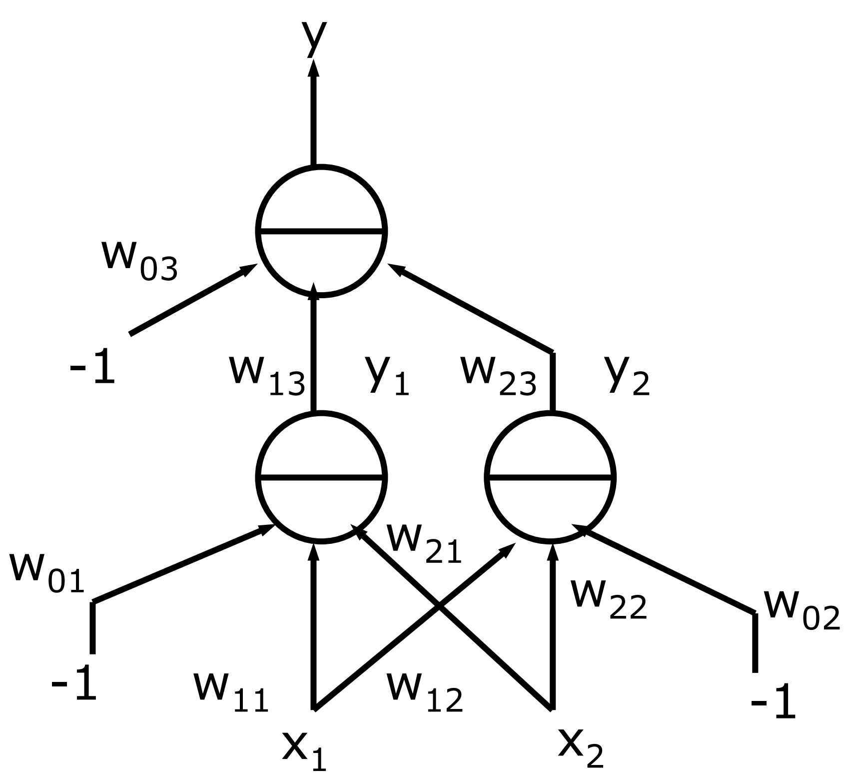
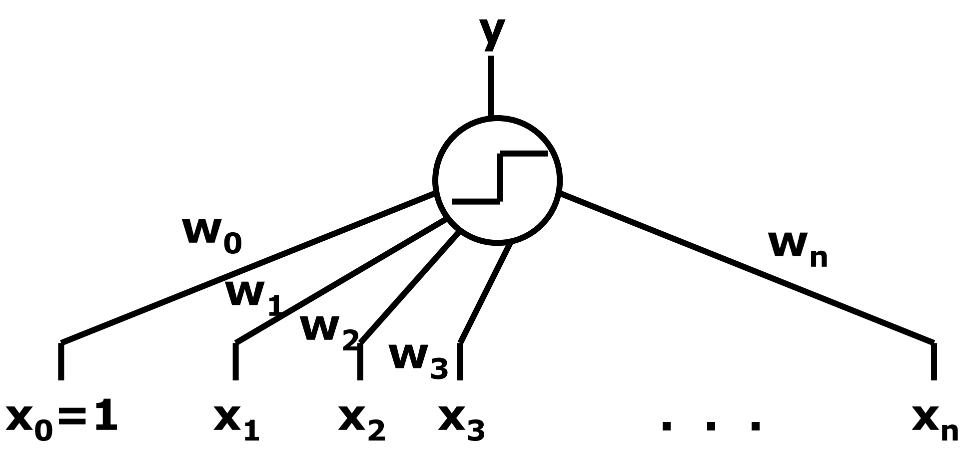
Artificial Neural Networks
(Feedforward Nets)



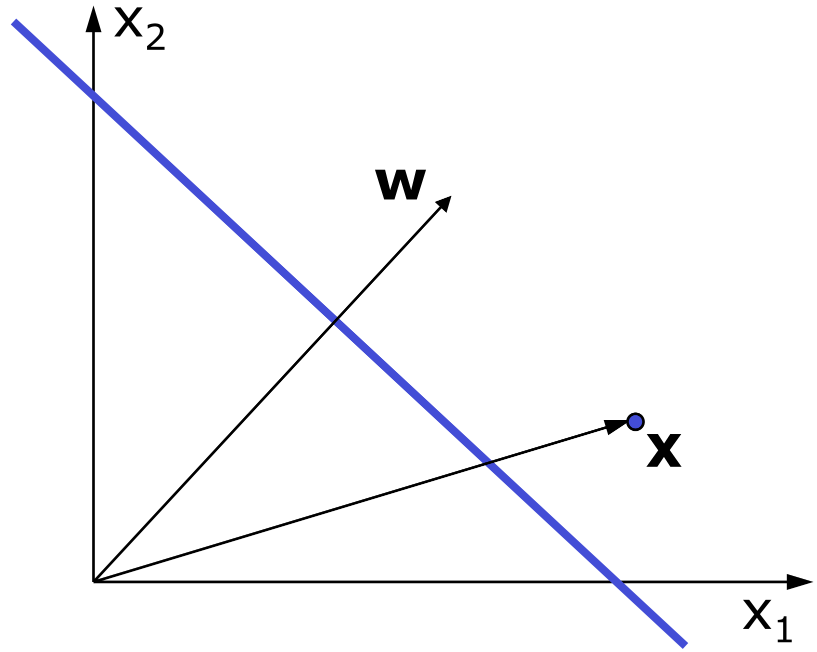
Single Perceptron Unit

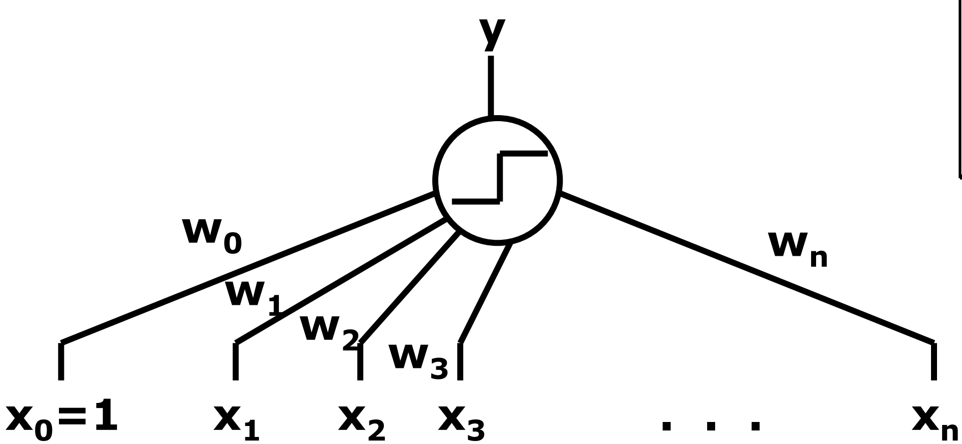


Linear Classifier
Single Perceptron Unit

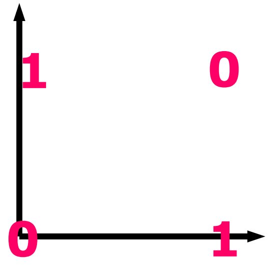
$$h(\mathbf{x}) = \theta(\mathbf{w} \cdot \mathbf{x} + b) \equiv \theta(\overline{\mathbf{w}} \cdot \overline{\mathbf{x}})$$

$$\theta(z) = \begin{cases} 1 & z \geq 0 \\ 0 & else \end{cases}$$



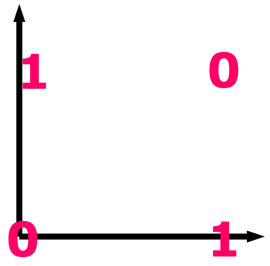


Beyond Linear Separability

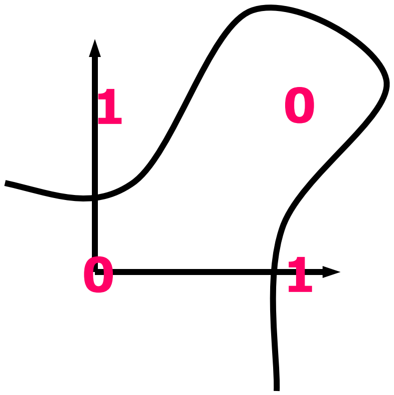


Not linearly separable

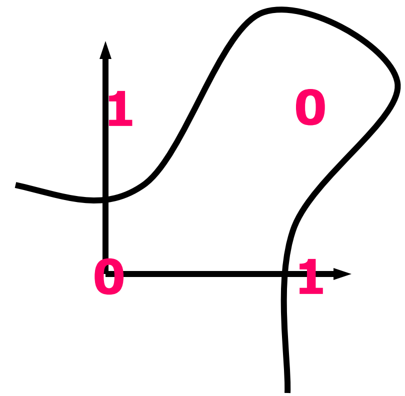
Beyond Linear Separability

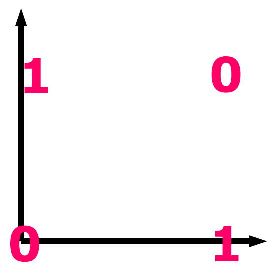


Not linearly
separable

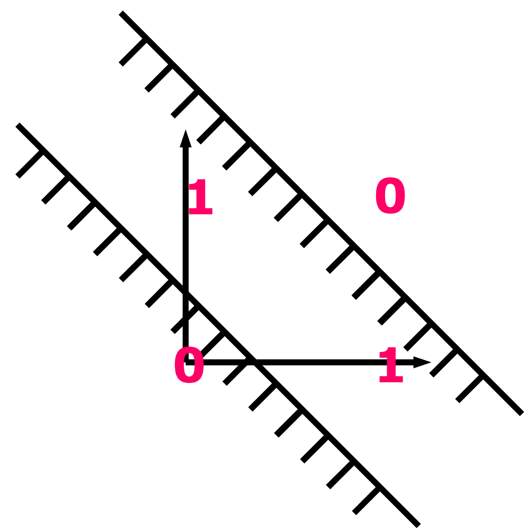


Beyond Linear Separability

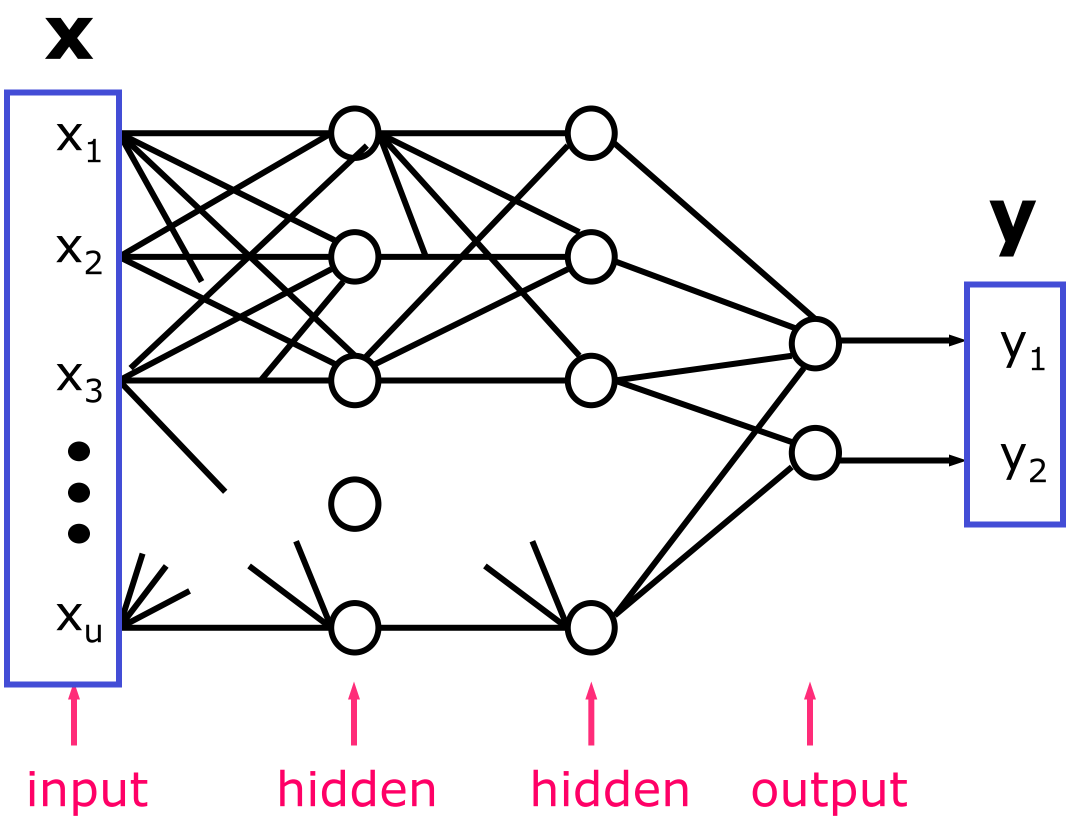




Not linearly
separable

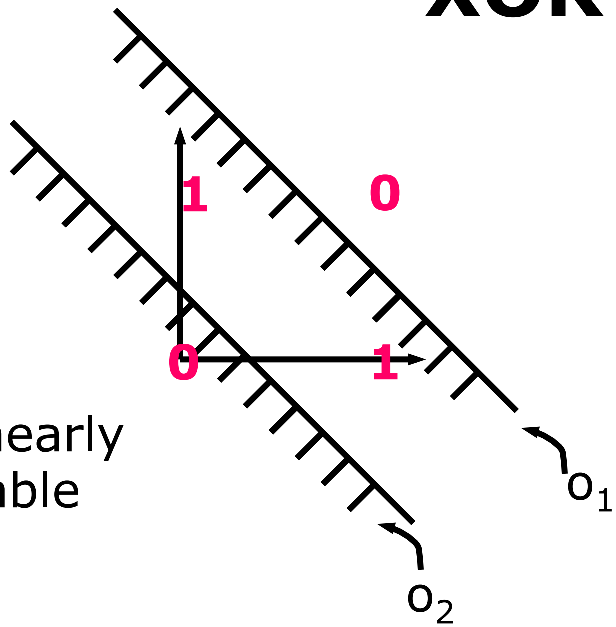


Multi-Layer Perceptron

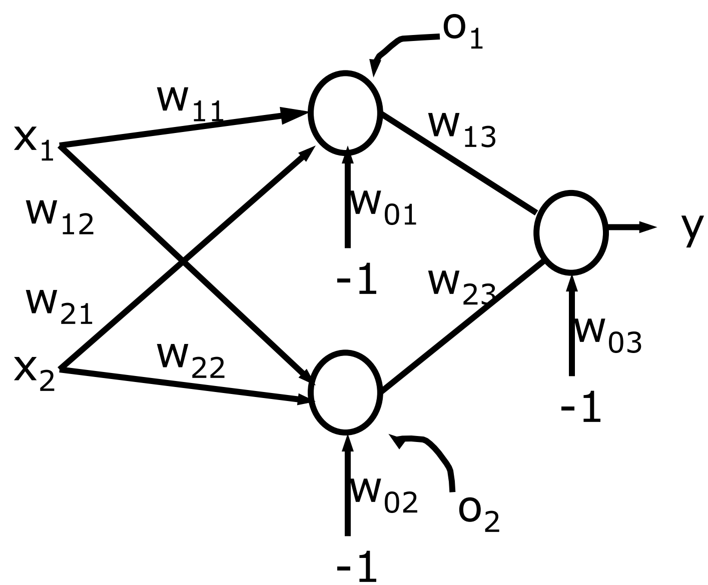


- More powerful than single layer.
- Lower layers transform the input problem into more tractable (linearly separable) problems for subsequent layers.

XOR Problem



Not linearly separable

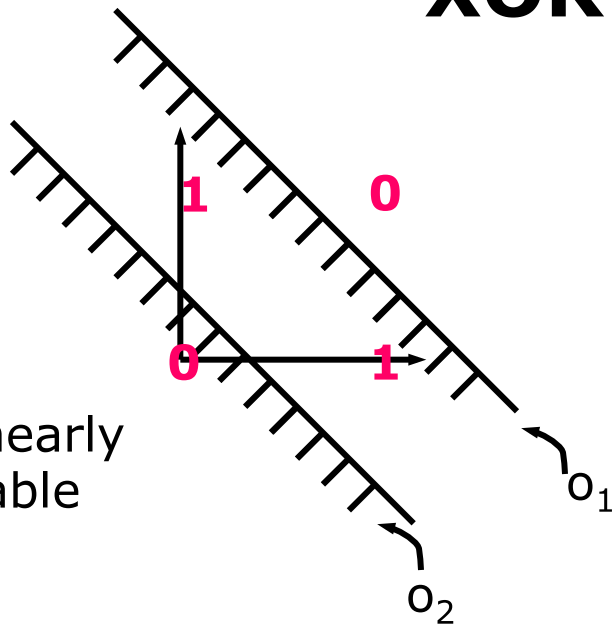


$w_{01} = 3/2 \quad w_{11} = w_{12} = 1$

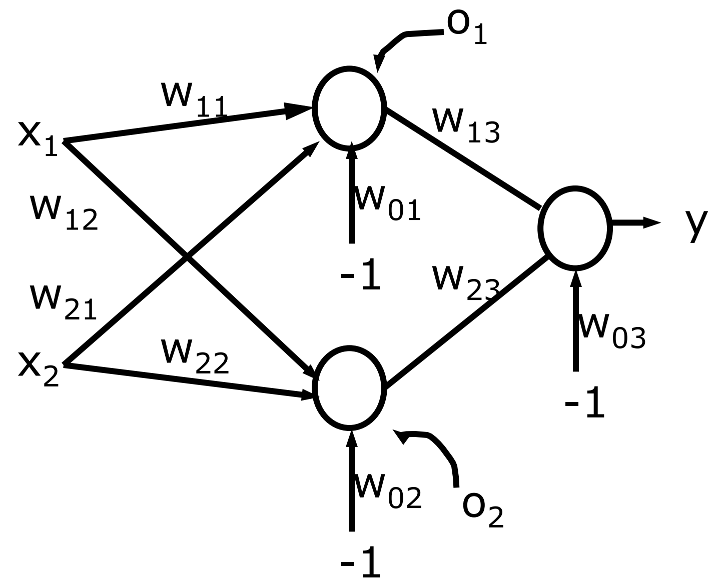
$w_{02} = 1/2 \quad w_{21} = w_{22} = 1$

