Lecture 10: Computational Learning Theory & Kernel

Winter 2018

Kai-Wei Chang

CS @ UCLA

kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Recap: Computational Learning Theory

- The Theory of Generalization
  - Using training instance to rule out incorrect hypotheses
- Probably Approximately Correct (PAC) learning
  - How many examples you need to see to obtain a learned function with error $\leq \epsilon$
- Shattering and the VC dimension

The setup

- Instance Space: X, the set of examples
- Concept Space: C, the set of possible target functions: $f \in C$ is the hidden target function
  - Eg: all n-conjunctions; all n-dimensional linear functions, …
- Hypothesis Space: H, the set of possible hypotheses
  - This is the set that the learning algorithm explores
- Training instances: S x {-1,1}: positive and negative examples of the target concept. (S is a finite subset of X)

$$< x_1, f(x_1) >, < x_2, f(x_2) >, \ldots < x_n, f(x_n) >$$

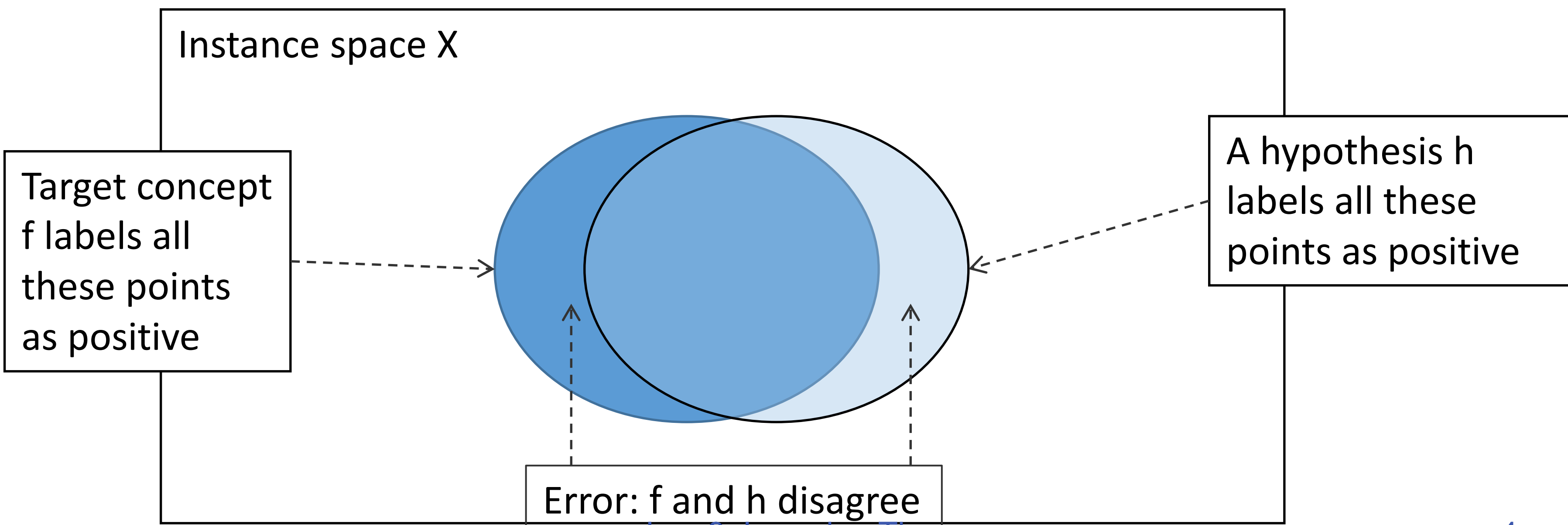
- What we want: A hypothesis h $\in$ H such that h(x) = f(x)
  - A hypothesis h $\in$ H such that h(x) = f(x) for all x $\in$ S ?
  - A hypothesis h $\in$ H such that h(x) = f(x) for all x $\in$ X ?

Recap: Error of a hypothesis

Definition

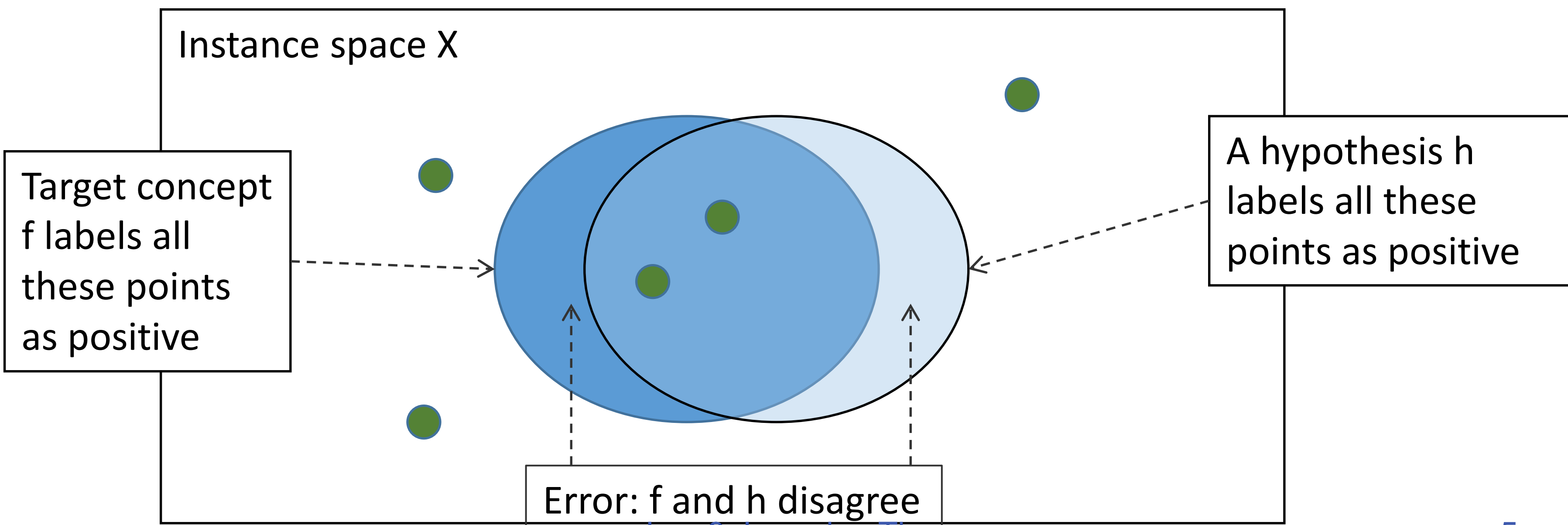
Given a distribution $\mathcal{D}$ over examples, the *error* of a hypothesis h with respect to a target concept f is

$$\mathrm{err}_{\mathcal{D}}(h) = \Pr_{x \sim \mathcal{D}}[h(x) \neq f(x)]$$



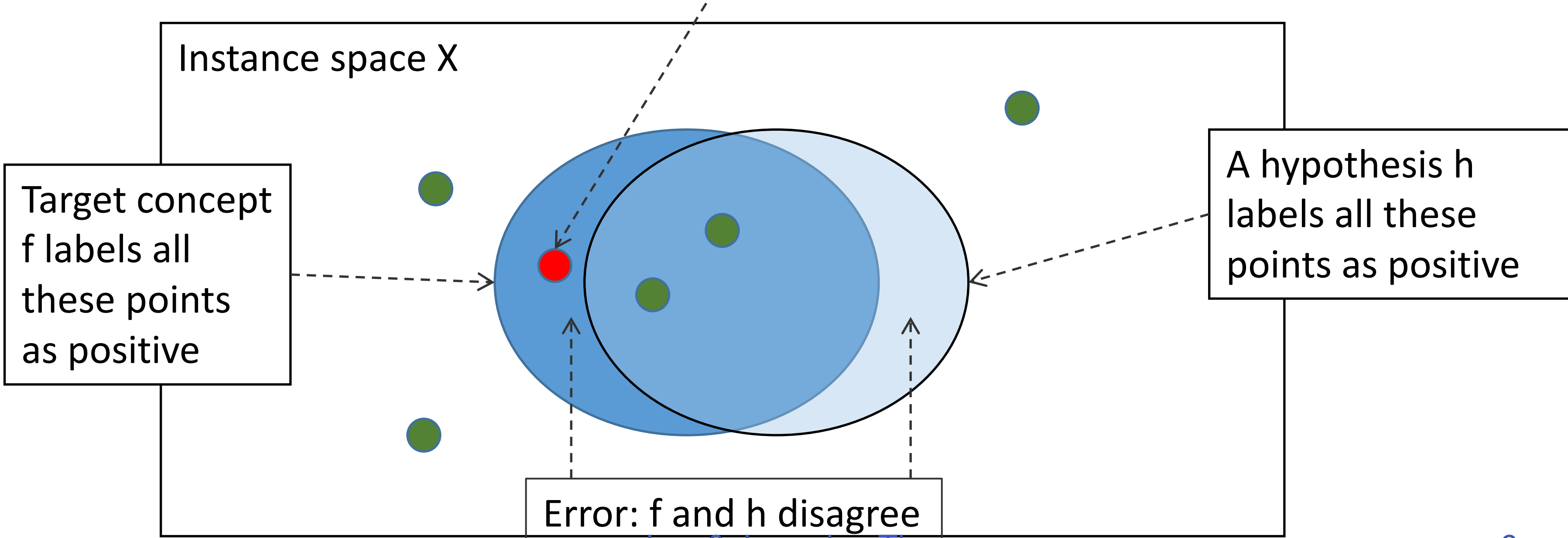
Recap: Error of a hypothesis

Overfitting: You may have a learned model that is consistent with the training data but still makes mistakes.



Recap: Error of a hypothesis

With the IID sampling assumption, we either have seen this example in the training phase, or it is unlikely to see it in the test time.



Requirements of Learning

- Cannot expect a learner to learn a concept exactly
  - There will generally be multiple concepts consistent with the available data
  - Unseen examples could *potentially* have any label
  - We “agree” to misclassify *uncommon* examples that do not show up in the training set

PAC Learnability

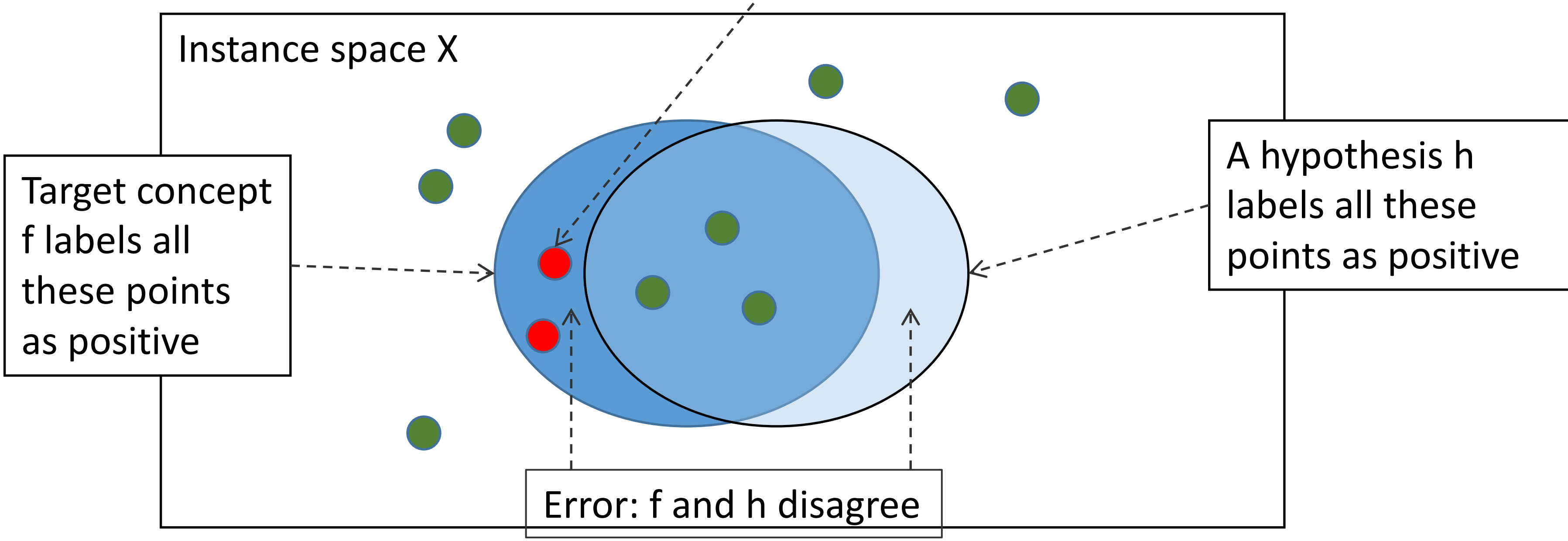
Turing Award: Leslie Valiant.

