

CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

Regularized Least Squares (1)

- Consider the error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

- With the sum-of-squares error function and a quadratic regularizer, we get

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

- which is minimized by

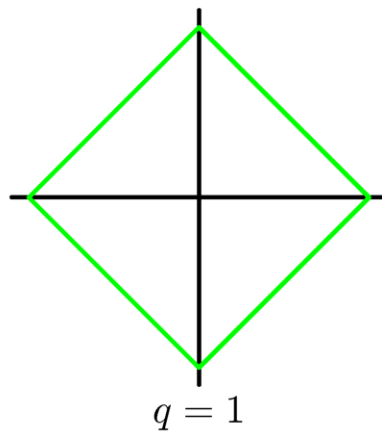
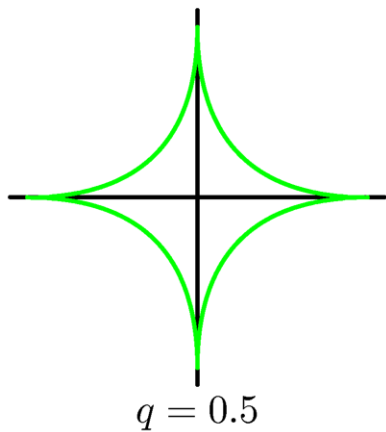
$$\mathbf{w} = \left(\lambda \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}.$$

λ , is called the regularization coefficient.

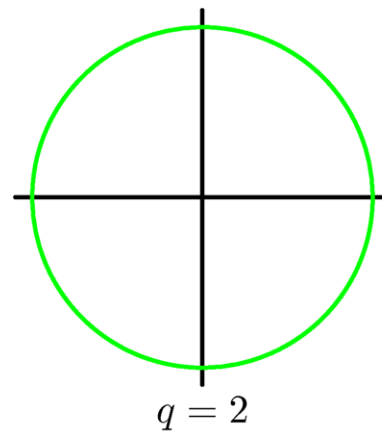
Regularized Least Squares (2)

- With a more general regularizer, we have

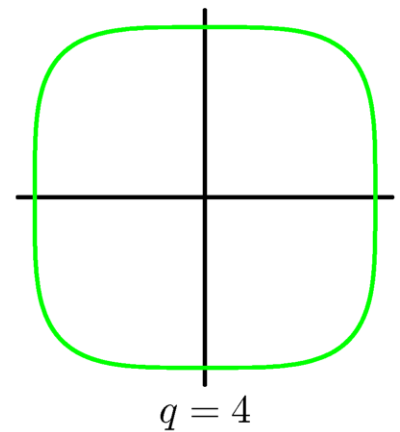
$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$



Lasso

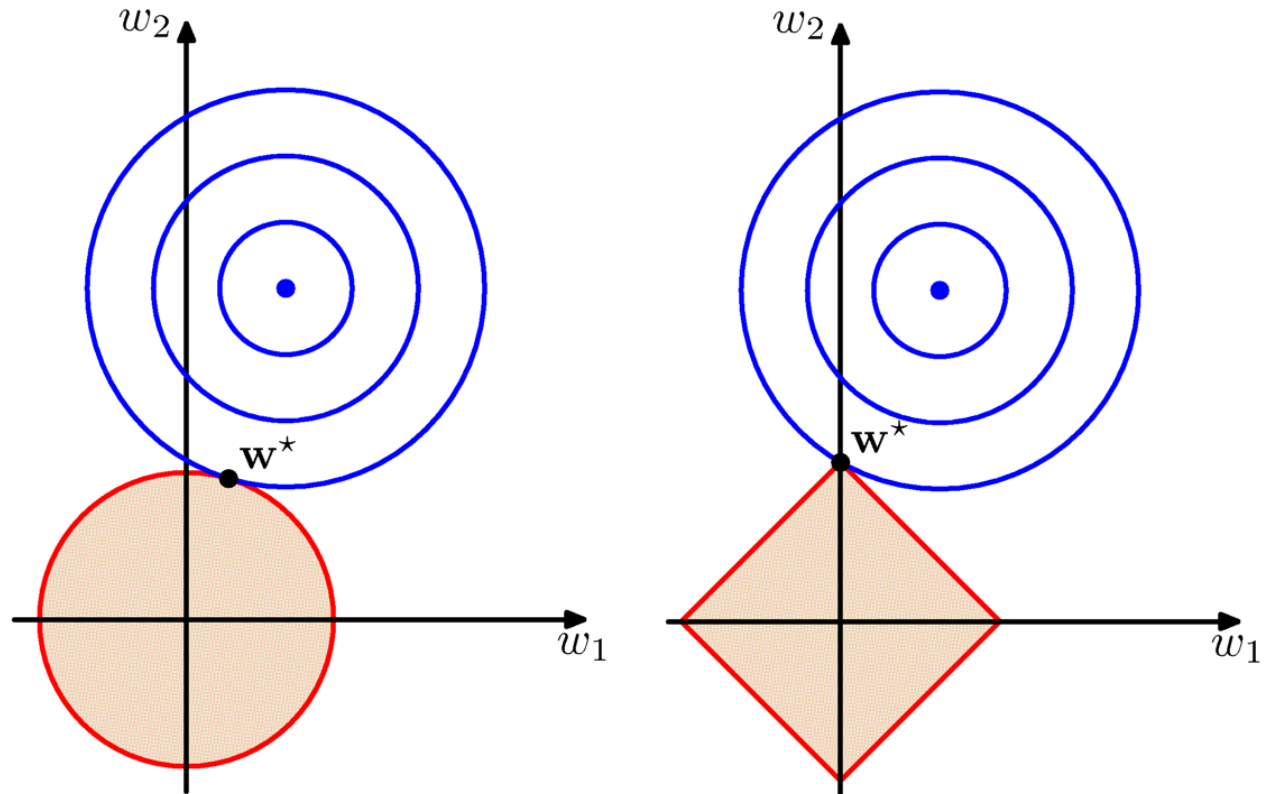


Quadratic



Regularized Least Squares (3)

- Lasso tends to generate sparser solutions than a quadratic regularizer.



The Squared Loss Function

$$\mathbb{E}[L] = \iint \{y(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) \, d\mathbf{x} \, dt$$

$$\begin{aligned} \{y(\mathbf{x}) - t\}^2 &= \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}] + \mathbb{E}[t|\mathbf{x}] - t\}^2 \\ &= \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 + 2\{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}\{\mathbb{E}[t|\mathbf{x}] - t\} + \{\mathbb{E}[t|\mathbf{x}] - t\}^2 \end{aligned}$$

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 p(\mathbf{x}) \, d\mathbf{x} + \int \text{var} [t|\mathbf{x}] p(\mathbf{x}) \, d\mathbf{x}$$

$$y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]$$

The Bias-Variance Decomposition (1)

- Recall the *expected squared loss*,

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x} + \underbrace{\iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt}_{\text{noise}}$$

- where

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int tp(t|\mathbf{x}) dt.$$

- The second term of $\mathbb{E}[L]$ corresponds to the noise inherent in the random variable t .
- What about the first term?

The Bias-Variance Decomposition (2)

- Suppose we were given multiple data sets, each of size N . Any particular data set, \mathcal{D} , will give a particular function $y(\mathbf{x}; \mathcal{D})$. We then have

$$\begin{aligned} & \{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2 \\ &= \{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 \\ &= \{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 \\ &\quad + 2\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}. \end{aligned}$$

The Bias-Variance Decomposition

(3)

- Taking the expectation over \mathcal{D} yields

$$\begin{aligned} & \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2] \\ &= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2]}_{\text{variance}}. \end{aligned}$$

The Bias-Variance Decomposition (4)

- Thus we can write

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

- where

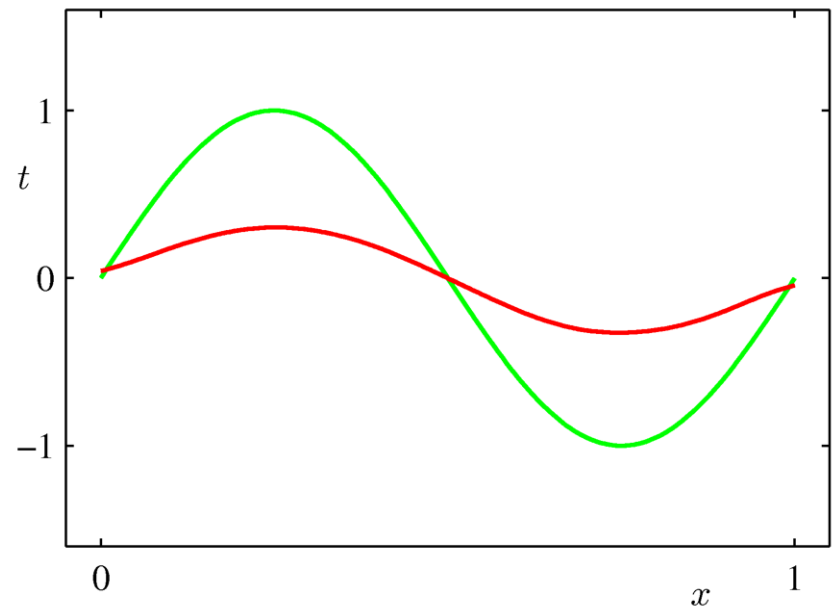
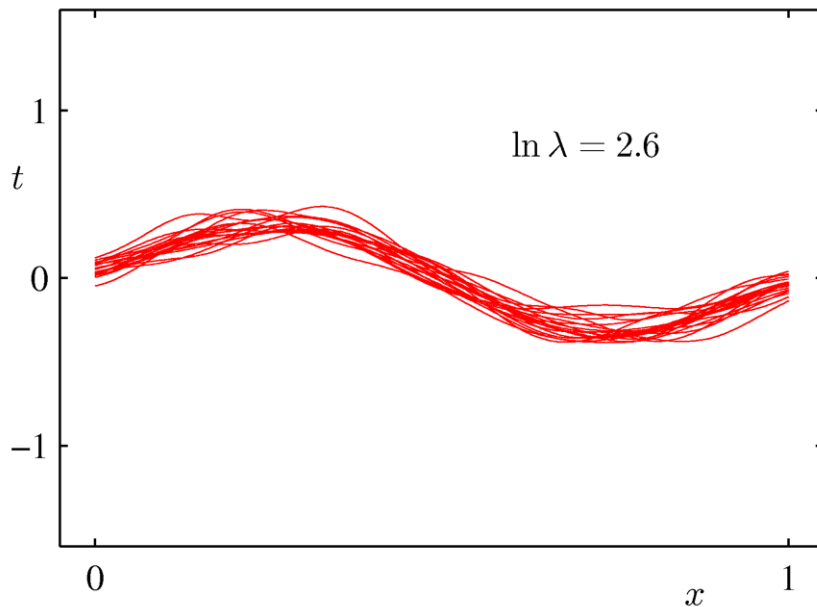
$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x}$$

$$\text{variance} = \int \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2] p(\mathbf{x}) d\mathbf{x}$$

$$\text{noise} = \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt$$

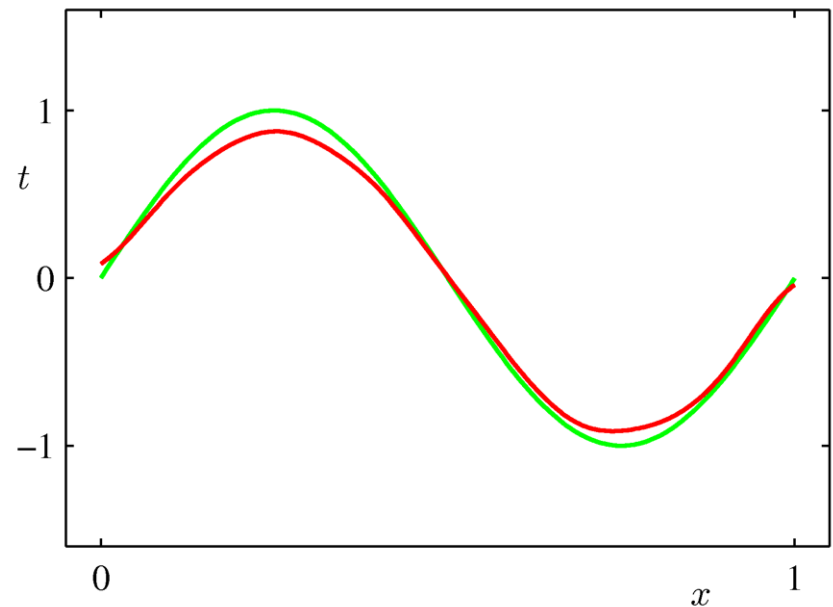
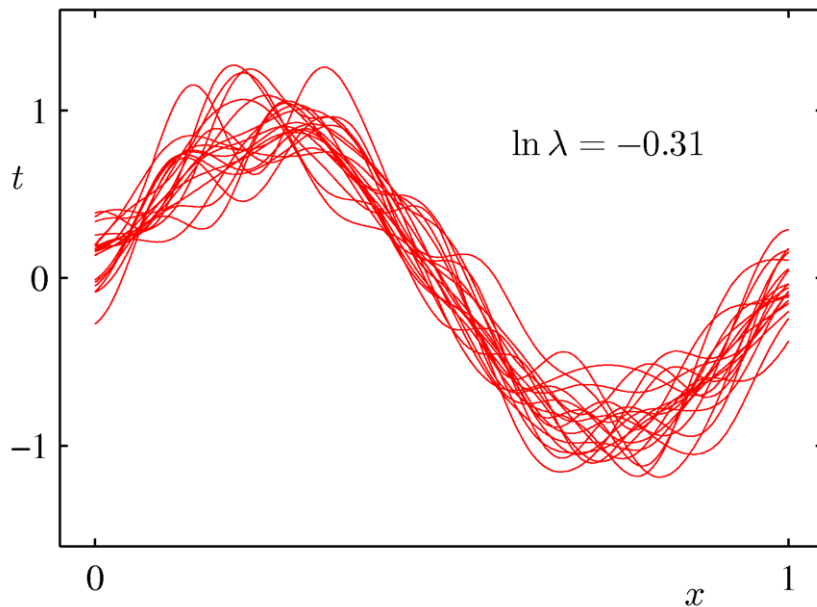
The Bias-Variance Decomposition (5)

- Example: 25 data sets from the sinusoidal, varying the degree of regularization, λ .



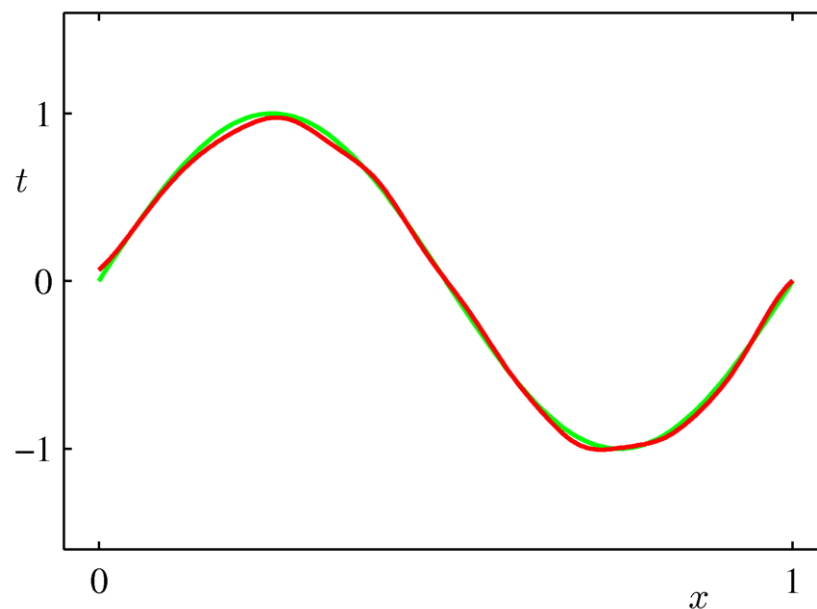
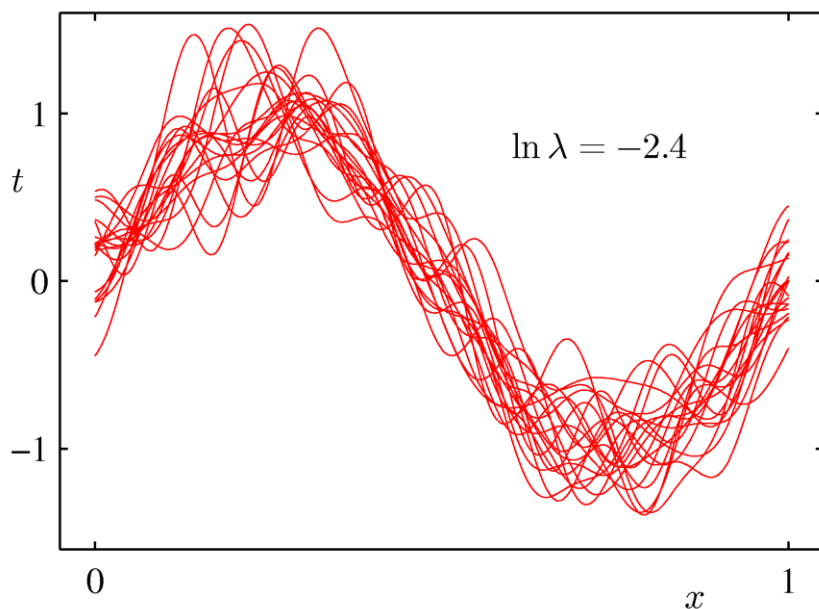
The Bias-Variance Decomposition (6)

- Example: 25 data sets from the sinusoidal, varying the degree of regularization, λ .



