells

Several criteria to measure uncertainty in cell *C*:

classification error: $u(C) := 1 - \max_y p_y$

Entropy: $u(C) := \sum_{y \in \mathcal{Y}} p_y \log \frac{1}{p_y}$

Gini index: $u(C) := 1 - \sum_{y \in \mathcal{Y}} p_y^2$

$p_y :=$ fraction of training data labelled *y* in *C*

Thus find the feature *F*, and threshold *T* that **maximally reduces uncertainty**

$$\arg\max_{F,T} \left[u(C) - \left(p_L \cdot u(C_L) + p_R \cdot u(C_R) \right) \right]$$

L = left cell (using *F*, *T*)

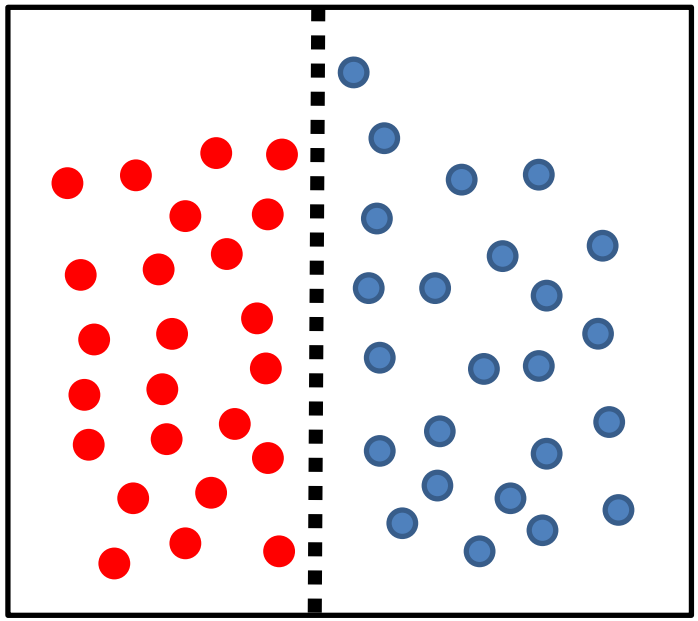
R = right cell (using *F*, *T*)

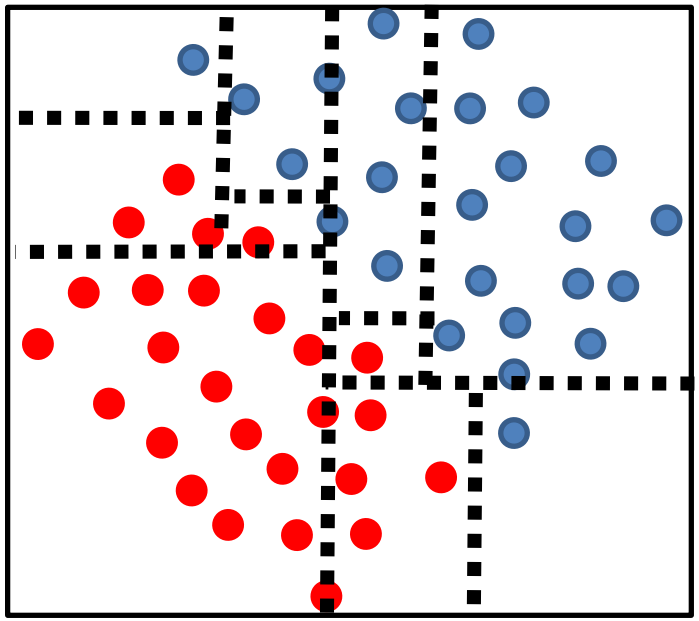
Decision Tree Observations

- The decision tree construction is via a **greedy approach**
- Finding the optimal decision tree is NP-hard!
- You quickly run out of training data as you go down the tree, so uncertainty estimates become very unstable
- Tree complexity is highly dependent on data geometry in the feature space
- **Popular instantiations** that are used in real-world: ID3, C4.5, CART