x_1	x_2	o_1
0	0	0
0	1	0
1	0	0
1	1	1

XOR Problem



Not linearly separable

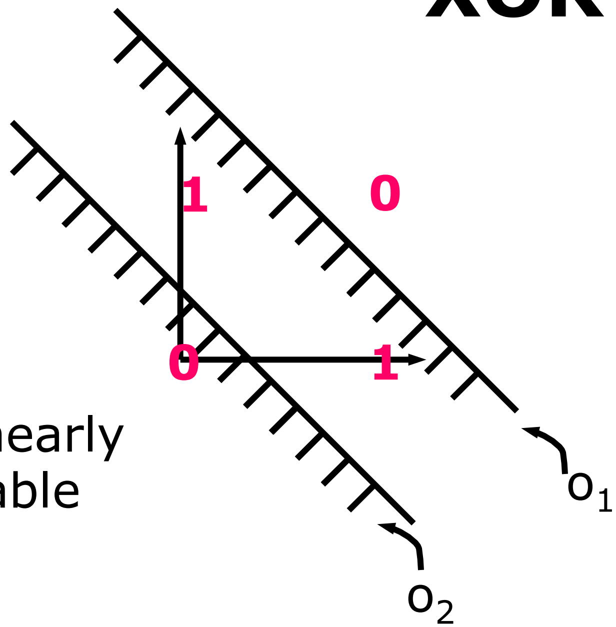


$w_{01} = 3/2 \quad w_{11} = w_{12} = 1$

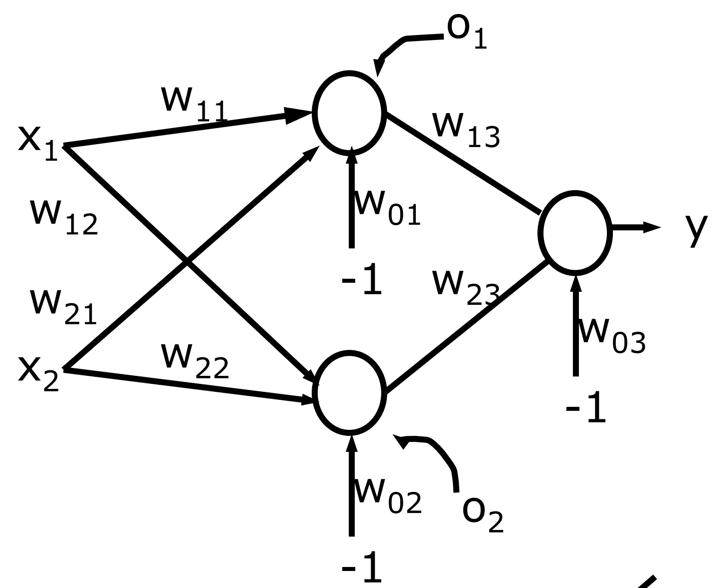
$w_{02} = 1/2 \quad w_{21} = w_{22} = 1$

x_1	x_2	o_1	o_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

XOR Problem

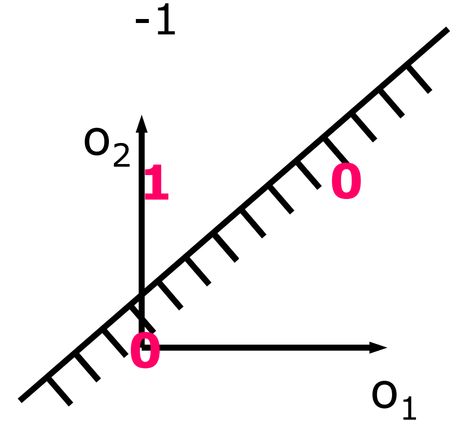


Not linearly separable



$w_{01} = 3/2$	$w_{11} = w_{12} = 1$
$w_{02} = 1/2$	$w_{21} = w_{22} = 1$
$w_{03} = 1/2$	$w_{31} = -1,\ w_{32} = 1$

x_1	x_2	o_1	o_2	y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



Linearly separable

Multi-Layer Perceptron Learning

- Any set of training points can be separated by a three-layer perceptron network.
- "Almost any" set of points separable by two-layer perceptron network.
- But, no efficient learning rule is known.

Multi-Layer Perceptron Learning

- Any set of training points can be separated by a three-layer perceptron network.
- "Almost any" set of points separable by two-layer perceptron network.
- But no efficient learning rule is known.

Two "hidden" layers and one output layer

May need an exponential number of units.

One "hidden" layer and one output layer

Multi-Layer Perceptron Learning

- Any set of training points can be separated by a three-layer perceptron network.
- “Almost any” set of points separable by two-layer perceptron network.
- But, no efficient learning rule is known.

- Could we use gradient ascent/descent?
- We would need smoothness: small change in weights produces small change in output.
- Threshold function is not smooth.

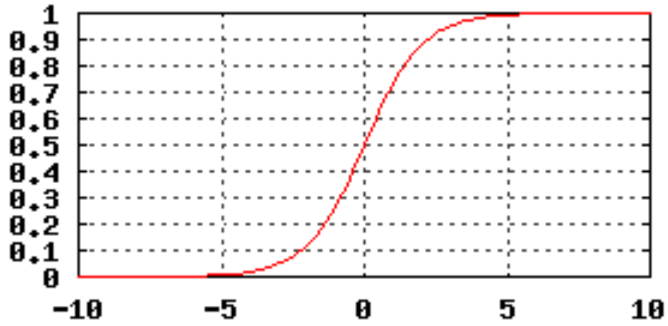
Multi-Layer Perceptron Learning

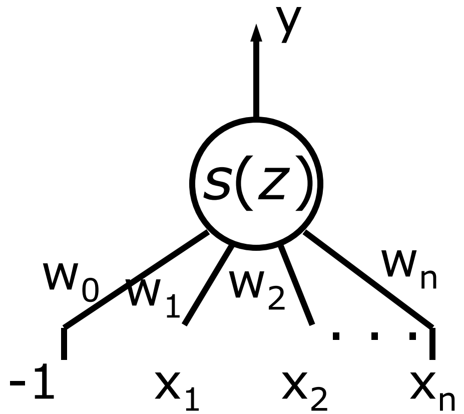
- Any set of training points can be separated by a three-layer perceptron network.
- “Almost any” set of points separable by two-layer perceptron network.
- But, no efficient learning rule is known.

- Could we use gradient ascent/descent?
- We would need smoothness: small change in weights produces small change in output.
- Threshold function is not smooth.

- Use a smooth threshold function!

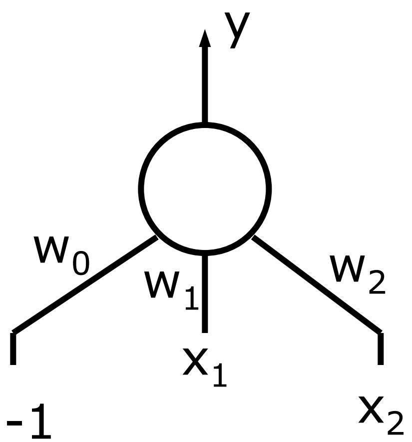
Sigmoid Unit

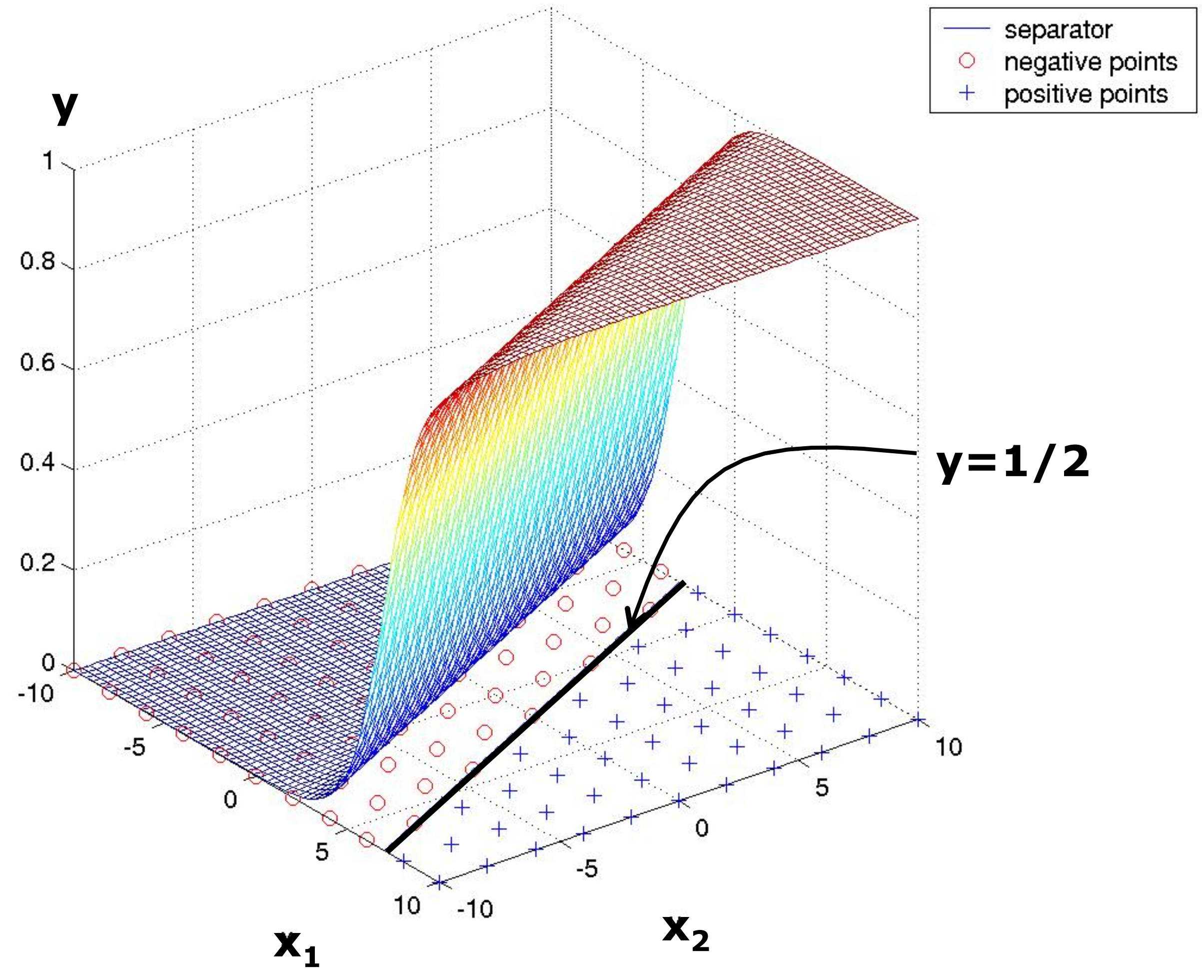




$$z = \sum_{i}^{n} w_i x_i \quad s(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid Unit



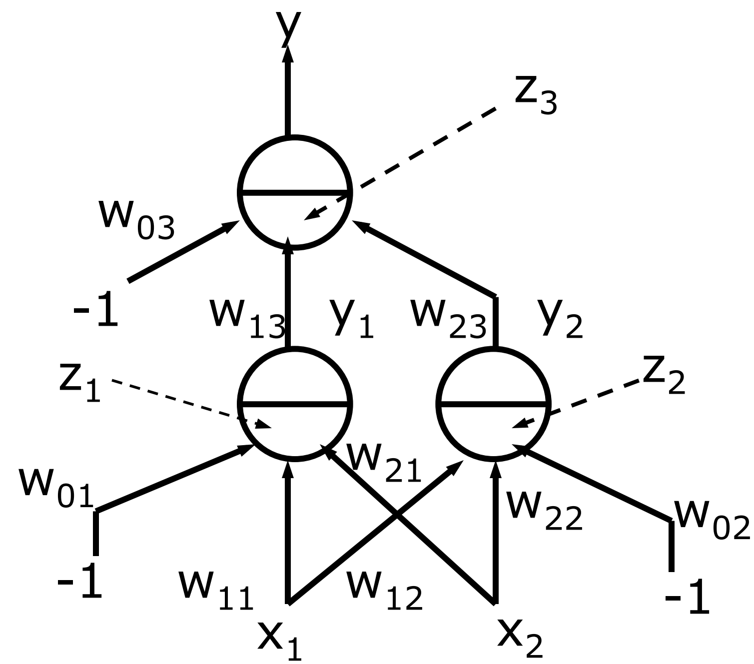


Training

$$y(\mathbf{x}, \mathbf{w})$$

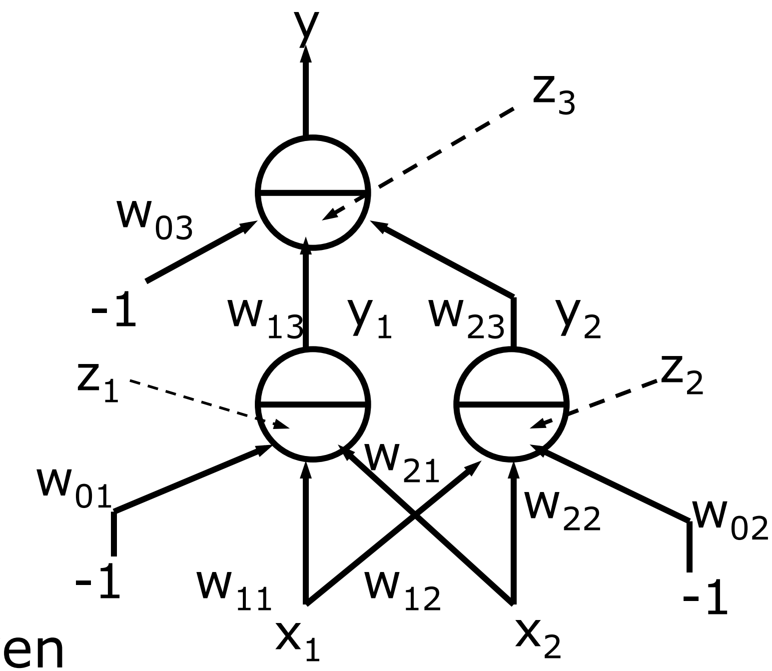
w is a vector of weights

x is a vector of inputs



$$y = \underbrace{s\left(w_{13} s\left(\overbrace{w_{11}x_1 + w_{21}x_2 - w_{01}}^{z_1}\right) + w_{23} s\left(\overbrace{w_{12}x_1 + w_{22}x_2 - w_{02}}^{z_2}\right)\right.}_{z_3} \left. - w_{03}\right)$$

Training



$$y(\mathbf{x}, \mathbf{w})$$

w is a vector of weights

x is a vector of inputs

y^i is desired output:

Error over the training set for a given weight vector:

$$E = \frac{1}{2}\sum_i (y(\mathbf{x}^i, \mathbf{w}) - y^i)^2$$

Our goal is to find weight vector that minimizes error

$$y = s(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 - w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 - w_{02}}^{z_2})}_{z_3} - w_{03})$$

Gradient Descent

$$E = \frac{1}{2}\sum_i (y(\mathbf{x}^i, \mathbf{w}) - y^i)^2$$

Error on training set

$$\nabla_{\mathbf{w}} E = \sum_i (y(\mathbf{x}^i, \mathbf{w}) - y^i)\nabla_{\mathbf{w}} y(\mathbf{x}^i, \mathbf{w})$$

Gradient of Error

$$\nabla_{\mathbf{w}} y = \left[\frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n}\right]$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E$$

Gradient Descent

Training Neural Nets

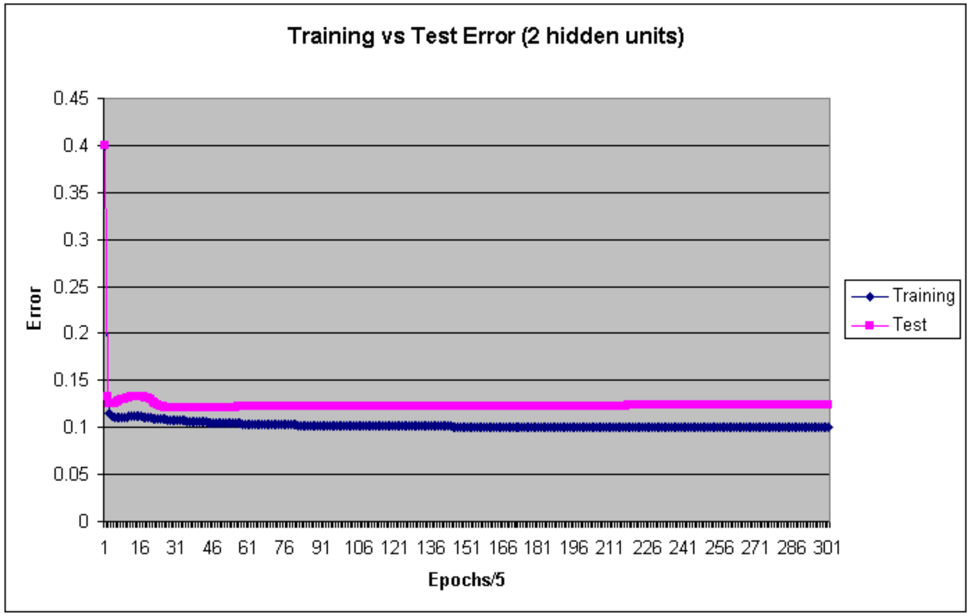
without overfitting, hopefully...

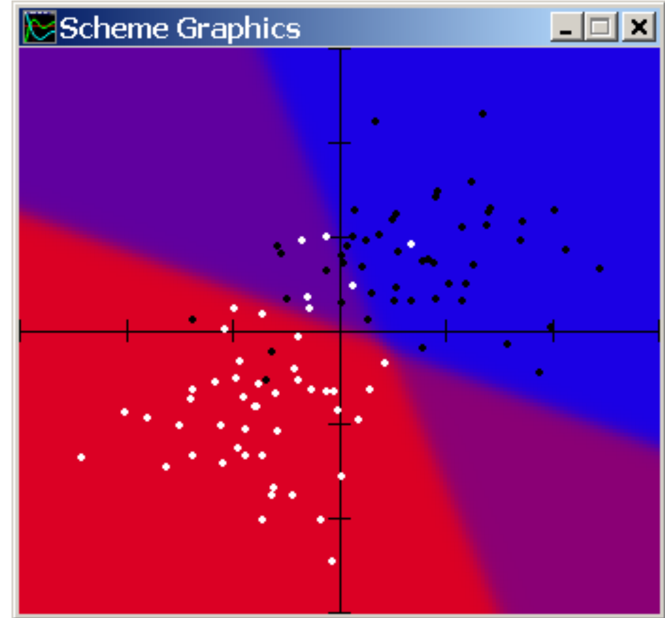
Given: Data set, desired outputs and a neural net with m weights. Find a setting for the weights that will give good predictive performance on new data. Estimate expected performance on new data.

1. Split data set (randomly) into three subsets:
   - Training set – used for picking weights
   - Validation set – used to stop training
   - Test set – used to evaluate performance
2. Pick random, small weights as initial values
3. Perform iterative minimization of error over training set.
4. Stop when error on validation set reaches a minimum (to avoid overfitting).
5. Repeat training (from step 2) several times (avoid local minima)
6. Use best weights to compute error on test set, which is estimate of performance on new data. Do not repeat training to improve this.

