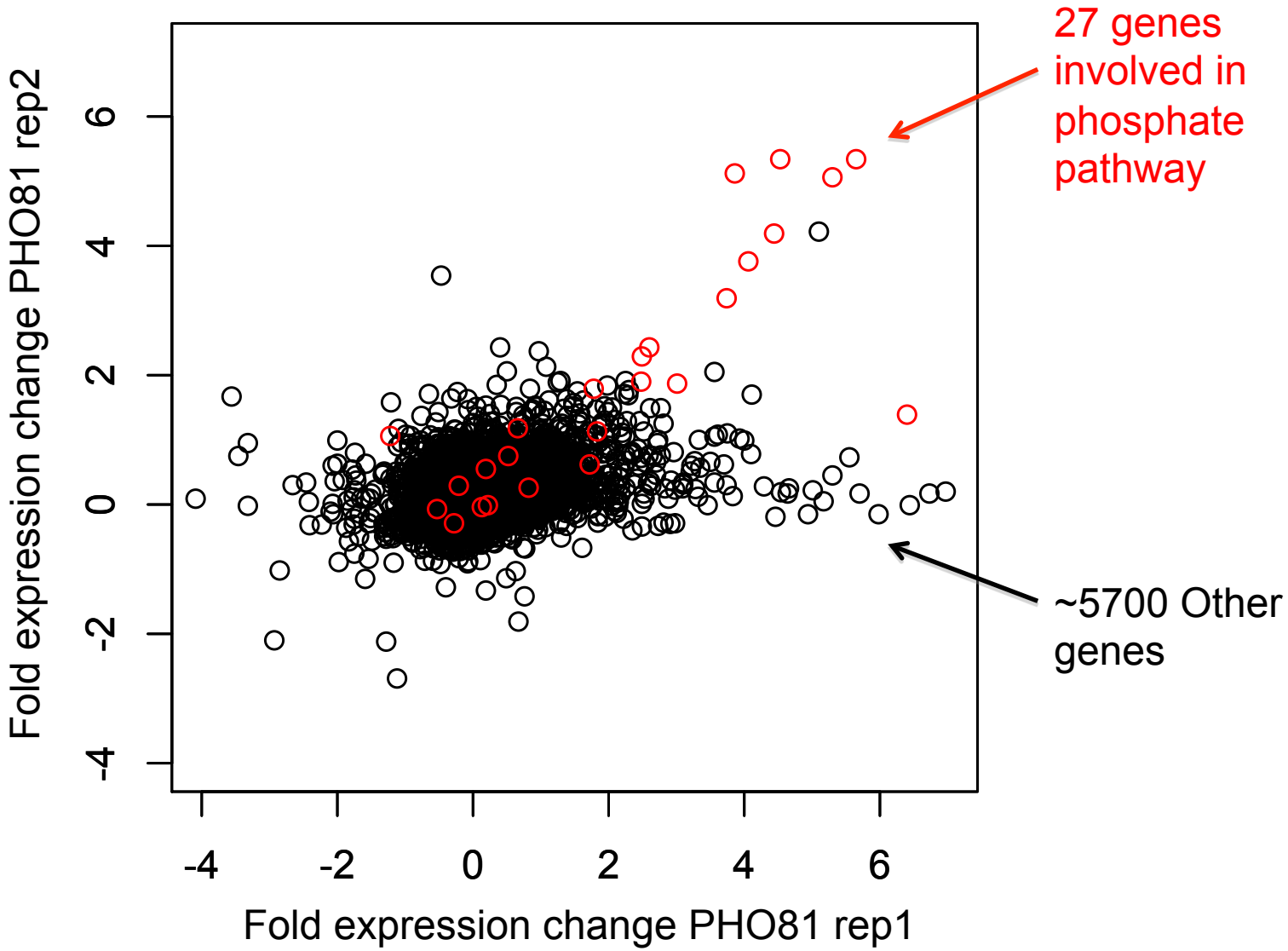
ML4Bio
Lecture #4: Classification

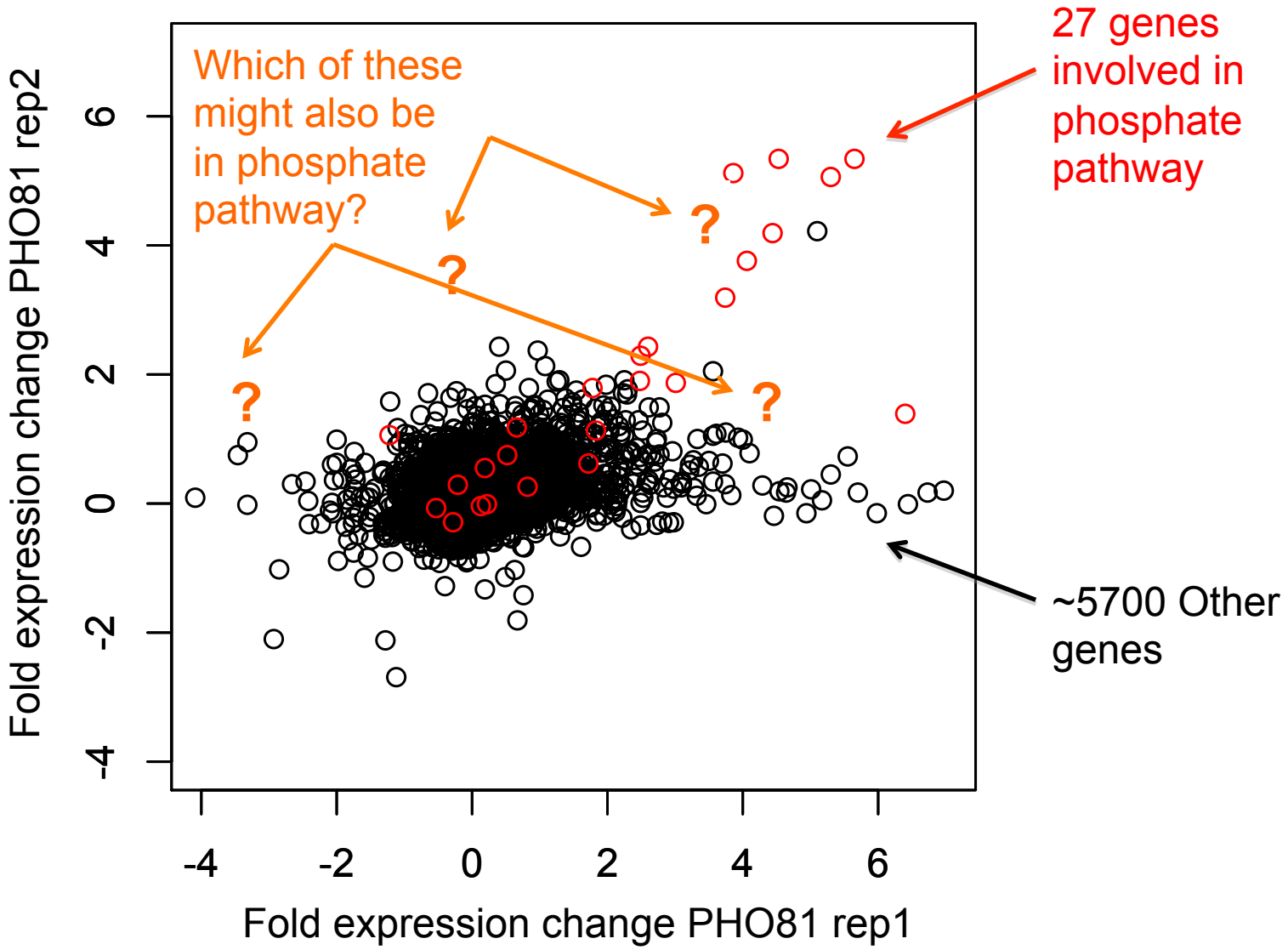
March 16th, 2016

Quaid Morris

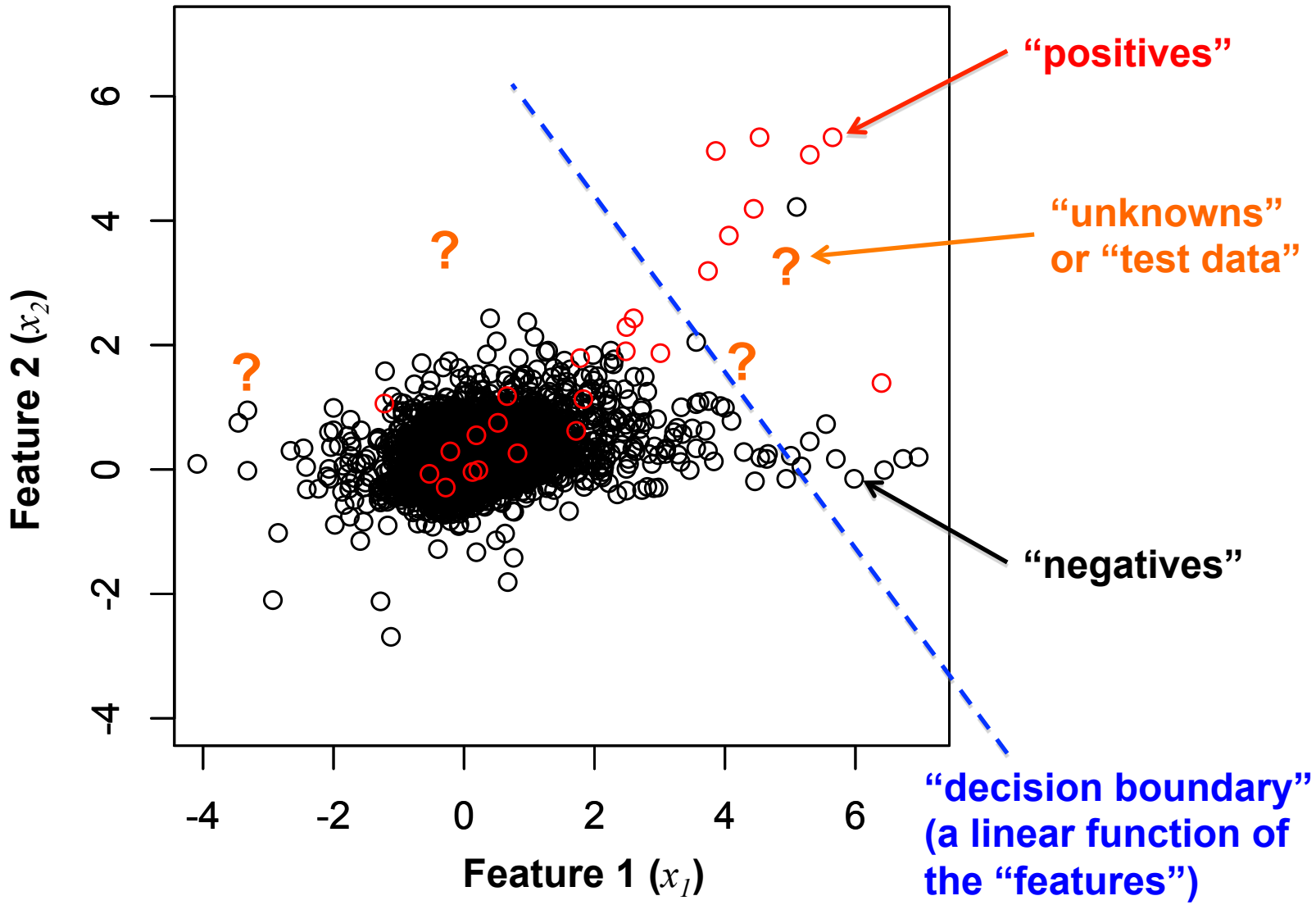
Predicting gene function



Predicting gene function

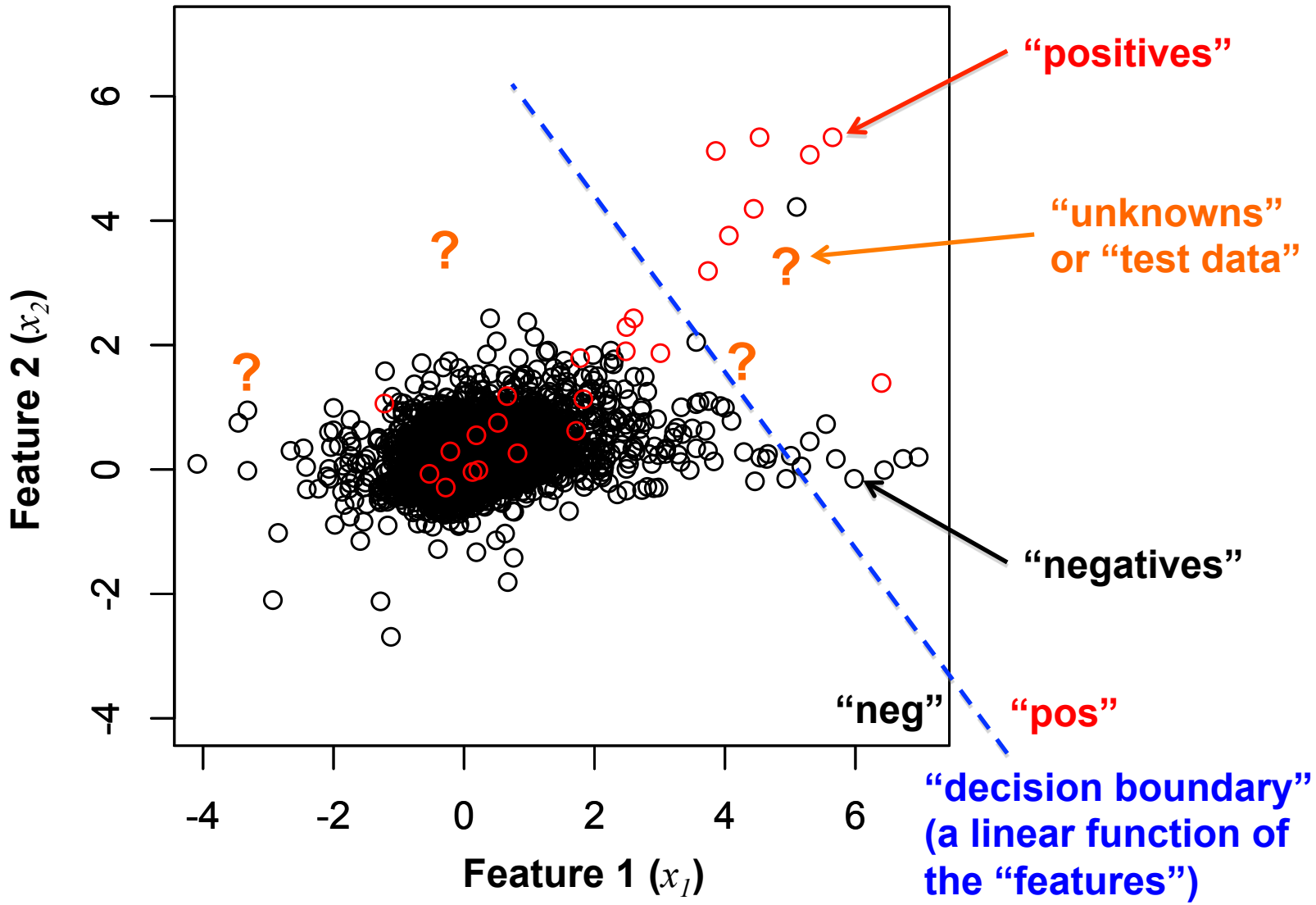


A ***linear classifier*** will try to find a line** that separates the two classes



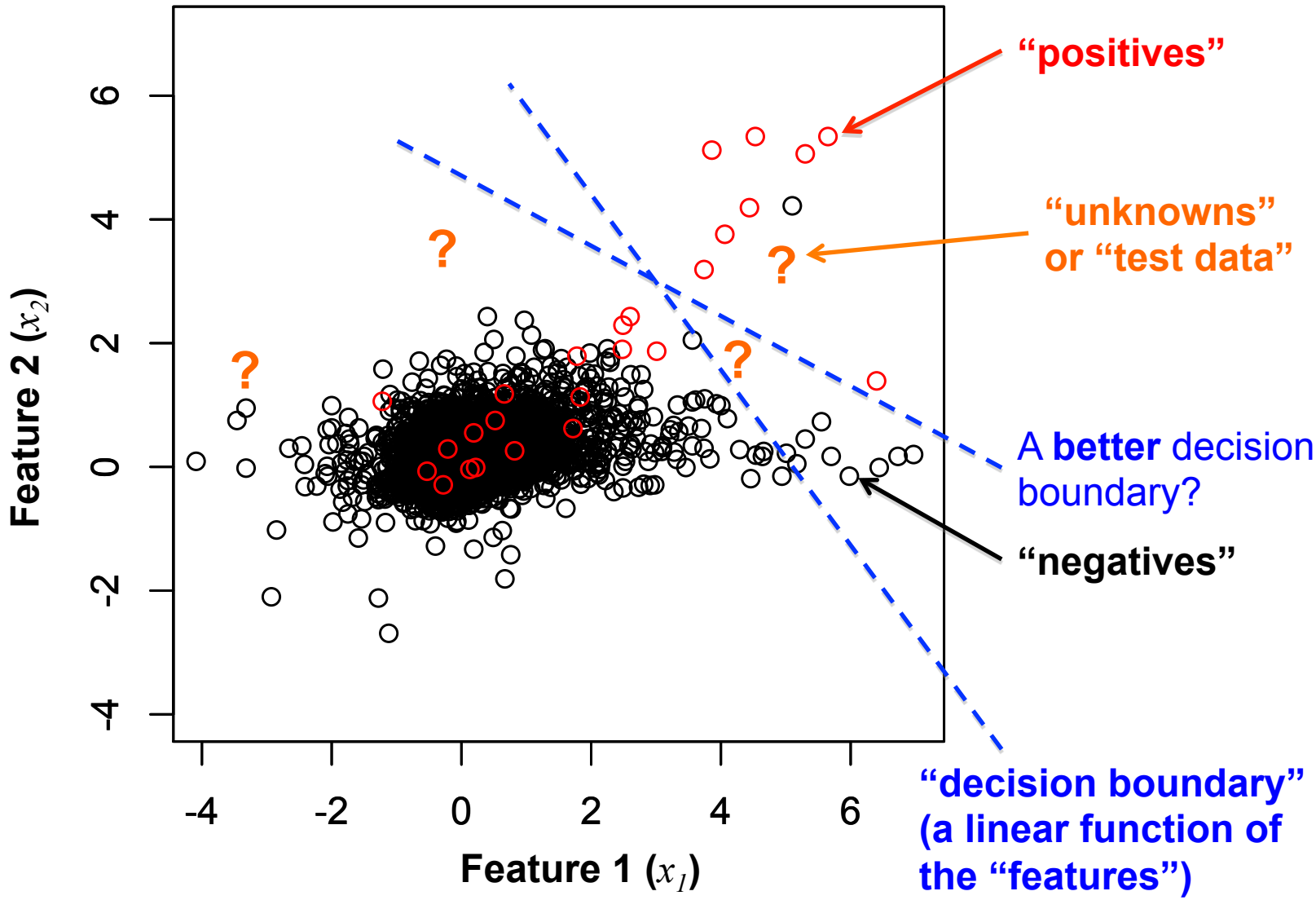
** only a line in 2-d, a plane in 3-d, & hyperplane in >3-d

A ***linear classifier*** will try to find a line**
that separates the two classes