Consider a concept class C defined over an instance space X (containing instances of length n), and a learner L using a hypothesis space H

The concept class C is PAC learnable by L using H if for all $f \in C$, for all distribution D over X, and fixed $\epsilon > 0$, $\delta < 1$, given m examples sampled i.i.d. according to D, the algorithm L produces, with probability at least $(1- \delta)$, a hypothesis $h \in H$ that has error at most ε, where m is *polynomial* in $1/ \varepsilon$, $1/ \delta$, n and size(H)

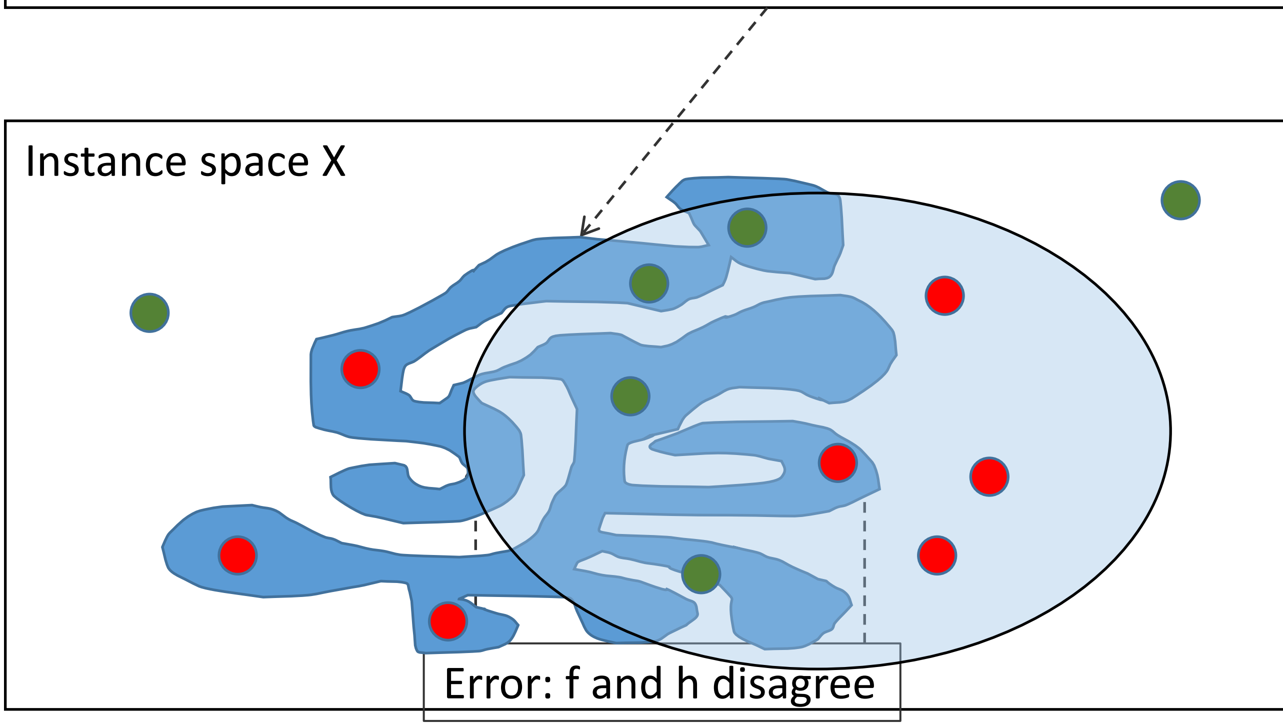
Intuition of PAC Learnability

With the IID sampling assumption, if a concept is reasonable. After, we saw enough samples, it is unlikely to have many these red points



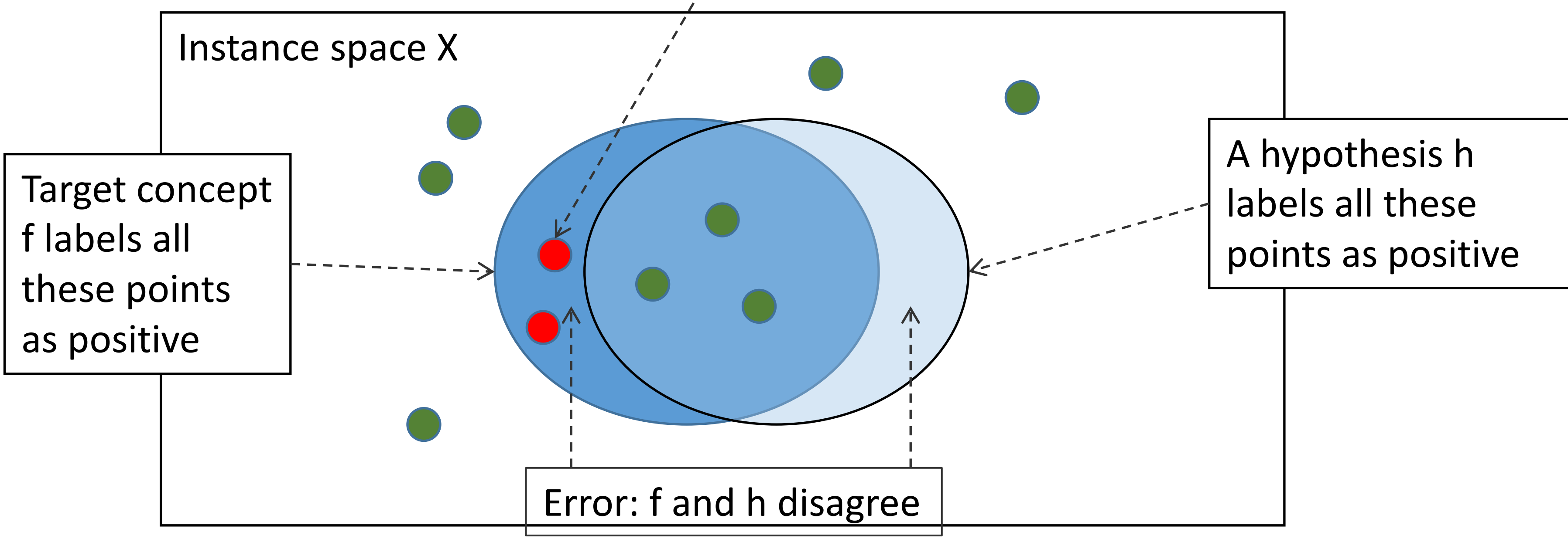
Intuition of PAC Learnability

With the IID sampling assumption, if a concept is too complicated. We need to see exponential number of samples, such that we can rule out those red points



Intuition of PAC Learnability

If a concept is simple:



$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Recap: Learning Conjunctions

- Protocol 1:
  Teacher provides a set of example (x, f(x))
  - <(1,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0>
  - <(1,1,1,1,1,0,...0,1,1), 1>
  - <(1,0,1,1,1,0,...0,1,1), 0>
  - <(1,1,1,1,1,0,...0,0,1), 1>
  - <(1,0,1,0,0,0,...0,1,1), 0>
  - <(1,1,1,1,1,1,…,0,1), 1>
  - <(0,1,0,1,0,0,...0,1,1), 0>

What would f look like?

Whenever the output is 1, x_1 is present

With the given data, we only learned an *approximation* to the true concept. Is it good enough?

Recap: Learning Conjunctions: Analysis

Theorem: Suppose we are learning a conjunctive concept with n dimensional Boolean features using m training examples. If

$$m > \frac{n}{\epsilon}\left(\log(n) + \log\left(\frac{1}{\delta}\right)\right)$$

then, with probability > 1 - δ, the error of the learned hypothesis $err_D(h)$ will be less than ϵ.

A general result

Let H be any hypothesis space.

With probability $1 - \delta$ a hypothesis $h \rightarrow H$ that is consistent with a training set of size m will have an error < ϵ on future examples if

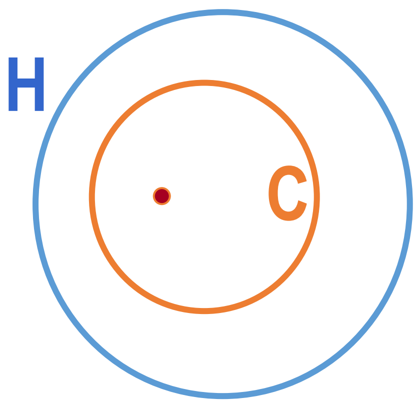
$$m > \boxed{\frac{1}{\epsilon}}\left(\boxed{\ln(|H|)} + \boxed{\ln\frac{1}{\delta}}\right)$$

1. Expecting lower error increases sample complexity (i.e more examples needed for the guarantee)

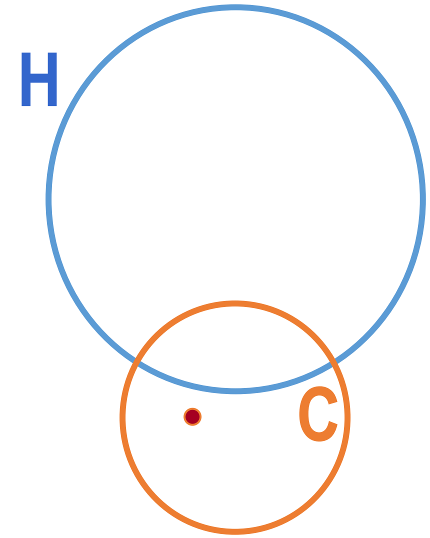
2. If we have a larger hypothesis space, then we will make learning harder (i.e higher sample complexity)

3. If we want a higher confidence in the classifier we will produce, sample complexity will be higher.

What if the concept space is different from the hypothesis space?



It is fine, we can still find the right function



The training error will not be zero

Agnostic Learning

1. An agnostic learner makes no commitment to whether f is in H and returns the hypothesis with least training error over at least m examples.

   It can guarantee with probability 1 - ϵ that the training error is *not* off by more than ϵ from the training error if

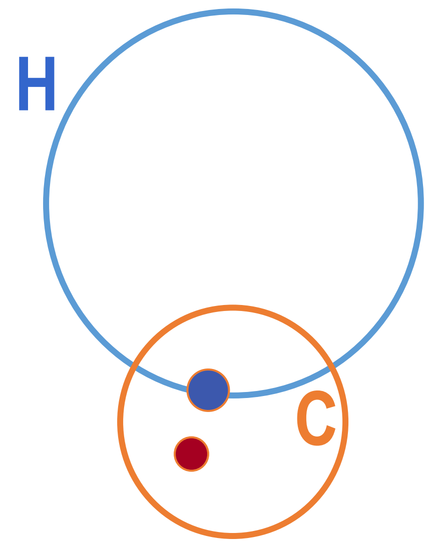
$$m \geq \frac{1}{2\epsilon^2}\left[\ln|H| + \ln\left(\frac{1}{\delta}\right)\right]$$

Agnostic Learning

An agnostic learner makes no commitment to whether f is in H and returns the hypothesis with least training error over at least m examples.

It can guarantee with probability 1 - ϵ that the training error is *not* off by more than ϵ from the training error if

$$m \geq \frac{1}{2\epsilon^2}\left[\ln|H| + \ln\left(\frac{1}{\delta}\right)\right]$$



Generalization bound

A bound on how much the true error will deviate from the training error. If we have more than m examples, then with high probability $1 - \delta$

$$err_D(h) - err_S(h) \leq \sqrt{\frac{\ln |H| + \ln(1/\delta)}{2m}}$$

Generalization error   Training error

Generalization bound

A bound on how much the true error will deviate from the

Now, we know if size(H) is finite, we can define what is learnable. This works for Boolean functions.

Next question: What if size(H) is infinity?

This lecture: Computational Learning Theory

- The Theory of Generalization
- Probably Approximately Correct (PAC) learning
- Shattering and the VC dimension

Infinite Hypothesis Space

- The previous analysis was restricted to finite hypothesis spaces
- Some infinite hypothesis spaces are more expressive than others
  - Linear threshold function vs. a combination of LTUs
- Need a measure of the expressiveness of an infinite hypothesis space other than its size

A general result

If $|H|$ is infinite, m is always infinite as well.

$$m > \boxed{\frac{1}{\epsilon}}\left(\boxed{\ln(|H|)} + \boxed{\ln\frac{1}{\delta}}\right)$$

1. Expecting lower error increases sample complexity (i.e more examples needed for the guarantee)

2. If we have a larger hypothesis space, then we will make learning harder (i.e higher sample complexity)

3. If we want a higher confidence in the classifier we will produce, sample complexity will be higher.

Vapnik-Chervonenkis dimension

- The Vapnik-Chervonenkis dimension (VC dimension) provides such a measure
  - "What is the expressive *capacity* of a set of functions?"
- Analogous to |H|, there are bounds for sample complexity using VC(H)

VC dimension and consistent learners

- Using VC(H) as a measure of expressiveness we have a sample complexity bound for infinite hypothesis spaces
- Given a sample D with m examples,

  find some h → H is consistent with all m examples. If

$$m > \frac{1}{\epsilon}\left(8VC(H)\log\frac{13}{\epsilon} + 4\log\frac{2}{\delta}\right)$$

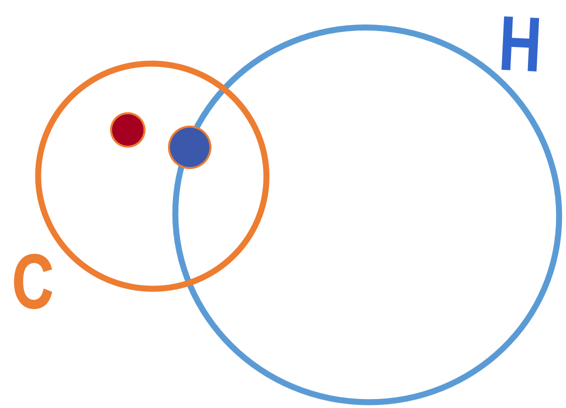
Then with probability at least (1-δ), h has error less than ε.

