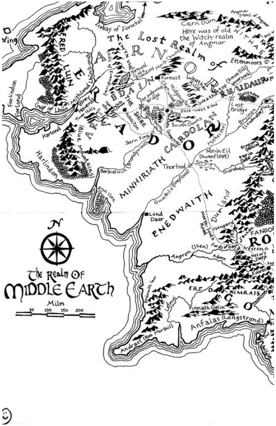
Where Are We?



We've looked at some simple (but high performing) supervised learning ideas for estimating the class of individual documents (Naive Bayes), and for estimating proportions.

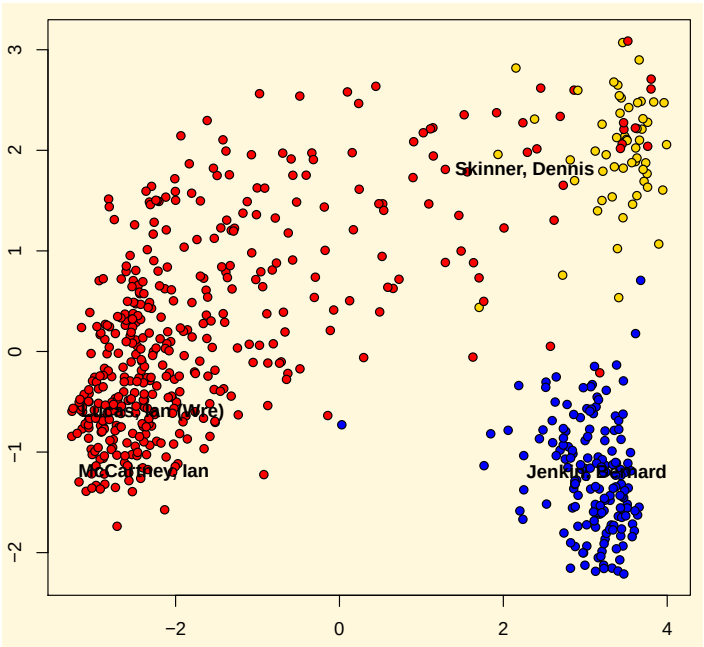
Now want to cover powerful, commonly used techniques from machine learning that appear in social science research: SVM, KNN, etc. . .

These techniques involve some important decisions about the bias-variance tradeoff, and the use of (cross) validation in checking model performance and selecting the best model.

Remember. . .

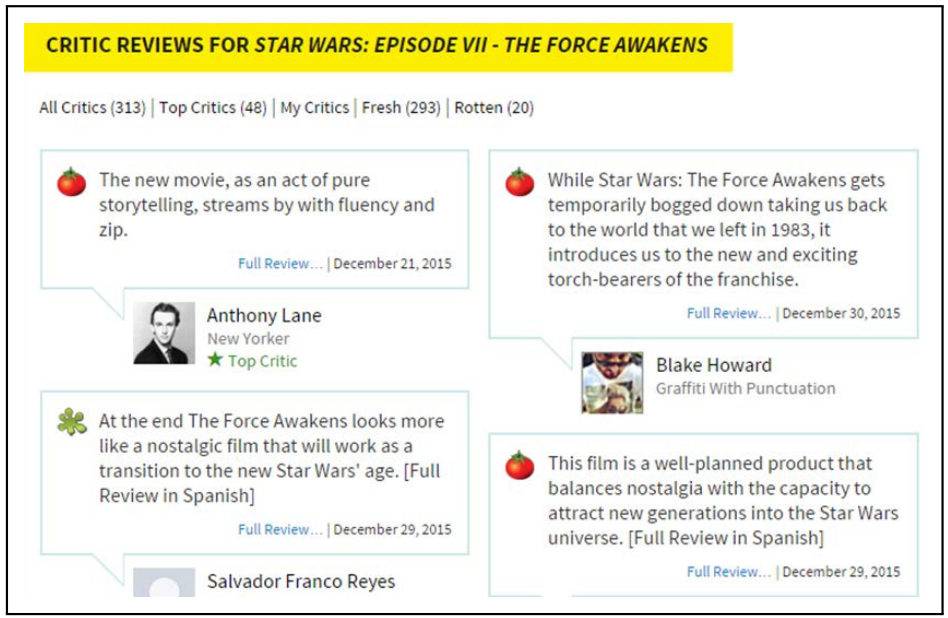
Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized— by party? by ideology? by race?



Supervised techniques: learning relationship between inputs and a labeled set of outputs.

e.g. opinion mining: what makes a critic like or dislike a movie ($y_i \in \{0, 1\}$)?



Workflow of Supervised Learning: Bias/Variance Tradeoff

Workflow of Supervised Techniques

1. Obtain/code the training set and decide on relevant features (preprocess).
2. Decide on the algorithm, possibly matched in some way to nature of problem.
3. Adjust algorithm for optimal performance, perhaps using validation set and/or some kind of cross-validation.
4. Report accuracy in test set.

Notes and Issues

Supervised techniques are about learning relationship between X and labeled data. Often used interchangeably with *machine learning*: idea that computers could 'learn' relationships without specific programming.

Often used when $p >> n$: number of parameters (e.g. coefficients on terms) is far larger than number of observations (e.g. speakers or documents).

$\rightarrow$ results in general curse of dimensionality wherein feature matrix is large (e.g. $100k$ columns) and sparse and thus obtaining meaningful estimates is difficult.

So techniques may require careful tuning of *regularization parameters* to obtain good performance.

Notes and Issues II

Once we have the training set we face a dilemma...

1 we can use our technique to fit a very complicated model to this data (perfectly), including noisy elements. This avoids bias, but it incurs variance (when we move to the test set).

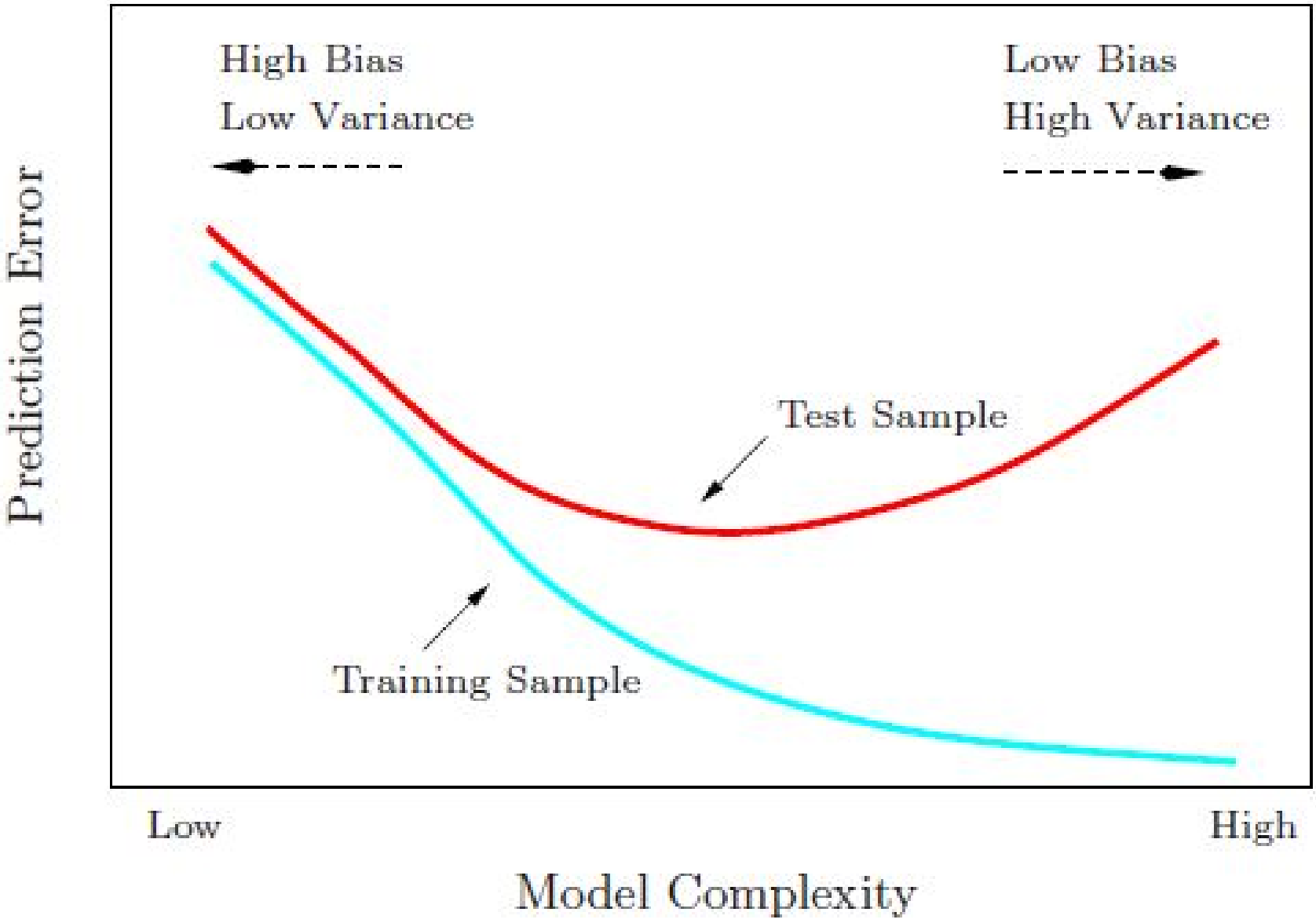
→ we have overfit to our training set, and may do poorly on our test set.

2 we can use be more relaxed about the fit of our algorithm in the training set, and accept some imperfections in performance. This avoids high variance, but may induce bias in the sense that we miss important relationships in the data.

→ we have underfit to our training set, and may do poorly on our test set.

So managing the *bias-variance tradeoff* is a key element of supervised learning, and we may need to tune our algorithms with that in mind.

Bias-Variance Tradeoff (Hastie et al, p38)



Cross-Validation

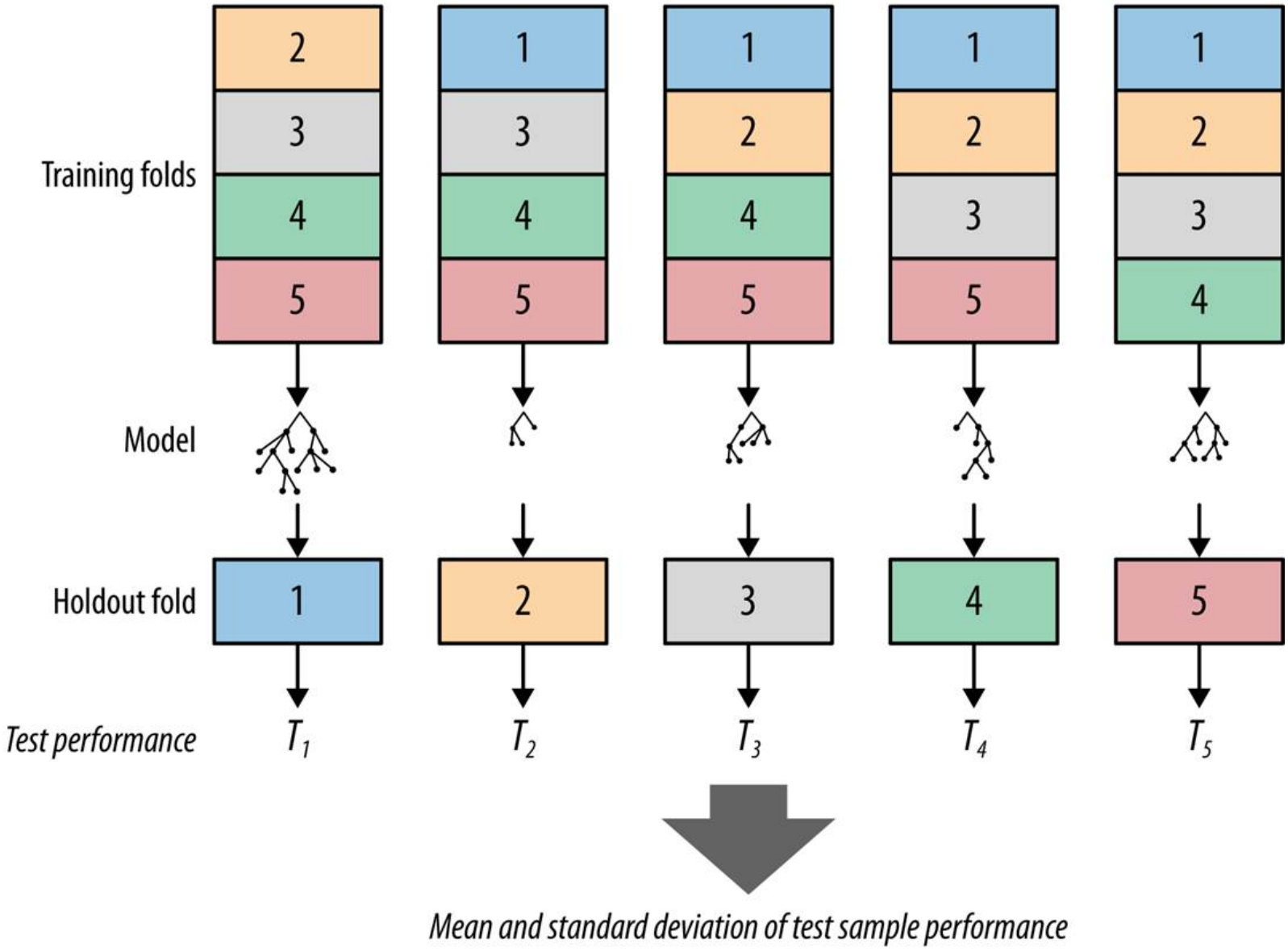
Want to properly estimate model performance (bias and especially variance).

Popular and efficient approach is k-fold cross validation, esp when we don't have enough data to form a completely separate validation set.

1. divide all data into k equal size chunks ($k = 10$ is common; $k = n$ is 'leave one out'), and set the parameter(s) of model at particular value,
2. repeat the following k times (folds):
   - 2.1 grab one of the k chunks as a validation set (each only used once)
   - 2.2 grab the other $k - 1$ chunks as a training set
   - 2.3 test on the validation set, record prediction error
3. Average over runs to get prediction error estimate.

$\rightarrow$ Can follow same steps for models of different specifications; variant on this approach can be used for model selection, directly.

Graphically



Support Vector Machines

Background

- Basics of the technique developed by Vladimir Vapnick for his PhD thesis in USSR in the 1960's.
- Never used because no computers powerful enough in USSR to apply it.
- Until... Vapnick emigrated to the US and was hired by Bell Labs in the 1990's.
- A very stable classifier. Used for text classification, character recognition, image recognition in general, etc....

Motivating Example Diermeier et al, 2011

Diermeier et al want to know what terms are most indicative of conservative or liberal positions in legislative debates.

Study the speeches of the 25 most liberal and the 25 most conservative senators (1989–2004), selected based on their NOMINATE scores (from roll call behavior, only)

Code the Republicans as $+1$, Dems as -1: $y_i \in \{-1, +1\}$

Methodological Problem

Want to know the relationship between using different terms and ideological views.

e.g. is '`Ethanol`' a 'Democrat' term or a 'Republican' term, and to what degree?

Have the (stemmed, stopped, weighted etc) speech term matrix for each Senator as X.

Their training set is speech output of most extreme Senators between the 101st and 107th Congress

Their test set is speech output of most extreme Senators in 108th Congress.

What method to use?

Support Vector Machines

Idea a classifier that builds a model from a binary labeled training set and returns an optimal hyperplane that puts new examples into one of the categories non-probabilistically.