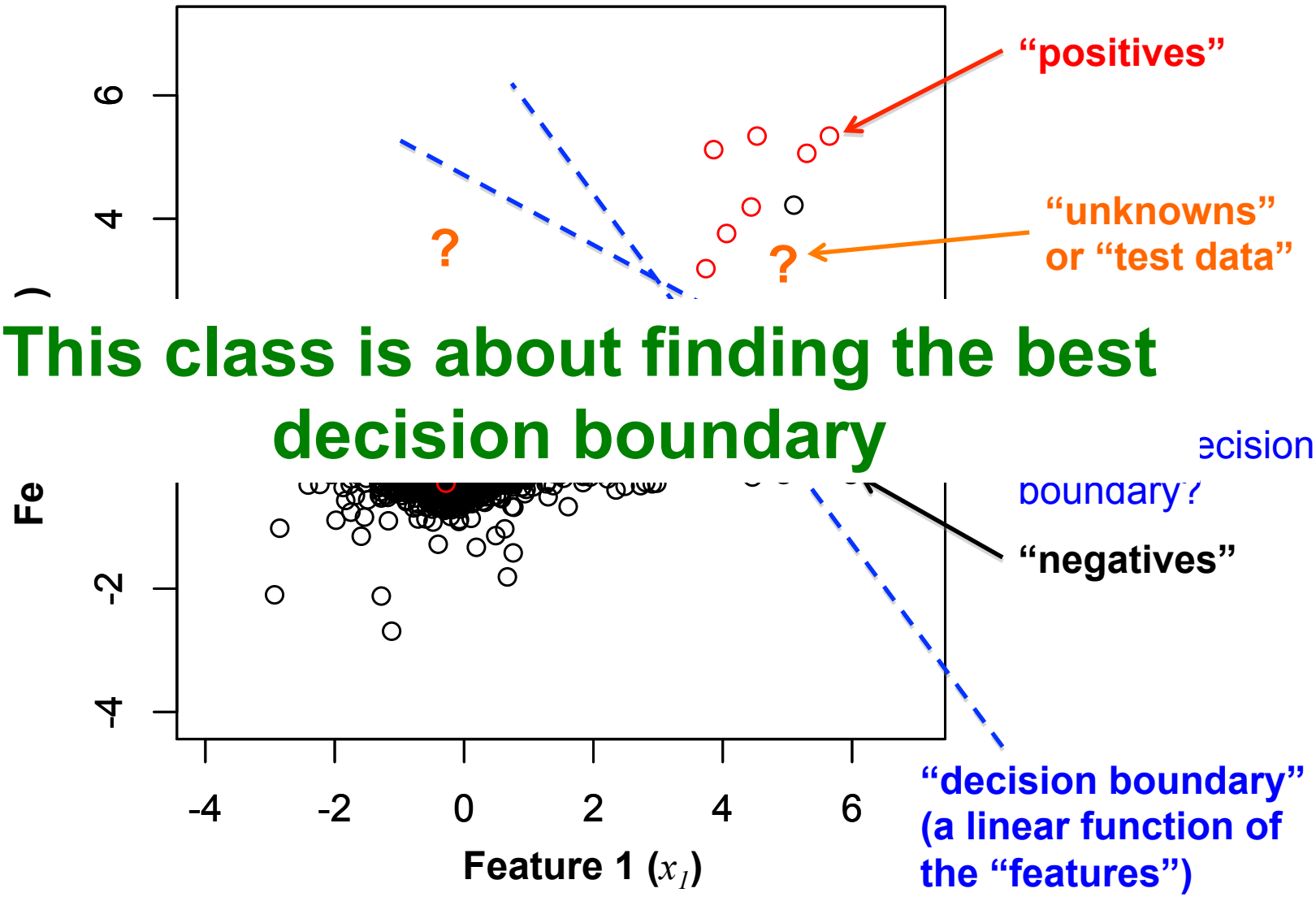
** only a line in 2-d, a plane in 3-d, & hyperplane in >3-d

A ***linear classifier*** will try to find a line** that separates the two classes



** only a line in 2-d, a plane in 3-d, & hyperplane in >3-d

A ***linear classifier*** will try to find a line**
that separates the two classes



This class is about finding the best decision boundary

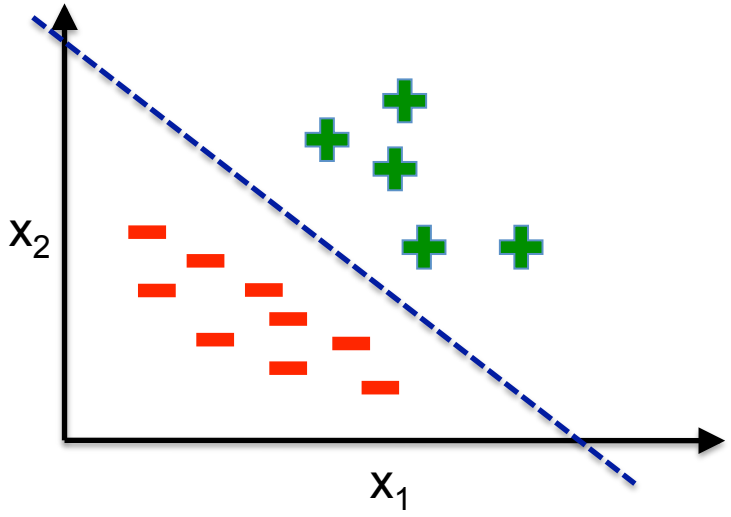
** only a line in 2-d, a plane in 3-d, & hyperplane in >3-d

Overview

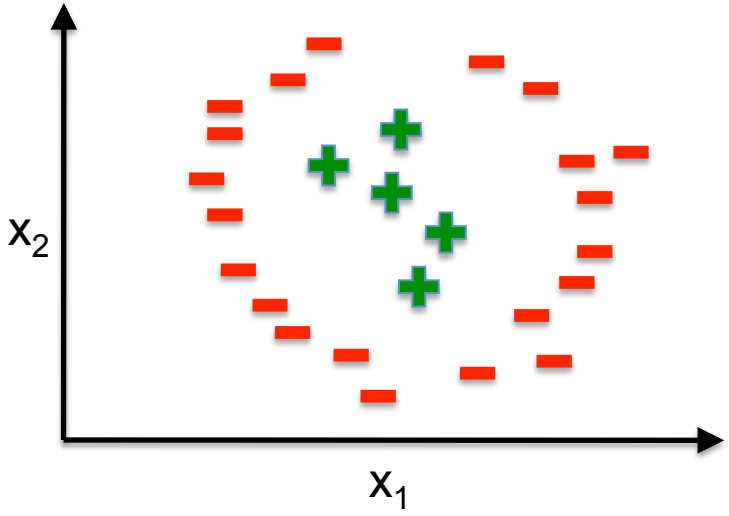
- Linear separability
- Linear classifiers
- Evaluation & cross-validation
- Non-linear classification:
  - K nearest neighbours
  - Support vector machines
  - Decision trees
  - Neural networks (aka deep learning)

Linear separability

Linearly separable



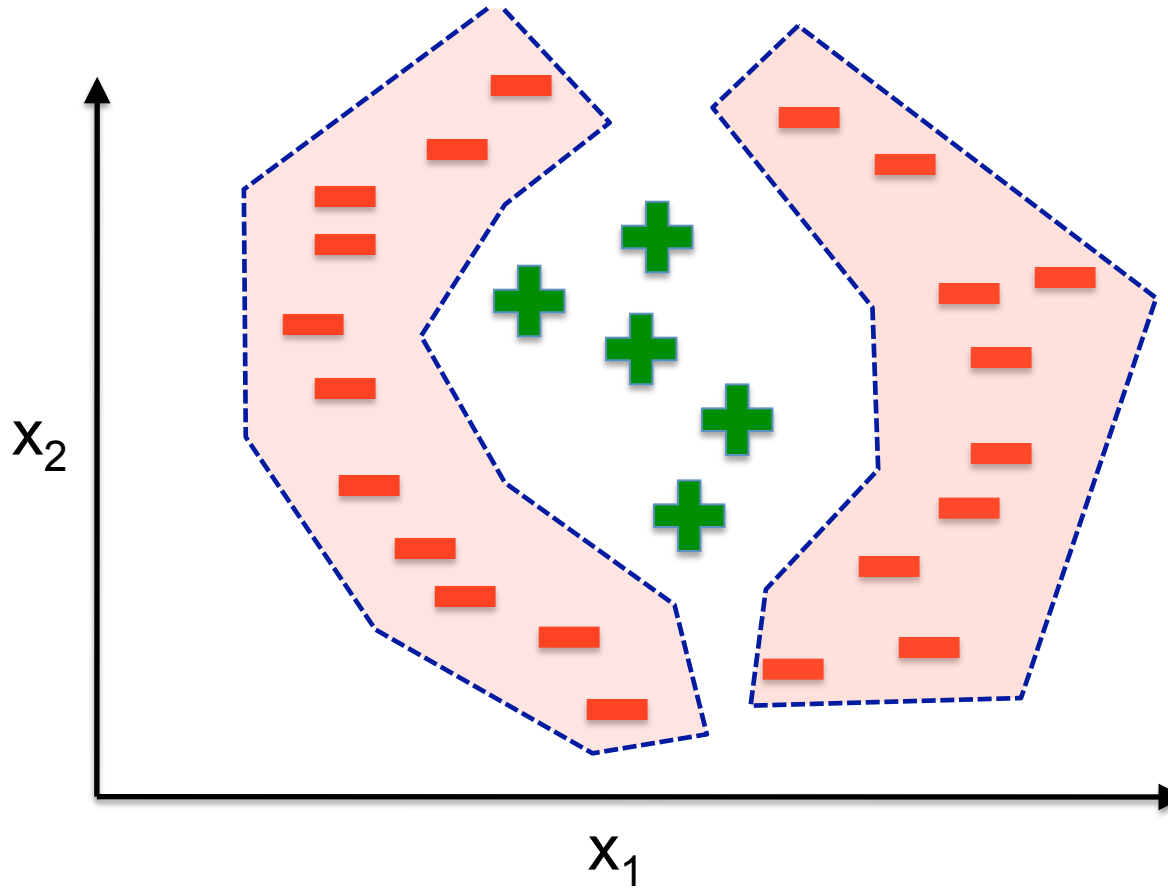
Not linearly separable



Two classes are **linearly separable**, if you can perfectly classify them with a linear decision boundary.

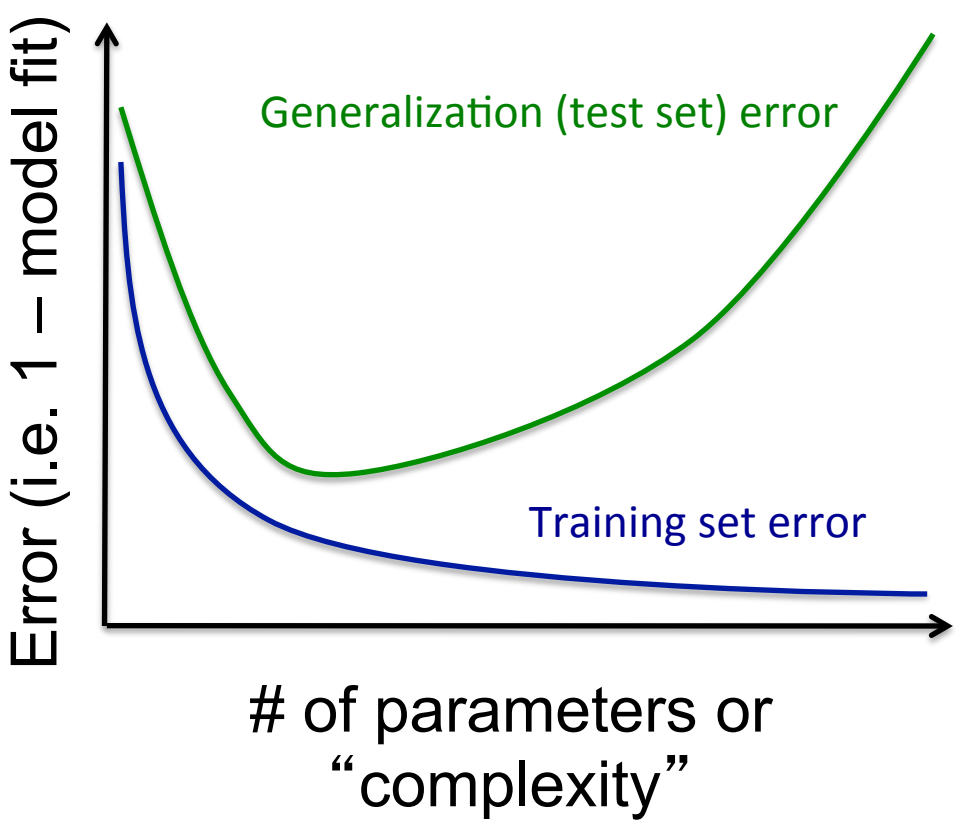
Why is this important? Linear classifiers can only draw linear decision boundaries.

Non-linear classification



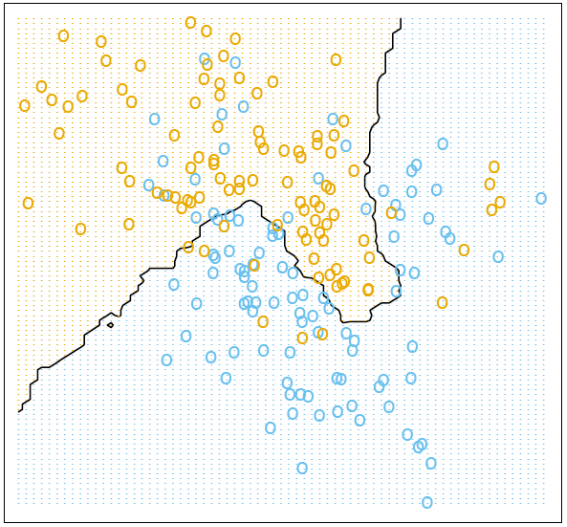
Non-linear classifiers have non-linear, and possibly discontinous decision boundaries

Overfitting

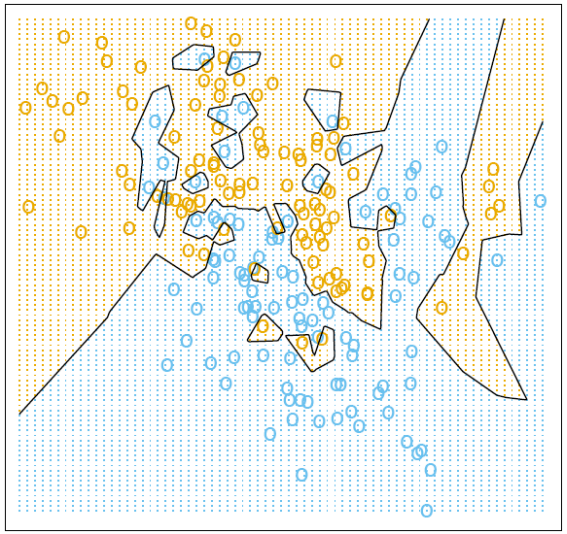


- More complex models better fit the training data.
- But after some point, more complex models *generalize* worse to new data.

Overfitting example



K = 15



K = 1

Classify **x** in the same way that a majority of the *K* nearest neighbours (KNN) to **x** are labeled.

Can also use # of nearest neighbors that are positive as a discriminant value.

Smaller values of K lead to more complex classifiers, you can set K using (nested) cross-validation.

Images from Elements of Stat Learning, Hastie, Tibshirani, Efron (available online)

Dealing with overfitting by regularization

- **Basic idea:** add a term to the objective function that penalizes # of parameters or model complexity, e.g.:

$$E(X, \Theta) = \textit{error}(X, \Theta) + \lambda\ \textit{complexity}(\Theta)$$

- **"Hyper-parameter"** λ controls the strength of regularization – could have a natural value, or be set by (nested) cross-validation,

- Increases in model complexity need to be balanced by improved model fit. (each "unit" of added complexity must reduce error by "λ units") – these units are sometimes called bits.

Examples of regularization

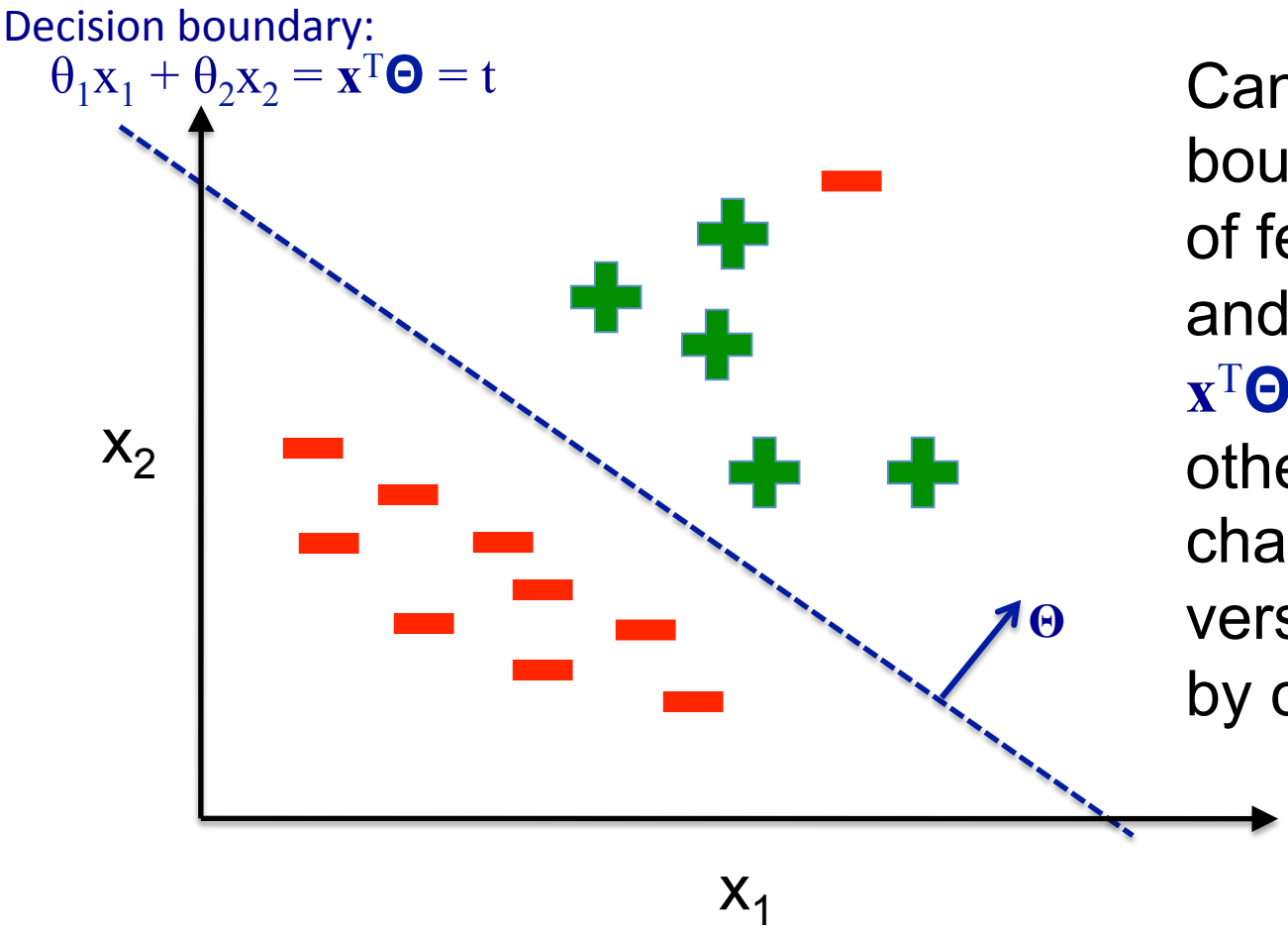
- **Clustering**:
  (K is # params***, N is # datapoints)
  - Bayes Information Criteria:
    - $K\,log(N) - 2\,LL(\mathrm{X};\Theta)$
  - Akaike IC:
    - $2K - 2\,LL(\mathrm{X};\Theta)$
- **Regression & Classification**:
  - L1 (aka LASSO): $LL - \lambda\,\Sigma_i\,|\,\theta_i\,|$
  - L2 (aka Ridge): $LL - \lambda\,\Sigma_i\,\theta_i^2$
  - Elastic net: $LL - \lambda_1\,\Sigma_i\,|\,\theta_i\,| - \lambda_2\,\Sigma_i\,\theta_i^2$