You don't need to remember this equation
but just need to understand the meaning

Generation bound for agnostic learner

If we have m examples, then with probability 1 - δ, a the true error of a hypothesis h with training error $err_S(h)$ is bounded by

$$err_D(h) \leq err_S(h) + \sqrt{\frac{VC(H)\left(\ln \frac{2m}{VC(H)} + 1\right) + \ln \frac{4}{\delta}}{m}}$$



Intuition of VC dimension

- Although there are infinitely many hypotheses, many of them are similar

- The idea of learning is by eliminating incorrect hypotheses
  - We can eliminate infinite # hypotheses for each training sample

Recap: Learning Conjunctions

- Protocol 1:
  Teacher provides a set of example (x, f(x))
  - <(1,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0>
  - <(1,1,1,1,1,0,...0,1,1), 1>
  - <(1,0,1,1,1,0,...0,1,1), 0>
  - <(1,1,1,1,1,0,...0,0,1), 1>
  - <(1,0,1,0,0,0,...0,1,1), 0>
  - <(1,1,1,1,1,1,…,0,1), 1>
  - <(0,1,0,1,0,0,...0,1,1), 0>

$x_1 \wedge x_2 \wedge x_3 \ldots \ldots x_{99} \wedge x_{100}$

$x_1 \wedge x_2 \wedge x_3 \ldots \ldots x_{99}$

$x_1 \wedge x_3 \ldots \ldots x_{99} \wedge x_{100}$

$x_2 \wedge x_3 \ldots \ldots x_{99} \wedge x_{100}$

$x_2 \wedge x_3 \ldots \ldots x_{99}$

$x_3 \ldots \ldots x_{99} \wedge x_{100}$

Recap: Learning Conjunctions

- Protocol 1:
  Teacher provides a set of example (x, f(x))
  - <(0,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0>
  - <(1,1,1,1,1,0,...0,1,1), 1>
  - <(1,0,1,1,1,0,...0,1,1), 0>
  - <(1,1,1,1,1,0,...0,0,1), 1>
  - <(1,0,1,0,0,0,...0,1,1), 0>
  - <(1,1,1,1,1,1,…,0,1), 1>
  - <(0,1,0,1,0,0,...0,1,1), 0>

$x_1 \wedge x_2 \wedge x_3 \dots \dots x_{99} \wedge x_{100}$

$x_1 \wedge x_2 \wedge x_3 \dots \dots x_{99}$

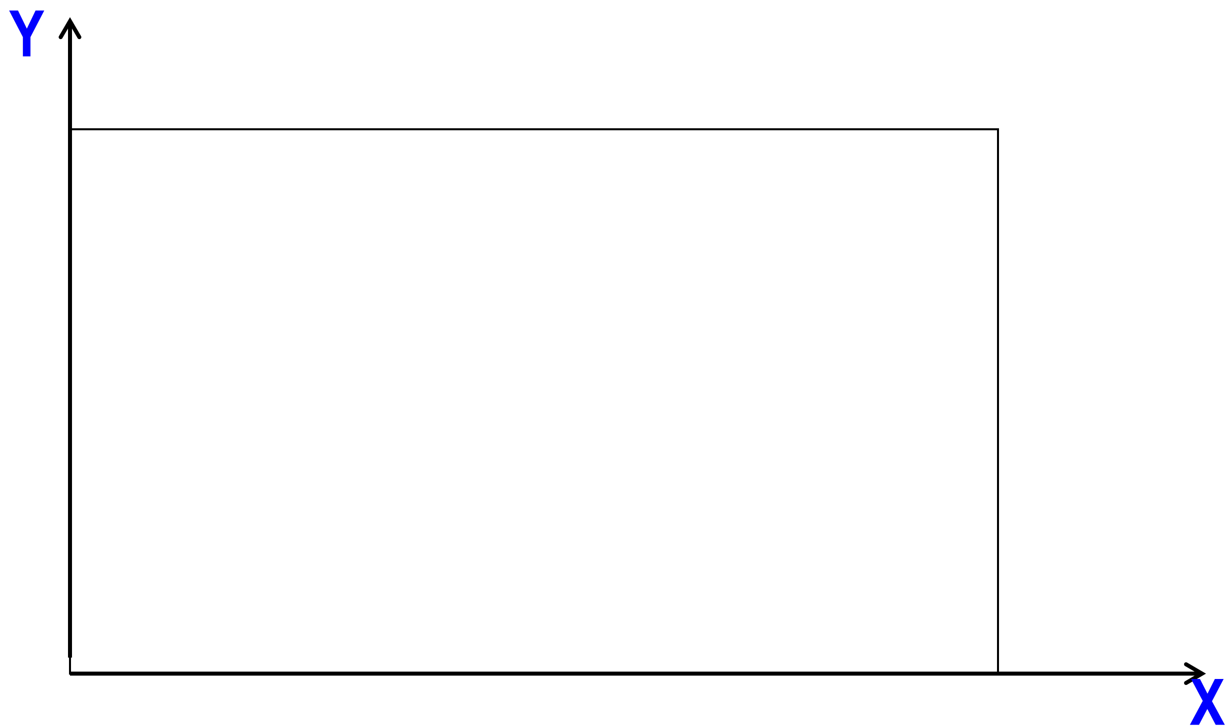
$x_1 \wedge x_3 \dots \dots x_{99} \wedge x_{100}$

$x_2 \wedge x_3 \dots \dots x_{99} \wedge x_{100}$

$x_2 \wedge x_3 \dots \dots x_{99}$

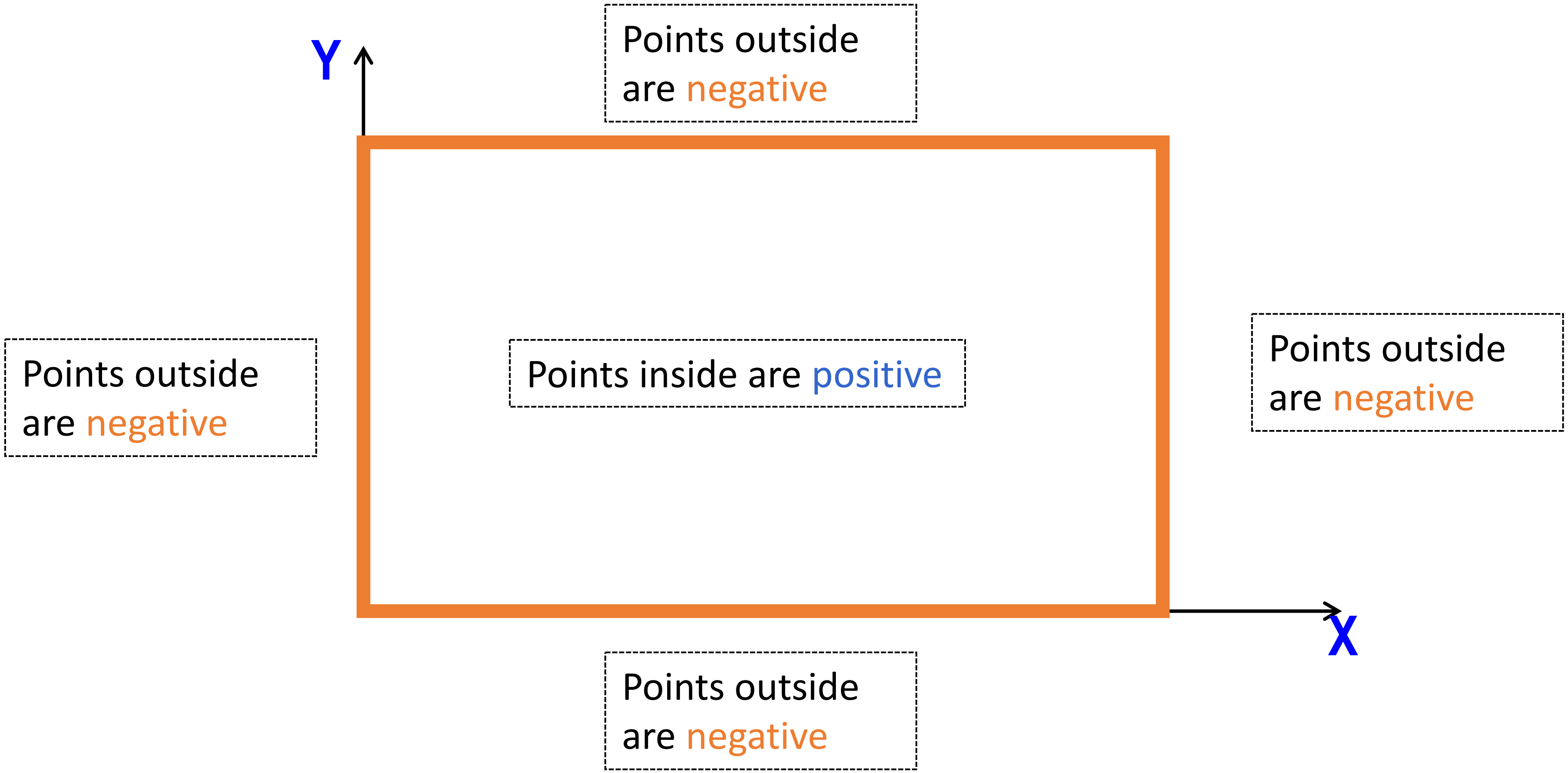
$x_3 \dots \dots x_{99} \wedge x_{100}$

Intuition of VC dimention: Learning Rectangles



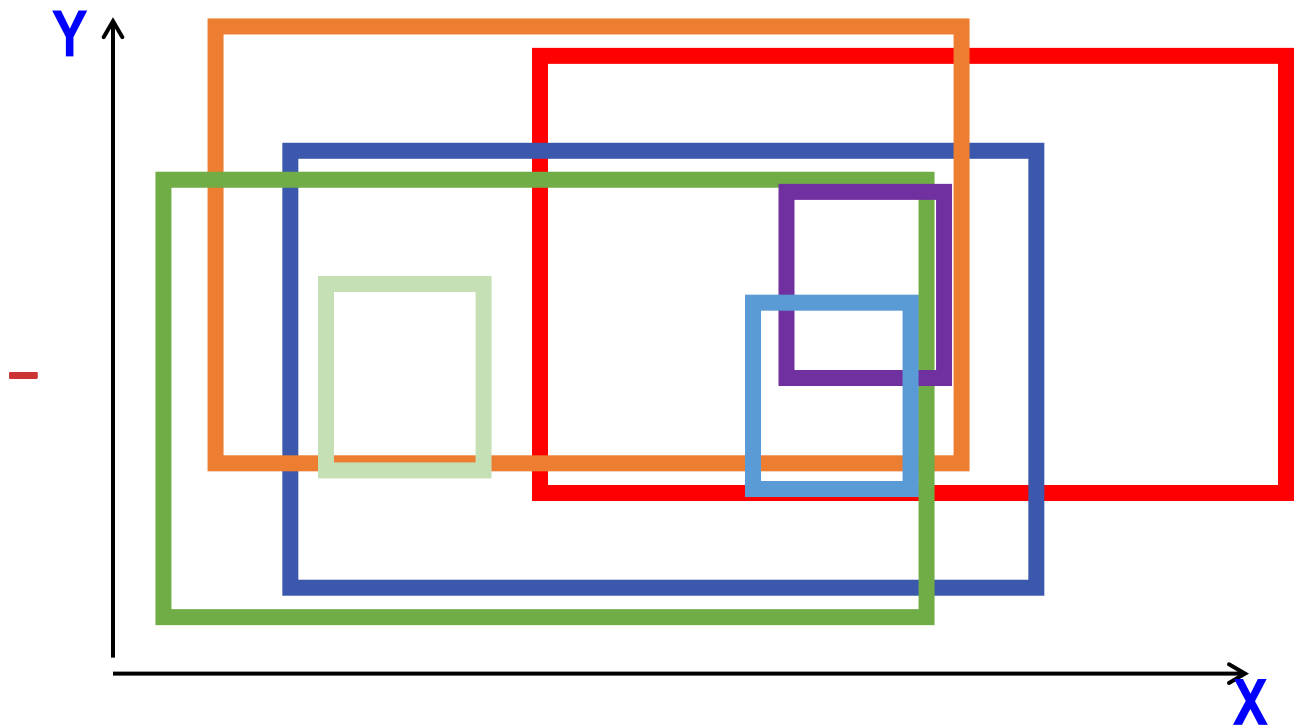
Assume the target concept is an axis parallel rectangle

