Classification methods

COS 424/524, SML 302: Fundamentals of Machine Learning

Professor Engelhardt

COS424/524, SML 302

Lecture 6

Classification methods

The naïve Bayes (NB) classifier we learned about in a previous lecture is the canonical example of a *generative classifier*.

Today, we will learn about four other classifiers:

- K nearest neighbors
- Support vector machines (SVMs)
- Decision trees
- Random forests

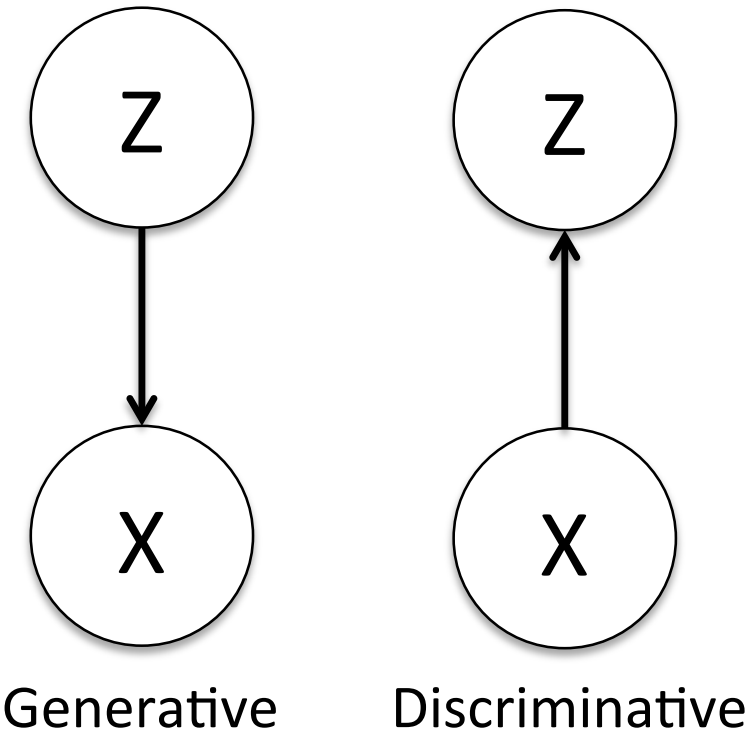
We will delay discussion of *logistic regression* until we cover *regression*.

Comparing classifiers

- *Model based?* Can a probabilistic model be written out?
- *Type?* Is the classifier generative, discriminative, or cluster-based?
- *Kernelizeable?* Can the kernel trick be exploited in this classifier?
- *Additive?* Is the prediction an additive function of the features?
- *Hyperparameters?* Are there hyperparameters I need to set?
- *Multiclass?* Can classifier be adapted for multiclass classification?
- *Interpretable?* Can I find predictive features from a trained classifier?
- *Missing data?* Can missing data be accommodated easily?
- *Training?* Computational complexity for training?
- *Test?* Computational complexity for testing?

Generative versus discriminative classifiers

Let random variable x be an observed feature vector, z its class label.



Generative classifier

A *generative classifier* represents the factorization:

$$p(x, z) = p(z)\, p(x|z)$$

Classification requires Bayes rule:

$$p(z|x) \propto p(z)\, p(x|z).$$

Discriminative classifier

A *discriminative classifier* represents the factorization:

$$p(x, z) = p(z|x)\, p(x)$$

Classification does not require Bayes rule: $p(z|x)$.

Generative versus discriminative classifiers

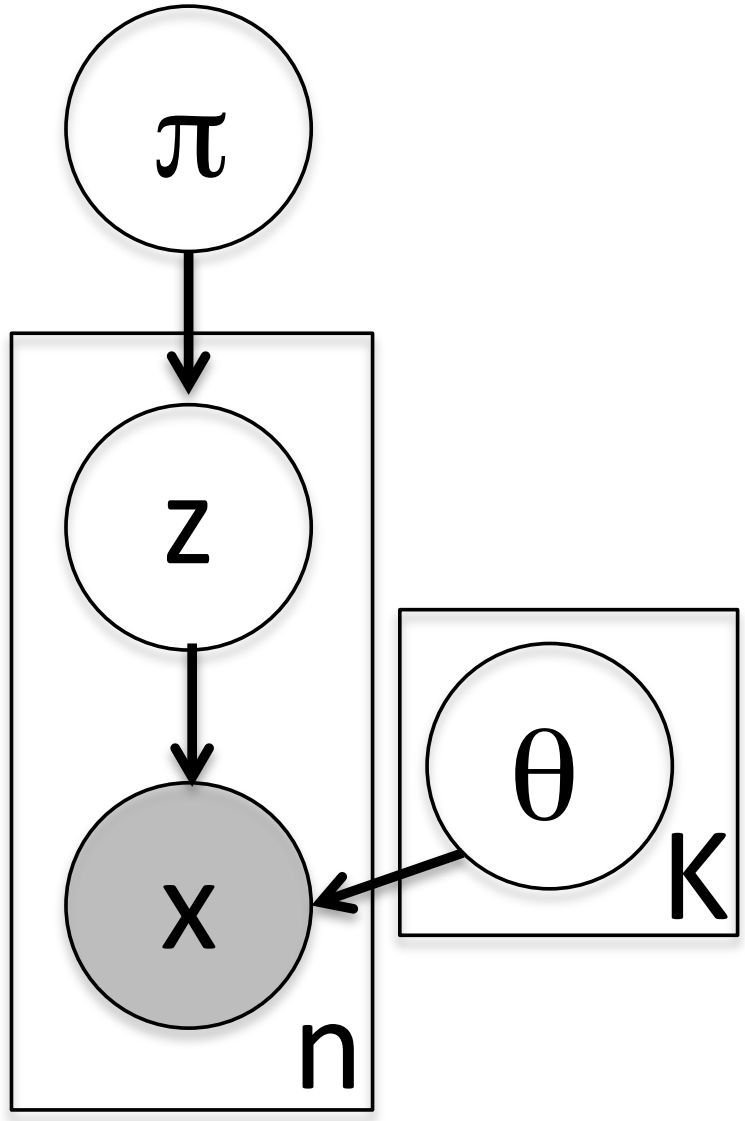
Generative classifiers	Discriminative classifiers
$p(x, z) = p(z)\, p(x\|z)$	$p(x, z) = p(z\|x)\, p(x)$
model *feature distribution* of each class	model distribution of classes in feature space
$O(K)$ p-dim distributions to estimate	$O(p)$ K-dim distributions to estimate
good performance with small training sets	poor performance with small training sets
accuracy stalls as training set grows	accuracy improves for large training sets
robust to missing data	missing data strategies need to be explicit

See *[Ng & Jordan 2001]* for more information.

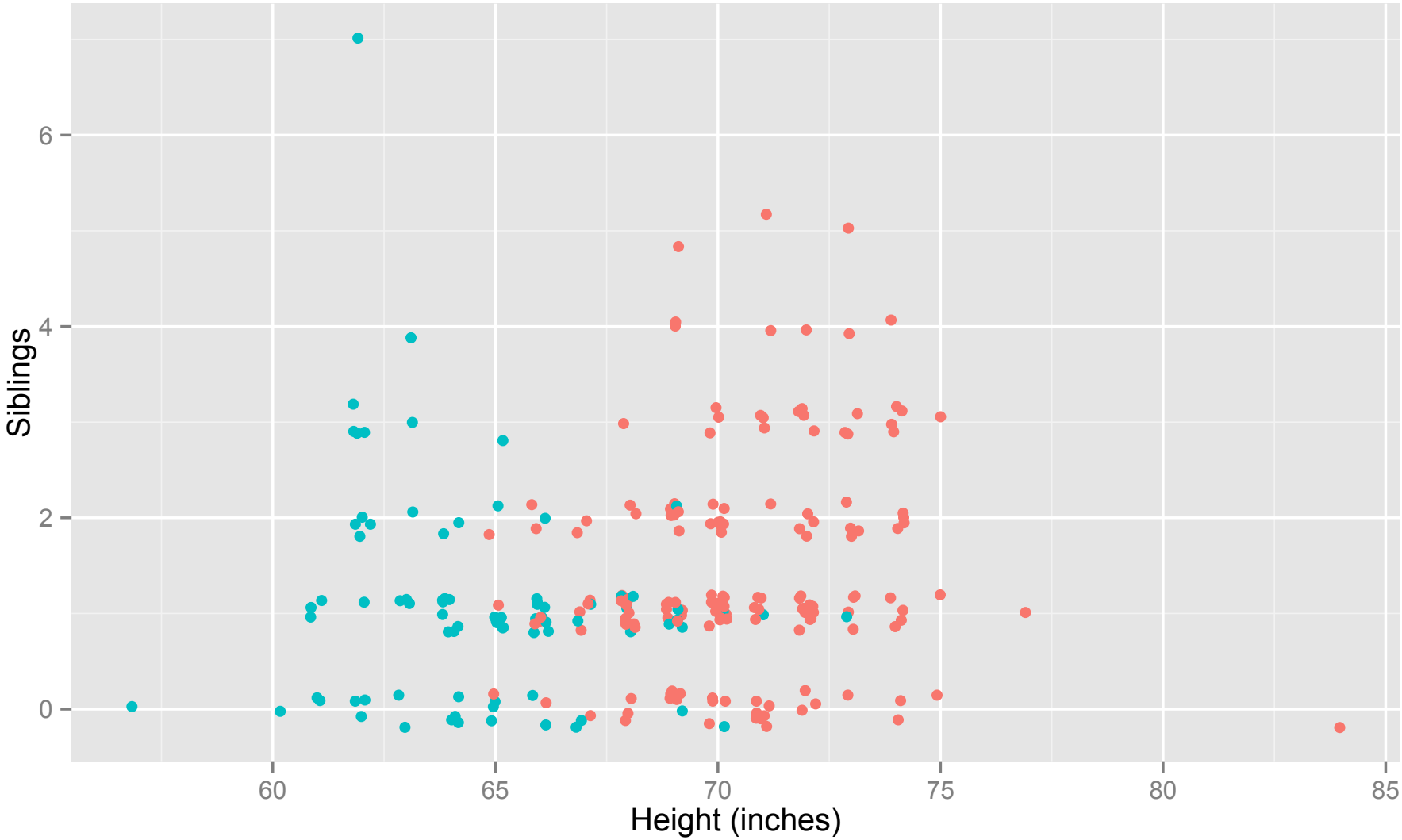
Recall: Naive Bayes classifier

K classes, n samples, p features:

- *Model based?* Yes
- *Classifier type?* Generative
- *Kernelizeable?* No
- *Additive?* Yes
- *Hyperparameters?* No
- *Multiclass?* Yes
- *Interpretable?* Not that easily
- *Missing data?* Yes
- *Training?* $O(np)$
- *Test?* $O(pK)$

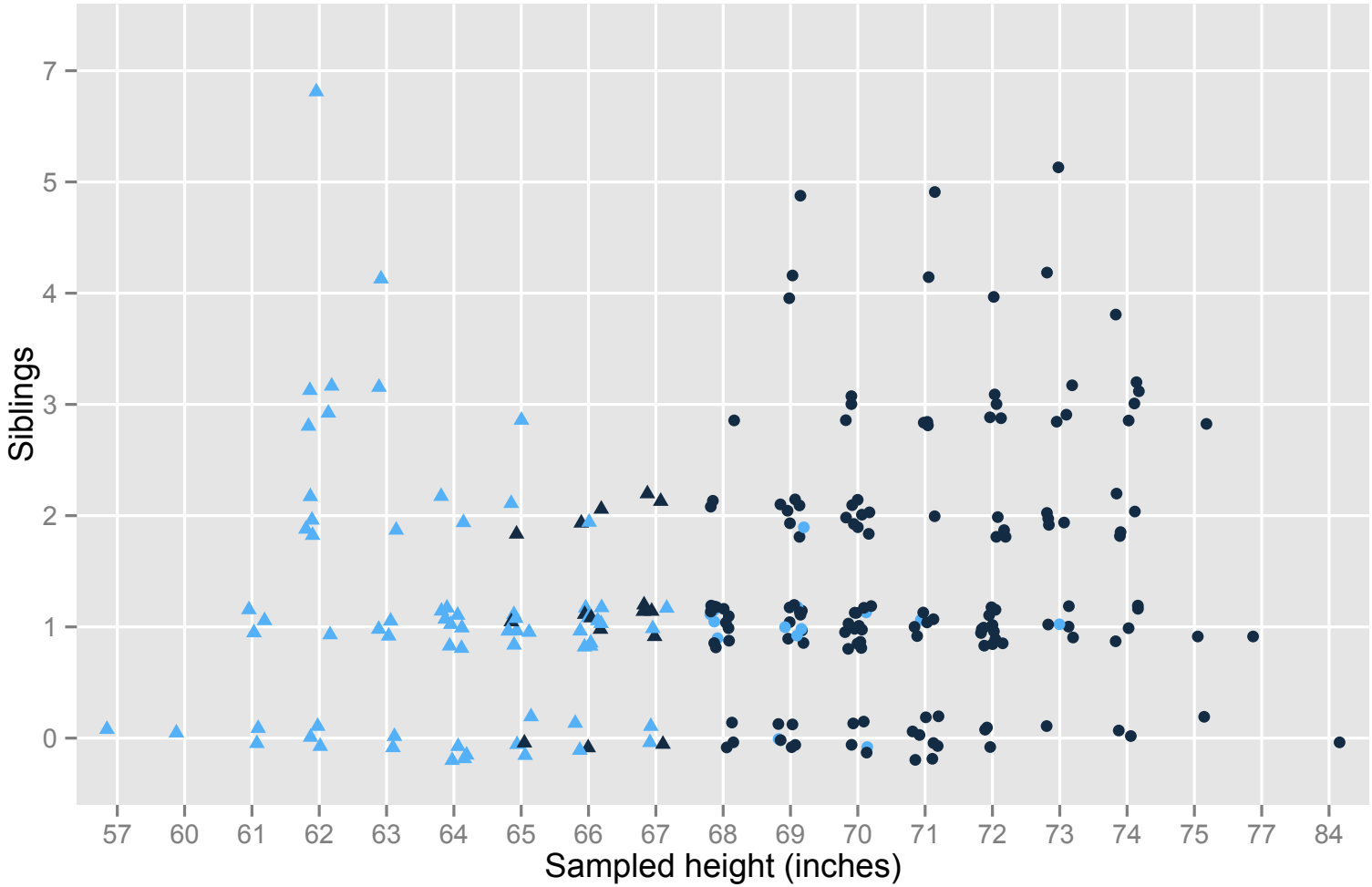


Example: predicting gender from height, siblings



Naïve Bayes: example

Using leave-one-out cross validation:



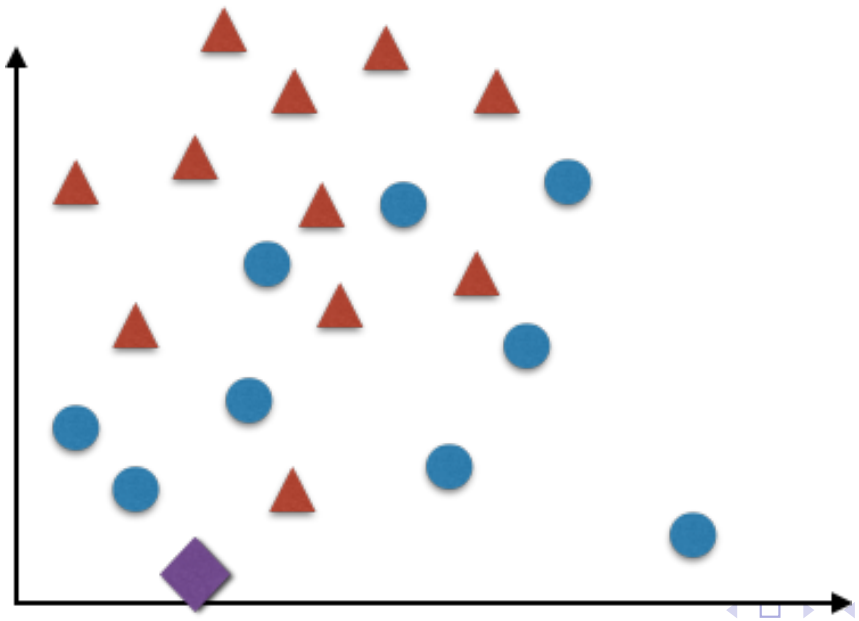
$$\text{accuracy} = 0.88$$

K nearest neighbor classification

K nearest neighbors (KNN) is a *nonparametric method* for classification.