LL stands for "*log likelihood*" it is a measure of how well the model fits the data, higher LL means less error, Alan will discuss this in the last class

*** sometimes it is hard to count # of params

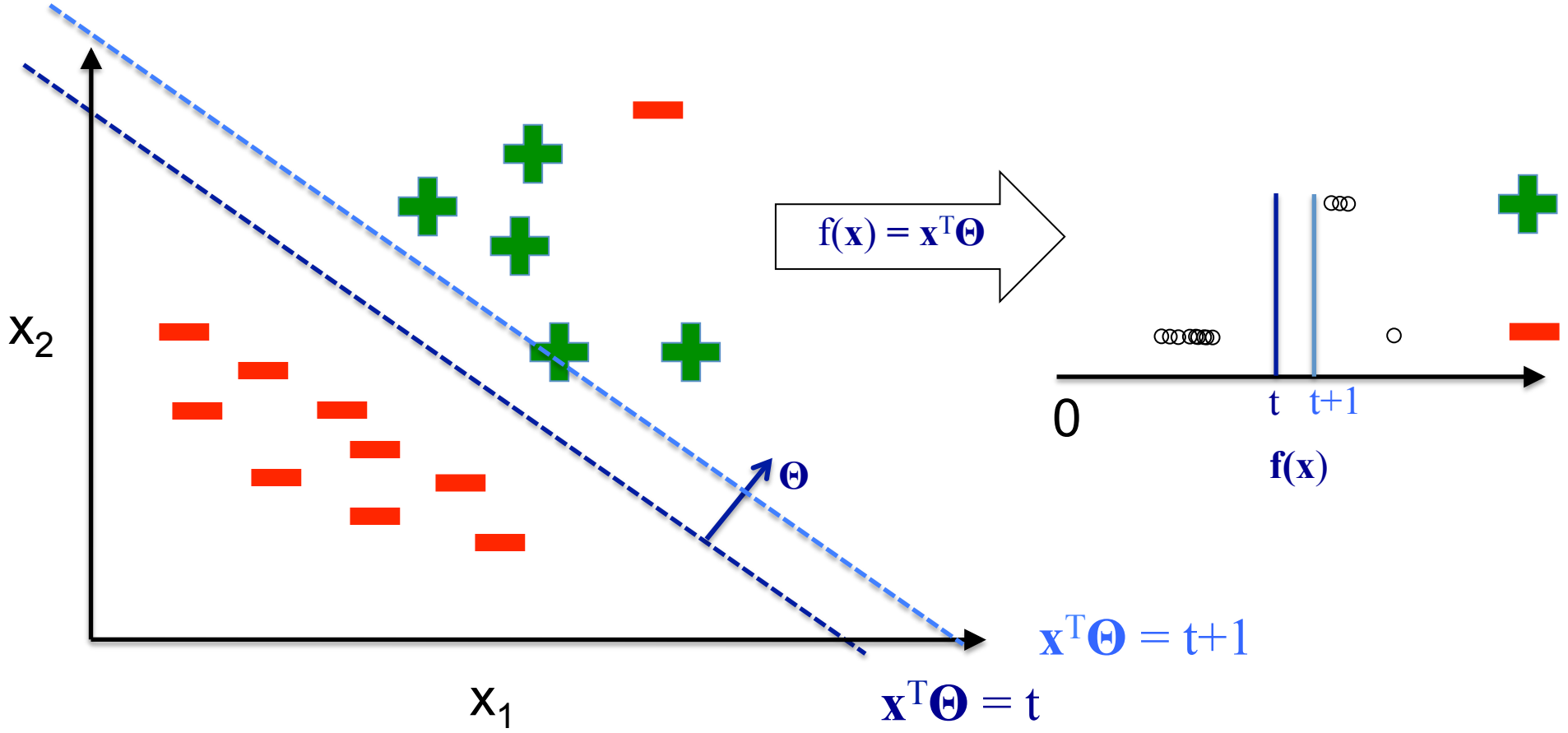
Linear classification

$f(\mathbf{x}) = \mathbf{x}^T\Theta$ **"discriminant function (or value)"**

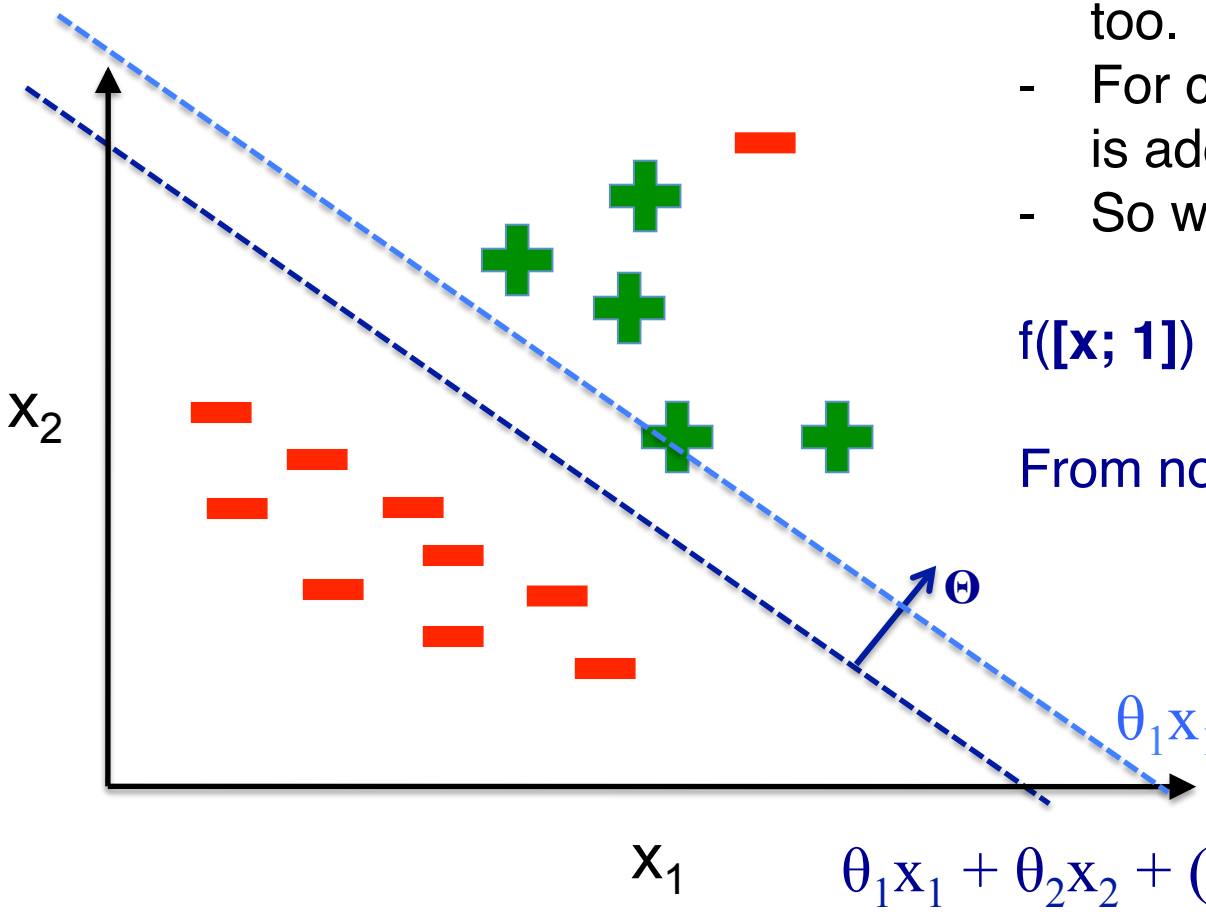


Can define decision boundary with a vector of feature weights, **b**, and a threshold, t, if $\mathbf{x}^T\Theta > t$, predict + and otherwise predict -, can change false positives versus false negatives by changing t

Discriminant function f(x) "projects" data points onto to a line



Practical implementation details



- Note that t is a parameter too.
- For convenience, often a "1" is added to x, to fit t
- So we write:

$f([\mathbf{x}; 1]) = [\mathbf{x}; 1]^T[\Theta; -t] > 0$

From now on **Θ** means **[Θ; -t]**

Linear classification methods

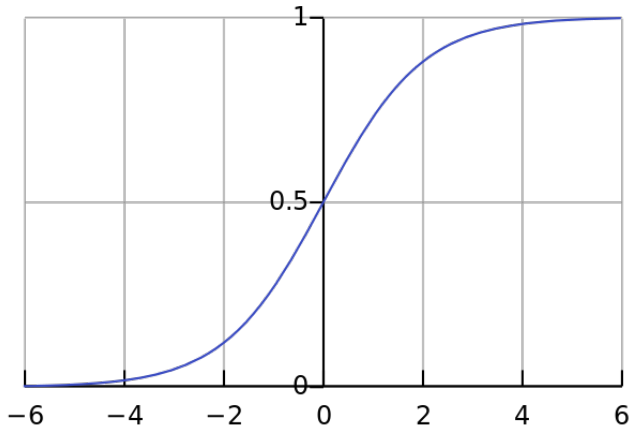
- Logistic Regression
- Linear Discriminant Analysis (LDA)
- Naïve Bayes (sometimes, e.g. with Gaussians)
- Fisher's linear discriminant
- Linear support vector machines (SVMs)

All methods have the same **model**; and use the parameters **Θ** in the same way, but they differ in their **objective function**

Logistic regression

Model:

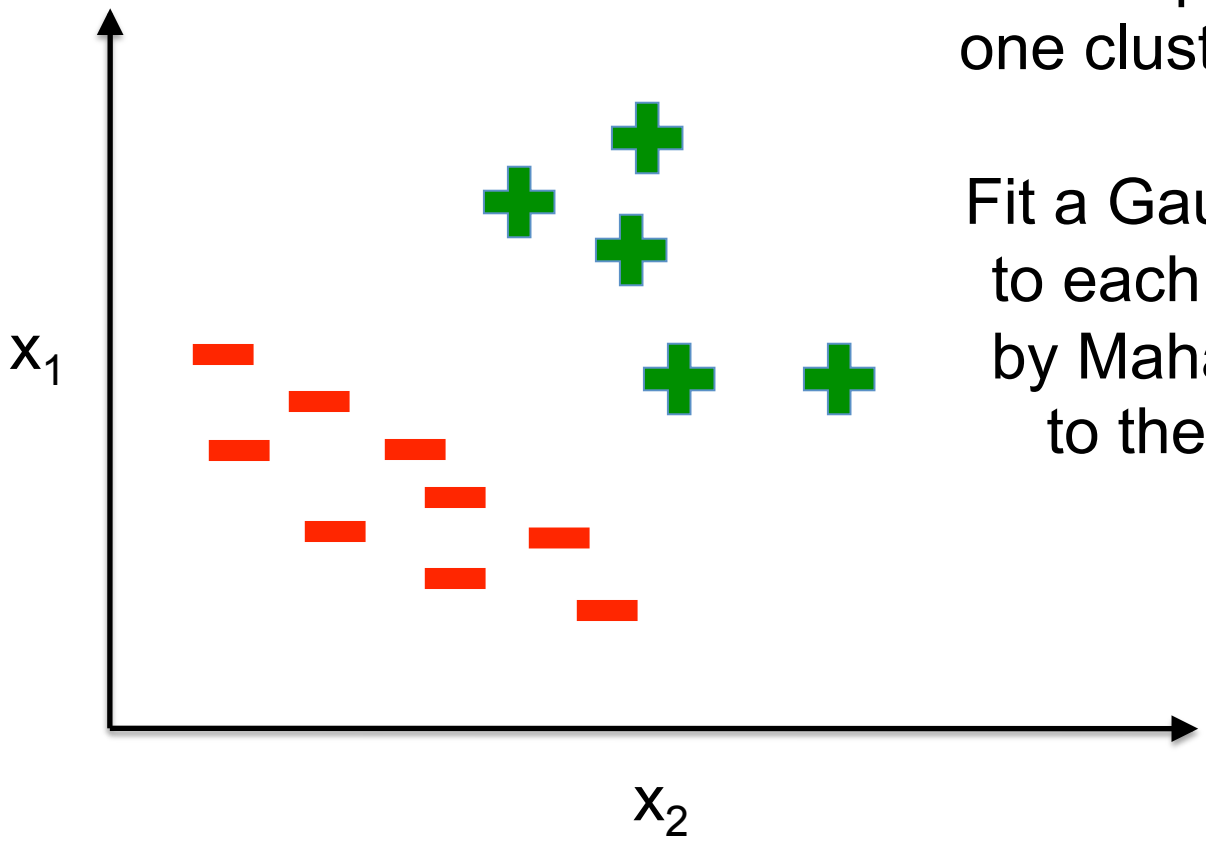
Probability that Y_i is + $= \dfrac{1}{1 + \exp(-\Theta^T \boldsymbol{x}_i)}$



Objective function:

maximize sum of log probabilities of correct classifications

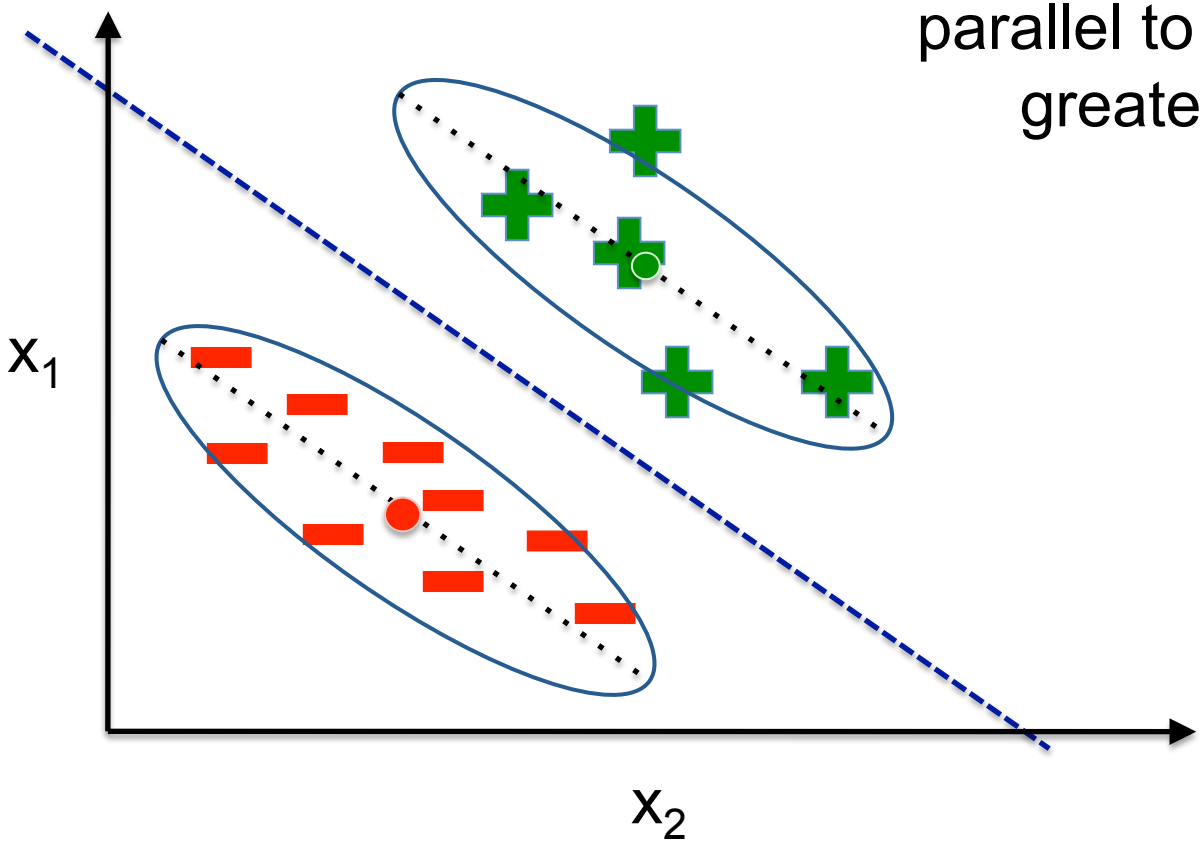
“Classification by clustering” (LDA)



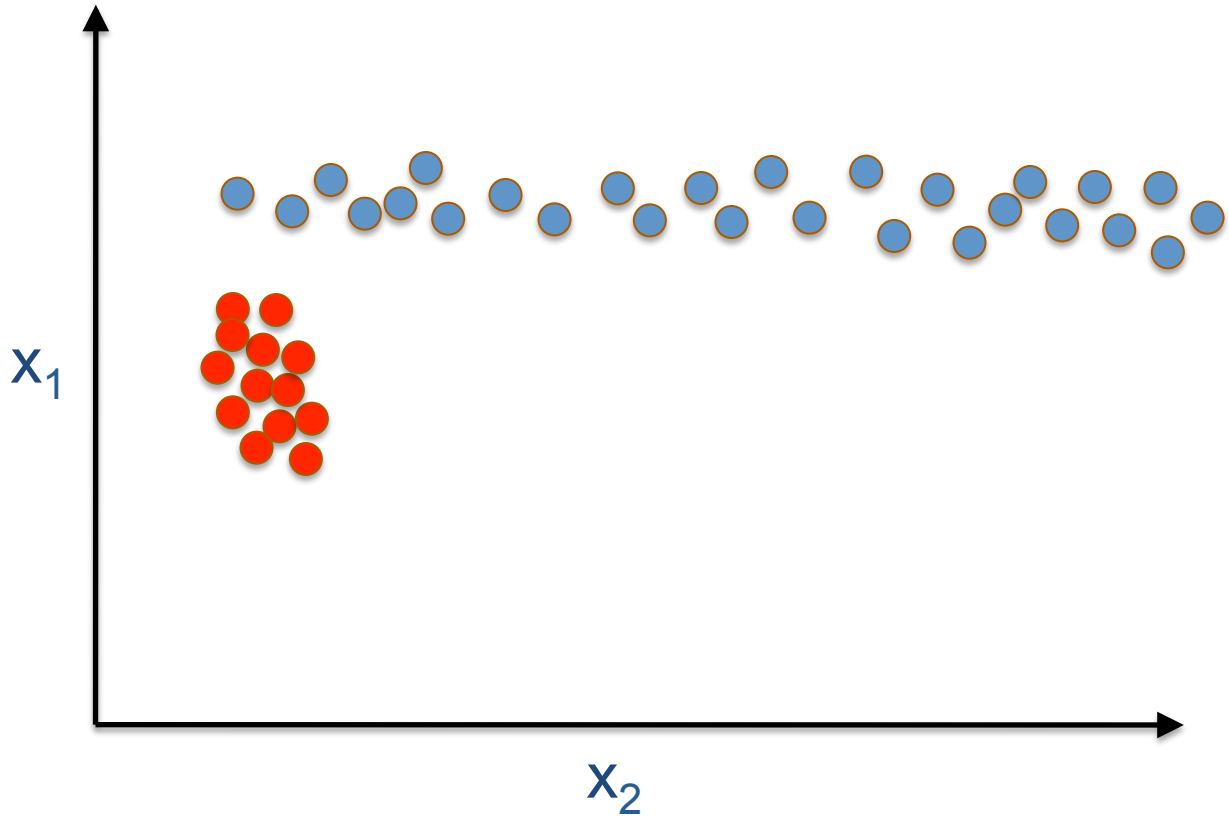
Idea: put all positives in one cluster, all negatives in the other,
Fit a Gaussian (i.e. ellipse) to each and then classify by Mahalanobis distance to the cluster means!

