Lecture 12: Evaluation

COMP 343, Spring 2022

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: These slides are based primarily on material from the book Machine Learning by Tom Mitchell (and associated slides), the book Machine Learning, An Applied Mathematics Introduction by Paul Wilmott, slides created by Vivek Srikumar (Utah), Dan Roth (Penn), Julia Hockenmaier (Illinois Urbana-Champaign), Jessica Wu (Harvey Mudd) and C. David Page (U of Wisconsin-Madison)

Today's Topics

Homework 5

- Due Friday, March 11 by 5p

Perceptron

- Variants
- Tips and tricks

Evaluation motivation

- Loss functions: motivation and usage for classification
- Bias and Variance
- Other metrics

vumanfredi@wesleyan.edu

Homework 5 discussion

What is cross-validation? Why use it

How to read through and use documentation?

Midterm: March 28 or March 30?

Perceptron

VARIANTS

Perceptron variants we've seen

1. Original perceptron algorithm → Homework 5
2. Standard perceptron algorithm
3. Voted perceptron algorithm
4. Standard perceptron algorithm
5. Margin perceptron algorithm
6. (Multi-class perceptron algorithm)

Original perceptron algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return final weight vector

Mistake can be written as $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$

Prediction on a new example with features $\mathbf{x}$: $sgn(\mathbf{w}^T \mathbf{x})$

Standard perceptron algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$
2. For epoch in $1 \ldots T$: → In homework, T is number of iterations
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Prediction on a new example with features $\mathbf{x}$: $sgn(\mathbf{w}^T \mathbf{x})$

Voted perceptron algorithm

- Remember every weight vector in your sequence of updates
- At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
- Comes with strong theoretical guarantees about generalization, impractical because of storage issues

Return sequence of (weights, # of examples survived)

Every one of those weight vectors votes on final prediction, gets as many votes as # of examples survived

What's the problem? Too many things to remember. What if 1 million features so weight vectors of 1 million?

Averaged perceptron algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$ and $\mathbf{a} = \mathbf{0} \in \Re^{\mathbf{d+1}}$
2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

     $\mathbf{a} \rightarrow \mathbf{a} + \mathbf{w}$
3. Return $\mathbf{a}$

Average vector

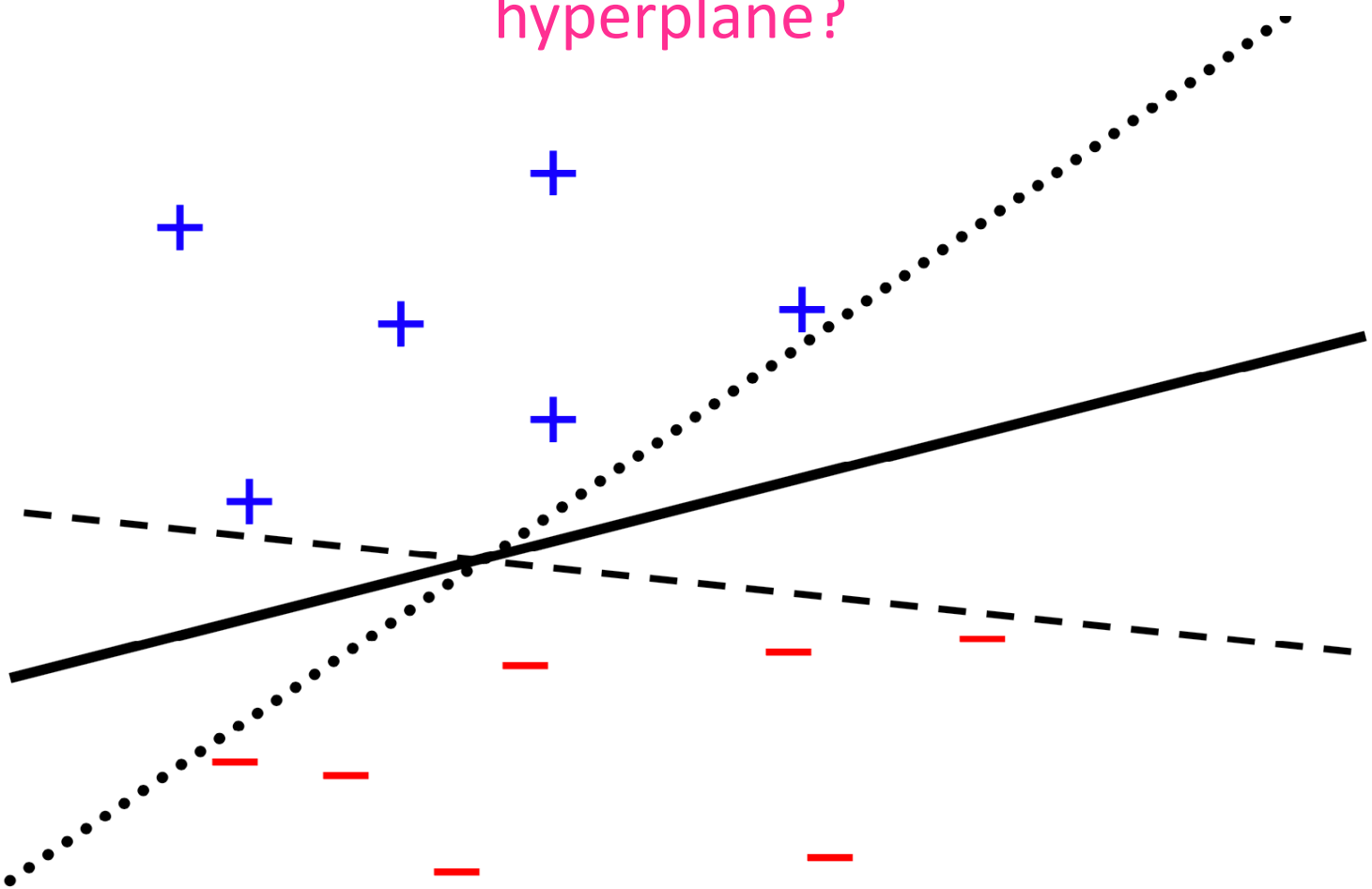
Remember every weight vector in your sequence of updates

Prediction on a new example with features $\mathbf{x}$: $sgn(\mathbf{a}^T \mathbf{x})$