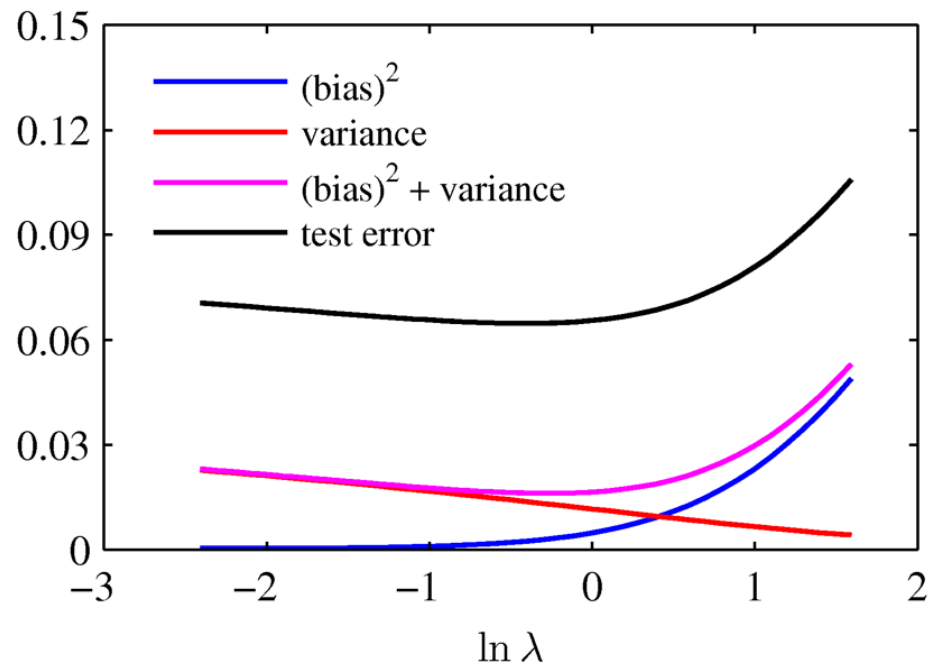
The Bias-Variance Decomposition (7)

- Example: 25 data sets from the sinusoidal, varying the degree of regularization, λ .



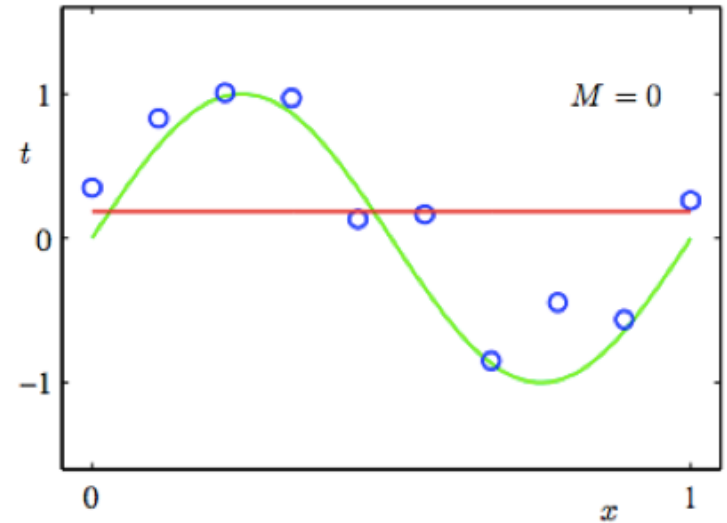
The Bias-Variance Trade-off

- From these plots, we note that an over-regularized model (large λ) will have a high bias, while an under-regularized model (small λ) will have a high variance.

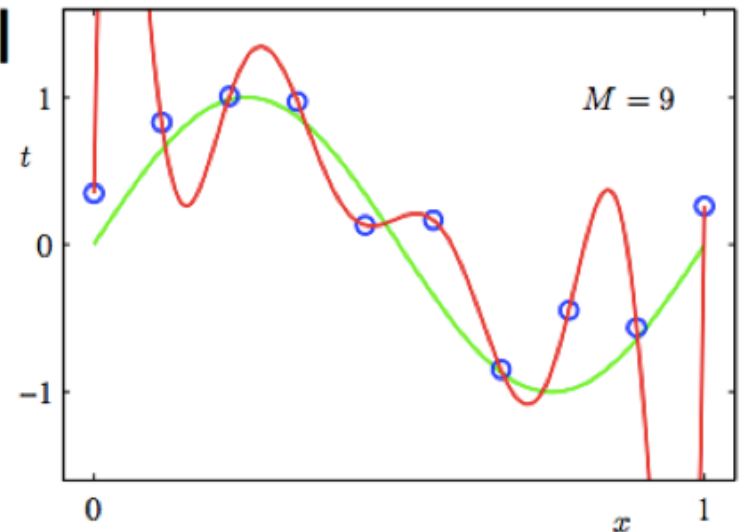


Bias-Variance Tradeoff

- **Model too simple:** does not fit the data well
 - A *biased* solution

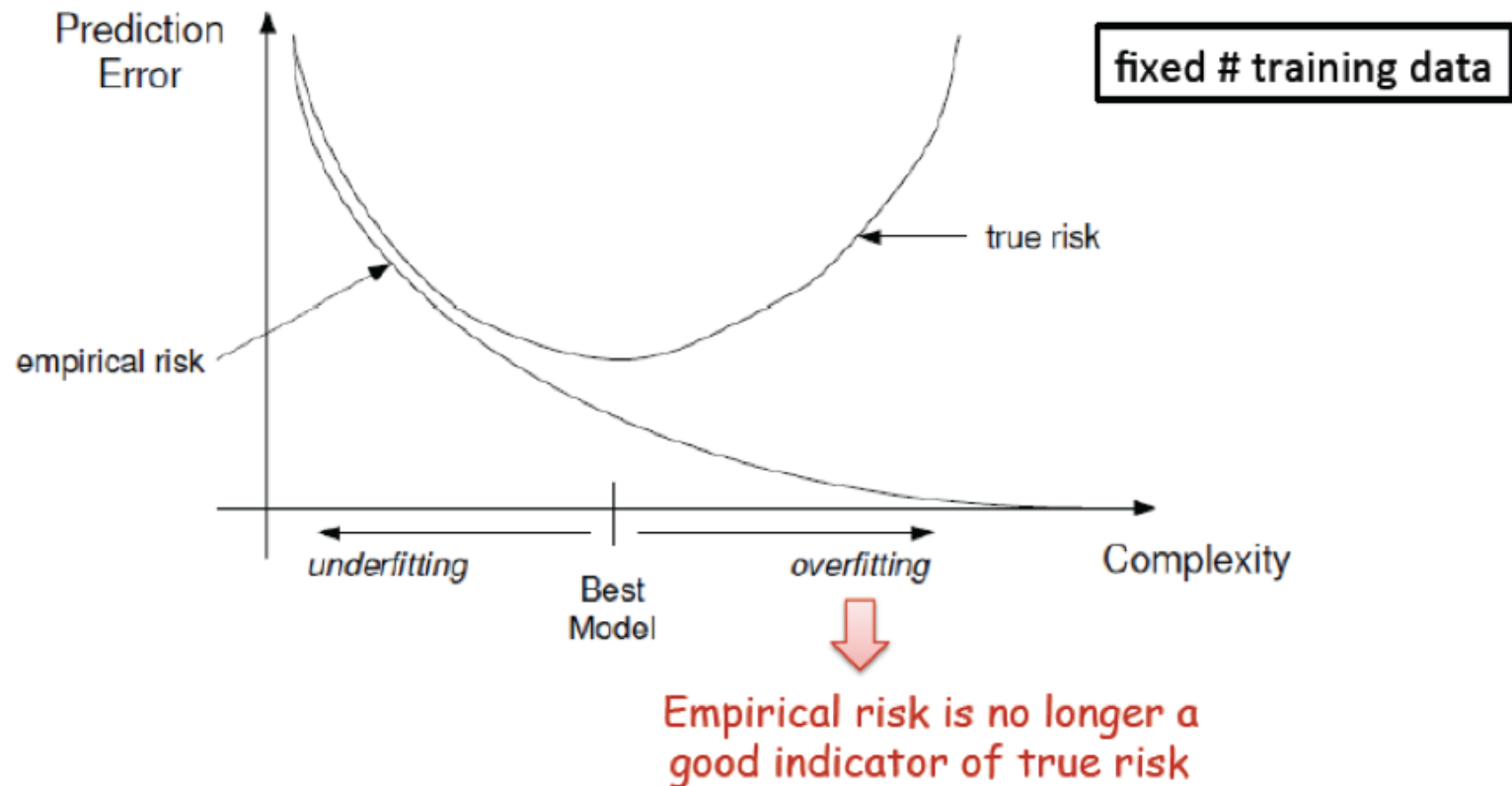


- **Model too complex:** small changes to the data, solution changes a lot
 - A *high-variance* solution



Effect of Model Complexity

- If we allow very complicated predictors, we could overfit the training data.



Model Selection

- We can select the model with right complexity in an adaptive way
- Standard practice: using Hold-out method

Hold-out method

- We would like to pick the model that has smallest generalization error.
- Can judge generalization error by using an independent sample of data.

Hold - out procedure:

n data points available $D \equiv \{X_i, Y_i\}_{i=1}^n$

1) Split into two sets: Training dataset Validation dataset **NOT test Data !!**
 $D_T = \{X_i, Y_i\}_{i=1}^m$ $D_V = \{X_i, Y_i\}_{i=m+1}^n$

2) Use D_T for training a predictor from each model class:

$$\hat{f}_\lambda = \arg \min_{f \in \mathcal{F}_\lambda} \hat{R}_T(f)$$

 Evaluated on training dataset D_T

Hold-out method

3) Use D_V to select the model class which has smallest empirical error on D_V

$$\hat{\lambda} = \arg \min_{\lambda \in \Lambda} \hat{R}_V(\hat{f}_\lambda)$$

 Evaluated on validation dataset D_V

4) Hold-out predictor

$$\hat{f} = \hat{f}_{\hat{\lambda}}$$

Intuition: Small error on one set of data will not imply small error on a randomly sub-sampled second set of data

Ensures method is “stable”

Hold-out method

Drawbacks:

- May not have enough data to afford setting one subset aside for getting a sense of generalization abilities
- Validation error may be misleading (bad estimate of generalization error) if we get an “unfortunate” split

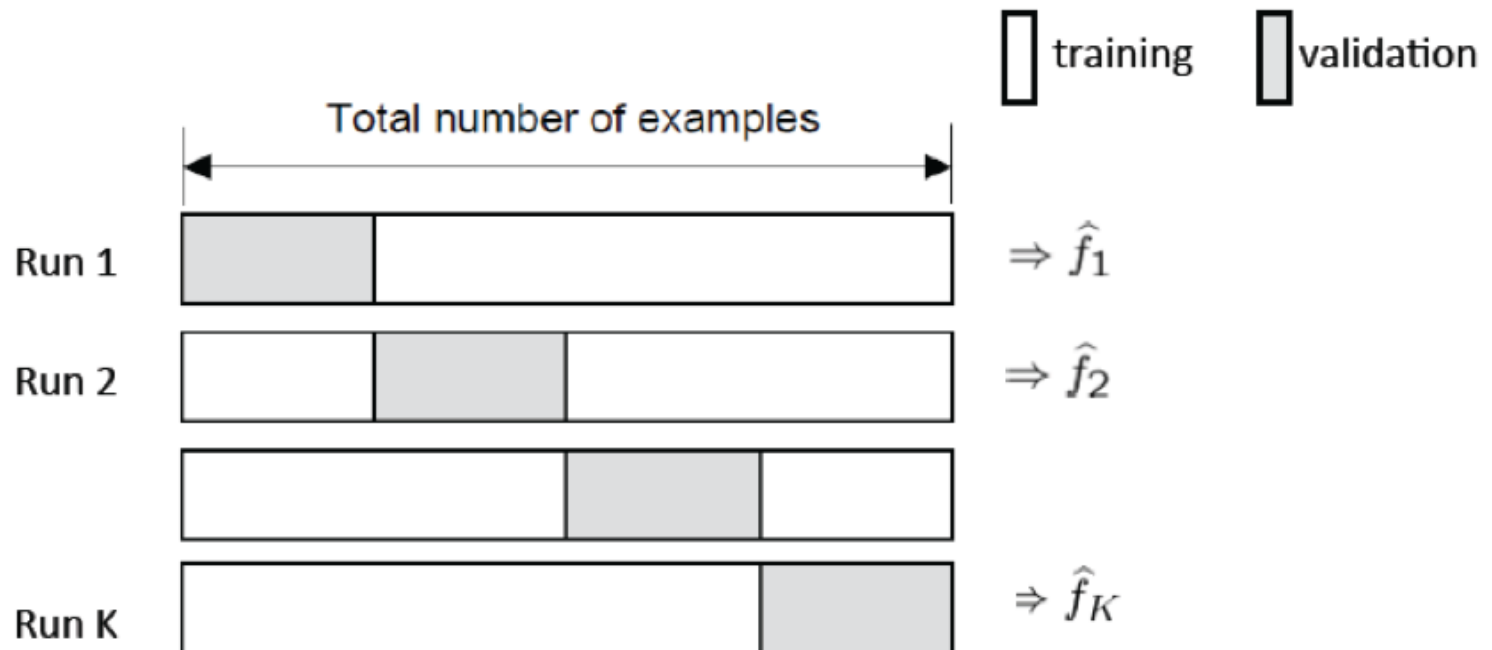
Limitations of hold-out can be overcome by a family of random sub-sampling methods at the expense of more computation.

Cross-Validation

K-fold cross-validation

- 1) Create K-fold partition of the dataset.
- 2) Form K hold-out predictors, each time using one partition as validation and rest K-1 as training datasets.

K predictors for each model class: $\{ \hat{f}_1, \hat{f}_2, \dots, \hat{f}_K \}_\lambda$

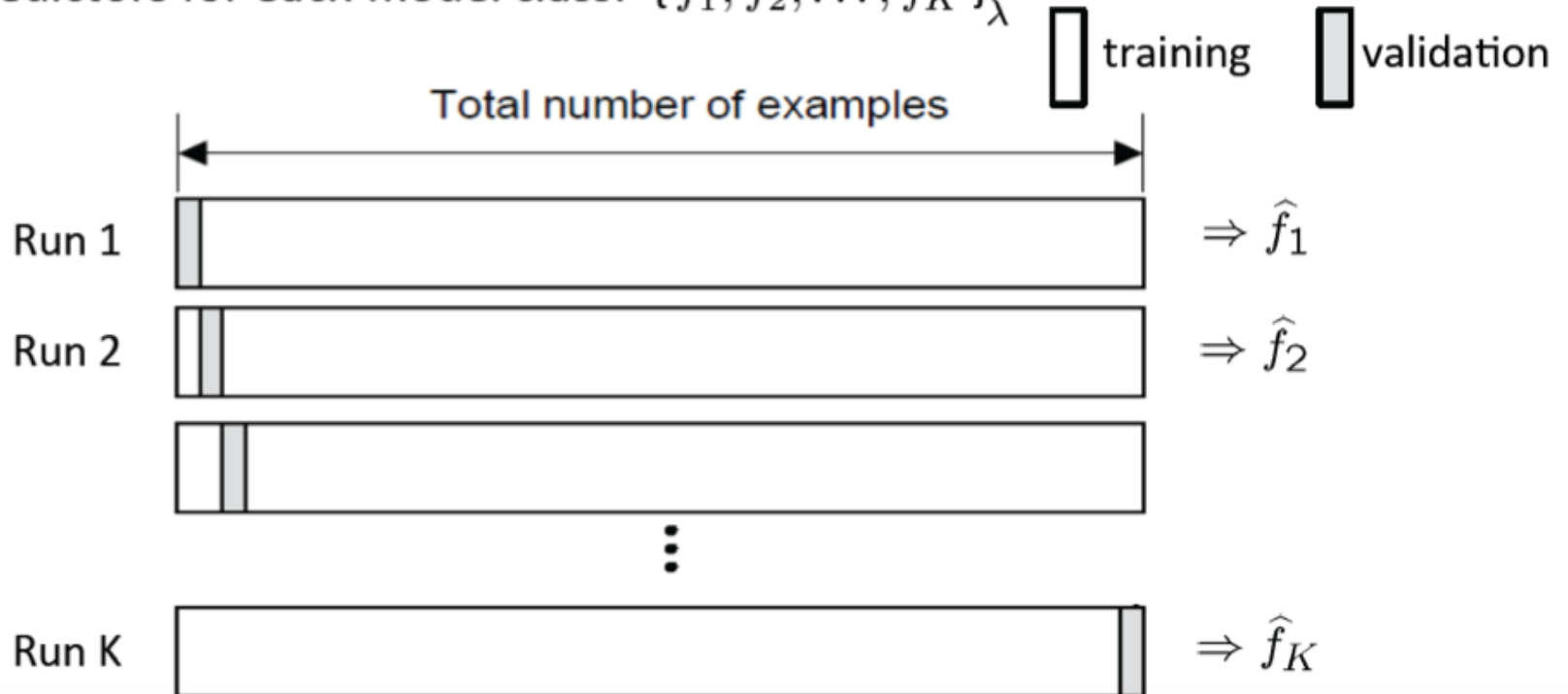


Cross-Validation

Leave-one-out (LOO) cross-validation

- 1) Special case of K-fold with $K=n$ partitions
- 2) Equivalently, train on $n-1$ samples and validate on only one sample per run for n runs

K predictors for each model class: $\{\hat{f}_1, \hat{f}_2, \dots, \hat{f}_K\}_\lambda$