Decision Trees

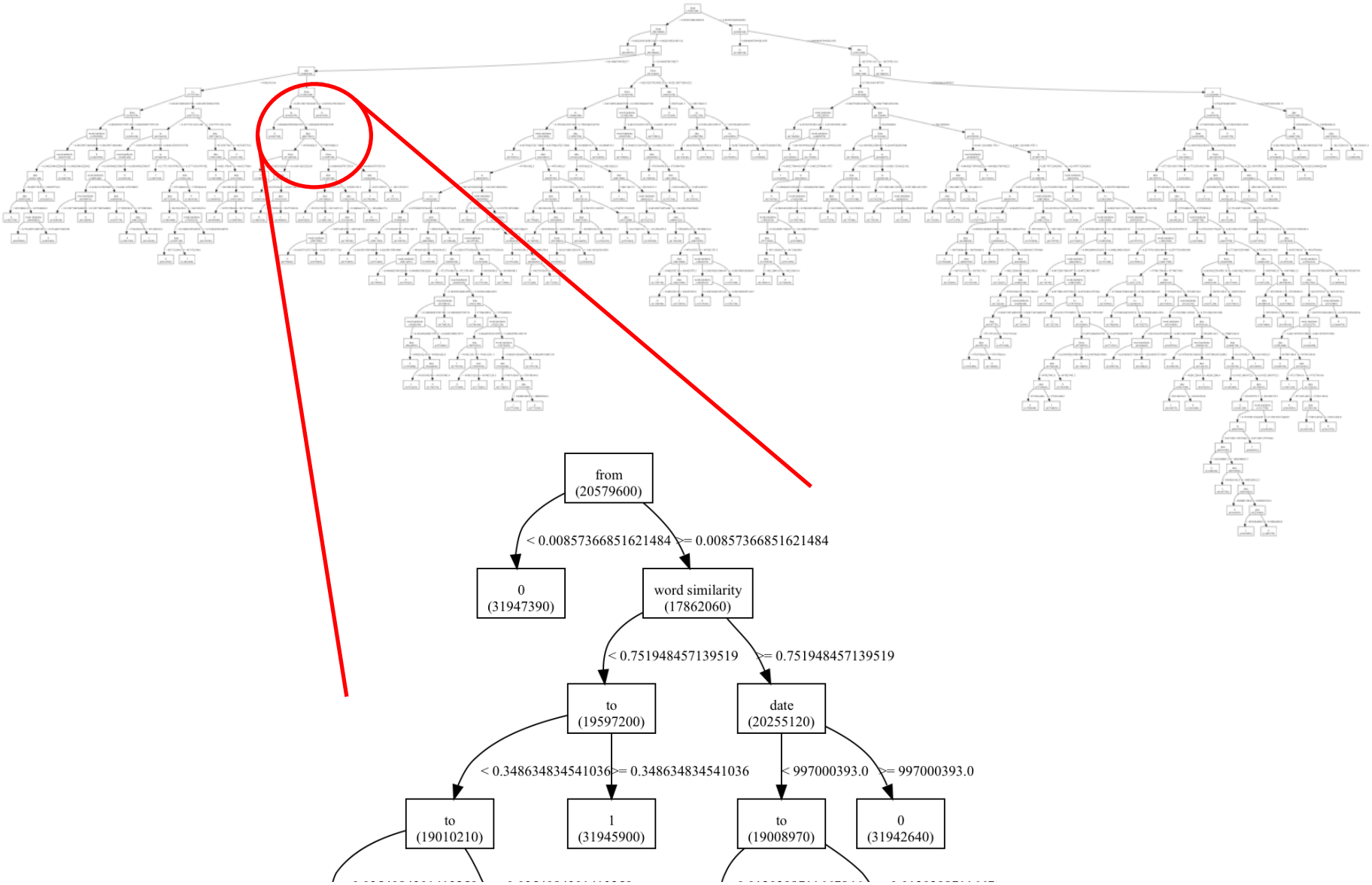
Tree complexity is highly dependent on data geometry in the feature space