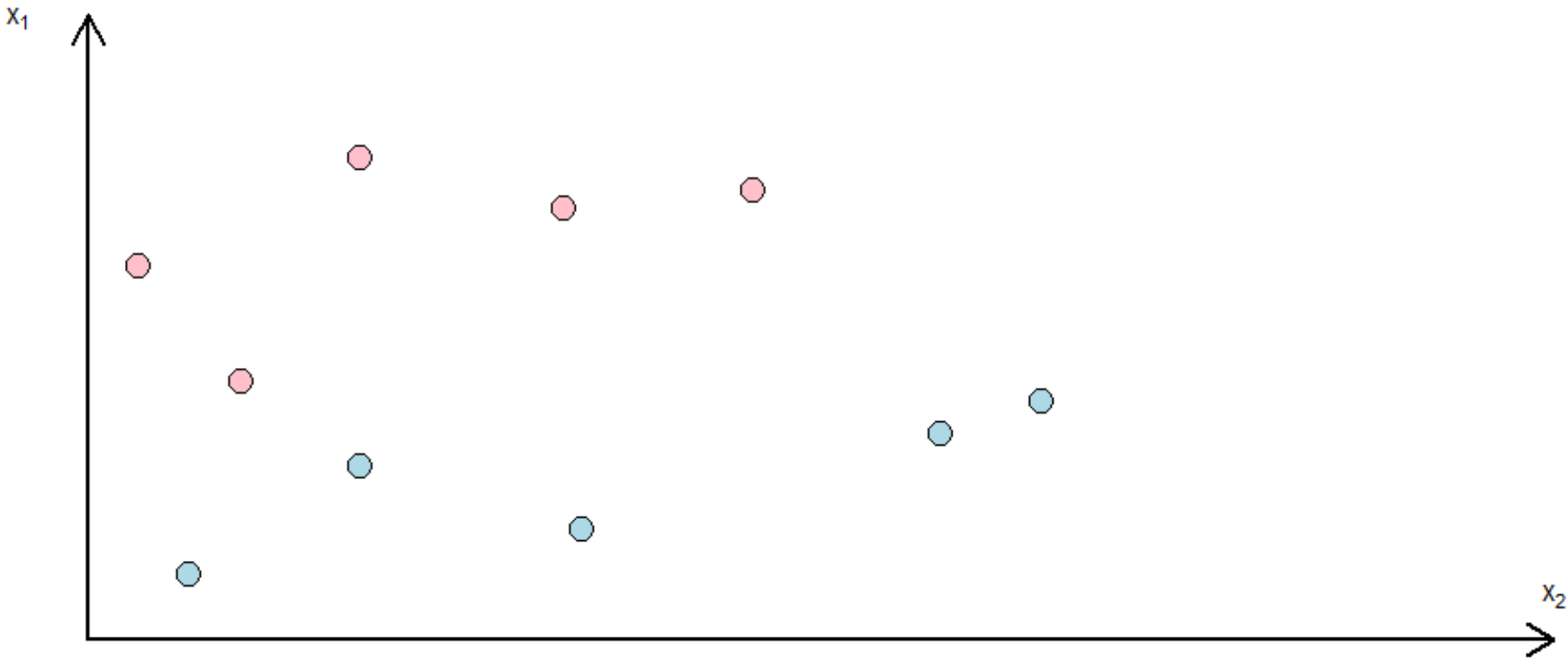
Here We have 10 Senators that belong to one of two classes: five Republicans, five Democrats. $y_i \in \{-1, +1\}$

Each senator has a number of p features, which are the term weights from their speeches. To make things simple, suppose that $p = 2$: there are (only) two features, x_1 and x_2 per observation.

Assume that the observations are linearly separable: if we plot the Senators in two dimensions (x_1, x_2), we can divide them (perfectly) into the two parties using a straight line.

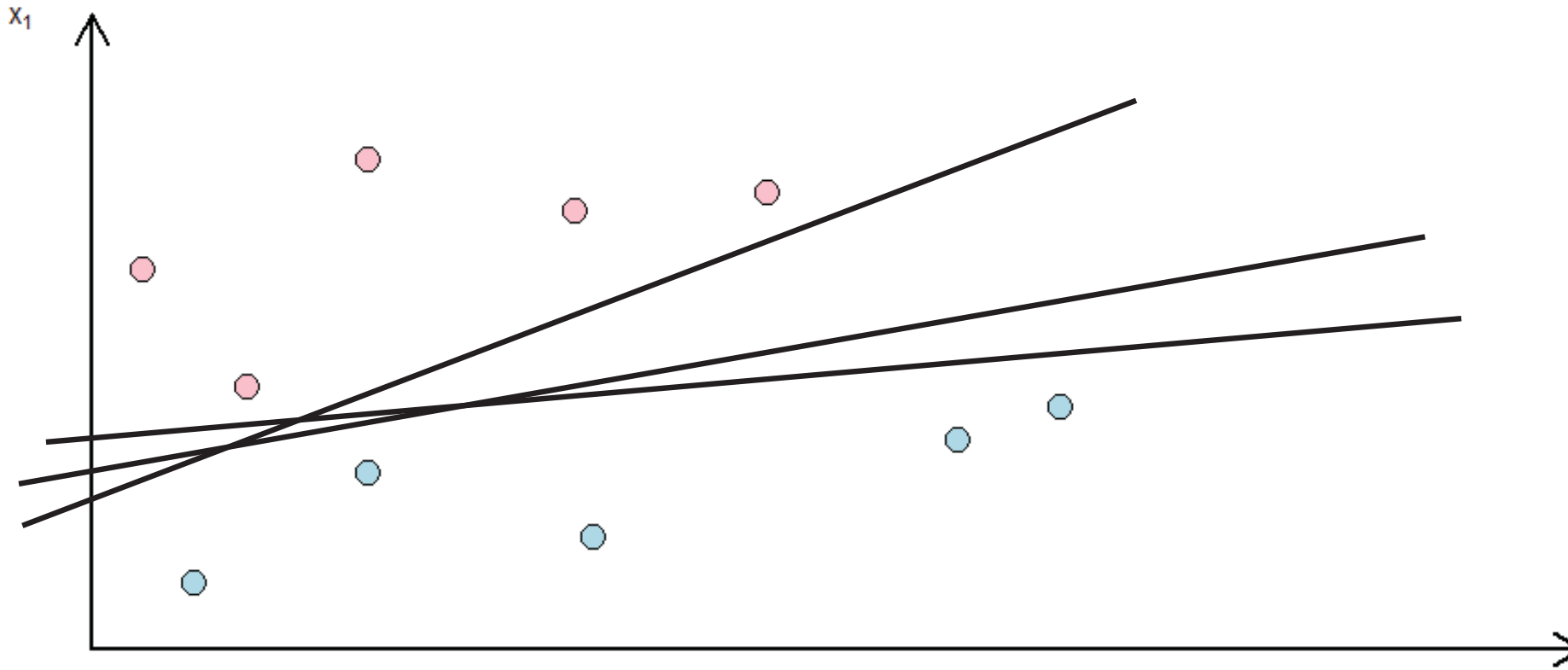
$\rightarrow$ of course, a real problem would have p being large, and the space being very high dimensional, but the logic is the same.

The 10 Senators



As the parties linearly separably? Where could you draw the line?

The 10 Senators



Which line should we prefer?

What is the Optimal Hyperplane?

There are many possible lines.

But we want to avoid ones that pass close to the points: such lines will tend to be sensitive to noise and make classification errors with future examples.

So pick line that gives largest minimum distance from the training cases. That is, the line that's as far as possible from the closest cases on both sides.

$\rightarrow$ That optimal line—the separating hyperplane—is the maximum margin hyperplane. It will maximize the margin of the training data.

How to get it I: Notation Variants

We can write any line as $y = ax + b$ or $y - ax - b = 0$.

Often rewrite terms as $\mathbf{w} = [-b, -a, 1]$ and $\mathbf{x} = [1, x, y]$

$\rightarrow$ $\mathbf{w}^T\mathbf{x} = -b \times 1 + (-a) \times x + 1 \times y = y - ax - b = 0$.

or use dot product: $\mathbf{w} \cdot \mathbf{x} = 0$

Also popular: can use a two dimensional form—

$\mathbf{w} = [-a, 1]$ and $\mathbf{x} = [x, y]$, so that $\mathbf{w} \cdot \mathbf{x} = y - ax$

thus $\mathbf{w} \cdot \mathbf{x} - b = 0$ (since $0 = y - ax - b$)

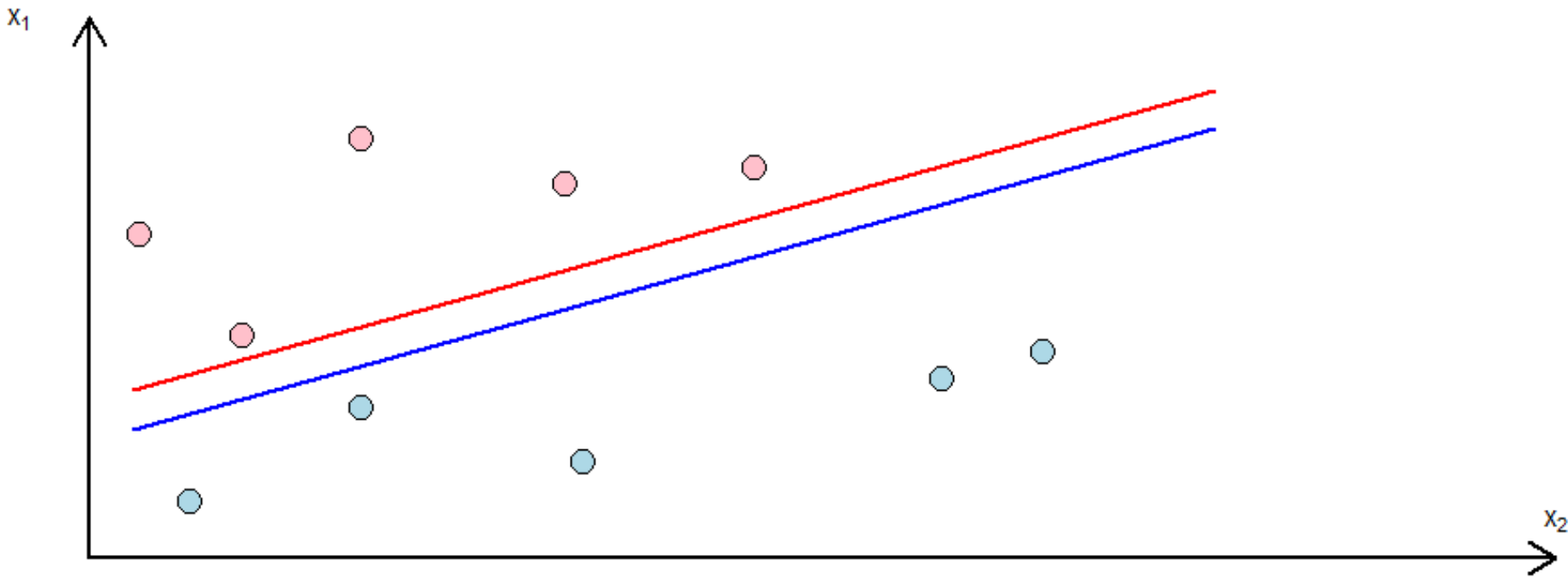
How to get it II: working with the equation

Our hyperplane (line) will separate the data, and will satisfy $\mathbf{w} \cdot \mathbf{x} - b = 0$

We can think of it as the line that is equidistant, i.e. half-way between, two parallel hyperplanes (H_R and H_D) that separate the Reps from the Dems.

Those parallel lines should be as far from each other as possible without misclassifying anybody: the distance between them is the margin and we want that to be maximized.

How Could We Do Better?



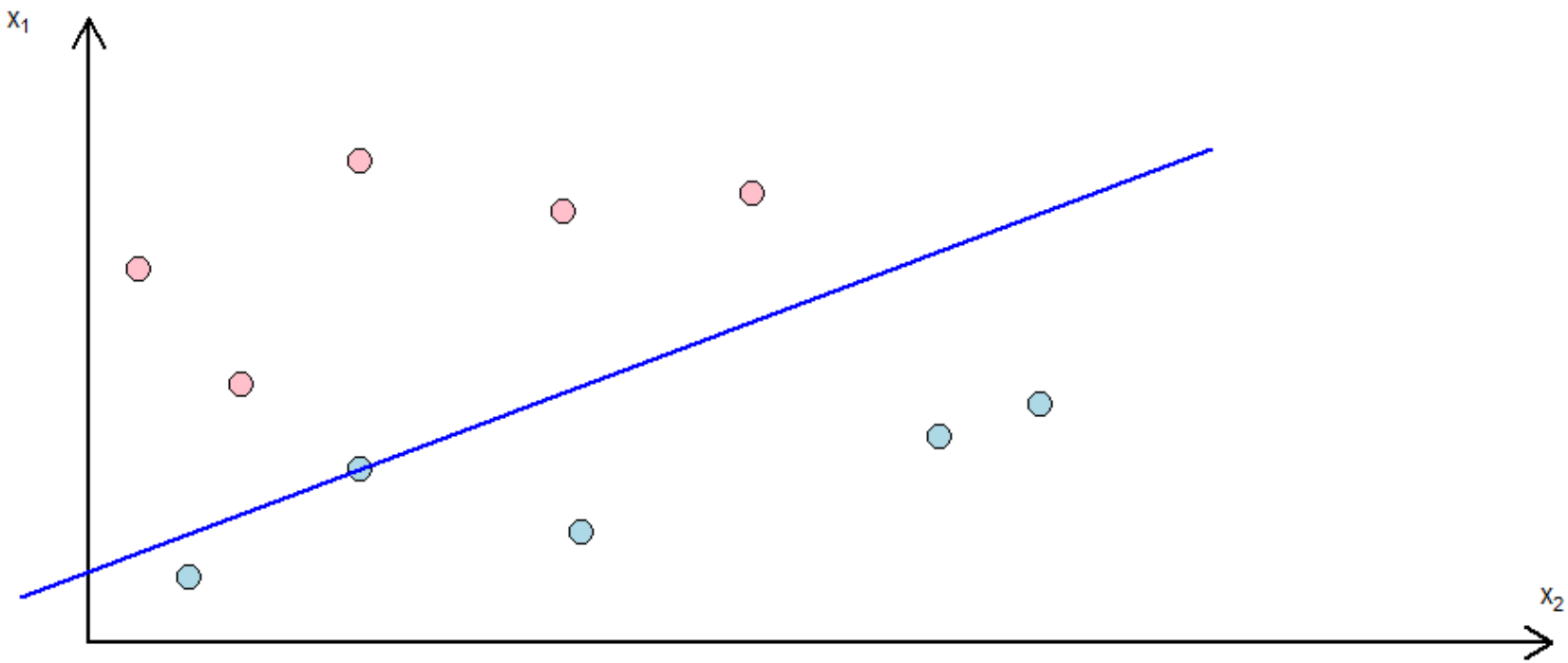
How to get it II: working with the equation

Our hyperplane (line) will separate the data, and will satisfy $\mathbf{w} \cdot \mathbf{x} - b = 0$

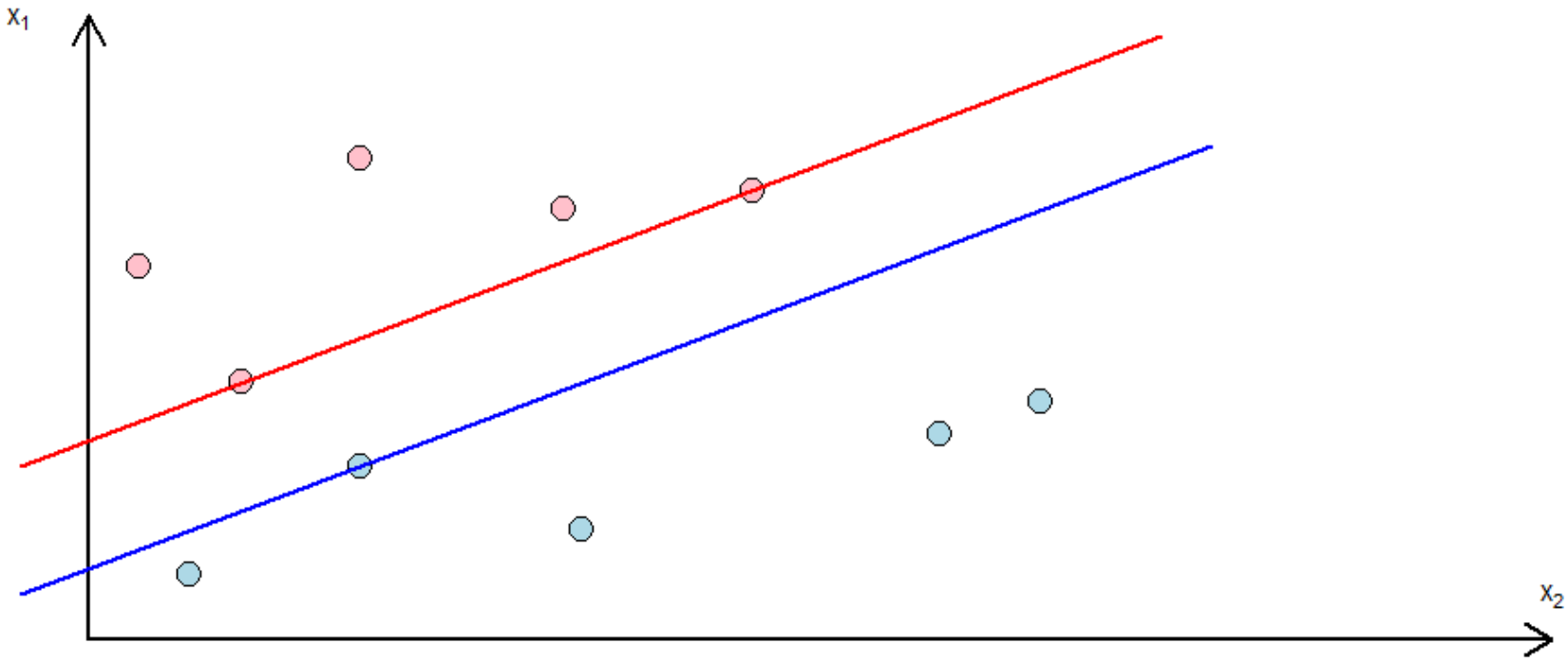
We can think of it as the line that is equidistant, i.e. half-way between, two parallel hyperplanes (H_R and H_D) that separate the Reps from the Dems.