Weight vector that survives longer should dominate the average

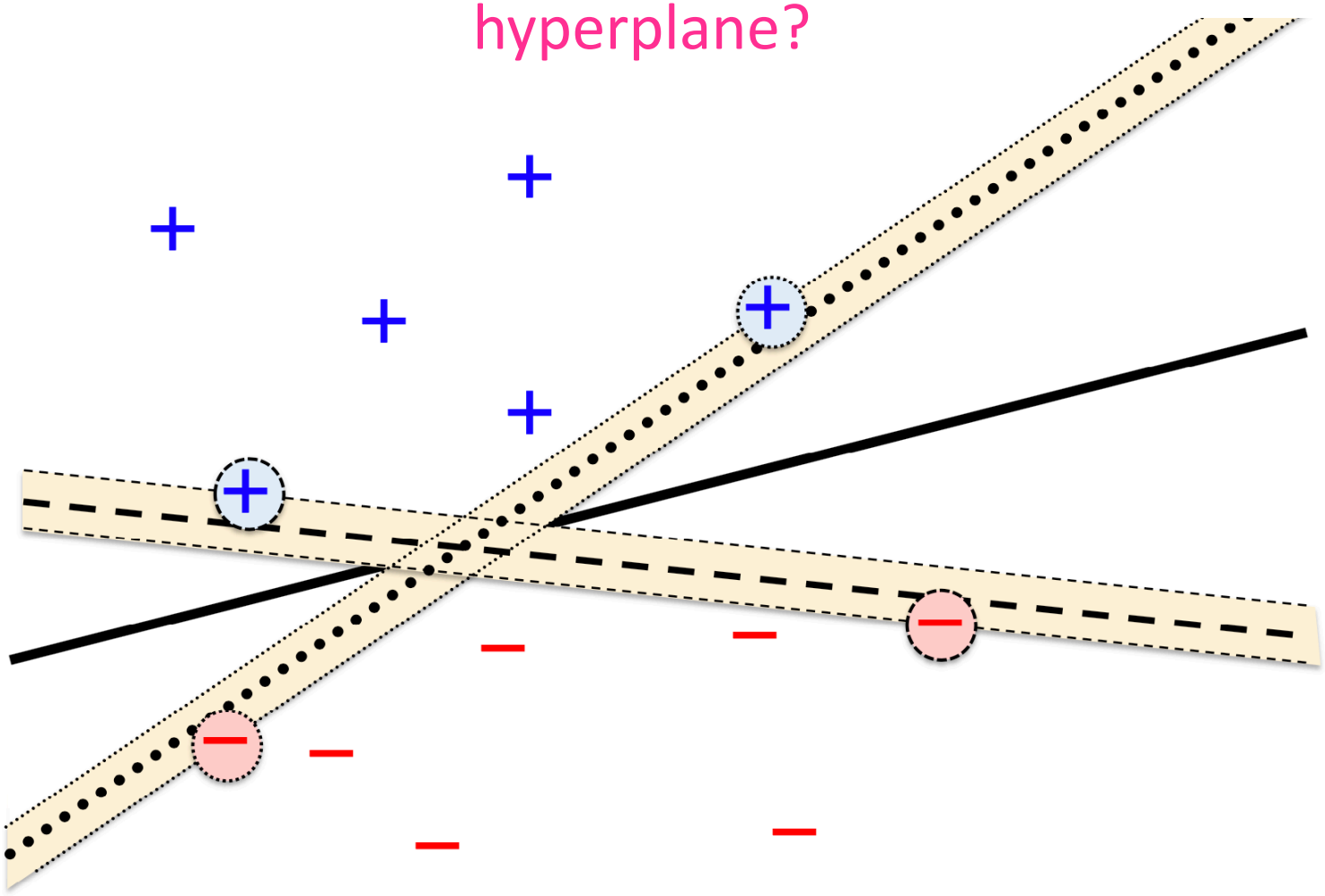
Margin perceptron algorithm: motivation

What is the best separating hyperplane?



Margin perceptron algorithm: motivation

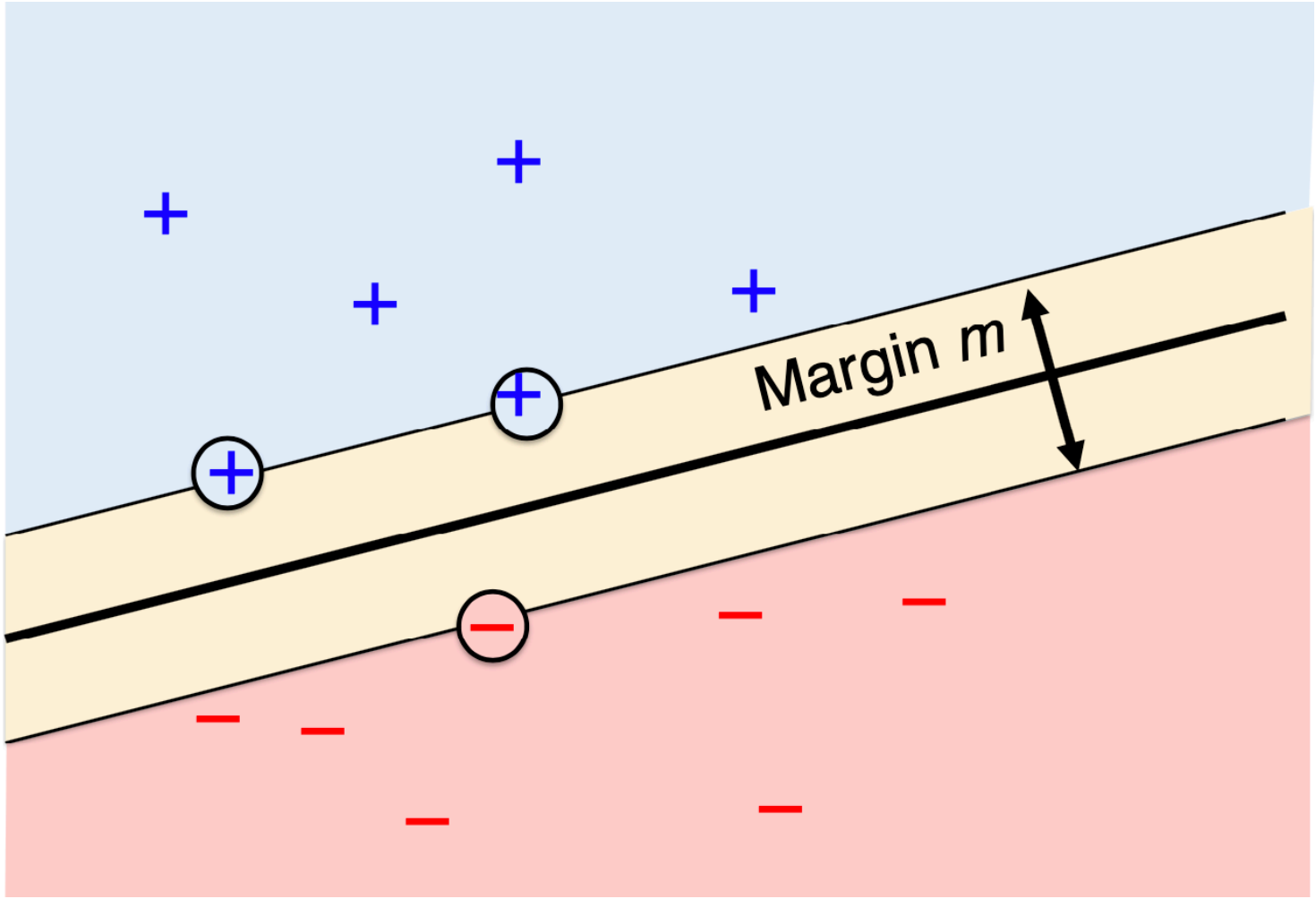
What is the best separating hyperplane?