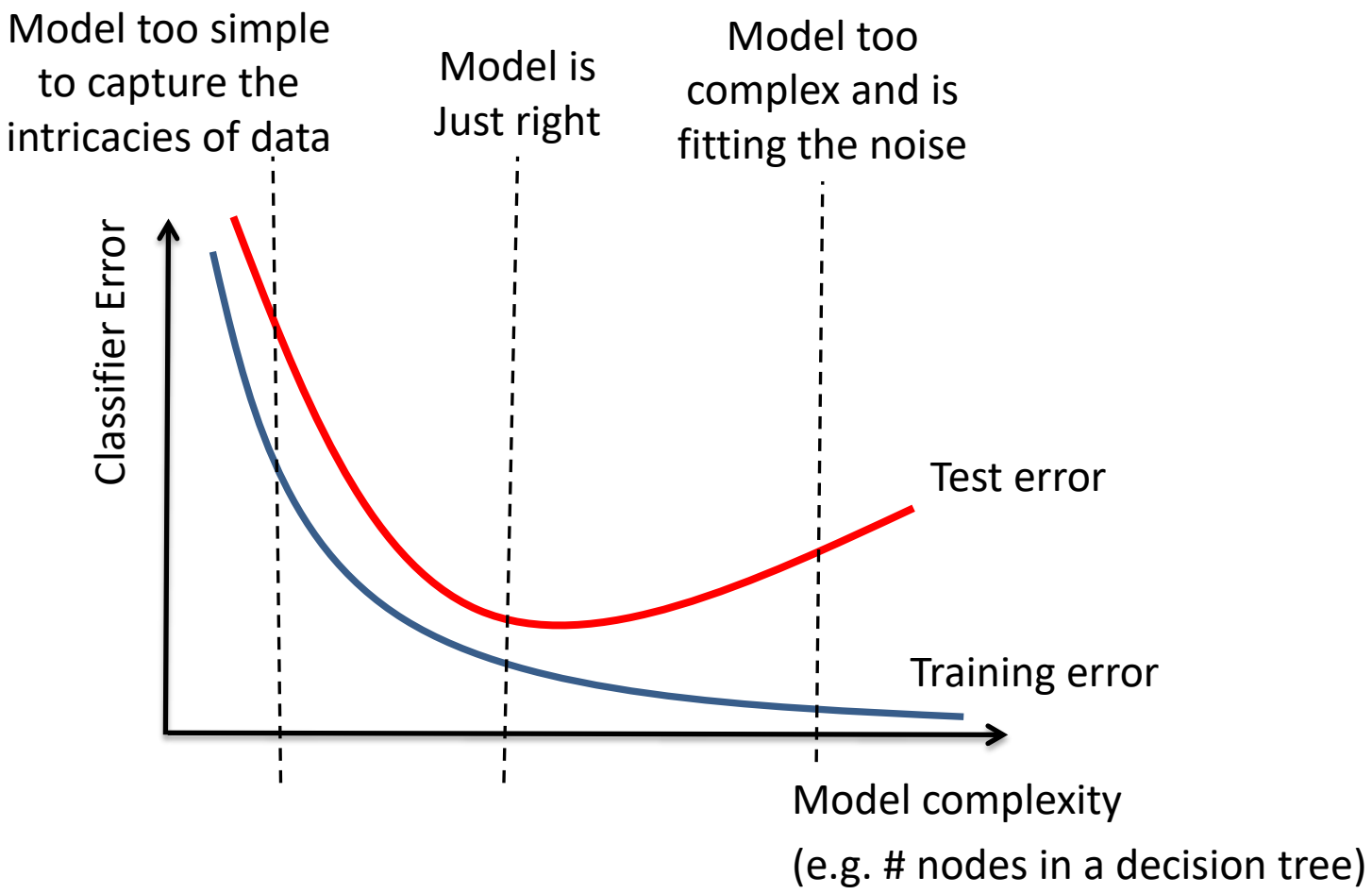


Classifier complexity should not depend on simple transformations of data!

Decision Tree Example (Spam Classification)



Overfitting the training data



How to select a model of the right complexity?

What we learned…

- Generative vs. Discriminative Classifiers
- Nearest Neighbor (NN) classification
- Optimality of *k*-NN
- Coping with drawbacks of *k*-NN
- Decision Trees
- The notion of overfitting in machine learning

Questions?

Next time…

How to construct classifiers using simple rules that can use multiple features at a time!