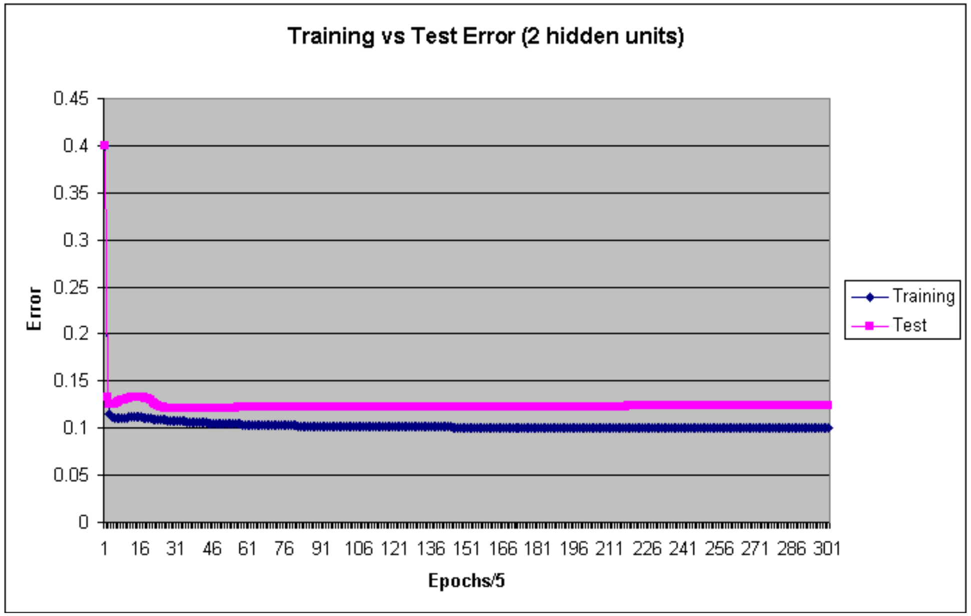
Can use cross-validation if data set is too small to divide into three subsets.

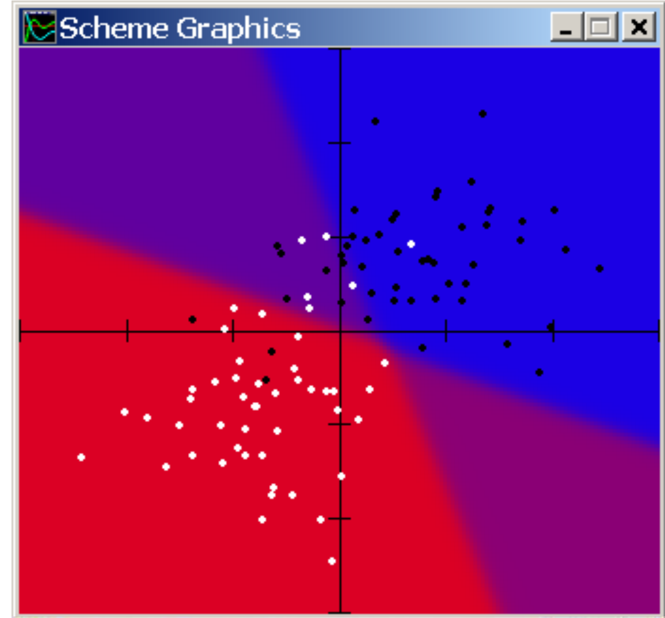
Training vs. Test Error

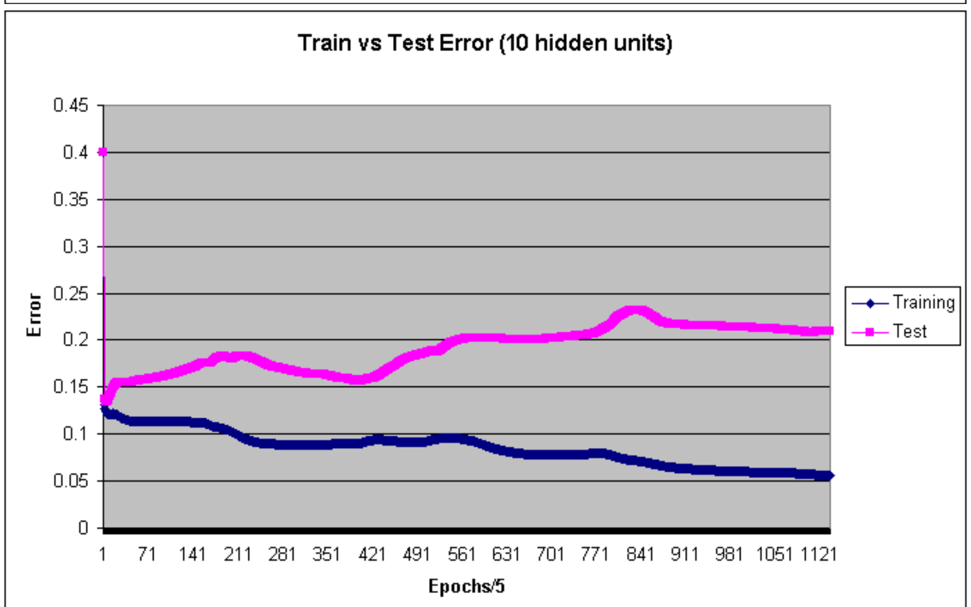


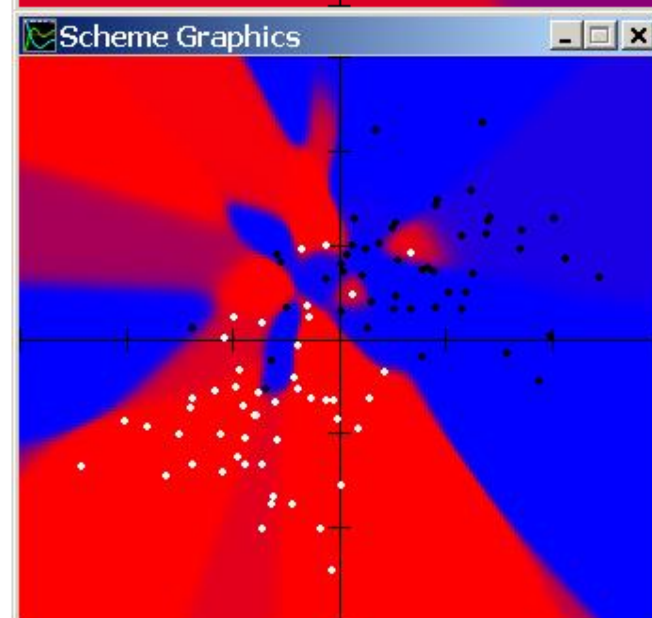


Training vs. Test Error

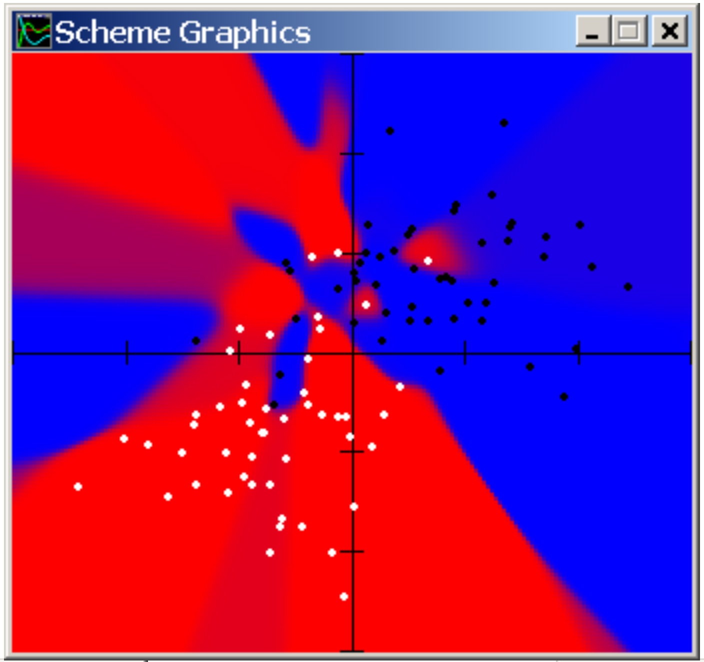




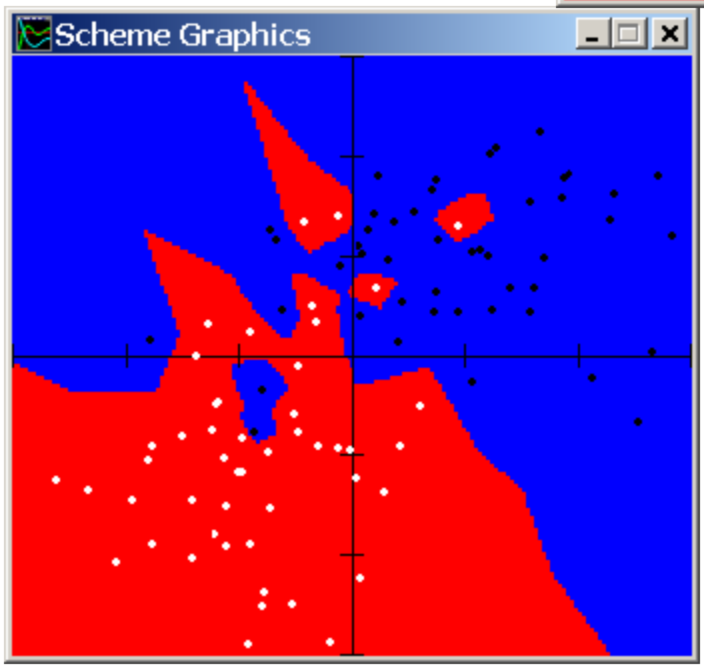




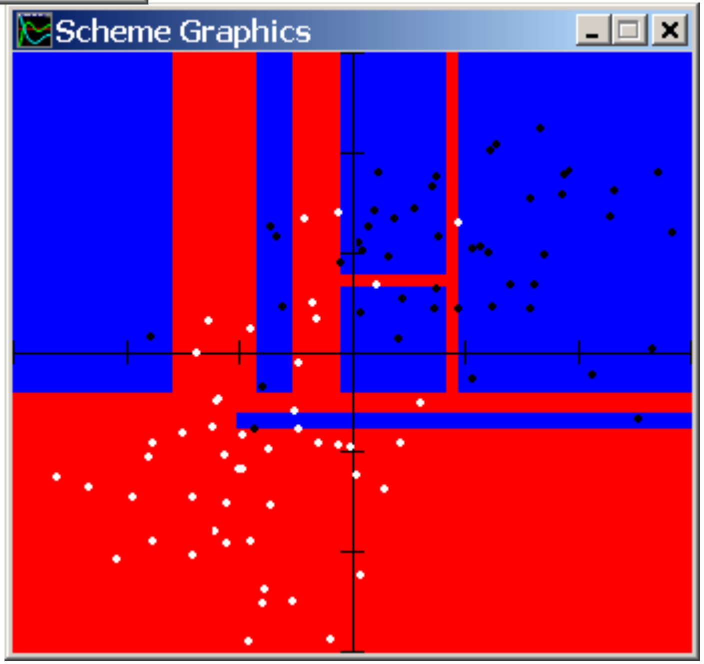
Overfitting is not unique to neural nets...



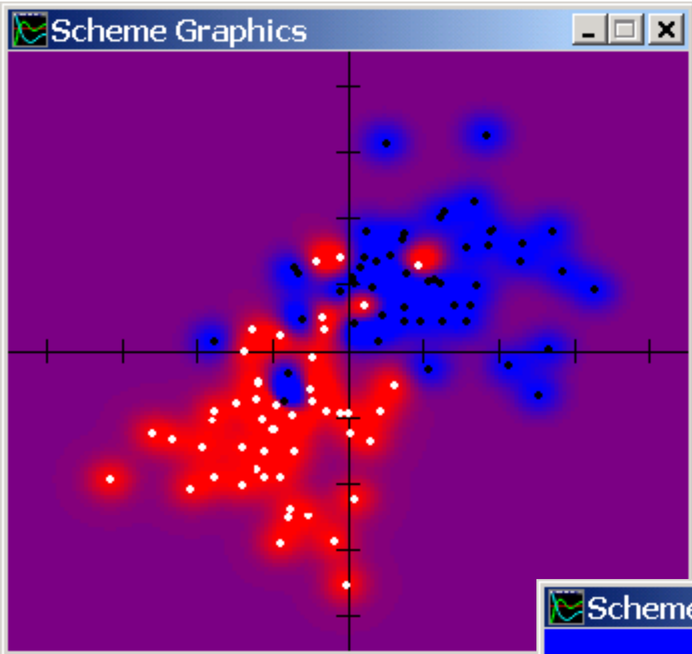
1-Nearest Neighbors



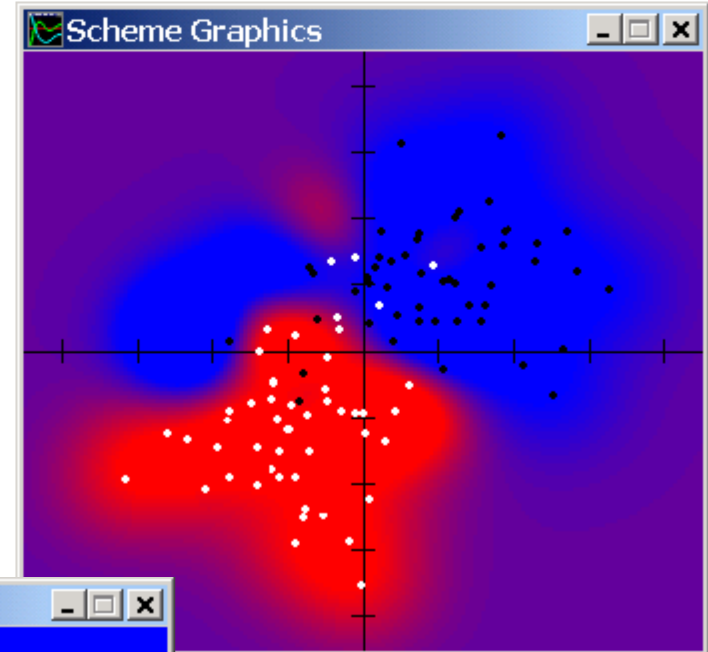
Decision Trees



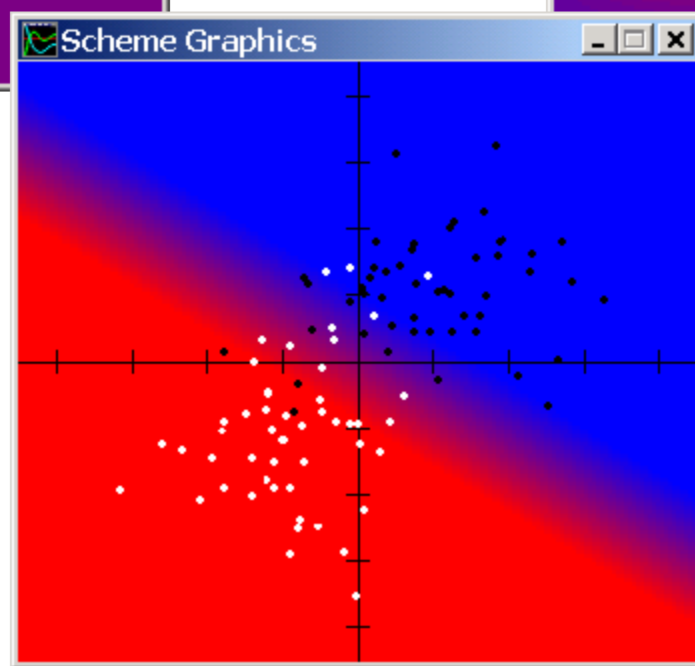
Overfitting in SVM



Radial Kernel $\sigma=0.1$



Radial Kernel $\sigma=1$



On-line vs off-line

There are two approaches to performing the error minimization:

- On-line training – present $\mathbf{x}^i$ and y^{i*} (chosen randomly from the training set). Change the weights to reduce the error on this instance. Repeat.
- Off-line training – change weights to reduce the total error on training set (sum over all instances).

On-line training is an approximation to gradient descent since the gradient based on one instance is “noisy” relative to the full gradient (based on all instances). This can be beneficial in pushing the system out of shallow local minima.

Feature Selection

- In many machine learning applications, there are huge numbers of features
  - text classification (# words)
  - gene arrays (5,000 – 50,000)
  - images (512 x 512 pixels)

Feature Selection

- In many machine learning applications, there are huge numbers of features
  - text classification (# words)
  - gene arrays (5,000 – 50,000)
  - images (512 x 512 pixels)
- Too many features
  - make algorithms run slowly
  - risk overfitting

Feature Selection

- In many machine learning applications, there are huge numbers of features
  - text classification (# words)
  - gene arrays (5,000 – 50,000)
  - images (512 x 512 pixels)
- Too many features
  - make algorithms run slowly
  - risk overfitting
- Find a smaller feature space
  - subset of existing features
  - new features constructed from old ones

Feature Ranking

- For each feature, compute a measure of its relevance to the output
- Choose the k features with the highest rankings
- Correlation between feature j and output

$$R(j) = \frac{\sum_i (x_j^i - \bar{x}_j)(y^i - \bar{y})}{\sqrt{\sum_i (x_j^i - \bar{x}_j)^2 \sum_i (y^i - \bar{y})^2}}$$

$$\bar{x}_j = \frac{1}{n}\sum_i x_j^i \qquad \bar{y} = \frac{1}{n}\sum_i y^i$$

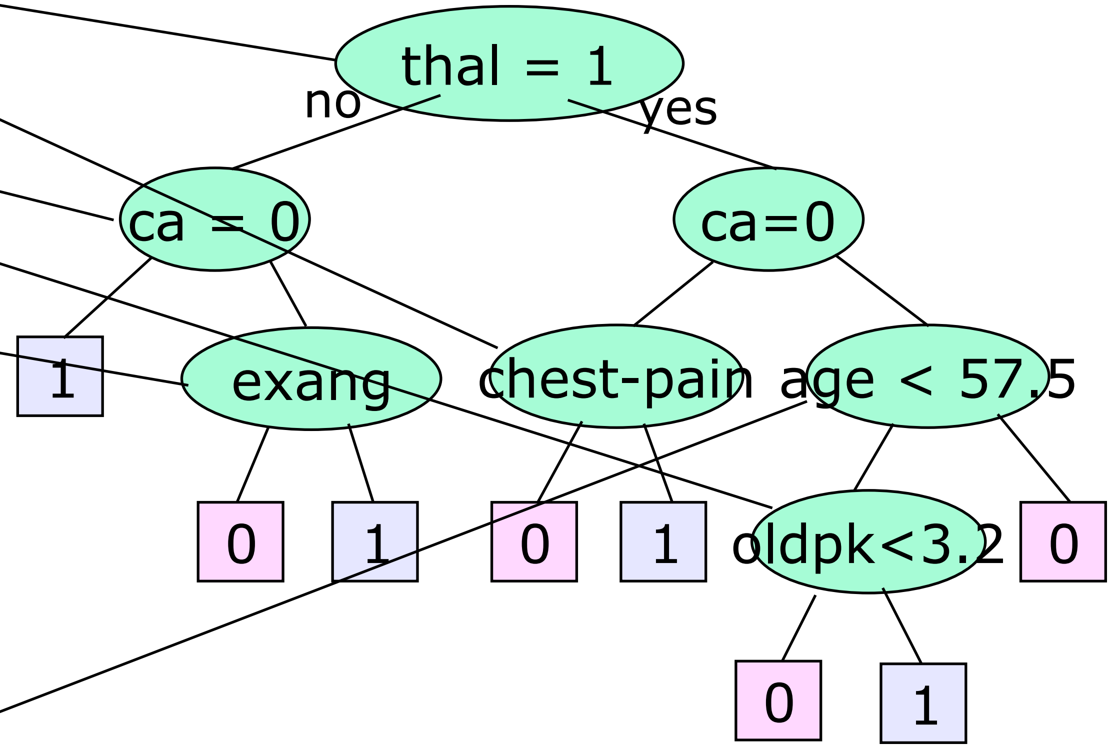
- Correlation measures how much x tends to deviate from its mean on the same examples on which y deviates from its mean