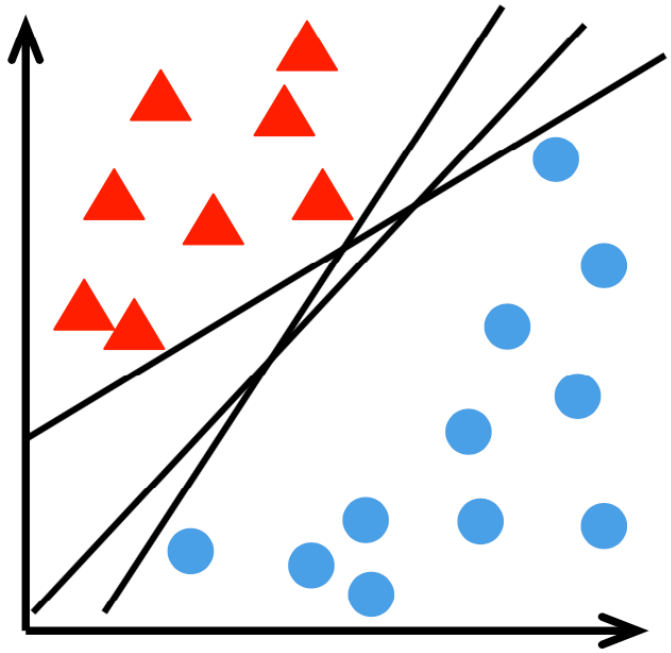
How far away are the points closest to the hyperplane?

Margin perceptron algorithm: motivation

What is the best separating hyperplane?



Margin perceptron algorithm: motivation

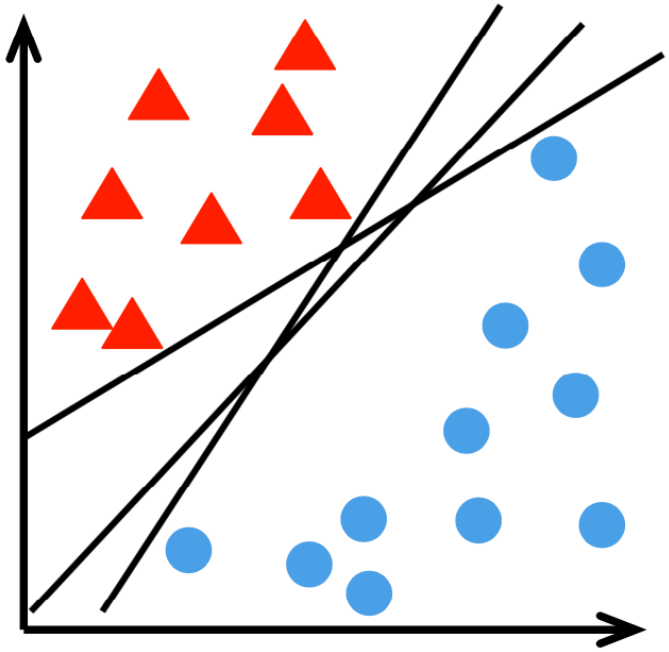


These decision boundaries are very close to some items in the training data

They have **small margins**

Minor changes in the data could lead to different decision boundaries

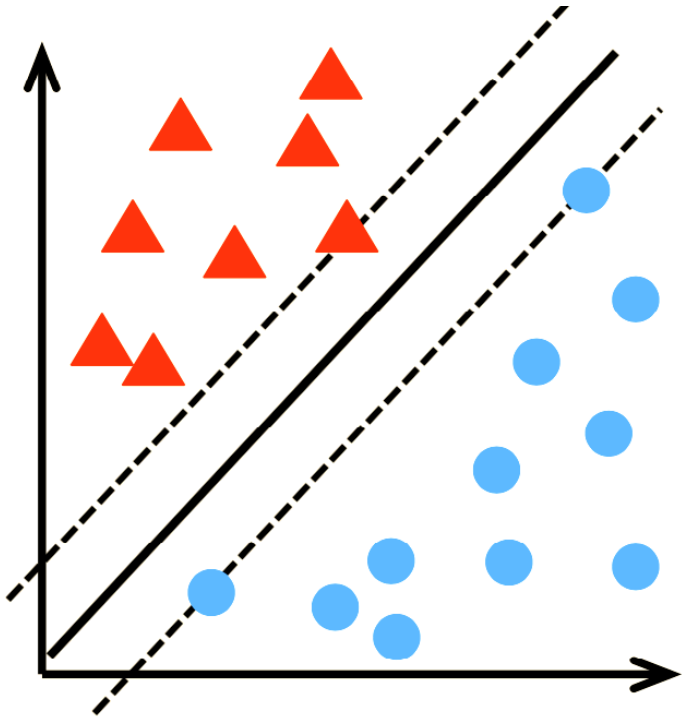
Margin perceptron algorithm: motivation



These decision boundaries are very close to some items in the training data

They have **small margins**

Minor changes in the data could lead to different decision boundaries



This decision boundary is as far away from any training items as possible

It has a **large margin**

Minor changes in the data result in (roughly) the same decision boundary

Margin perceptron algorithm: motivation

Margin = the distance of the decision boundary to the closest items in the training data