“Classification by clustering” (LDA)

Best decision boundary (in 2-d): parallel to the direction of greatest variance.



How do we deal with this situation?



Fisher's discriminant

- Very similar to LDA, except:
  - Allow each class its own covariance
  - Classify based on

$$(\Sigma_{Y=1} + \Sigma_{Y=0})^{-1} \overrightarrow{(\mu_{Y=1} - \mu_{Y=0})} \bullet \vec{X} > \text{threshold}$$

- Fisher figured out that this maximizes the ratio of "between" to "within" class variance

$$\frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}}$$

How to choose best method?

- Logistic Regression
- LDA
- Naïve Bayes
- Fisher's linear discriminant

Same model, different objective function,
which one is best?

Classification performance

- Need to quantify how well a classifier does
- Always a 'trade-off' between:

-True Positives
-False Positives
-True Negatives
-False Negatives

These 4 numbers are combined in every possible way:

Precision: #TP / (#TP + #FP)
(also known as **positive predictive value**)

Recall: #TP / (#TP + #FN)
(also known as **sensitivity**)
and **true positive rate**

Specificity: #TN / (#FP + #TN)

Negative predictive value: #TN / (#FN + #TN)

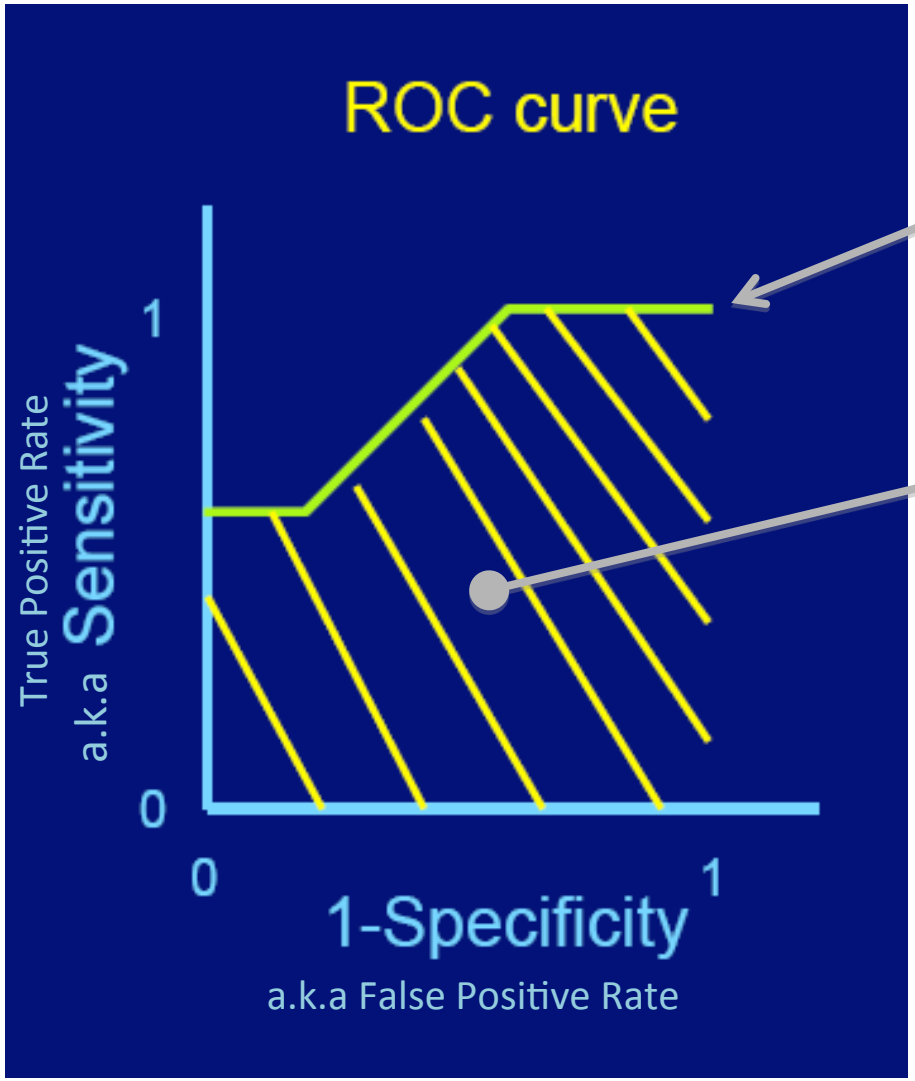
Accuracy: (#TP + #TN) / (#TP + #FP + #TN + #FN)

False positive rate: #FP/(#FP + #TN)

Summarized graphically in '**ROC curves**' or '**Precision-recall plots**'

Summarized numerically using **AUC, AUPRC, MCC, F1** etc…

Quaid Morris 2011



Quaid Morris 2011

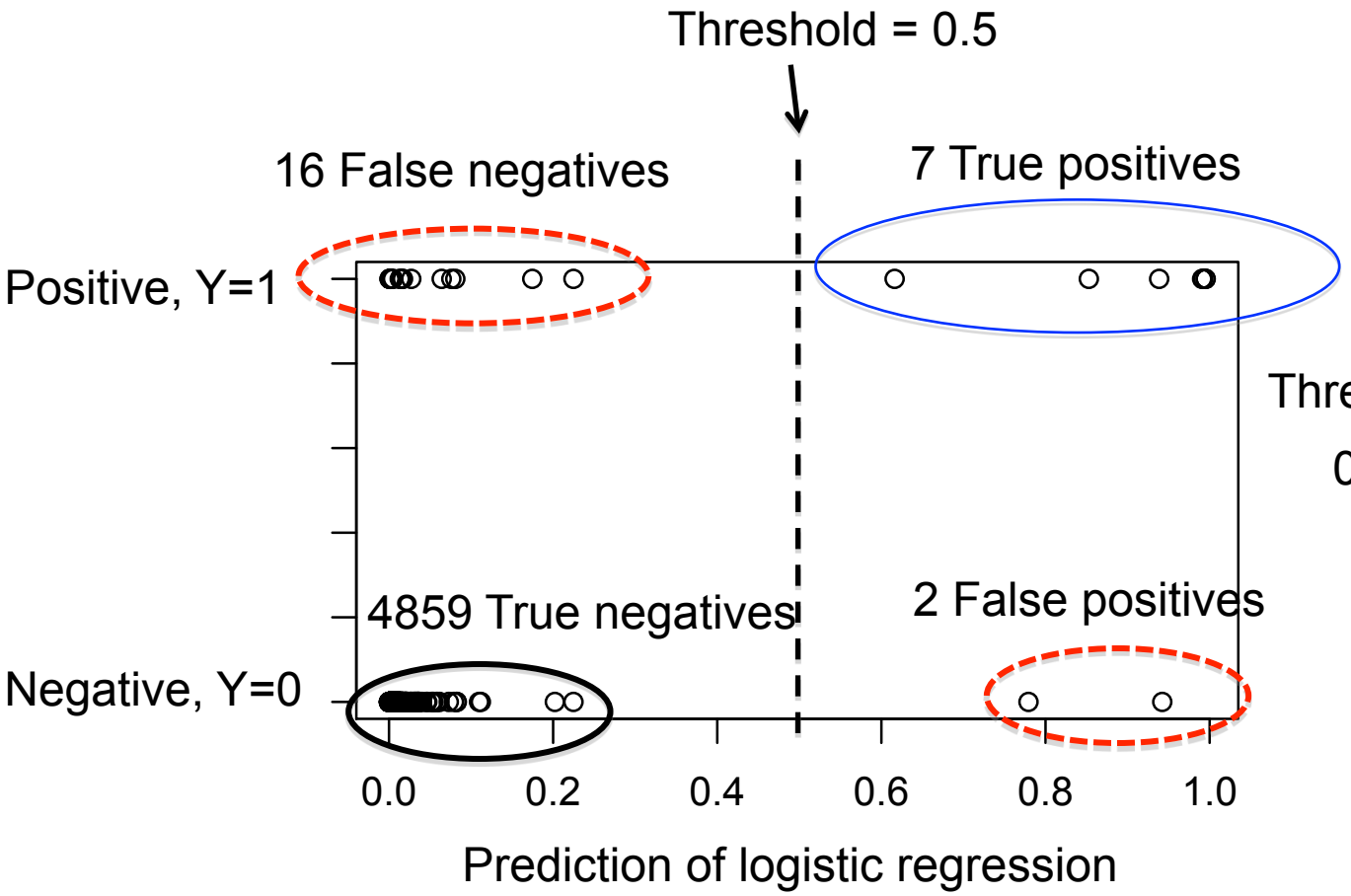
Performance as 'threshold' is varied

Area under the ROC curve (or AUC)

AUC can be interpreted:

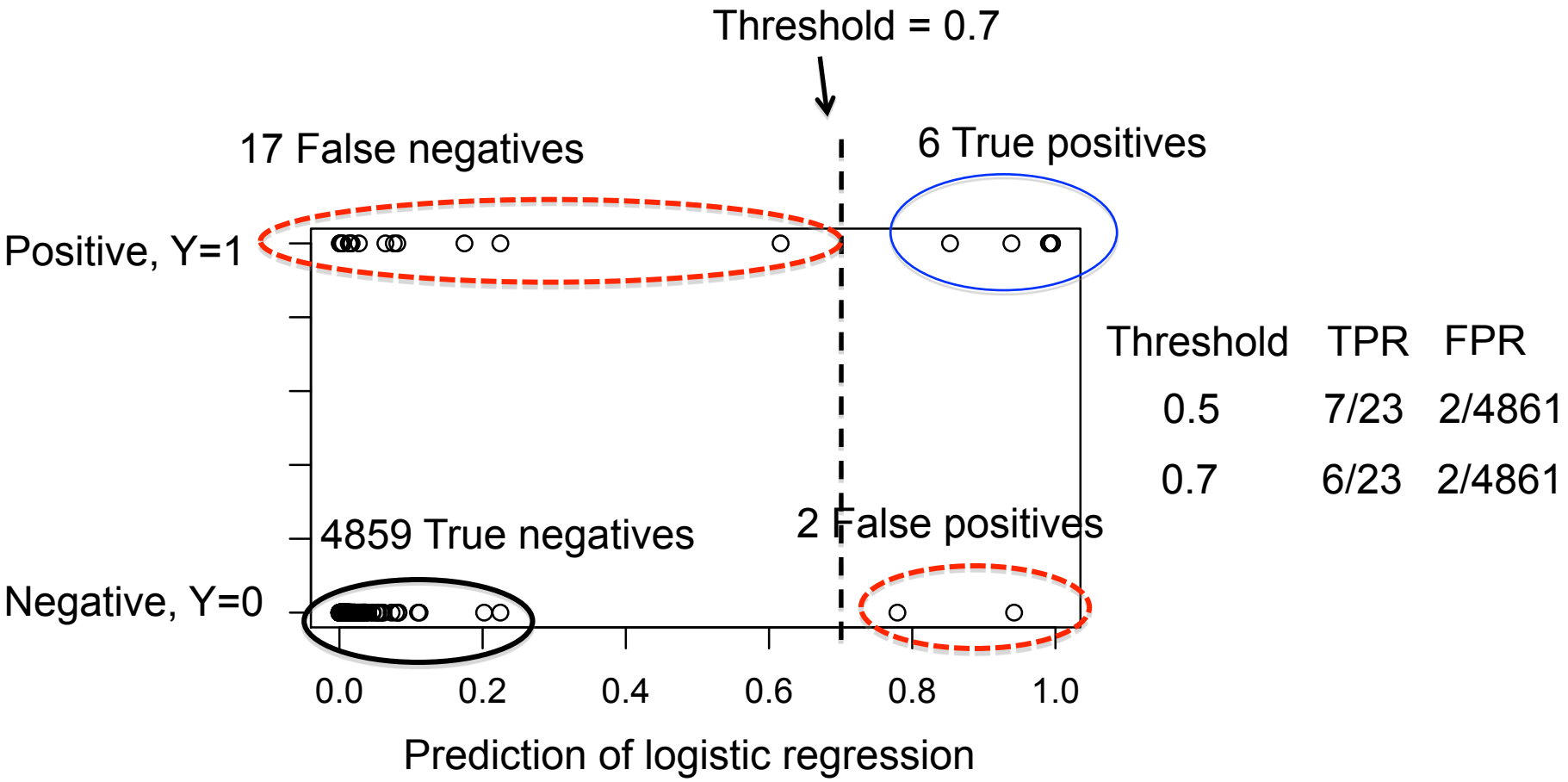
AUC = 0.5 for a random guesser

Distribution of AUC is known, so any difference from 0.5 can be assessed



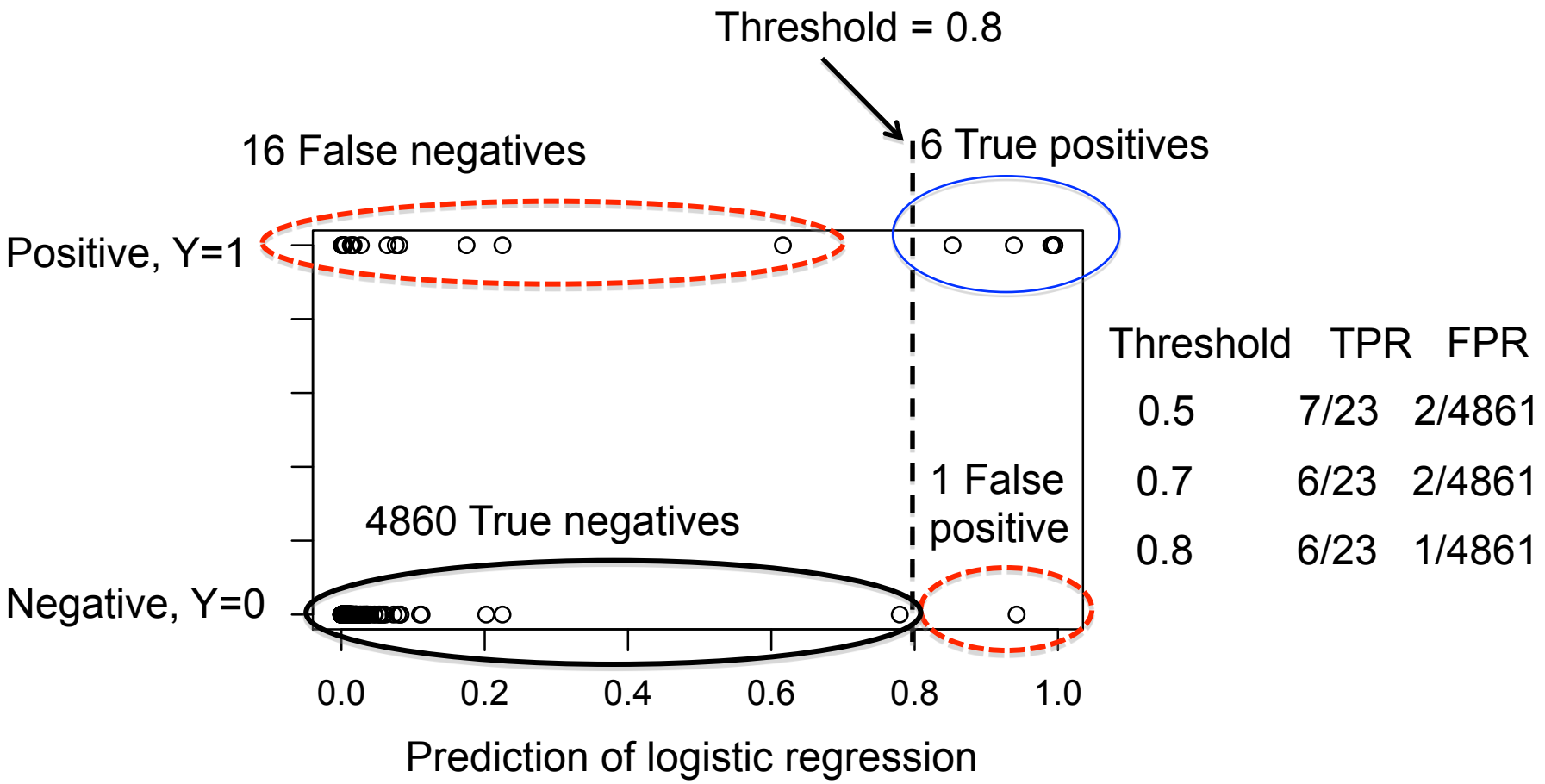
Threshold	TPR	FPR
0.5	7/23	2/4861

FPR = FP/(FP+TN) TPR = TP/(TP+FN)



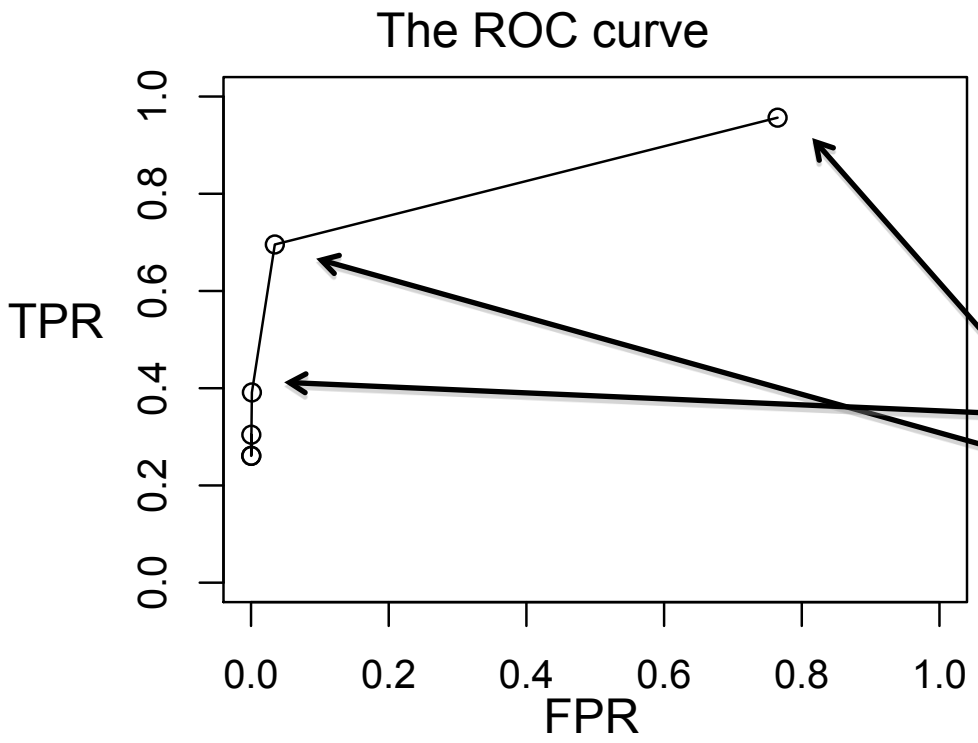
Threshold	TPR	FPR
0.5	7/23	2/4861
0.7	6/23	2/4861