Correlations in Heart Data

Top 15 of 25

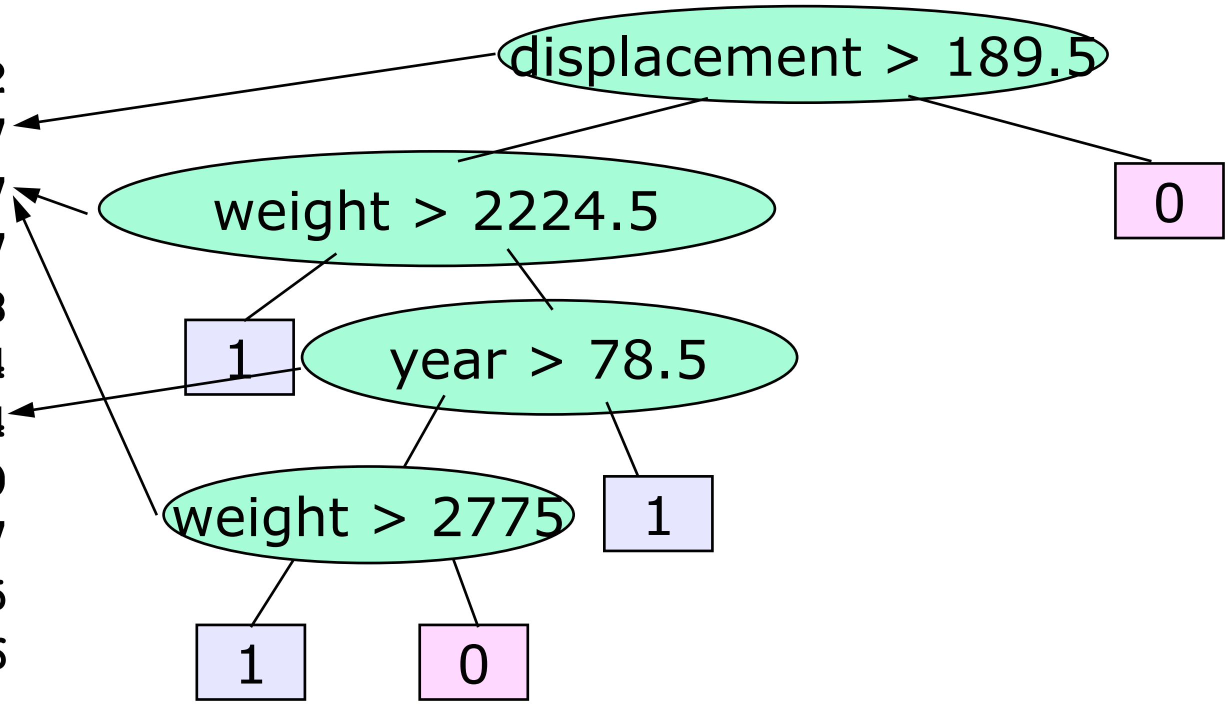
thal=1	-0.52
cp=4	0.51
thal=3	0.48
ca=0	-0.48
oldpeak	0.42
thalach	-0.42
exang	0.42
slope=1	-0.38
slope=2	0.35
cp=3	-0.31
sex	0.28
ca=2	0.27
cp=2	-0.25
ca=1	0.23
age	0.23

…



Correlations in MPG > 22 data

cyl=4	0.82
displacement	-0.77
weight	-0.77
horsepower	-0.67
cyl=8	-0.58
origin=1	-0.54
model-year	0.44
origin=3	0.40
cyl=6	-0.37
acceleration	0.35
origin=2	0.26



XOR Bites Back

- As usual, functions with XOR in them will cause us trouble
  - Each feature will, individually, have a correlation of 0 (it occurs positively as much as negatively for positive outputs)
  - To solve XOR, we need to look at groups of features together

Subset Selection

- Consider subsets of variables
  - too hard to consider all possible subsets
  - wrapper methods: use training set or cross-validation error to measure the goodness of using different feature subsets with your classifier
  - greedily construct a good subset by adding or subtracting features one by one

Forward Selection

Given a particular classifier you want to use

```
F = {}
   For each f_j
      Train classifier with inputs F + {f_j}
   Add f_j that results in lowest-error classifier
     to F
Continue until F is the right size, or error has
  quit decreasing
```

Forward Selection

Given a particular classifier you want to use

```
F = {}
    For each f_j
        Train classifier with inputs F + {f_j}
    Add f_j that results in lowest-error classifier
      to F
Continue until F is the right size, or error has
  quit decreasing
```

- Decision trees, by themselves, do something similar to this

Forward Selection

Given a particular classifier you want to use

```
F = {}
    For each f_j
        Train classifier with inputs F + {f_j}
    Add f_j that results in lowest-error classifier
      to F
Continue until F is the right size, or error has
  quit decreasing
```

- Decision trees, by themselves, do something similar to this
- Trouble with XOR

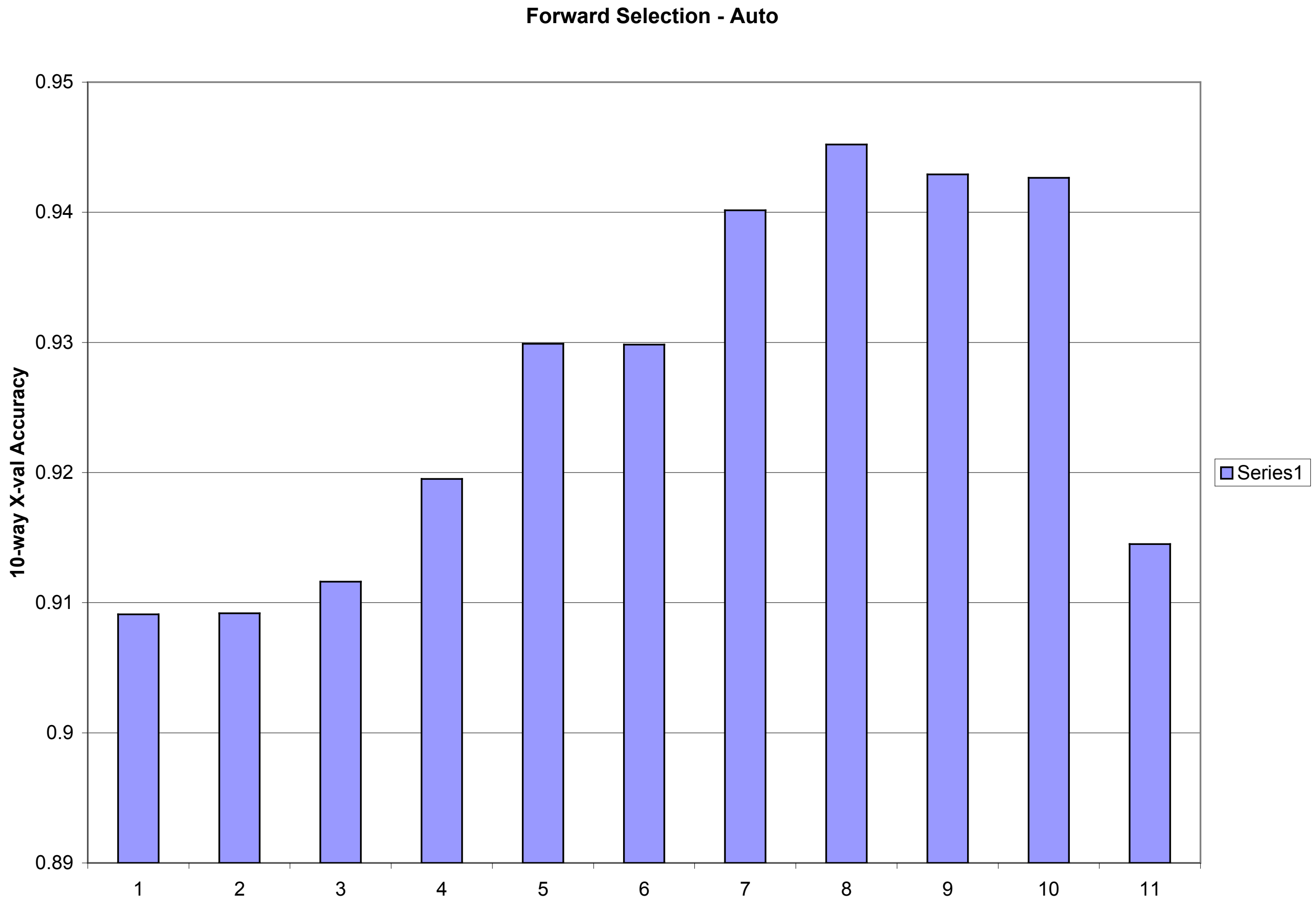
Backward Elimination

Given a particular classifier you want to use

```
F = all features
    For each f_j
        Train classifier with inputs F - {f_j}
    Remove f_j that results in lowest-error
      classifier from F
Continue until F is the right size, or error
  increases too much
```