We want to find a classifier whose decision boundary is furthest away from the nearest data points. (This classifier has the largest margin)

This additional requirement (bias) reduces the variance (i.e. reduces overfitting). We'll define bias and variance later in the slides

Margin perceptron algorithm

Perceptron makes updates only when the prediction is incorrect

$$y_i \mathbf{w}^T \mathbf{x}_i \leq 0$$

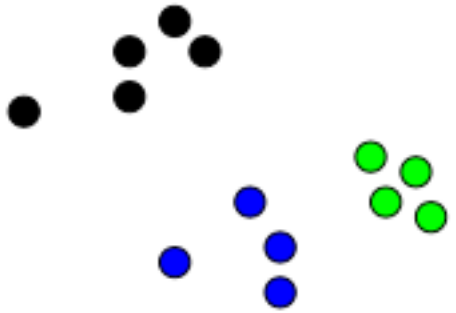
What if the prediction is close to being incorrect? Pick a small positive η and update when

$$y_i \mathbf{w}^T \mathbf{x}_i \leq \eta$$

Can generalize better, but need to choose η

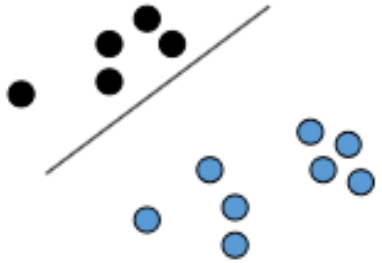
Multi-class perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}$, $y_i \in \{1,2,3,\ldots,k\}$

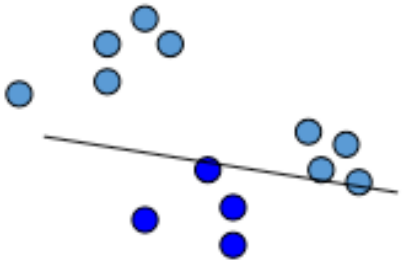


One approach: reduce multi-class problem to binary problems

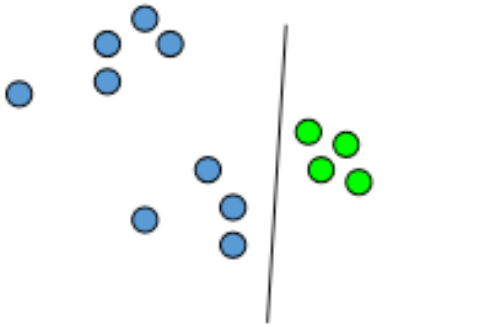
Ideally: only correct label has positive score



$w_{black}^T x > 0$



$w_{blue}^T x > 0$



$w_{green}^T x > 0$

Perceptron

TIPS AND TRICKS

How to improve likelihood of convergence

Useful for Perceptron algorithm but also stochastic gradient descent

- Cross-validation to determine how best to set hyper-parameters like number of epochs or learning rate
- Randomize order of examples seen per epoch
- Check for weights converging
- Check for number of errors converging
- Check for accuracy converging
- Decrease learning rate over time
- Normalize values of features and labels: very different ranges for feature values can make learning more difficult

Evaluation

MOTIVATION FOR LOSS FUNCTIONS

A general framework for learning

Goal: predict an unobserved output value $y \in Y$ based on an observed input vector $\mathbf{x} \in X$

How? Estimate a functional relationship $y \sim f(\mathbf{x})$ from $D = \{(\mathbf{x}_i, y_i)\}$

- Classification: $y \in \{0,1\}$ (or $y \in \{1,2,\ldots,k\}$),
- Regression: $y \in \mathfrak{R}$

A general framework for learning

Goal: predict an unobserved output value $y \in Y$ based on an observed input vector $\mathbf{x} \in X$

How? Estimate a functional relationship $y \sim f(\mathbf{x})$ from $D = \{(\mathbf{x}_i, y_i)\}$

- Classification: $y \in \{0,1\}$ (or $y \in \{1,2,\ldots,k\}$),
- Regression: $y \in \mathfrak{R}$

Question: what is this function $f(\mathbf{x})$ for perceptron?

A general framework for learning

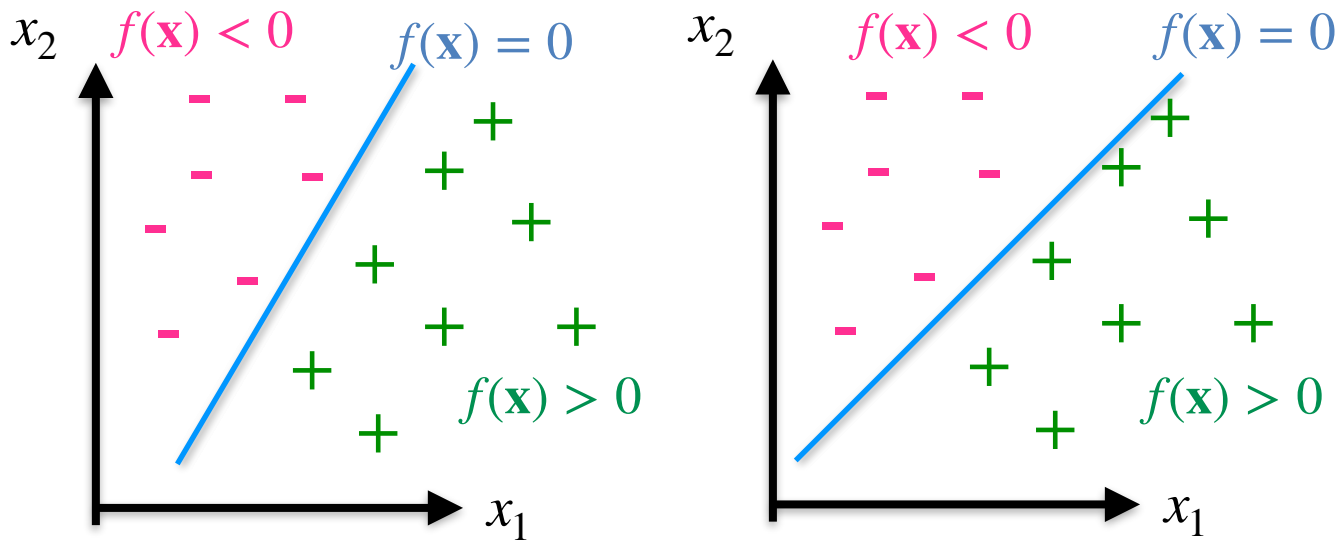
Goal: predict an unobserved output value $y \in Y$ based on an observed input vector $\mathbf{x} \in X$