Learning Rectangles



Learning Rectangles

Assume the target concept is an axis parallel rectangle

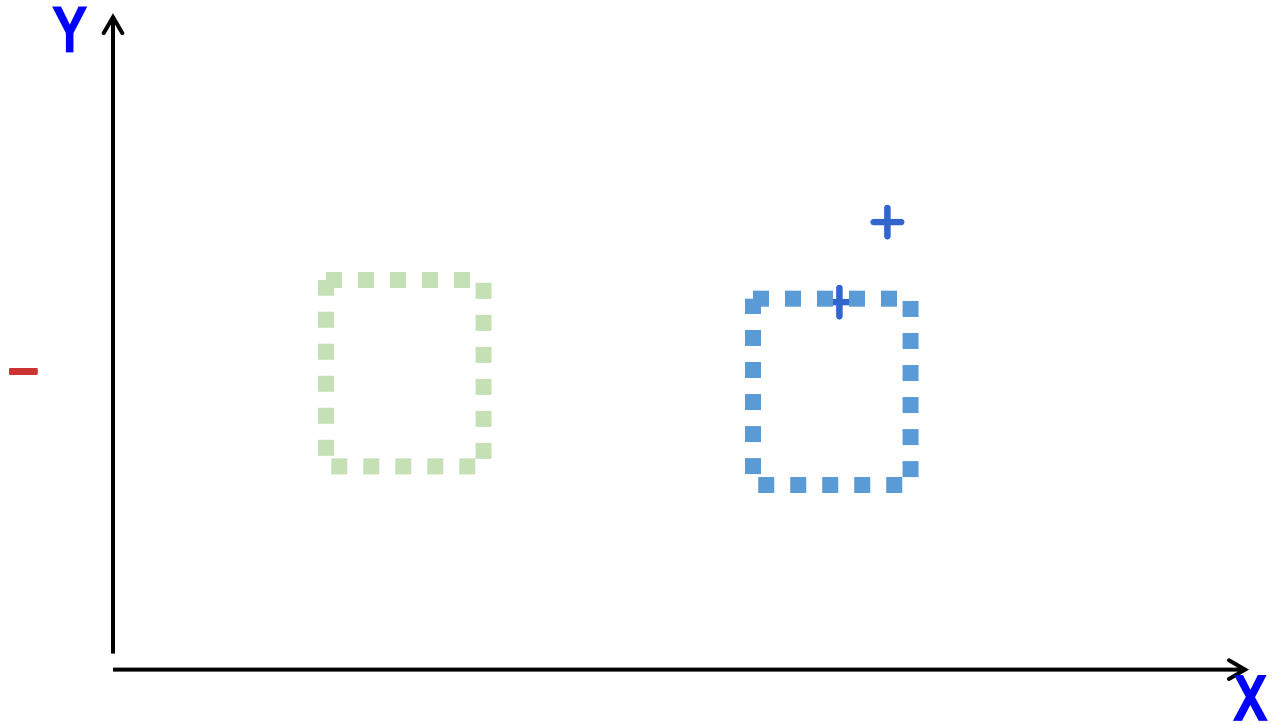


Learning Rectangles

Assume the target concept is an axis parallel rectangle

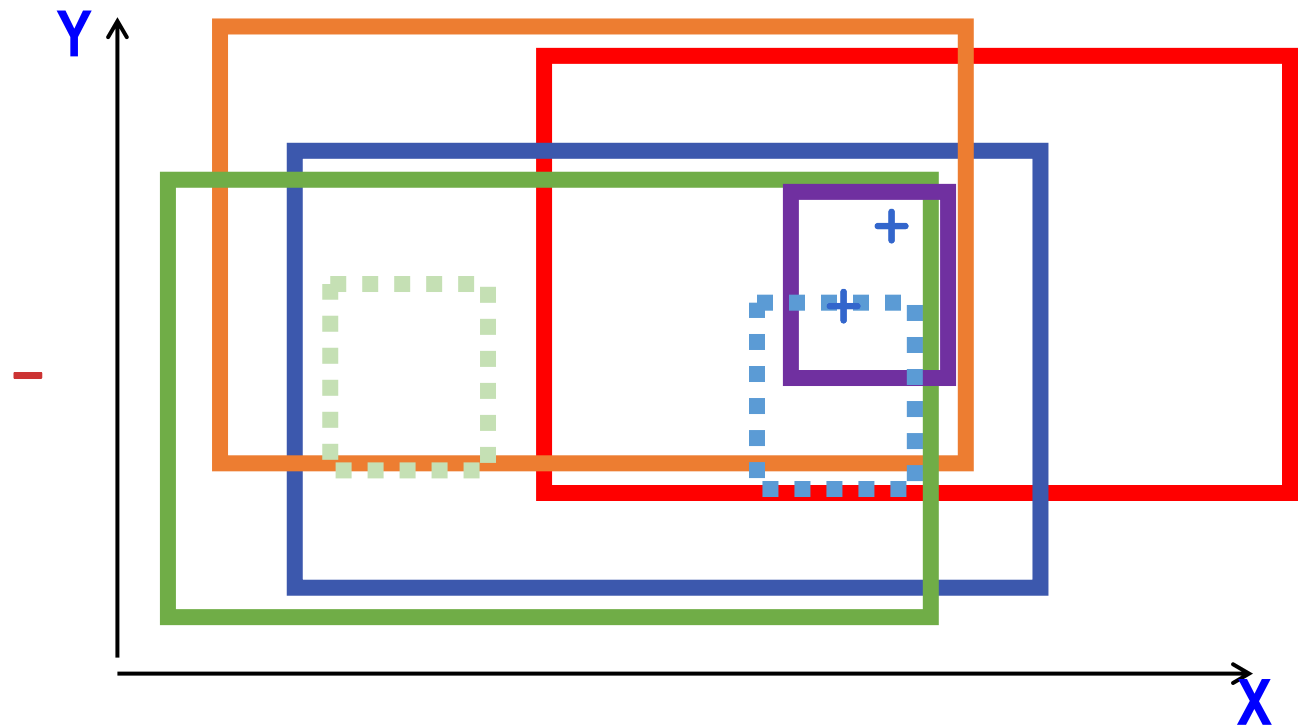
Learning Rectangles

Assume the target concept is an axis parallel rectangle



Learning Rectangles

Assume the target concept is an axis parallel rectangle

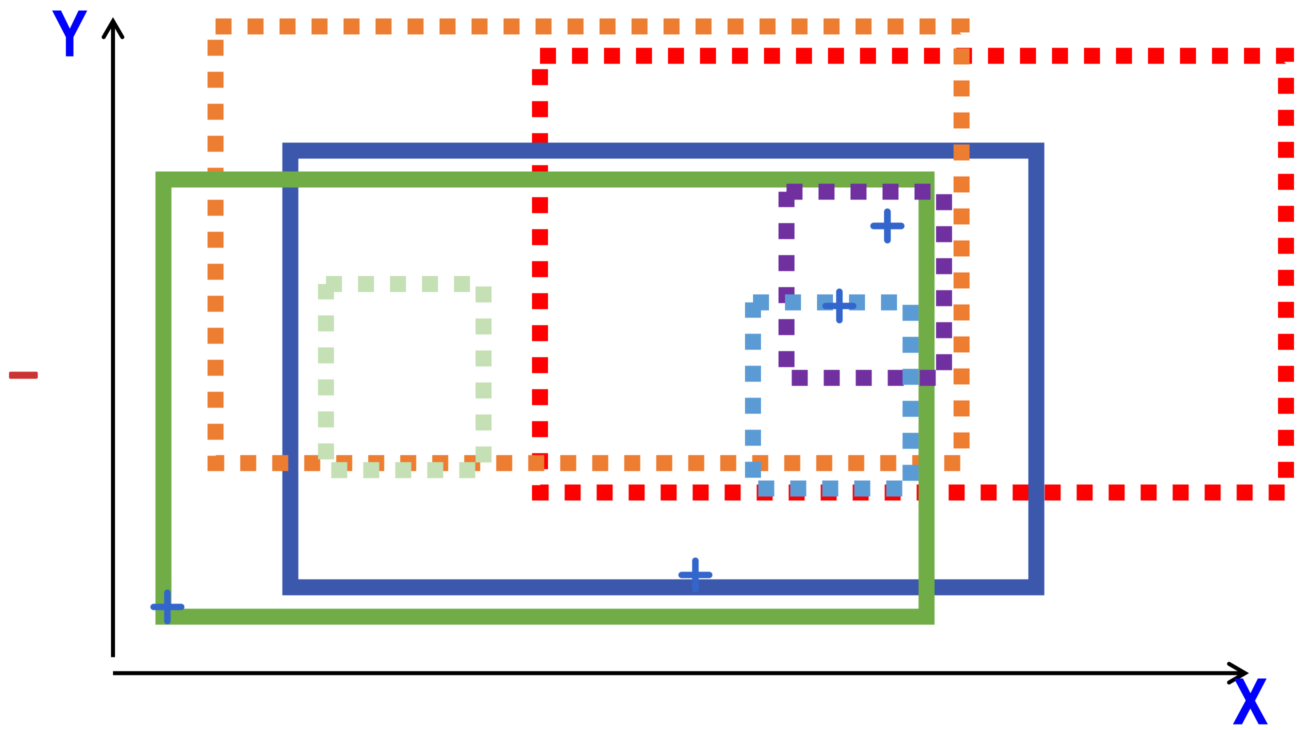


Learning Rectangles

Assume the target concept is an axis parallel rectangle

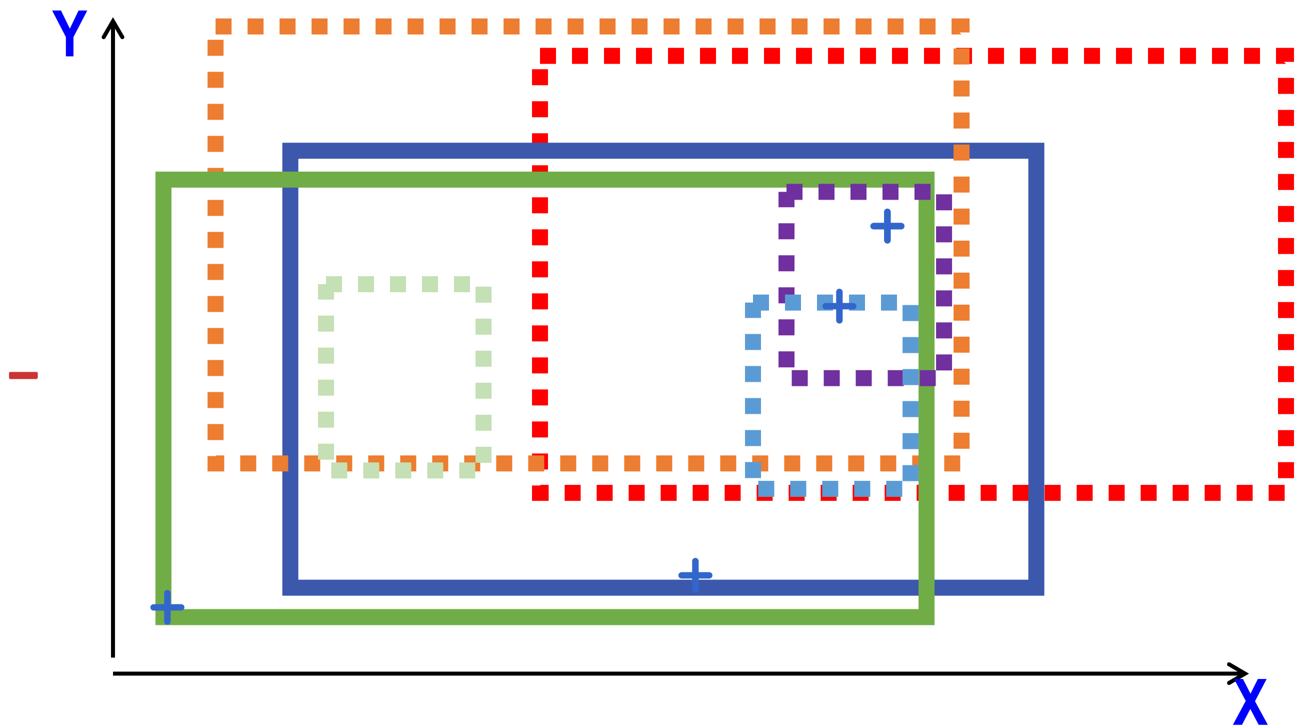
Learning Rectangles

Assume the target concept is an axis parallel rectangle



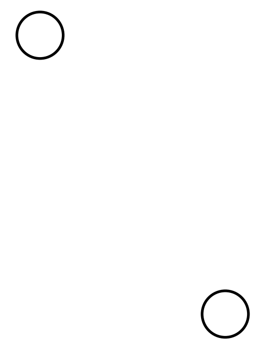
Learning Rectangles

Assume the target concept is an axis parallel rectangle



Key observation: Despite there are infinite # hypothesis
The blue & red rectangles have the same predictions