Forward Selection on Auto Data

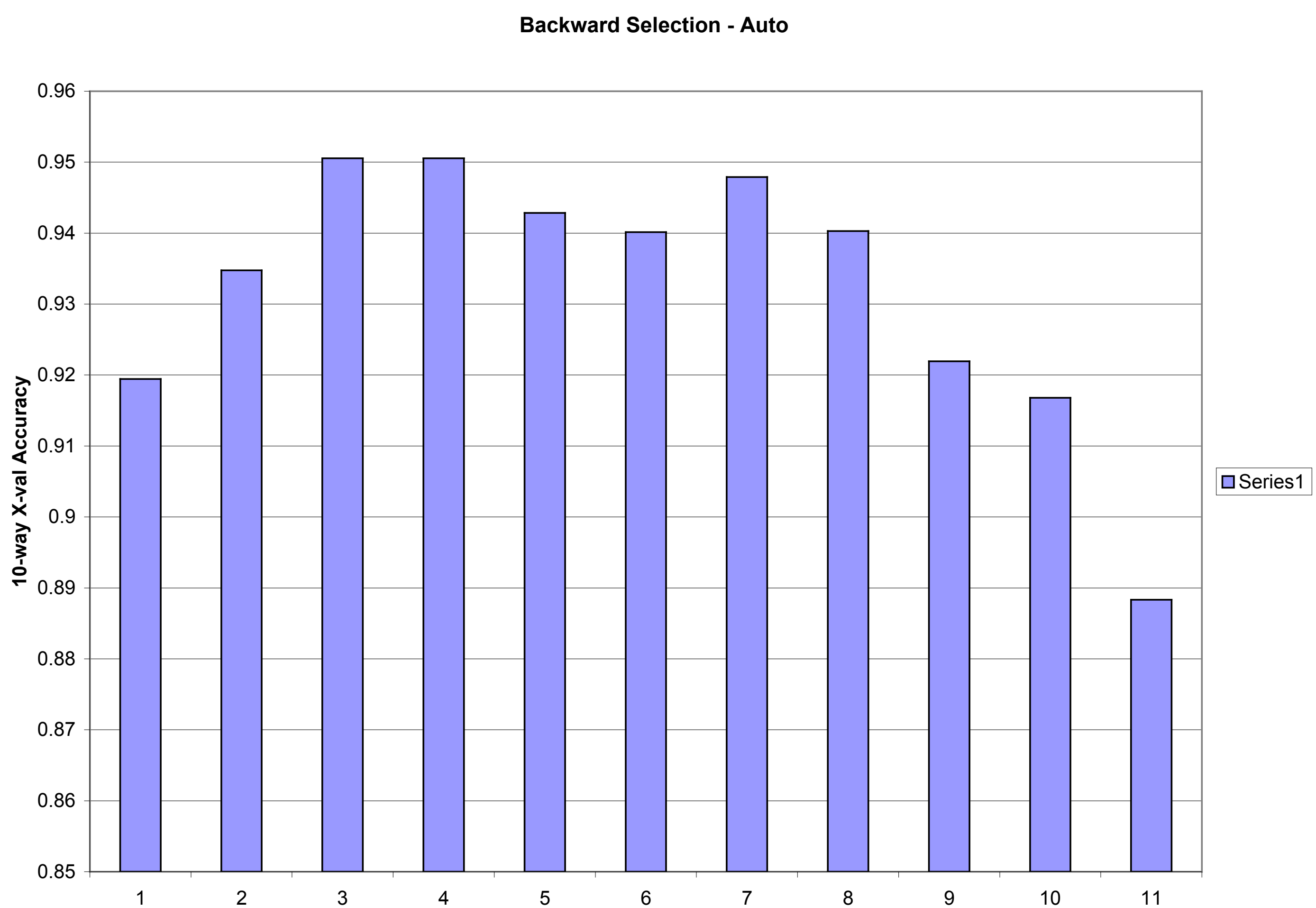
cross-validation accuracy



number of features added

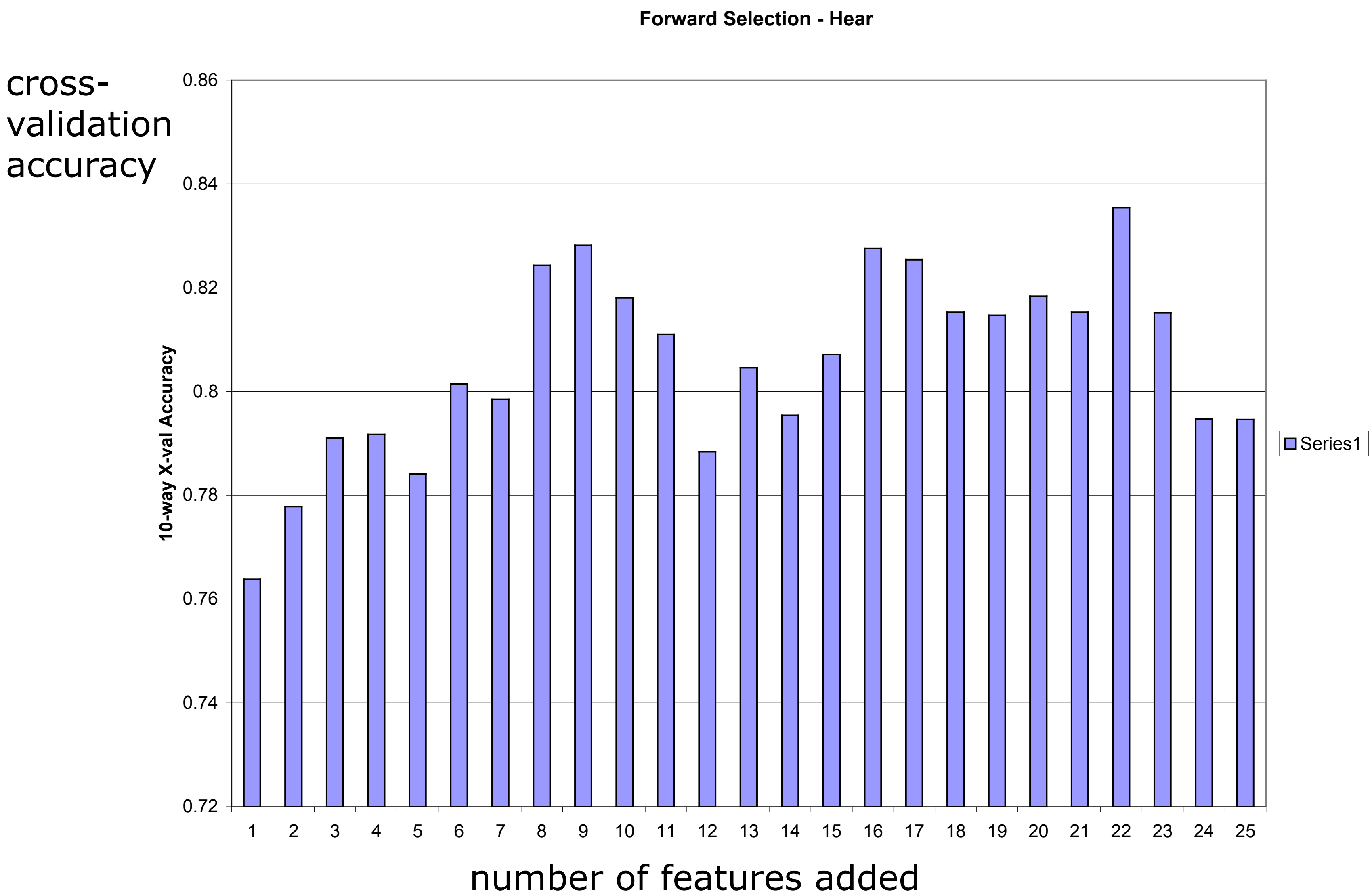
Backward Elimination on Auto Data

cross-validation accuracy

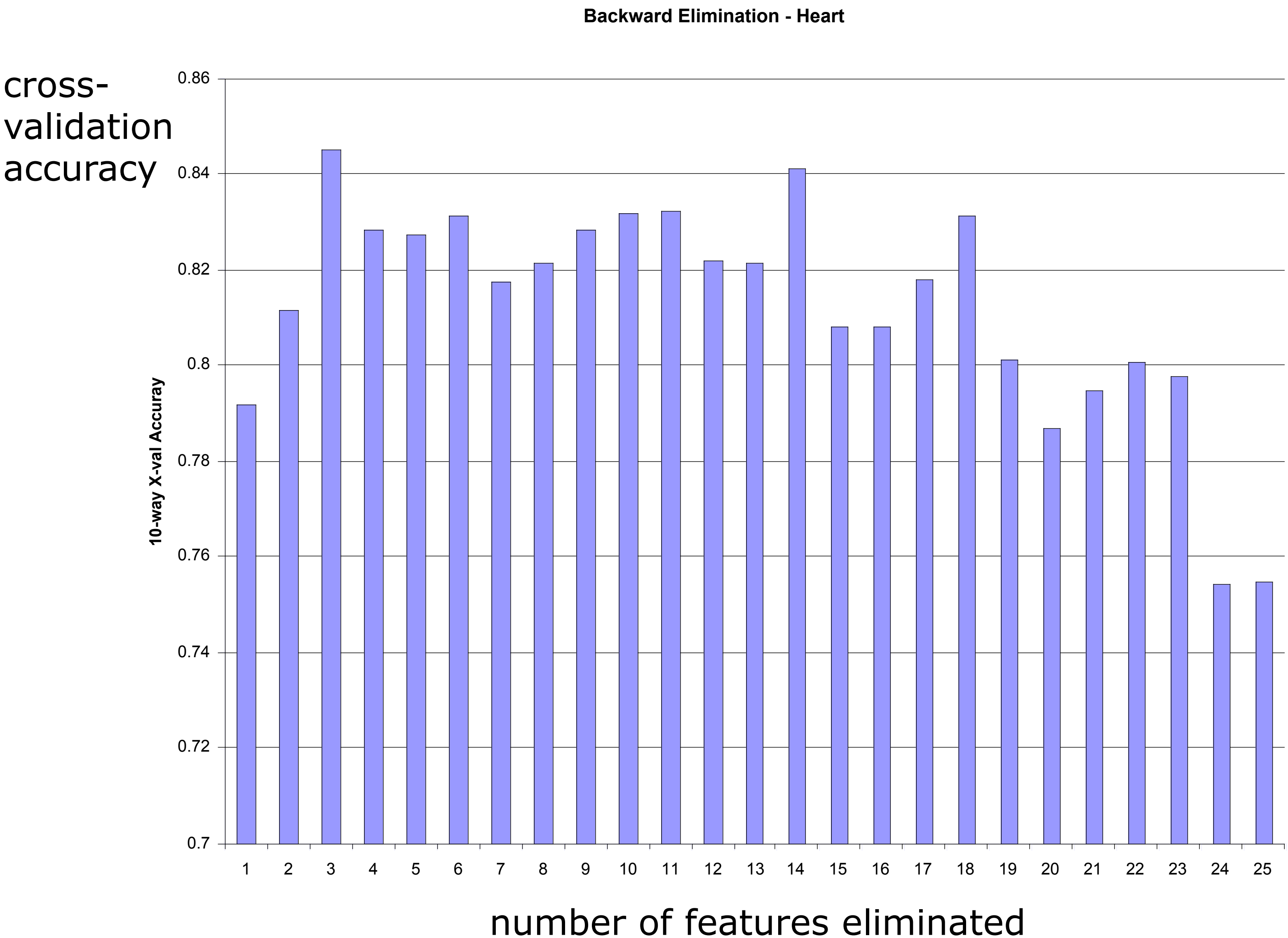


number of features eliminated

Forward Selection on Heart Data



Backward Elimination on Heart Data



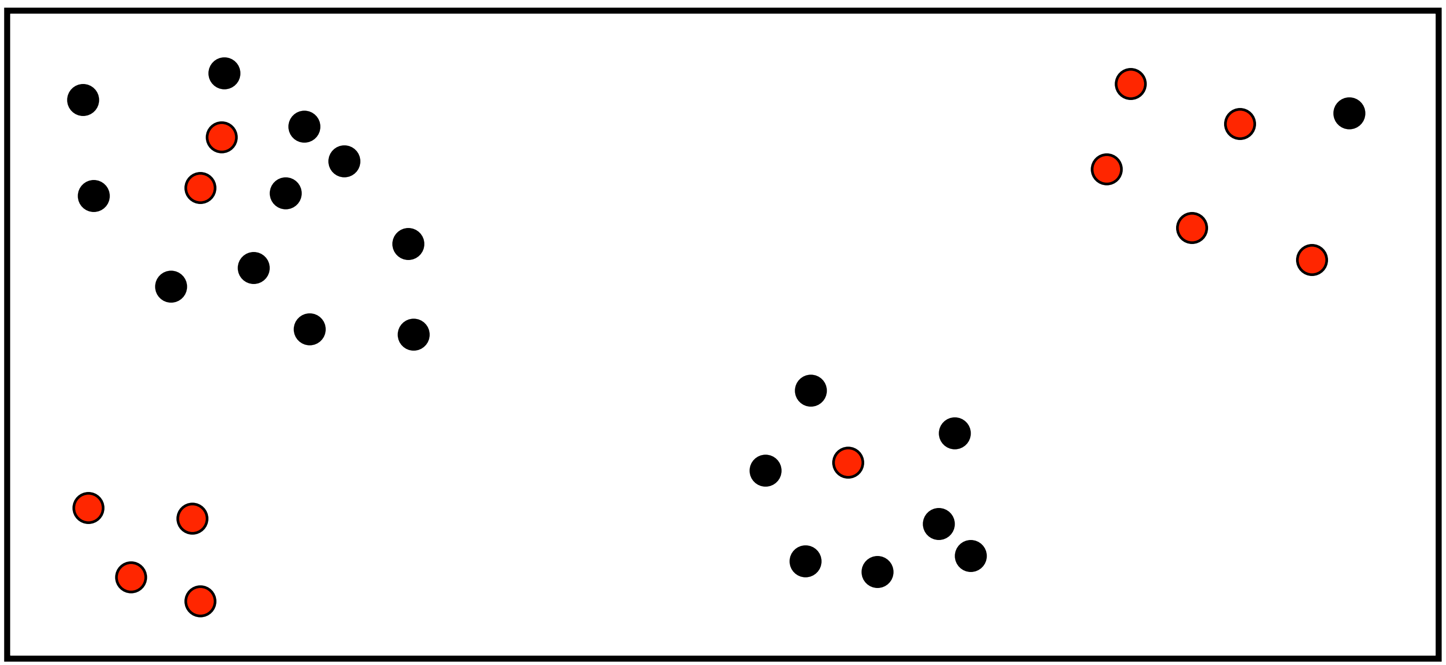
Recursive Feature Elimination

```
Train a linear SVM or neural network
Remove the feature with the smallest weight
Repeat
```

- More efficient than regular backward elimination
- Requires only one training phase per feature

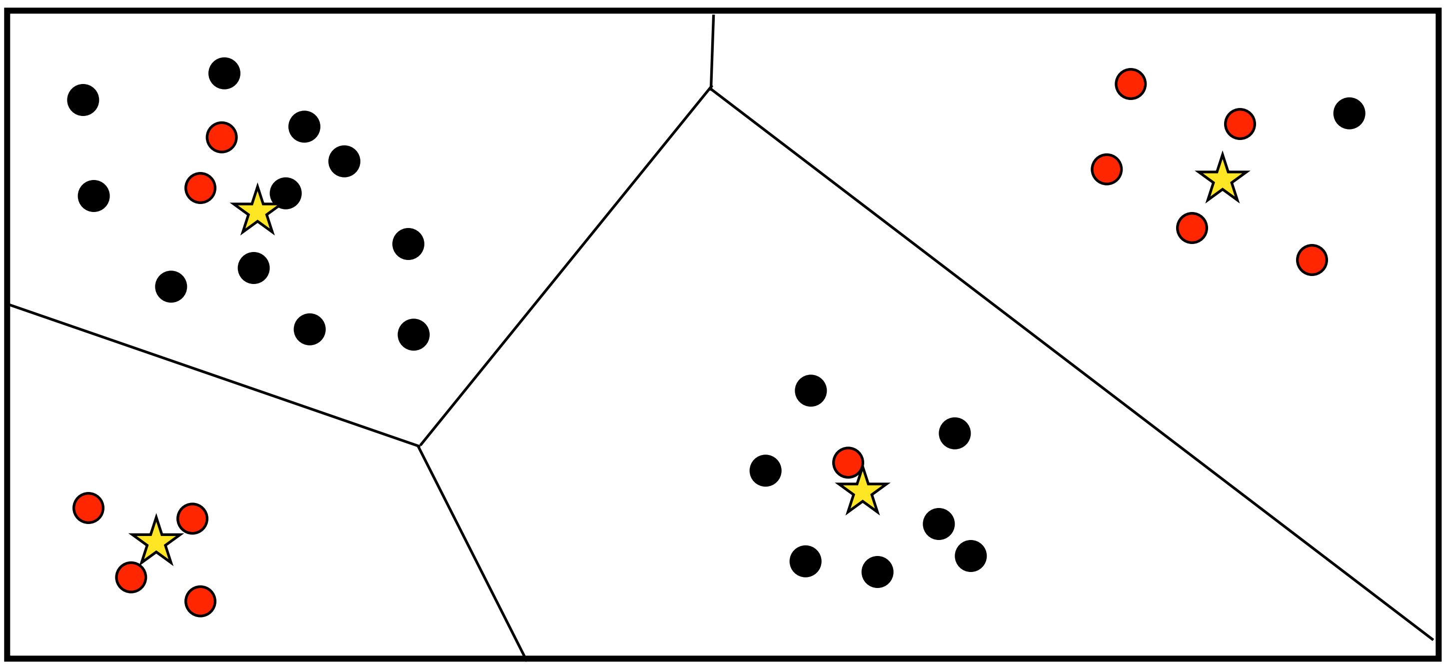
Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output



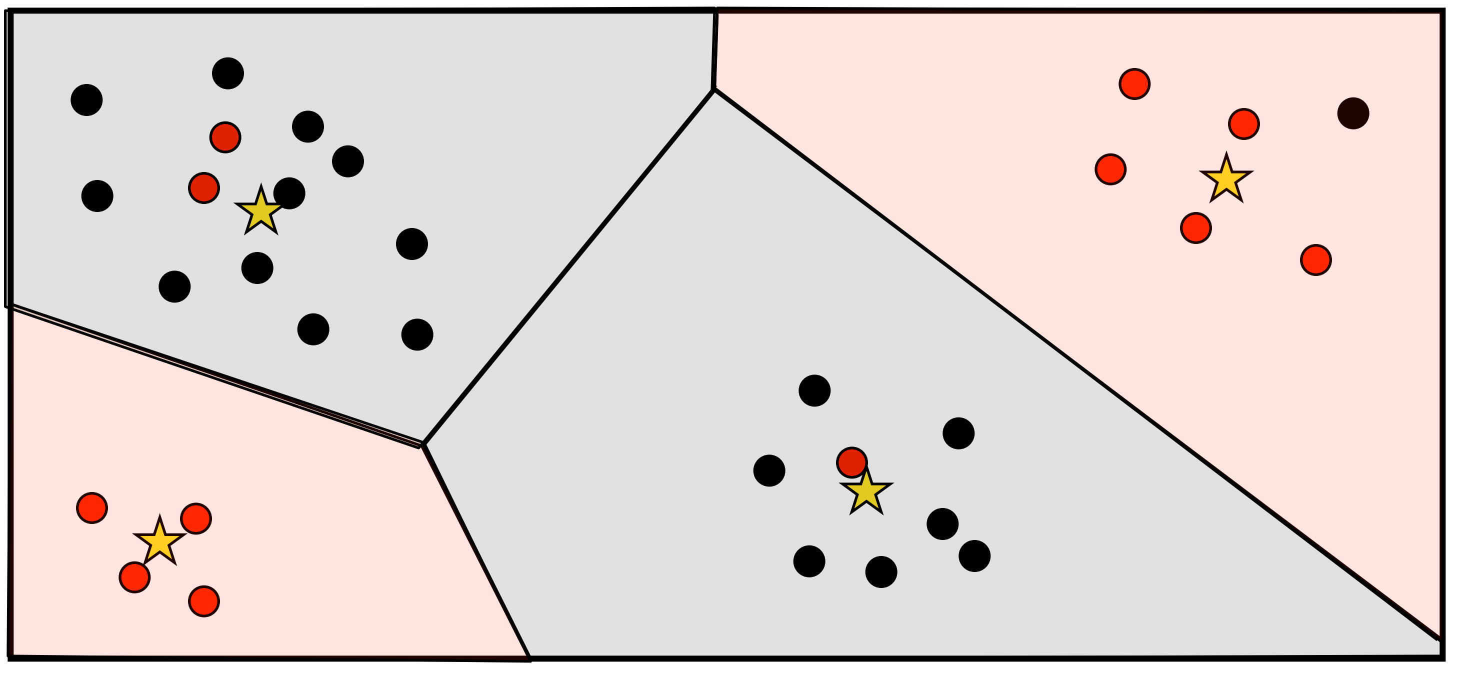
Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output



Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output



Clustering Criteria

- small distances between points within a cluster
- large distances between clusters

- Need a distance measure, as in nearest neighbor

K-Means Clustering

- Tries to minimize

$$\sum_{j=1}^{k} \sum_{i \in S_j} \overbrace{\left\| x^i - \mu_j \right\|^2}^{\text{squared dist from point to mean}}$$

# of clusters

elements of cluster j

mean of elts in cluster j

- Only gets, greedily, to a local optimum

K-means Algorithm

Choose k

Randomly choose k points Cj to be cluster centers

K-means Algorithm

```
Choose k
Randomly choose k points Cj to be cluster centers
Loop
   Partition the data into k classes Sj according
     to which of the Cj they're closest to
   For each Sj, compute the mean of its elements
     and let that be the new cluster center
```

K-means Algorithm

```
Choose k
Randomly choose k points Cj to be cluster centers
Loop
   Partition the data into k classes Sj according
     to which of the Cj they're closest to
   For each Sj, compute the mean of its elements
     and let that be the new cluster center
Stop when centers quit moving
```

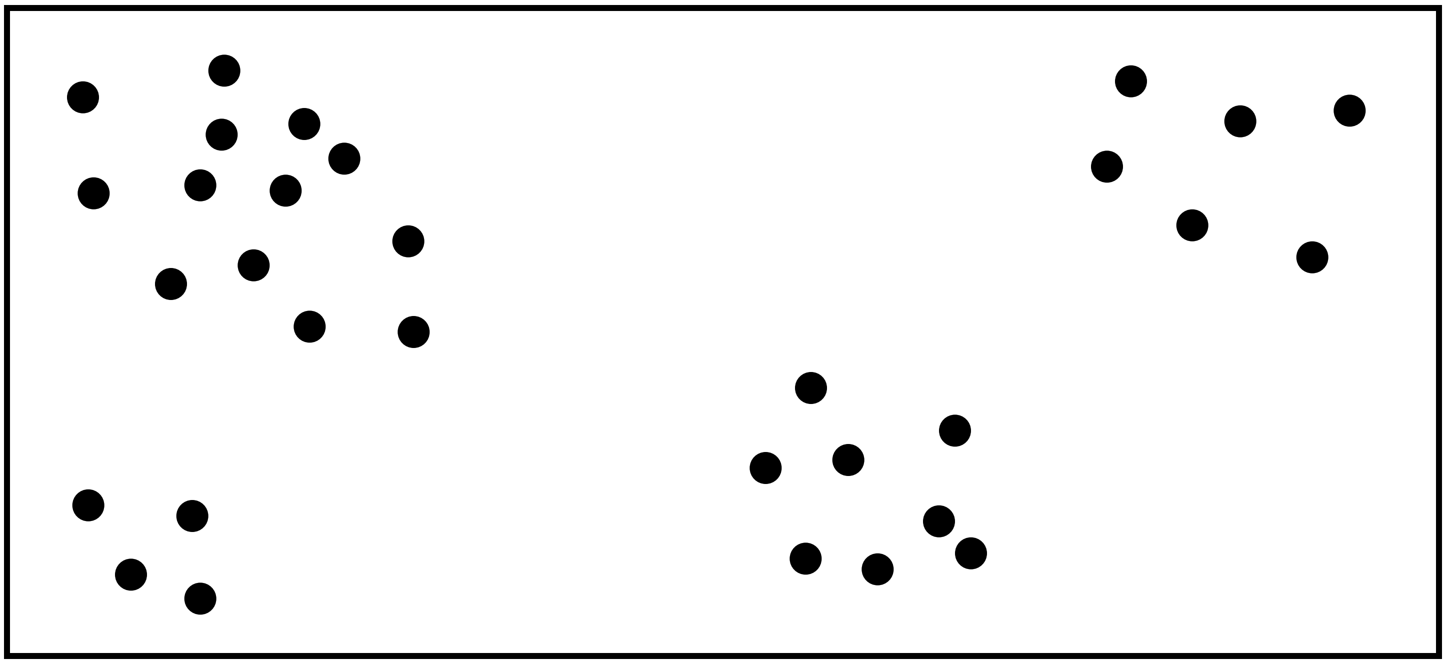
- Guaranteed to terminate

K-means Algorithm

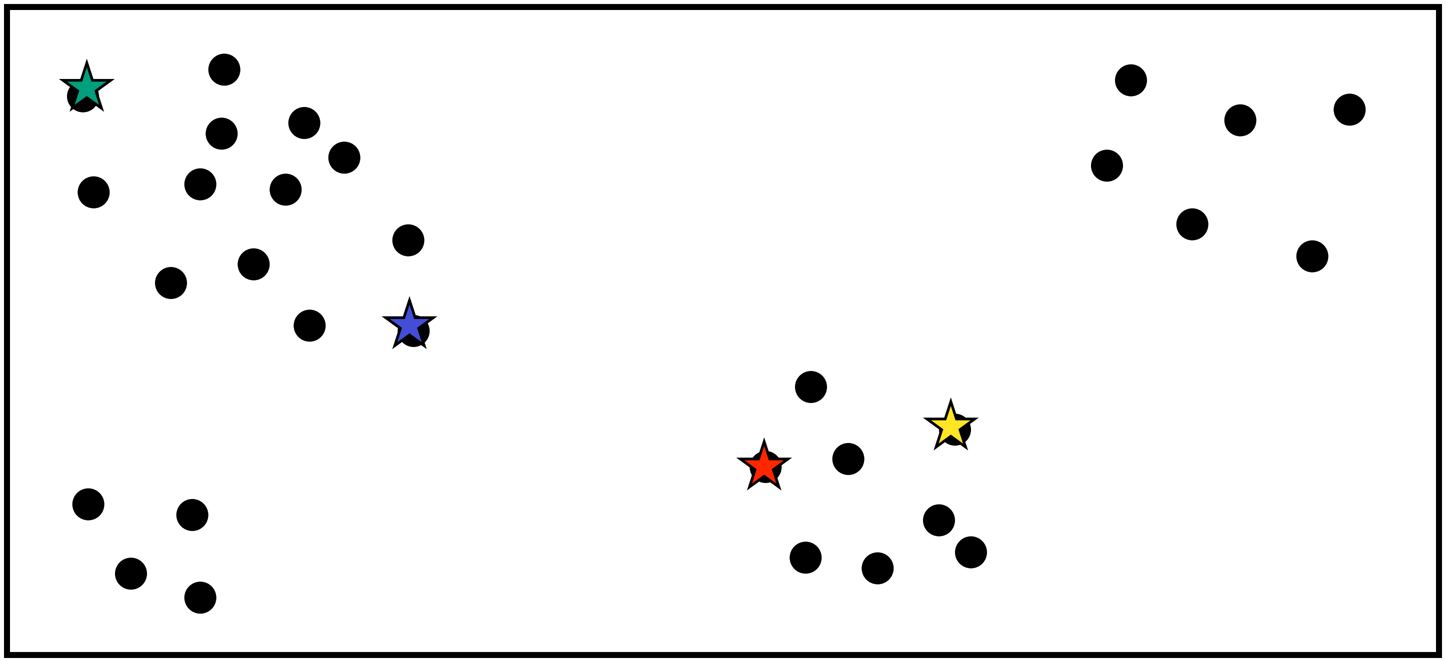
```
Choose k
Randomly choose k points Cj to be cluster centers
Loop
   Partition the data into k classes Sj according
     to which of the Cj they're closest to
   For each Sj, compute the mean of its elements
     and let that be the new cluster center
Stop when centers quit moving
```