FPR = FP/(FP+TN) TPR = TP/(TP+FN)

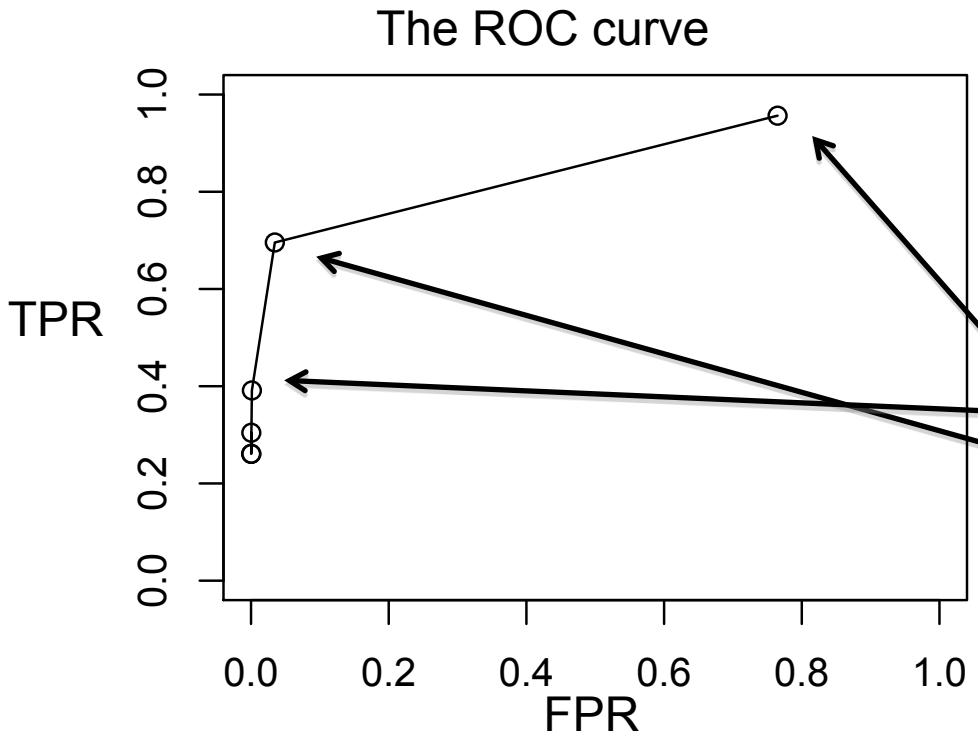


Threshold	TPR	FPR
0.5	7/23	2/4861
0.7	6/23	2/4861
0.8	6/23	1/4861

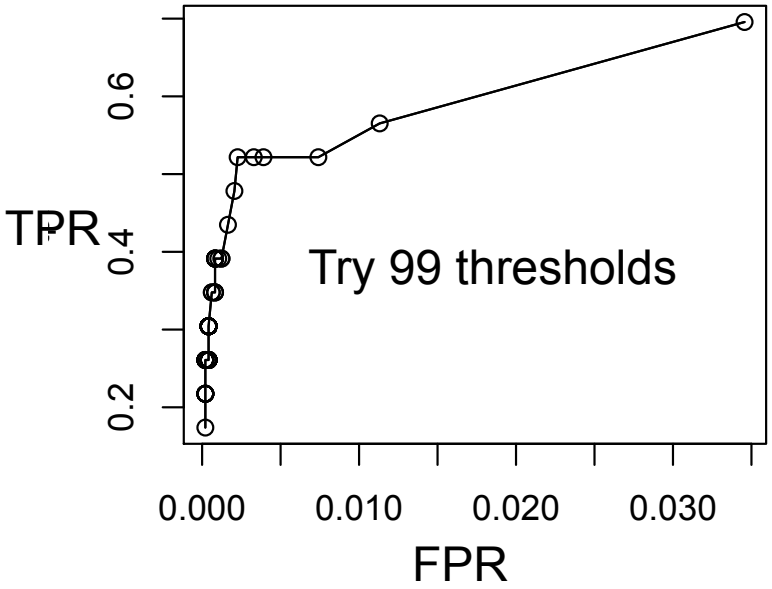
FPR = FP/(FP+TN) TPR = TP/(TP+FN)



Threshold	TPR	FPR
0.5	7/23	2/4861
0.7	6/23	2/4861
0.8	6/23	1/4861
0.1	9/23	6/4861
0.01	16/23	168/4861
0.001	22/23	3718/4861



Threshold	TPR	FPR
0.5	7/23	2/4861
0.7	6/23	2/4861
0.8	6/23	1/4861
0.1	9/23	6/4861
0.01	16/23	168/4861
0.001	22/23	3718/4861



Note that we are not recalculating anything – just changing the threshold

Ideal classification set up

‘Training’:

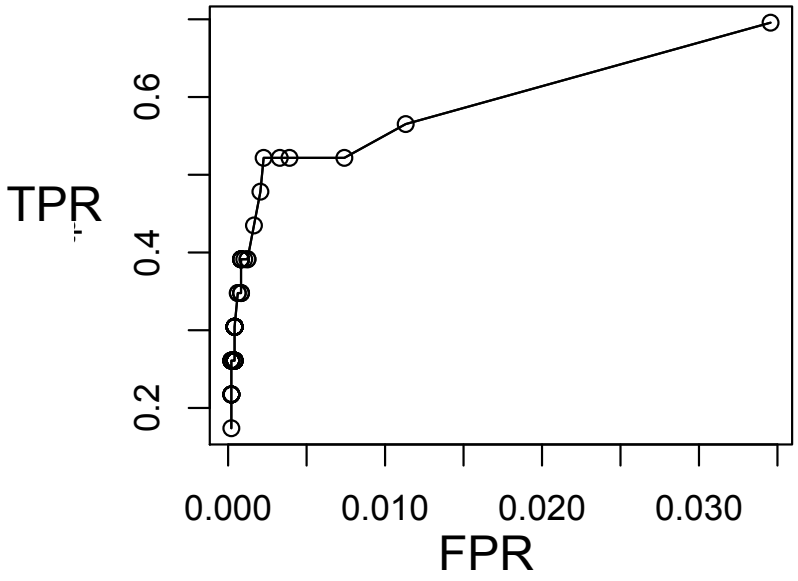
1. Estimate the parameters of the classifier using known examples
2. Determine the best hyper-parameters using a ‘validation set’ of additional known examples

‘Testing’:

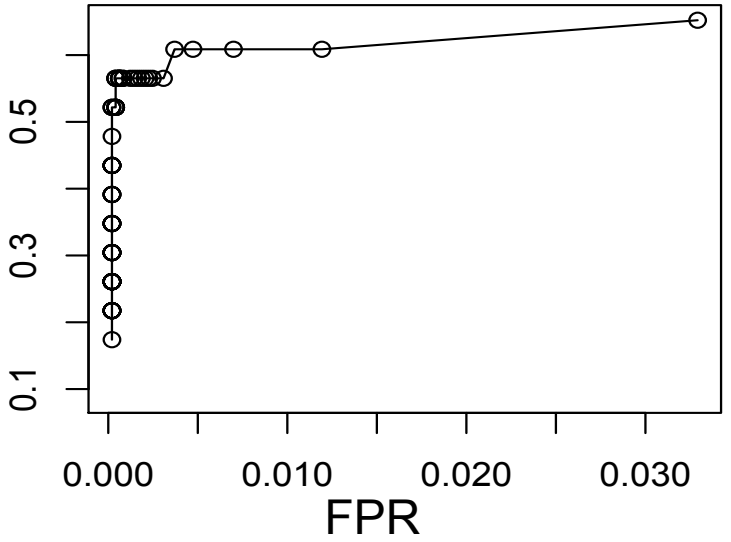
3. See how your classifier does on an unseen dataset of even more known examples

A single ‘threshold’ parameter is left free to control the tradeoff between precision and recall (or TPR and FPR)

PHO81 rep2



PHO81.rep2 + PHO81.rep1 +
PHO4 * pho85 + pho80



- More complex model has better ROC on training data…

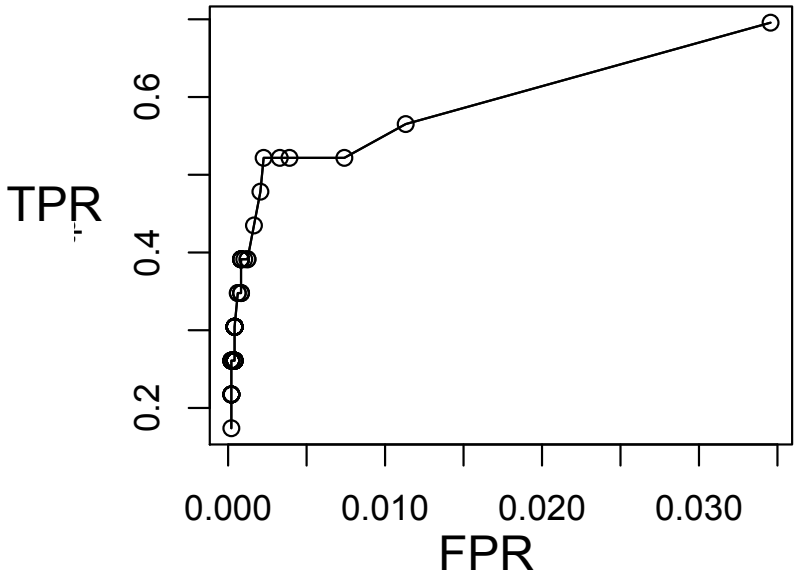
- 23 positive examples is probably not enough to make a ‘test’ set

Typical classification set up

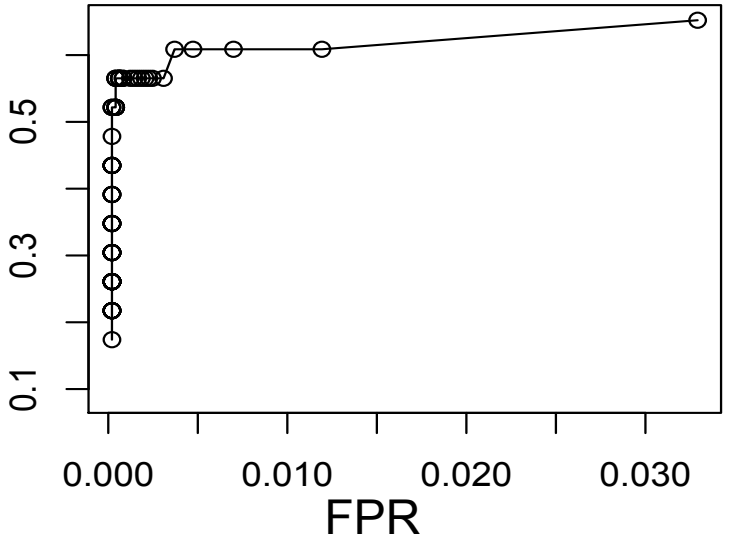
- Don’t have enough data for ‘ideal’ set up.
- Instead, ‘leave out’ random subsets of the data from parameter estimation
- Assess prediction using these as validation sets
- Combine the performance on these ‘held out’ sets
- Typically, “10-fold” or “leave one out”

“cross-validation”

PHO81 rep2



PHO81.rep2 + PHO81.rep1 +
PHO4 * pho85 + pho80



- More complex model has better ROC on training data…
- 23 positive examples is probably not enough to make a 'test' set
- Use 10-fold cross-validation:

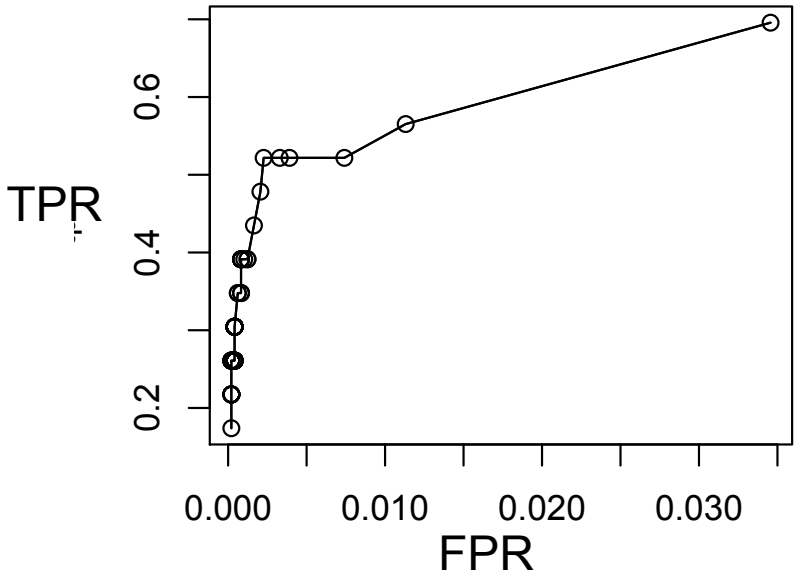
```
10*cv.glm(phodata,mod1,cost,K=10)$delta

10*cv.glm(phodata,mod2,cost,K=10)$delta
```

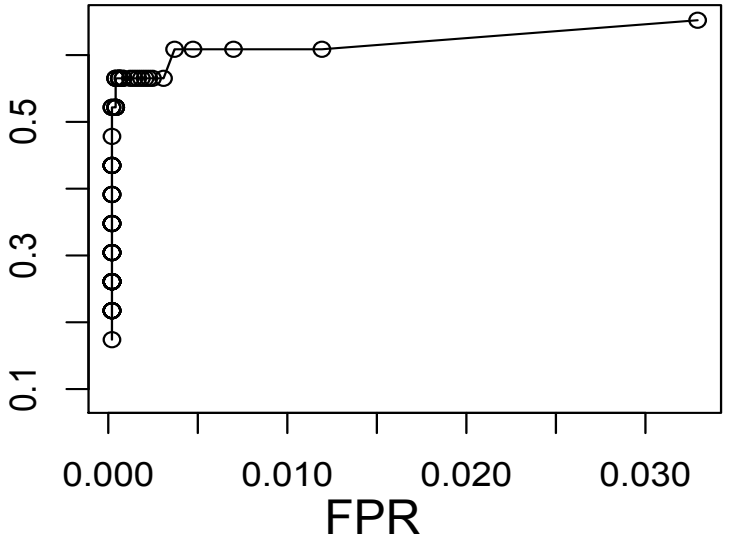
Note that we are fitting each model 10 times

- In R, `glm` has cross-validation functions in the '`boot`' package
- Cross-validation can be used to evaluate any performance measure, '`cost`'

PHO81 rep2



PHO81.rep2 + PHO81.rep1 +
PHO4 * pho85 + pho80



- More complex model has better ROC on training data…
- 23 positive examples is probably not enough to make a 'test' set
- Use 10-fold cross-validation:

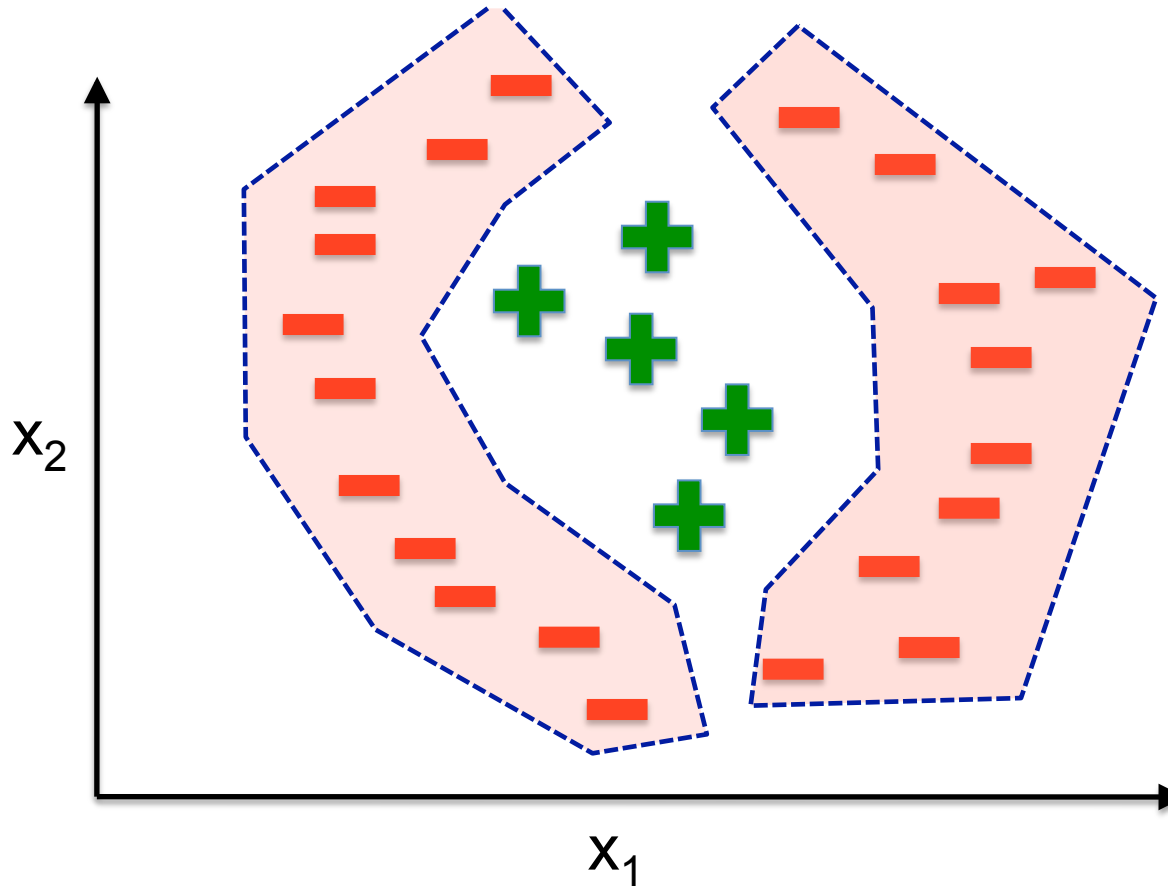
Threshold	TPR	FPR
0.5	7/23	2/4861
0.1	9/23	5/4861
0.05	12/23	16/4861