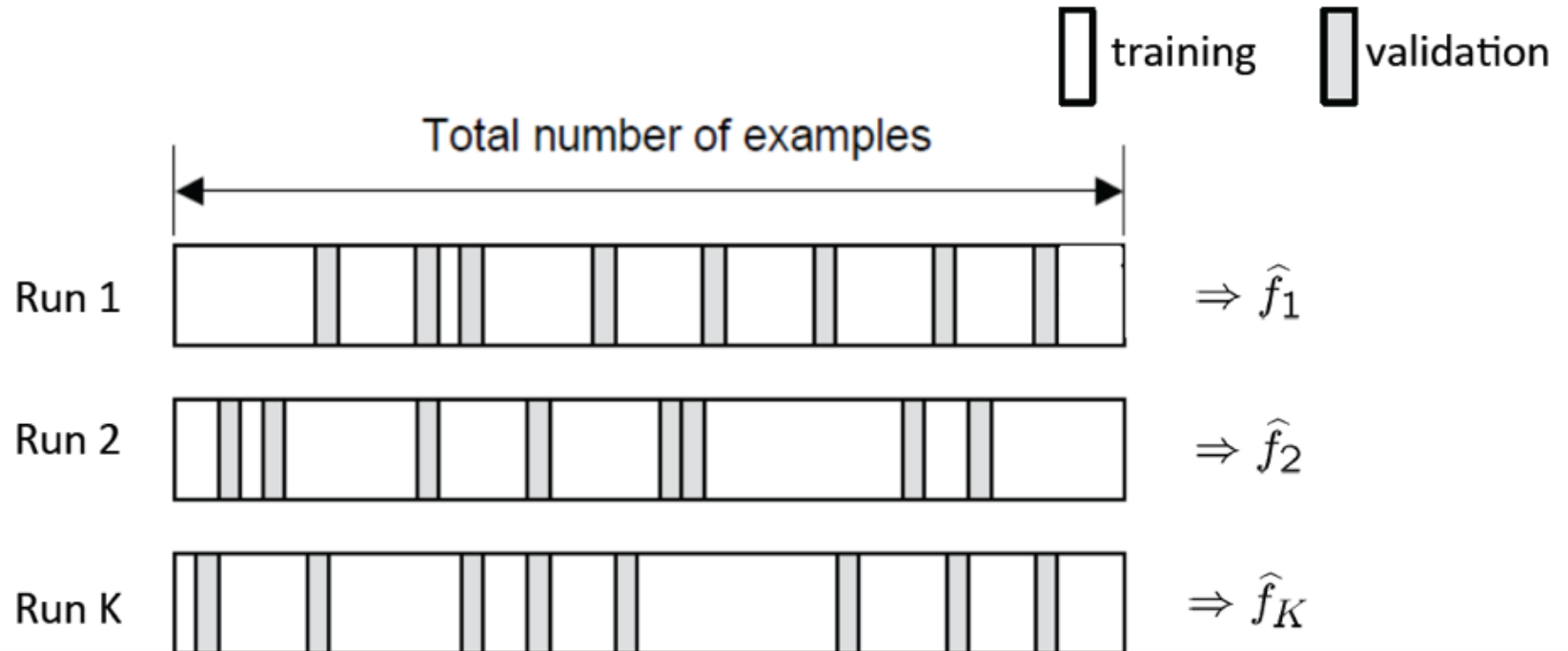
Cross-Validation

Random subsampling

- 1) Randomly subsample a fixed fraction αn ($0 < \alpha < 1$) of the dataset for validation.
- 2) Form hold-out predictor with remaining data as training data.

Repeat K times

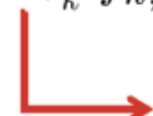
K predictors for each model class: $\{ \hat{f}_1, \hat{f}_2, \dots, \hat{f}_K \}_\lambda$



Model Selection by Cross-Validation

3) Use D_v to select the model class which has smallest empirical error on D_v

$$\hat{\lambda} = \arg \min_{\lambda \in \Lambda} \frac{1}{K} \sum_{k=1}^K \hat{R}_{V_k}(\hat{f}_{k,\lambda})$$

 Evaluated on validation dataset D_v

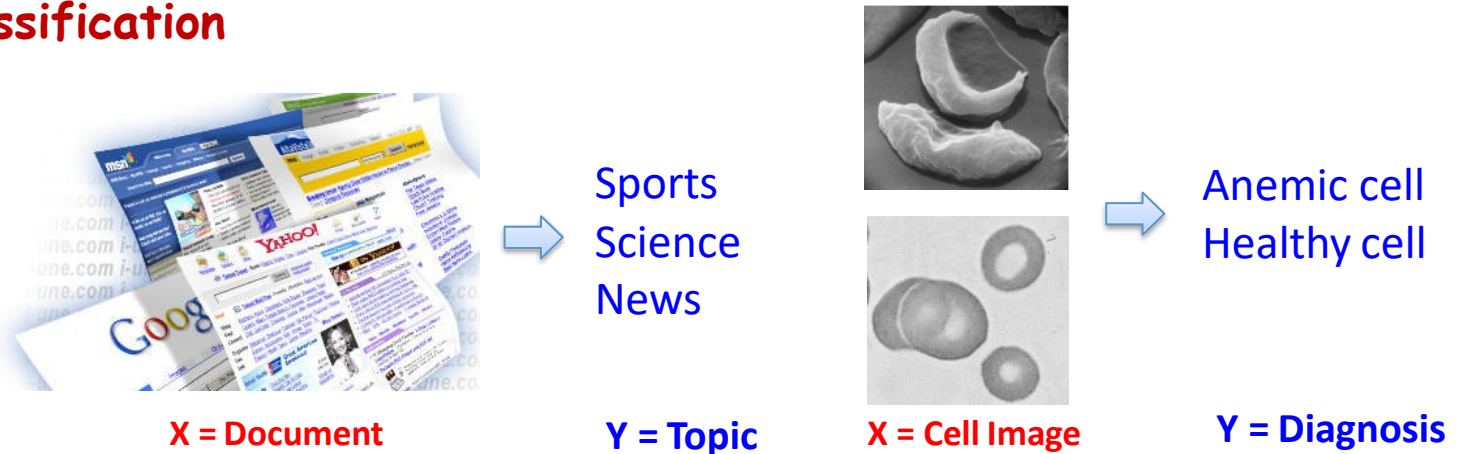
4) Cross-validated predictor

Final predictor \hat{f} is average/majority vote over the K hold-out estimates

$$\{\hat{f}_1, \hat{f}_2, \dots, \hat{f}_K\}_{\hat{\lambda}}$$

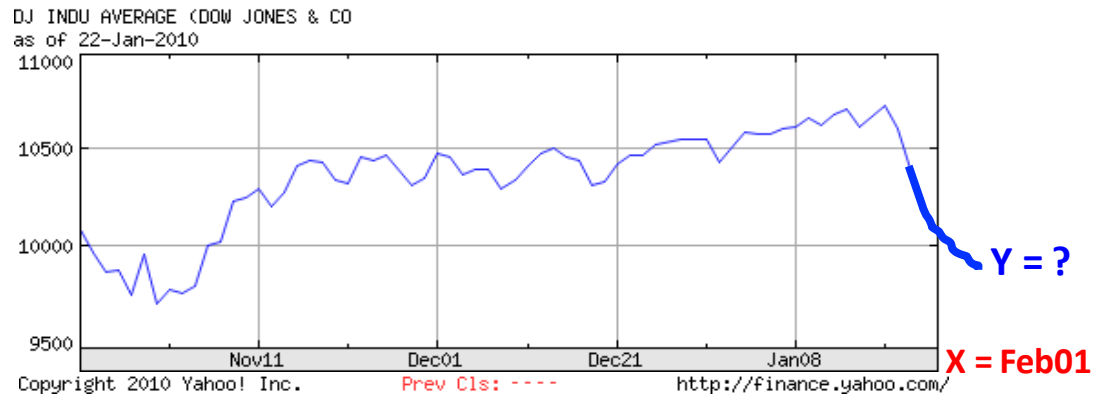
Discrete and Continuous Labels

Classification



Regression

Stock Market Prediction



An example application

- An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- **A decision is needed:** whether to put a new patient in an intensive-care unit.
- Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- **Problem:** to predict **high-risk patients** and discriminate them from **low-risk patients**.

Another application

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,
 - age
 - Marital status
 - annual salary
 - outstanding debts
 - credit rating
 - etc.
- **Problem:** to decide whether an application should be approved, or to classify applications into two categories, **approved** and **not approved**.

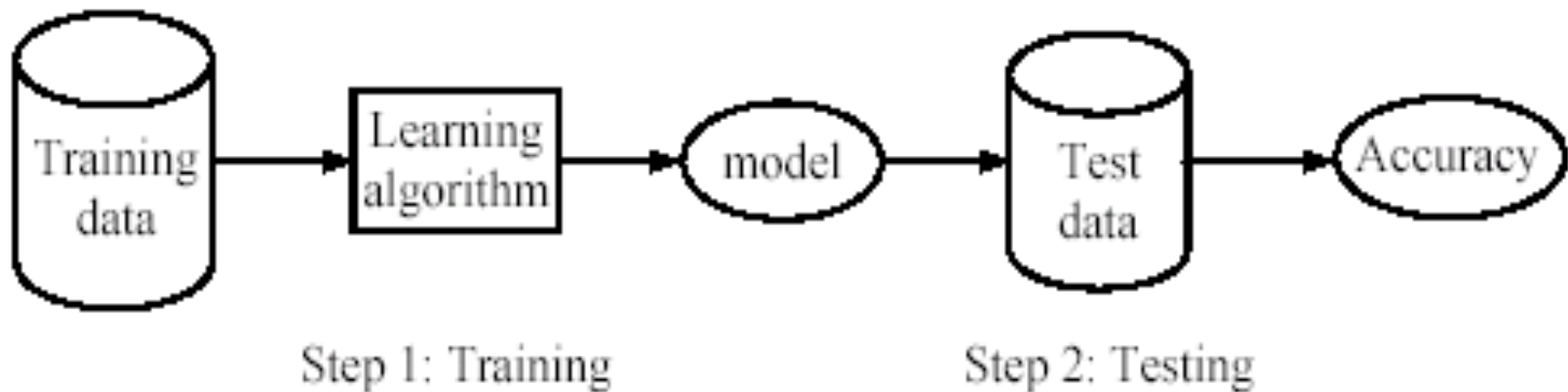
The data and the goal

- **Data:** A set of data records (also called examples, instances or cases) described by
 - *k* attributes: A_1, A_2, \dots, A_k .
 - a class: Each example is labelled with a pre-defined class.
- **Goal:** To learn a **classification model** from the data that can be used to predict the classes of new (future, or test) cases/instances.

Supervised learning process: two steps

- **Learning (training)**: Learn a model using the training data
- **Testing**: Test the model using **unseen test data** to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



Least squares classification

- Binary classification.
- Each class is described by its own linear model:

$$y(x) = w^T x + w_0$$

- Compactly written as:

$$y(\mathbf{x}) = \mathbf{W}^T \mathbf{x}$$

- \mathbf{W} is $[w \ w_0]$.
- $E_D(\mathbf{W}) = 1/2 (\mathbf{XW} - \mathbf{t})^T (\mathbf{XW} - \mathbf{t})$
- n^{th} row of \mathbf{X} is x_n , the n^{th} datapoint.
- \mathbf{t} is vector of +1, -1.

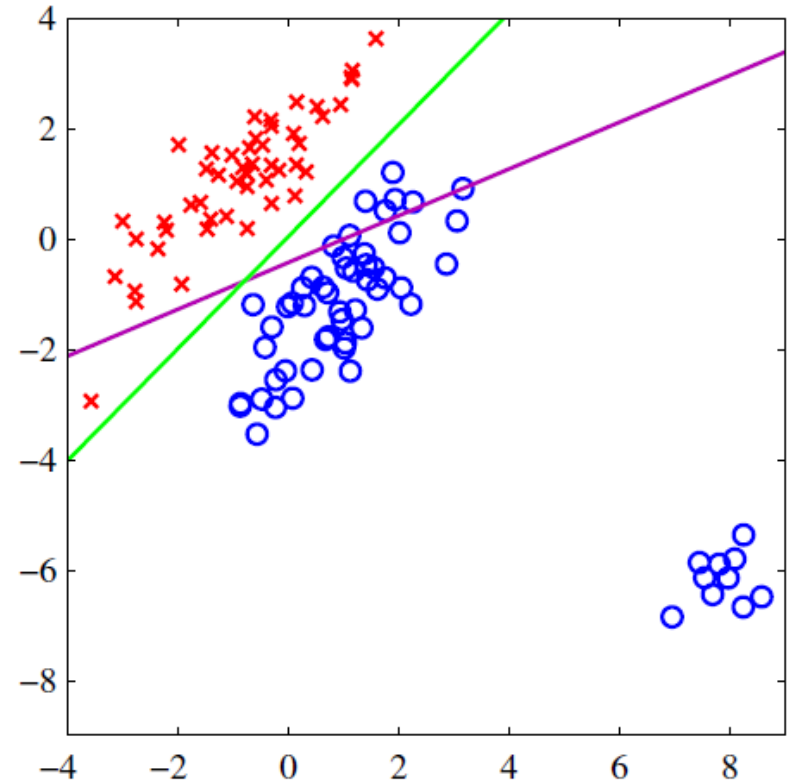
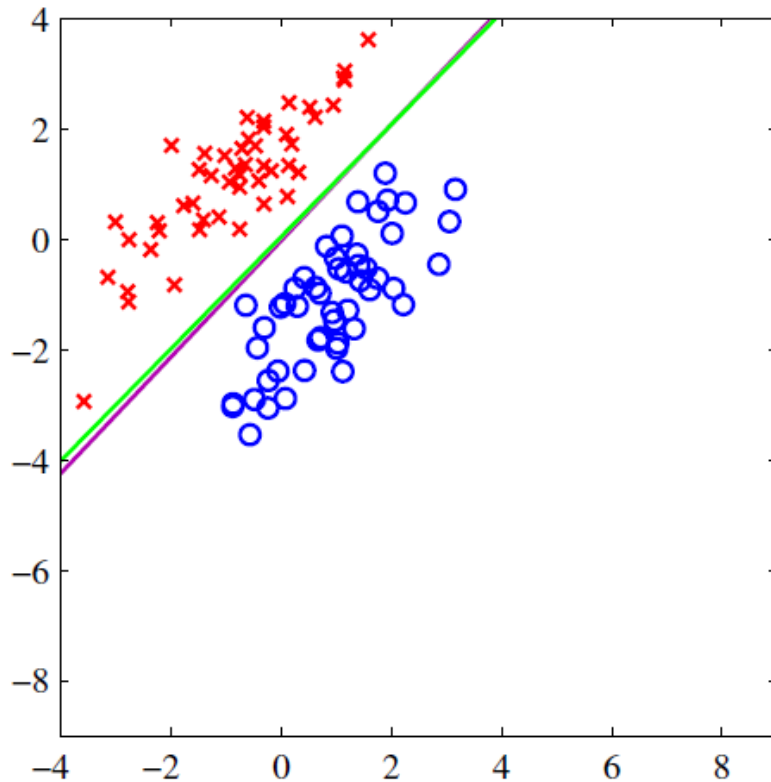
Least squares classification

- Least squares W is:

$$W = (X^T X)^{-1} X^T t$$

- Problem is affected by outliers.

Least squares classification



Fisher's linear discriminant

- Predictor: $y = \mathbf{w}^T \mathbf{x}$.
- If $y > \mathbf{w}_0$ predict C_1 else C_2 .
- Training dataset has N_1 points from C_1 and N_2 points from C_2 .

- $\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{x}_n$ and $\mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{x}_n$

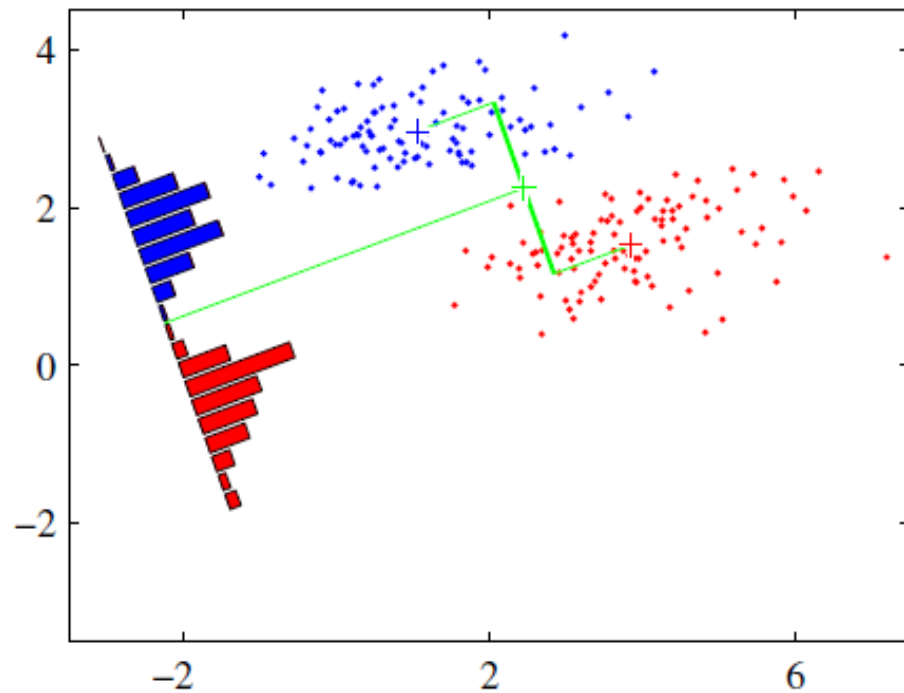
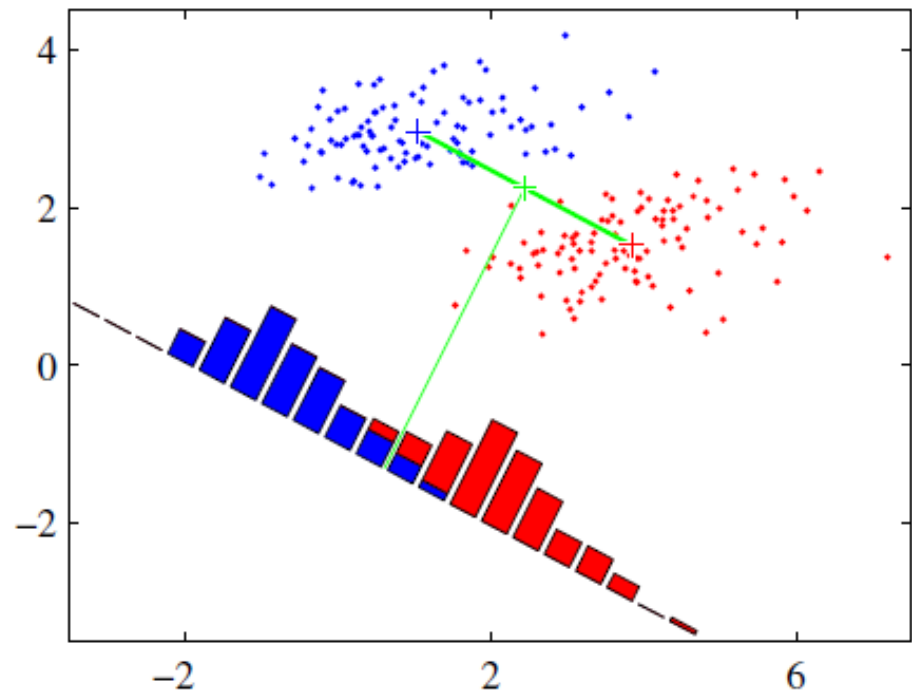
- Maximize separation of projected means:

$$\mathbf{m}_2 - \mathbf{m}_1 = \mathbf{w}^T (\mathbf{m}_2 - \mathbf{m}_1)$$

Fisher's linear discriminant

- This measure can increase arbitrarily by increasing $\|\mathbf{w}\|$.
- Constrain: $\|\mathbf{w}\|^2 = 1$
- Lagrangian: $L(\mathbf{w}, \lambda) = \mathbf{w}^T (\mathbf{m}_2 - \mathbf{m}_1) + \lambda(\|\mathbf{w}\|^2 - 1)$.
- Solution: $\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$.