Classify unlabeled sample x^* given *training data* $\mathcal{D} = \{(x_1, z_1), \ldots, (x_n, z_n)\}$, and *similarity function* $f(x, x^*)$:

- Find the most similar K samples to x^* from $\mathcal{D}$ according to $f(\cdot, \cdot)$: these are the K *nearest neighbors* to x^*
- Set $\hat{z}^*$ to the most common class label from the K nearest neighbors



K nearest neighbors: example

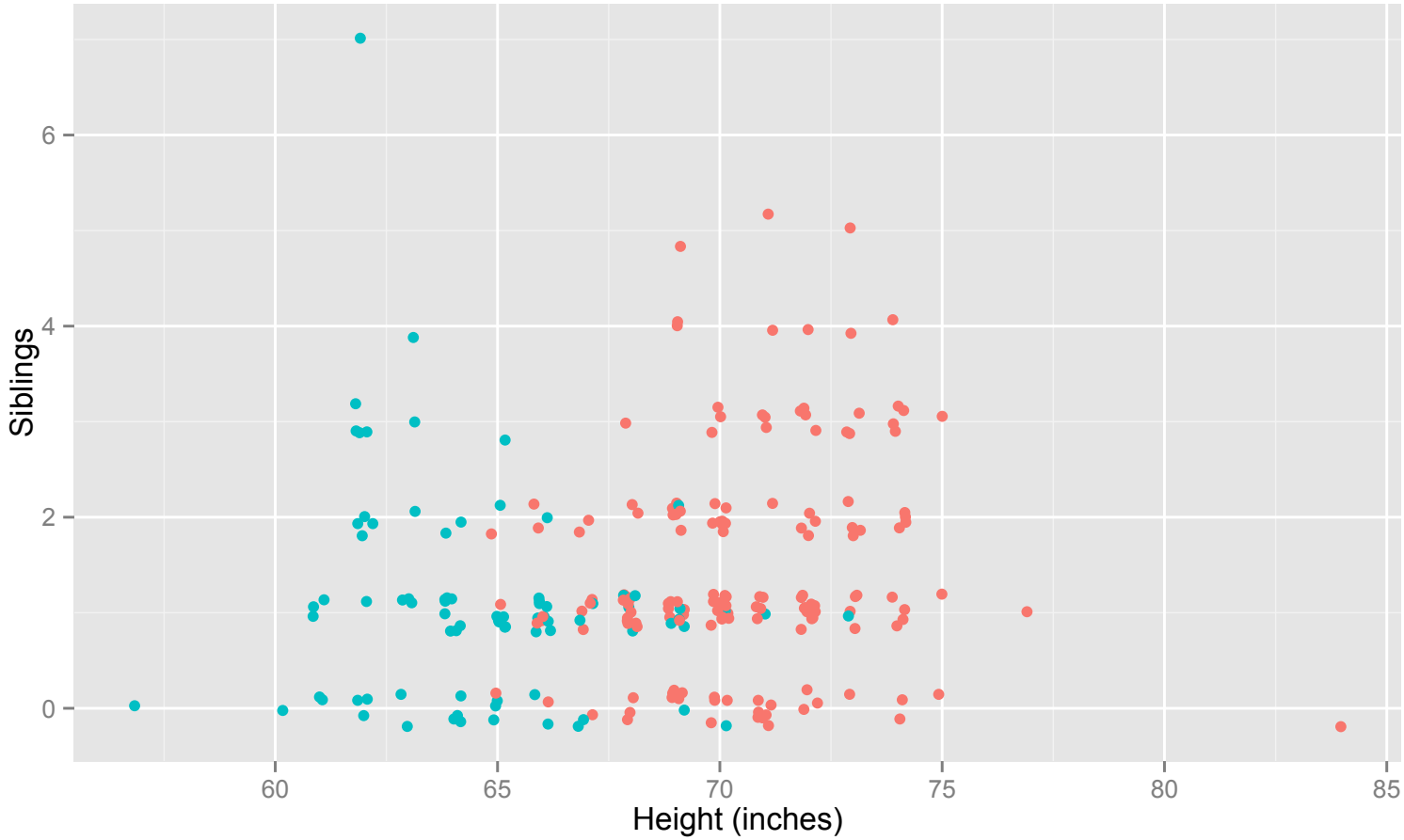
Classify sample as *male* or *female* based on height and number of siblings.

Let the training set be our class survey data $\mathcal{D}$; let the distance metric be *Euclidean distance*:

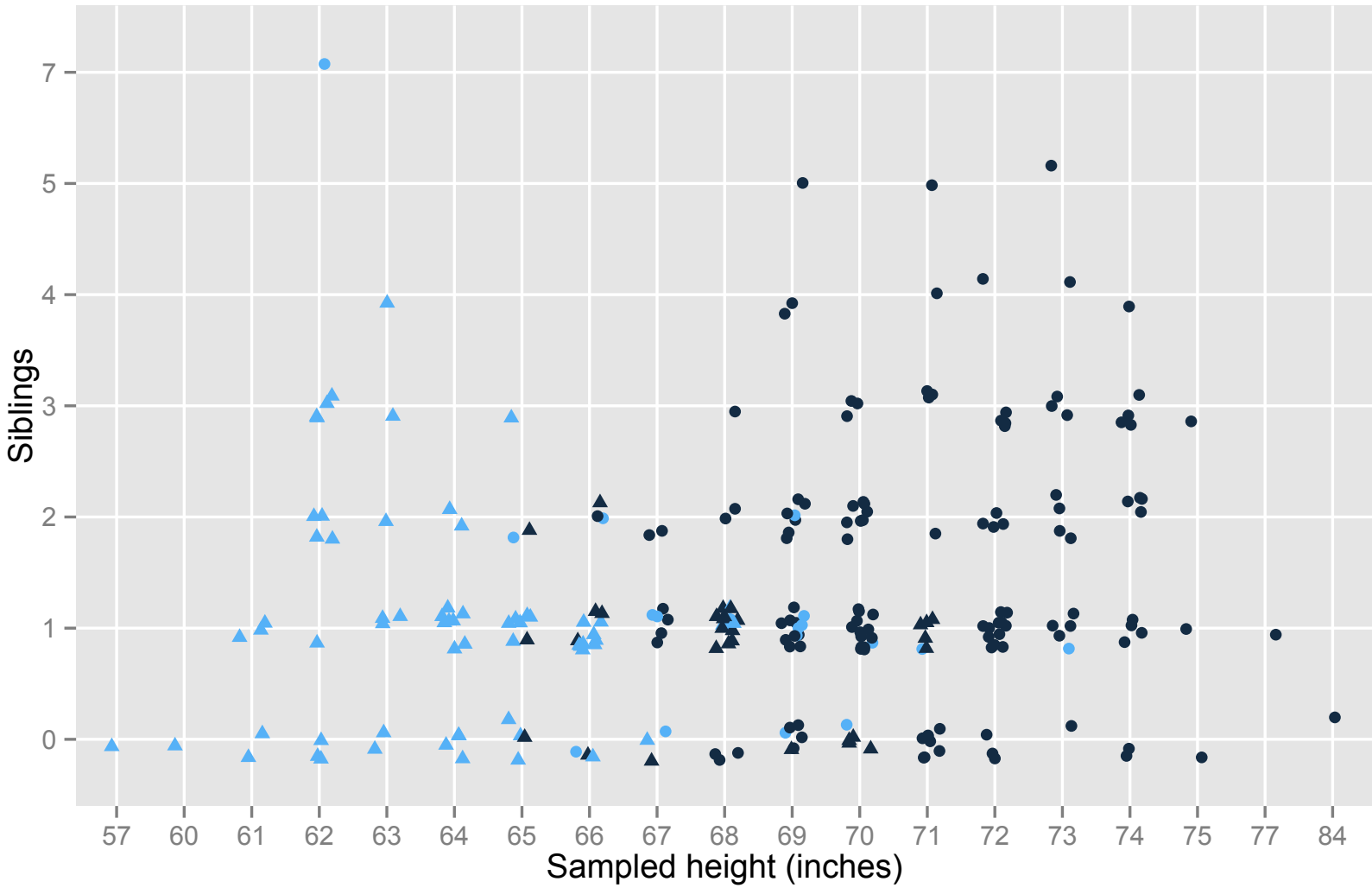
$$f(x, x') = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}.$$

We can start by choosing number of nearest neighbors $K = 1$.

Example: predicting gender from height, siblings



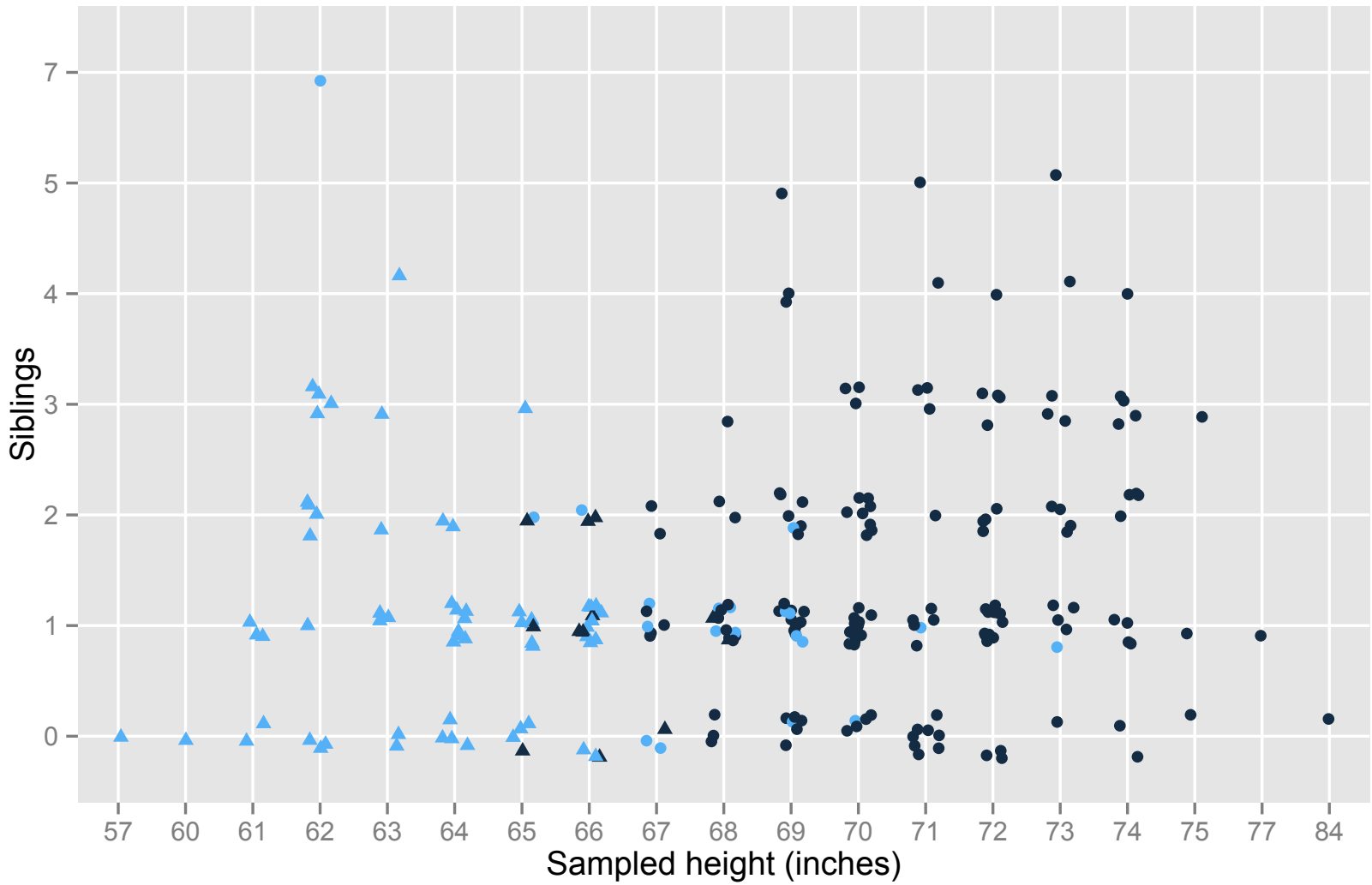
K nearest neighbors: example with K=1



accuracy = 0.80

How should you choose the parameter K?

K nearest neighbors: example with K=3



accuracy = 0.86

Kernelized classifiers

In the last lecture, we used kernels to project our data to a high dimensional *feature space*. We can use a simple classifier in this space.

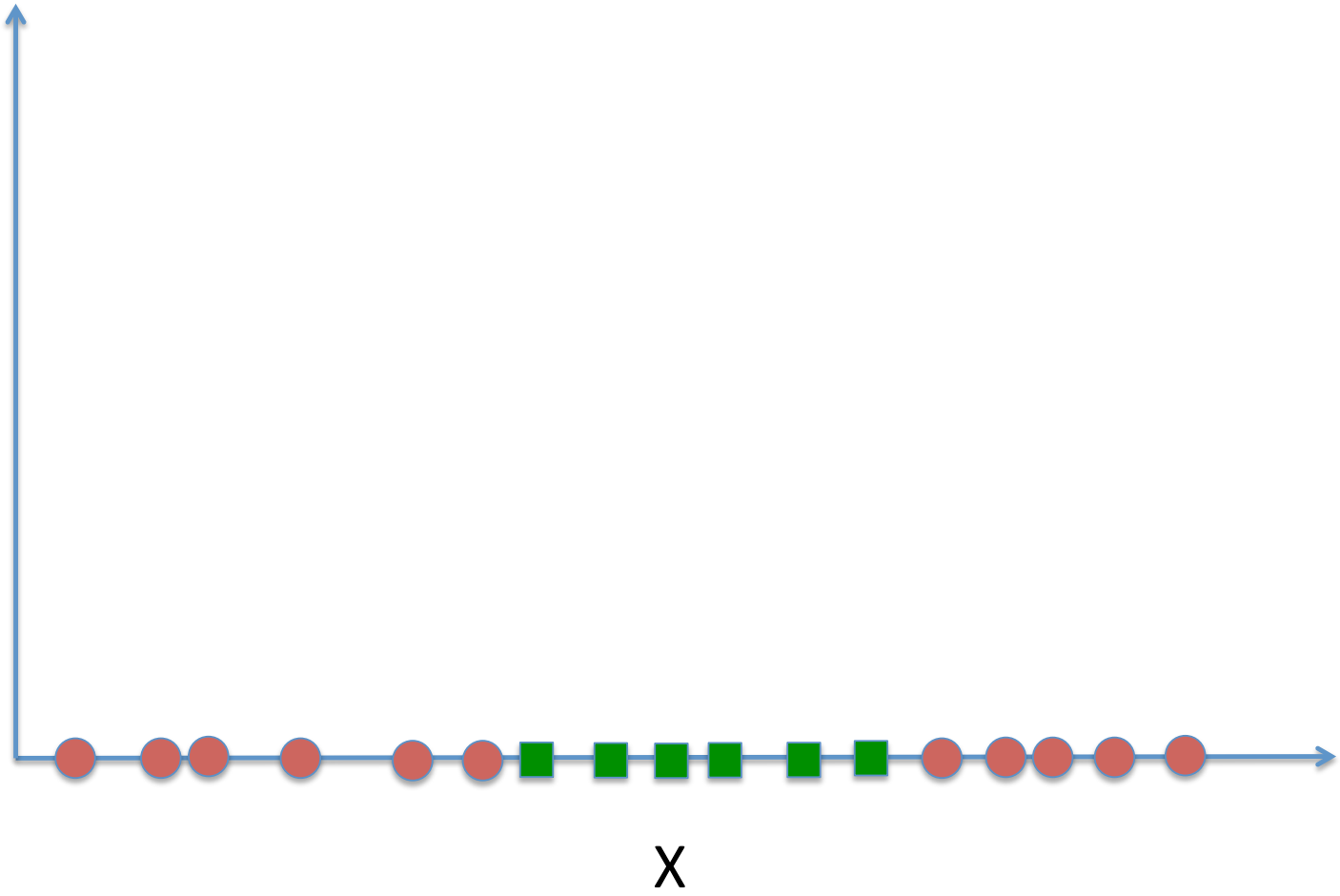
Consider the relationship between a high-dimension linear classifier and a two-dimension linear classifier:

- We first map the input two-dimensional space $\mathbf{x}$ up to a high-dimensional feature space;
- we fit a hyperplane in feature space (linear classifier);
- the hyperplane projected back onto the two-dimensional input space is not necessarily linear

Ex: polynomial kernel

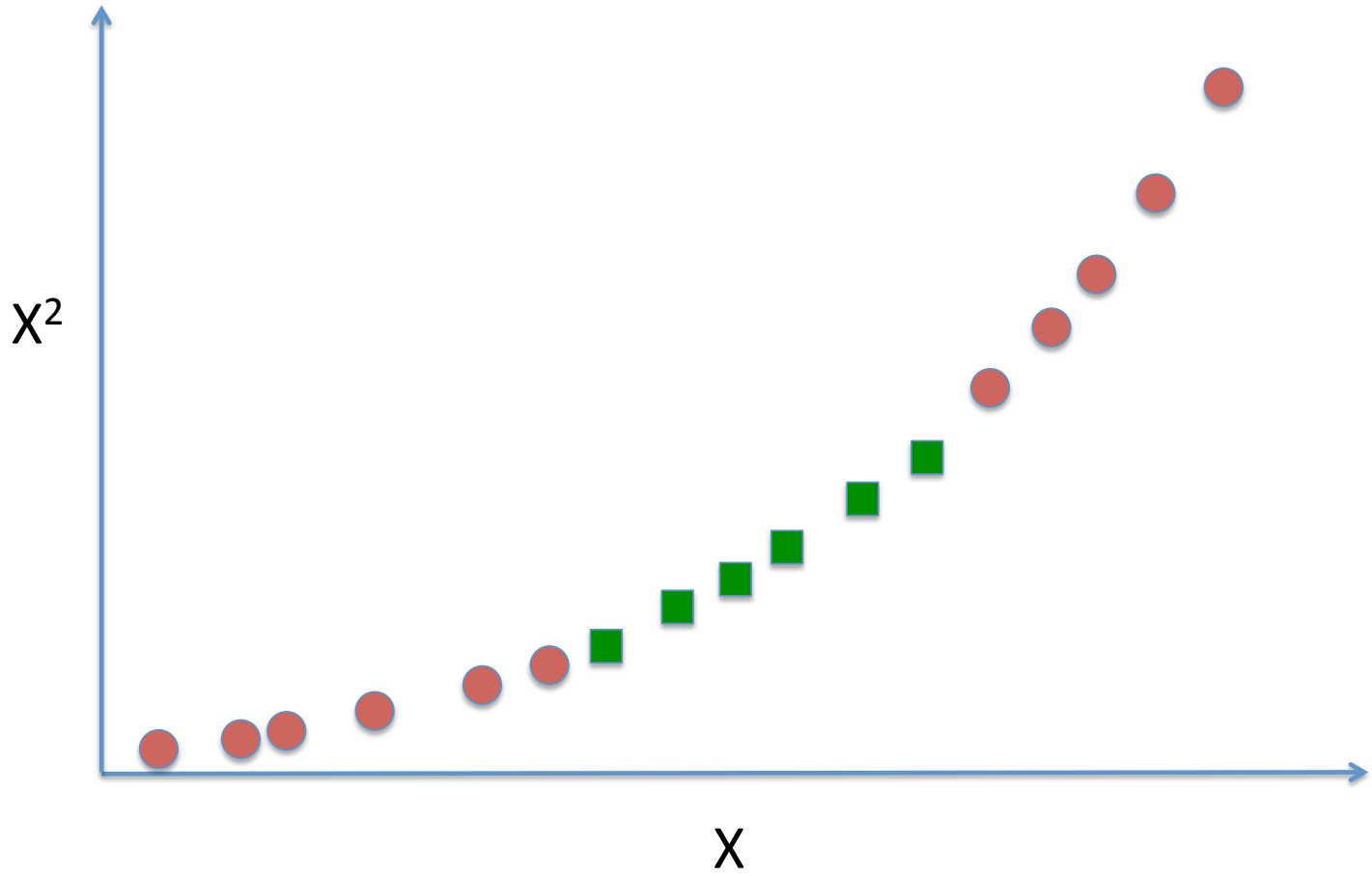
Recall: Kernels for classification

One dimensional feature space and linear kernel:



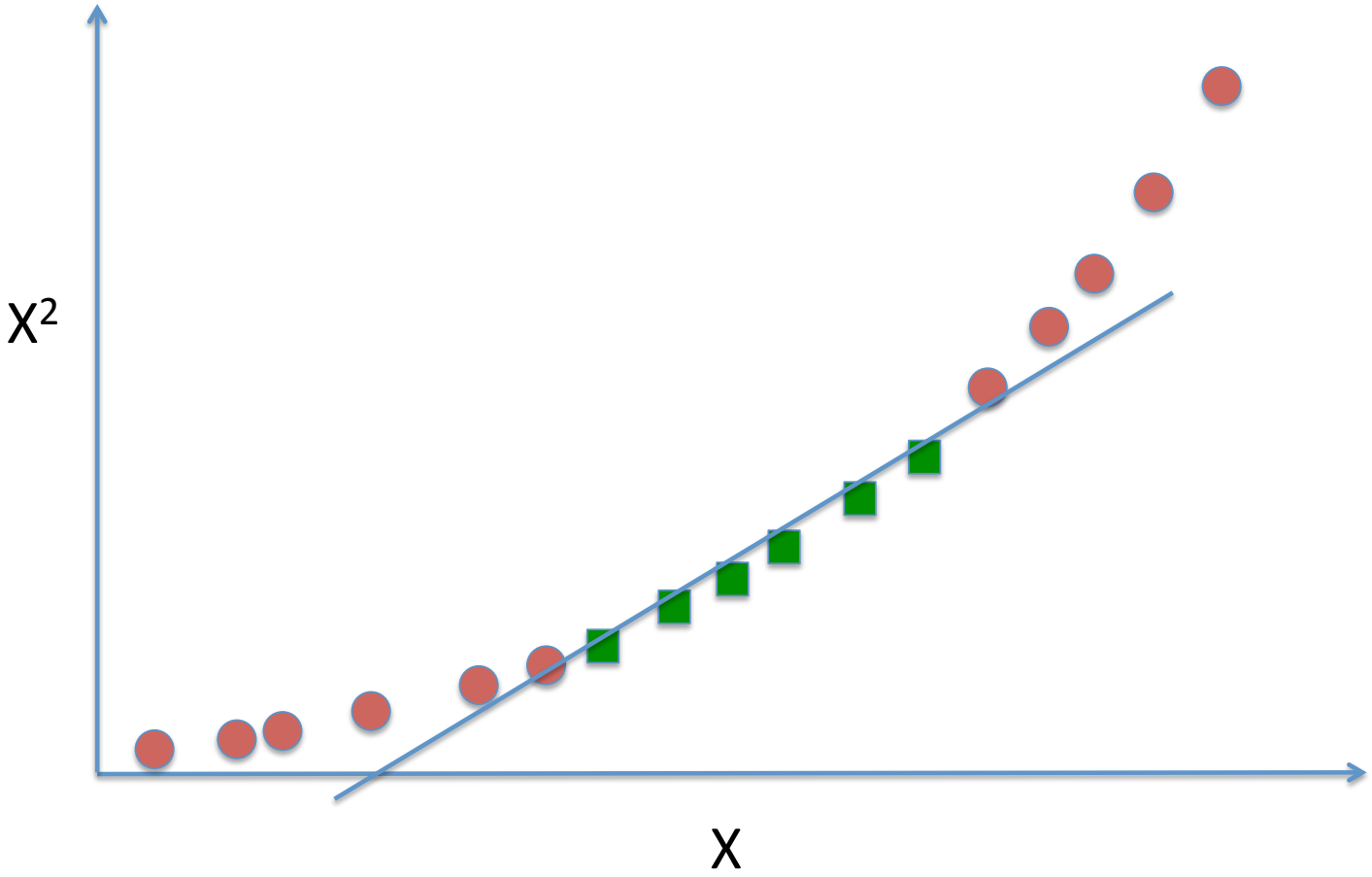
Recall: Kernels for classification

Use $\phi(x) = [x, x^2]$ to project up to 2D feature space:



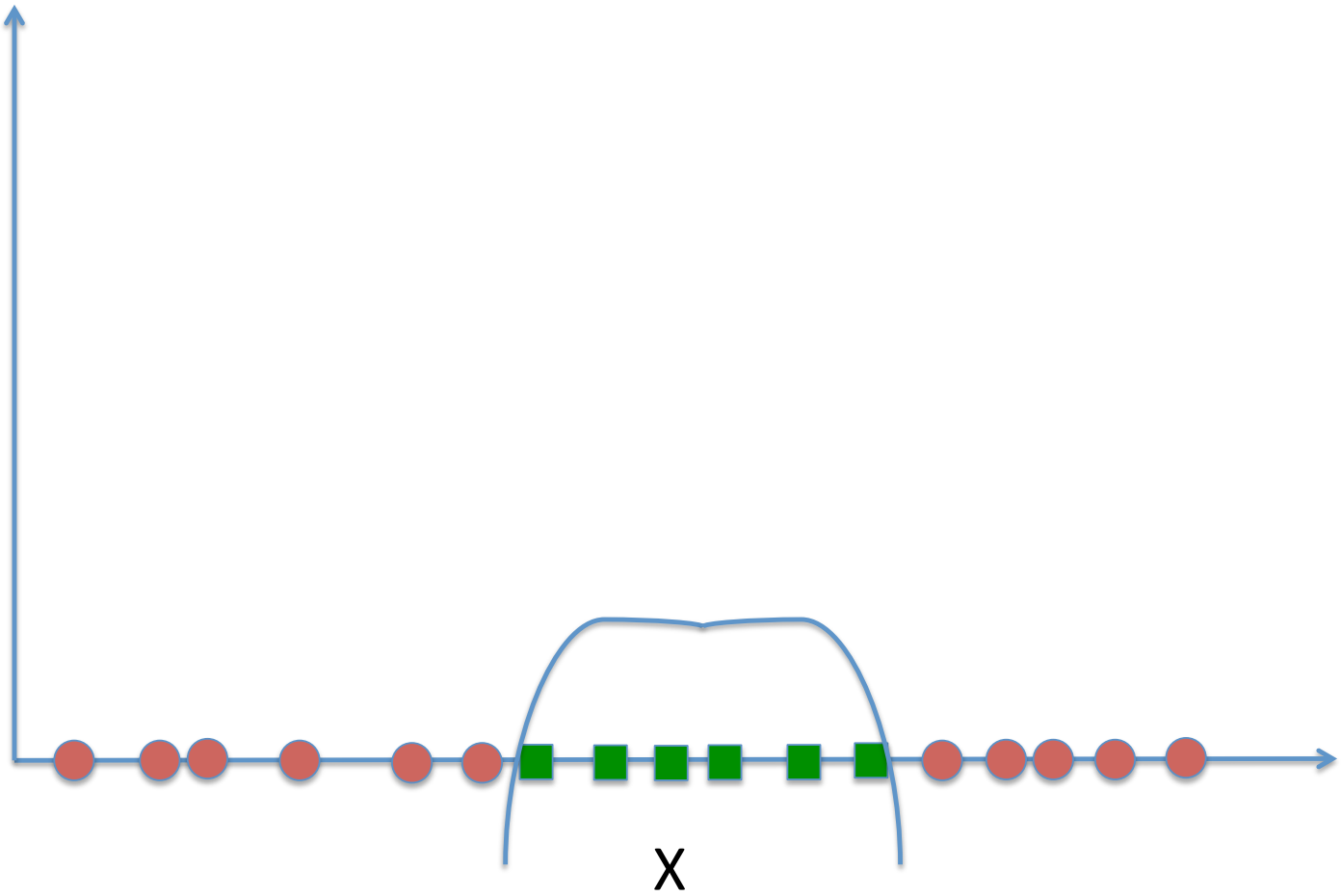
Recall: Kernels for classification

Linear classifier is successful in this space:



Recall: Kernels for classification

The linear classifier projected down to the one-dimensional space:



Kernelized Machine

What classifiers can be kernelized?

All feature vectors can be *kernelized*;

not all classifiers are able to exploit the *kernel trick*.

If $\mu_d \in \mathcal{X}$ is a set of $d = 1 : D$ *centroids*, or points in the feature space, and $\kappa(x, x')$ is a kernel, we can write:

$$\phi(x) = [\kappa(x, \mu_1), \kappa(x, \mu_2), \ldots, \kappa(x, \mu_D)]$$

Here, $\phi(x)$ is called a **kernelized feature vector**.

Kernelized feature vector

Project point x into the D-dimensional feature space by computing the similarity between x and each centroid $\mu_{1:D}$ via $\kappa(\cdot, \cdot)$.

We can use this *kernelized feature vector* for any type of analysis by replacing $\mathbf{x}$ with $\phi(\mathbf{x})$.

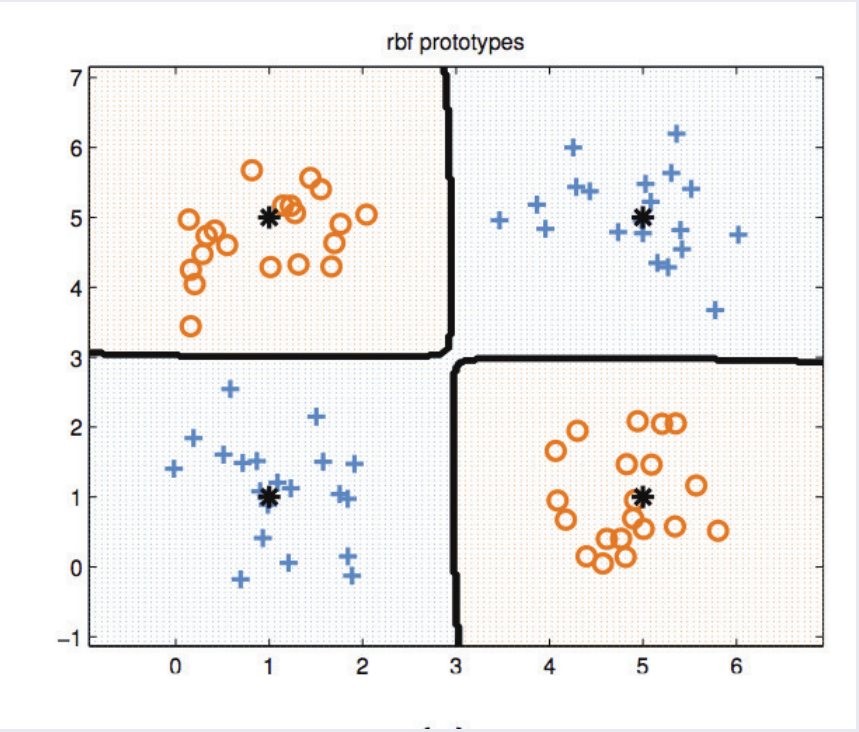
Note: for D centroids, computation is $O(nD)$.

Kernelized machine and linear classifiers

The kernelized feature vector represents the similarity of each x to each centroid.

This provides a simple way to define a non-linear decision boundary using a linear classifier for well chosen centroids.