Those parallel lines should be as far from each other as possible without misclassifying anybody: the distance between them is the margin and we want that to be maximized.

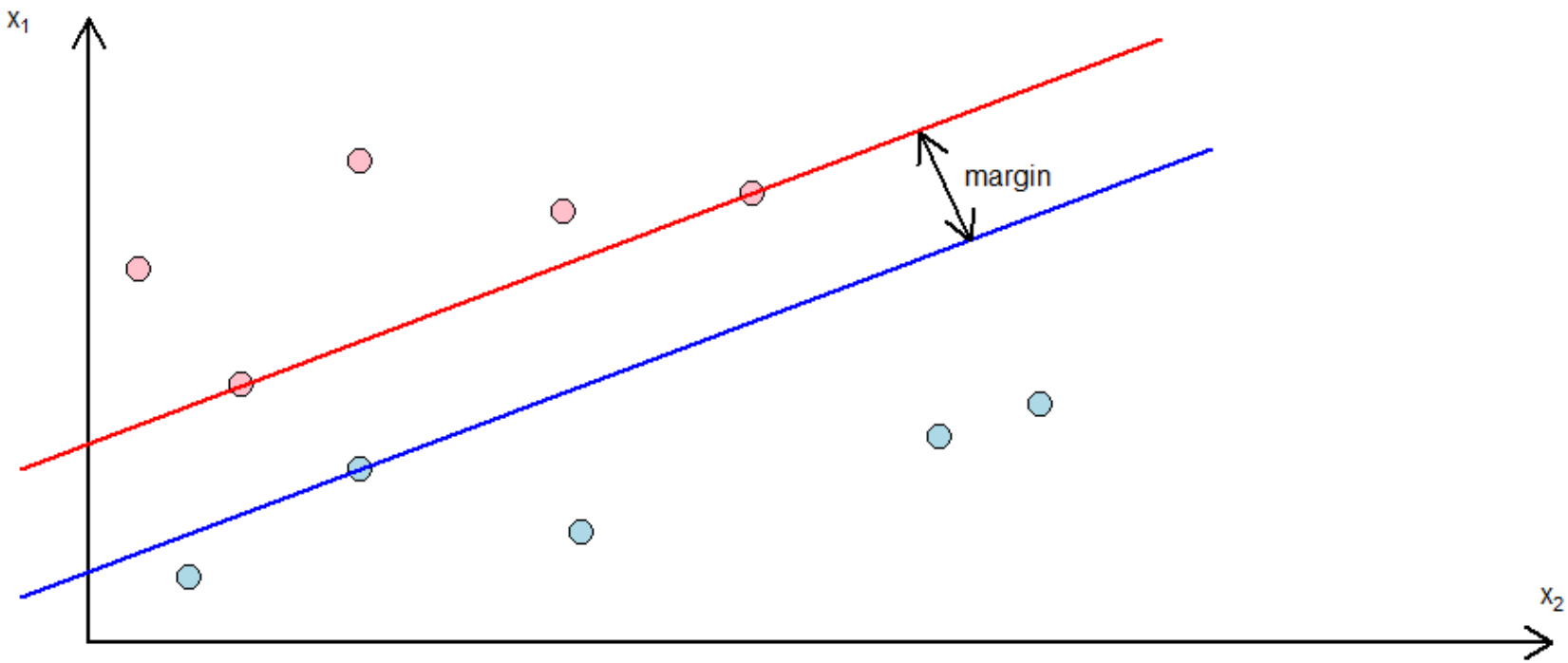
Graphically. . .



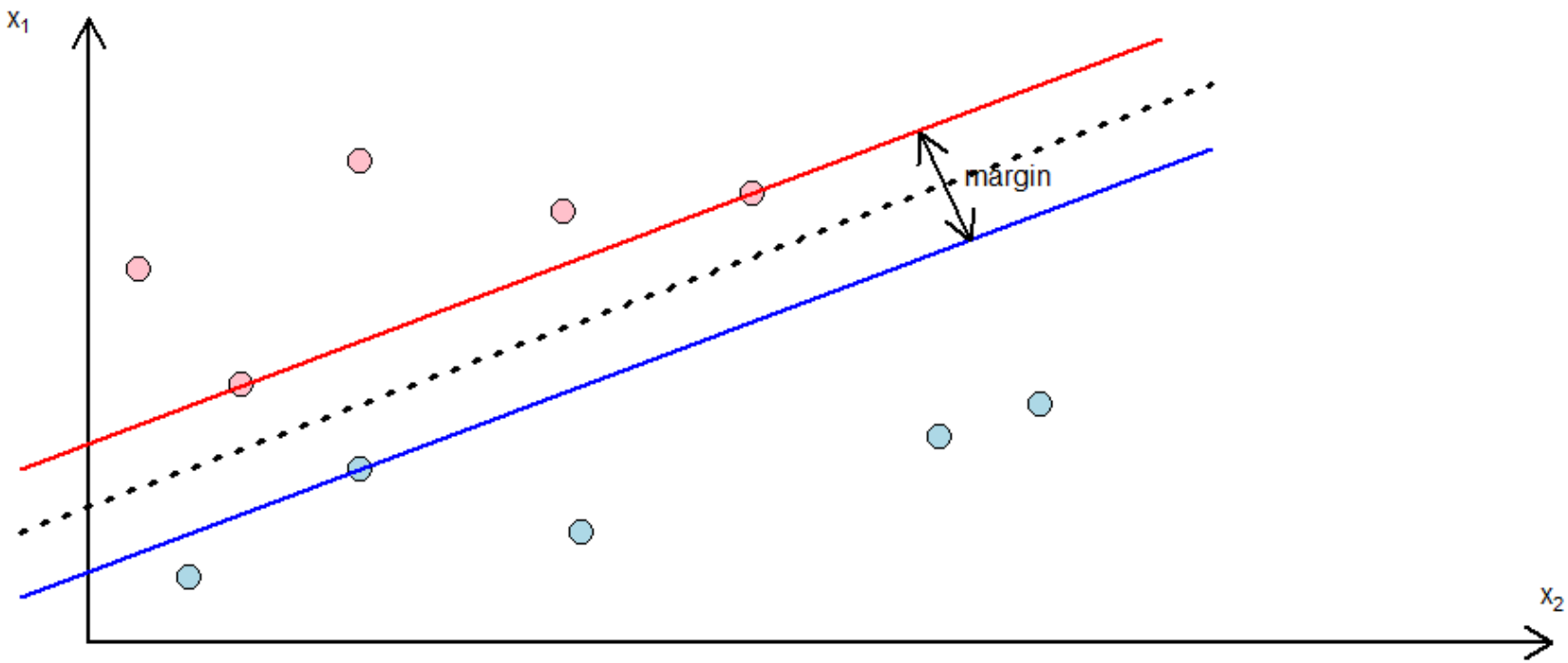
Graphically. . .



Graphically. . .



Graphically. . .



The parallel hyperplanes

We want all the Republicans ($y_i = 1$)—and *only* the Republicans—to be classified as Republicans (given their xs).

This means that 'their' hyperplane should 'capture' them all, and separate them fully from the Democrats (in our 2D space).

→ requires that $\mathbf{w} \cdot \mathbf{x} - b \geq 1$, if $y_i = 1$

Same idea for the Democrats: $\mathbf{w} \cdot \mathbf{x} - b \leq -1$, if $y_i = -1$

The distance between the two hyperplanes (H_R and H_D) we construct is the margin, and its width is $\frac{2}{||\mathbf{w}||}$, where $||\mathbf{w}||$ is the norm of $\mathbf{w}$

→ $||x|| = \sqrt{x_1^2 + x_2^2 \ldots x_n^2}$

→ This is not trivially obvious! If you want more background, here is an incredible lecture by Patrick Winston at MIT....

→ By the way this talk by Winston on "How to Speak" was legendary at MIT. You should all watch it as well.

Optimization Problem

minimize $||\mathbf{w}||$ subject to $y_i(\mathbf{w} \cdot \mathbf{x} - b) \geq 1 \ \ \forall i$

This will give us the two hyperplanes H_R and H_D, and the line equidistant between them will be our classifier.

Turns out that the minimization of $||\mathbf{w}||$ is amenable to quadratic programming methods (the economists in the room are familiar with at least one of these—constrained optimization using the Lagrangian multiplier).

Notes

At least one of the Senators, in one of the parties, will be on 'their' line. Why?

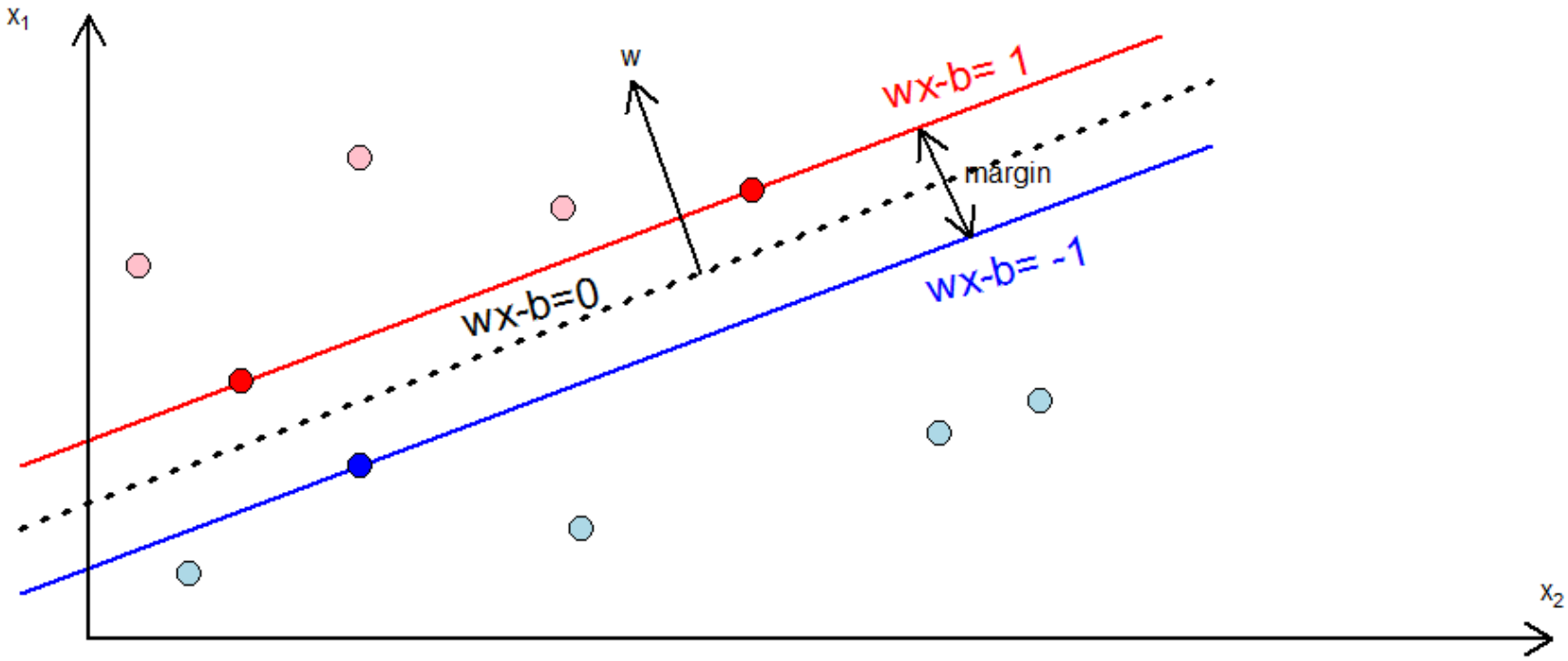
A: Suppose nobody is on either line. Then we must be able to move the lines 'apart' a little further and make the margin bigger...

But as soon as someone is pushed between the lines, we've broken the rule about no misclassifications, and that's a constraint we have to follow.

In fact, the points closest to the separating hyperplane will be the Senators lying on their (respective) parallel hyperplanes.

We use the term support vectors to describe the training examples closest to the hyperplane. Those support vectors completely determine where our maximum-margin hyperplane will be.

The Support Vectors



The support vectors lie on...

$\mathbf{w} \cdot \mathbf{x} - b = 1$ or $\mathbf{w} \cdot \mathbf{x} - b = -1$

SVM weights

So We have a hyperplane that separates the Democrats from the Republicans.

The vector **w** is orthogonal to that, and when we multiply it (dot product) by **x** (and add b term) we get the predicted class of a 'new' Senator.

i.e. if the product is positive $\rightarrow$ Republican; if negative $\rightarrow$ Democrat

Plus for each feature, $x_1, x_2, \ldots$, the vector **w** gives us a weight. The absolute size of the weights—relative to one another—is a measure of how important that feature is for separating the Senators into Democrat and Republican.

NB SVMs typically have few features with non-zero weights, and those that are non-zero come from the support vectors (the 'important' observations)

Back to Diermeier et al, 2011

Achieve 92% accuracy (!)

Sort words according to coefficients: very positive weights imply conservative words; very negative weights imply liberal words. Argue that it is 'values' rather than economics that separates liberals from conservatives.

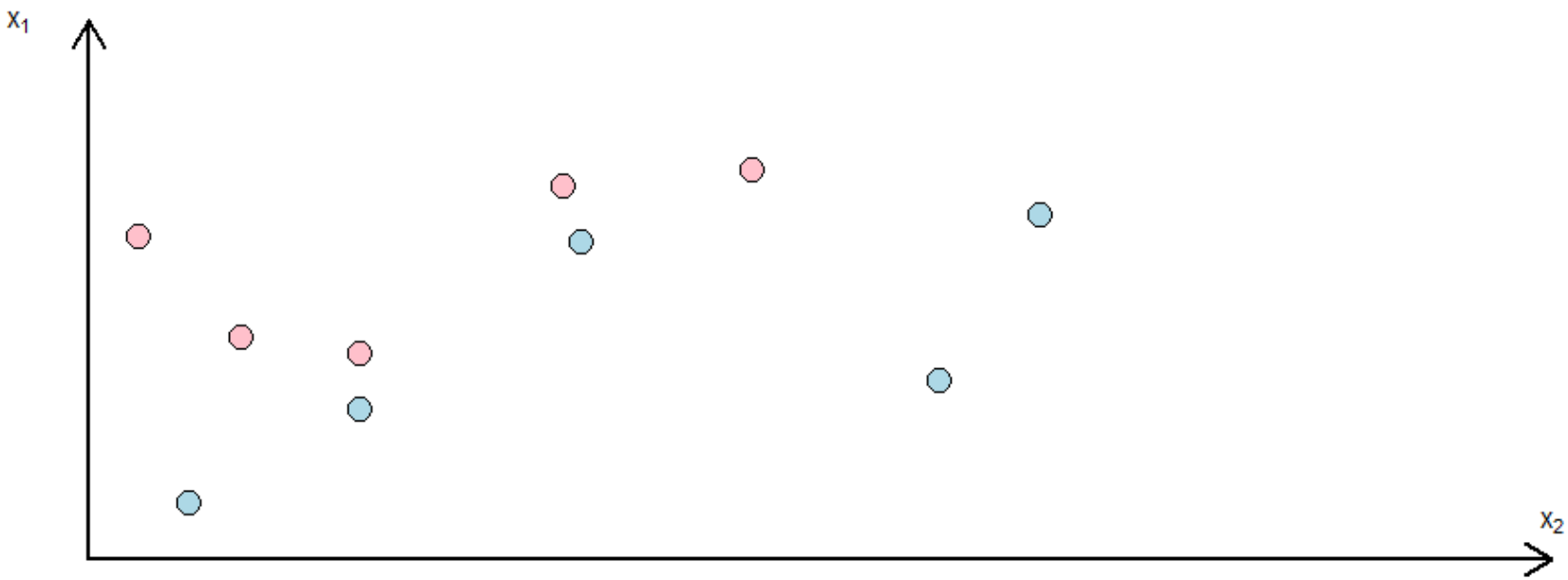
Words			
Liberal		Conservative	
FAS: −199.49	SBA: −113.10	habeas: 193.55	homosexual: 103.07
Ethanol: −198.92	Nursing: −109.38	CFTC: 187.16	everglades: 102.87
Wealthiest: −159.74	Providence: −108.73	surtax: 151.81	tower: 101.67
Collider: −142.28	Arctic: −108.30	marriage: 145.79	tripartisan: 101.23
WIC: −140.14	Orange: −107.98	cloning: 141.71	PRC: 102.90
ILO: −139.89	Glaxo: −107.81	tritium: 133.49	scouts: 97.55
Handgun: −129.01	Libraries: −107.70	ranchers: 132.95	nashua: 99.32
Lobbyists: −128.95	Disabilities:−106.44	BTU: 121.92	ballistic: 97.22
Enron: −127.71	Prescription: −106.31	grazing: 121.59	salting: 94.28
Fishery: −127.30	NIH: −105.52	unfunded: 120.82	abortion: 91.94
Hydrogen: −122.59	Lobbying: −105.35	catfish: 120.82	NTSB: 93.81
Souter: −121.40	NRA: −105.20	IRS: 114.91	Haiti: 97.28
PTSD: −119.87	Trident: −104.15	unborn: 111.88	PAC: 92.85
Gun: −119.52	RNC: −103.46	Taiwan: 111.13	taxing: 90.39

Beyond basic (hard margin) SVM

What if. . .