How? Estimate a functional relationship $y \sim f(\mathbf{x})$ from $D = \{(\mathbf{x}_i, y_i)\}$

- Classification: $y \in \{0,1\}$ (or $y \in \{1,2,\ldots,k\}$),
- Regression: $y \in \mathfrak{R}$

Perceptron function: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

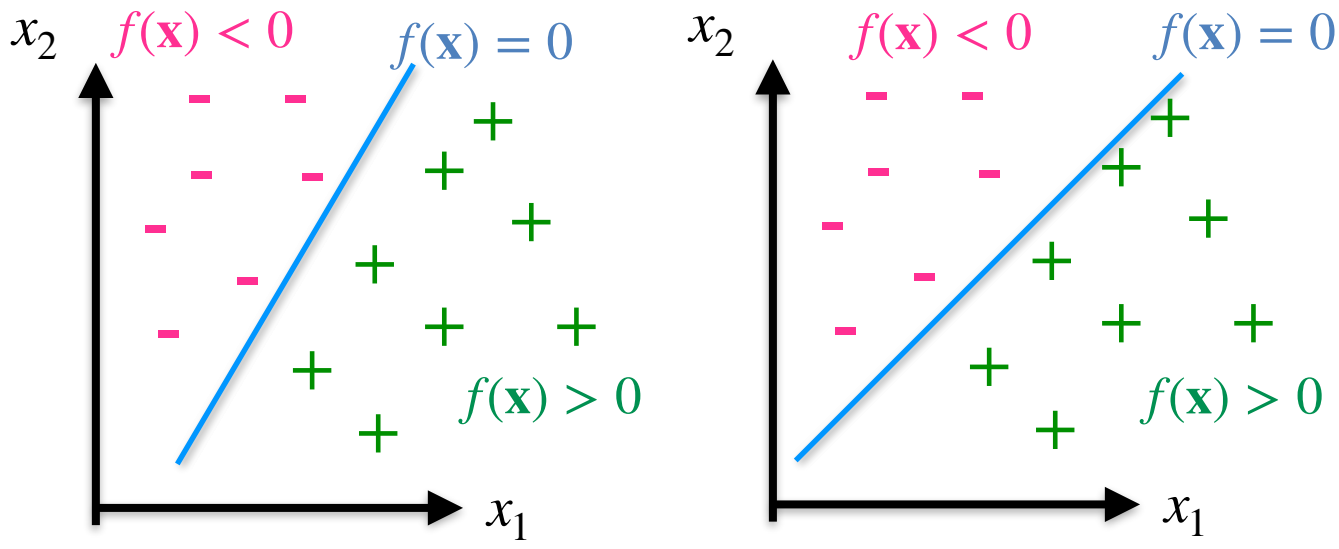
Which model should we pick?



We need a metric (aka an objective function)

We would like to minimize the probability of misclassifying ***unseen*** examples, but we can't measure that probability.

Which model should we pick?

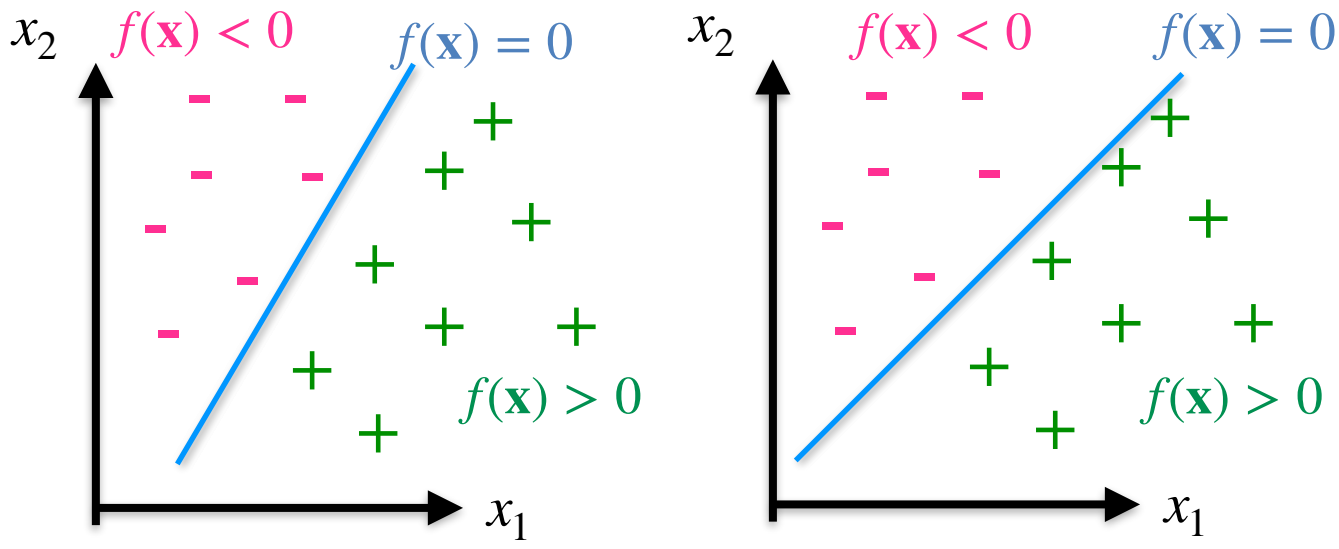


We need a metric (aka an objective function)

We would like to minimize the probability of misclassifying ***unseen*** examples, but we can't measure that probability.

Remember, this probability is with respect to the true data (or instance space) distribution which we don't know

Which model should we pick?