- Guaranteed to terminate
- If a cluster becomes empty, re-initialize the center

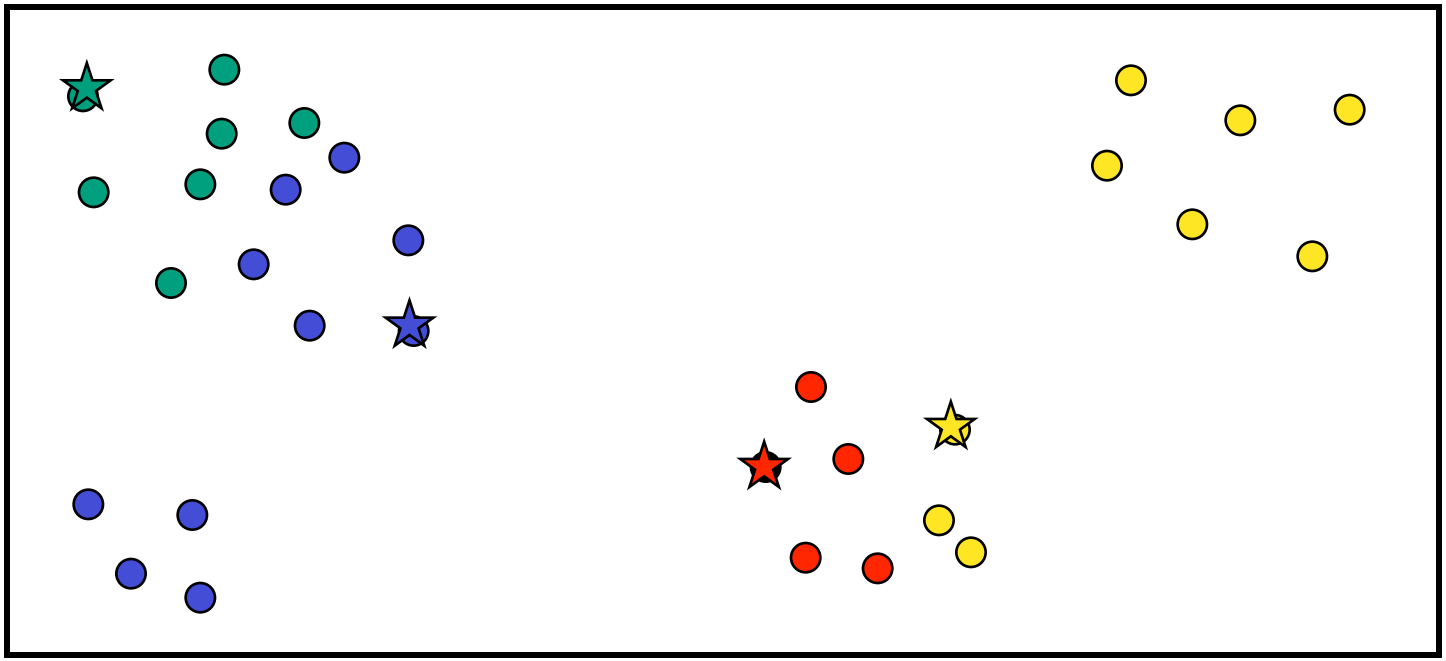
K-Means Example



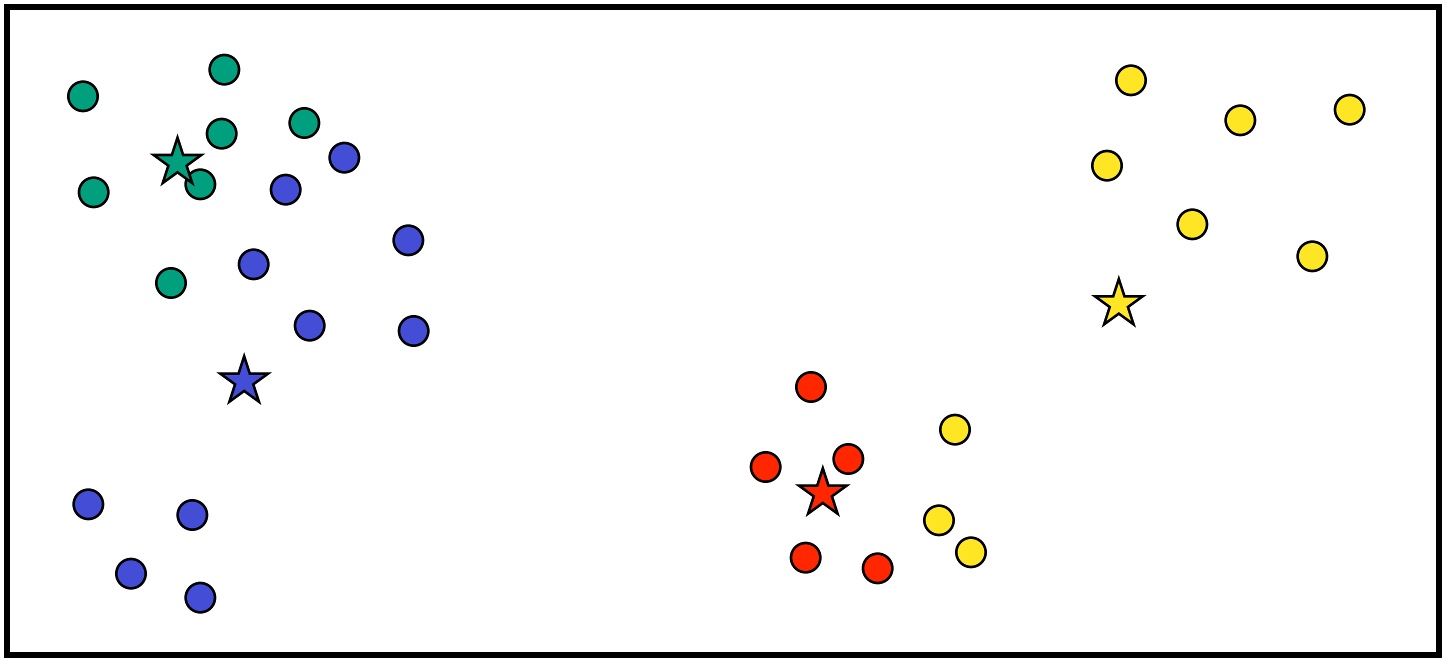
K-Means Example



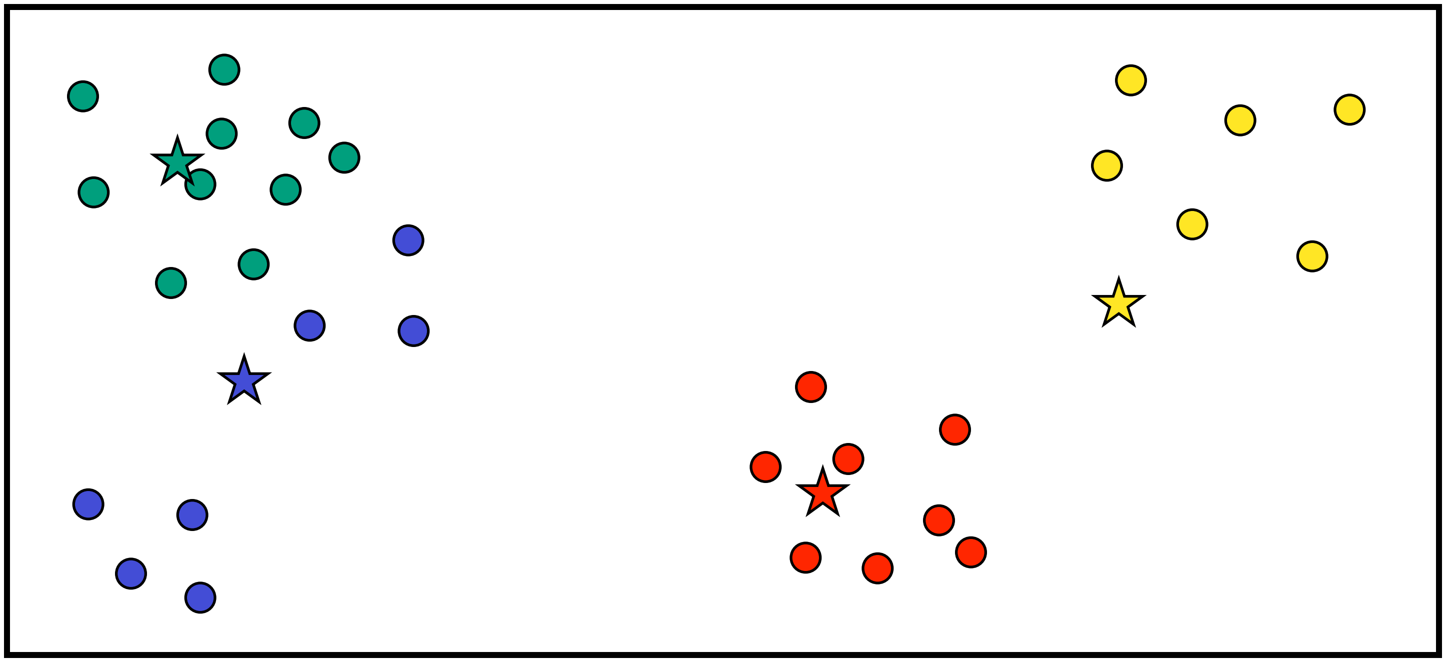
K-Means Example



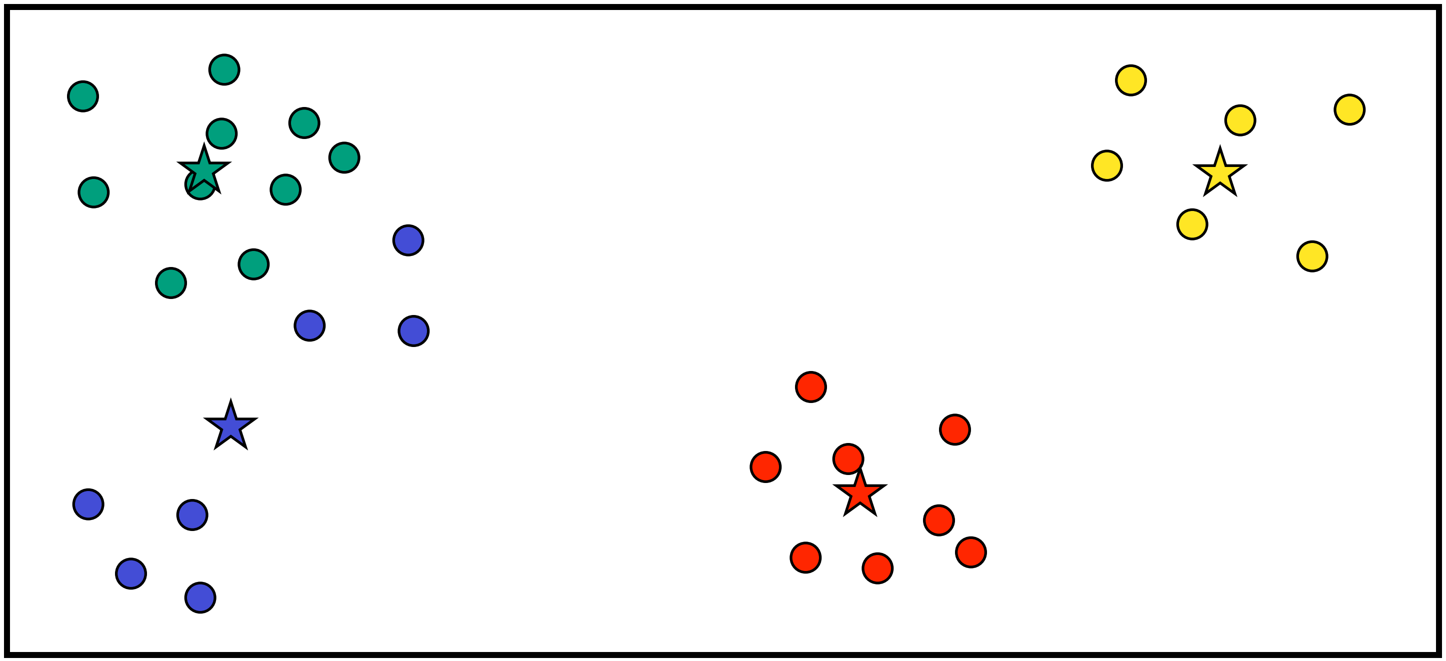
K-Means Example



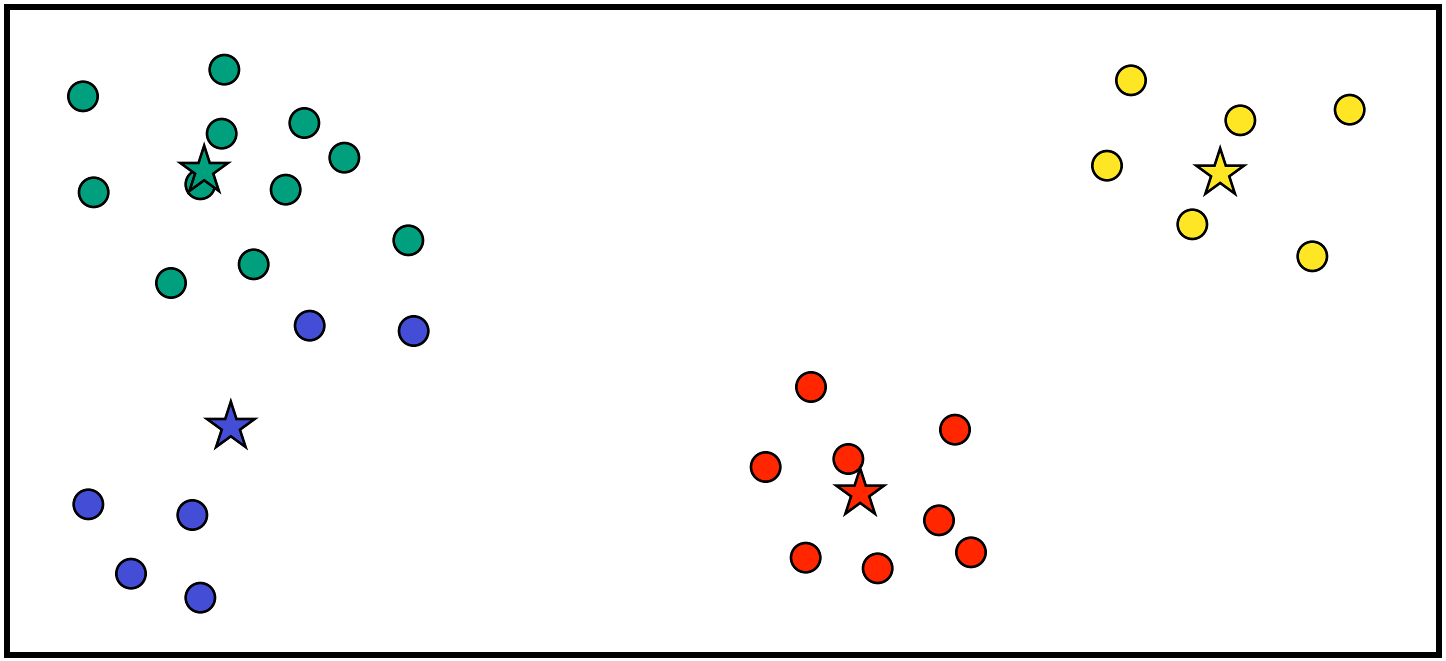
K-Means Example



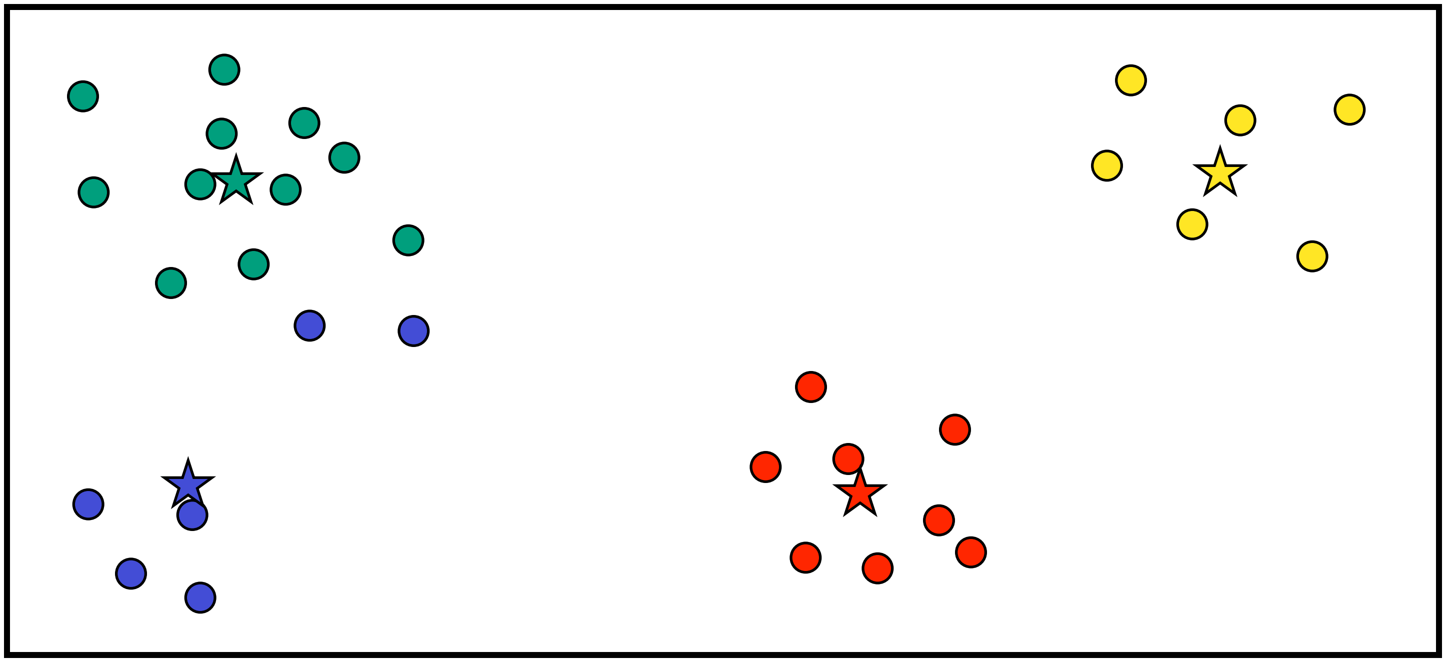
K-Means Example



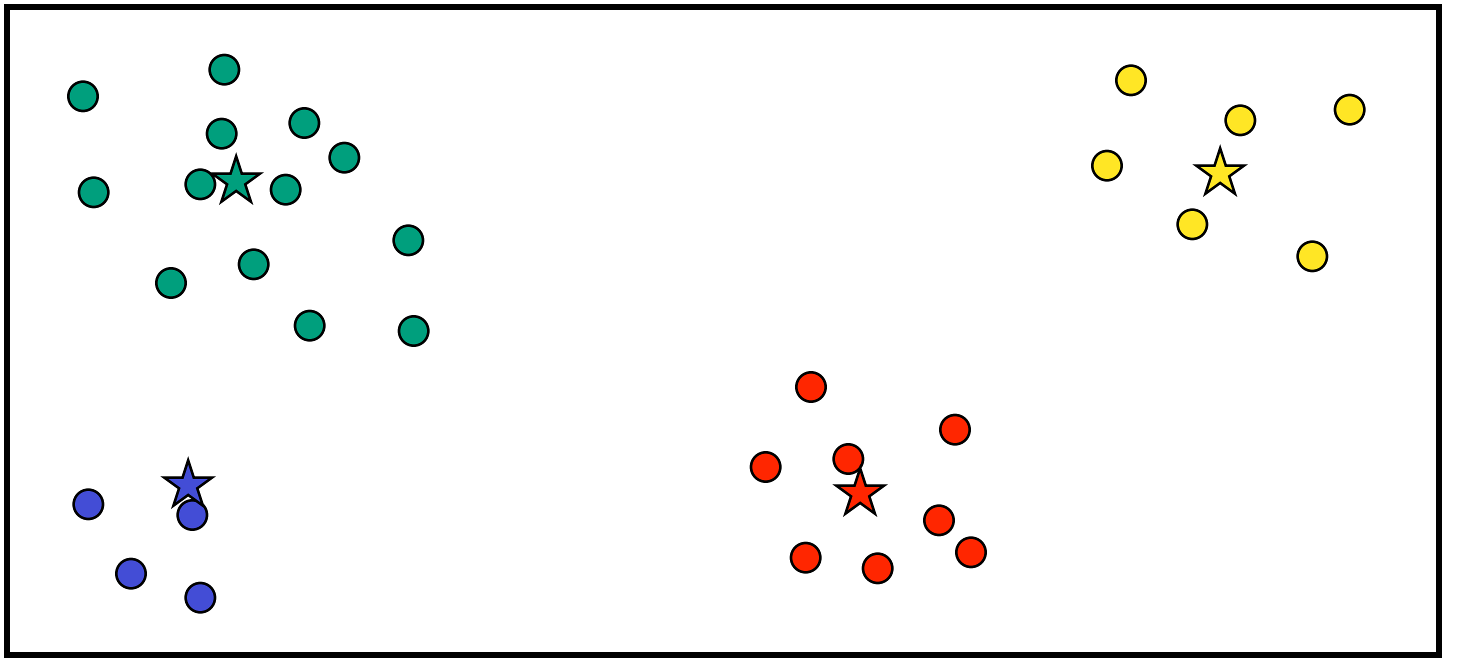
K-Means Example



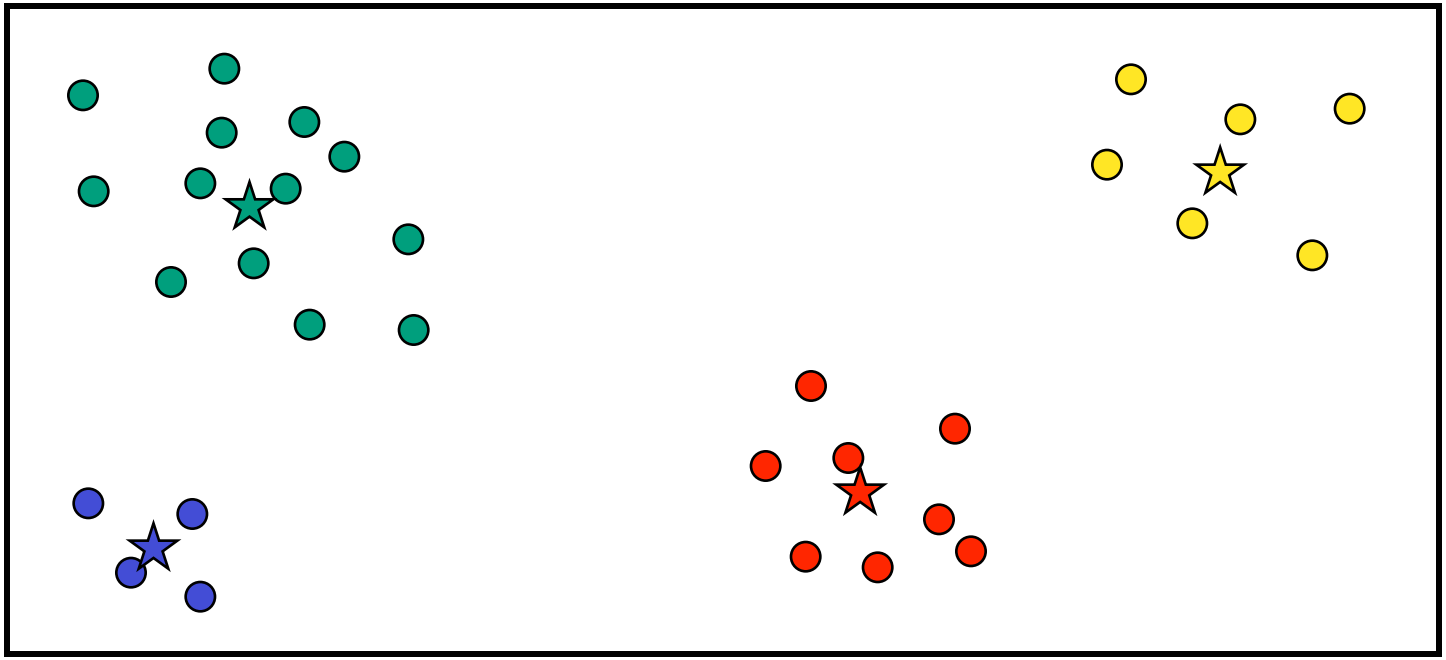
K-Means Example



K-Means Example



K-Means Example

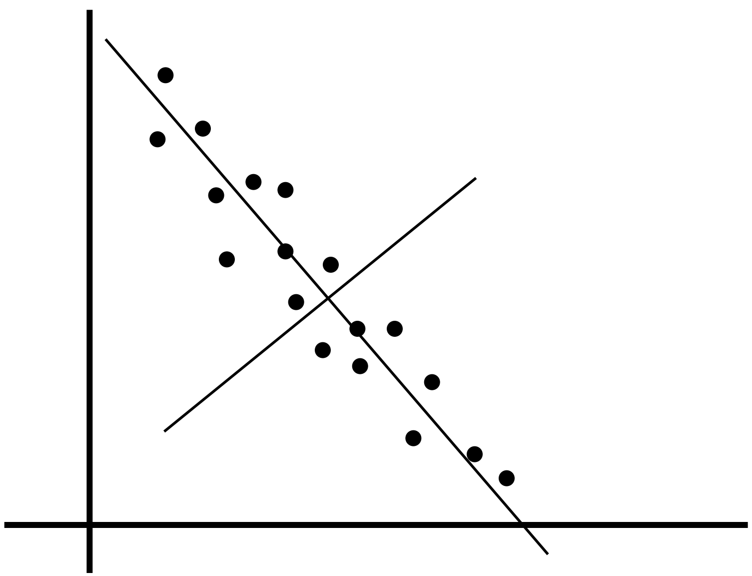


Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
- PCA helps us find such a subspace whose coordinates are linear functions of the originals

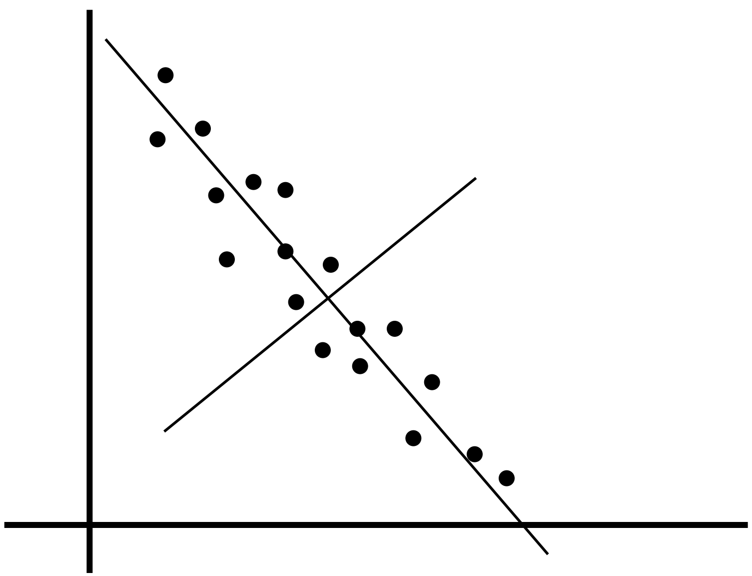
Principal Components Analysis

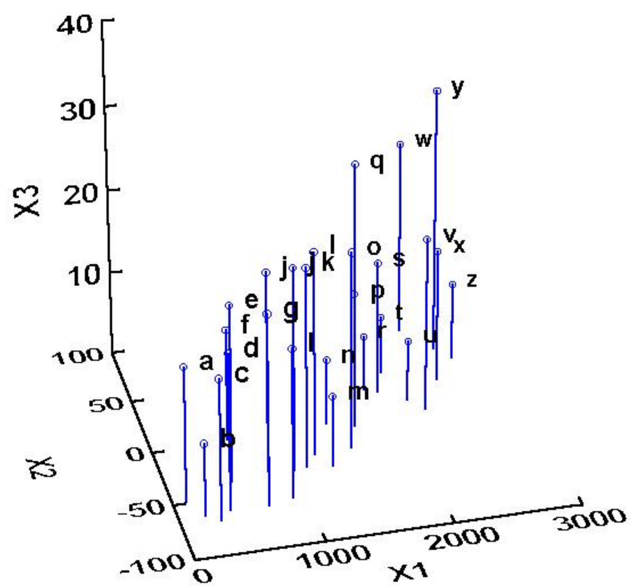
- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
- PCA helps us find such a subspace whose coordinates are linear functions of the originals



Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
- PCA helps us find such a subspace whose coordinates are linear functions of the originals





http://www.okstate.edu/artsci/botany/ordinate/PCA.htm

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