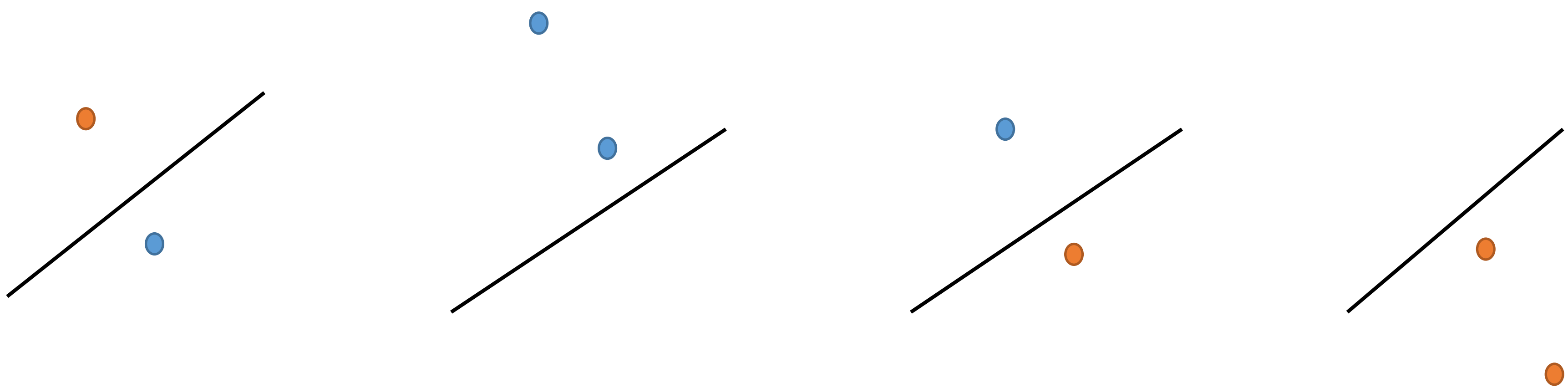
Let’s think about expressivity of functions



Suppose we have two points.
Can linear classifiers correctly classify any labeling of these points?

Linear functions are expressive enough to ***shatter*** 2 points

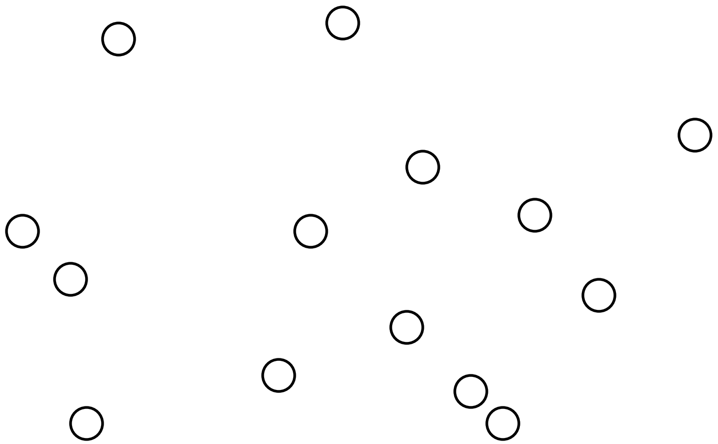
Let's think about expressivity of functions



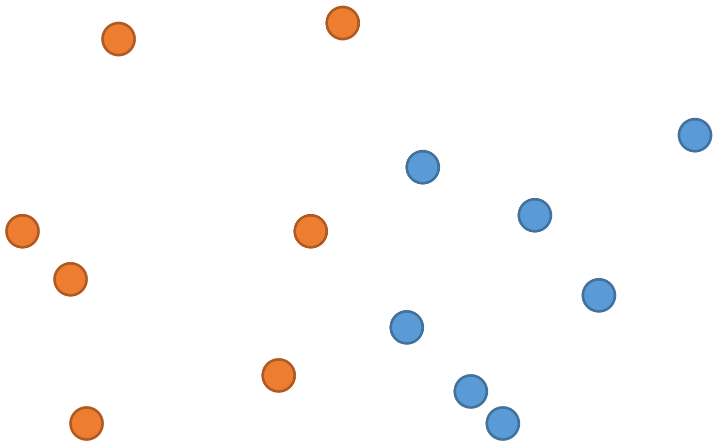
Suppose we have two points.
Can linear classifiers correctly classify any labeling of these points?

Linear functions are expressive enough to ***shatter*** 2 points

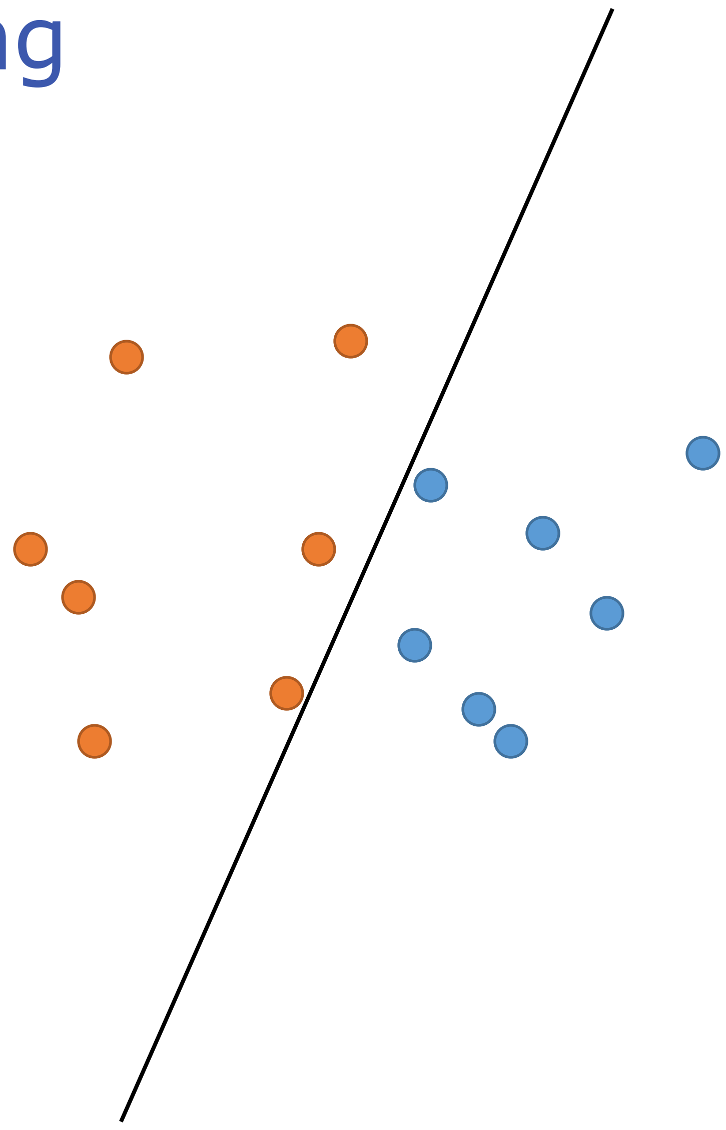
Shattering



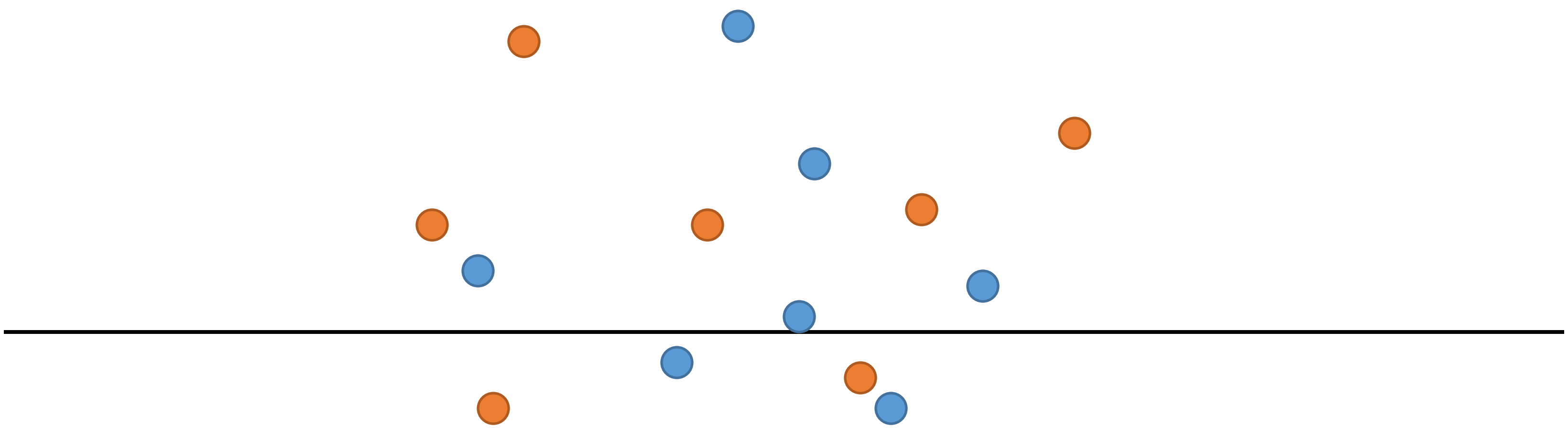
Shattering



Shattering

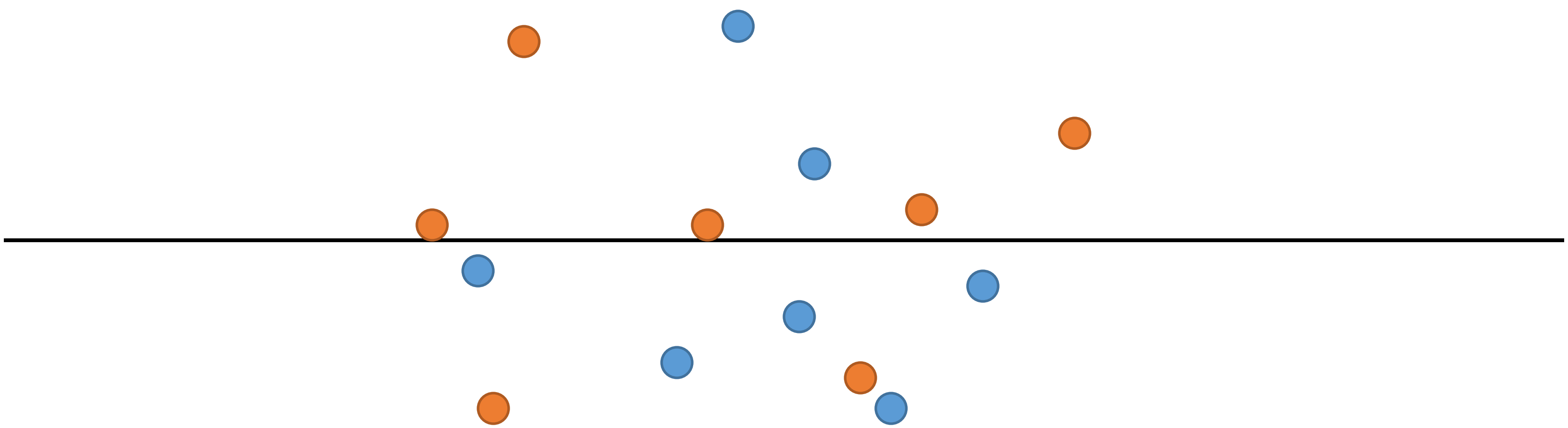


Shattering



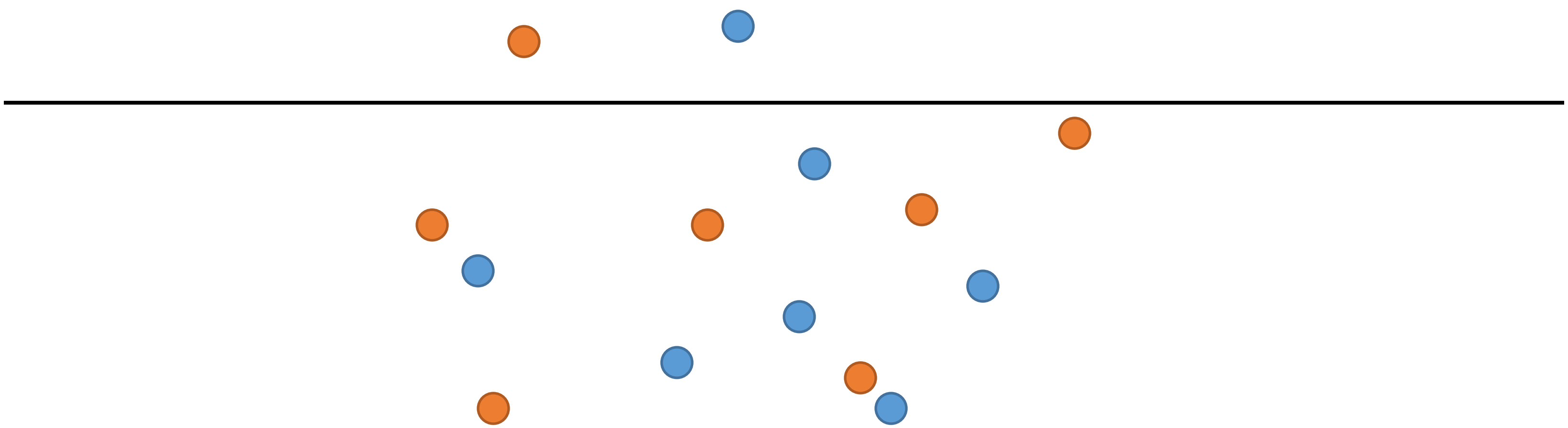
This particular labeling of the points can not be separated by *any* line