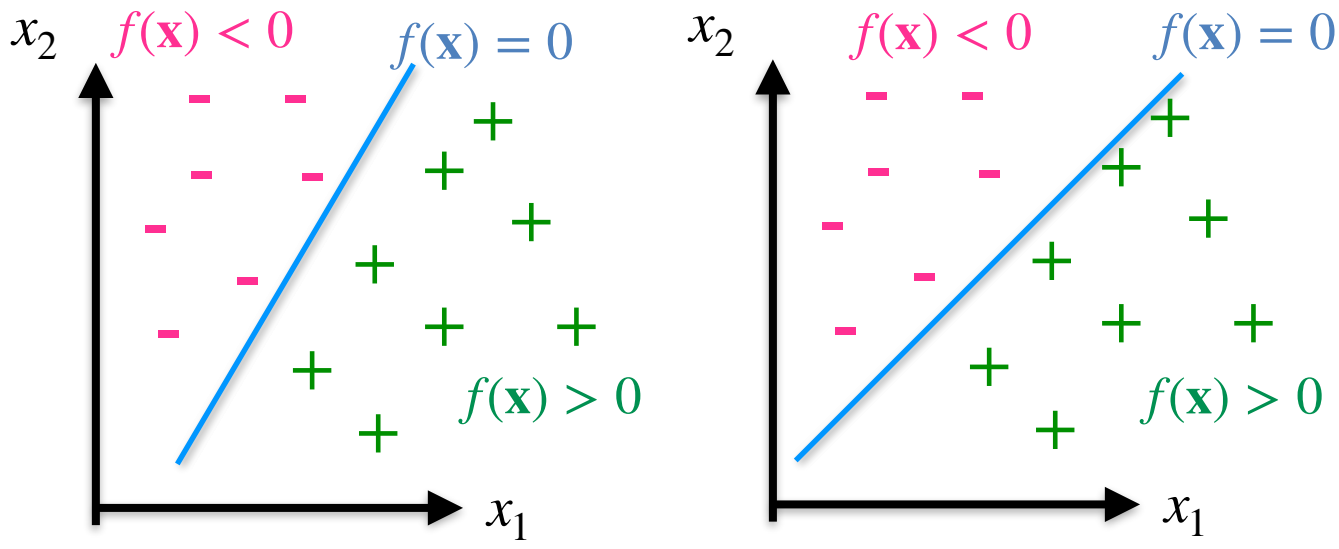


We need a metric (aka an objective function)

We would like to minimize the probability of misclassifying ***unseen*** examples, but we can't measure that probability.

What could we minimize instead?

Which model should we pick?

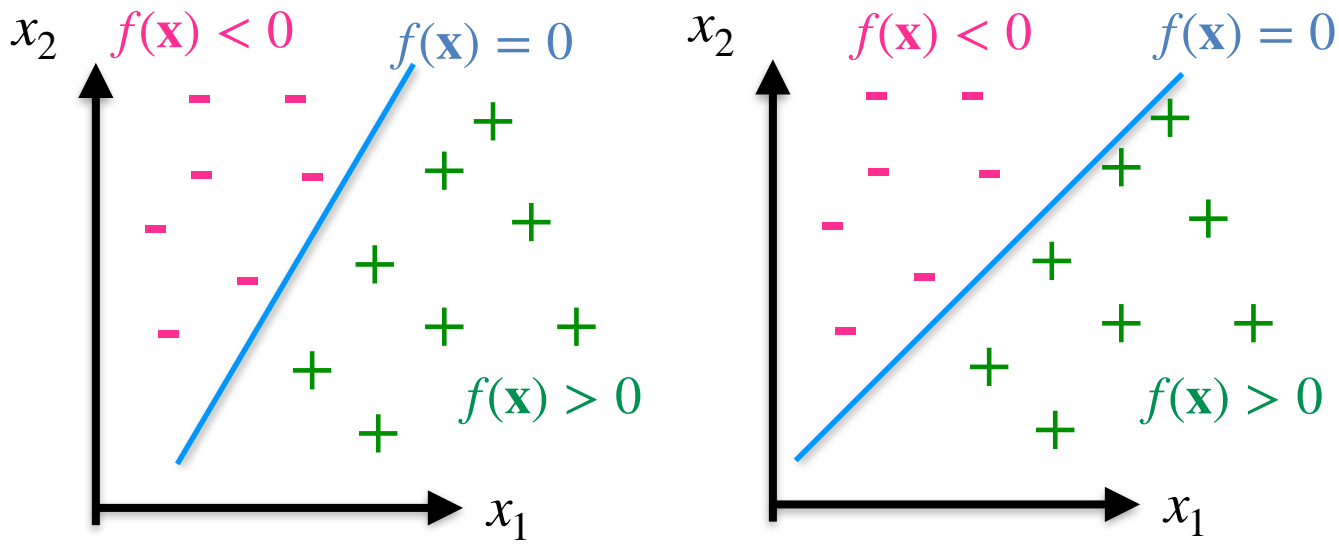


We need a metric (aka an objective function)

We would like to minimize the probability of misclassifying ***unseen*** examples, but we can't measure that probability.