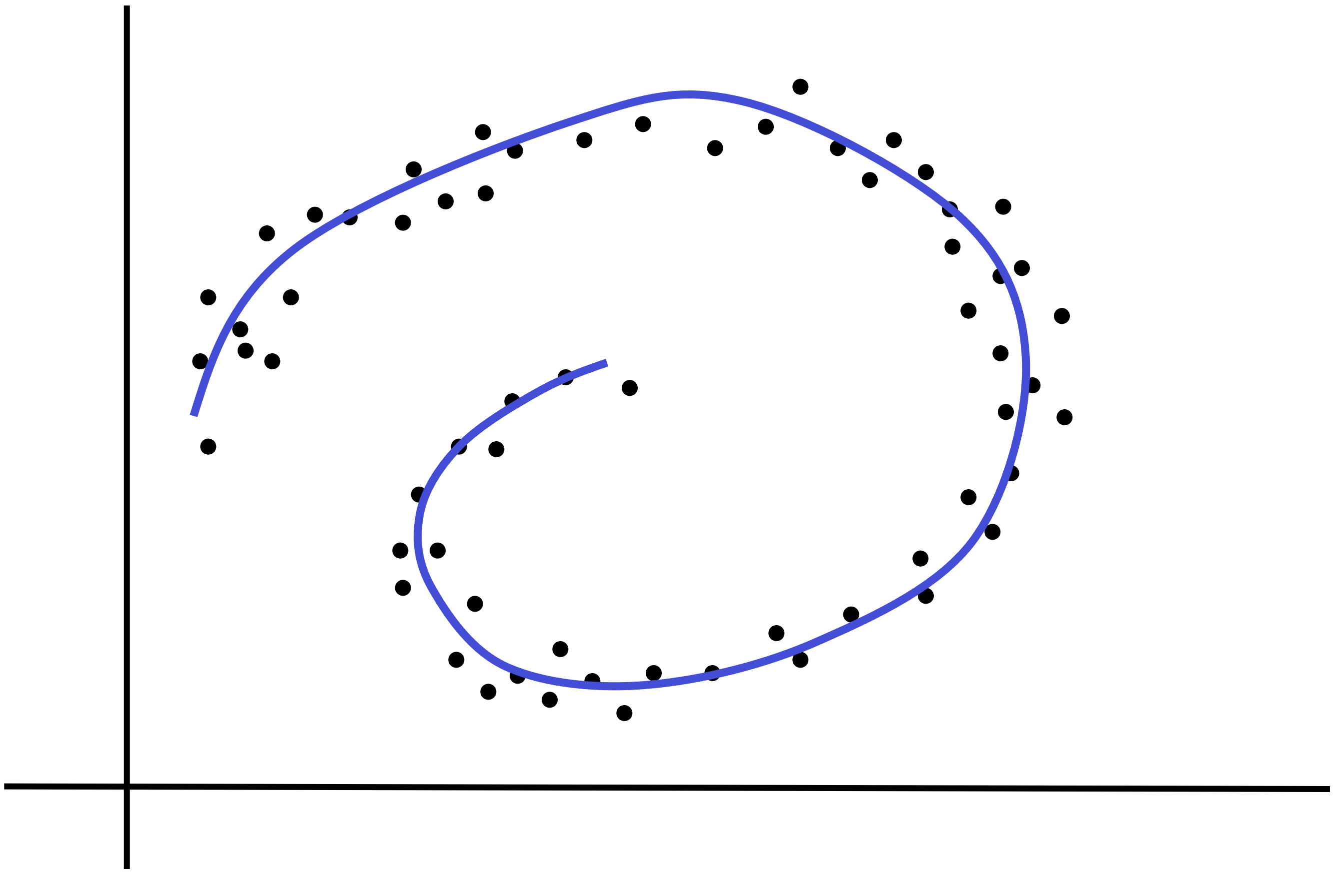
- Result is a new orthogonal set of axes
- First k give a lower-D space that represents the variability of the data as well as possible

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

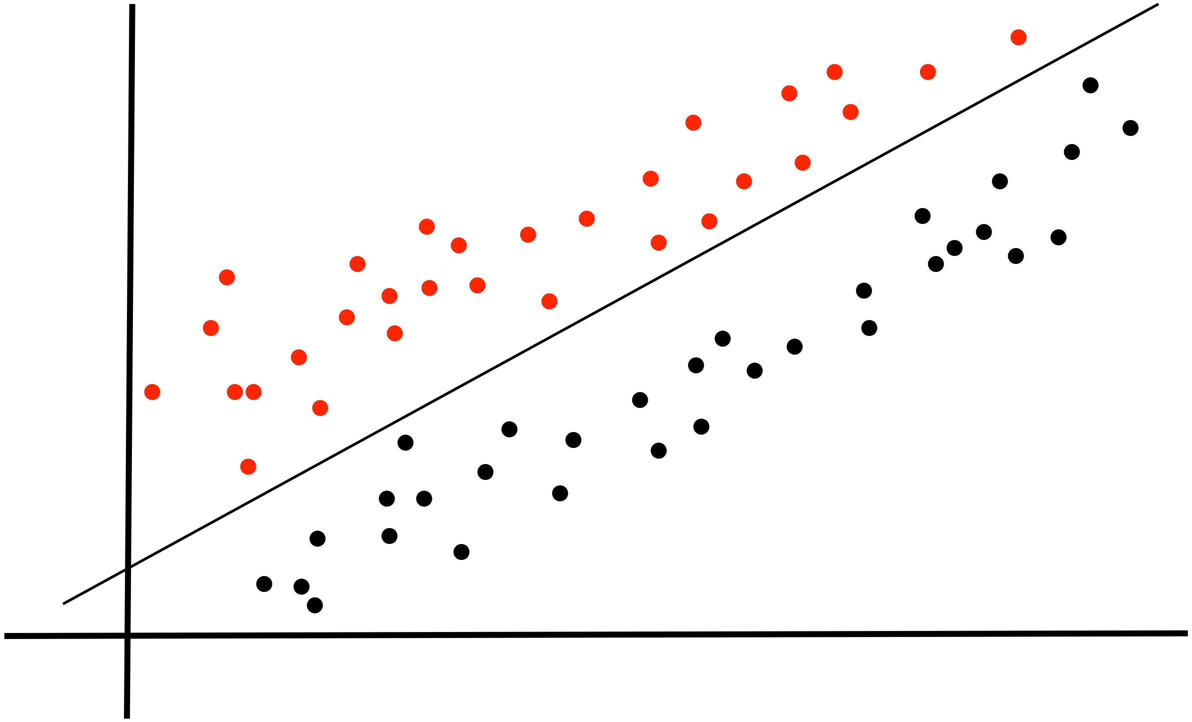
- Result is a new orthogonal set of axes
- First k give a lower-D space that represents the variability of the data as well as possible
- Really: find the eigenvectors of the covariance matrix with the k largest eigenvalues

Linear Transformations Only

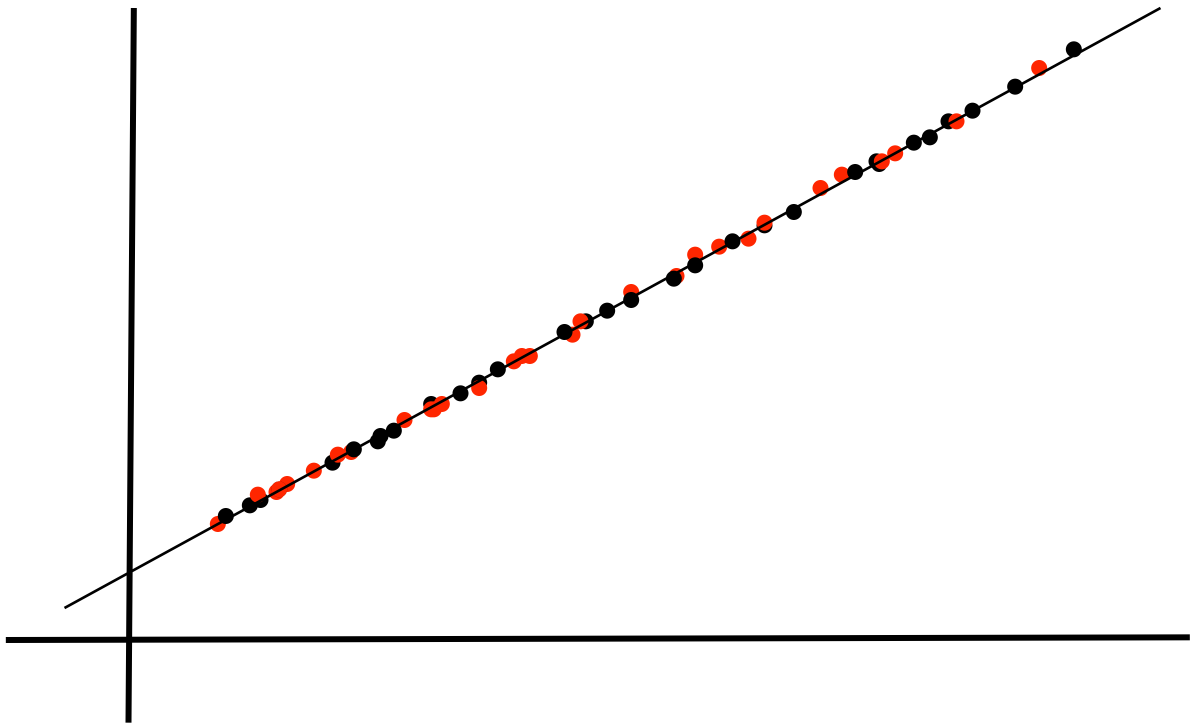


There are fancier methods that can find this structure

Insensitive to Classification Task



Insensitive to Classification Task



There are fancier methods that can take class into account

Validating a Classifier

		predicted y 0	predicted y 1
true y	0	A	B
	1	C	D

Validating a Classifier

		predicted y	
		0	1
true y	0	A	B
	1	C	D

false positive
type 1 error

Validating a Classifier

		predicted y 0	predicted y 1
true y	0	A	B
	1	C	D

false positive
type 1 error

false negative
type 2 error

Validating a Classifier

		predicted y 0	predicted y 1
true y	0	A	B
	1	C	D

false positive
type 1 error

false negative
type 2 error

- sensitivity: P(predict 1 | actual 1) = D/(C+D)
  - “true positive rate” (TP)