Shattering



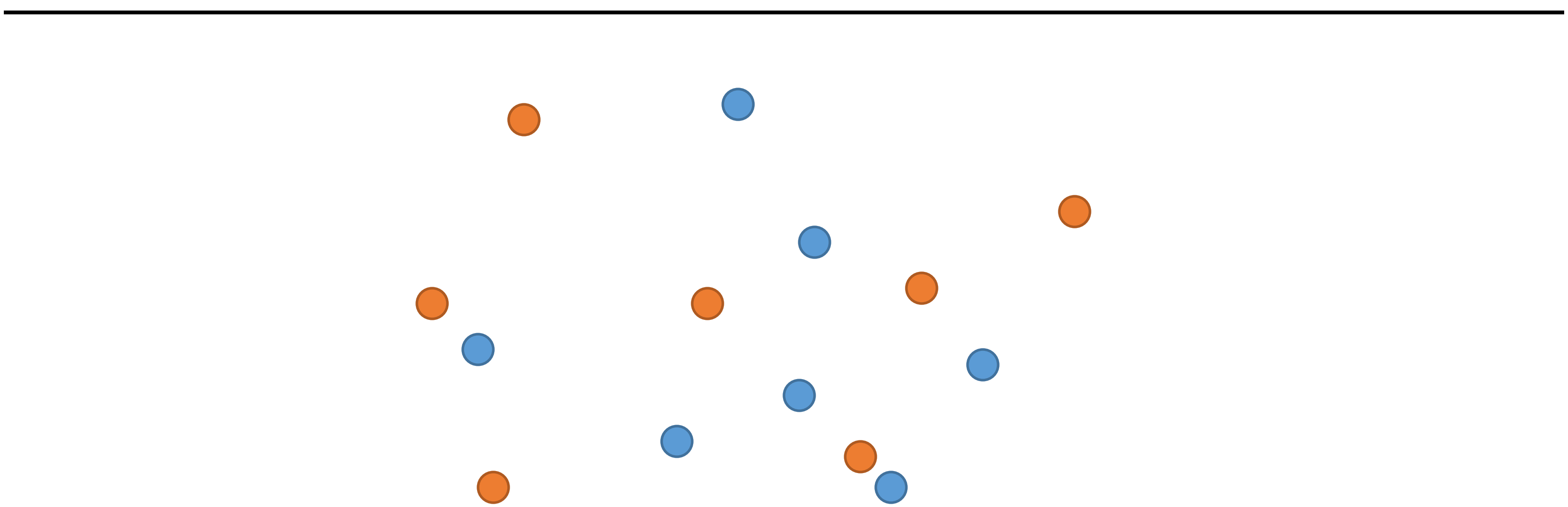
This particular labeling of the points can not be separated by *any* line

Shattering



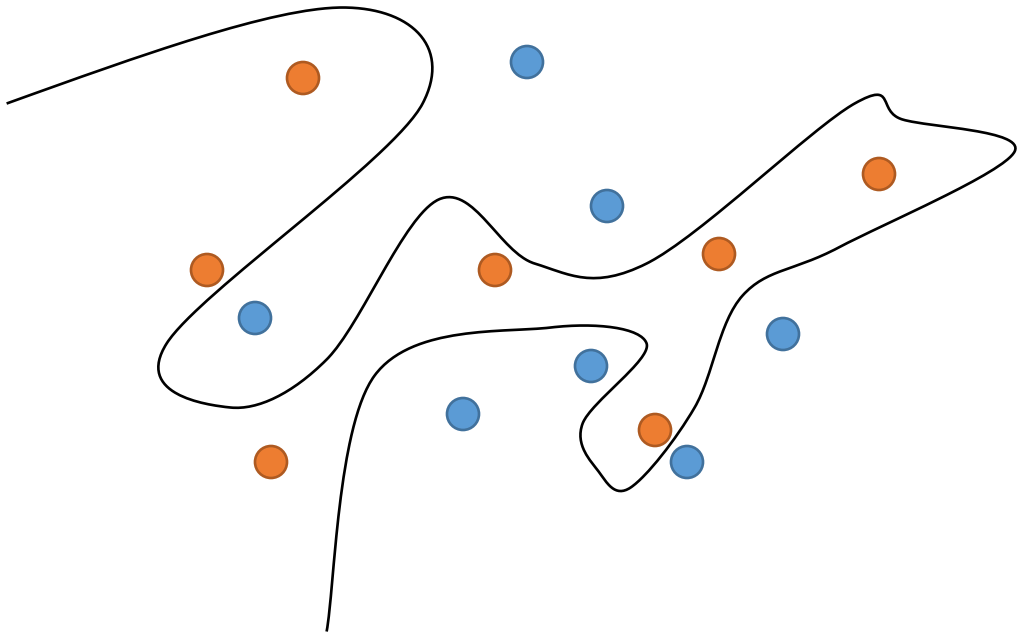
This particular labeling of the points can not be separated by *any* line

Shattering



This particular labeling of the points can not be separated by *any* line

Shattering



Linear functions are not expressive to shatter fourteen points
Because there is a labeling that can not be separated by them

Of course, a more complex function could separate them

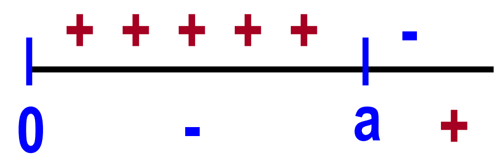
Shattering

Definition: A *set S of examples* is shattered by a *set of functions* H if for *every* partition of the examples in S into positive and negative examples there is a function in H that gives exactly these labels to the examples

Intuition: A rich set of functions shatters large sets of points

Left bounded intervals

Example 1: Hypothesis class of left bounded intervals on the real axis: [0,a) for some real number a>0

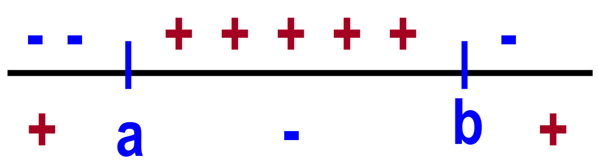


Sets of **two** points cannot be shattered

That is: given two points, you can label them in such a way that no concept in this class will be consistent with their labeling

Real intervals

Example 2: Hypothesis class is the set of intervals on the real axis: [a,b],for some real numbers b>a

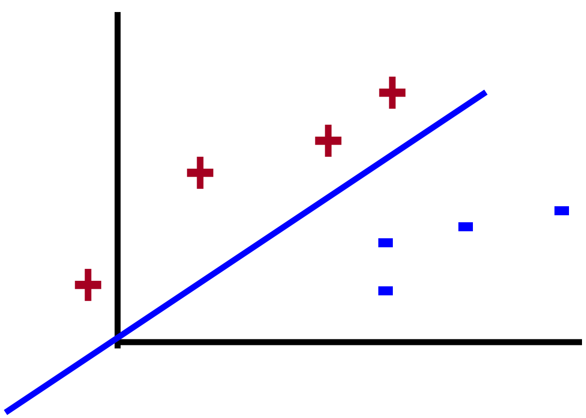


All sets of one or two points can be shattered
But some sets of three points cannot be shattered

Shattering

Definition: A *set S of examples* is shattered by a *set of functions* H if for *every* partition of the examples in S into positive and negative examples there is a function in H that gives exactly these labels to the examples

Example 3: 2-D Half spaces in a plane



Can one point be shattered?

Is there any two points can be shattered?

Is there any three points?
Can any three points be shattered?

Vapnik-Chervonenkis Dimension

Definition: The VC dimension of hypothesis space H over instance space X is the size of the largest *finite* subset of X that is shattered by H

- If there exists any subset of size d that can be shattered, VC(H) >= d
  - Even one subset will do
- If no subset of size d can be shattered, then VC(H) < d

Shattering: The adversarial game

You

An adversary

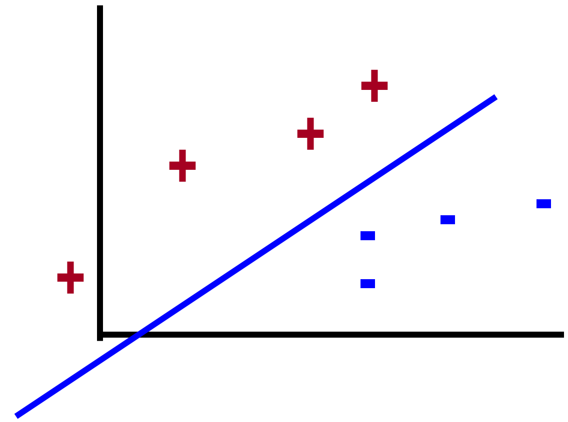
You: Hypothesis class H can shatter these d points

Adversary: That's what you think! Here is a labeling that will defeat you.

You: Aha! There is a function h ∈ H that correctly predicts your evil labeling

Adversary: Argh! You win this round. But I'll be back.....

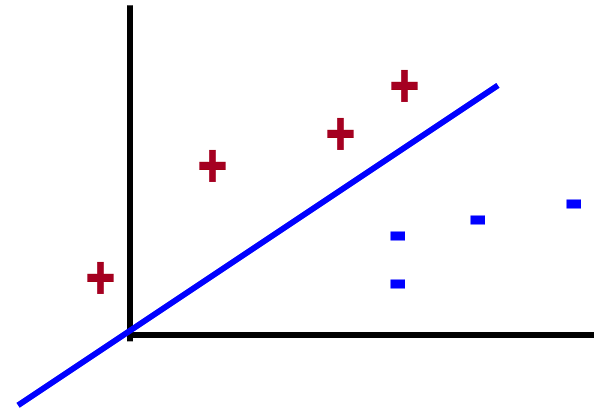
Example Half spaces in a plane



- Prove VC >=1
  - Show any point can be shattered
- Prove VC >=2
  - Show there exists 2 points can be shattered
- Prove VC >=3
  - Show there exists 3 points can be shattered

Example Half spaces in a plane

- Prove VC <4
  - Show no 4 points can be shattered
- Therefore, VC = 3



- Suppose three of them lie on the same line, label the outside points + and the inner one –
- Other wise, make a convex hull. Label points outside + and the inner one –
- Four points cannot be shattered!