The Senators were not linearly separable? ('soft margin' SVM problem)

Graphically. . .



What if. . .

The Senators were not linearly separable? ('soft margin' SVM problem)

Can introduce a hinge loss function into the minimization problem. . .

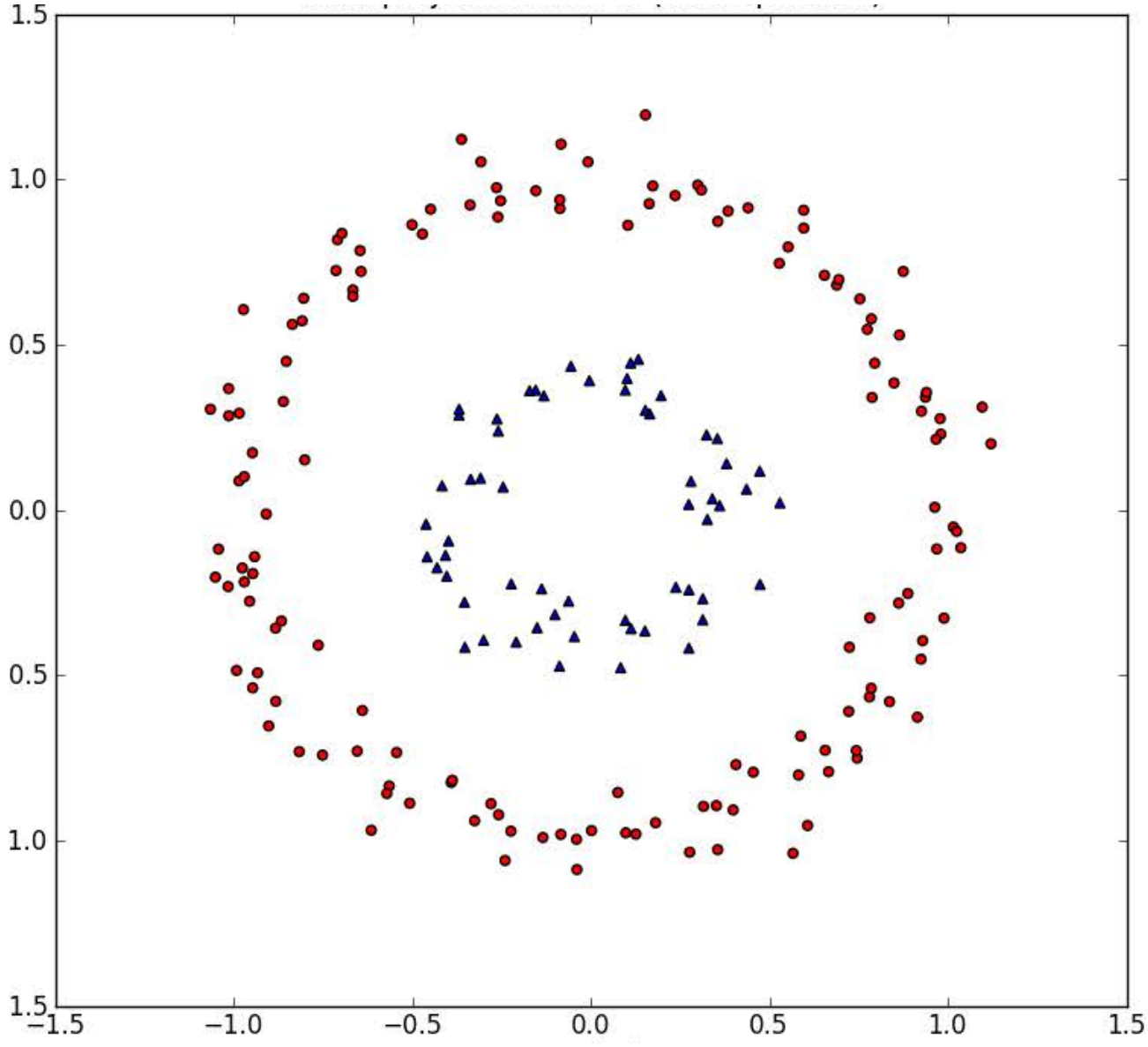
$= 0$ if the xs are on the 'correct' side of the margin. . .

And proportional to the distance from the margin *if* the point is on the 'wrong' side of the margin.

Hyperplane(s) will be drawn in way that is more sensitive to 'bigger' mistakes in classification.

What if. . .

The situation was considerably ‘worse’, such that neither a ‘hard margin’ nor ‘soft margin’ linear approach would work?



from www.eric-kim.net

What if...

The situation was considerably 'worse', such that neither a 'hard margin' nor 'soft margin' linear approach would work?

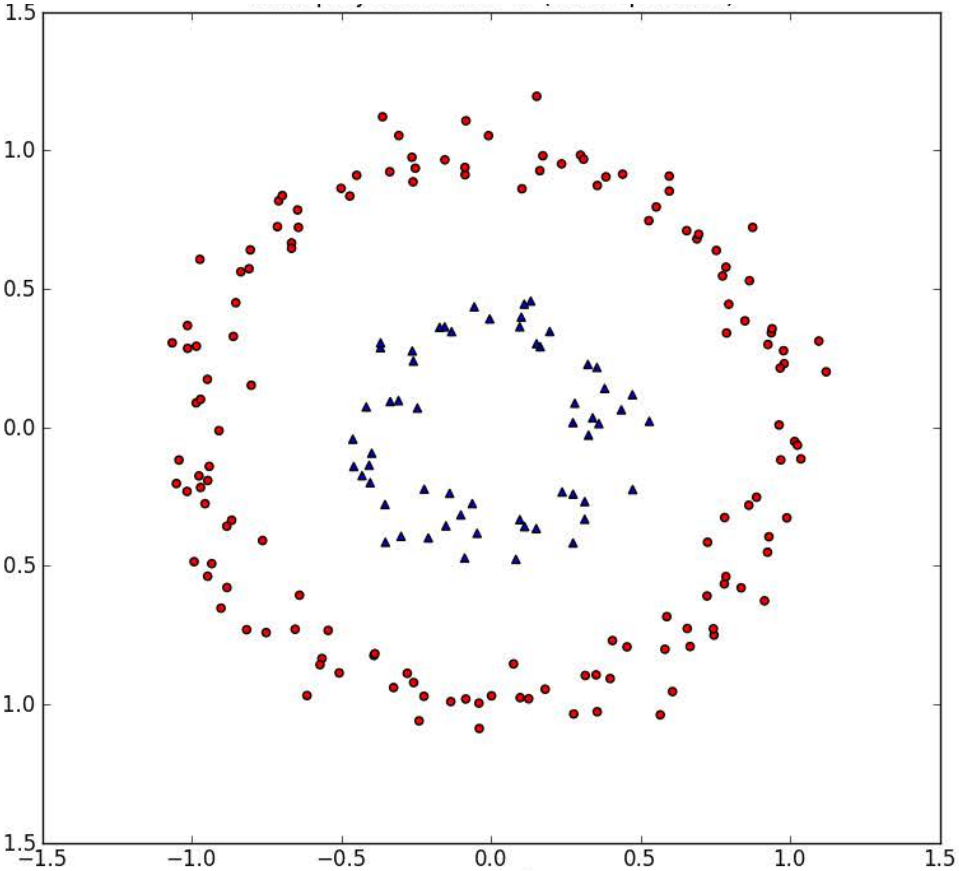
May be a way to transform the data, using a transformation ϕ, such that it *can* now be separated using a linear classifier.

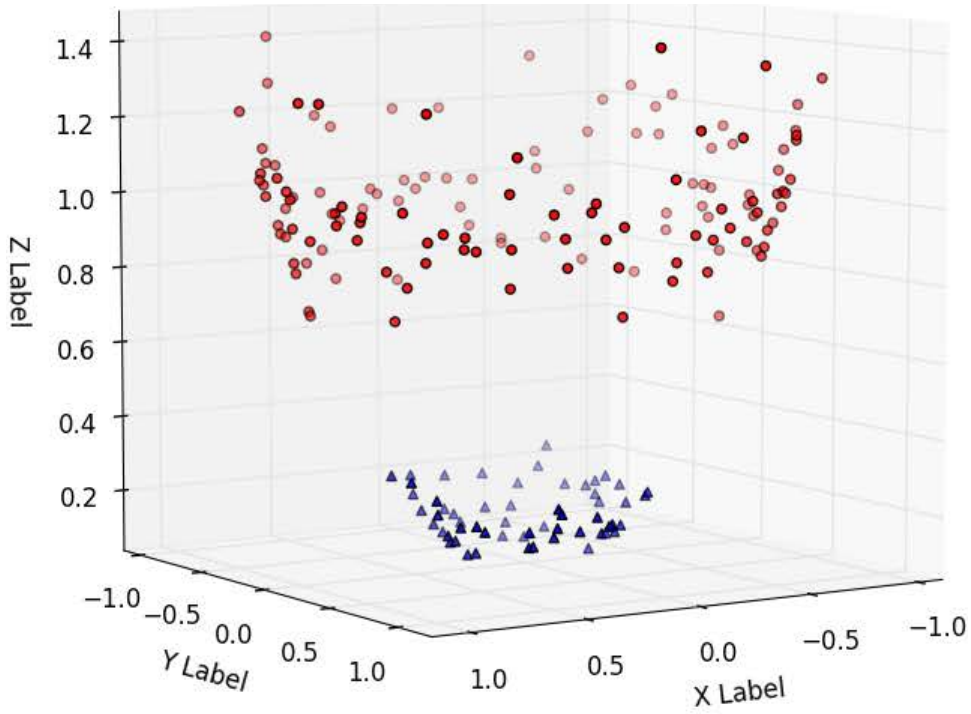
When the data is in a two dimensional feature space, we can lift it into a three dimensional feature space using

$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2).$$

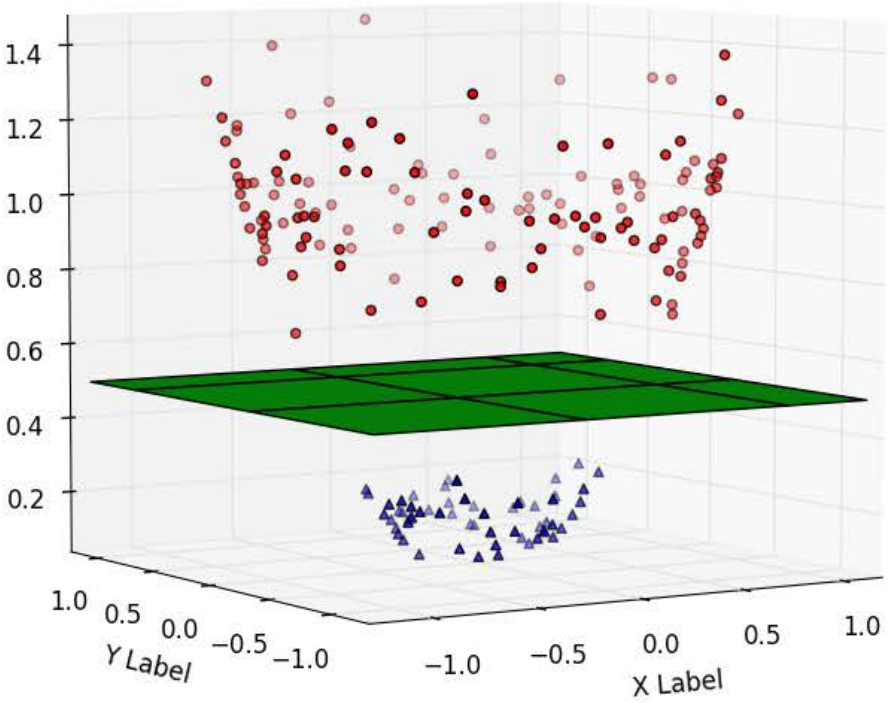
Then use a linear SVM on the transformed data set, and then map back to the original 2D space.

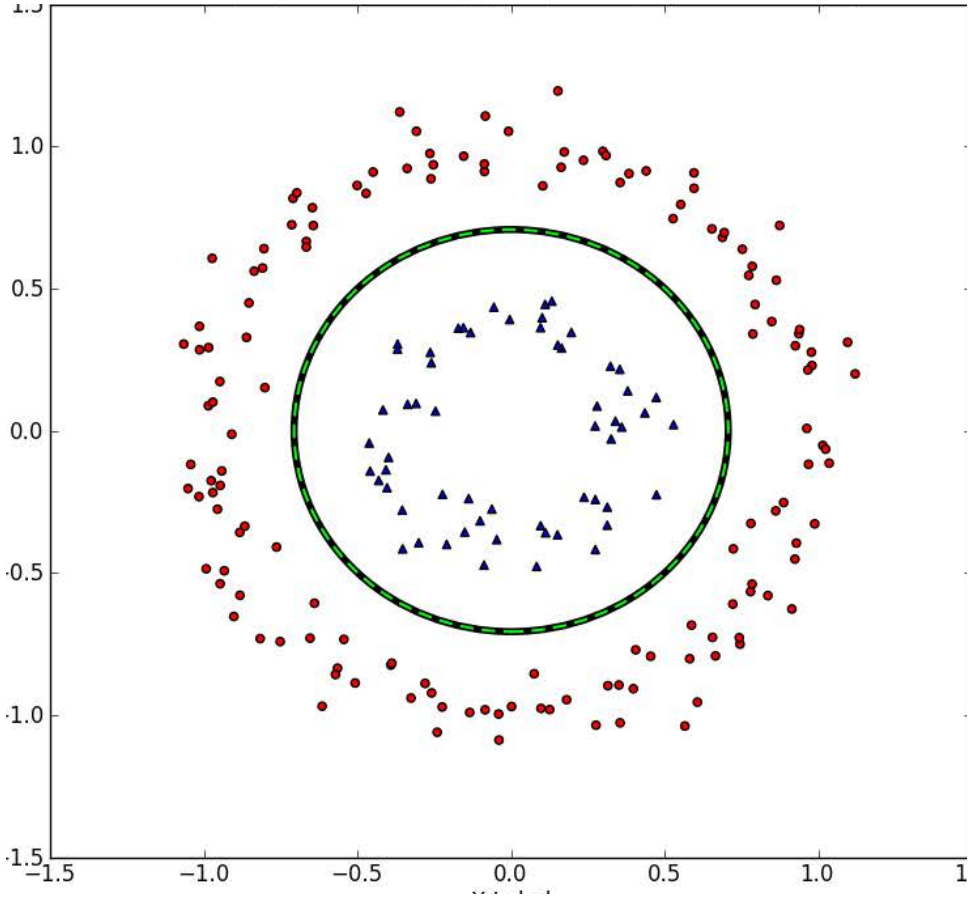
$\rightarrow$ Results in a non-linear hyperplane once back in 2 dimensions.





from www.eric-kim.net





from www.eric-kim.net

Kernel Methods

Explicitly transforming data into a new space can be very expensive in terms of computation.