Fisher linear discriminant



Fisher's linear discriminant

- Maximize separation between means while minimizing within class variance.
- Within class variance:

$$s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2$$

- Objective:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

Fisher's linear Discriminant

- Same as:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

- Between class variance:

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$

- Within class variance:

$$\begin{aligned} & \mathbf{S}_W \\ &= \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T \end{aligned}$$

Fisher's linear discriminant

- Same as:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mathbf{w}^T \mathbf{S}_B \mathbf{w} \\ \text{s. t.} \quad & \mathbf{w}^T \mathbf{S}_W \mathbf{w} = 1 \end{aligned}$$

- Solution given by generalized eigenvalue problem:

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}$$

- Or

$$(\mathbf{S}_W)^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$$

- Solution:

$$\mathbf{w} \propto (\mathbf{S}_W)^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$$

From Linear to Logistic Regression

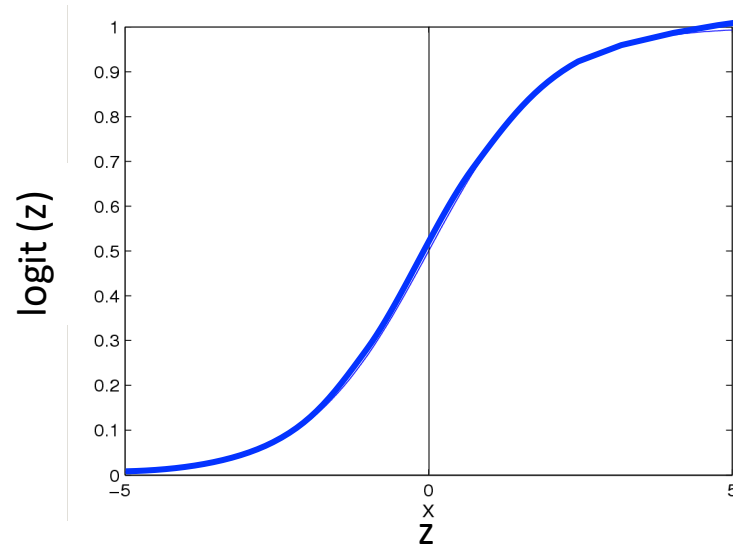
Assumes the following functional form for $P(Y|X)$:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

Logistic function applied to a linear function of the data

Logistic function (or Sigmoid): $\frac{1}{1 + \exp(-z)}$

Features can be discrete or continuous!



Logistic Regression is a Linear Classifier!

Assumes the following functional form for $P(Y|X)$:

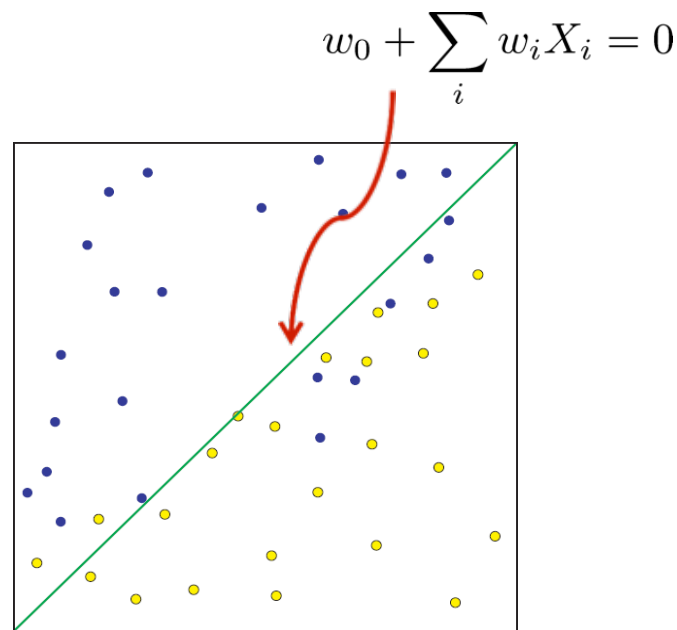
$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

Decision boundary:

$$P(Y = 0|X) \underset{1}{\overset{0}{\geq}} P(Y = 1|X)$$

$$w_0 + \sum_i w_i X_i \underset{1}{\overset{0}{\geq}} 0$$

(Linear Decision Boundary)



Logistic Regression is a Linear Classifier!

Assumes the following functional form for $P(Y|X)$:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$\Rightarrow P(Y = 0|X) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$\Rightarrow \frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i) \stackrel{0}{\geq} \stackrel{1}{1}$$

$$\Rightarrow w_0 + \sum_i w_i X_i \stackrel{0}{\geq} \stackrel{1}{0}$$

Logistic Regression

- Label $t \in \{+1, -1\}$ modeled as:

$$P(t = 1|x, w) = \sigma(w^T x)$$

- $P(y|x, w) = \sigma(yw^T x), y \in \{+1, -1\}$

- Given a set of parameters w , the probability or likelihood of a datapoint (x, t) :

$$P(t|x, w) = \sigma(tw^T x)$$

Logistic Regression

- Given a training dataset $\{(x_1, t_1), \dots, (x_N, t_N)\}$, log likelihood of a model w is given by:

$$L(w) = \sum_n \ln(P(t_n | x_n, w))$$

- Using principle of maximum likelihood, the best w is given by:
 $w^* = \arg \max_w L(w)$

Logistic Regression

- Final Problem:

$$\max_w \sum_{i=1}^n -\log(1 + \exp(-t_n w^T x_n))$$

Or, $\min_w \sum_{i=1}^n \log(1 + \exp(-t_n w^T x_n))$

- Error function:

$$E(w) = \sum_{i=1}^n \log(1 + \exp(-t_n w^T x_n))$$

- $E(w)$ is convex.

Properties of Error function

- Derivatives:

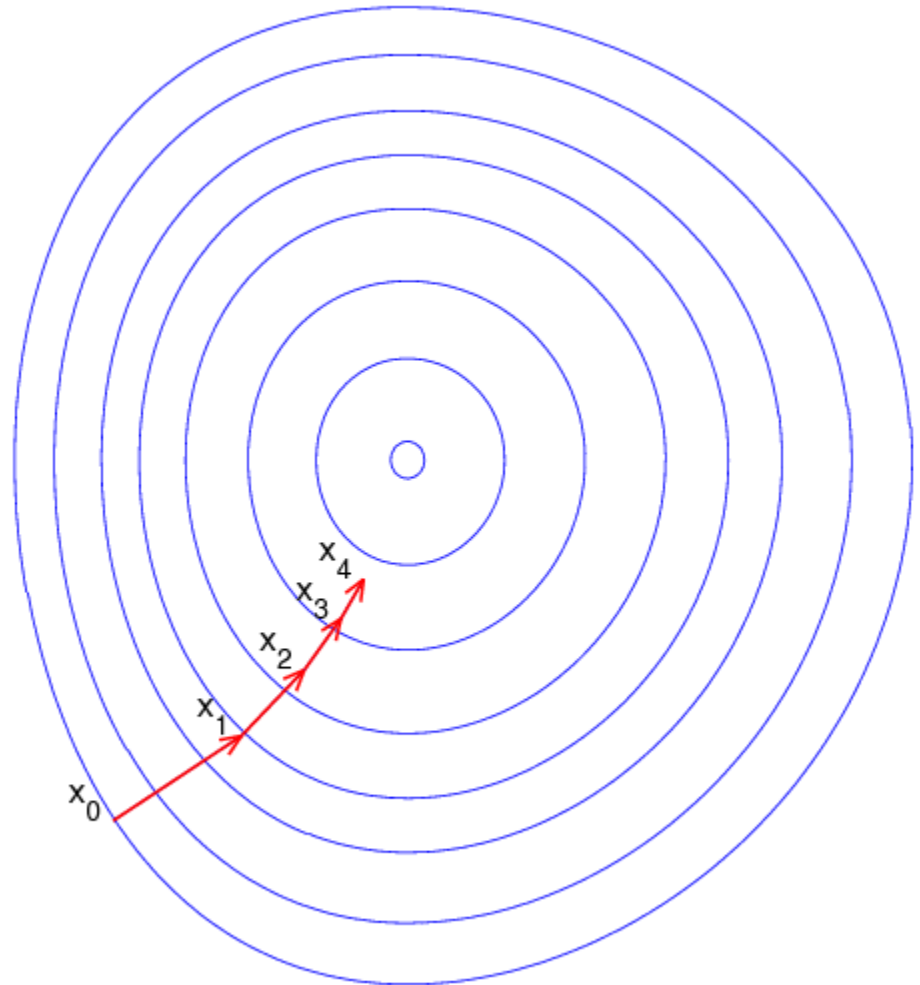
$$\nabla E(w) = \sum_{i=1}^n -(1 - \sigma(t_i w^T x_i))(t_i x_i)$$

$$\nabla E(w) = \sum_{i=1}^n (\sigma(w^T x_i) - t_i) x_i$$

$$\nabla^2 E(w) = \sum_{i=1}^n \sigma(t_i w^T x_i)(1 - \sigma(t_i w^T x_i)) x_i x_i^T$$

Gradient Descent

- Problem: $\min f(x)$
- $f(x)$: differentiable
- $g(x)$: gradient of $f(x)$
- Negative gradient is steepest descent direction.
- At each step move in the gradient direction so that there is “sufficient decrease”.



Gradient Descent

input : Function f , Gradient ∇f

output: Optimal solution w^*

Initialize $w_0 \leftarrow 0, k \leftarrow 0$

while $|\nabla f_k| > \epsilon$ **do**

 Compute $\alpha_k \leftarrow \text{linesearch}(f, -\nabla f_k, w_k)$

 Set $w_{k+1} \leftarrow w_k - \alpha_k \nabla f_k$

 Evaluate ∇f_{k+1}

$k \leftarrow k + 1$

end

$w^* \leftarrow w_k$

Generative vs. Discriminative Classifiers

Discriminative classifiers (e.g. **Logistic Regression**)

- Assume some functional form for $P(Y|X)$ or for the decision boundary
- Estimate parameters of $P(Y|X)$ directly from training data

Generative classifiers (e.g. **Naïve Bayes**)

- Assume some functional form for $P(X,Y)$ (or $P(X|Y)$ and $P(Y)$)
- Estimate parameters of $P(X|Y)$, $P(Y)$ directly from training data

$$\arg \max_Y P(Y|X) = \arg \max_Y P(X|Y) P(Y)$$

Logistic Regression is a Linear Classifier!

Assumes the following functional form for $P(Y|X)$:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$\Rightarrow P(Y = 0|X) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$\Rightarrow \frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i) \stackrel{0}{\geq} \stackrel{1}{1}$$

$$\Rightarrow w_0 + \sum_i w_i X_i \stackrel{0}{\geq} \stackrel{1}{0}$$

Logistic Regression for more than 2 classes

- Logistic regression in more general case, where
 $Y \in \{y_1, \dots, y_K\}$

for $k < K$

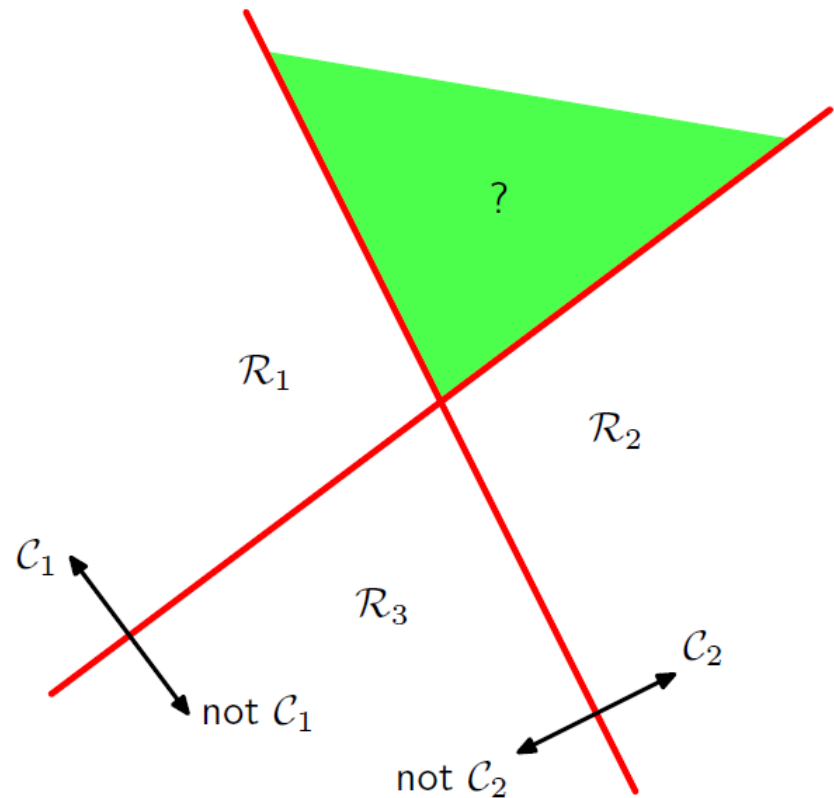
$$P(Y = y_k | X) = \frac{\exp(w_{k0} + \sum_{i=1}^d w_{ki} X_i)}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^d w_{ji} X_i)}$$

for $k=K$ (normalization, so no weights for this class)

$$P(Y = y_K | X) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^d w_{ji} X_i)}$$

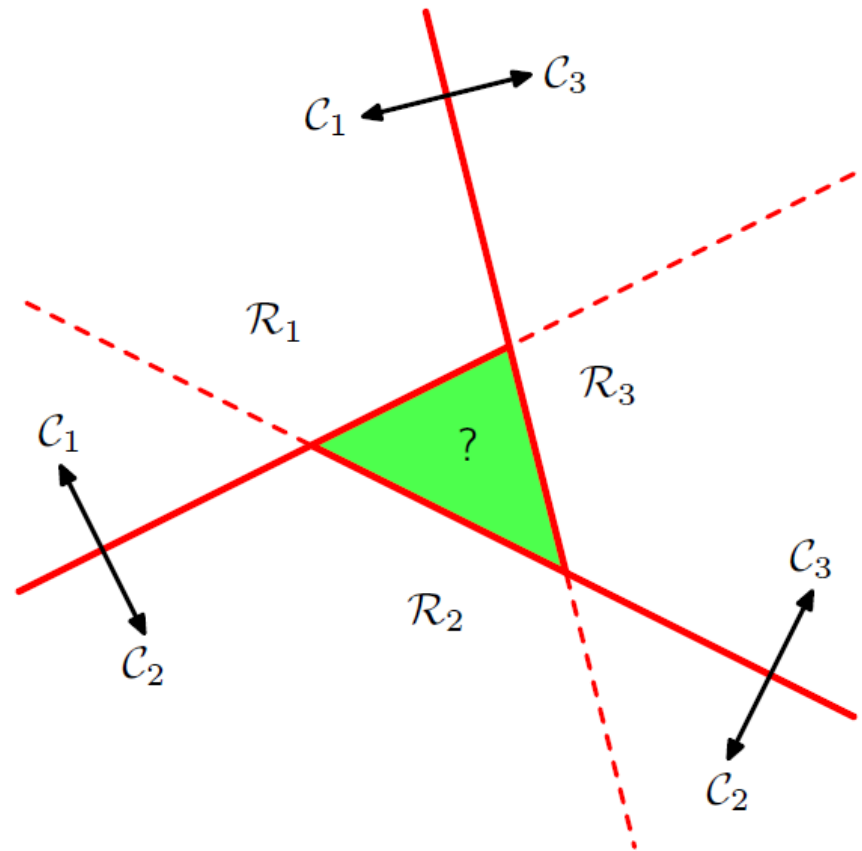
Multiple classes

- One-vs-all: $K - 1$ hyperplanes each separating C_1, \dots, C_{K-1} classes from rest.
- Otherwise C_K
- Low number of classifiers.