A Gaussian kernel κ with four centroids and a linear classifier



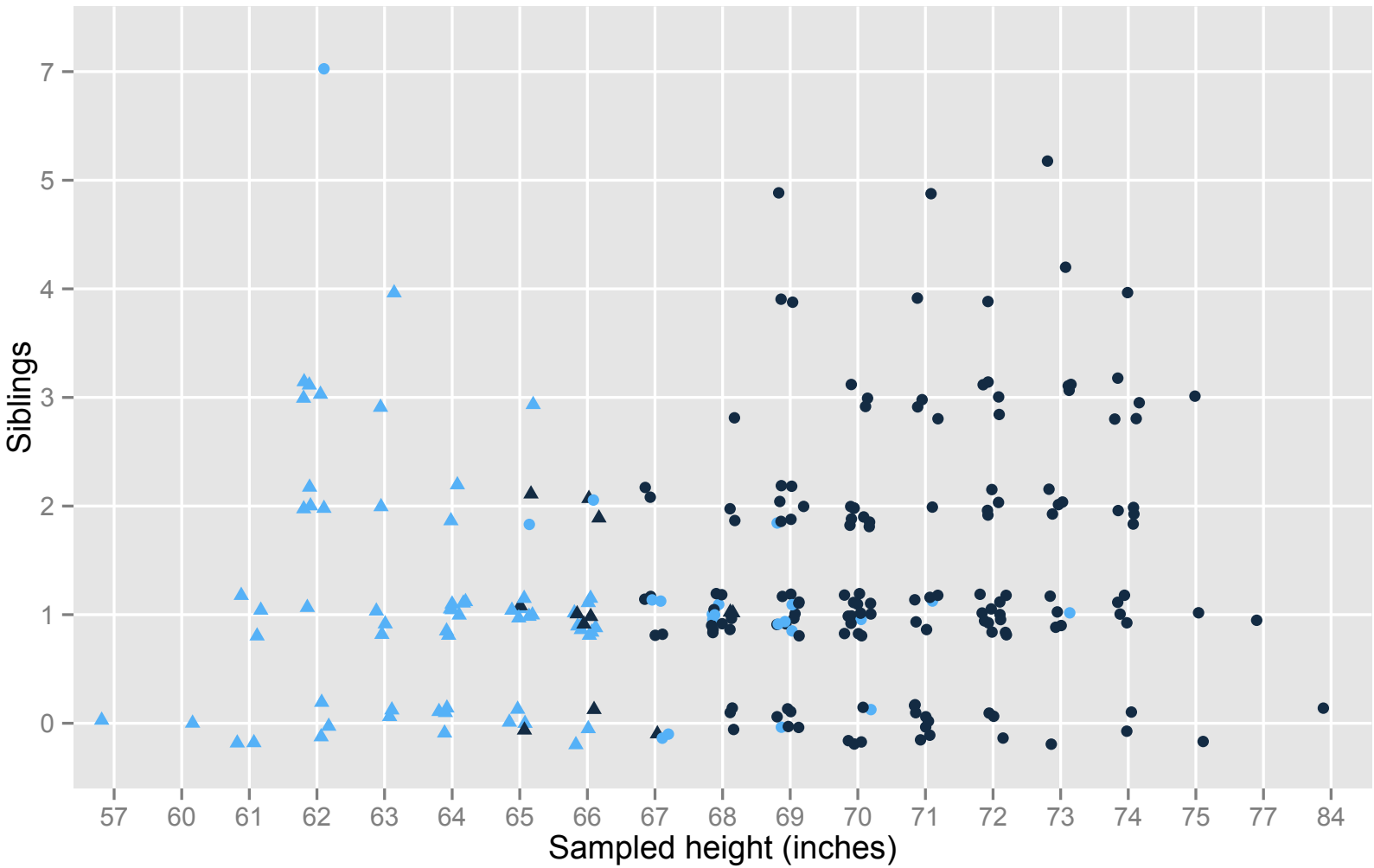
Kernelized K-nearest neighbor classifier

We can *kernelize* the KNN classifier:

- Given data set $\mathcal{D} = \{(x_1, z_1), \ldots, (x_n, z_n)\}$, and a kernel $\kappa(\mathbf{x}, \mathbf{x}')$
- For $\mathbf{x}^*$, compute the vector $\kappa(\mathbf{x_i}, \mathbf{x}^*)$ for all $\mathbf{x_i}, \mathbf{i} = \mathbf{1} : \mathbf{n}$.
- Find the K nearest neighbors by kernel similarity: samples most similar to $\mathbf{x}^*$ in the feature space.
- Then $\hat{z}^*$ is the most frequent class label from K nearest neighbors.

This method exploits the kernel trick: computing kernel function is $O(n)$, but similarity is quantified in a (possibly) high dimension feature space.

K nearest neighbors: example with K=3, Gaussian kernel



$$\text{accuracy} = 0.86$$

Kernelized K nearest neighbors: summary

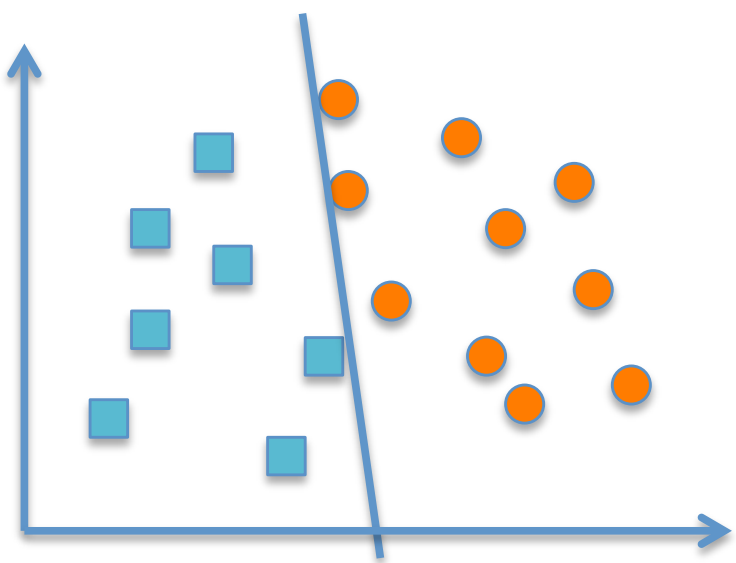
K classes, n samples, p features:

Classifier	Naive Bayes	KNN
Model based?	Y	N
Classifier type?	generative	cluster
Kernelizeable?	N	Y
Additive?	Y	non-linear distance
Parameters?	N	K
Multiclass?	Y	Y
Interpretable?	Y	N
Missing data?	Y	N
Training?	O(np)	None
Test?	O(Kp)	O(np)

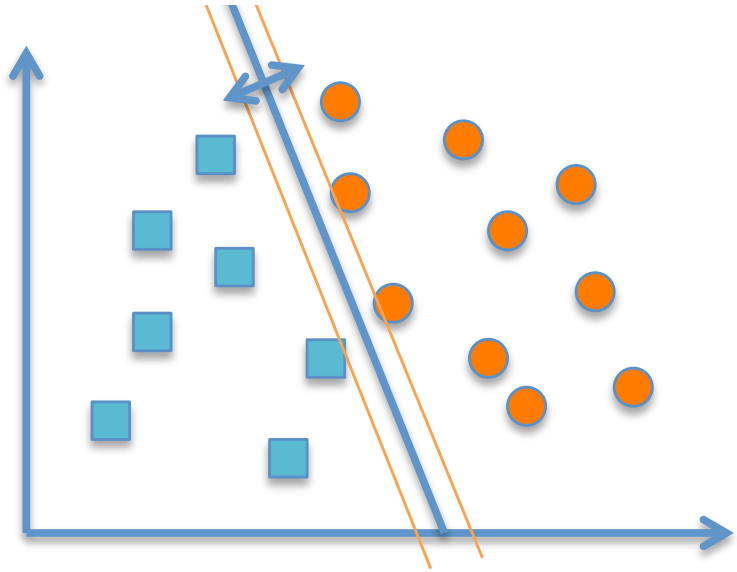
Support Vector Machines (SVMs)

A SVM classifier finds a linear separator (hyperplane) between training samples that *maximizes the margin* between the two classes of samples.

The SVM is a type of "large margin classifier"



Linear classifier



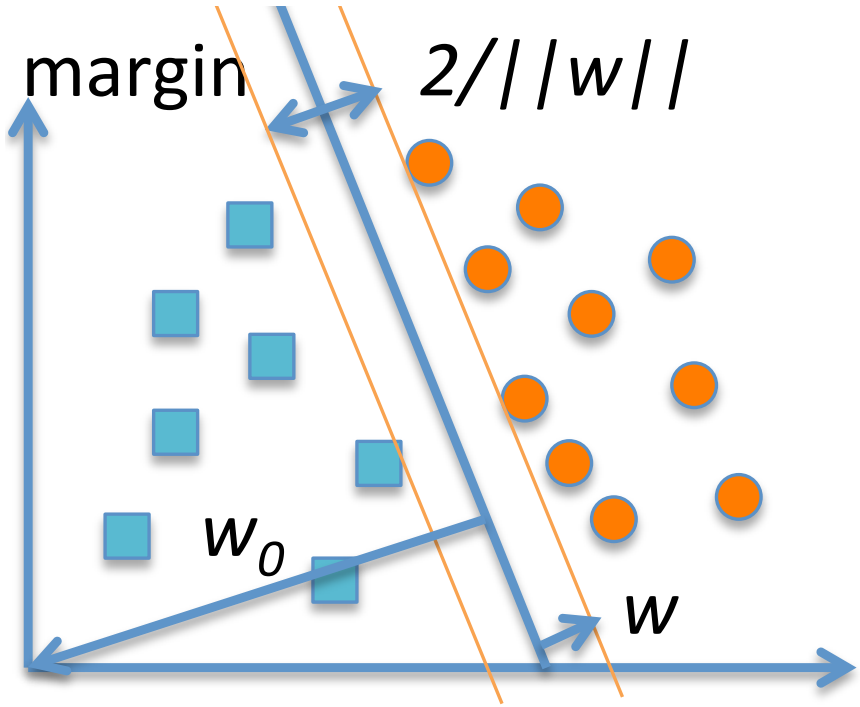
Max-margin linear classifier

SVMs for binary classification

A **support vector machine (SVM)** is used for binary classification ($z \in \{-1, 1\}$)

We can fit an SVM to training data

- $\mathbf{w}$ is *normal vector to the hyperplane*;
- $\mathbf{w}_0$ the *offset from the origin* to the hyperplane.



The separating hyperplane is characterized by: $\mathbf{w}^T\mathbf{x} + \mathbf{w_0} = \mathbf{0}$;

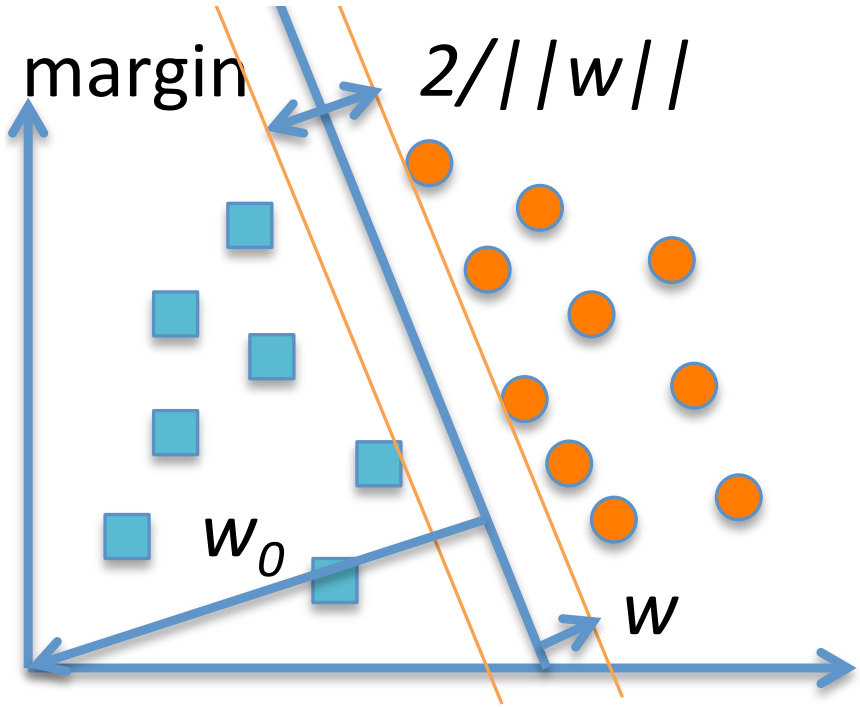
The margin is defined by hyperplanes: $\mathbf{w}^T\mathbf{x} + \mathbf{w_0} = -\mathbf{1}$ and $\mathbf{w}^T\mathbf{x} + \mathbf{w_0} = \mathbf{1}$

SVMs for binary classification

Recall that the distance from a point (x_i, z_i) and a hyperplane $\mathbf{w}^T\mathbf{x} + \mathbf{w_0} = \mathbf{0}$ is:

$$|\mathbf{w}^T\mathbf{x_i} + \mathbf{w_0}|/||\mathbf{w}|| \quad = \quad 1/||w||,$$

So the width of the margin, for a set of *support vectors*, has width $2/||w||$.



Fitting an SVM to training data

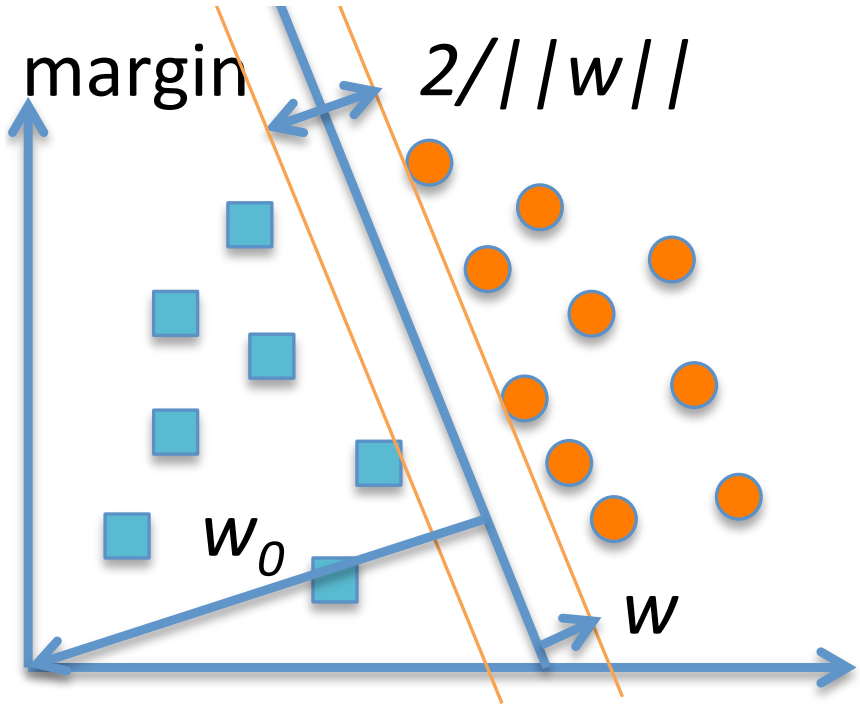
We can fit an SVM to training data, $\mathcal{D} = \{(x_1, z_1), \ldots, (x_n, z_n)\}$, $z \in \{-1, 1\}$ by solving the following optimization problem:

$$\mathbf{w} = \arg\min ||w||^2$$

Subject to:

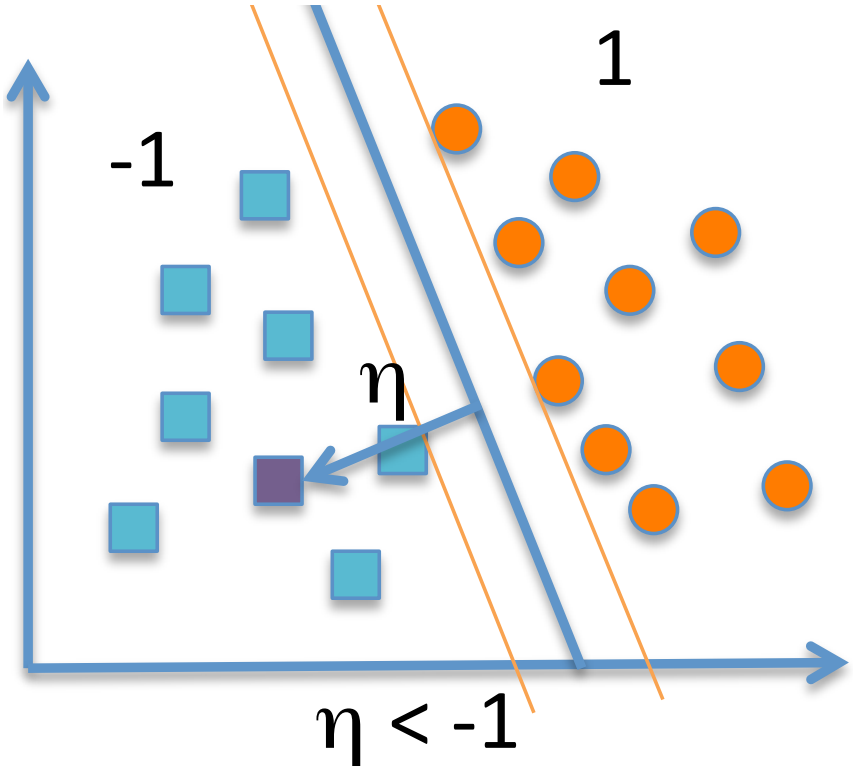
$\mathbf{w}^T\mathbf{x}+\mathbf{w_0} > +\mathbf{1}$, for positive examples

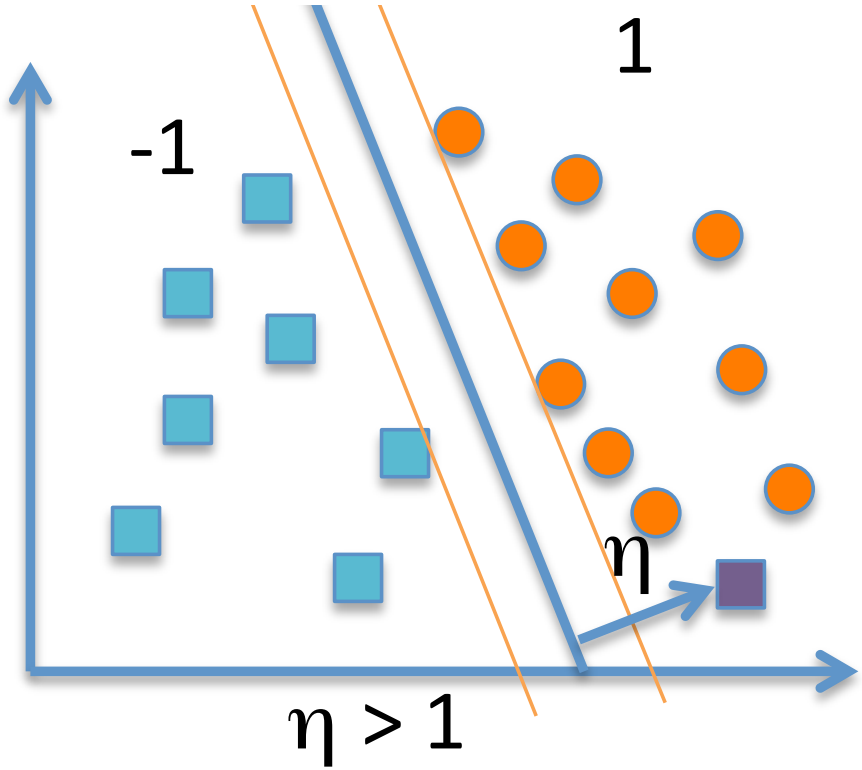
$\mathbf{w}^T\mathbf{x}+\mathbf{w_0} < -\mathbf{1}$, for negative examples



SVMs for binary classification

- Given a fitted SVM ($\mathbf{w}$ and $\mathbf{w}_0$),
- compute $\eta^* = f(\mathbf{x}^*) = \mathbf{w}^\mathbf{T}\mathbf{x}^* + \mathbf{w_0}$
- η^* is the distance from the new point to the hyperplane.
- Then our prediction for class label $\hat{z}^* = \mathit{sign}(\eta^*)$.



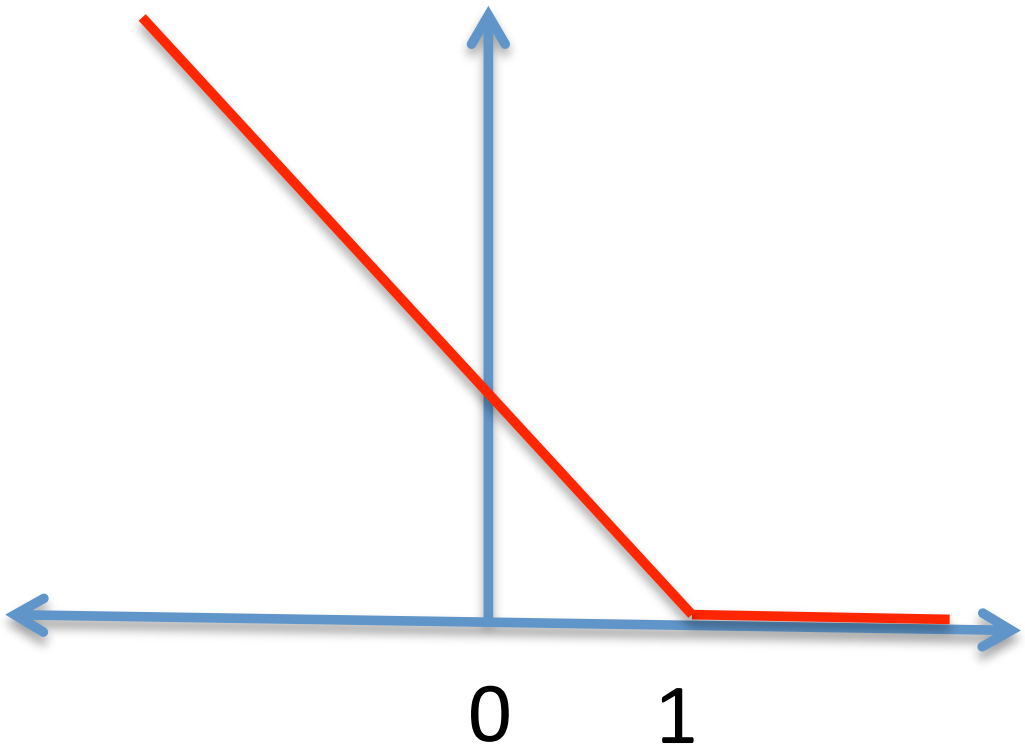


Hinge loss and SVMs

We define the **hinge loss** as follows:

$$L_{hinge}(z, \eta) = \max(0, 1 - z\eta)$$

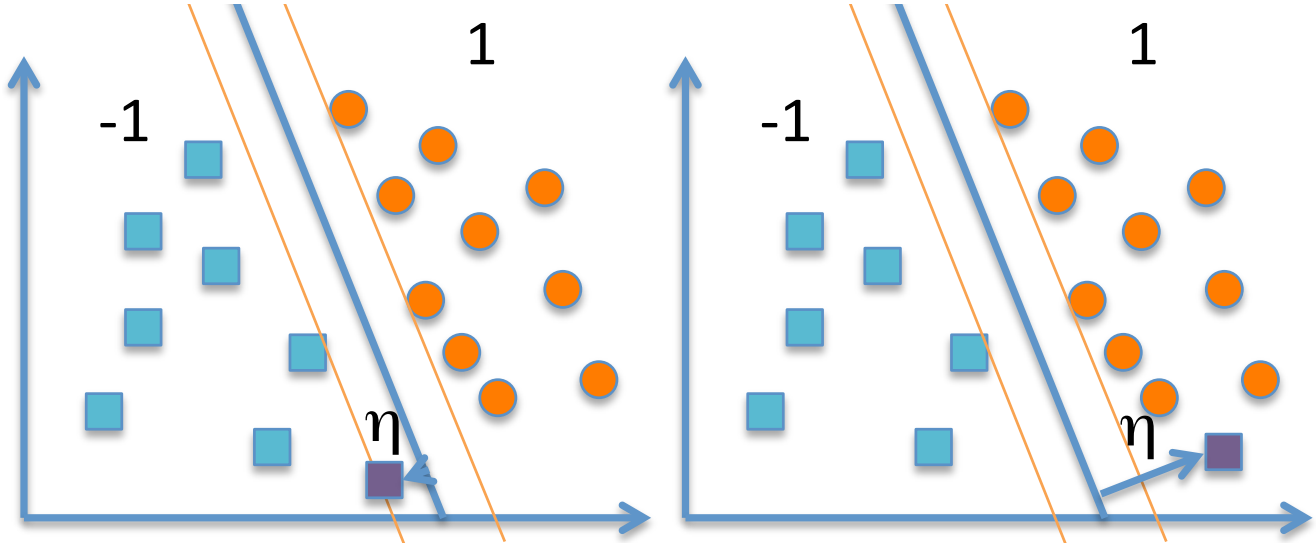
We can define this problem of finding the maximum margin classifier as an *optimization problem* where the objective is to *minimize the hinge loss* with respect to the training data.



SVMs for binary classification

Poor predictions result in a larger value for hinge loss: $\max(0, 1 - z\eta)$.

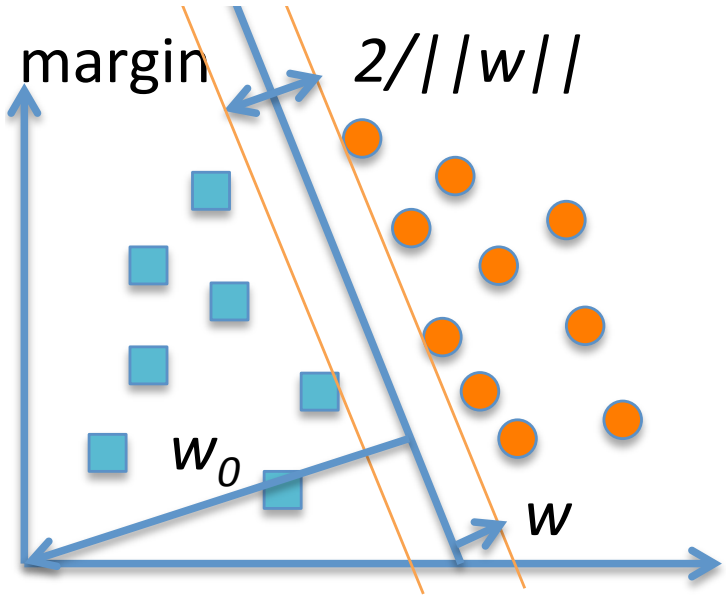
- When prediction $|\eta| \geq 1$, and prediction sign matches truth z, hinge loss is zero.
- When prediction is between the margin and hyperplane (i.e., $0 \geq |\eta| \geq 1$), hinge loss is small.
- When prediction has unmatched sign to truth, hinge loss is large.



SVM implicitly performs feature selection

SVMs perform *feature selection*: any point that does not lie on the margin does not play a role in the optimization problem.

The *support vectors* are the points that define the margin.



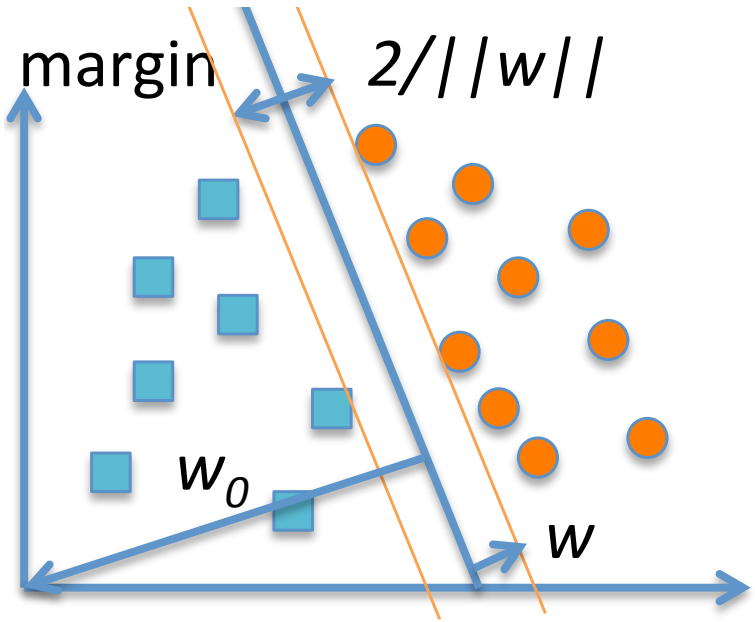
Where are the support vectors?

SVM: optimization problem

This is formally defined as the following optimization problem:

$$\min_{\mathbf{w},w_0} \frac{1}{2}||\mathbf{w}||^2 + c \cdot \sum_{i=1}^{n}(1 - z_i\eta_i))_+$$

This expression is not differentiable.



SVM: optimization problem, with errors

Let's include a *slack term* ξ_i: replace hard constraint $z_i\eta_i \geq 1$ with *soft margin constraints* $z_i\eta_i \geq 1 - \xi_i$ to allow mistakes in classification.

Then, we have the following optimization *[Vapnik & Cortes 1995]*:

$$\min_{\mathbf{w},\mathbf{w}_0,\xi} \frac{1}{2}||\mathbf{w}||^2 + c \cdot \sum_{i=1}^{n} \xi_i, \ \ s.t.$$

$$\xi_i \geq 0$$

$$z_i(\mathbf{x_i^T w} + \mathbf{w_0}) \geq 1 - \xi_i$$

This is a quadratic program, and it takes $O(n^2)$ time to solve.

Its solution takes the form $\hat{w} = \sum_{i=1}^{n} \alpha_i z_i \mathbf{x_i}$, where α_i is sparse and selects only support vectors that define the margin.

SVMs: prediction revisited

Recall that prediction for x^* is performed using:

$$\begin{aligned}
\hat{z}(x^*) &= \text{sign}\left(\hat{w}_0 + \hat{\mathbf{w}}^T \mathbf{x}^*\right) \\
&= \text{sign}\left(\hat{w}_0 + \left(\sum_{i=1}^{n} \alpha_i z_i x_i\right)^T \mathbf{x}^*\right) \\
&= \text{sign}\left(\hat{w}_0 + \sum_{i=1}^{n} \alpha_i z_i x_i^T \mathbf{x}^*\right).
\end{aligned}$$

SVMs: using kernels

Let's look at the form of this classifier:

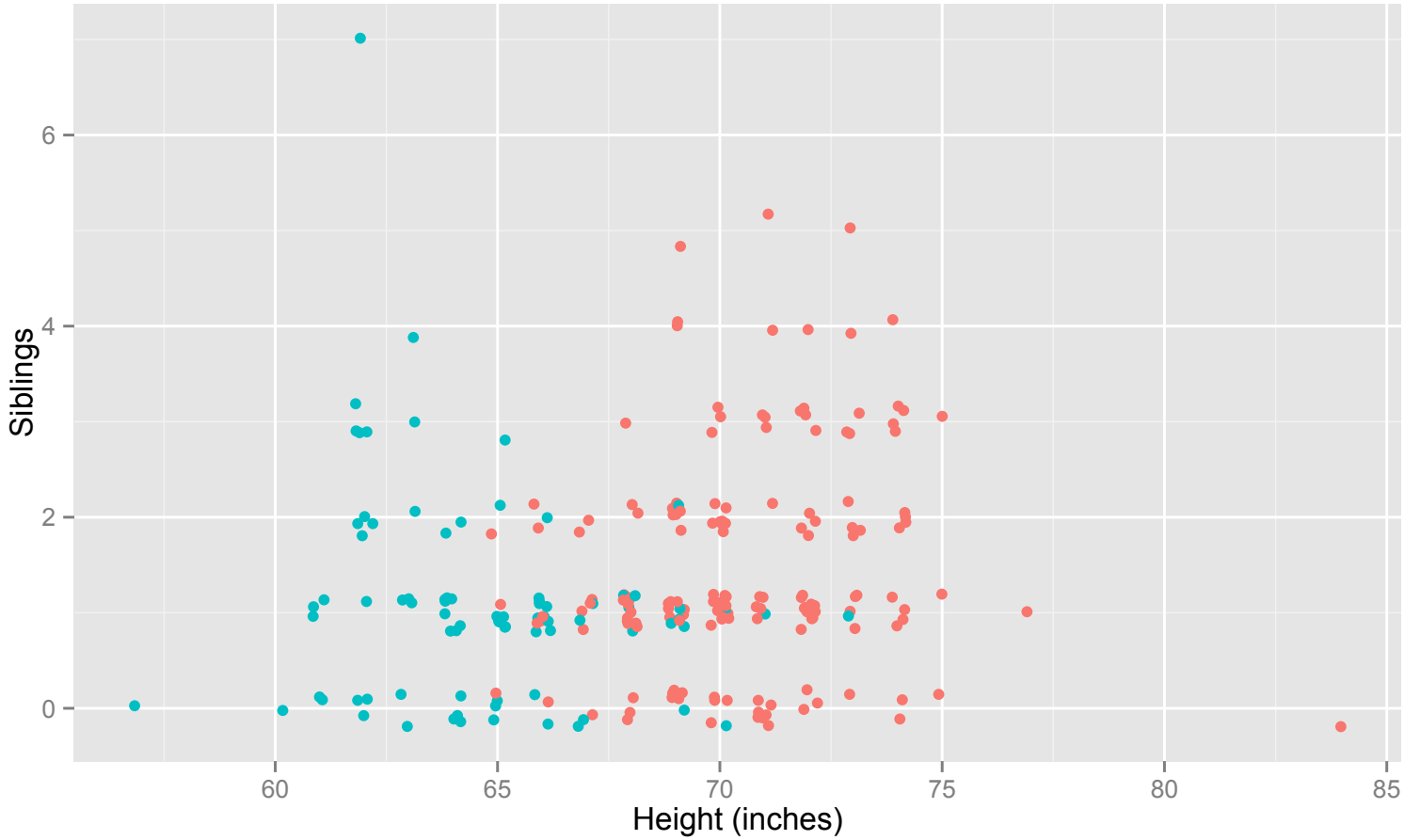
$$\hat{z}(x^*) \quad = \quad \text{sign}\left(\hat{w}_0 + \sum_{i=1}^{n} \alpha_i z_i x_i^T \mathbf{x}^*\right).$$

An SVM can exploit the kernel trick to define a margin in feature space; given n training samples:

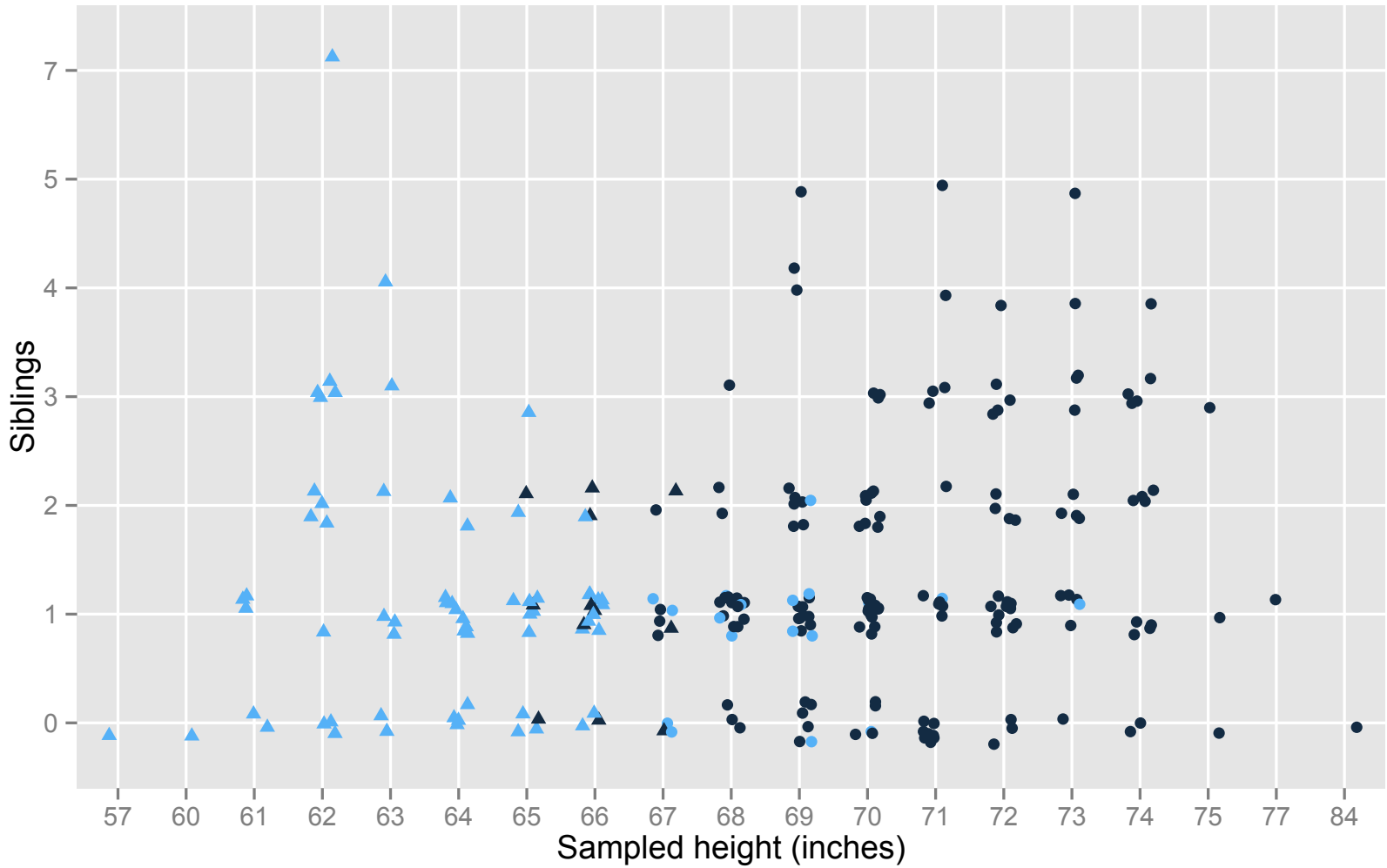
$$\hat{z}(\mathbf{x}^*) = \text{sign}\left(\mathbf{\hat{w}_0} + \sum_{\mathbf{i=1}}^{\mathbf{n}} \alpha_\mathbf{i} \mathbf{z_i} \kappa(\mathbf{x_i}, \mathbf{x}^*)\right).$$

What does a kernelized linear classifier's separating hyperplane look like in the original feature space?

Example: predicting gender from height, siblings

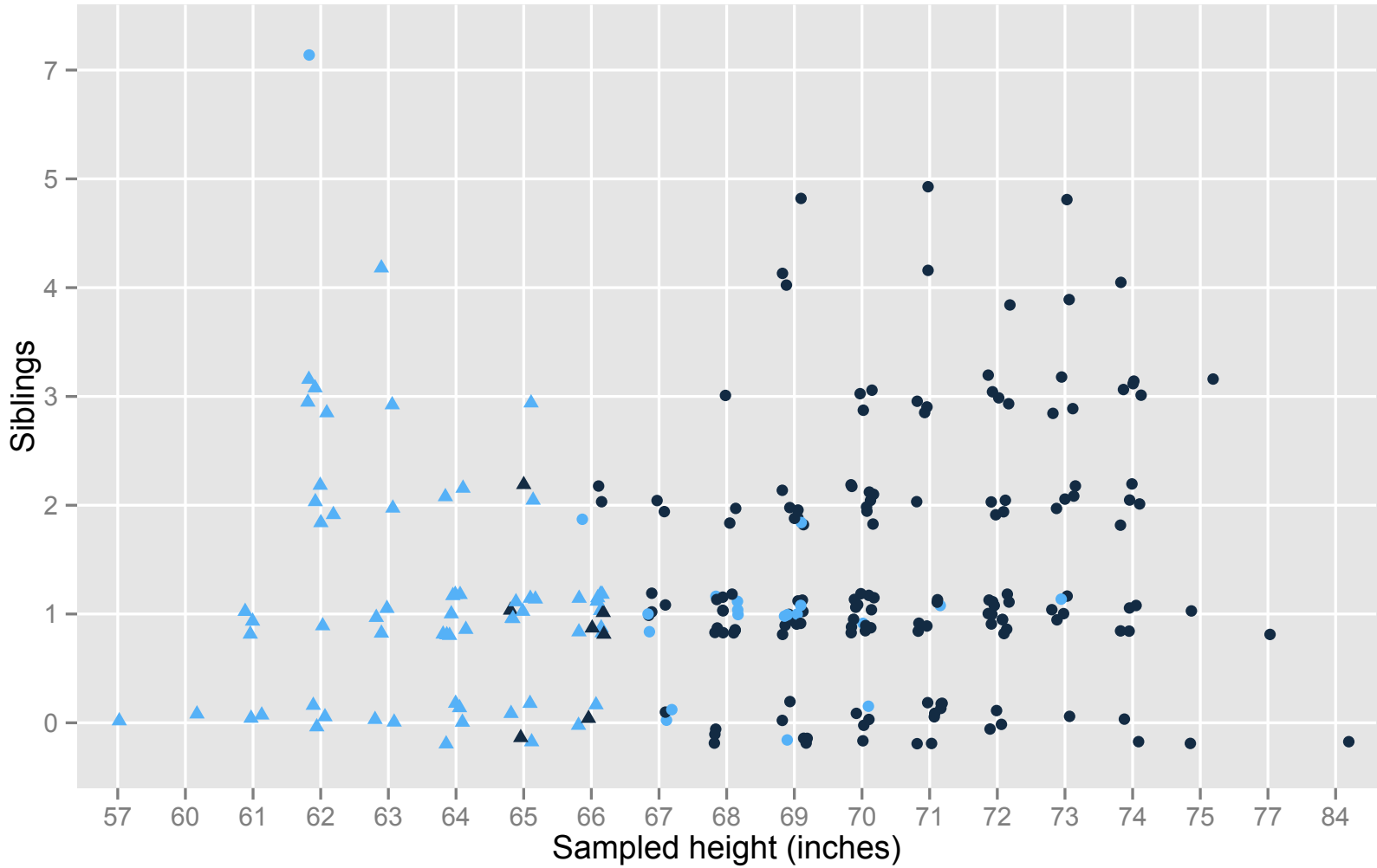


Example: SVM, linear kernel



accuracy = 0.88

Example: SVM, Gaussian kernel



$$\text{accuracy} = 0.89$$

SVMs: summary

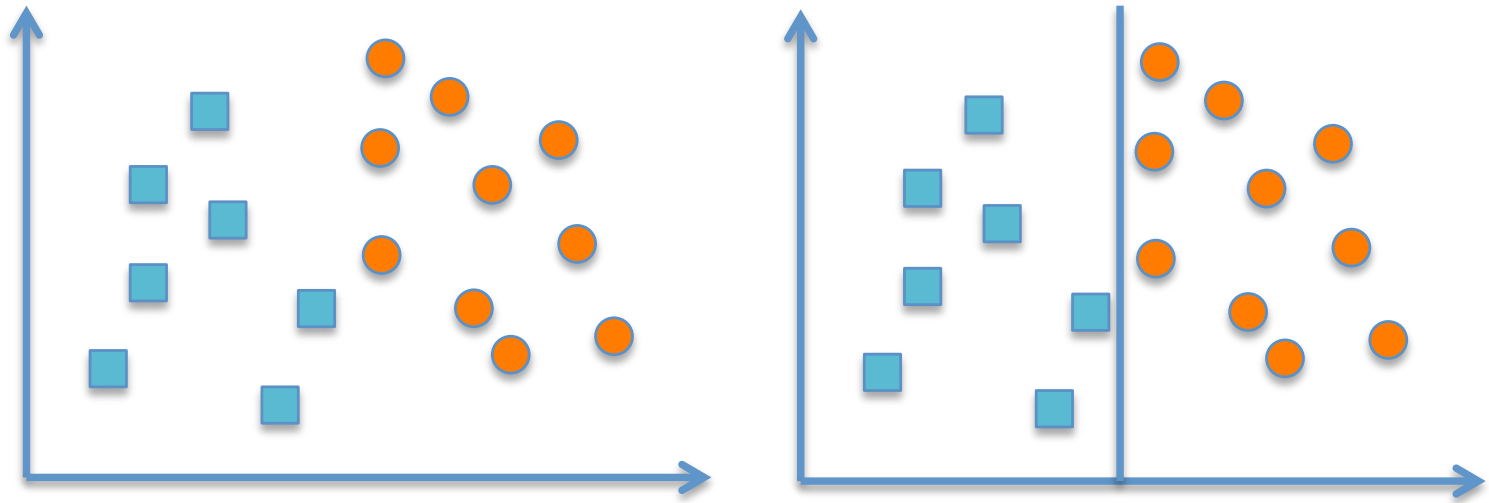
For K number of classes, n training samples, p features:

Classifier	Naive Bayes	KNN	SVMs
Model based?	Y	N	N
Classifier type?	generative	cluster	discriminative
Kernelizeable?	N	Y	Y
Additive?	Y	non-linear dist	non-linear kern
Parameters?	N	K	c
Multiclass?	Y	Y	N
Interpretable?	Y	N	linear kernel
Missing data?	Y	N	N
Training?	O(np)	None	$O(n^2)$
Test?	O(Kp)	O(np)	$O(\|SVs\|p)$

Decision trees

A *decision tree* partitions the feature space in such as way as to have each partition have as little class uncertainty as possible.

- *Information gain*: the difference in Shannon entropy in a subspace and a partition of that subspace.
- *Shannon entropy*: a measure of unpredictability.



High entropy

Low entropy

Shannon entropy and a biased coin

What is the bias of the most unpredictable coin?

What is the bias of the least unpredictable coin?

Decision trees: build from a training data set

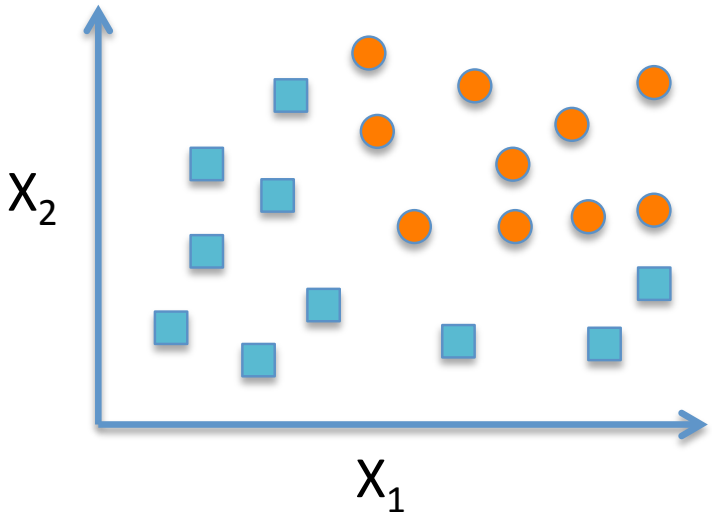
To build a decision tree, given a data set $\mathcal{D} = \{(\mathbf{x_i}, \mathbf{z_i})\}_\mathbf{n}$

- Repeat until all samples at a node are in the same class:
  - For each feature j
    - Find the information gain from splitting on j
  - Create a decision node that splits on feature j^* with greatest information gain
  - Recur on the sublists obtained by splitting on j^*, and add those nodes as children of node j^*

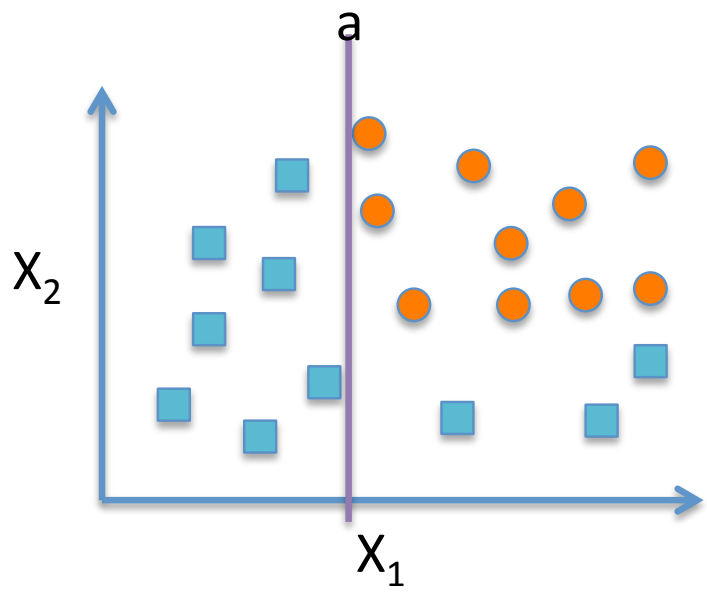
Once a tree is built, leaf nodes can be pruned. Why is this a good idea?