Instead: minimize the number of misclassified training examples

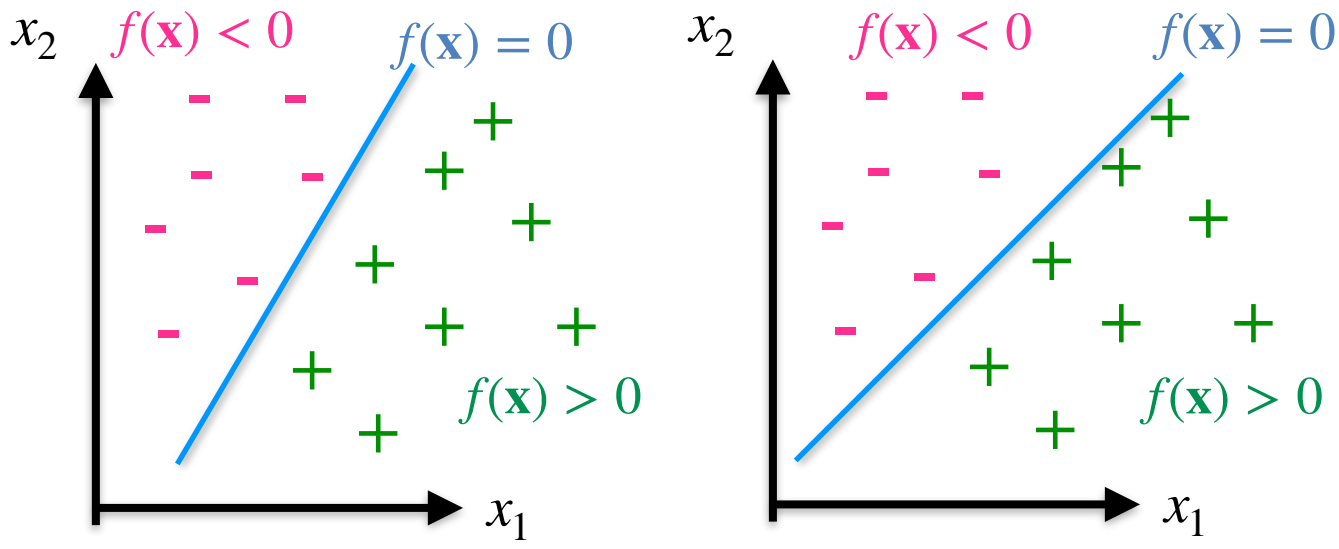
But what if many models minimize misclassified?



Problem: there may still be many models that are consistent with the training data (i.e., minimize number of misclassified examples)

We therefore need a more specific metric

But what if many models minimize misclassified?



Problem: there may still be many models that are consistent with the training data (i.e., minimize number of misclassified examples)

We therefore need a more specific metric

Loss functions provide such metrics

Evaluation

LOSS FUNCTIONS FOR CLASSIFICATION

Original perceptron algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}$, $y_i \in \{-1, 1\}$

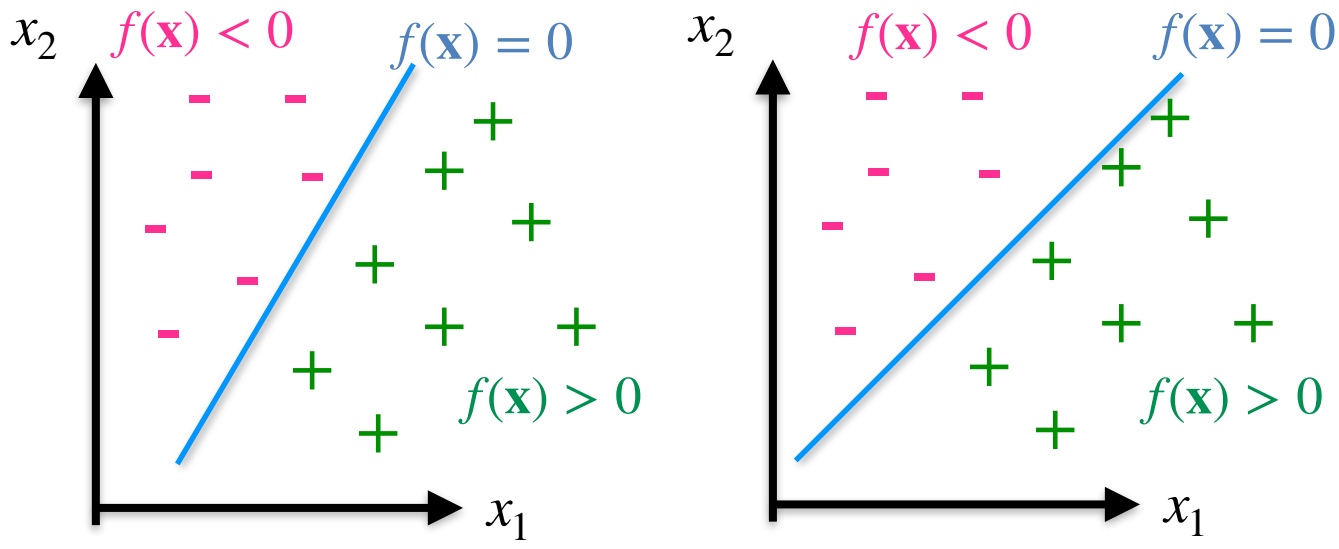
1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return final weight vector

Mistake can be written as $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$

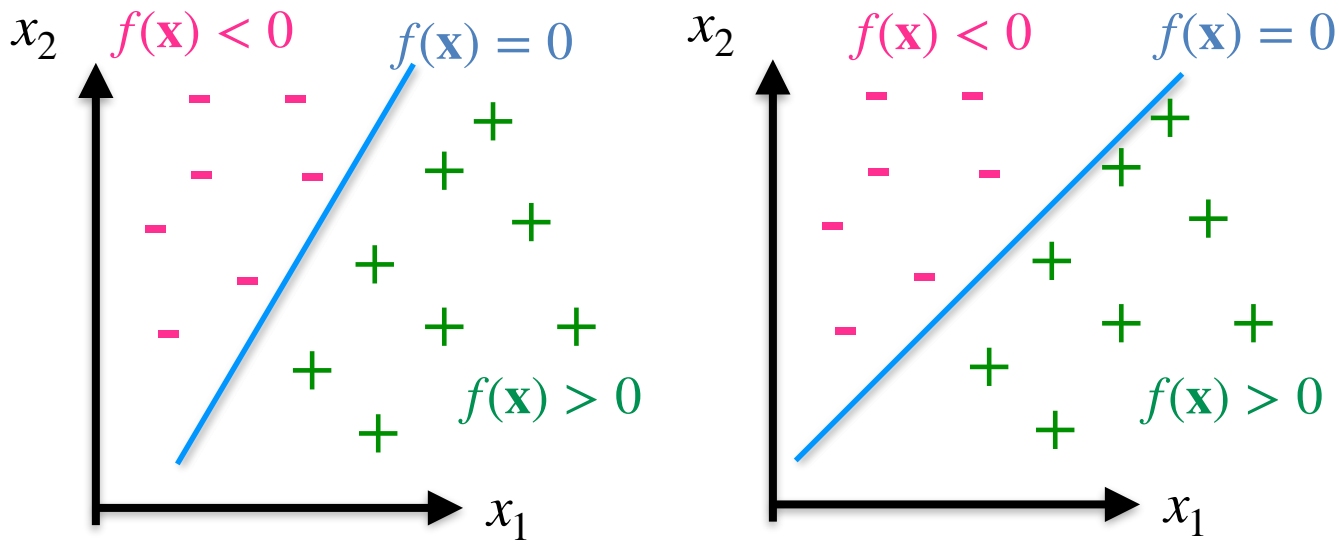
Prediction on a new example with features $\mathbf{x}$: $sgn(\mathbf{w}^T \mathbf{x})$