Multiple classes

- One-vs-one: Every pair $C_i - C_j$ get a boundary.
- Final by majority vc
- High number of classifiers.



Multiple classes

- K-linear discriminant functions:

$$y_k(x) = w_k^T x + w_{k0}$$

- Assign x to C_k if $y_k(x) \geq y_j(x)$ for all $j \neq k$
- Decision boundary:

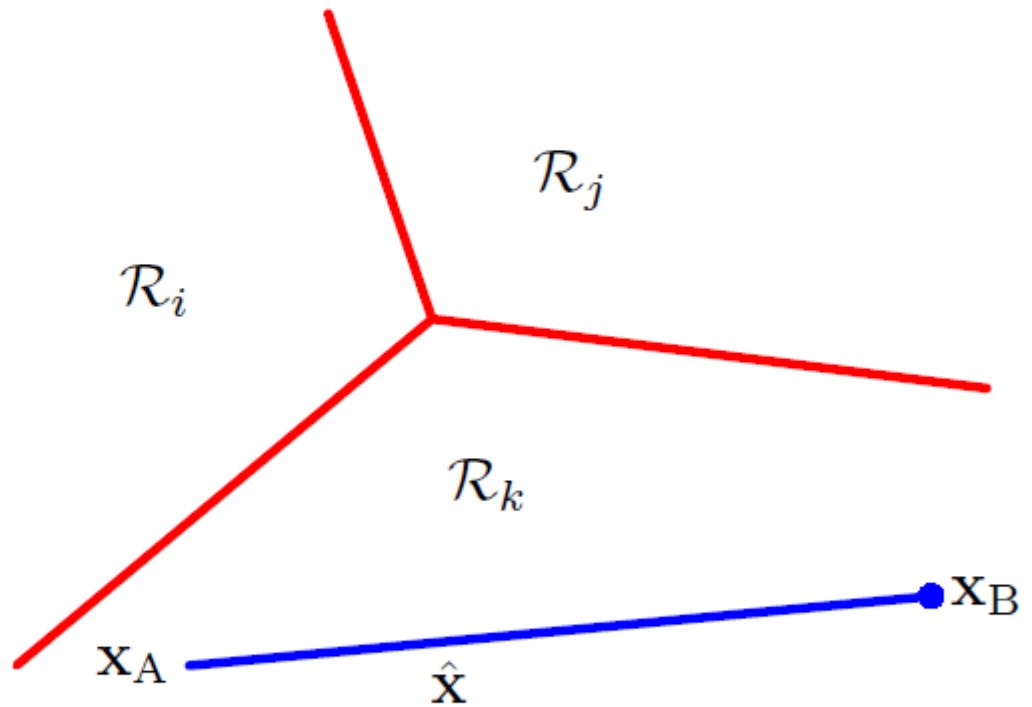
$$(w_k - w_j)^T x + (w_{k0} - w_{j0}) = 0$$

- Decision region is singly connected:

$$x = \lambda x_A + (1 - \lambda)x_B$$

- If x_A and x_B have same label, so does x .

Multiple Classes



A text classification task: Email spam filtering

From: '''' <takworl1d@hotmail.com>
Subject: real estate is the only way... gem oalvgkay
Anyone can buy real estate with no money down
Stop paying rent TODAY !
There is no need to spend hundreds or even thousands for
similar courses
I am 22 years old and I have already purchased 6 properties
using the
methods outlined in this truly INCREDIBLE ebook.
Change your life NOW !

=====
Click Below to order:

<http://www.wholesaledaily.com/sales/nmd.htm>
=====

How would you write a program that would automatically detect
and delete this type of message?

Formal definition of TC: Training

Given:

- A **document set** X
 - Documents are represented typically in some type of high-dimensional space.
- A fixed set of **classes** $C = \{c_1, c_2, \dots, c_j\}$
 - The classes are human-defined for the needs of an application (e.g., relevant vs. nonrelevant).
- A **training set** D of labeled documents with each labeled document $\langle d, c \rangle \in X \times C$

Using a learning method or **learning algorithm**, we then wish to

learn a **classifier** γ that maps documents to classes:

$$\gamma : X \rightarrow C$$

Formal definition of TC: Application/Testing

Given: a description $d \in X$ of a document Determine: $\Upsilon(d) \in C$,
that is, the class that is most appropriate for d

Examples of how search engines use classification

- Language identification (classes: English vs. French etc.)
- The automatic detection of spam pages (spam vs. nonspam)
- Topic-specific or *vertical* search – restrict search to a “vertical” like “related to health” (relevant to vertical vs. not)

Derivation of Naive Bayes rule

We want to find the class that is most likely given the document:

$$C_{\text{map}} = \arg \max_{c \in \mathbb{C}} P(c|d)$$

Apply Bayes rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}:$$

$$C_{\text{map}} = \arg \max_{c \in \mathbb{C}} \frac{P(d|c)P(c)}{P(d)}$$

Drop denominator since $P(d)$ is the same for all classes:

$$C_{\text{map}} = \arg \max_{c \in \mathbb{C}} P(d|c)P(c)$$

Too many parameters / sparseness

$$\begin{aligned}c_{\text{map}} &= \arg \max_{c \in \mathbb{C}} P(d|c)P(c) \\ &= \arg \max_{c \in \mathbb{C}} P(\langle t_1, \dots, t_k, \dots, t_{n_d} \rangle | c)P(c)\end{aligned}$$

- There are too many parameters $P(\langle t_1, \dots, t_k, \dots, t_{n_d} \rangle | c)$, one for each unique combination of a class and a sequence of words.
- We would need a very, very large number of training examples to estimate that many parameters.
- This is the problem of [data sparseness](#).

Naive Bayes conditional independence assumption

To reduce the number of parameters to a manageable size, we make the **Naive Bayes conditional independence assumption**:

$$P(d|c) = P(\langle t_1, \dots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

We assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(X_k = t_k | c)$.

The Naive Bayes classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document d being in a class c as follows:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- n_d is the length of the document. (number of tokens)
- $P(t_k | c)$ is the conditional probability of term t_k occurring in a document of class c
- $P(t_k | c)$ is a measure of **how much evidence** t_k contributes that c is the correct class.
- $P(c)$ is the prior probability of c .
- If a document's terms do not provide clear evidence for one class vs. another, we choose the c with highest $P(c)$.

Maximum a posteriori class

- Our goal in Naive Bayes classification is to find the “best” class.
- The best class is the most likely or maximum a posteriori (MAP) class c_{map} :

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg \max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

Taking the log

- Multiplying lots of small probabilities can result in floating point underflow.
 - Since $\log(xy) = \log(x) + \log(y)$, we can sum log probabilities instead of multiplying probabilities.
 - Since log is a monotonic function, the class with the highest score does not change.
-
- So what we usually compute in practice is:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k | c)]$$

Naive Bayes classifier

- Classification rule:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

- Simple interpretation:

- Each conditional parameter $\log \hat{P}(t_k|c)$ is a weight that indicates how good an indicator t_k is for c .
- The prior $\log \hat{P}(c)$ is a weight that indicates the relative frequency of c .
- The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
- We select the class with the most evidence.

Parameter estimation take 1: Maximum likelihood

- Estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from train data: How?

- Prior:

$$\hat{P}(c) = \frac{N_c}{N}$$

- N_c : number of docs in class c ; N : total number of docs

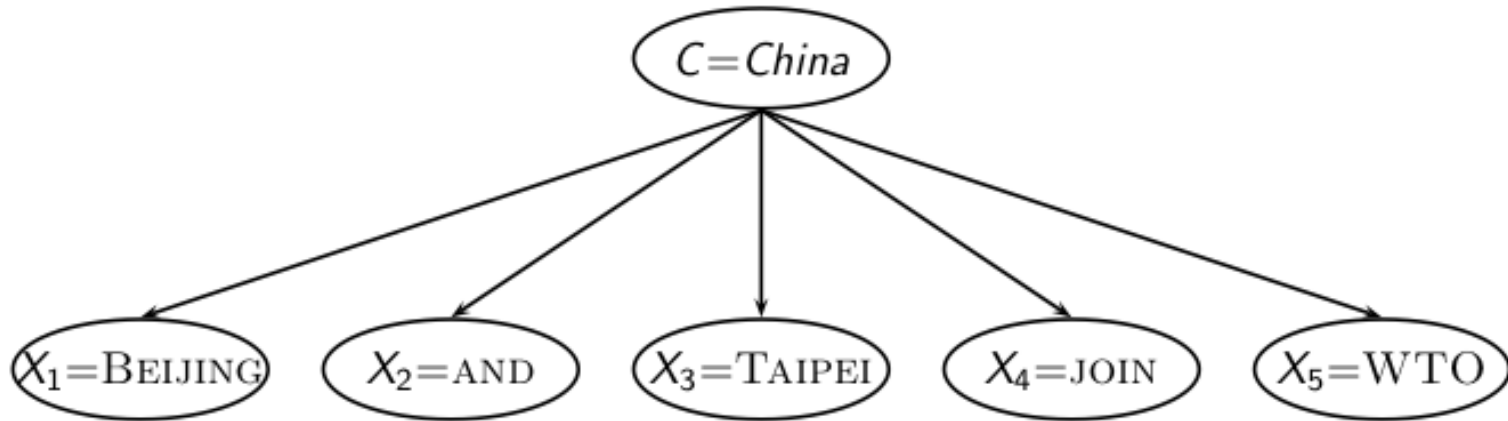
- Conditional probabilities:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- T_{ct} is the number of tokens of t in training documents from class c (includes multiple occurrences)

- We've made a [Naive Bayes independence assumption](#) here:

The problem with maximum likelihood estimates: Zeros



$$P(\text{China} | d) \propto P(\text{China}) \cdot P(\text{BEIJING} | \text{China}) \cdot P(\text{AND} | \text{China}) \\ \cdot P(\text{TAIPEI} | \text{China}) \cdot P(\text{JOIN} | \text{China}) \cdot P(\text{WTO} | \text{China})$$

- If WTO never occurs in class China in the train set:

$$\hat{P}(\text{WTO} | \text{China}) = \frac{T_{\text{China}, \text{WTO}}}{\sum_{t' \in V} T_{\text{China}, t'}} = \frac{0}{\sum_{t' \in V} T_{\text{China}, t'}} = 0$$

The problem with maximum likelihood estimates: Zeros (cont)

- If there were no occurrences of WTO in documents in class China, we'd get a zero estimate:

$$\hat{P}(\text{WTO} | \text{China}) = \frac{T_{\text{China}, \text{WTO}}}{\sum_{t' \in V} T_{\text{China}, t'}} = 0$$

- → We will get $P(\text{China} | d) = 0$ for any document that contains WTO!
- Zero probabilities cannot be conditioned away.

To avoid zeros: Add-one smoothing

- Before:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- Now: Add one to each count to avoid zeros:

- B is the number of different words (in this case the size of the vocabulary: $|V| = B$)

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

To avoid zeros: Add-one smoothing

- Estimate parameters from the training corpus using add-one smoothing
- For a new document, for each class, compute sum of (i) log of prior and (ii) logs of conditional probabilities of the terms
- Assign the document to the class with the largest score

Exercise

	docID	words in document	in $c = \textit{China}$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

- Estimate parameters of Naive Bayes classifier
- Classify test document

Example: Parameter estimates

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ Conditional probabilities:

$$\begin{aligned}\hat{P}(\text{CHINESE}|c) &= (5 + 1)/(8 + 6) = 6/14 = 3/7 \\ \hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) &= (0 + 1)/(8 + 6) = 1/14 \\ \hat{P}(\text{CHINESE}|\bar{c}) &= (1 + 1)/(3 + 6) = 2/9 \\ \hat{P}(\text{TOKYO}|\bar{c}) = \hat{P}(\text{JAPAN}|\bar{c}) &= (1 + 1)/(3 + 6) = 2/9\end{aligned}$$

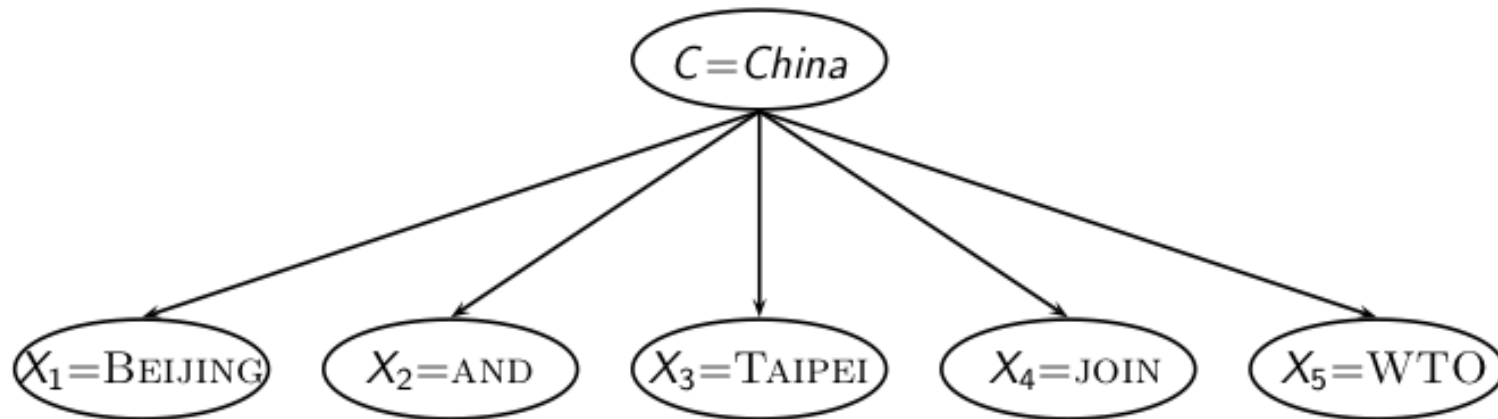
The denominators are $(8 + 6)$ and $(3 + 6)$ because the lengths of text_c and $\text{text}_{\bar{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Classification

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$
$$\hat{P}(\bar{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

Thus, the classifier assigns the test document to $c = \textit{China}$. The reason for this classification decision is that the three occurrences of the positive indicator CHINESE in d_5 outweigh the occurrences of the two negative indicators JAPAN and TOKYO.

Generative model



$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- Generate a class with probability $P(c)$
- Generate each of the words (in their respective positions), conditional on the class, but independent of each other, with probability $P(t_k | c)$
- To classify docs, we “reengineer” this process and find the class that is most likely to have generated the doc.

On naïve Bayesian classifier

- **Advantages:**
 - Easy to implement
 - Very efficient
 - Good results obtained in many applications
- **Disadvantages**
 - Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated (those highly correlated data sets)

NON-PARAMETRIC MODELS

Instance-Based Classifiers

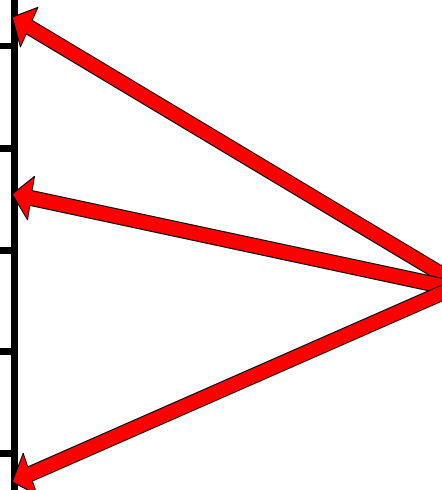
Set of Stored Cases

Atr1	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

- Store the training records
- Use training records to predict the class label of unseen cases