Threshold	TPR	FPR
0.5	8/23	1/4861
0.1	12/23	10/4861
0.05	12/23	21/4861

Note that we are fitting each model 10 times

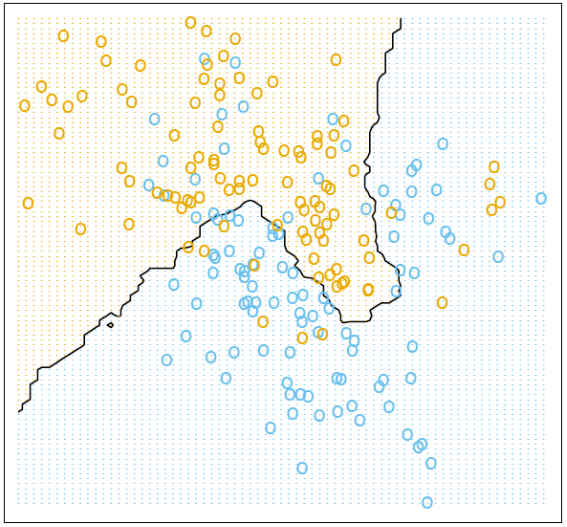
- Remember, these are the numbers based on 10 random sub-samples. Repeating the cross-validation can give different results…

Non-linear classification



Non-linear classifiers have non-linear, and possibly discontinous decision boundaries

K-nearest neighbours

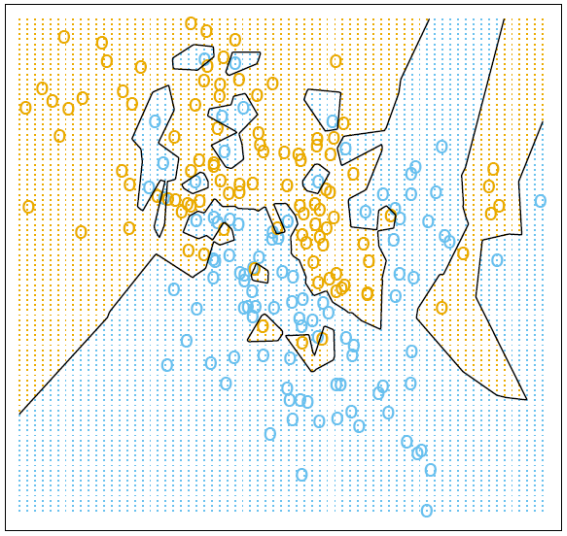


K = 15

Model:

$$f(\boldsymbol{x}) = 1/K \sum_{i \mid \underline{\boldsymbol{x}_i} \text{ is one of K closest points to } \boldsymbol{x}} y_i$$

$y_i = 1$ if datapoint i is a positive, -1 otherwise



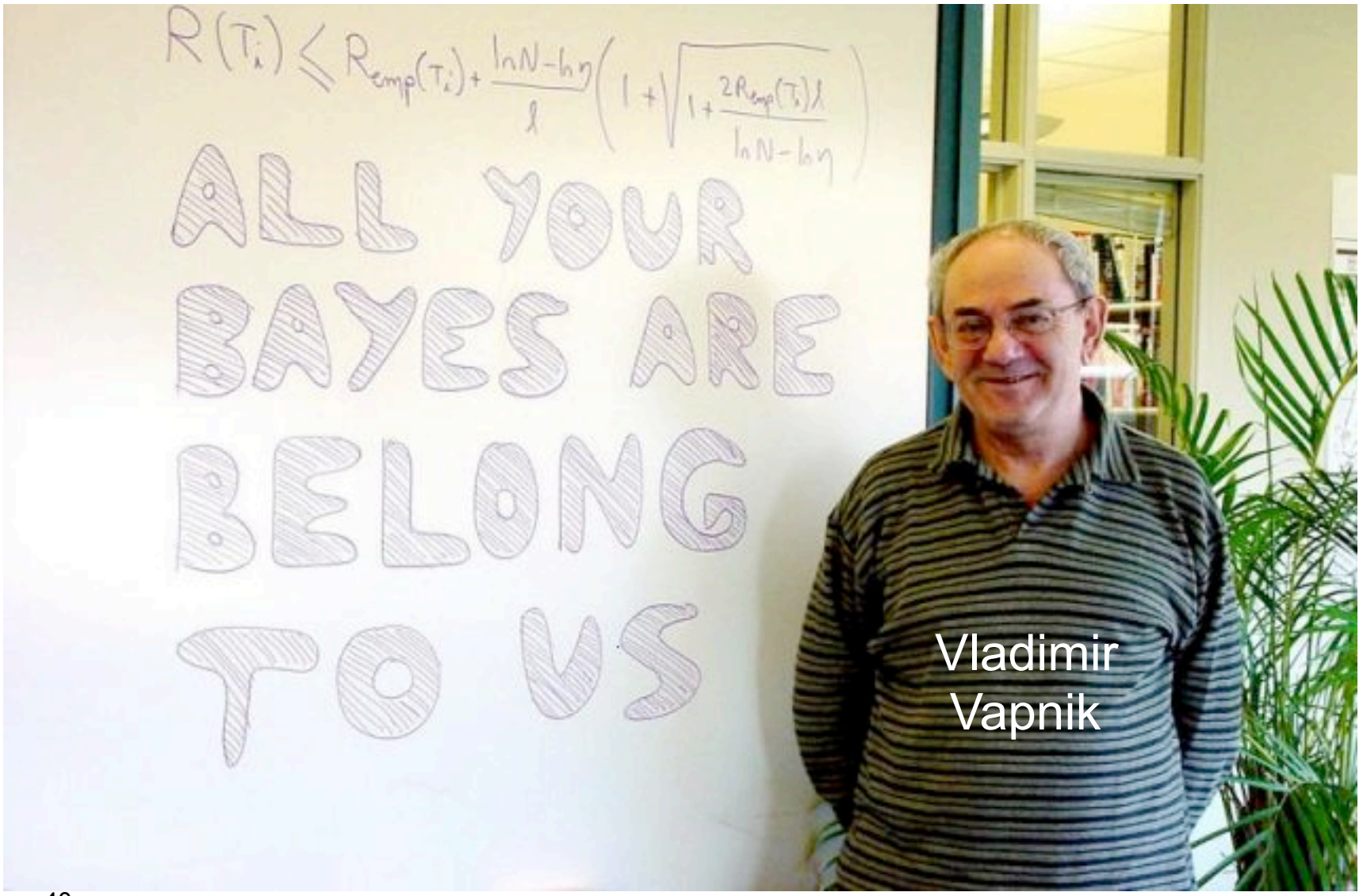
K = 1

Images from Elements of Stat Learning, Hastie, Tibshirani, Efron (available online)

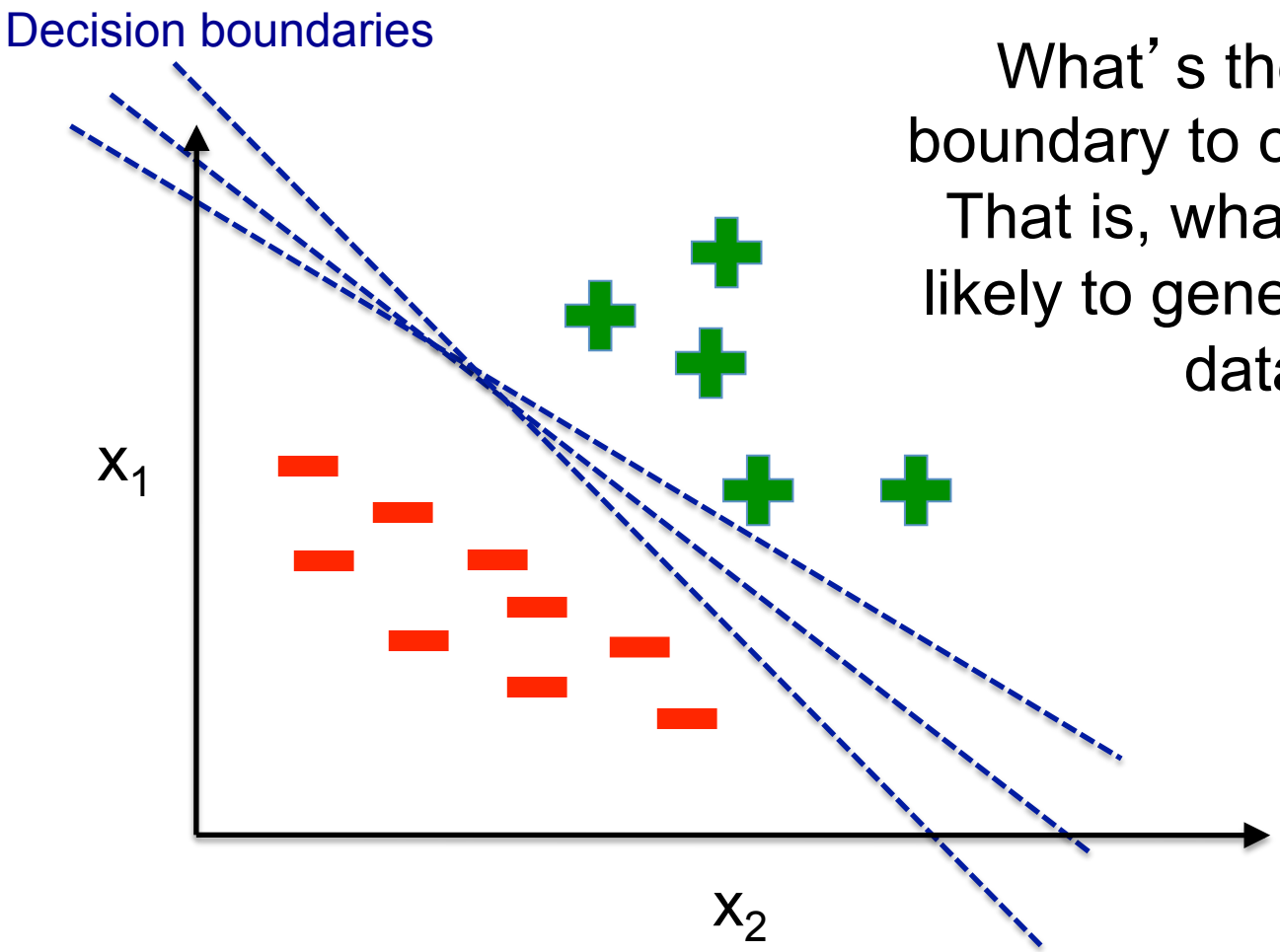
Non-linear classification: Idea #1

- Fit a linear classifier to non-linear functions of the input features.
- E.g.: use “similarity to datapoint i” as the i-th input feature
- Problem: model has one parameter per training example, so it becomes too complex and prone to overfitting

Support Vector Machines



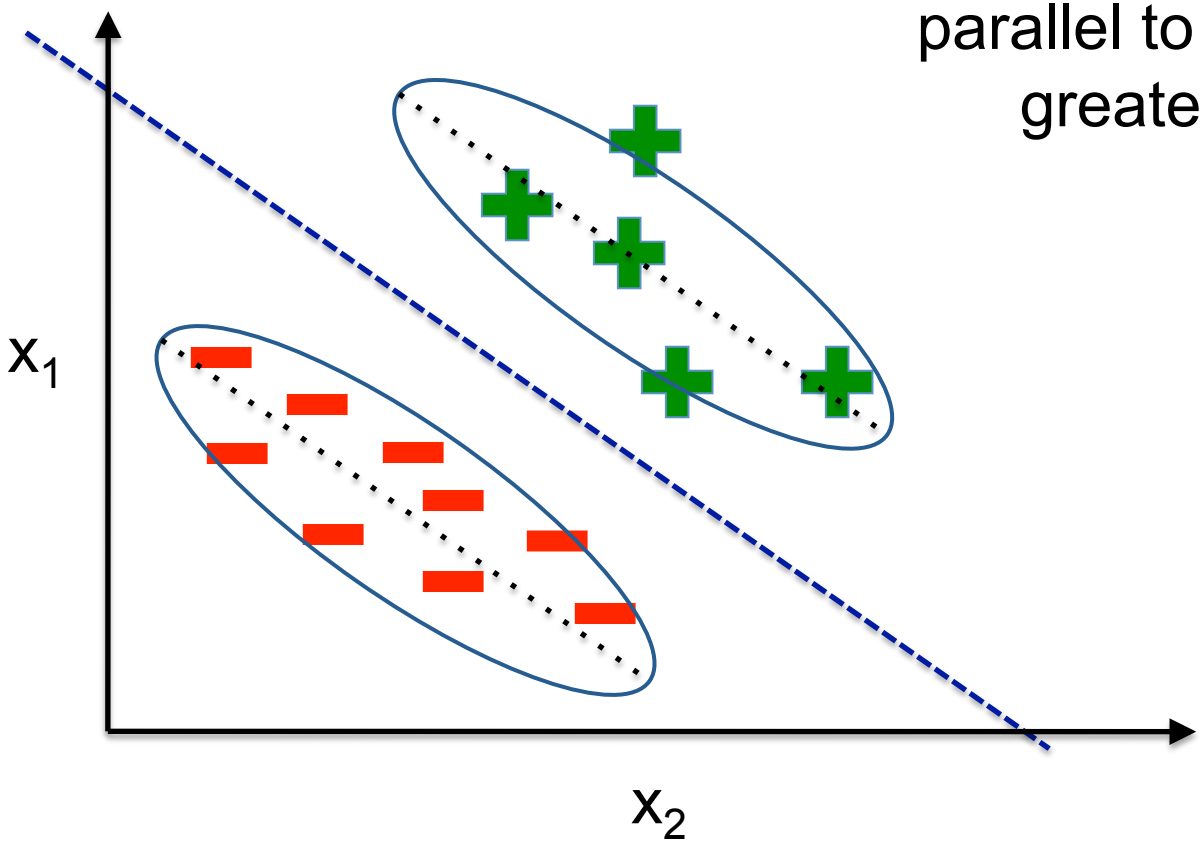
Picking the best decision boundary



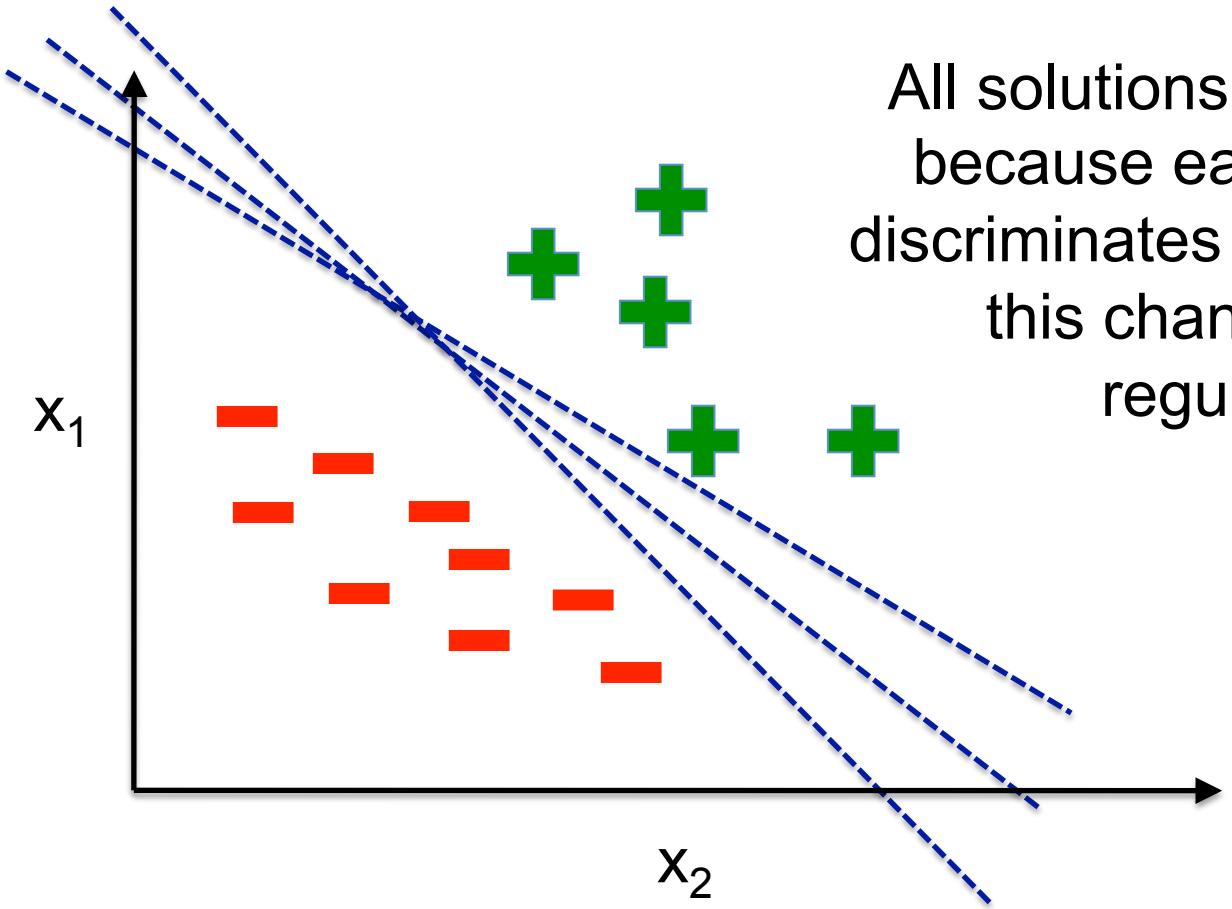
What's the best decision boundary to choose when one? That is, what is the one most likely to generalize well to new datapoints?

LDA & Fisher sol'n

Best decision boundary (in 2-d): parallel to the direction of greatest variance.