Validating a Classifier

predicted y

		0	1
true y	0	A	B
	1	C	D

false positive
type 1 error

false negative
type 2 error

- sensitivity: P(predict 1 | actual 1) = D/(C+D)
  - “true positive rate” (TP)

- specificity: P(predict 0 | actual 0) = A/(A+B)

Validating a Classifier

		predicted y 0	predicted y 1
true y	0	A	B
	1	C	D

false positive
type 1 error

false negative
type 2 error

- sensitivity: P(predict 1 | actual 1) = D/(C+D)
  - "true positive rate" (TP)

- specificity: P(predict 0 | actual 0) = A/(A+B)

- false-alarm rate: P(predict 1 | actual 0) = B/(A+B)
  - "false positive rate" (FP)

Cost Sensitivity

- Predict whether a patient has pseudititis based on blood tests
  - Disease is often fatal if left untreated
  - Treatment is cheap and side-effect free

Cost Sensitivity

- Predict whether a patient has pseuditis based on blood tests
  - Disease is often fatal if left untreated
  - Treatment is cheap and side-effect free

- Which classifier to use?
  - Classifier 1: TP = 0.9, FP = 0.4

Cost Sensitivity

- Predict whether a patient has pseudītis based on blood tests
  - Disease is often fatal if left untreated
  - Treatment is cheap and side-effect free

- Which classifier to use?
  - Classifier 1: TP = 0.9, FP = 0.4
  - Classifier 2: TP = 0.7, FP = 0.1

Build Costs into Classifier

- Assess costs of both types of error
  - use a different splitting criterion for decision trees
  - make error function for neural nets asymmetric; different costs for each kind of error
  - use different values of C for SVMs depending on kind of error

Tunable Classifiers

- Classifiers that have a threshold (naïve Bayes, neural nets, SVMs) can be adjusted, post learning, by changing the threshold, to make different trade-offs between type 1 and type 2 errors

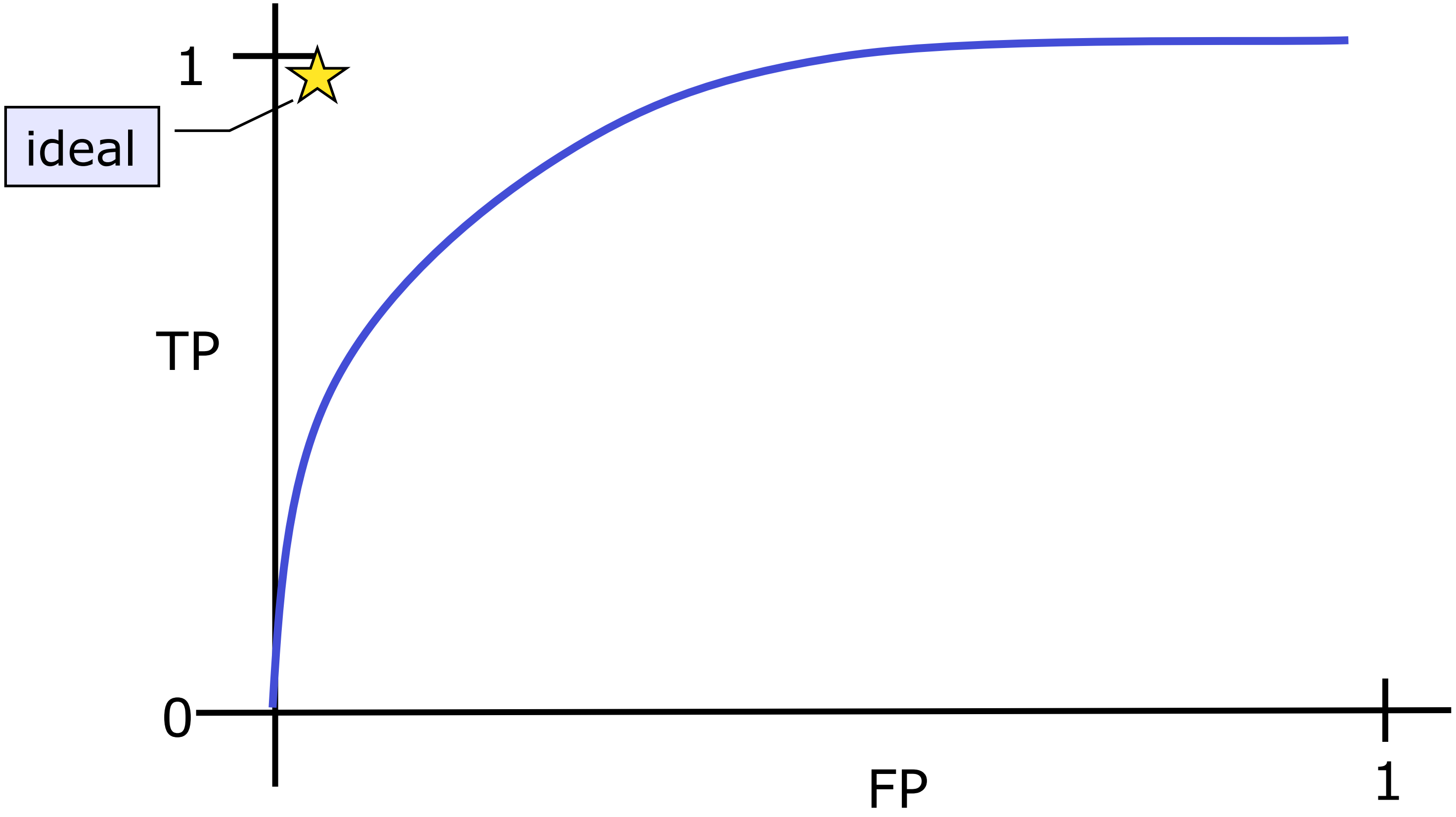
Tunable Classifiers

- Classifiers that have a threshold (naïve Bayes, neural nets, SVMs) can be adjusted, post learning, by changing the threshold, to make different trade-offs between type 1 and type 2 errors

  - C_1,C_2: costs of errors
  - P: percentage of positive examples
  - x: tunable threshold
  - TP(x): true positive rate at threshold x
  - FP(x): false positive rate at threshold x
- Expected Cost = $C_2P(1\text{-}TP(x)) + C_1(1\text{-}P)FP(x)$
- choose x to minimize expected cost

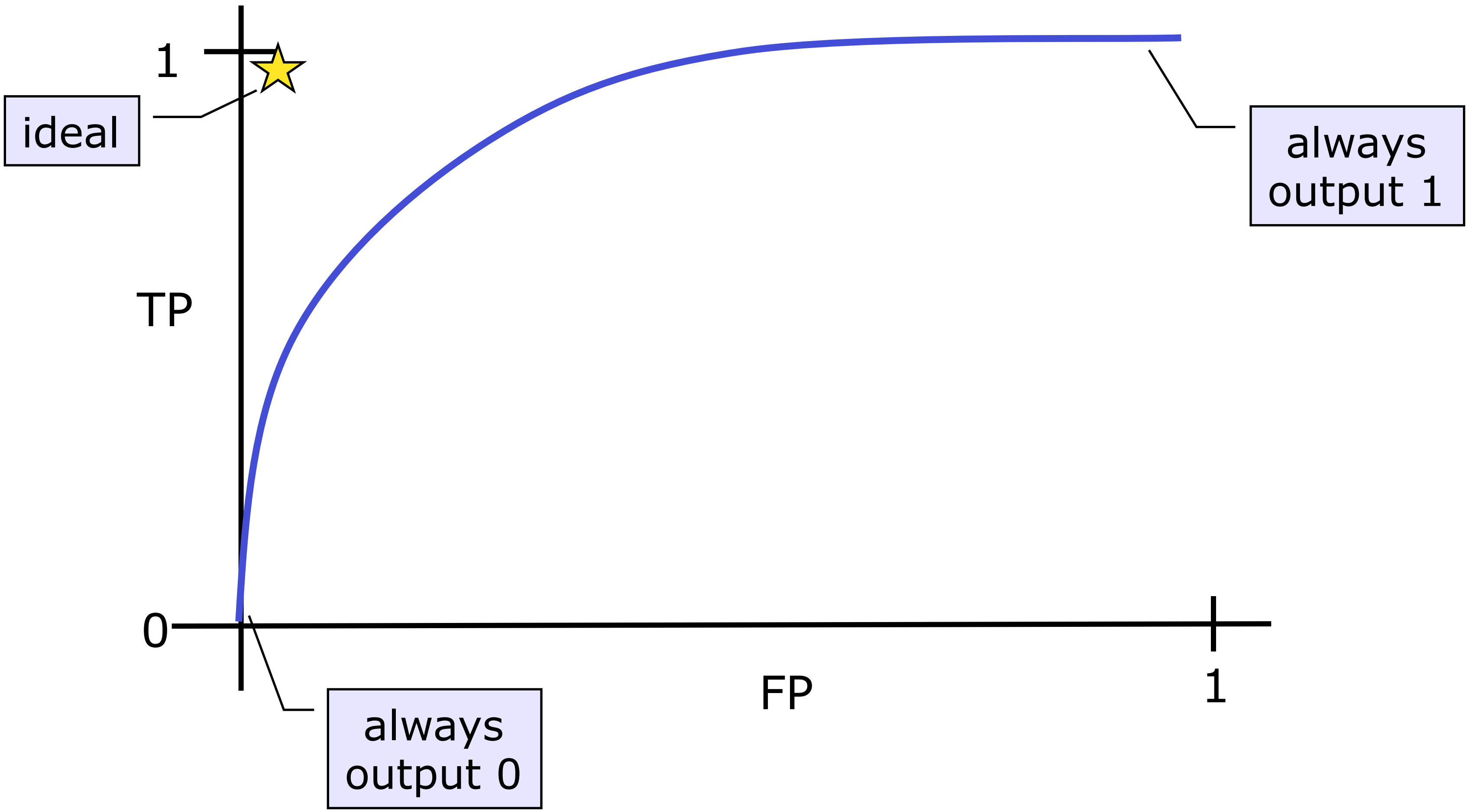
ROC Curves

- “receiver operating characteristics”



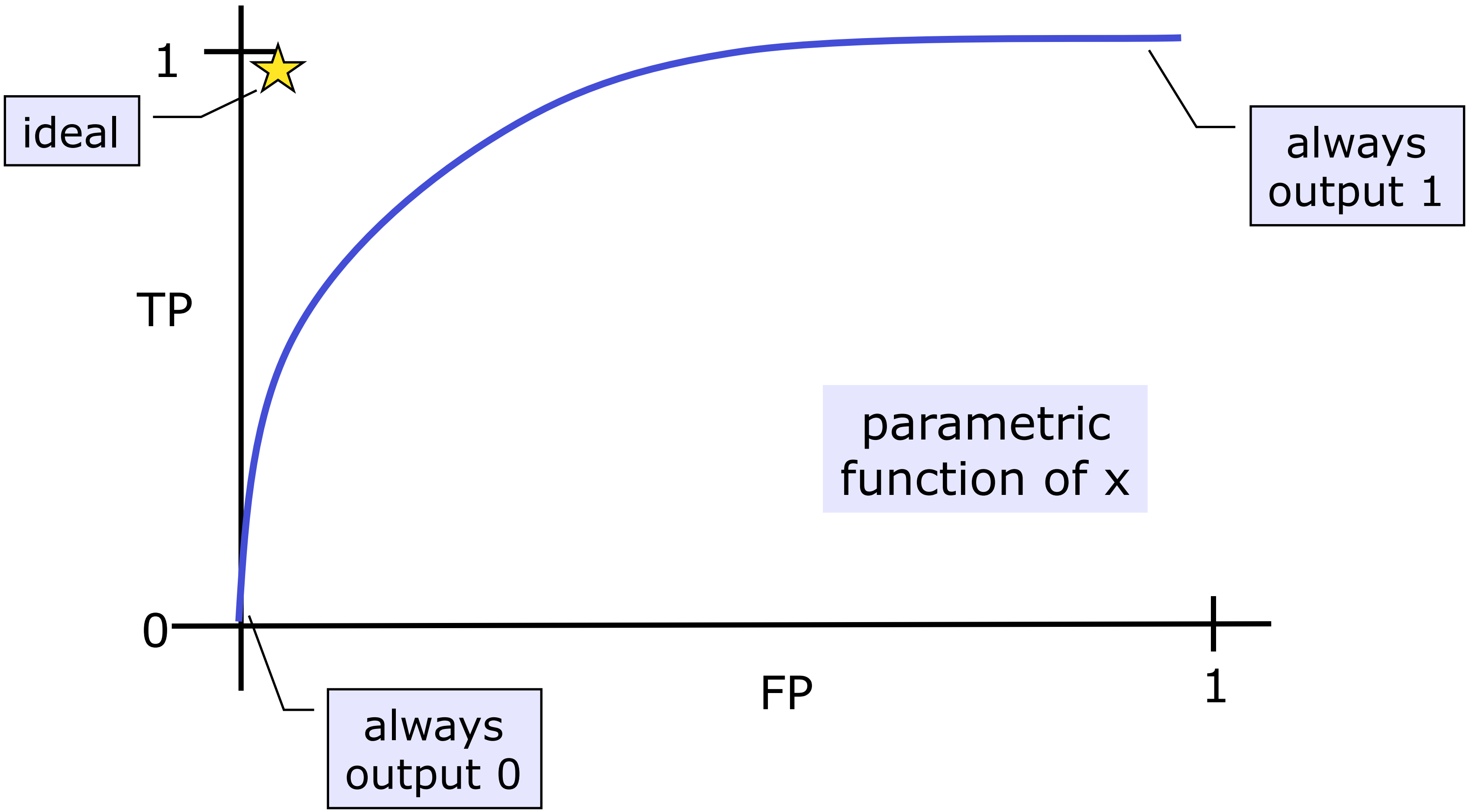
ROC Curves

- “receiver operating characteristics”



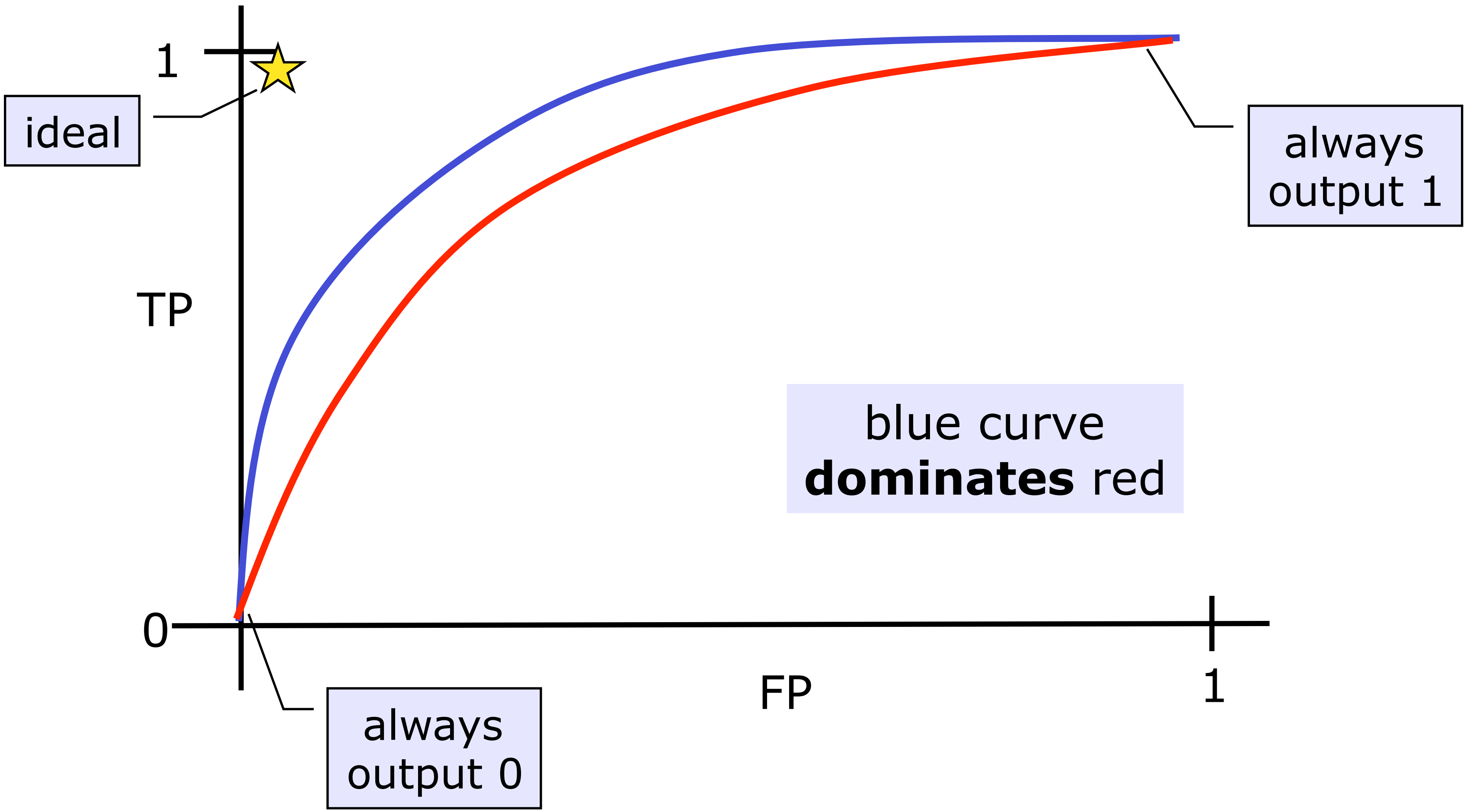
ROC Curves

- "receiver operating characteristics"



ROC Curves

- "receiver operating characteristics"



Many more issues!

- Missing data
- Many examples in one class, few in other (fraud detection)
- Expensive data (active learning)
- …