Unseen Case

Atr1	AtrN

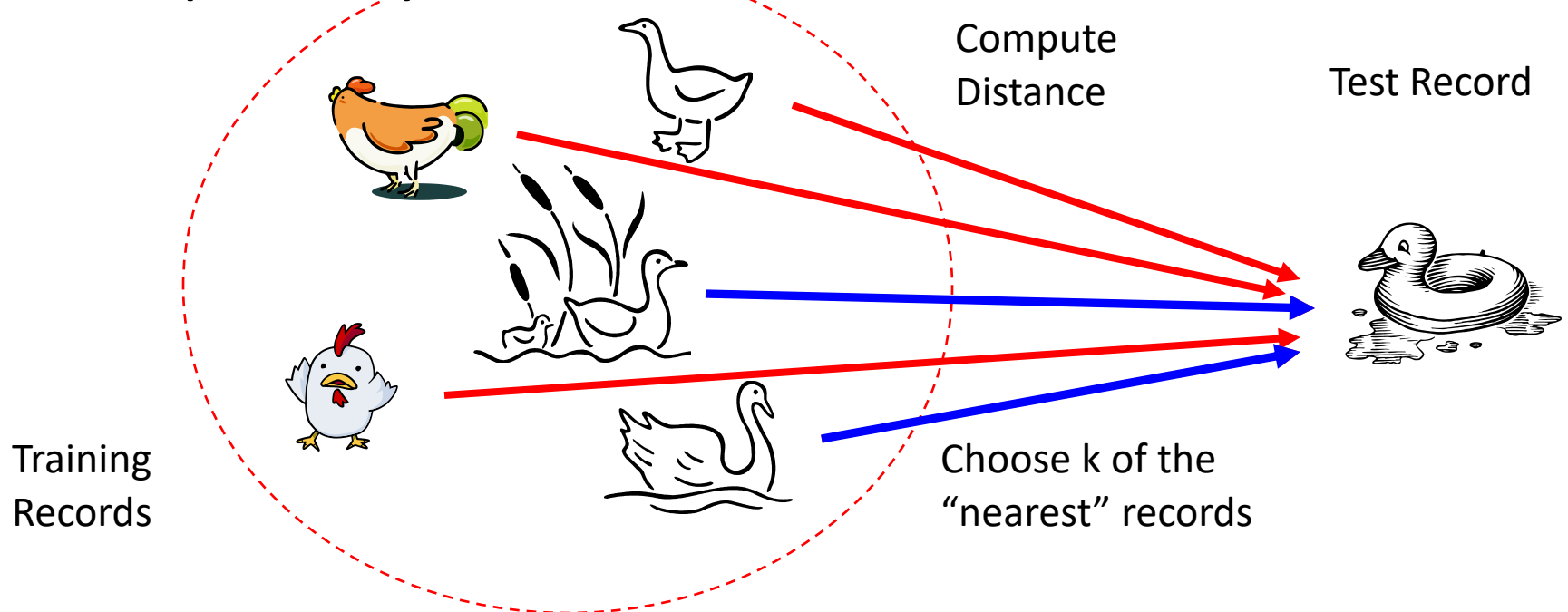


Instance Based Classifiers

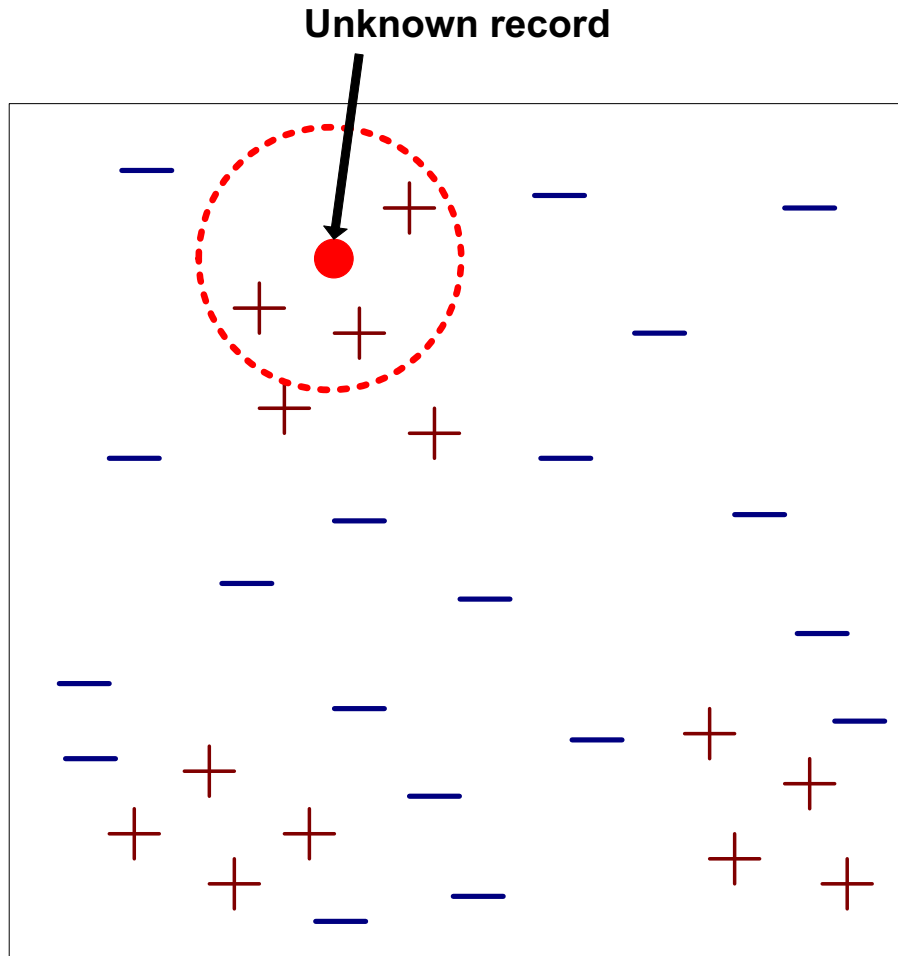
- Examples:
 - Rote-learner
 - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
 - Nearest neighbor
 - Uses k “closest” points (nearest neighbors) for performing classification

Nearest Neighbor Classifiers

- Basic idea:
 - If it walks like a duck, quacks like a duck, then it's probably a duck

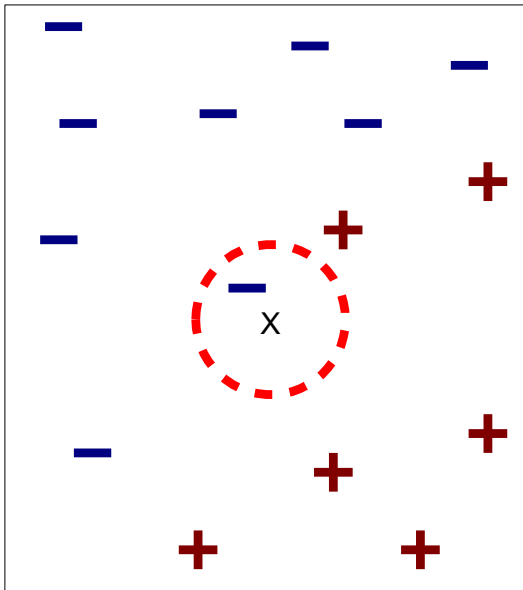


Nearest-Neighbor Classifiers

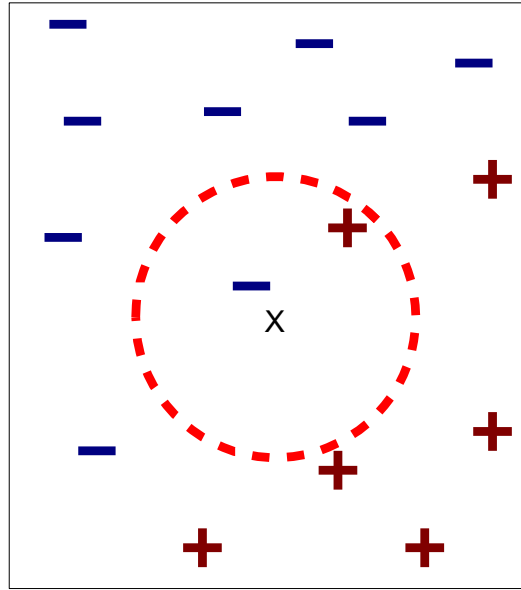


- Requires three things
 - The set of stored records
 - Distance Metric to compute distance between records
 - The value of k , the number of nearest neighbors to retrieve
- To classify an unknown record:
 - Compute distance to other training records
 - Identify k nearest neighbors
 - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

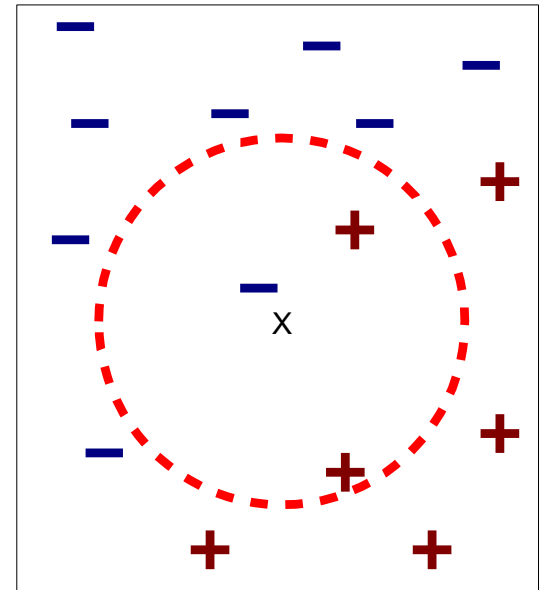
Definition of Nearest Neighbor



(a) 1-nearest neighbor



(b) 2-nearest neighbor



(c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Nearest Neighbor Classification

- Compute distance between two points:

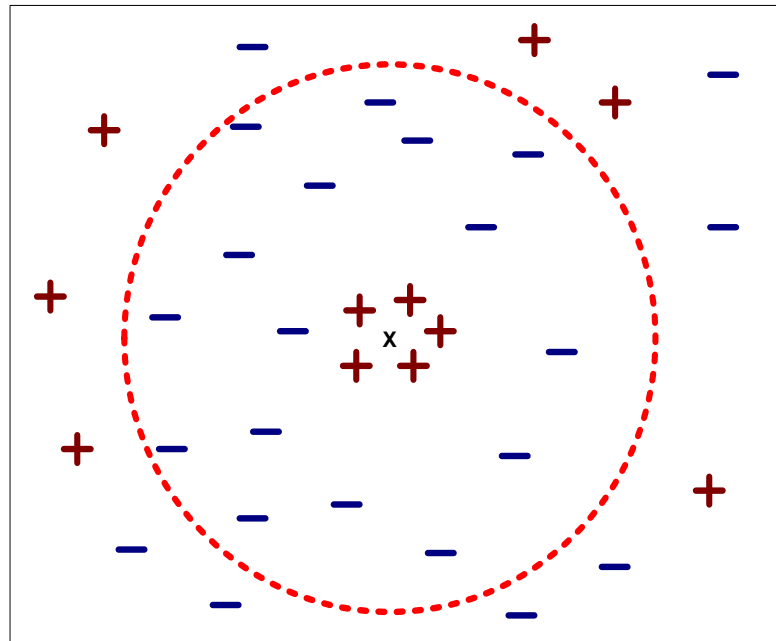
- Euclidean distance

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
 - take the majority vote of class labels among the k-nearest neighbors
 - Weigh the vote according to distance
 - weight factor, $w = 1/d^2$

Nearest Neighbor Classification...

- Choosing the value of k :
 - If k is too small, sensitive to noise points
 - If k is too large, neighborhood may include points from other classes



Nearest Neighbor Classification...

- Scaling issues
 - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
 - Example of an attribute dominating distance computation:
 - height of a person may vary from 1.5m to 1.8m
 - weight of a person may vary from 90lb to 300lb
 - income of a person may vary from \$10K to \$1M

The K-Nearest Neighbor Method

- K neighbors in training data to the input data x : break ties arbitrarily
- All k neighbors will vote: majority wins
- Extension:
 - Weighted KNN

“K” is a variable:

- Often we experiment with different values of $K=1, 3, 5$, to find out the optimal one
- Why is KNN important?
 - Often a baseline
 - Must beat this one to claim innovation
- Applications of KNN
 - Document similarity

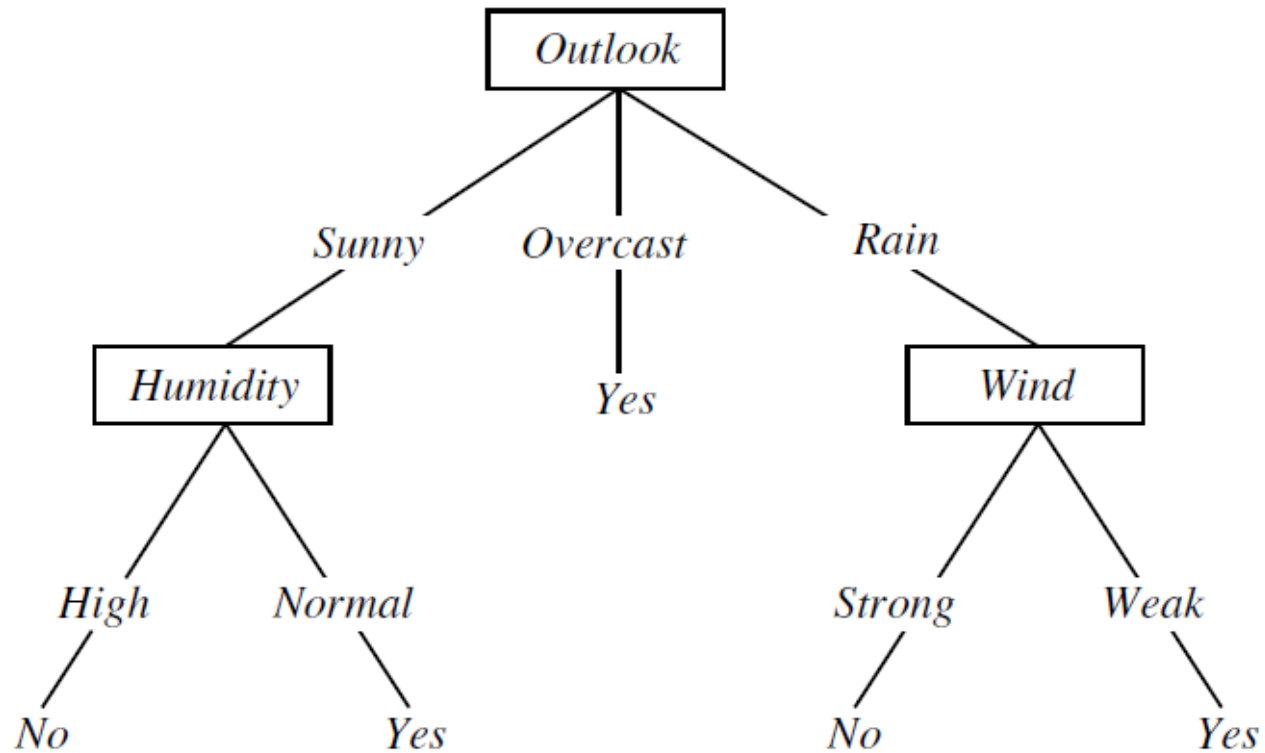
Decision Tree

- Setting:
 - Instances describable by attribute value pair.
 - Target function is discrete valued.
 - Disjunctive hypothesis is required.
 - Training data may contain errors.
 - Training data may contain missing attribute values.

Decision Tree

- Each internal node tests an attribute.
- Each branch corresponds to a value of an attribute.
- Each leaf node assigns a classification.

Decision tree - Example



Decision tree learning

- For a test datapoint:
 - Determine the branches to take at each level based on attribute value.
 - At a leaf level, return the label of current node.
- For training, at each level:
 - Select an attribute split the training dataset on.
 - Examine the purity of data at each splitted node and determine whether it is a leaf node.
 - Determine the label at each node

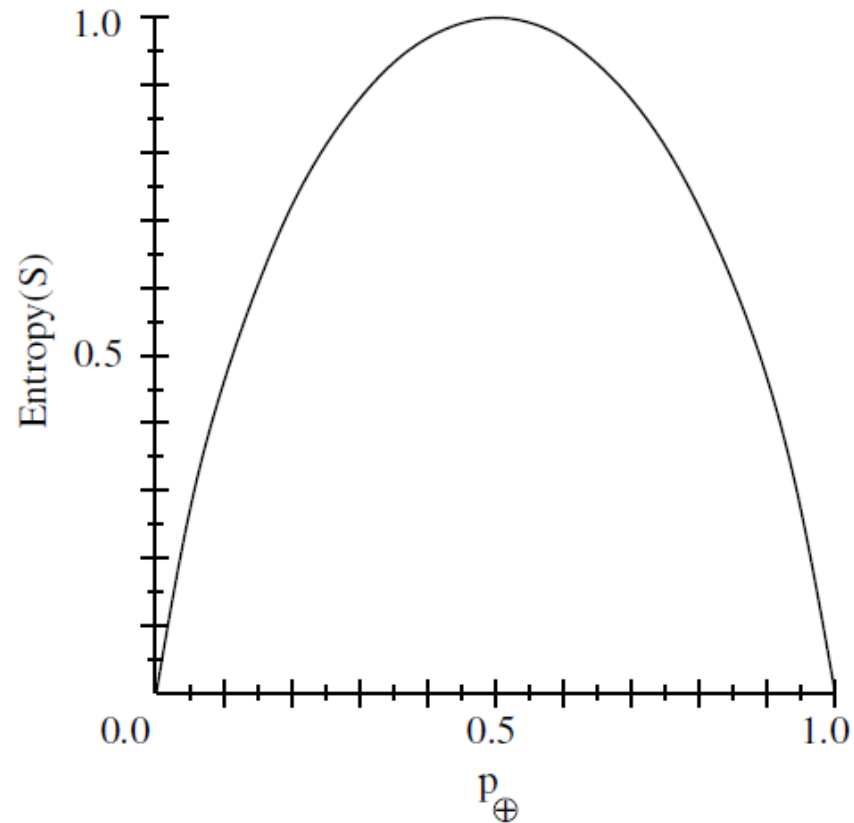
What attribute to select ?

- Entropy:
- S is a sample of training examples
- p_{\oplus} is the proportion of positive examples in S
- p_{\ominus} is the proportion of negative examples in S
- Entropy measures the impurity of S

$$\text{Entropy}(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

What attribute to select ?

- Entropy:



What attribute to select ?

Entropy(S) = expected number of bits needed to encode class (\oplus or \ominus) of randomly drawn member of S (under the optimal, shortest-length code)

Why?

Information theory: optimal length code assigns $-\log_2 p$ bits to message having probability p .