VC dimension of Half spaces

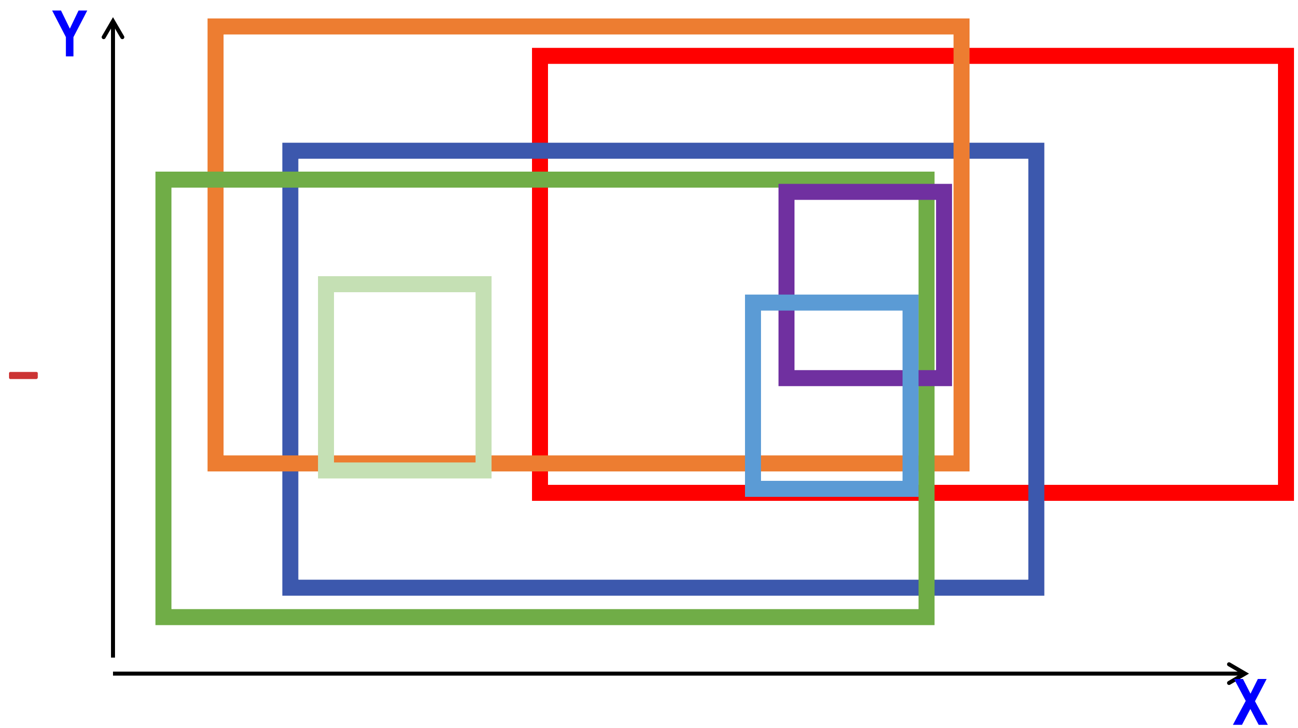
- In general, the VC dimension of an n-dimensional linear function is n+1
- Give the same δ and m

This term may decrease

This term will decrease

$$err_D(h) \leq err_S(h) + \sqrt{\frac{VC(H)\left(\ln \frac{2m}{VC(H)} + 1\right) + \ln \frac{4}{\delta}}{m}}$$

Exercise



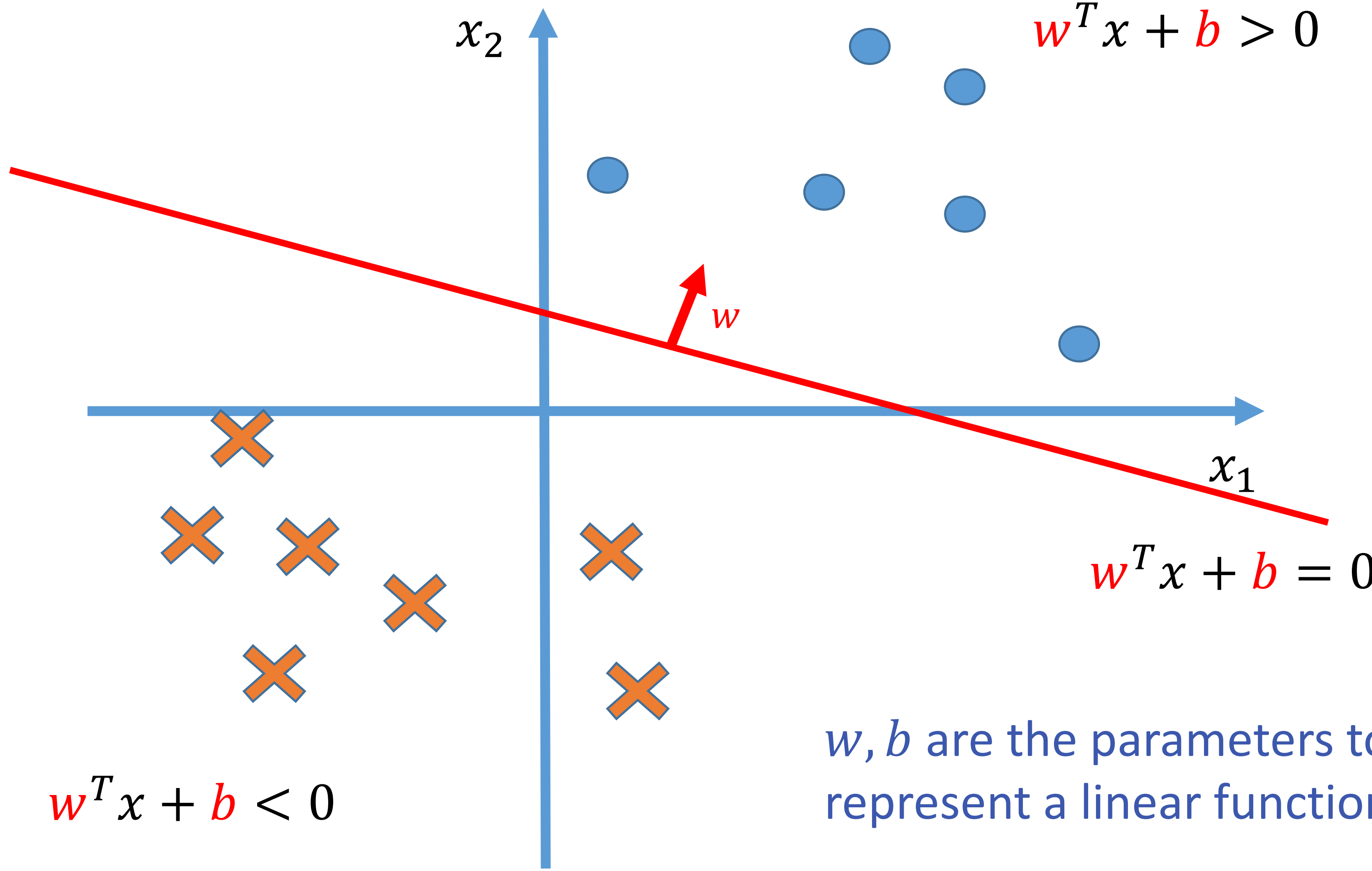
What is the VC dimension for the rectangle concept space?

Computational Learning Theory

- The Theory of Generalization
  - Using training instance to rule out incorrect hypotheses
- Probably Approximately Correct (PAC) learning
  - How many examples you need to see to obtain a learned function with error $\leq \epsilon$
- Shattering and the VC dimension

Kernel and Kernel methods

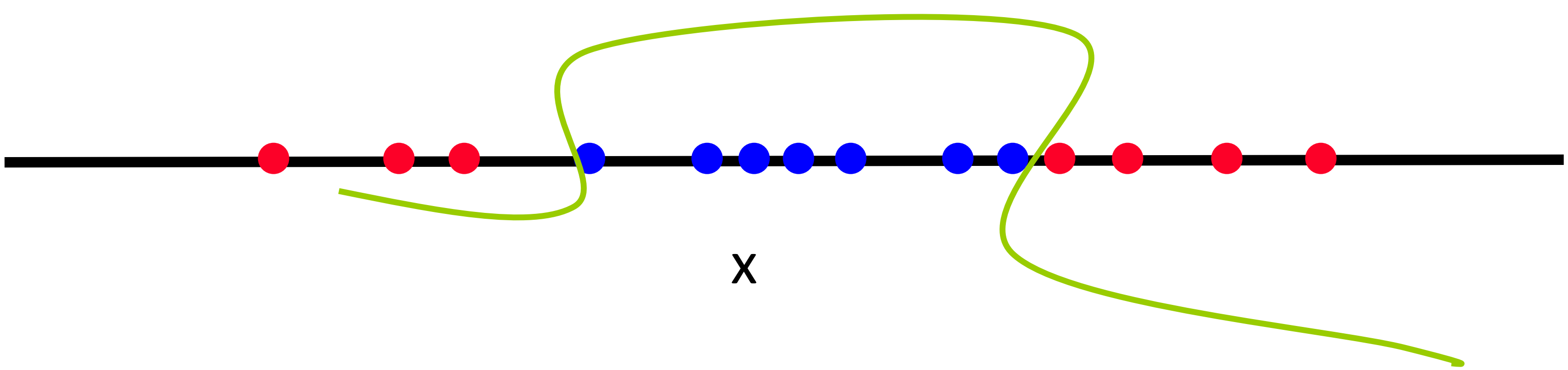
Hypothesis space: linear model



w, b are the parameters to represent a linear function

Functions Can be Made Linear

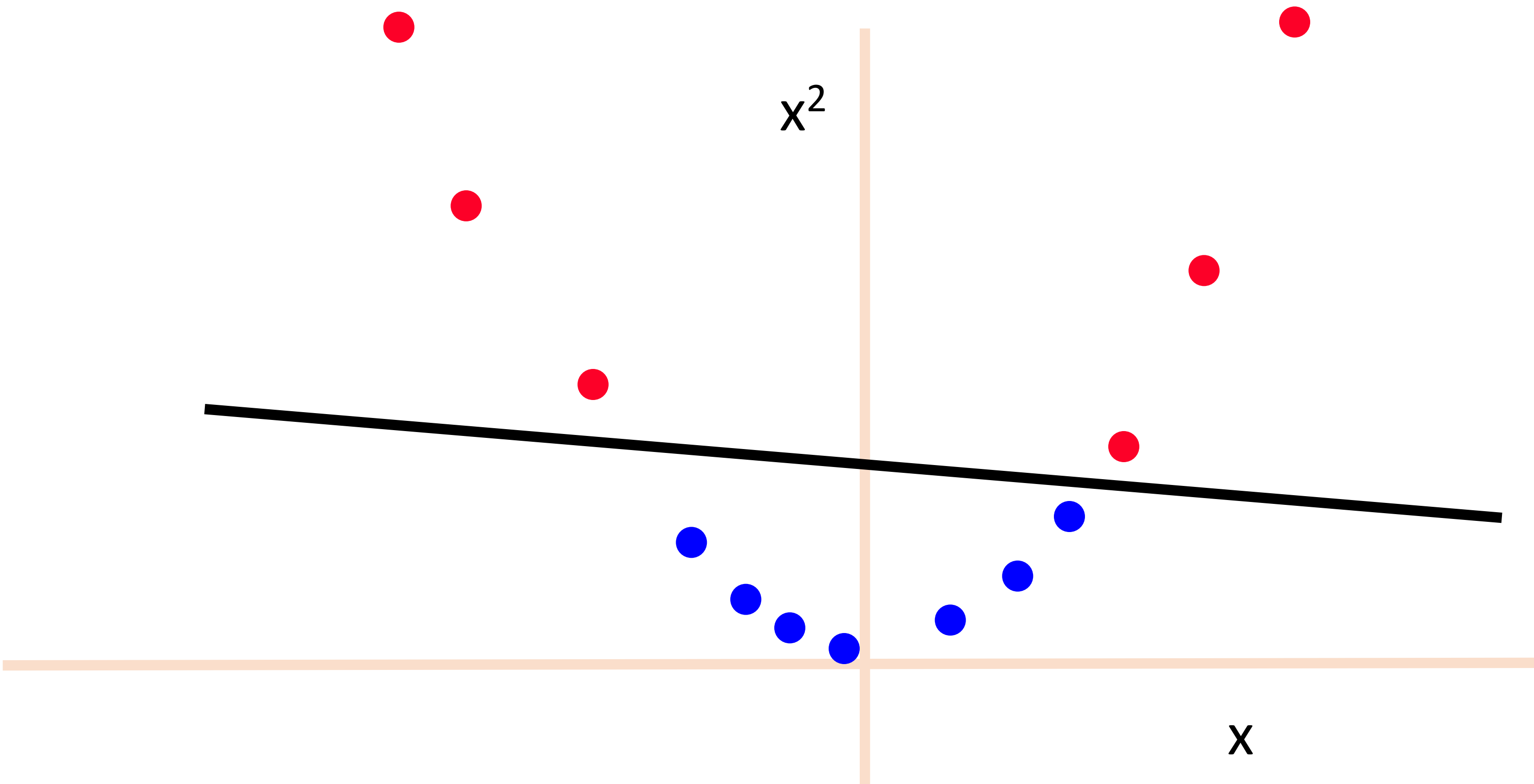
- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions