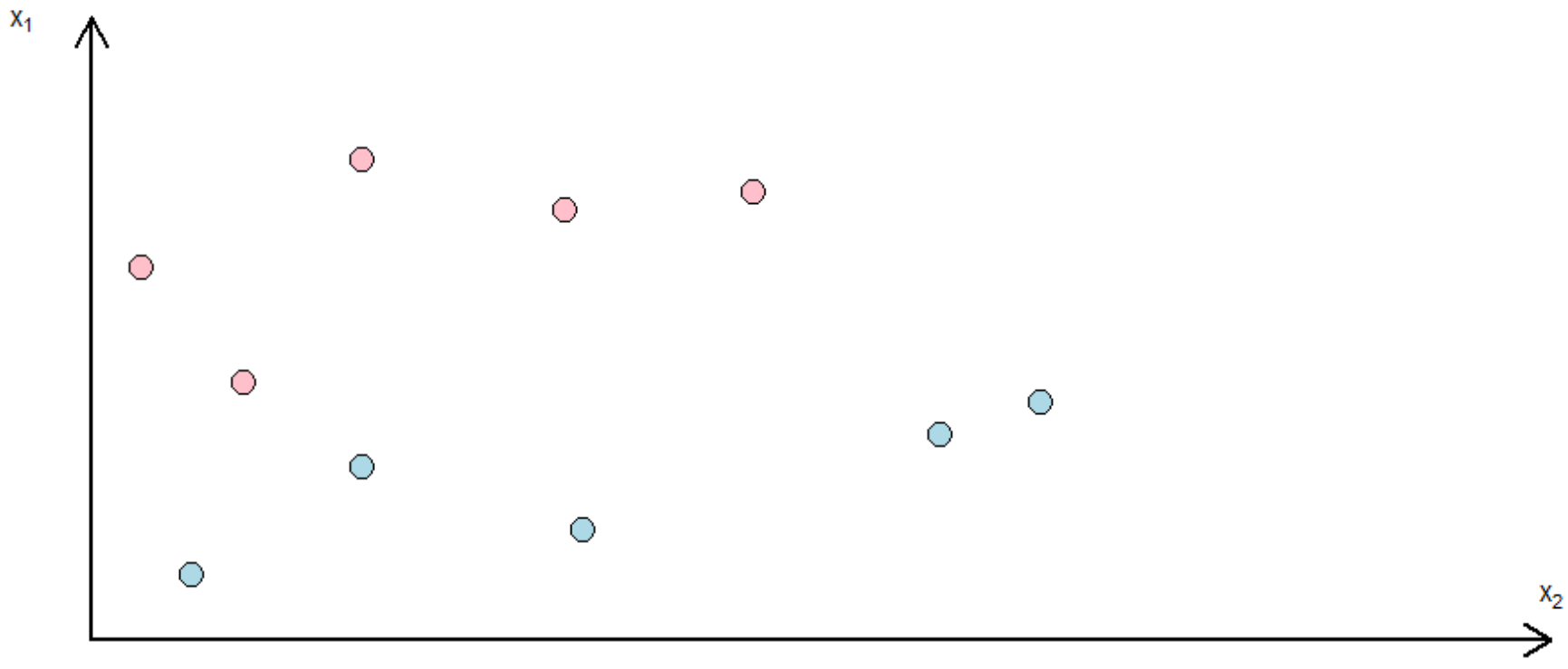
But What if we could do the transformation, and take the distances between observations implicitly, and use them for our classification?

$\rightarrow$ exactly what kernel methods do: use kernel functions, $K(\mathbf{x}_i, \mathbf{x}_j)$ to compute the i, j pairwise dot products for training data *as if* it had been transformed. Can then feed those inner products to the classifier.

this 'kernel trick' cuts cost considerably, though choosing and tuning the 'correct' kernel may be difficult.

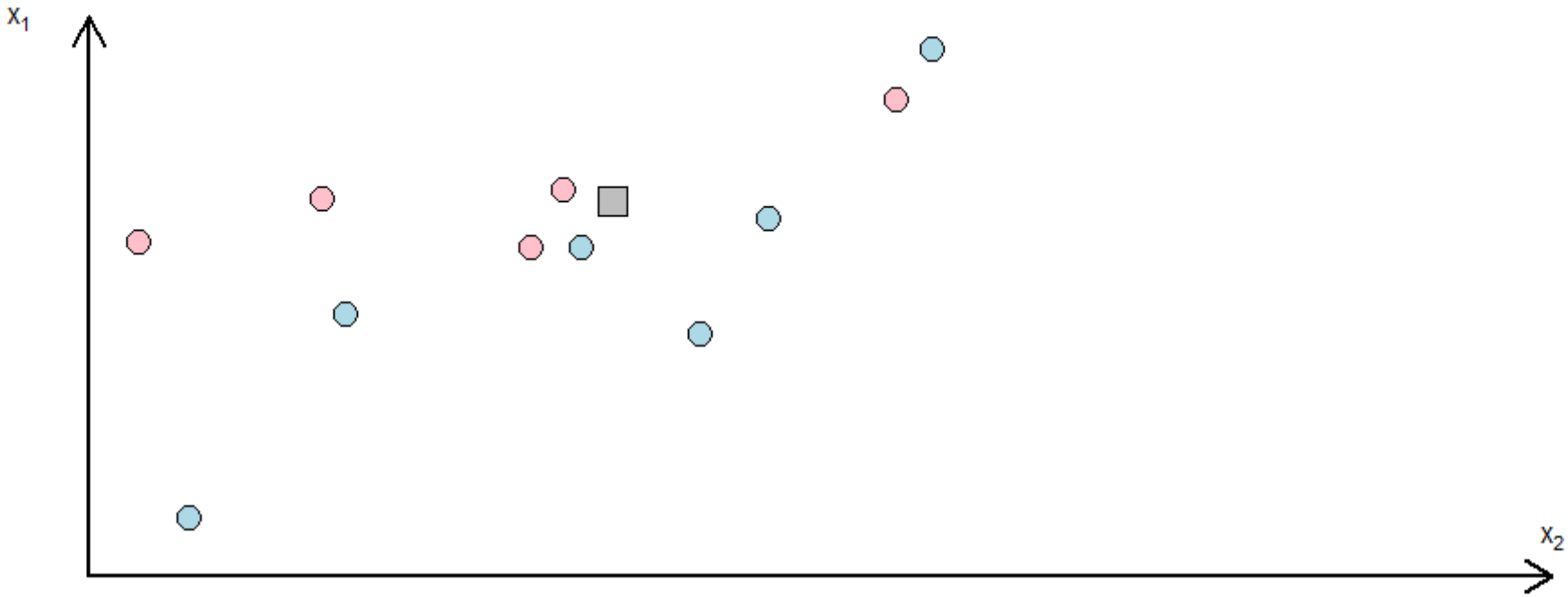
For text analysis, string kernels use a function $K(a, b)$ to implicity calculate the distance between strings of characters via the number of subsequences they have in common.

Reminder: The 10 Senators



k-nearest neighbors

Variant of the Senate example: now suppose we are interested in classifying a 'new' (test set) Senator (■) based on her feature values.

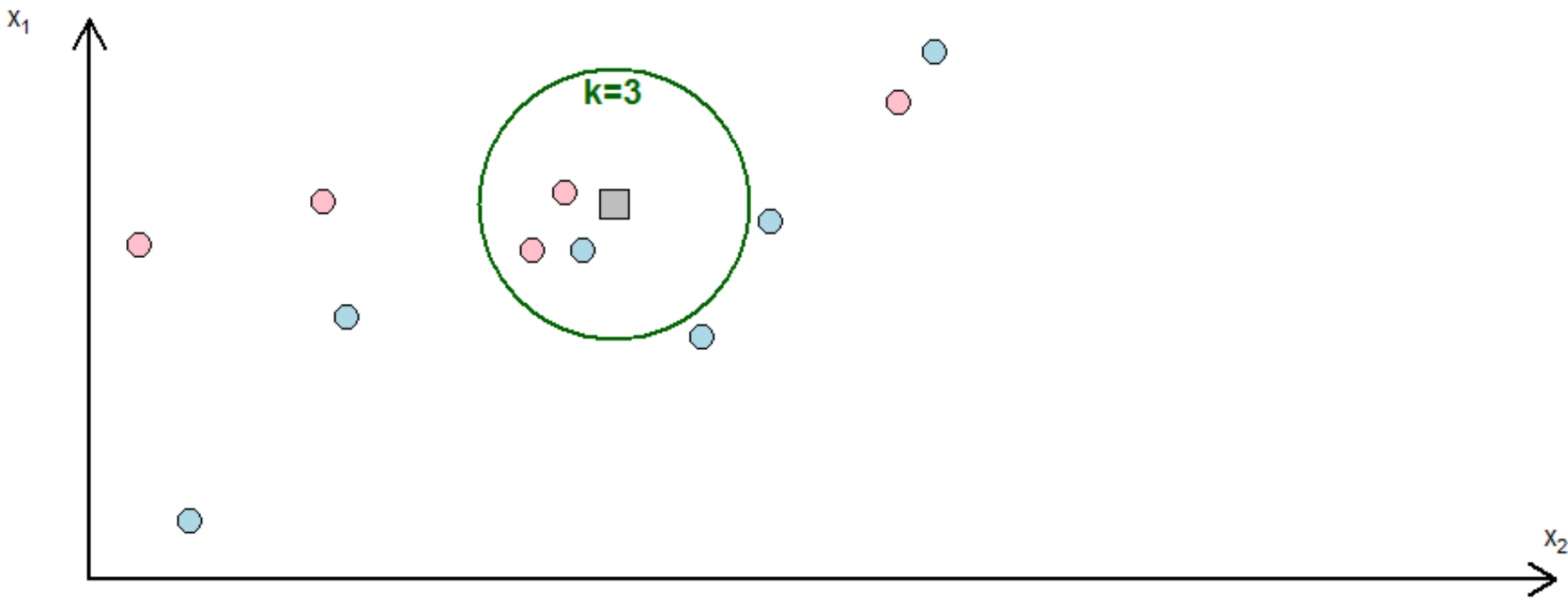

x_1
x_2

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)


x1
k=3
x2

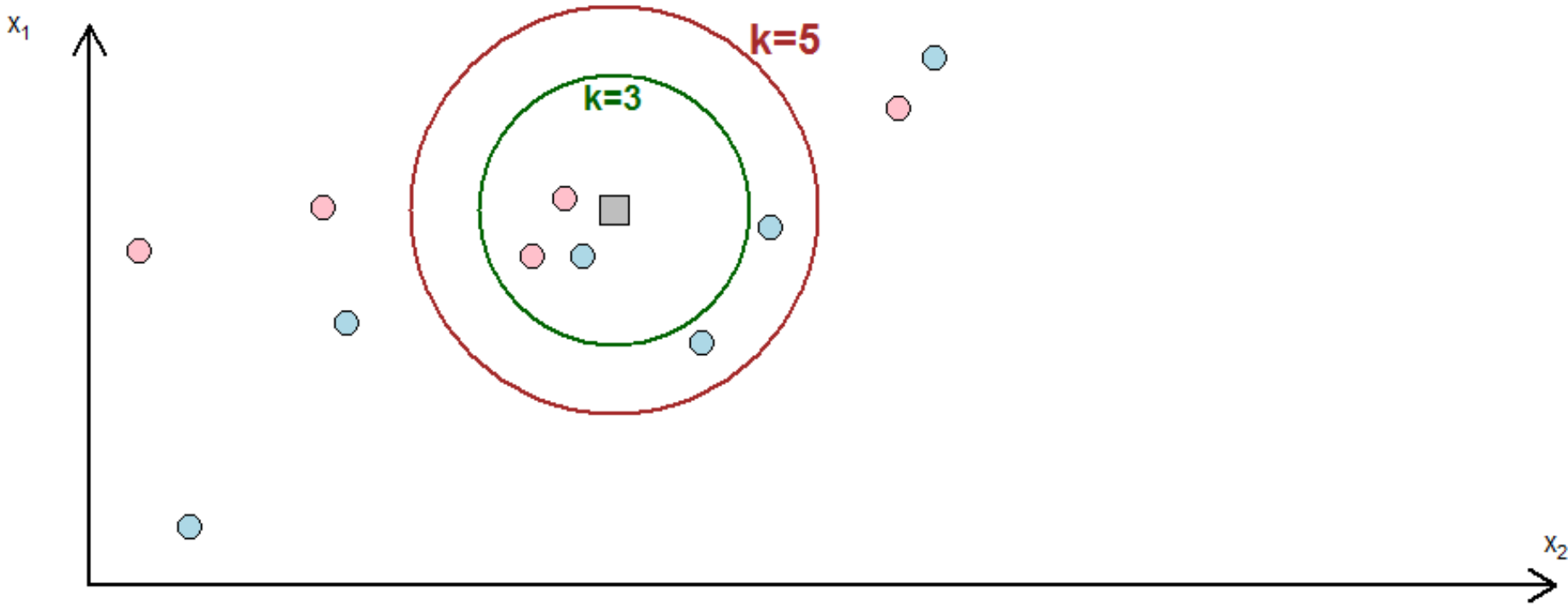
k-nearest neighbors

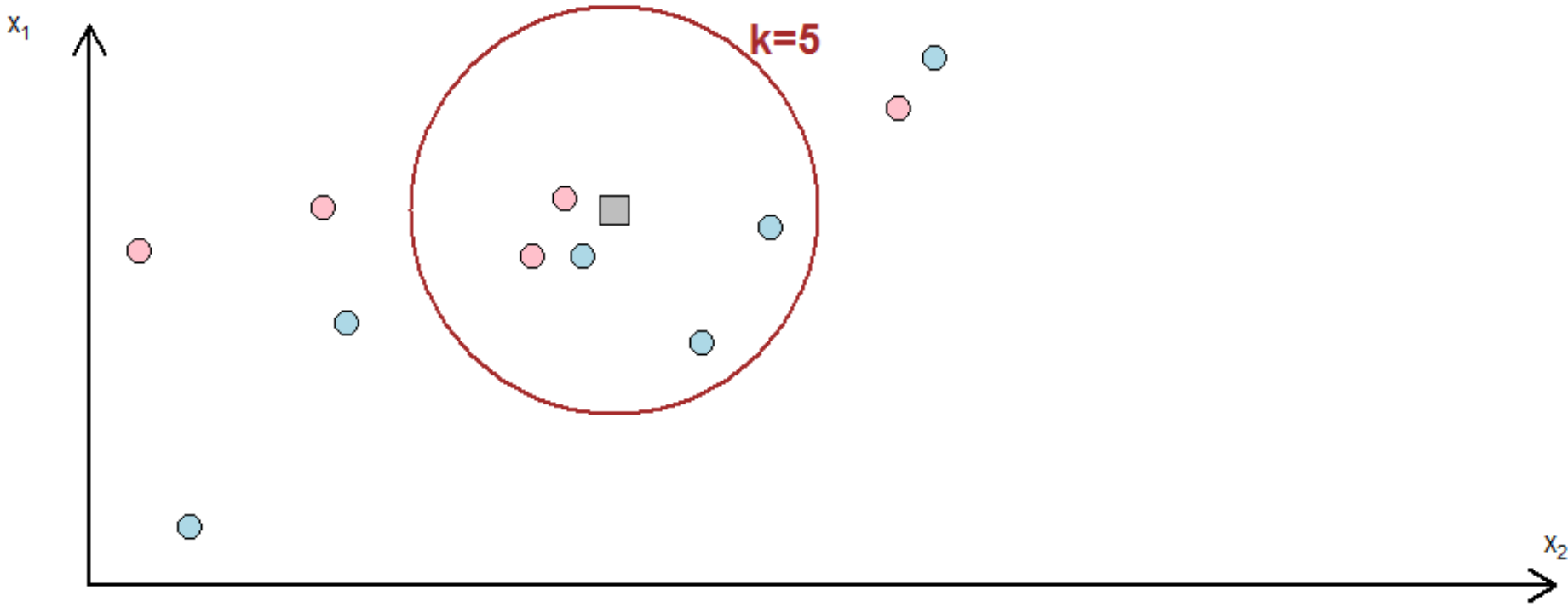
Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

e.g. $k = 5$: she is assigned to the Democrats (3 vs 2)


x1
k=5
k=3
x2


x1
k=5
x2

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

e.g. $k = 5$: she is assigned to the Democrats (3 vs 2)

Advantages of *kNN*

- It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

- Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model. Can still inspect accuracy though.