So, expected number of bits to encode \oplus or \ominus of random member of S :

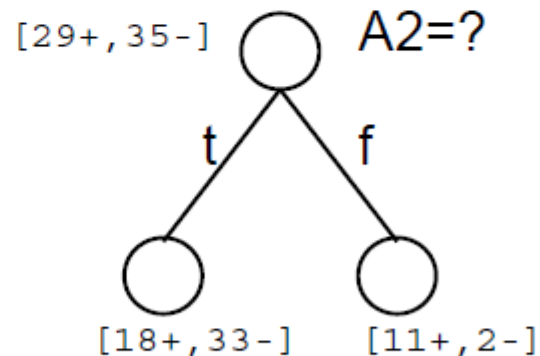
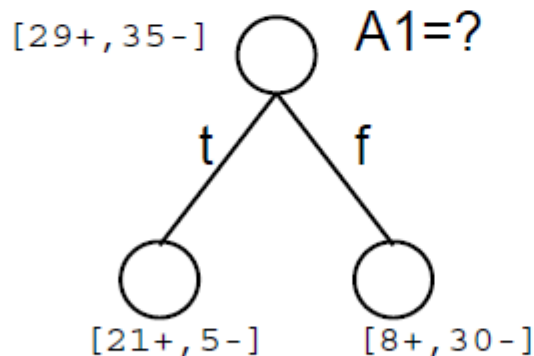
$$p_{\oplus}(-\log_2 p_{\oplus}) + p_{\ominus}(-\log_2 p_{\ominus})$$

$$\textit{Entropy}(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Information Gain

$Gain(S, A) =$ expected reduction in entropy due to sorting on A

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$



ID3 Algorithm

```
function ID3 (R: attributes, S: a training set, default_value) returns a
    decision tree;
begin
    if S= $\emptyset$ , then return default_value;
    else if S consists of records all with the value  $v$  then return
        value  $v$ ;
    else if R= $\emptyset$ , then return the most frequent value  $v$  for records of
        S;
    else
        let A be the attribute with largest Gain(A, S) among attributes in R;
        let { $a_j$  |  $j=1, 2, \dots, m$ } be the values of attribute A;
        let { $S_j$  |  $j=1, 2, \dots, m$ } be the subsets of S consisting respectively of
            records with value  $a_j$  for A;
        return a tree with root labeled A and arcs labeled  $a_1, a_2, \dots, a_m$ 
            going respectively to the trees (ID3(R-{A},  $S_1$ ), ID3(R-{A},  $S_2$ ),
            ....., ID3(R-{A},  $S_m$ ));
```

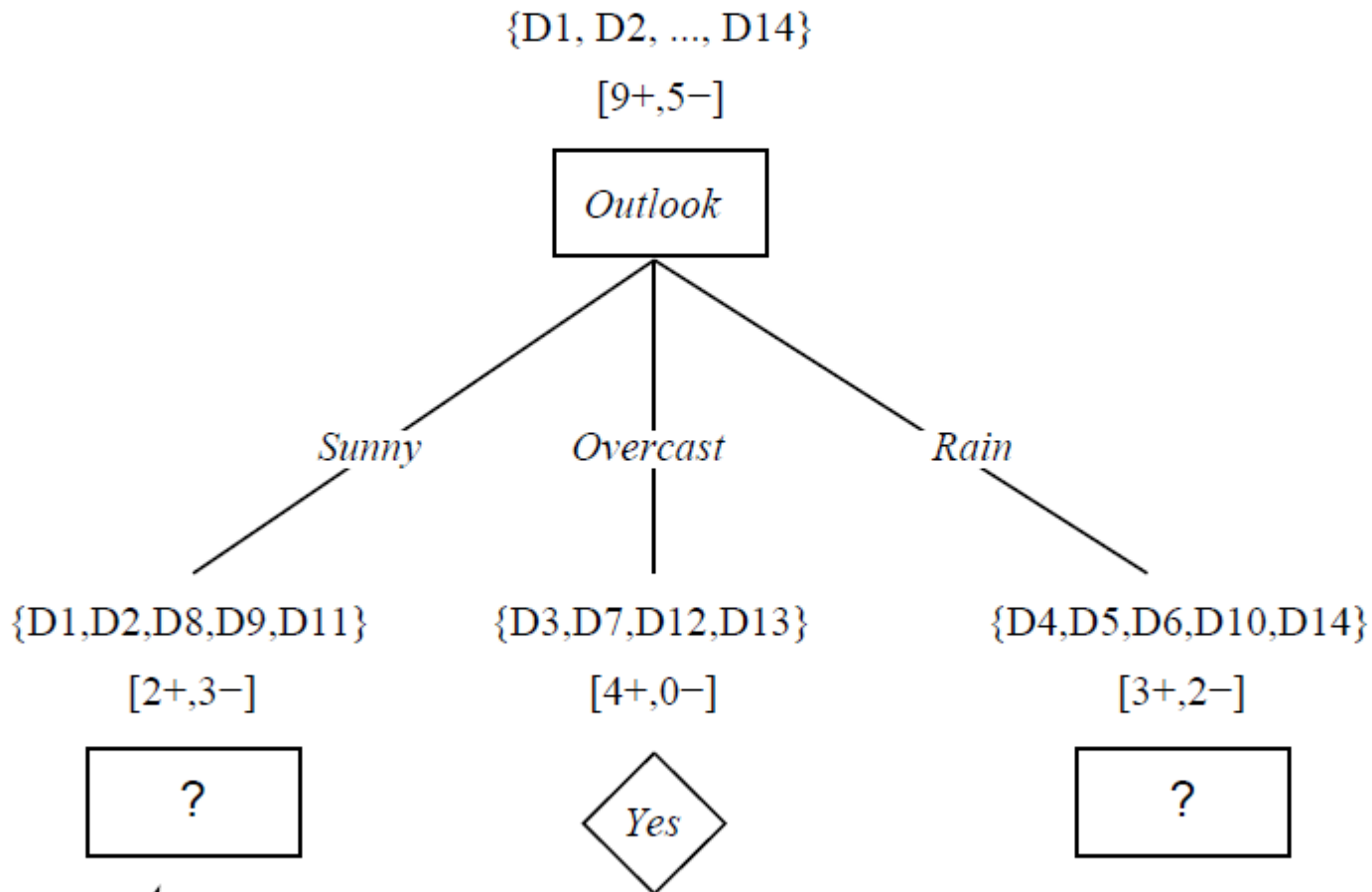
ID3 example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

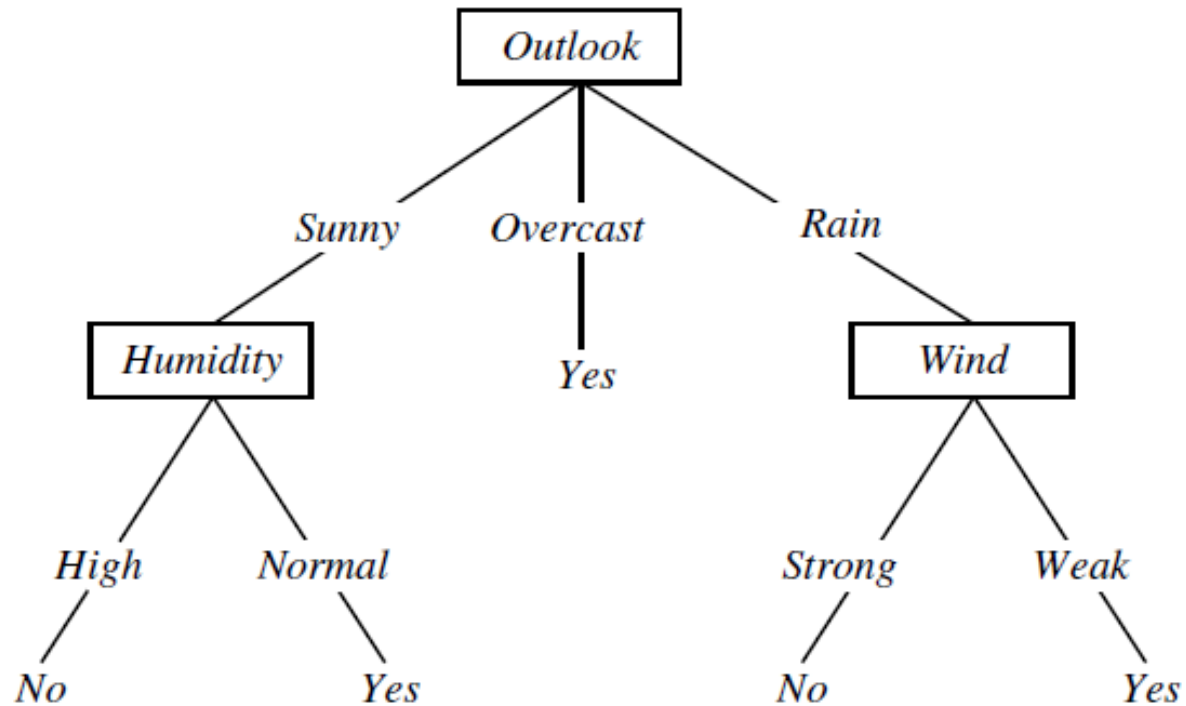
Root level choice

- $\text{Gain}(S, \text{outlook}) = 0.246$
- $\text{Gain}(S, \text{Humidity}) = 0.151$
- $\text{Gain}(S, \text{Wind}) = 0.048$
- $\text{Gain}(S, \text{Temperature}) = 0.029$

Root level tree



Tree



ID3

- Hypothesis space: set of all finite discrete valued functions.
 - The hypothesis space contains all functions.
- ID3 maintains a single hypothesis.
 - Does not enumerate all consistent hypothesis – cannot recommend best training example.
- Utilizes all the data at each stage.
 - Higher computational cost compared to one pass algorithms.
- Performs hill climbing without backtracking
 - Can converge to a local minimum.
- Feature selection criteria robust to errors in data.

Inductive Bias

- Shorter trees are preferred over longer trees.
- Occam's razor: "Simpler" hypothesis are better than more complex hypothesis.
- Shorter trees are "simpler" because there are less number of them.
- Actually: shorter trees with attributes having more information gain are preferred.

Occam's razor

Why prefer short hypotheses?

Argument in favor:

- Fewer short hyps. than long hyps.

→ a short hyp that fits data unlikely to be coincidence

→ a long hyp that fits data might be coincidence

Argument opposed:

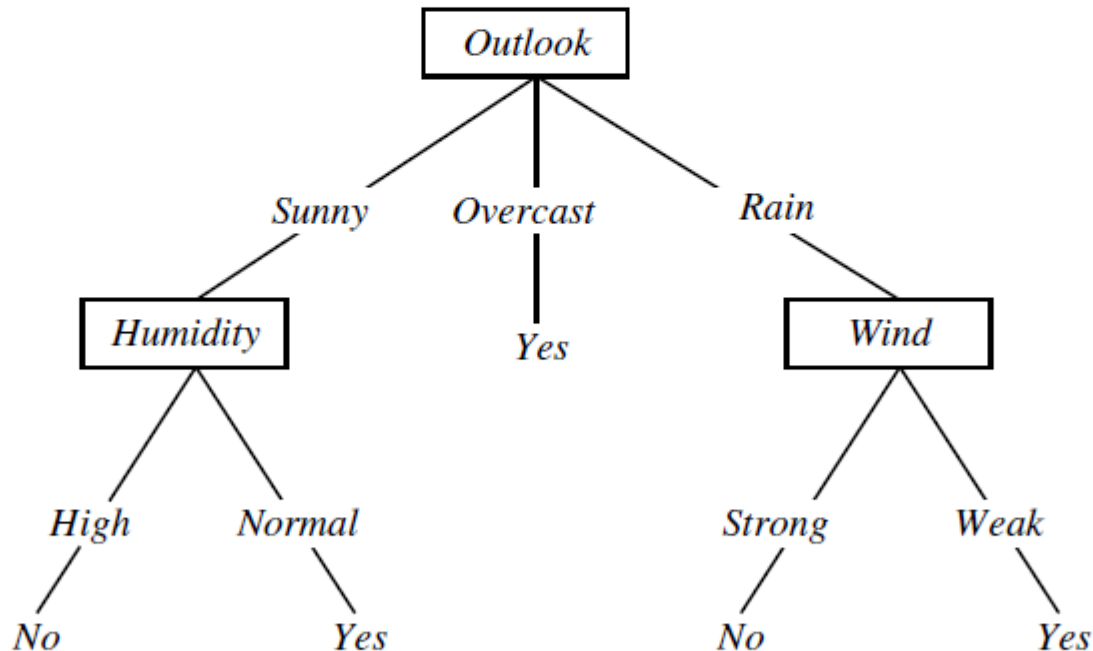
- There are many ways to define small sets of hyps
- e.g., all trees with a prime number of nodes that use attributes beginning with "Z"
- What's so special about small sets based on *size* of hypothesis??

Overfitting

Consider adding noisy training example #15:

Sunny, Hot, Normal, Strong, PlayTennis = No

What effect on earlier tree?



Overfitting

Consider error of hypothesis h over

- training data: $error_{train}(h)$
- entire distribution \mathcal{D} of data: $error_{\mathcal{D}}(h)$

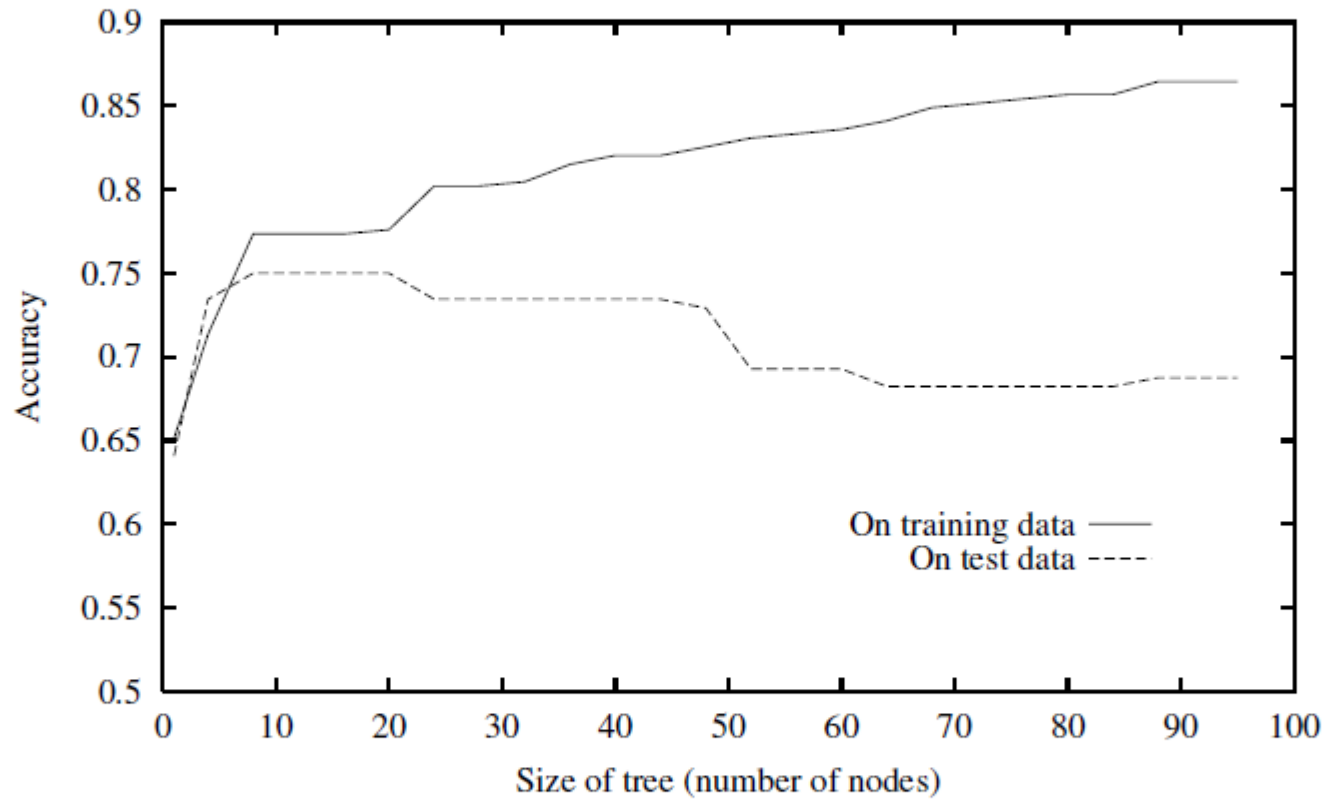
Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Overfitting



Avoiding overfitting

How can we avoid overfitting?

- stop growing when data split not statistically significant
 - grow full tree, then post-prune
-
- Which is computationally better ?

Avoiding overfitting

Which tree is best ?

- Measure the accuracy over a separate validation set.
- Perform statistical tests over training data, e.g. chi square tests.
- Minimize an explicit performance measure which comprises of training set performance and “complexity” of the model, e.g. MDL.