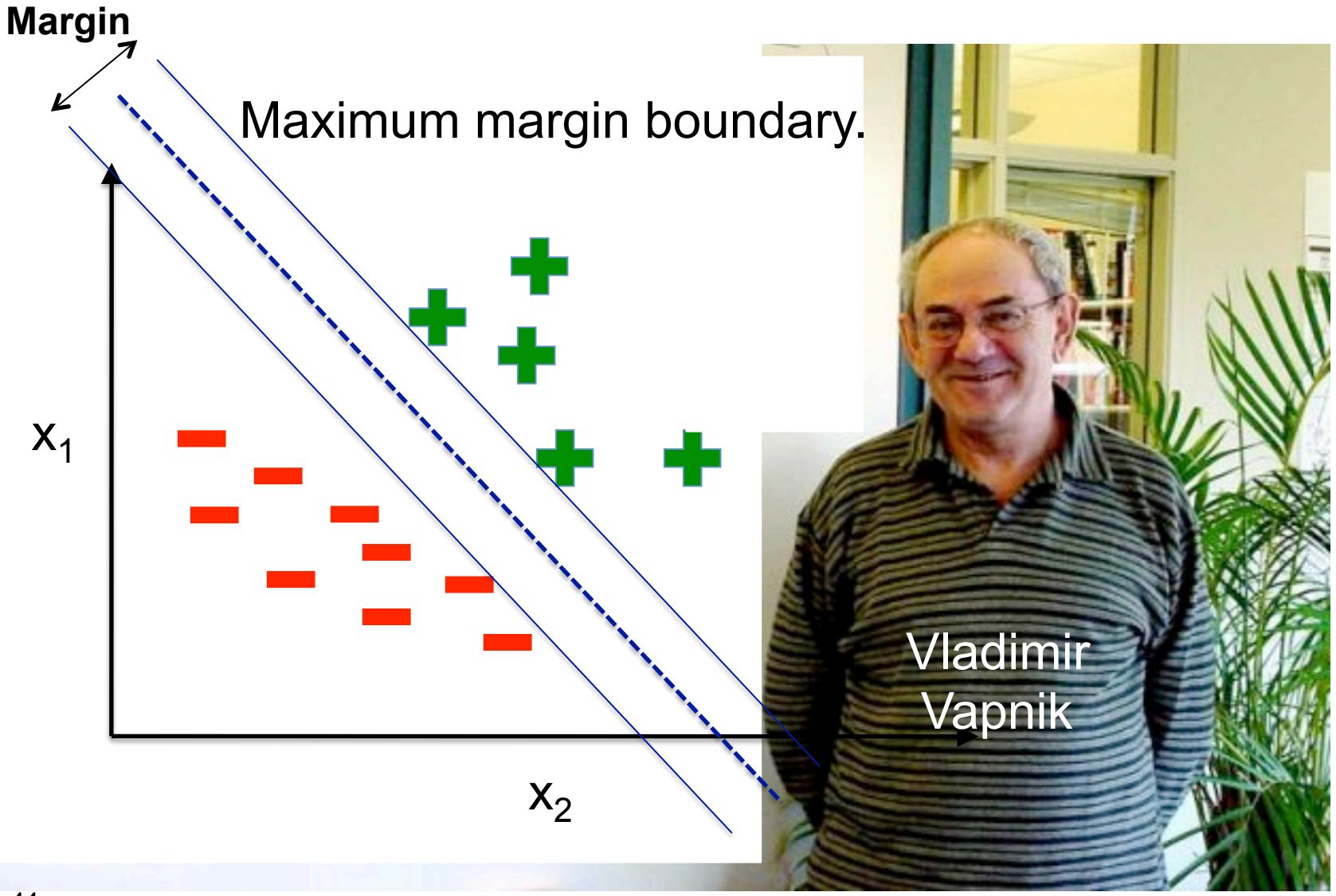


Unregularized logistic regression

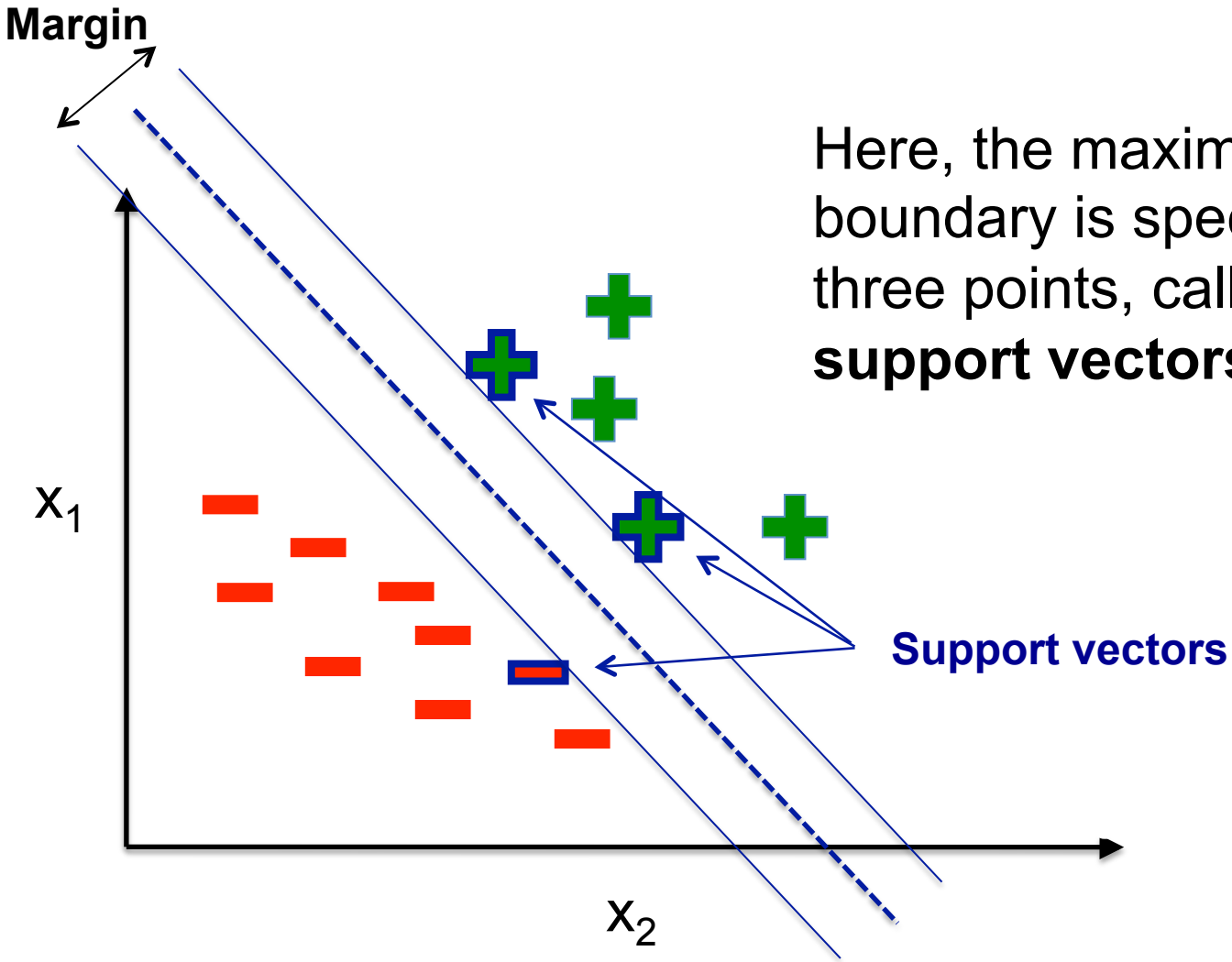


All solutions are equally good because each one perfectly discriminates the classes. (note: this changes if there's regularization)

Linear SVM solution



SVM decision boundaries



Here, the maximum margin boundary is specified by three points, called the **support vectors**.

SVM objective function

- "Primal" objective (cost) function:

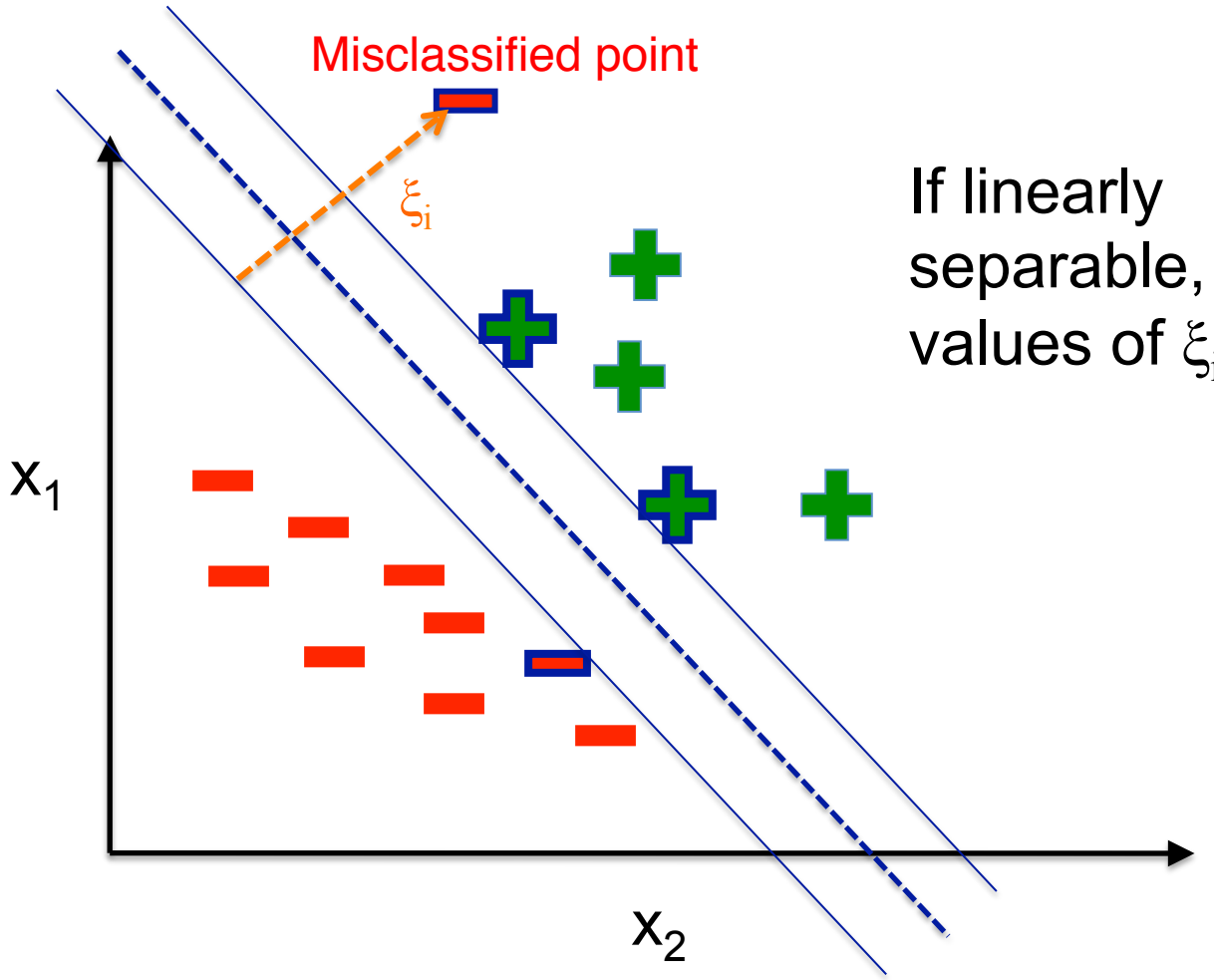
$$E(\Theta) = C \Sigma_i \xi_i + \Theta^T\Theta/2$$

data fit L_2 reg

where $y_i (\Theta^T x_i) > 1-\xi_i$ for all i

  - $\xi_i >= 0$ is degree of "mis-classification" for datapoint i, $C>0$ is the hyperparameter balancing regularization and data fit.

Visualization of ξ_i



If linearly separable, all values of $\xi_i = 0$

SVM summary

A "sparse" linear classifier (i.e. $\theta_i = 0$ for almost all i) that uses as features "kernel functions" that measure similarity to each data point in the training set.

Discriminant function: $f(\mathbf{x}) = \Sigma_i \theta_i K(\mathbf{x_i}, \mathbf{x})$

*Sometimes written with $\theta_i = \alpha_i y_i$

Dot products of transformed vector as "kernel functions"

- Let $\mathbf{x} = (x_1, x_2)$ and $\phi^2(x) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)$
- Here $\phi_2(\mathbf{x})$ [or $\phi_p(\mathbf{x})$] is a vector valued function that returns all possible powers of two [or p] of x_1 and x_2
- Then $\phi_2(\mathbf{x})^T\phi_2(\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2 = K(\mathbf{x},\mathbf{z})$ "kernel function"
- And in general, $\phi_p(\mathbf{x})^T\phi_p(\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^p$
- Every kernel function $K(\mathbf{x},\mathbf{z})$ that satisfies some simple conditions corresponds to a dot product of some transformation $\phi(\mathbf{x})$ (aka "projection function")

Kernelization (FYI)

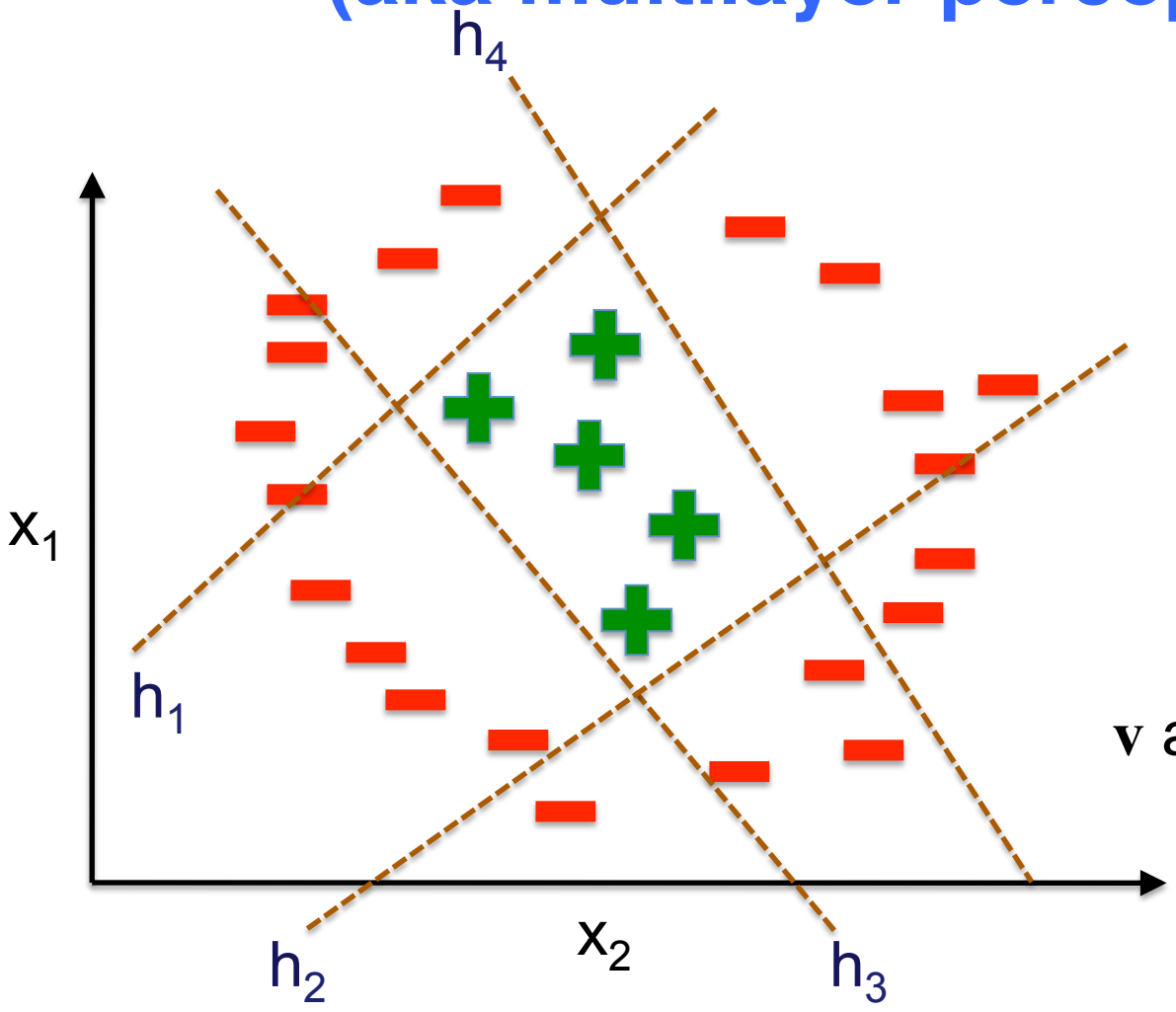
- Often instead of explicitly writing out the non-linear feature set, one simply calculates a kernel function of the two input vectors, i.e., $K(\mathbf{x}^i, \mathbf{x}^j) = \phi(\mathbf{x}^i)^T \phi(\mathbf{x}^j)$
- Here's why: any kernel function $K(\mathbf{x}^i, \mathbf{x}^j)$ that satisfies certain conditions, e.g., $K(\mathbf{x}^i, \mathbf{x}^j)$ is a symmetric positive semi-definite function (which implies that the matrix K, where $K_{ij} = K(\mathbf{x}^i, \mathbf{x}^j)$ is a symmetric positive semi-definite matrix), corresponds to a dot product $K(\mathbf{x}^i, \mathbf{x}^j) = \phi(\mathbf{x}^i)^T \phi(\mathbf{x}^j)$ for some projection function $\phi(\mathbf{x})$.
- Often it's easy to think of defining a kernel function that captures some notation of "similarity"
- Non-linear SVMs use the discriminant function

$$f(\mathbf{x}; \mathbf{w}) = \Sigma_i w_i K(\mathbf{x}^i, \mathbf{x})$$

Some popular kernel functions

- $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1)^P$
  - Inhomogeneous Polynomial kernel of degree P
- $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)^P$
  - Homogeneous Polynomial kernel
- $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\{-(\boldsymbol{x}_i - \boldsymbol{x}_j)^T (\boldsymbol{x}_i - \boldsymbol{x}_j) / s^2\}$
  - Gaussian "radial basis function" kernel
- $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \text{Pearson}(\boldsymbol{x}_i, \boldsymbol{x}_j) + 1$
  - "Bioinformatics" kernel
- *Also, can make a kernel out of an interaction network!*

Neural Networks
(aka multilayer perceptrons)



$f(\boldsymbol{x}, \Theta) = \Sigma_k v_k h_k(\boldsymbol{x})$

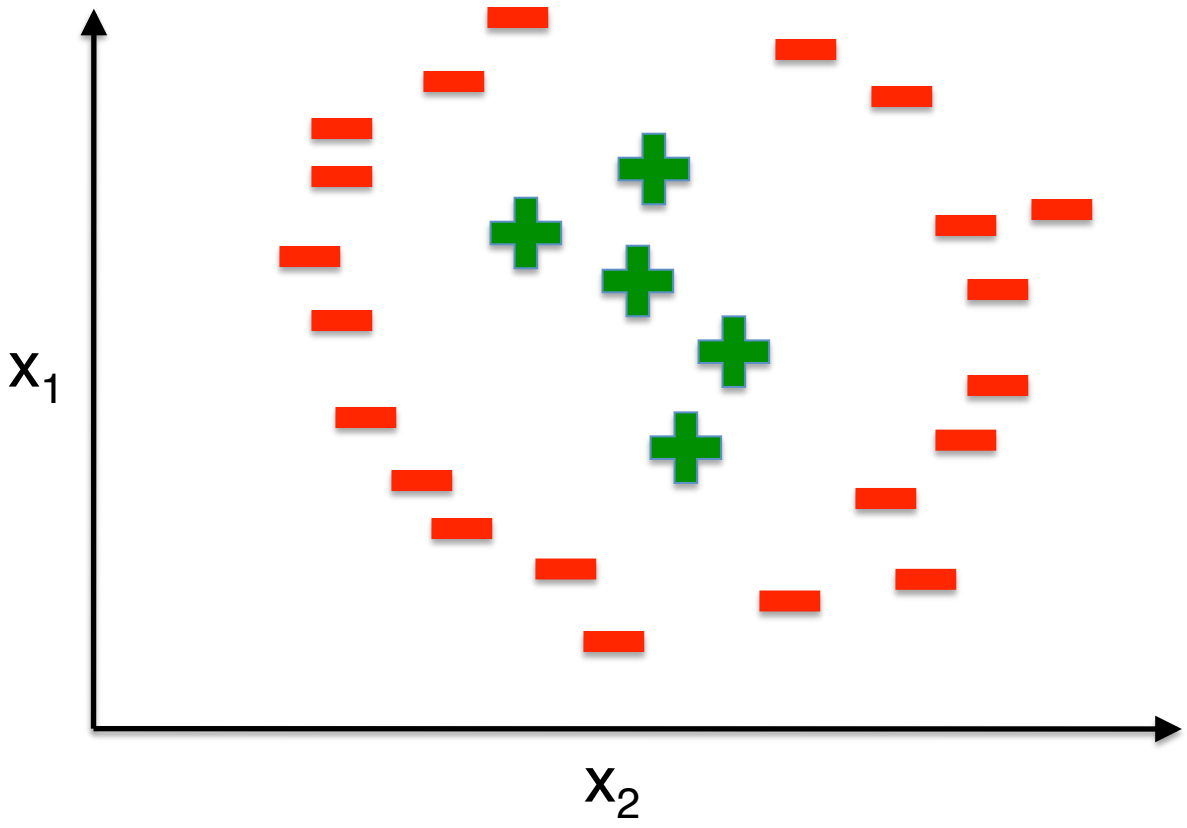
Where "hidden unit":
$h_k(\boldsymbol{x}) = \sigma(\Sigma_i w_{ik} x_i)$