- Works with any types of features, though typically requires rescaling (normalizing) to ensure that one unit of one variable is not treated same as one unit of another (e.g. gender vs income: male is more different to female than \$10,000 is to \$10,001)

ID	Age	Income(rupees)
1	25	80,000
2	30	100,000
3	40	90,000
4	30	50,000
5	40	110,000

The Euclidean distance between observation 1 and 2 will be given as:

Euclidean Distance = [(100000–80000)^2 + (30–25)^2]^(1/2)

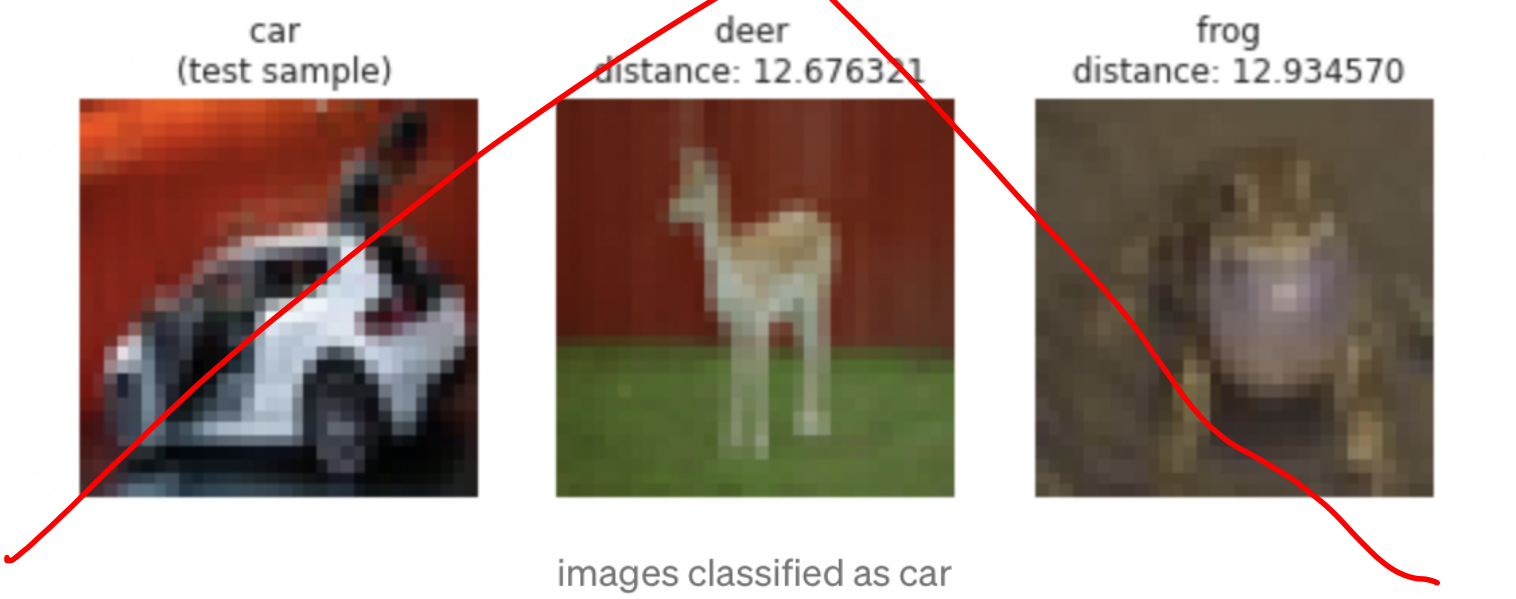
which will come out to be around ***20000.000625***. It can be noted here that

$$z = \frac{x - \mu}{\sigma}$$

Euclidean Distance = [(0.608+0.260)^2 + (-0.447+1.192)^2]^(1/2)

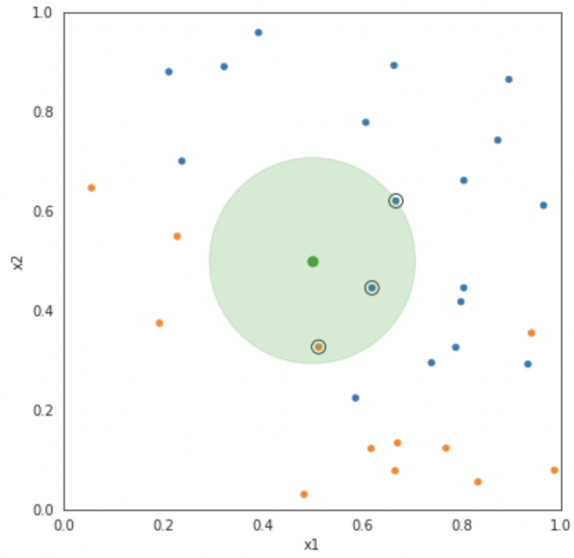
This time the distance is around ***1.1438***. We can clearly see that the distance


car
(test sample)
deer
distance: 12.676321
car
distance: 29.444304
images classified as car

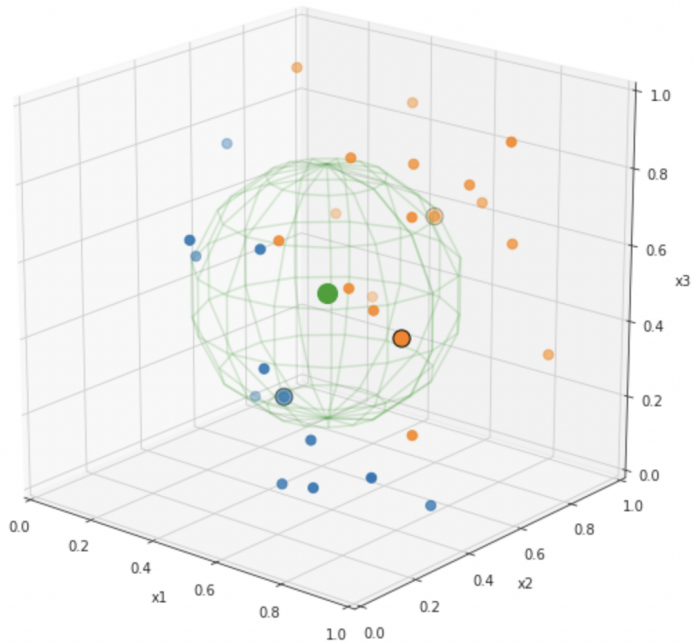

car
(test sample)
deer
distance: 12.676321
frog
distance: 12.934570
images classified as car

Disadvantages of *kNN*

- While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.
- Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.
- The curse of dimensionality. As number of features grows large, points look more similar than they really are (especially if features are irrelevant to actual decision). So, *kNN* often requires prior feature selection from theory, or via automated methods. May also have to tune distance function manually (e.g weight up 'safety rating' for baby products)

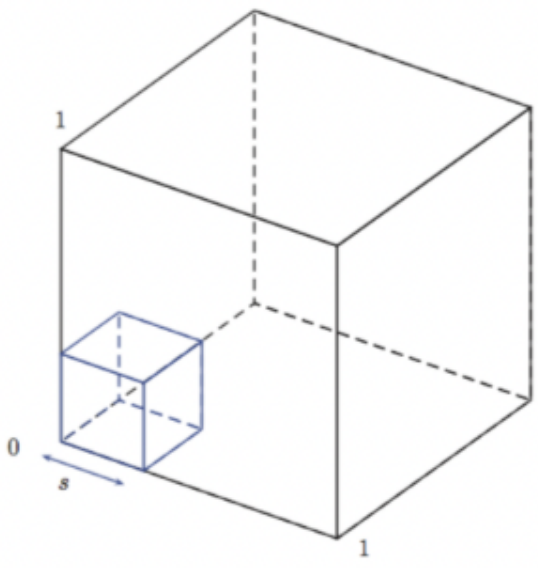


2D space radius search

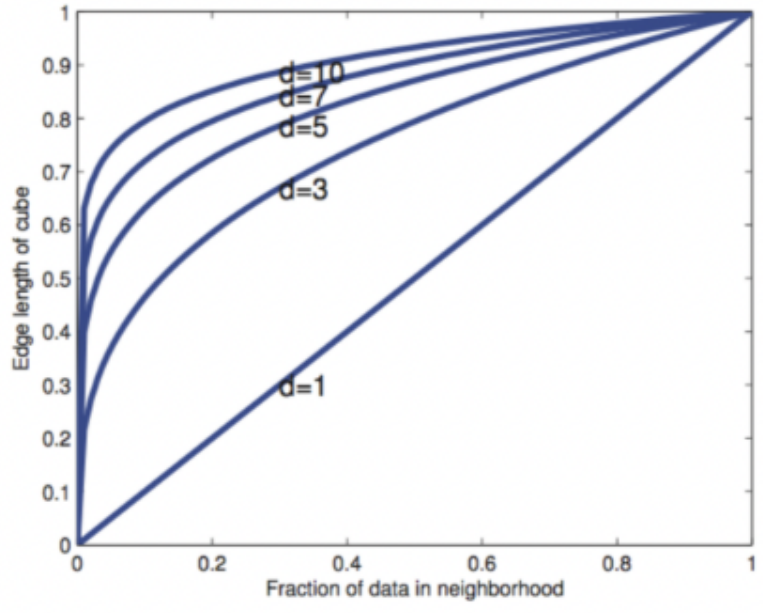


3D space radius search

$$e_D(f) = f^{\frac{1}{D}}$$



(a)



(b)

Trees and Forests

Tree-based Methods

- Here we describe *tree-based* methods for regression and classification.
- These involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision-tree* methods.

Pros and Cons

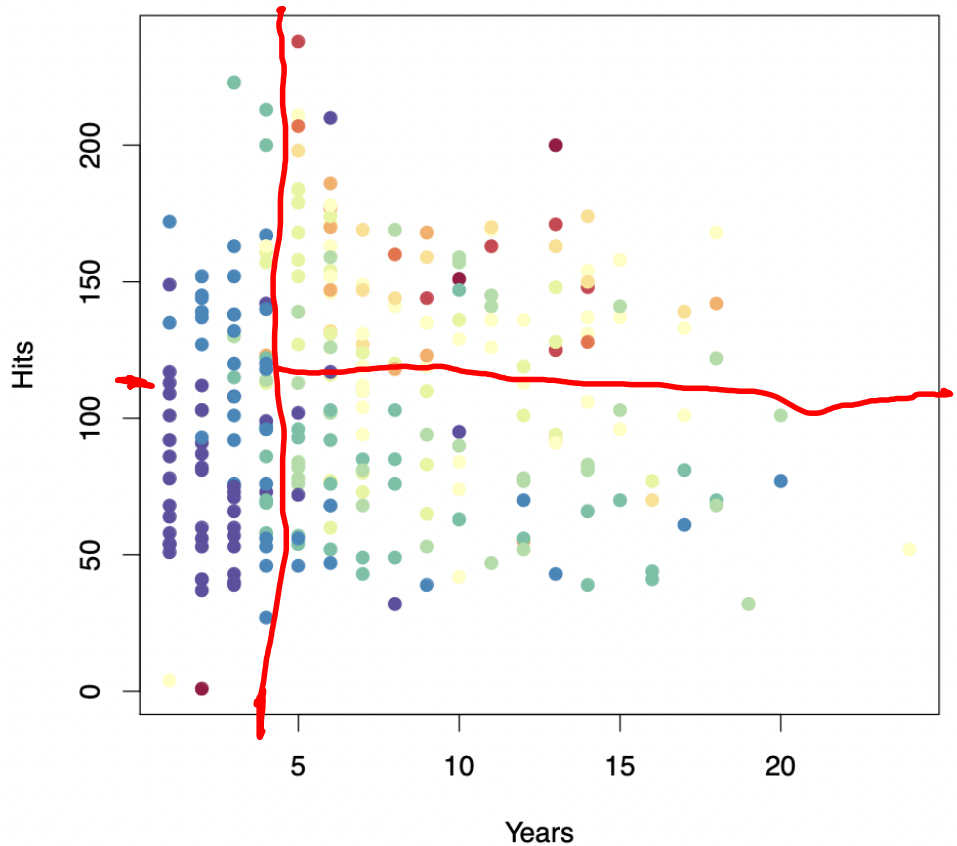
- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

The Basics of Decision Trees

- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

Baseball salary data: how would you stratify it?

Salary is color-coded from low (blue, green) to high (yellow,red)



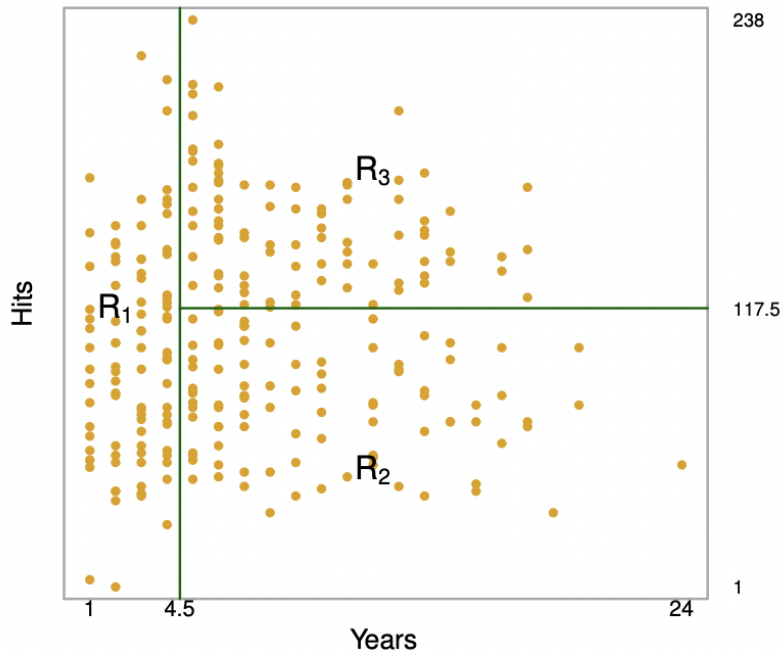
Decision tree for these data

Details of previous figure

- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to `Years`<4.5, and the right-hand branch corresponds to `Years`>=4.5.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Results

- Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X \mid \texttt{Years} < 4.5\}$, $R_2 = \{X \mid \texttt{Years} >= 4.5, \texttt{Hits} < 117.5\}$, and $R_3 = \{X \mid \texttt{Years} >= 4.5, \texttt{Hits} >= 117.5\}$.



Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text `Years`<4.5 and `Hits`<117.5.

Interpretation of Results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

More details of the tree-building process

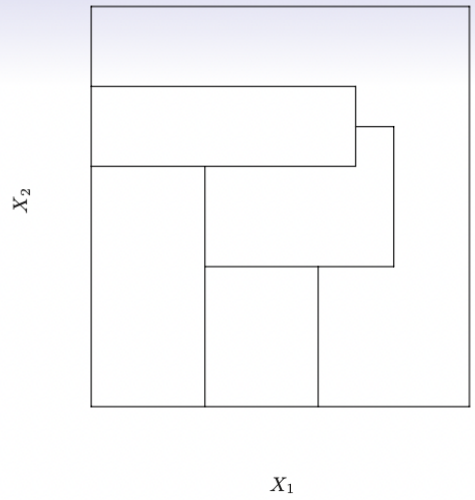
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as recursive binary splitting.
- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

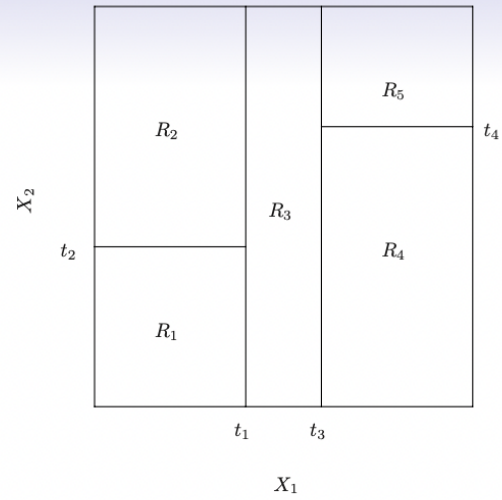
Details— Continued

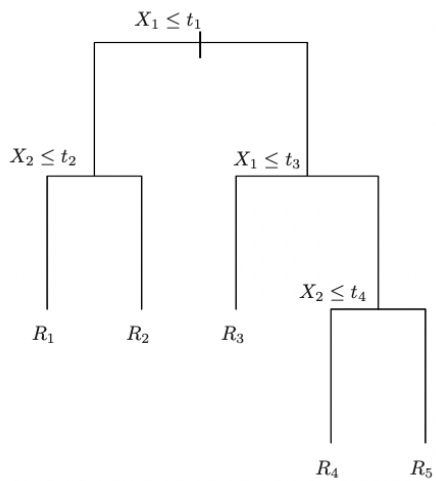
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

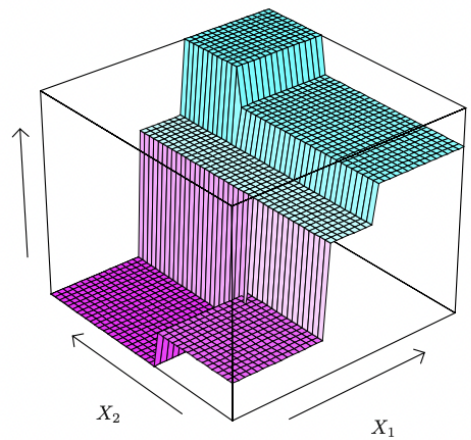
Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.


X_2
X_1


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_2
X_1


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

Details of previous figure

Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.

Top Right: The output of recursive binary splitting on a two-dimensional example.

Bottom Left: A tree corresponding to the partition in the top right panel.

Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted:* a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

Pruning a tree— continued

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\ x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: tree algorithm

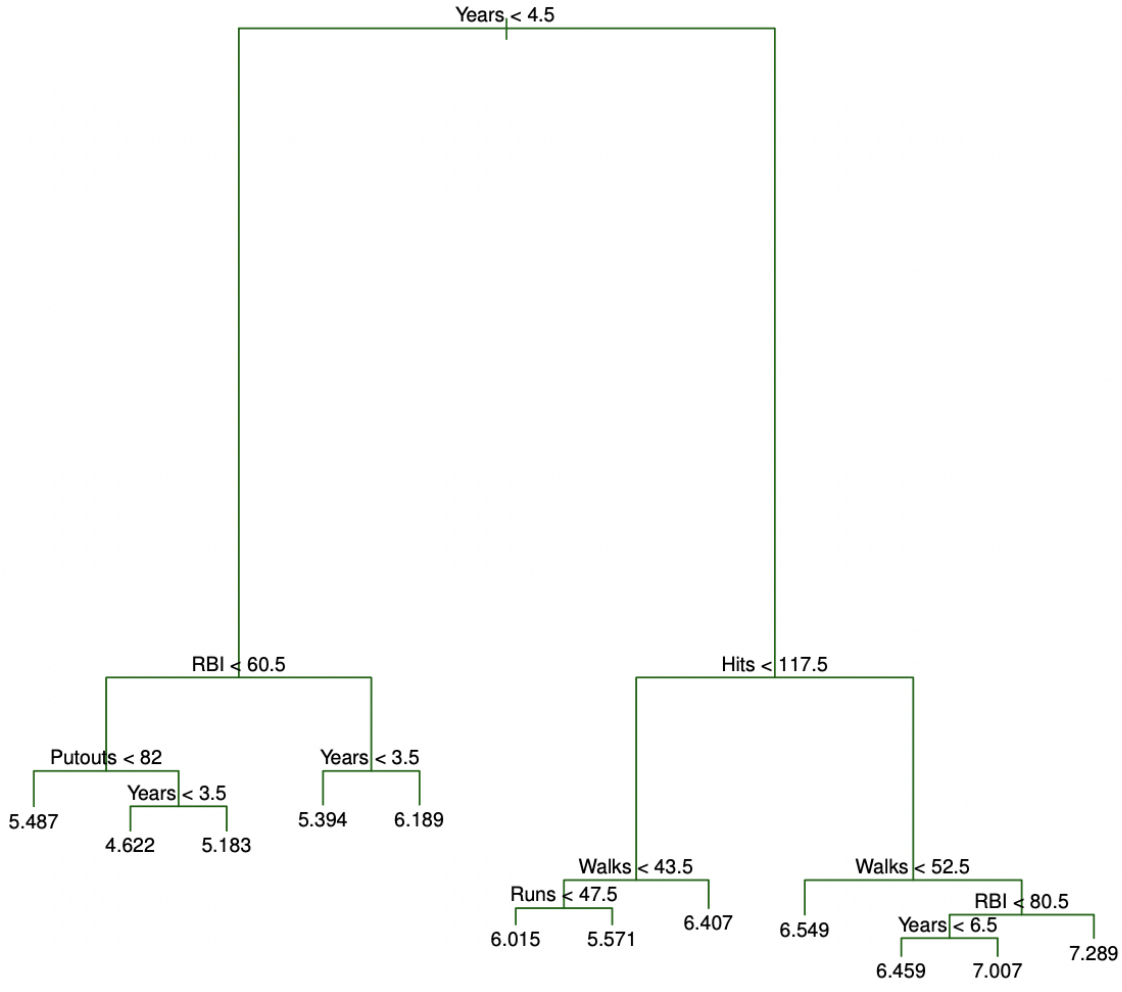
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

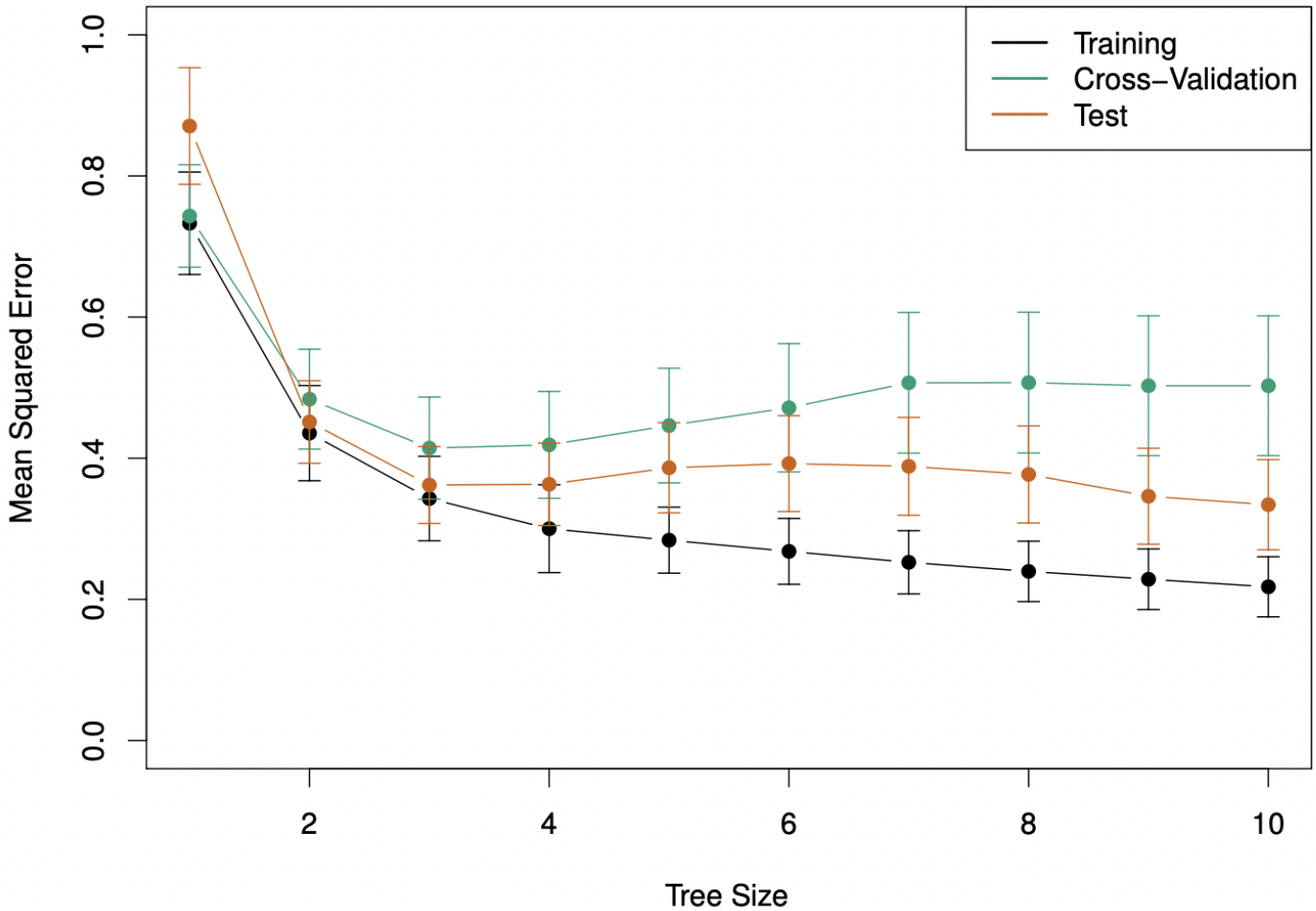
Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



Baseball example continued



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the *classification error rate*. this is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

- The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

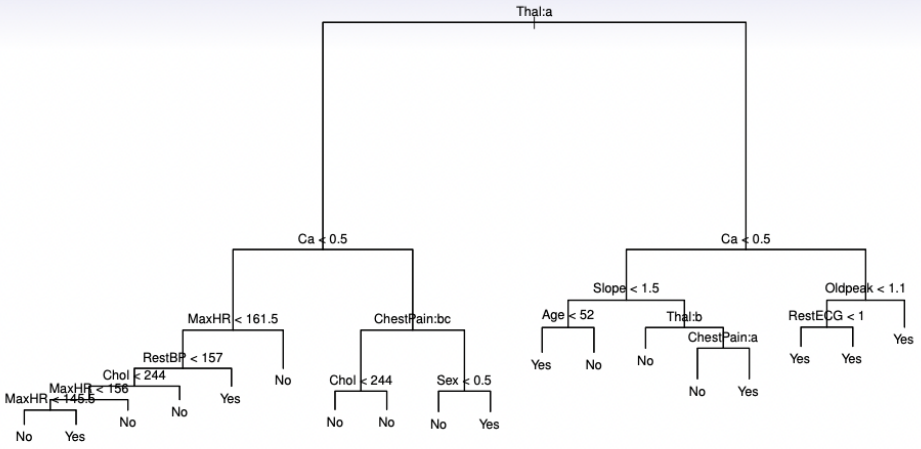
  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is *cross-entropy*, given by

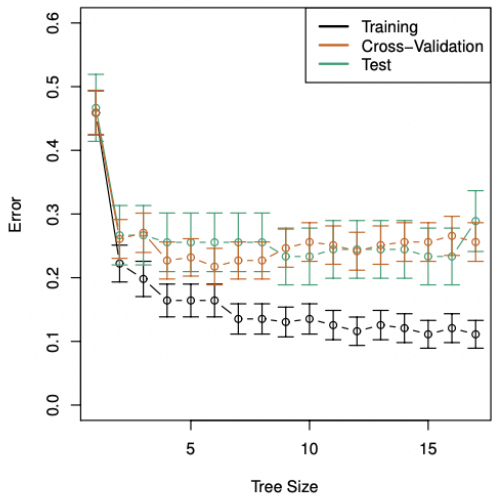
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

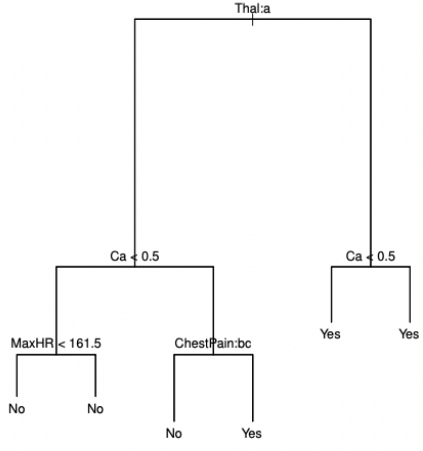
- It turns out that the Gini index and the cross-entropy are very similar numerically.

Example: heart data

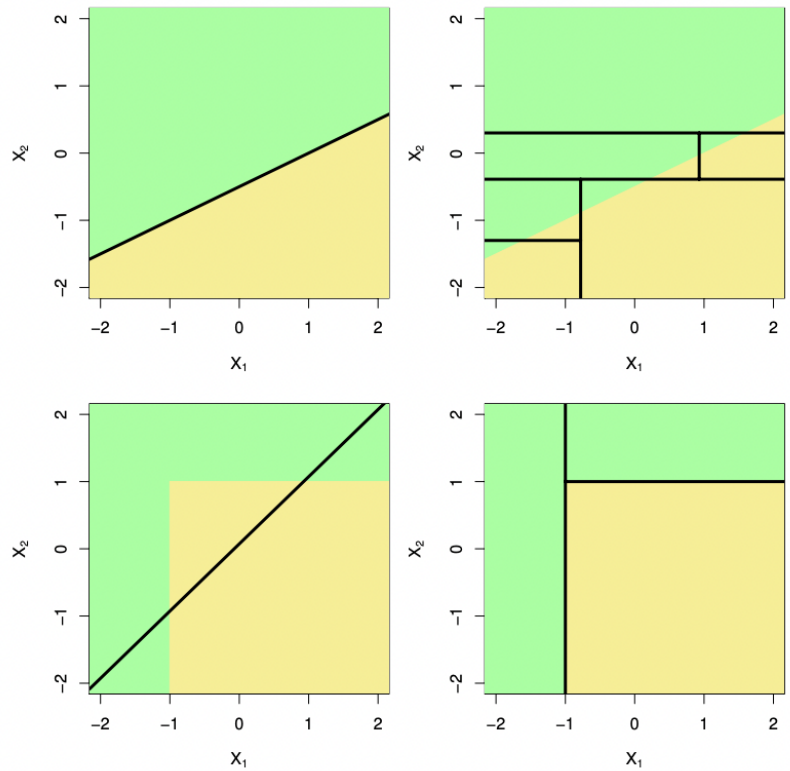
- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
Age < 52
Thal:b
RestECG < 1
ChestPain:a
RestBP < 157
Chol < 244
MaxHR < 156
MaxHR < 145.5
Chol < 244
Sex < 0.5
No
Yes


Training
Cross−Validation
Test
Error
Tree Size
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
No
Yes

Trees Versus Linear Models



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model

Advantages and Disadvantages of Trees

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ Trees can easily handle qualitative predictors without the need to create dummy variables.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Single-Tree Models

- Basic idea: divide covariate space into B (non-overlapping) regions that are fairly homogenous with respect to Y. Then make a prediction (c_b) for all observations in region R_B.

- a tree model for J covariates is a function

$$f(X_i) = T(X_i; \Theta) \equiv \sum_{b=1}^{B} c_b I(X_i \in R_b)$$

where Θ is the parameter vector, which contains tree depth (size), region definitions (how to split up the Xs), predicted values (c_b). And $I(\cdot)$ is the indicator function. Thus, a given model takes a value of X_i and gives back a $\hat{Y}$ for that.

Choosing Θ

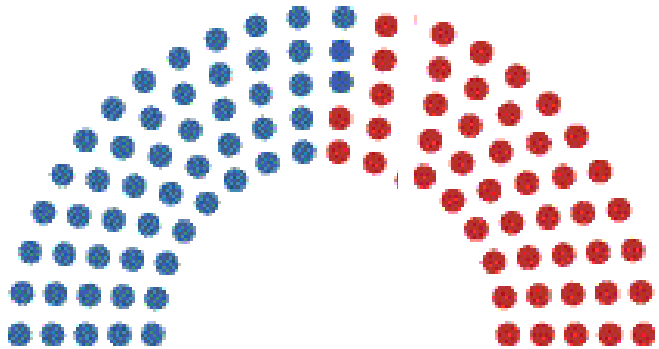
- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

  where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is: algorithm will fit a node to every observation. So, prune.
- Very simple, and fast. But sensitive to initial conditions, and can't assess uncertainty.

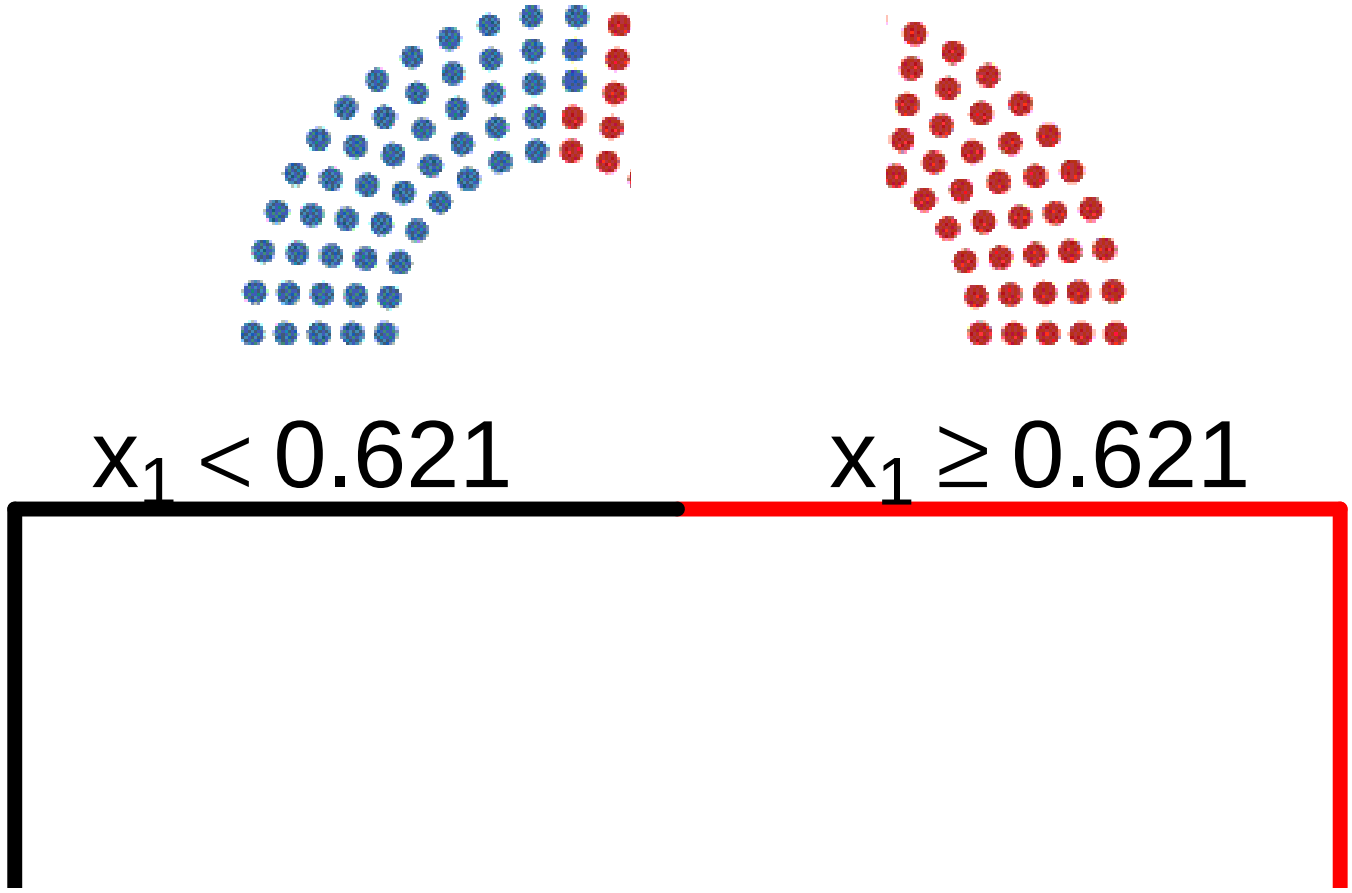
Partitioning the Senators With Trees



Idea our Senators are defined by their attributes. Suppose we (optimally) split ('partition') the Senators with respect to x_1, such that we form two subsets of our training data.

e.g suppose that Republicans generally use '`guns`' more than Democrats, such that grabbing all the observations for which $x_{guns} > 0.621$ captures, say, 80% of the Republicans in our data.