Decision tree: illustration

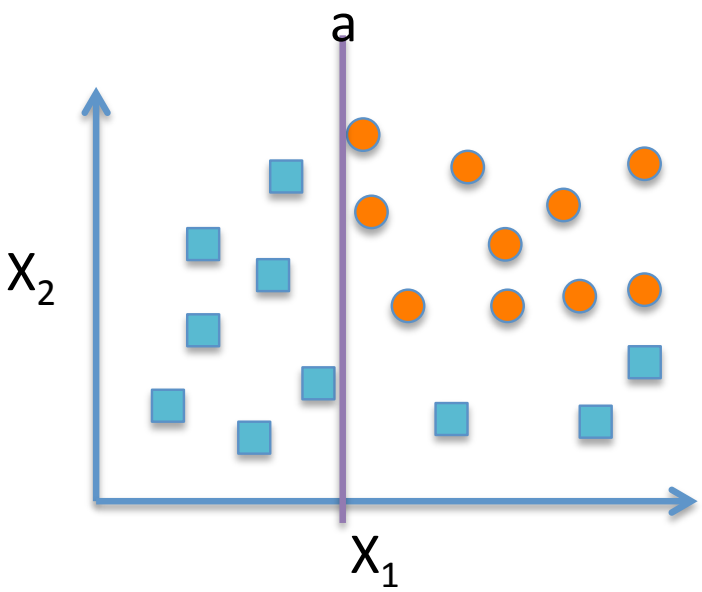
Starting with our data set $\mathcal{D}$:



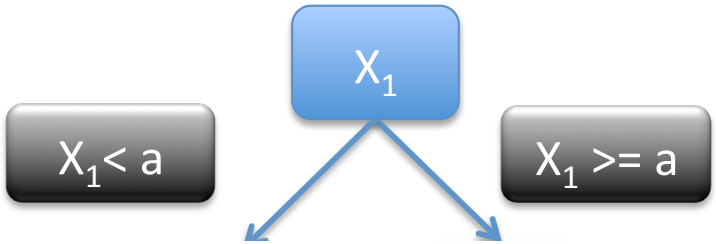
The best feature to split on in terms of information gain is X_1.



Decision tree: illustration

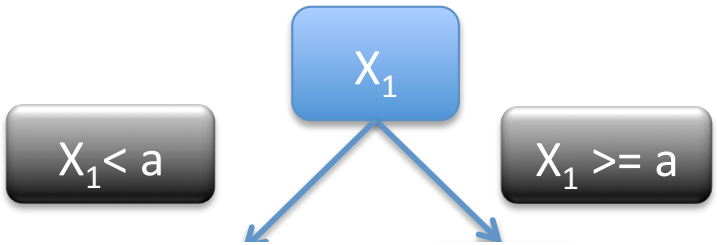


We can build the first node to partition the data in the tree:

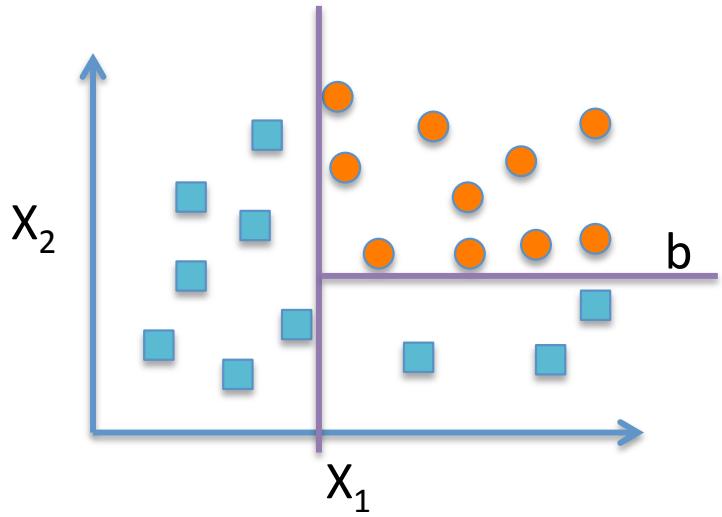


Decision tree: illustration

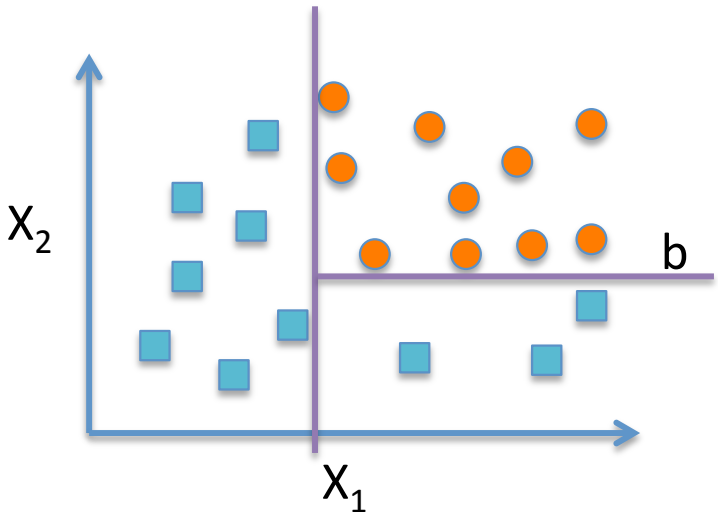
In the $X_1 < a$ partition, we have no uncertainty in the class; we can stop there.



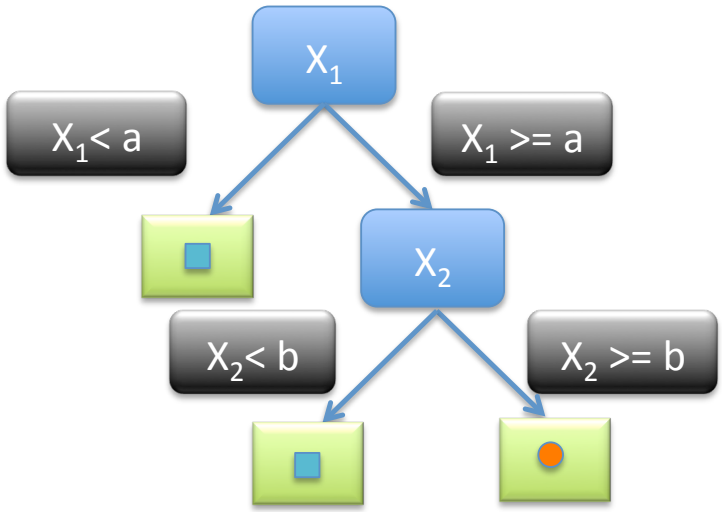
Turning to the $X_1 \geq a$ partition, partitioning now on X_2 gives us the greatest information gain:



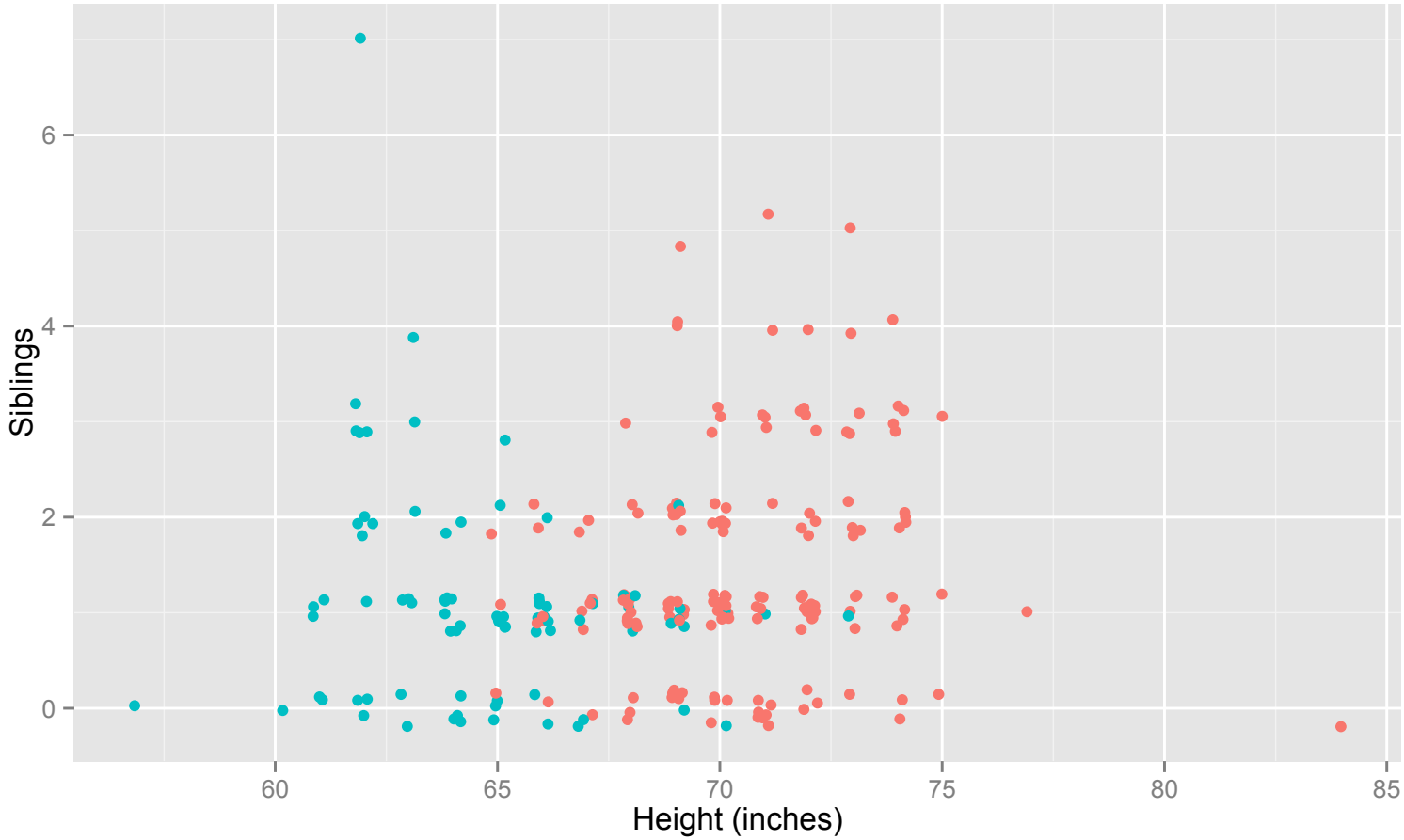
Decision tree: illustration



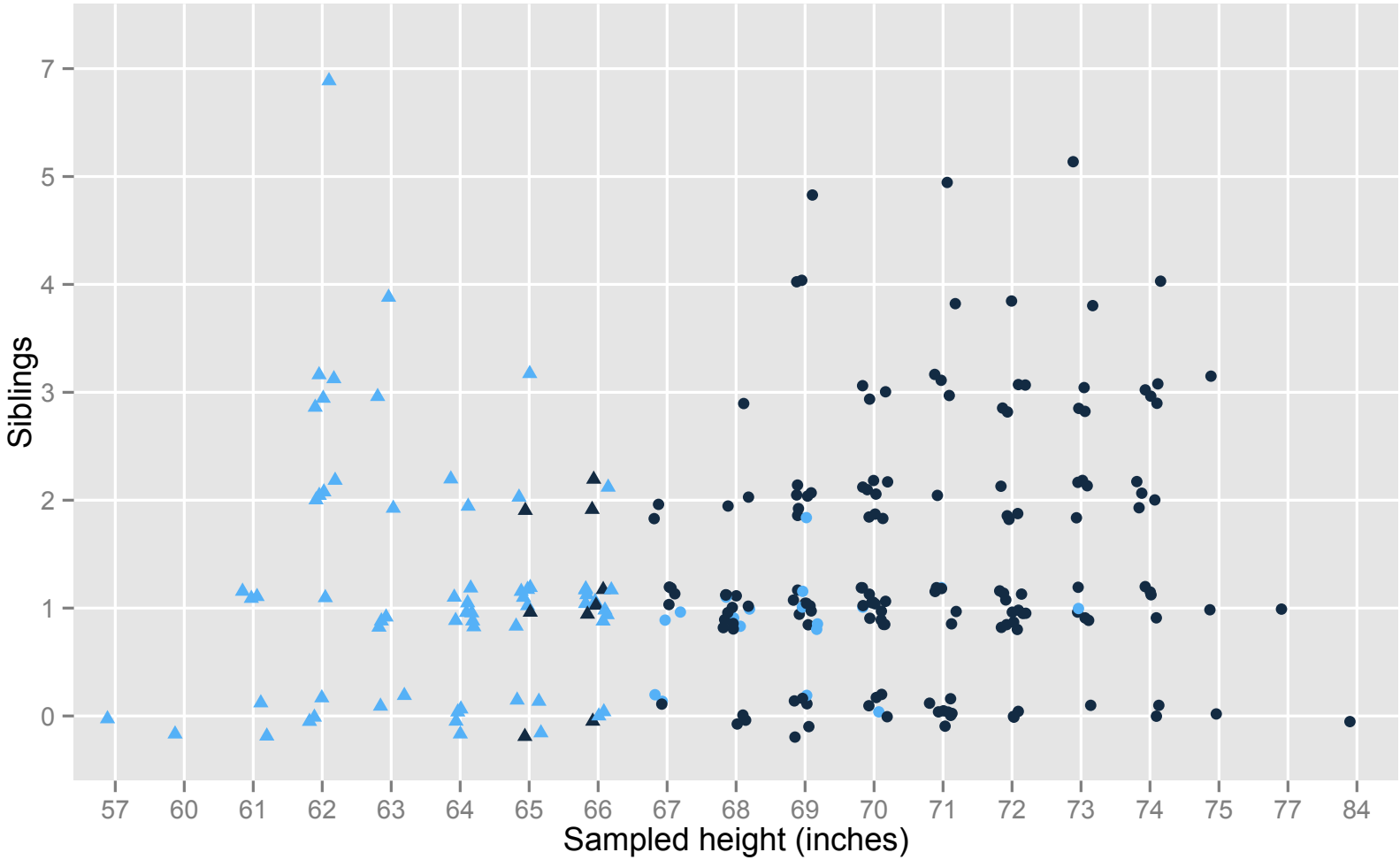
We can build the second node to partition the data in the tree, and stop there because we have no uncertainty in any leaf nodes:



Example: predicting gender from height, siblings

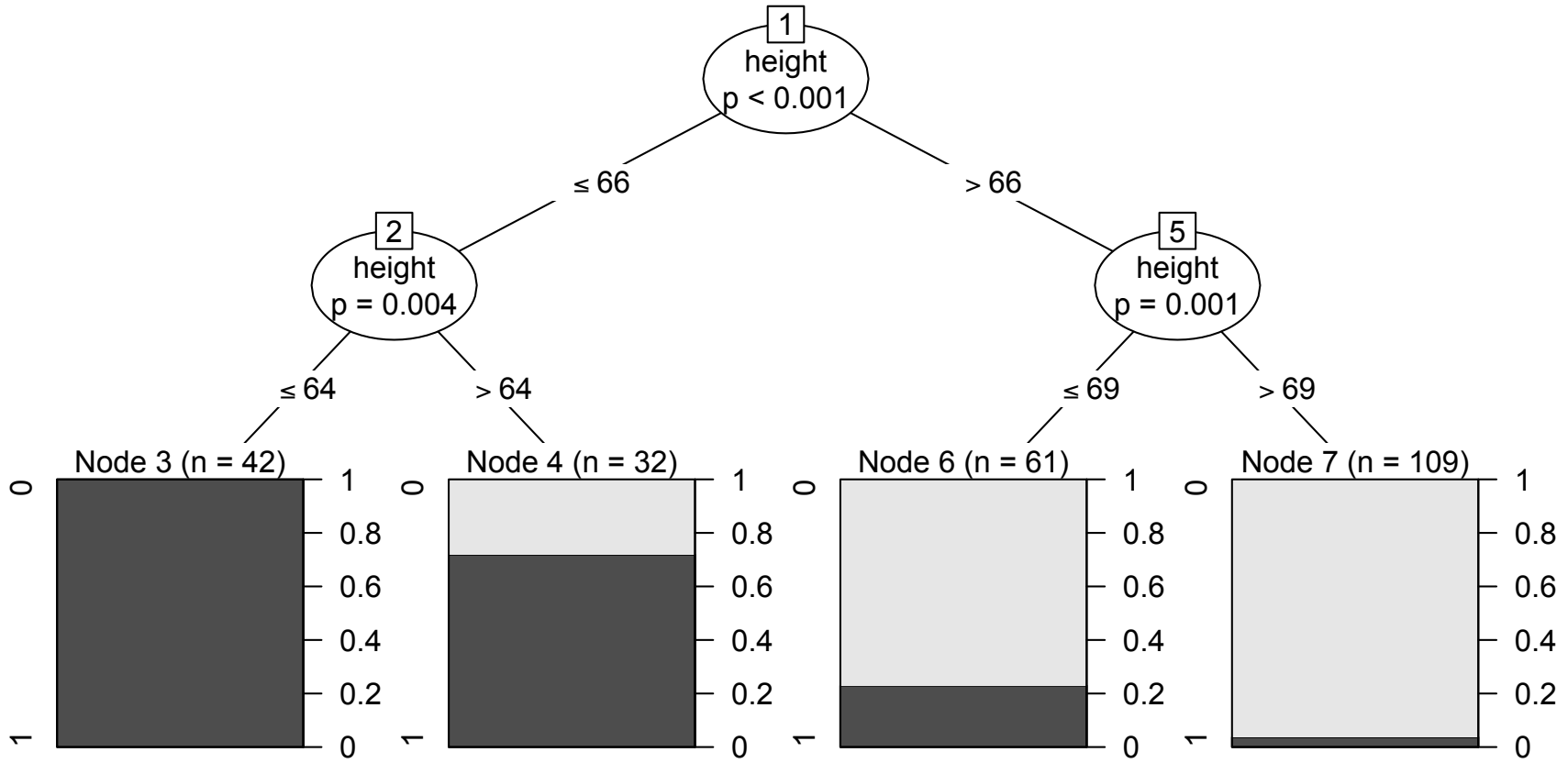


Example: Decision tree



accuracy $= 0.89$

Example: Decision tree



Where is *siblings*?

Decision trees: summary

For K number of classes, n training samples, p features:

Classifier	NB	KNN	SVMs	DT
Model based?	Y	N	N	N
Classifier type?	gen	cluster	discr	discr
Kernelizeable?	N	Y	Y	N
Additive?	Y	non-linear dist	non-linear kern	N
Parameters?	N	K	c	N
Multiclass?	Y	Y	N	Y
Interpretable?	Y	N	linear kern	Y
Missing data?	Y	N	N	N
Training?	$O(np)$	None	$O(n^2)$	$O(np^2)$
Test?	$O(Kp)$	$O(np)$	$O(\lvert SVs \rvert)$	$O(\log p)$

Random forests: Ensemble learning

Random forests build K decision trees; during classification, each tree gets one vote on the class label (predicted class label is the mode).

But each tree is built a bit differently than the DT method

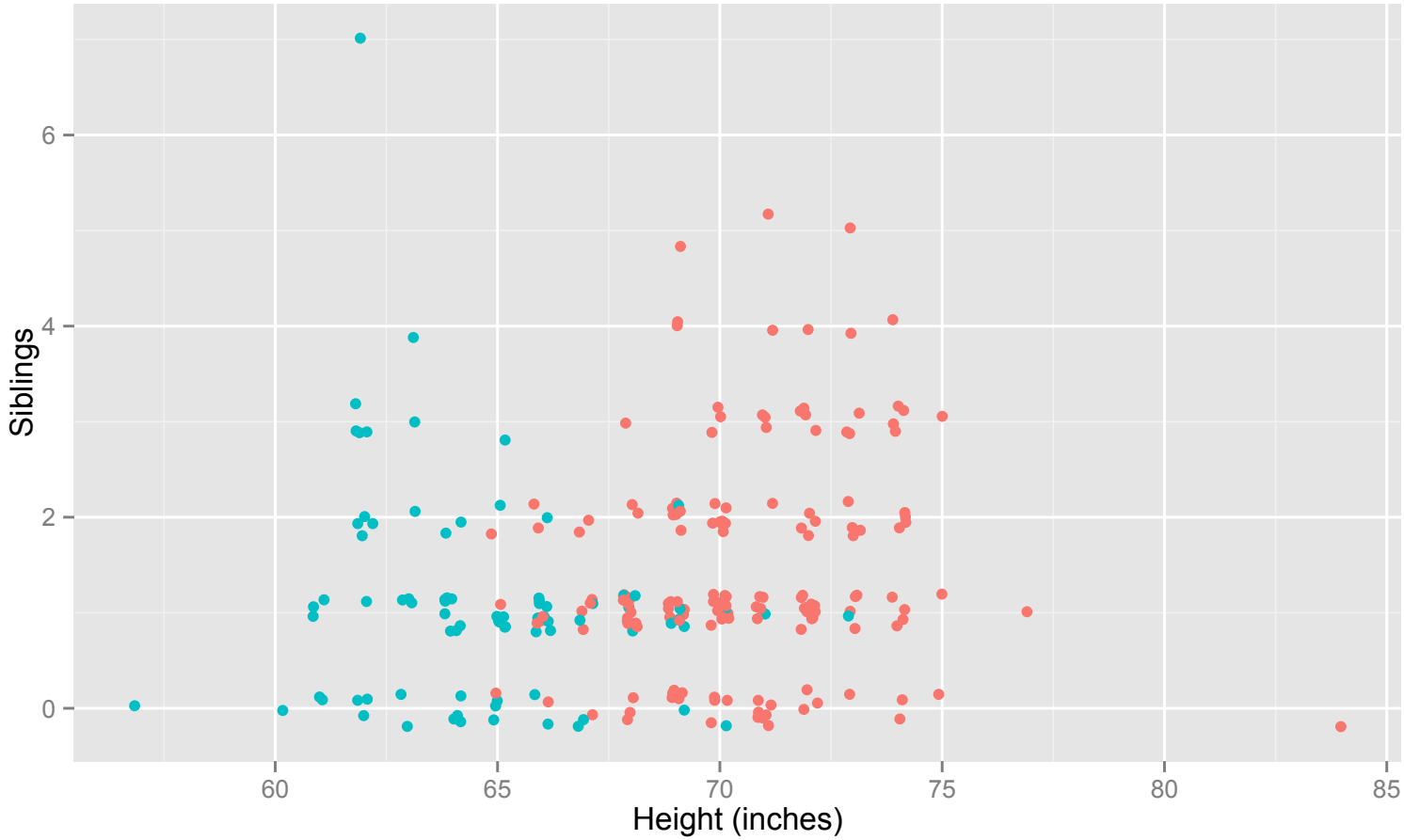
- Each tree is fit to a subset of the samples *sampled with replacement*
- Each split in a tree is selected from a *random subset of the features*

Now every tree has few guarantees, and they are all different.

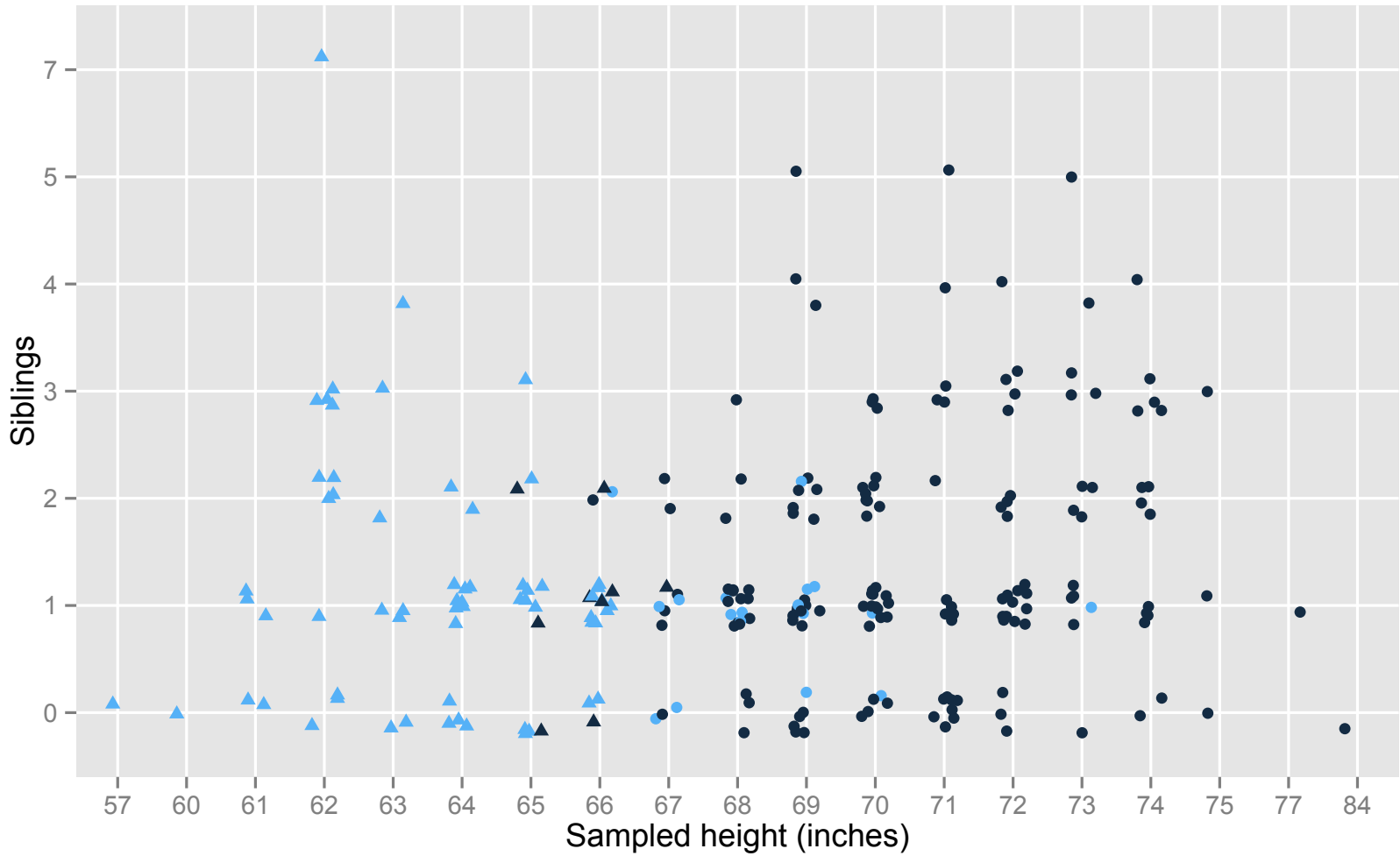
Why is this a good idea?

- Decision trees may overfit training data
- Multiple (weak) hypotheses combine to create a (strong) hypothesis
- Linear classifiers may not be appropriate

Example: predicting gender from height, siblings

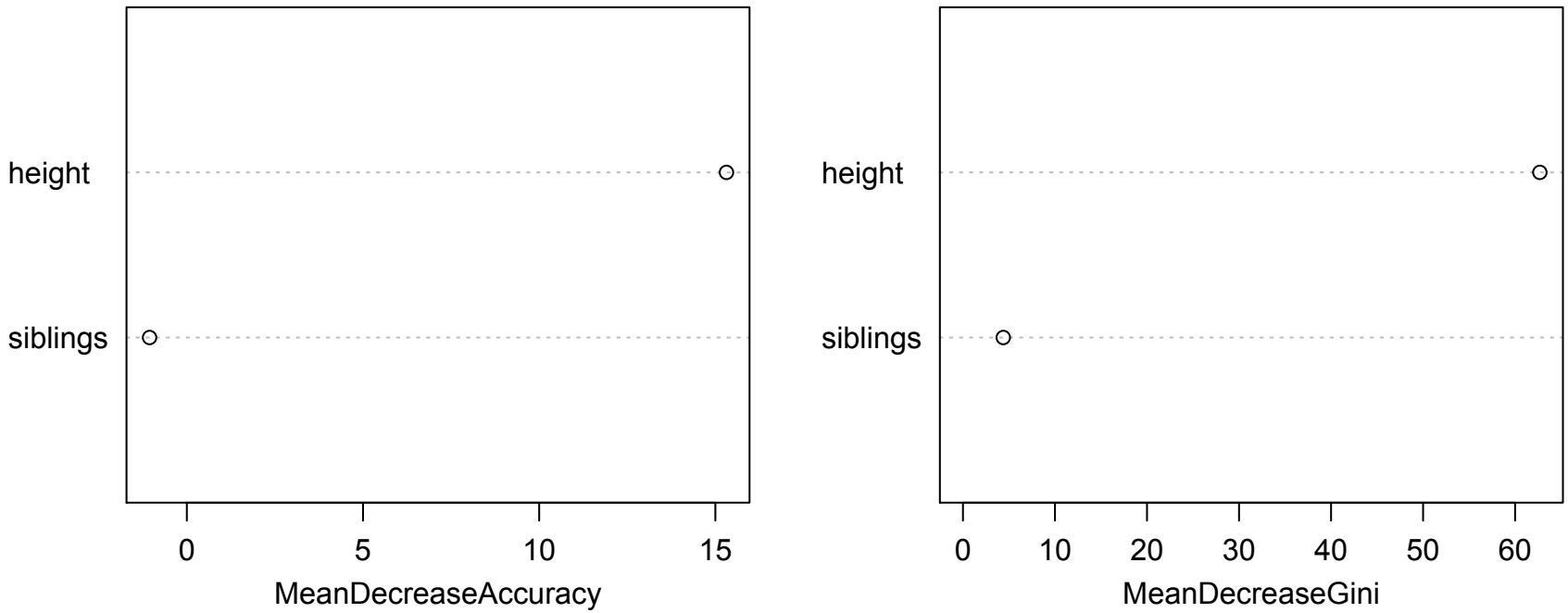


Example: Random forest, K=20



accuracy = 0.89

Example: Random forest, K=20, Interpretability



Information criterion improvements can be averaged across all trees to give an information score for each feature.

Random forests: summary

For K number of classes, n training samples, p features:

Classifier	NB	KNN	SVMs	DT	RFs
Model based?	Y	N	N	N	N
Classifier type?	gen	cluster	discr	discr	discr
Kernelizeable?	N	Y	Y	N	N
Additive?	Y	nonlinear dist	non-linear kern	N	N
Parameters?	N	K	c	N	K
Multiclass?	Y	Y	N	Y	Y
Interpretable?	Y	N	linear kern	Y	Y
Missing data?	Y	N	N	N	N
Training?	O(np)	None	$O(n^2)$	$O(np^2)$	$O(Knp^2)$
Test?	O(Kp)	O(np)	$O(\lvert SVs \rvert)$	$O(\log p)$	$O(K \log p)$

Summary

- Today we saw four different classifiers:
  - K nearest neighbors
  - Support vector machines (SVMs)
  - Decision trees
  - Random forests
- We saw how kernels could be used to make linear classifiers into non-linear classifiers (e.g., SVMs)
- We compared these four classifiers plus NB on 11 different metrics
- Logistic regression will be covered in an upcoming class

Additional resources

- MLAPA: Chapter 14
- *Elements of Statistical Learning*: Chapters 6, 9, 10, 12, 13
- *Pattern Recognition and Machine Learning*, Chapter 4

- Discriminative vs generative classifiers: *[Ng & Jordan 2001]*
- Two cultures paper: *[Breiman 2001]*
- (video) Alex Smola: *Kernel Methods and Support Vector Machines*
- Metacademy: Binary linear classifiers
- Metacademy: Kernel Support Vector Machine