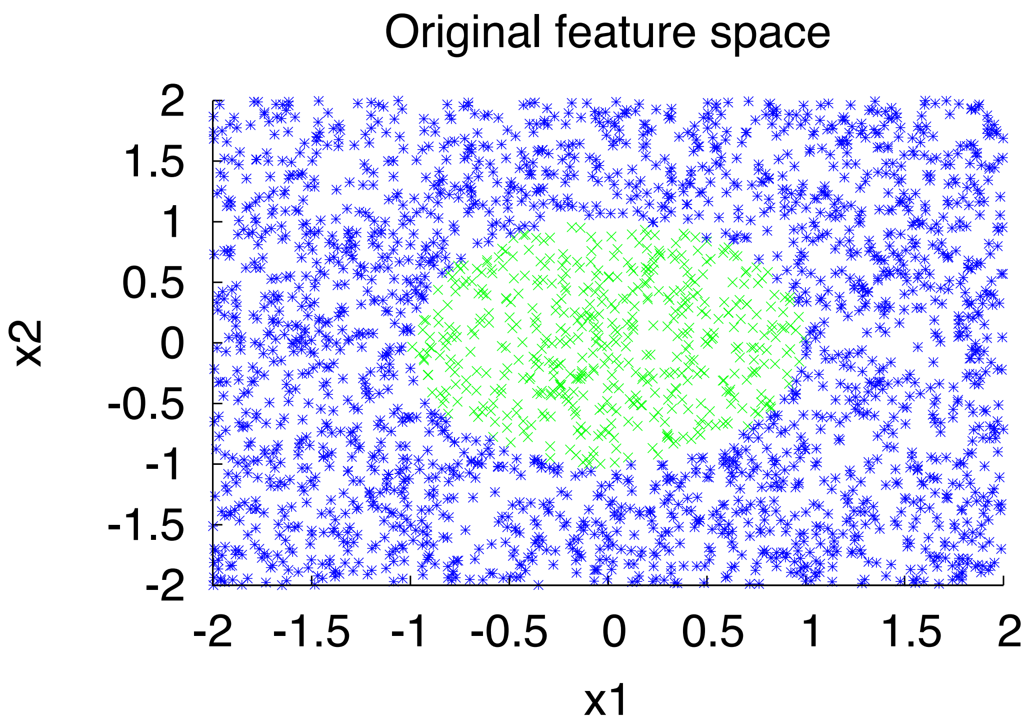
Can we do some mapping to make it linear spreadable?

Blown Up Feature Space

- Data are separable in <x, x^2> space

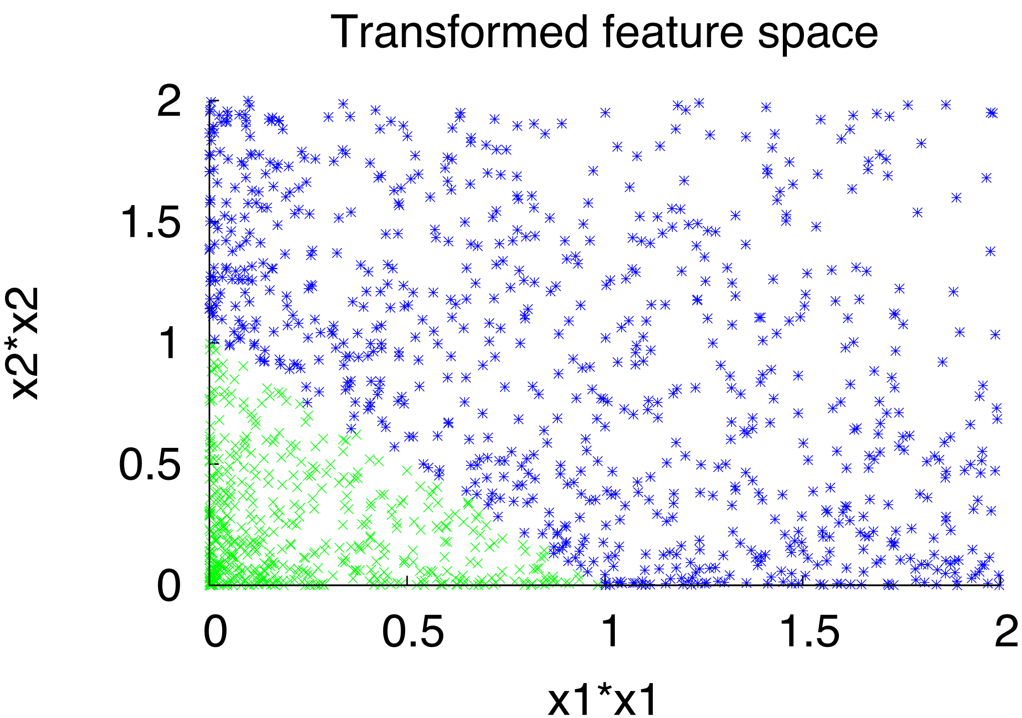


Making data linearly separable



$$f(\mathbf{x}) = 1 \text{ iff } x_1^2 + x_2^2 \leq 1$$

Making data linearly separable



Transform data: $\mathbf{x} = (x_1, x_2)$ => $\mathbf{x'} = (x_1^2, x_2^2)$

$f(\mathbf{x'}) = 1$ iff $x'_1 + x'_2 \leq 1$

The Perceptron Algorithm [Rosenblatt 1958]

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\boldsymbol{w} \leftarrow \boldsymbol{0} \in \mathbb{R}^n$
2. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
3. if $y(\boldsymbol{w}^\top \boldsymbol{x}) \leq \boldsymbol{0}$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$
5.
6. Return $\boldsymbol{w}$

Assume $y \in \{1, -1\}$

Prediction: $y^{\text{test}} \leftarrow sgn(\boldsymbol{w}^\top \boldsymbol{x}^{\text{test}})$

The Perceptron Algorithm [Rosenblatt 1958]

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

Initialize $\boldsymbol{w} \leftarrow \boldsymbol{0} \in \mathbb{R}^{2n}$

For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:

if $y\, \boldsymbol{w}^T \begin{bmatrix} x \\ x^2 \end{bmatrix} \leq \boldsymbol{0}$

$\boldsymbol{w} \leftarrow \boldsymbol{w} + y \begin{bmatrix} x \\ x^2 \end{bmatrix}$

Assume $y \in \{1, -1\}$

What if our mapping function is more complex?

Prediction: $y^{\text{test}} \leftarrow \text{sgn}(\boldsymbol{w}^{\mathsf{T}} \begin{bmatrix} x \\ x^2 \end{bmatrix})$

The Perceptron Algorithm [Rosenblatt 1958]

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\boldsymbol{w} \leftarrow \boldsymbol{0} \in \mathbb{R}^n$
2. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
3. if $y(\boldsymbol{w}^\top \boldsymbol{x}) \leq \boldsymbol{0}$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$
5.
6. Return $\boldsymbol{w}$

Assume $y \in \{1, -1\}$

Observation: w is a combination of the input instances!!

Prediction: $y^{\text{test}} \leftarrow sgn(\boldsymbol{w}^\top \boldsymbol{x}^{\text{test}})$

Dual Representation

$$\text{if } y(\boldsymbol{w}^\top \boldsymbol{x}) \leq \boldsymbol{0}$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$$

- Let w be an initial weight vector for perceptron. Let (x_1,+), (x_2,+), (x_3,-), (x_4,-) be examples and assume mistakes are made on x_1, x_2 and x_4.
- What is the resulting weight vector?

  w = w + x_1 + x_2 - x_4

- In general, the weight vector w can be written as a linear combination of examples:

$$w = \sum_{1..m} \alpha_i \, y_i x_i$$

- Where α_i is the number of mistakes made on x_i.

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- That is, we just showed that

$$\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x}$$

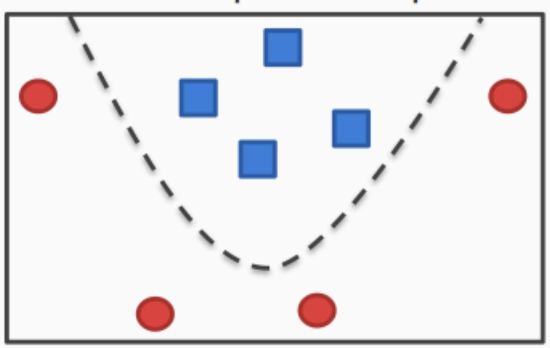
  - We only need to compute dot products between training examples and the new example **x**
- This is true even if we map examples to a high dimensional space

$$\mathbf{w}^T\phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$$

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



Transform all input points as

$$\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using $\mathbf{w}^T\phi(x_1, x_2) \geq b$

Many learning algorithm require to compute inner products

- Perceptron:

$$y(\boldsymbol{w}^{\top}\boldsymbol{x}) \leq \boldsymbol{0}$$

- K-NN:

$$similarity\left(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{neighbor}}\right) = \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x}^{\boldsymbol{neighbor}}$$

$$dist\left(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{neighbor}}\right) = ||\boldsymbol{x} - \boldsymbol{x}^{\boldsymbol{neighbor}}||^{2}$$

$$dist\left(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{neighbor}}\right) = ||x||^{2} + \left||x^{\boldsymbol{neighbor}}\right||^{2} \text{-} \boldsymbol{2}\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x}^{\boldsymbol{neighbor}}$$

Is there a smarter way to compute the inner product?

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

So prediction with this *high dimensional lifting map* is

$$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

because $\mathbf{w}^T \phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

So prediction with this *high dimensional lifting map* is

If we can compute the value of K without explicitly writing the blown up representation, then we will have a computational advantage.

because $\mathbf{w}^T \phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

All degree zero terms

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, \underbrace{x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n}_{\text{All degree two terms}}]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, \underbrace{x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1}x_n}_{\text{All degree two terms}}]^T$$

and compute the dot product $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$

[takes time]

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product A $= \phi(\mathbf{x})^T \phi(\mathbf{z})$

[takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Theorem: A = B (Coefficients do not really matter)

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product A = $\phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Claim: Compute B instead of A (Coefficients do not really matter)

The Kernel Trick

Suppose we wish to compute

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

Here ϕ maps **x** and **z** to a high dimensional space

The Kernel Trick: Save time/space by computing the value of K(**x**, **z**) by performing operations in the original space (without a feature transformation!)

Which functions are kernels?

- Can we use any function K(.,.)?
  - No! A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

- General condition: construct the Gram matrix $\{K(\mathbf{x}_i, \mathbf{z}_j)\}$; check that it's positive semi definite

Example

- Let x and z are 2-dimentional vector, show that K(**x**, **z**) = $(1 + x^T z)^2$ is a valid kernel

- i.e., show that K(x, z) can be represented as ϕ(x)T ϕ (z) using some ϕ mapping.

Which functions are kernels?

- General condition: construct the Gram matrix $\{K(\mathbf{x}_i, \mathbf{z}_j)\}$; check that it's positive semi definite

A symmetric matrix M is positive semi-definite if it is
For any vector non-zero $\mathbf{z}$, we have $\mathbf{z}^T M \mathbf{z} \geq 0$

Mercer's condition

Let K(**x**, **z**) be a function that maps two n dimensional vectors to a real number

K is a valid kernel if for every finite set $\{x_1, x_2, \cdots\}$, for any choice of real valued $c_1, c_2, \cdots$, we have

$$\sum_i \sum_j c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Kernels that are commonly used

- Linear kernel: $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$

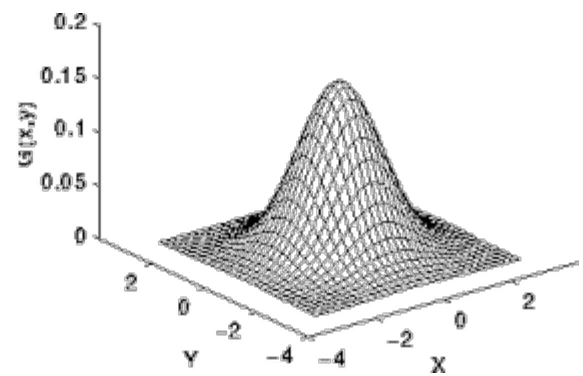
- Polynomial kernel up to degree *d*:
  $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$
  all interactions of order *d* or lower

Gaussian Kernel
(or the radial basis function kernel)

$$K_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{c}\right)$$

- $(x - z)^2$: squared Euclidean distance between **x** and **z**
- $c = \sigma^2$: a free parameter
- very small c: K ≈ identity matrix (every item is different)
- very large c: K ≈ unit matrix (all items are the same)

- k(**x**, **z**) ≈ 1 when **x**, **z** close
- k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar



Summary: Kernel trick

- To make the final prediction, we are computing dot products

- The kernel trick is a computational trick to compute dot products in higher dimensional spaces

- Important: All the bounds we have seen (e.g.: Perceptron bound, etc) depend on the underlying dimensionality
  - By moving to a higher dimensional space, we are incurring a penalty on sample complexity