Often:
$\sigma(x) = 1 / (1 + e^{-x})$

v and W are parameters

Notes:
-Fitting the hidden units has become a lot easier using deep belief networks
-Could also fit means of RBF for hidden units

Decision trees

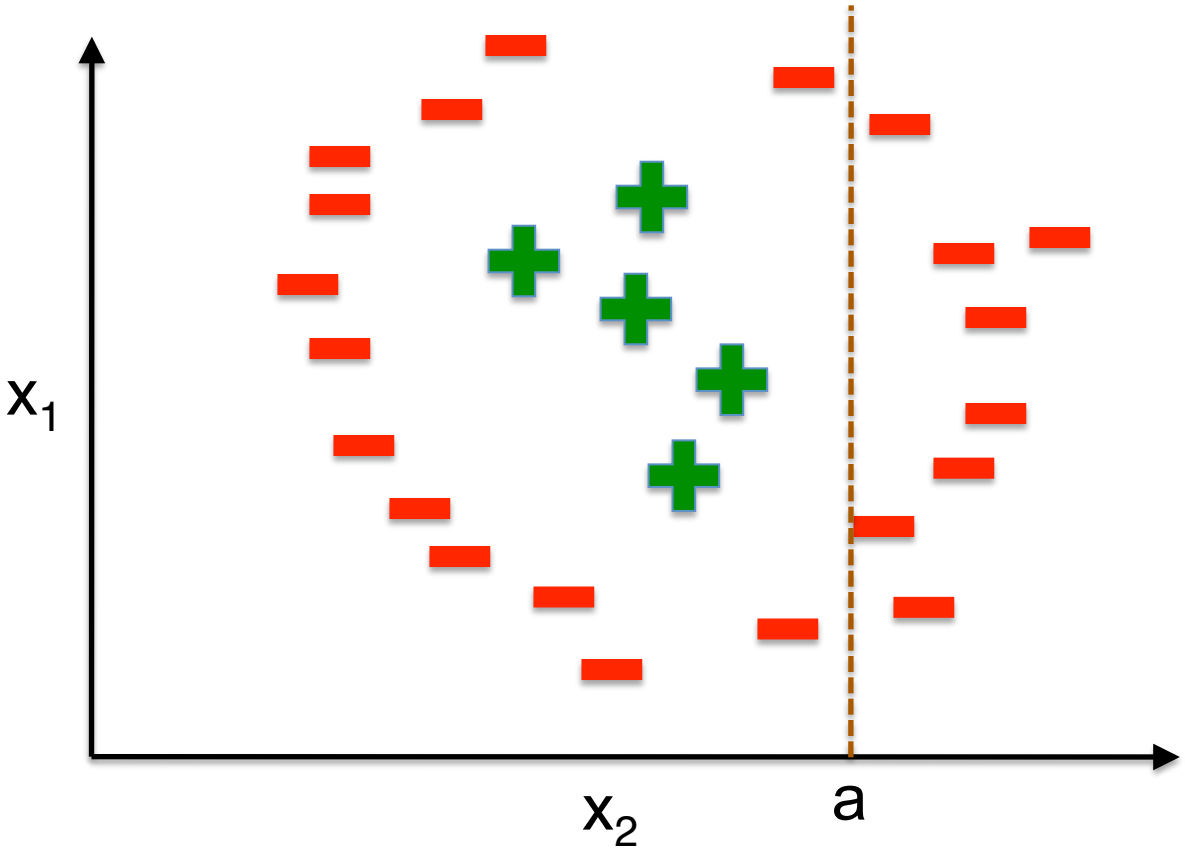


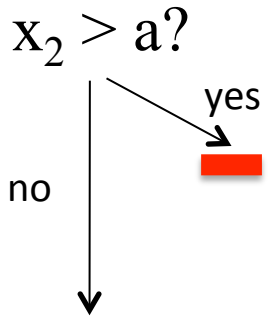
For each leaf:
1) Choose a dimension to split along, and choose best split using a "split criterion" (see below)
2) Split along dimension, creating two new leaves, return to (1) until leaves are pure.

To reduce overfitting, it is advisable to prune the tree by removing leaves with small numbers of examples

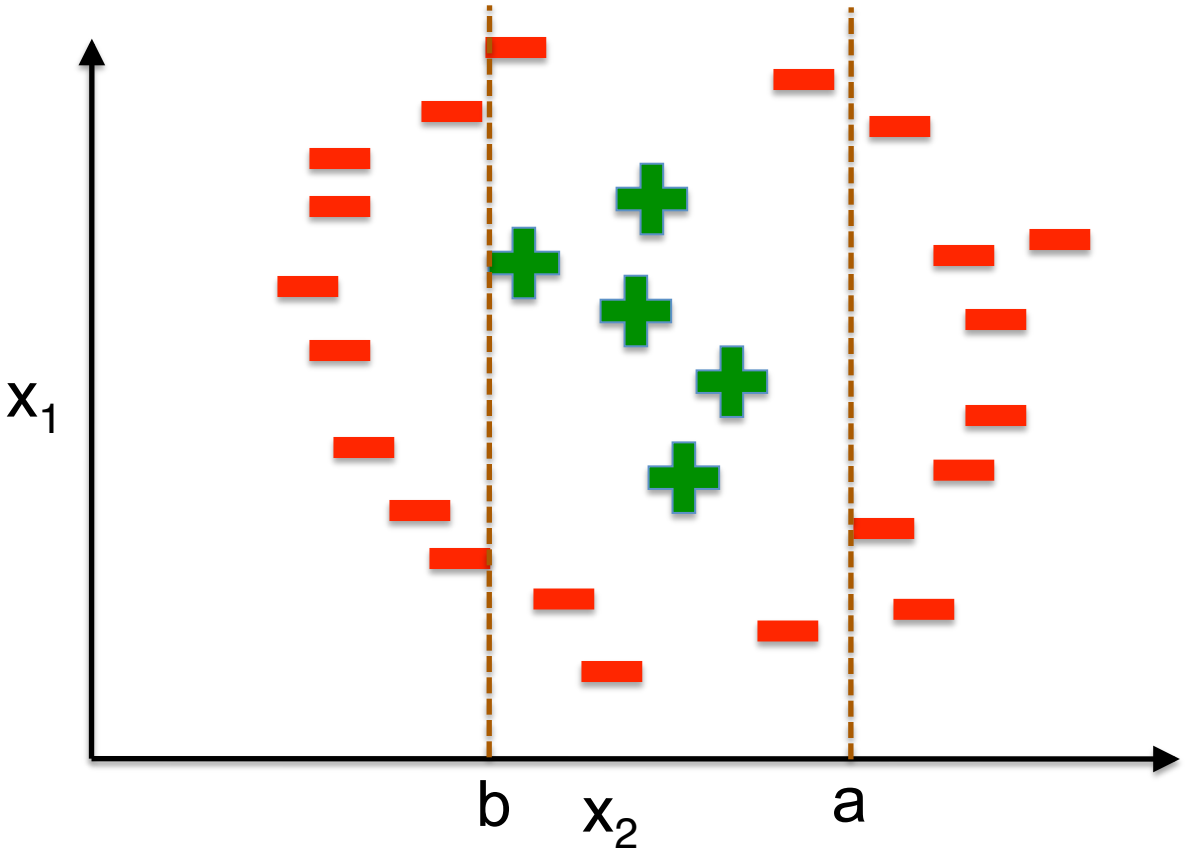
Split criterion:
C4.5 (Quinlan 1993): Expected decrease in (binary) entropy
CART (Breiman et al 1984): Expected decrease in Gini impurity (1-sum of squared class probabilities)

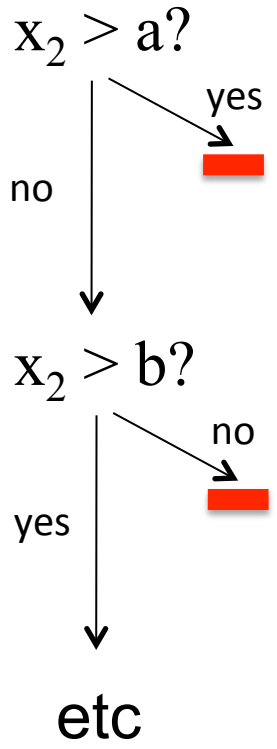
Decision trees



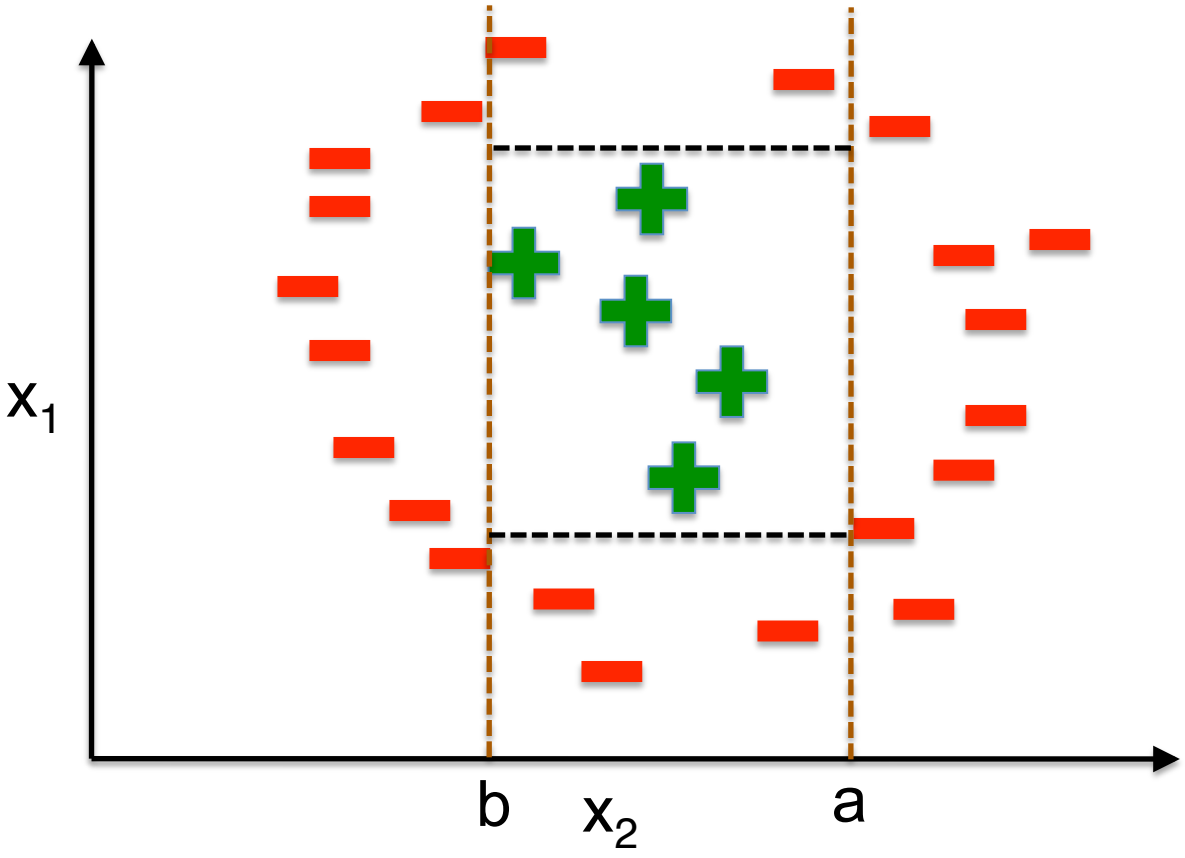


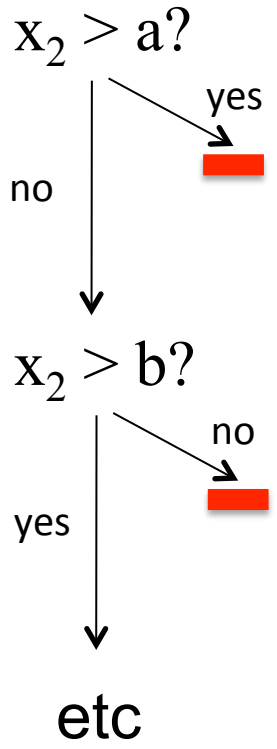
Decision trees





Decision trees





Decision tree summary

Decision trees learn a recursive splits of the data along individual features that partition the input space into “homogeneous” groups of data points with the same labels.

Ensemble classification

- Combining multiple classifiers together by training them separately and then averaging their predictions is a good way to avoid overfitting.
- **Bagging** (Breiman 1996):
  - Resample training set, train separate classifiers on each sample, have the classifiers vote for the classification
- **Boosting** (e.g. Adaboost, Freund and Schapire 1997):
  - Iteratively reweight training sets based on errors of a weighted average of classifiers:
    - Train classifier ("weak learner") to minimize weighted error on training set
    - Weight new classifier according to prediction error, reweight training set according to prediction error
    - Repeat
  - Minimizes exponential loss on training set over a convex set of functions

Non-linear classification summary

- Support vector machines:
  - Linear classification for “derived” features that are functions of the original features.
  - Can do linear classification using “kernel functions” that often measure the similarity of each new data point to those in the training set
  - Could try doing the same thing with elastic net regularized logistic regression
- Multi-layer perceptions or neural networks:
  - Also like linear classification but you learn the “derived” features.
- K-nearest neighbors:
  - Classify based on the majority vote of your k nearest neighbours.
- Decision tree:
  - Progressively splits the feature space into smaller and smaller boxes, so that each box is homogeneous. But you need to prune to avoid overfitting.

Which classifier should you choose?

- In practice, you should try all of the basic ones, and choose the best one.
- Or, choose one based on your prior knowledge about the problem.
- However, the “no free lunch” theorem suggests that every classifier is the best at least one task.
- People who win online contests, often combine the output of different methods – this is called *“ensemble learning”*.

Bagging

$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

Bootstrap samples

Train separate classifiers

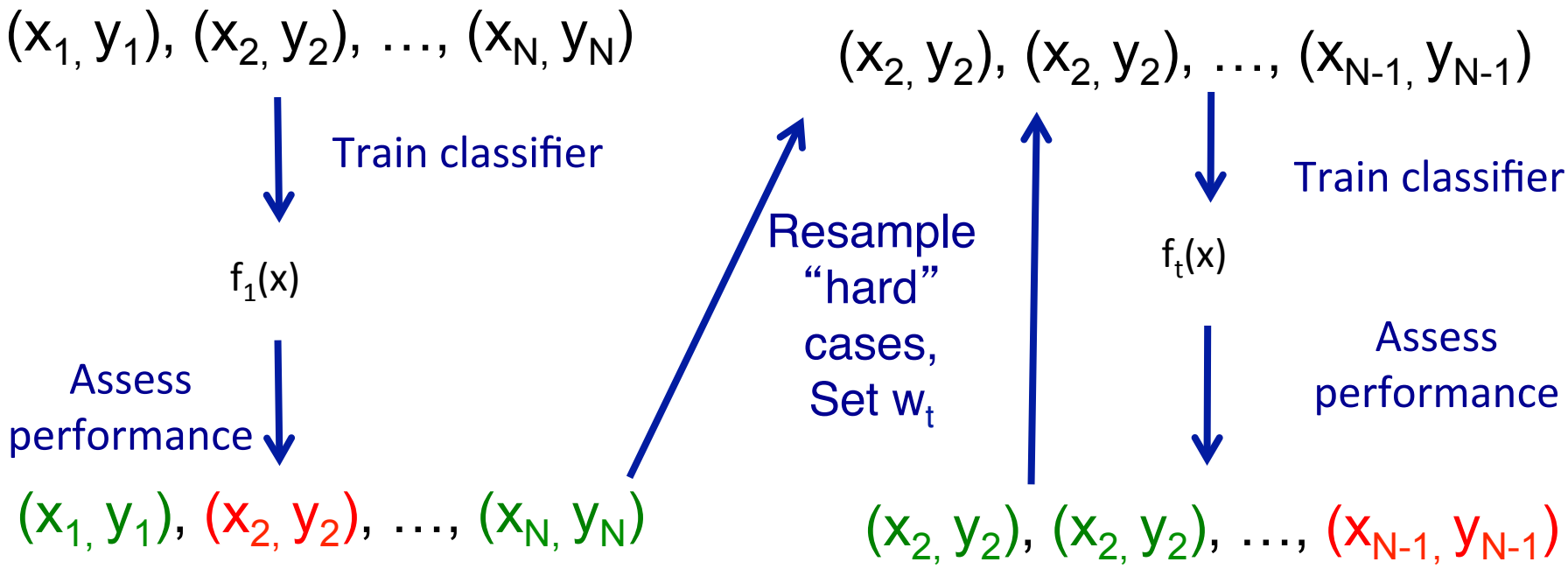
$(x_1, y_1), (x_1, y_1), (x_3, y_3), \ldots, (x_{N-1}, y_{N-1})$ → $f_1(x)$

$(x_1, y_1), (x_2, y_2), (x_4, y_4), \ldots, (x_N, y_N)$ → $f_2(x)$

etc (M samples in total)

$$\text{bagged}(\mathbf{x}) = \Sigma_j f_j(\mathbf{x}) / M$$

Boosting



$$\text{boost}(\mathbf{x}) = \Sigma_j w_j f_j(\mathbf{x})$$

Legend

(x_i, y_i) -- correct

(x_i, y_i) -- incorrect

Ensemble learning summary

- **Bagging:** classifiers trained separately (in parallel) on different bootstrapped samples of the training set, and make equally weighted votes. E.g. "Random Forests" is bagged decision trees.

- **Boosting:** classifiers trained sequentially, weighted by performance, on samples of the training set that focus on "hard examples". Final classification is based on weighted votes. E.g. gbm: Generalized Boosted Regression models in R

Random Forests (Breiman 2001)

- Construct M bootstrapped samples of the training set (of size N)
- For each sample, build DT using CART (no pruning), but split optimally on randomly chosen features – random feature choice reduces correlation among trees, this is a good thing.
- Since bootstrap resamples training set with replacement, leaves out, on average, $(1-1/N)^N$ x 100% of the examples (~100/e% = 36.7%), can use these out-of-bag samples to estimate performance
- Bag predictions (i.e. average them)
- Can assess "importance" of features by evaluating performance of trees containing those features

Networks as kernels

- Can use matrix representations of graphs to generate kernels.
- One popular graph-based kernel is the diffusion kernel:
  - $K = (\lambda I - L)^{-1}$
  - Where $L = D - W$, D is a diagonal matrix with the row sums, and W is the matrix representation of the graph.
- GeneMANIA label propagation:
  - $\mathbf{f} = (\lambda I - L)^{-1}\mathbf{y}$