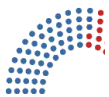
Tree, stage 1



Now, x_2 ...

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.

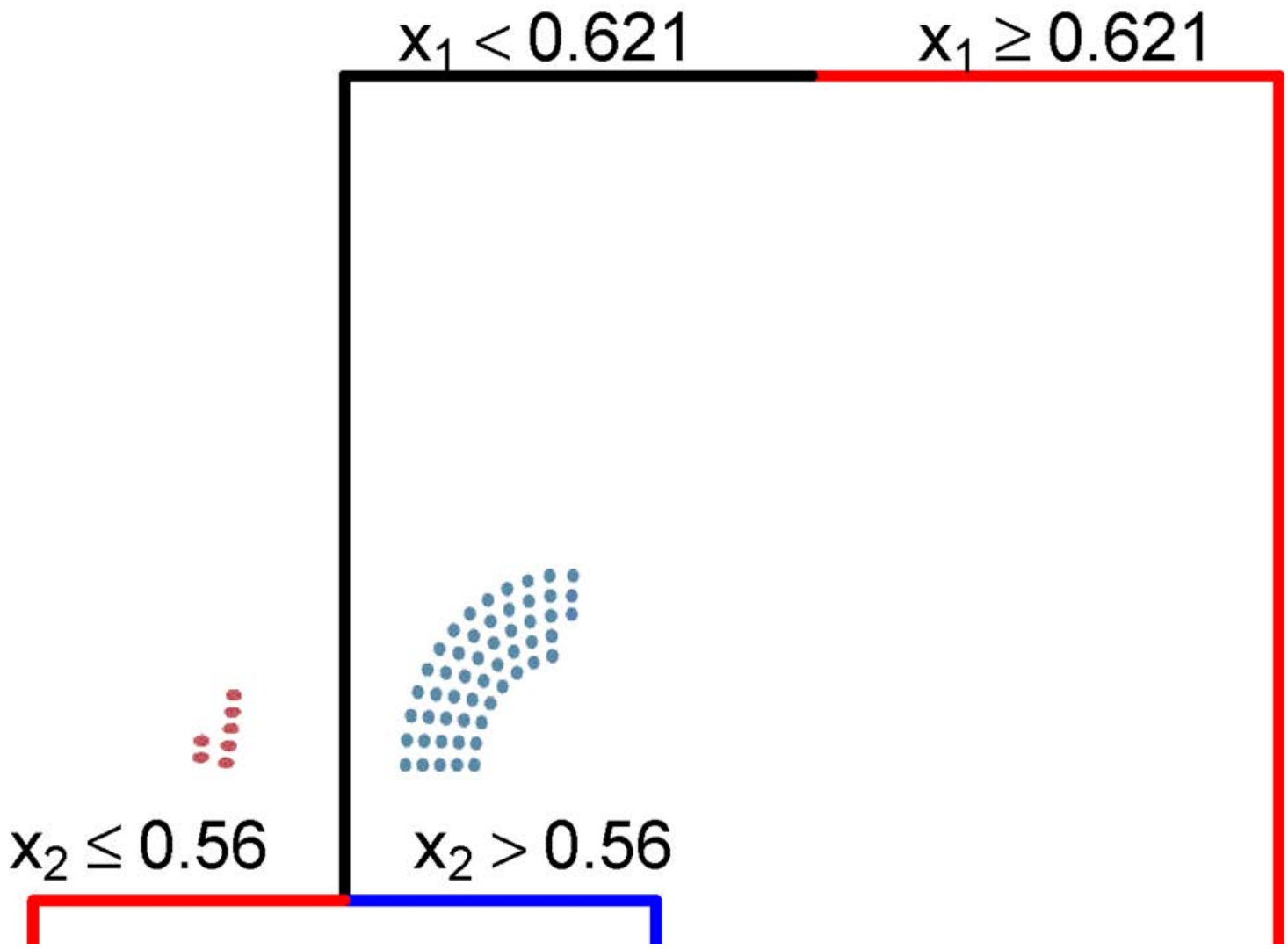


btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

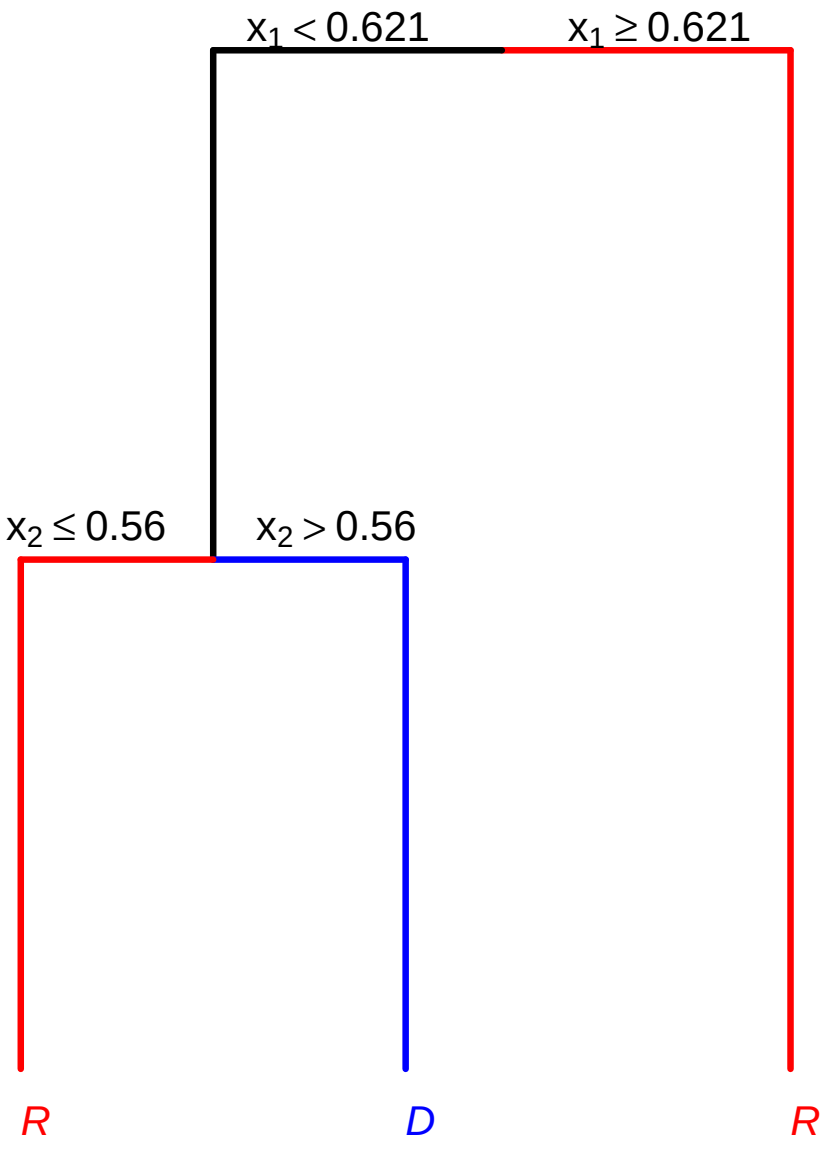
now suppose we take the mixed group remaining ('internal node') and split them based on x_2, which is their use of '`equality`'.

and it turns out that the (remaining) Republicans tend to use this less. So, when we partition according to, say, $x_2 \leq 0.56$ this enables us to perfectly divide this remaining subset into Democrats and Republicans.

Tree, stage 2



Complete Tree



Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

→ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x: typically based on how homogenous the resulting subset of the data is

e.g. 'Gini impurity' and 'Variance Reduction'

\+ Trees are easy to interpret, and we can report relative variable importance statistics.

But...

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

→ related to problems of overfitting in the training data.

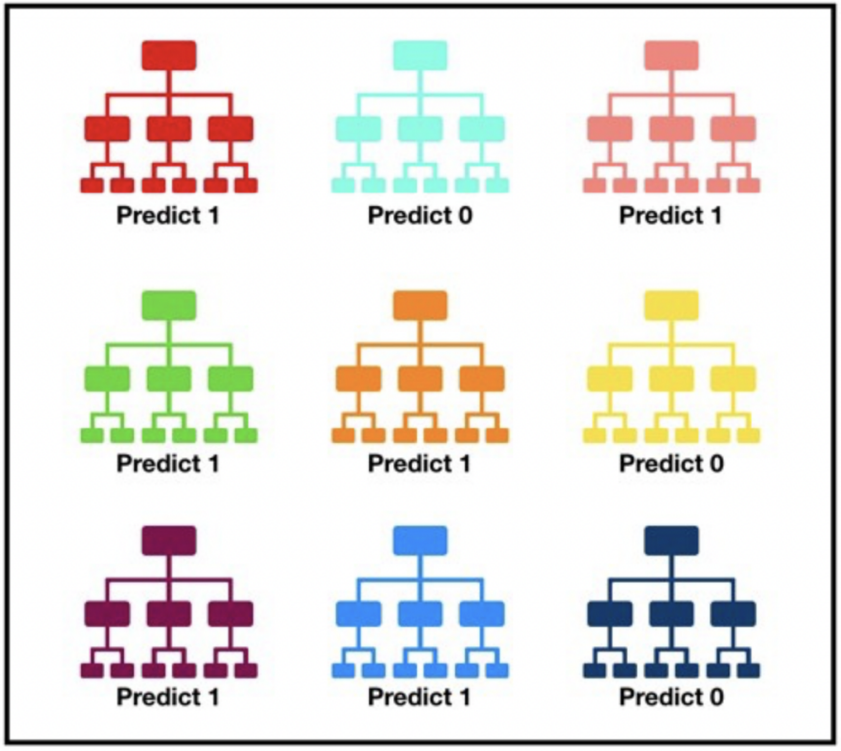
So effort is made to prune the trees back—remove less helpful branches.

Or can construct many trees (from slightly different samples of the data) and average over them: known as bagging ('bootstrap aggregating').

→ random forests combines *many* trees (forest) and at each split a *random sample* of features is considered (rather than all features).

Random Forest Classifier

- Generate many trees and then let them vote.



Tally: Six 1s and Three 0s

Prediction: 1

Assumptions

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.

2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

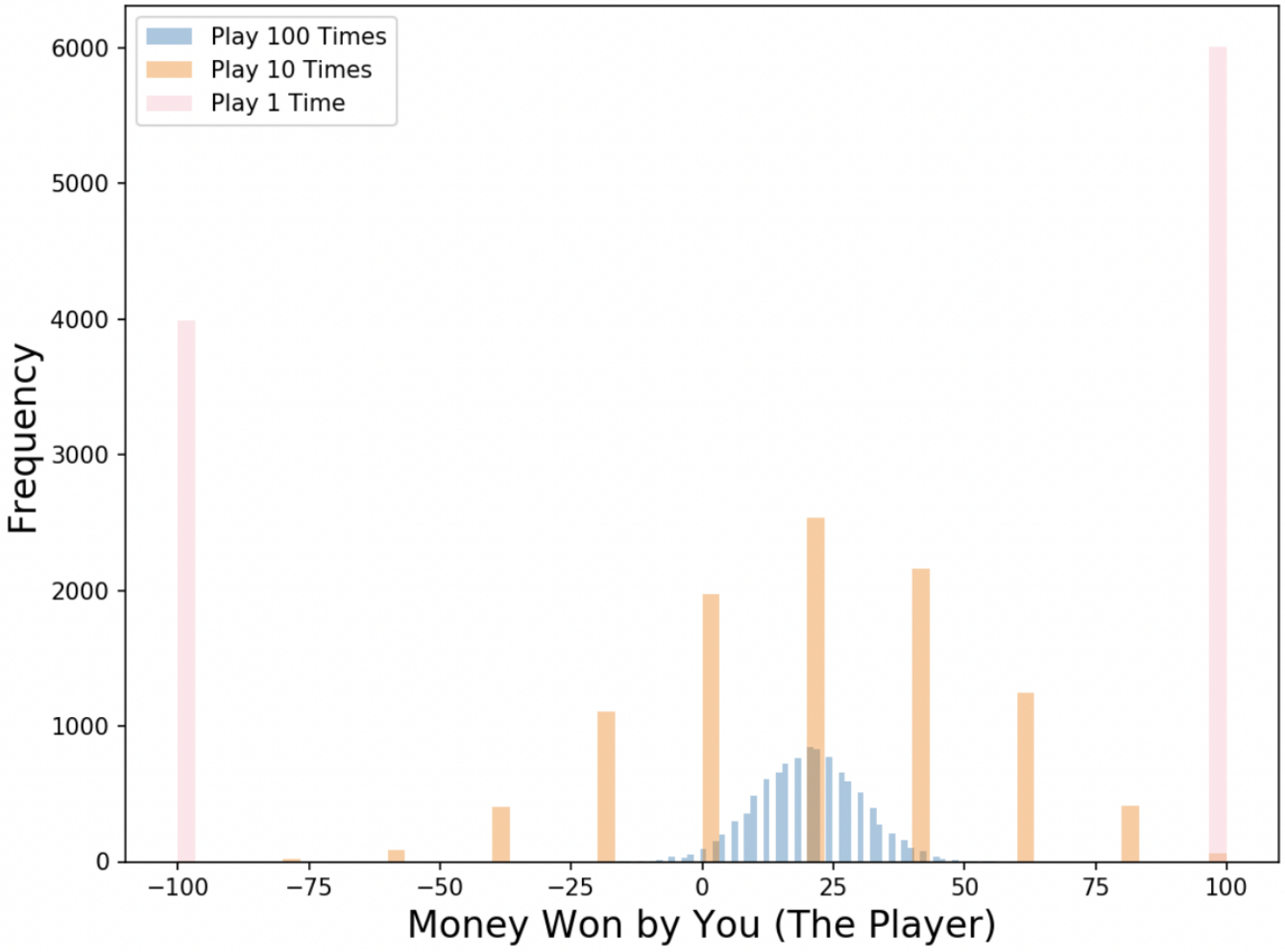
Uncorrelated Outcomes and Random Sampling are Good Things. . .

- I use a uniformly distributed random number generator to produce a number.
- If the number I generate is greater than or equal to 40, you win (so you have a 60% chance of victory) and I pay you some money. If it is below 40, I win and you pay me the same amount.
- Now I offer you the the following choices. We can either:

1. **Game 1** — play 100 times, betting $1 each time.
2. **Game 2**— play 10 times, betting $10 each time.
3. **Game 3**— play one time, betting $100.

Which would you pick? The expected value of each game is the same:

> *Expected Value Game 1 = (0.60*1 + 0.40*-1)*100 = 20*
>
> *Expected Value Game 2= (0.60*10 + 0.40*-10)*10 = 20*
>
> *Expected Value Game 3= 0.60*100 + 0.40*-100 = 20*



Outcome Distribution of 10,000 Simulations for each Game

Tree Bagging and Random Forests

- Bagging—bootstrap aggregating—rests on observation that trees fit to different subsets of data (observations) will give different predictions.

- If those trees are independent, then we get a low bias, low variance estimate of the true response. So, grow the trees (fully) and just take an average over the M samples:

$$\widehat{f_{\text{bag}}}(X_i) = \frac{1}{M} \sum_{m=1}^{M} T_m(X_i; \hat{\Theta}_m)$$

- In practice, this can result in quite correlated trees, so random forests does splits of random subset of variables at each node in a tree.

Boosting

- Build trees sequentially, with focus on observations that current ensemble (so far) struggles with.