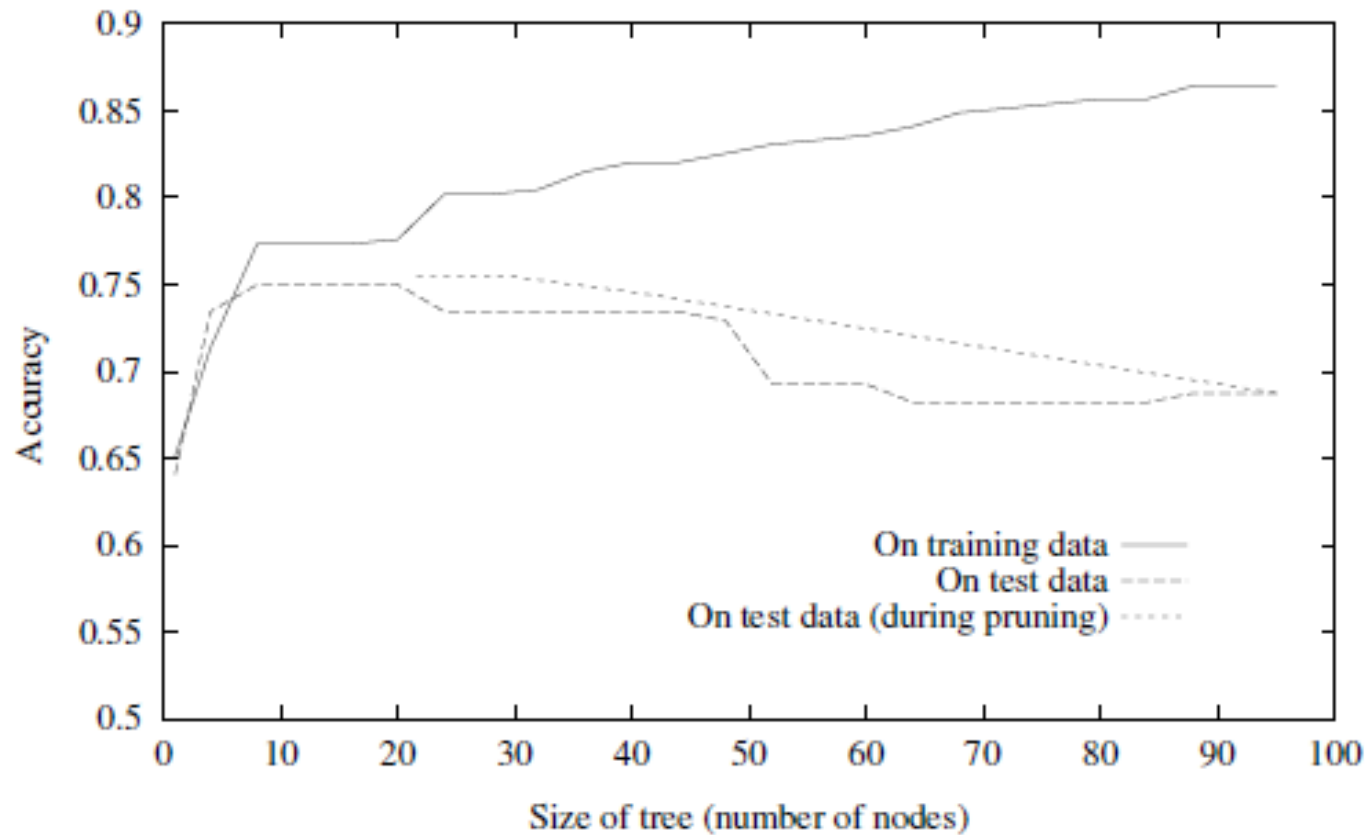
Reduced error pruning

Split data into *training* and *validation* set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves *validation* set accuracy

Reduced error pruning



Rule post pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

Other issues

- Handling continuous valued attribute.
 - Create a split which produces the highest information gain.
- Handling large number of discrete valued attributes.

$$\textit{GainRatio}(S, A) \equiv \frac{\textit{Gain}(S, A)}{\textit{SplitInformation}(S, A)}$$

$$\textit{SplitInformation}(S, A) \equiv - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Other issues

- Handling missing attributes.
- Training set:
 - Assign a distribution based on existing values.
 - Default branch for the set of attributed.
- Test set:
 - Default branch which is populated using missing values.

BAGGING

Ensemble methods

- A single decision tree does not perform well
- But, it is super fast
- What if we learn multiple trees?

We need to make sure they do not all just learn the same

Bagging

If we split the data in random different ways, decision trees give different results, **high variance**.

Bagging: Bootstrap aggregating is a method that result in low variance.

If we had multiple realizations of the data (or multiple samples) we could calculate the predictions multiple times and take the average of the fact that averaging multiple onerous estimations produce less uncertain results

Bagging

Say for each sample b , we calculate $f^b(x)$, then:

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

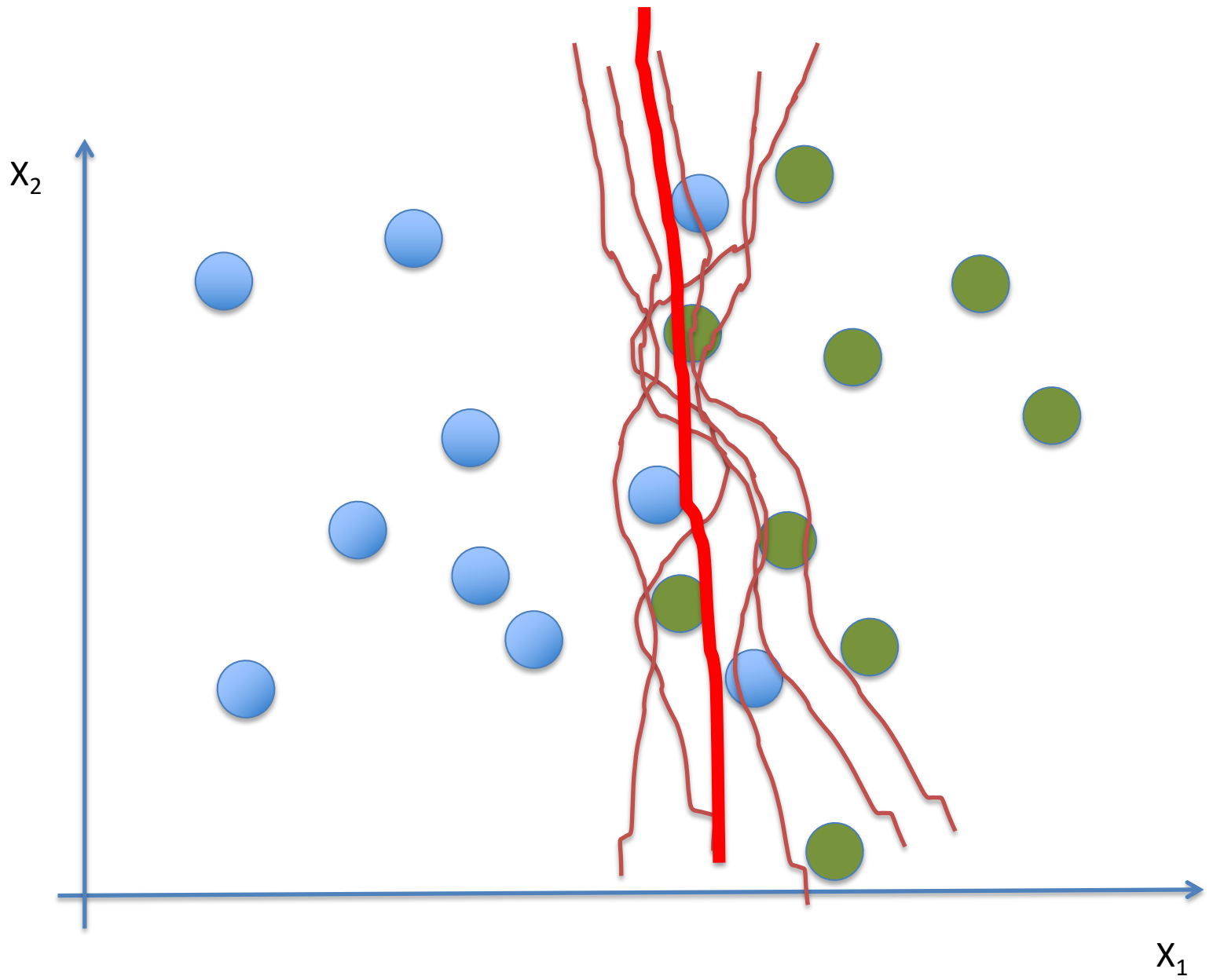
How?

Bootstrap

Construct B (hundreds) of trees (no pruning)

Learn a classifier for each bootstrap sample and average them

Very effective



Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample**. On average $1/3$ of them are not used!
- We call them out-of-bag samples (OOB)
- We can predict the response for the i -th observation using each of the trees in which that observation was OOB and do this for n observations
- Calculate overall OOB MSE or classification error

Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

Bagging - issues

Each tree is identically distributed (i.d.)

→ the expectation of the average of B such trees is the same as the expectation of any one of them

→ the bias of bagged trees is the same as that of the individual trees

i.d. and not i.i.d

Bagging - issues

An average of B i.i.d. random variables, each with variance σ^2 , has variance: σ^2/B

If i.d. (identical but not independent) and pair correlation ρ is present, then the variance is:

$$\rho \sigma^2 + \frac{1 - \rho}{B} \sigma^2$$

As B increases the second term disappears but the first term remains

Why does bagging generate correlated trees?

Bagging - issues

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

RANDOM FORESTS

Random Forests

As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.

Note that if $m = p$, then this is bagging.

Random Forests

Random forests are popular. Leo Breiman's and Adele Cutler maintains a random forest website where the software is freely available, and of course it is included in every ML/STAT package

<http://www.stat.berkeley.edu/~breiman/RandomForests/>

Random Forests Algorithm

For $b = 1$ to B :

(a) Draw a bootstrap sample Z^* of size N from the training data.

(b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

i. Select m variables at random from the p variables.

ii. Pick the best variable/split-point among the m .

iii. Split the node into two daughter nodes.

Output the ensemble of trees.

To make a prediction at a new point x we do:

For regression: average the results

For classification: majority vote

Random Forests Tuning

The inventors make the following recommendations:

- For classification, the default value for m is \sqrt{p} and the minimum node size is one.
- For regression, the default value for m is $p/3$ and the minimum node size is five.

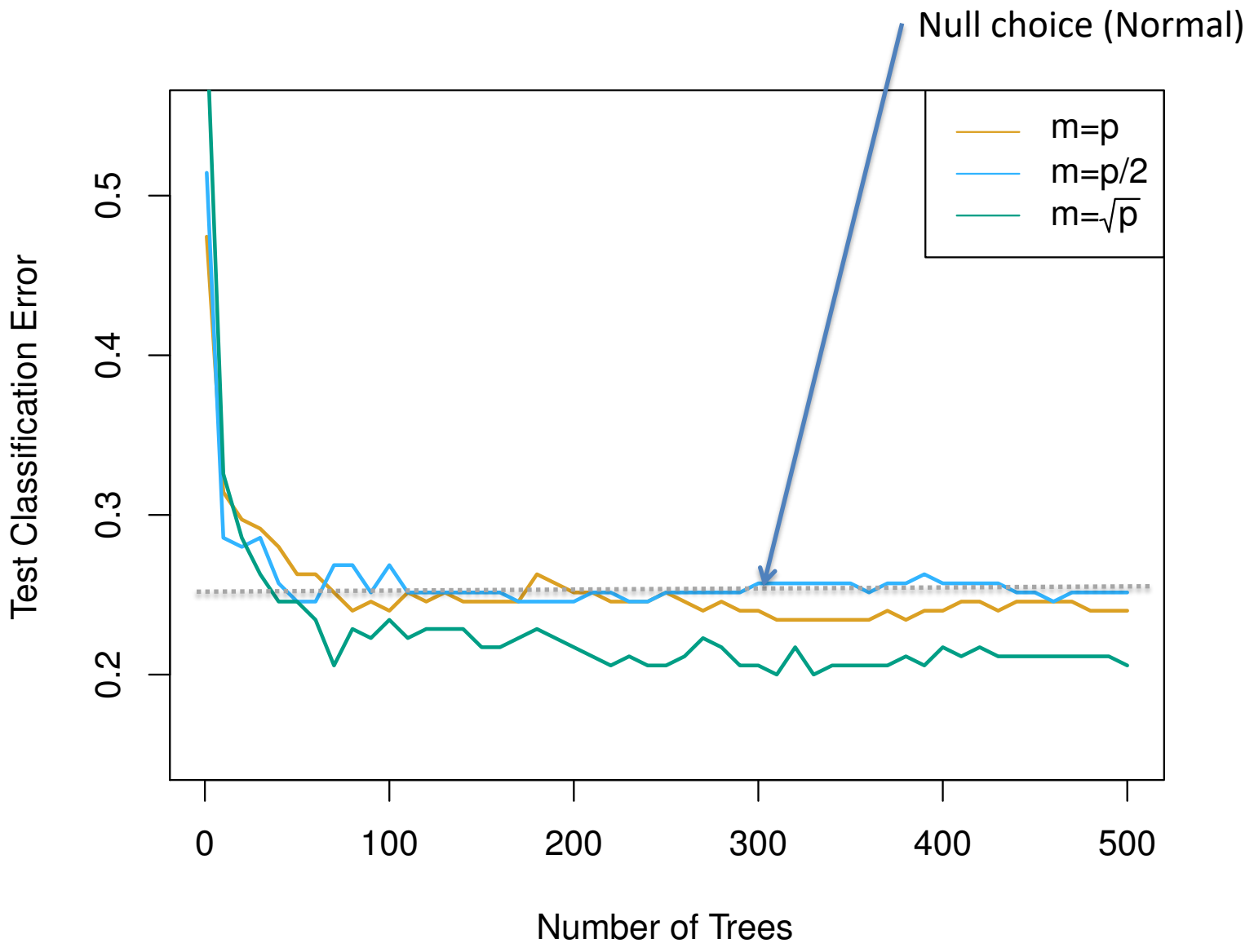
In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Example

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.

Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.



Random Forests Issues

When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when m is small

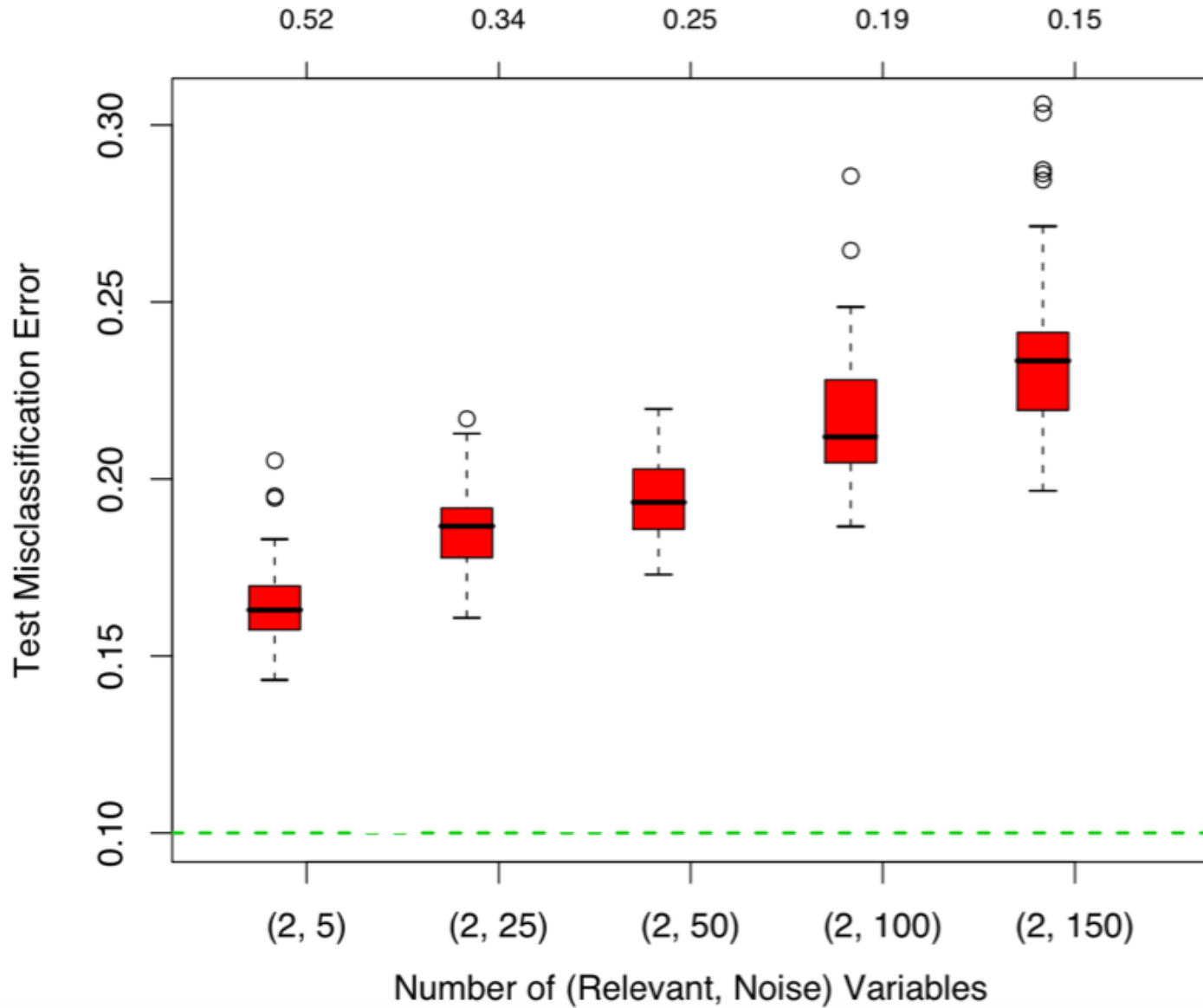
Why?

Because:

At each split the chance can be small that the relevant variables will be selected

For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~ 0.25

Probability of being selected



Can RF overfit?

Random forests “cannot overfit” the data wrt to number of trees.

Why?

The number of trees, B does not mean increase in the flexibility of the model

Constructing Confusion Matrix for multiple classes

For each pair of classes $\langle c_1, c_2 \rangle$ how many documents from c_1 were incorrectly assigned to c_2 ? (when $c_2 \neq c_1$)

Docs in test set	Assigned UK	Assigned poultry	Assigned wheat	Assigned coffee	Assigned interest	Assigned trade
True UK	95	1	13	0	1	0
True poultry	0	1	0	0	0	0
True wheat	10	90	0	1	0	0
True coffee	0	0	0	34	3	7
True interest	-	1	2	13	26	5
True trade	0	0	2	14	5	10

Averaging: Micro vs. Macro

- We now have an evaluation measure (F_1) for **one class**.
- But we also want a single number that measures the **aggregate performance** over all classes in the collection.
- **Macroaveraging**
 - Compute F_1 for each of the C classes
 - Average these C numbers
- **Microaveraging**
 - Compute TP, FP, FN for each of the C classes
 - Sum these C numbers (e.g., all TP to get aggregate TP)
 - Compute F_1 for aggregate TP, FP, FN

Micro- vs. Macro-average: Example

Class 1

	Truth: yes	Truth: no
Classifier: yes	10	10
Classifier: no	10	970

Class 2

	Truth: yes	Truth: no
Classifier: yes	90	10
Classifier: no	10	890

Micro Ave. Table

	Truth: yes	Truth: no
Classifier: yes	100	20
Classifier: no	20	1860

- Macro-averaged precision: $(0.5 + 0.9)/2 = 0.7$
- Micro-averaged precision: $100/120 = 0.83$

Micro-averaged score is dominated by score on common classes