$yf(\mathbf{x}) > 0$: Correct classification



An example $(\mathbf{x}, y)$ is correctly classified by $f(\mathbf{x})$ if and only if $yf(\mathbf{x}) > 0$:

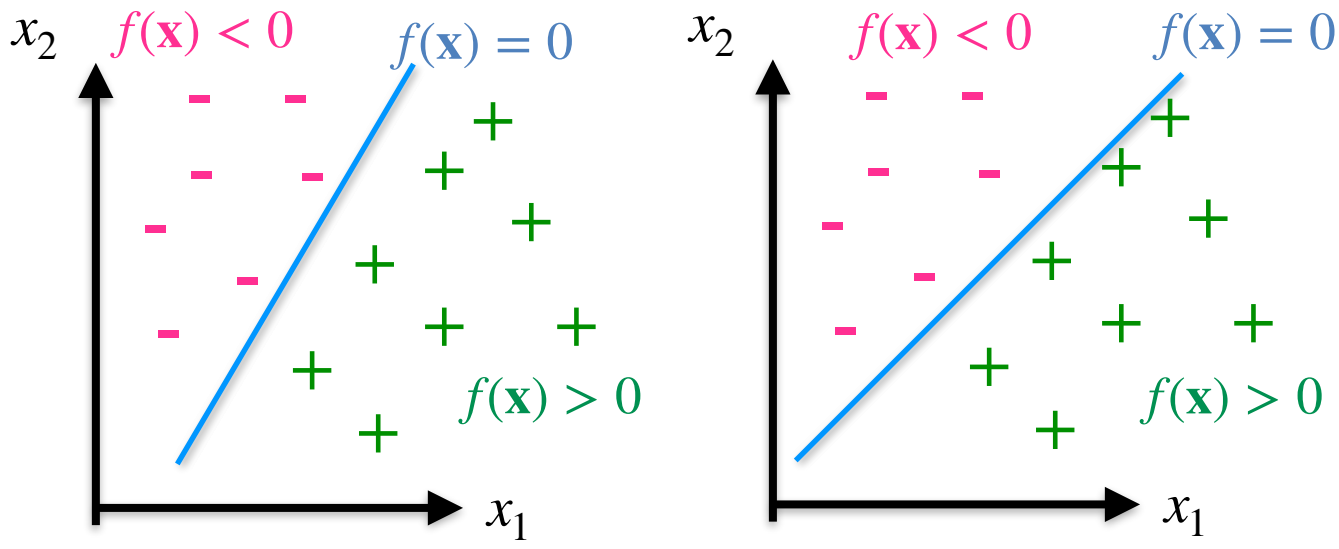
$yf(\mathbf{x}) > 0$: Correct classification



An example $(\mathbf{x}, y)$ is correctly classified by $f(\mathbf{x})$ if and only if $yf(\mathbf{x}) > 0$:

- Case 1 $(y = +1 = y')$: $f(\mathbf{x}) > 0 \Rightarrow yf(\mathbf{x}) > 0$
- Case 2 $(y = -1 = y')$: $f(\mathbf{x}) < 0 \Rightarrow yf(\mathbf{x}) > 0$
- Case 3 $(y = +1 \neq y' = -1)$: $f(\mathbf{x}) > 0 \Rightarrow yf(\mathbf{x}) < 0$
- Case 4 $(y = -1 \neq y' = +1)$: $f(\mathbf{x}) < 0 \Rightarrow yf(\mathbf{x}) < 0$

$yf(\mathbf{x}) > 0$: Correct classification



An example $(\mathbf{x}, y)$ is correctly classified by $f(\mathbf{x})$ if and only if $yf(\mathbf{x}) > 0$:

- Case 1 $(y = +1 = y')$: $f(\mathbf{x}) > 0 \Rightarrow yf(\mathbf{x}) > 0$
- Case 2 $(y = -1 = y')$: $f(\mathbf{x}) < 0 \Rightarrow yf(\mathbf{x}) > 0$
- Case 3 $(y = +1 \neq y' = -1)$: $f(\mathbf{x}) > 0 \Rightarrow yf(\mathbf{x}) < 0$
- Case 4 $(y = -1 \neq y' = +1)$: $f(\mathbf{x}) < 0 \Rightarrow yf(\mathbf{x}) < 0$

Loss function plots will be a function of $yf(\mathbf{x})$

Loss functions for classification

Loss functions, $L(y, f(\mathbf{x}))$, evaluate the performance of a given classifier

Loss functions for classification

Loss functions, $L(y, f(\mathbf{x}))$, evaluate the performance of a given classifier

Loss = what penalty do we incur if we misclassify $\mathbf{x}$?

Loss functions for classification

Loss functions, $L(y, f(\mathbf{x}))$, evaluate the performance of a given classifier

Loss = what penalty do we incur if we misclassify $\mathbf{x}$?

$L(y, f(\mathbf{x}))$ is the loss of classifier f on example $\mathbf{x}$ when the true label of $\mathbf{x}$ is y

- We assign label $y' = sgn(f(\mathbf{x}))$ to $\mathbf{x}$

Loss functions for classification

Loss functions, $L(y, f(\mathbf{x}))$, evaluate the performance of a given classifier

Loss = what penalty do we incur if we misclassify $\mathbf{x}$?

$L(y, f(\mathbf{x}))$ is the loss of classifier f on example $\mathbf{x}$ when the true label of $\mathbf{x}$ is y

- We assign label $y' = sgn(f(\mathbf{x}))$ to $\mathbf{x}$

Plots of $L(y, f(\mathbf{x}))$: x-axis is typically $yf(\mathbf{x})$

Many different possible loss functions

Misclassification error (0-1 loss)

$$L(y, f(\mathbf{x})) = 0 \text{ if } f(\mathbf{x}) = y; \quad 1 \text{ otherwise}$$

Squared loss

$$L(y, f(\mathbf{x})) = (f(\mathbf{x}) - y)^2$$

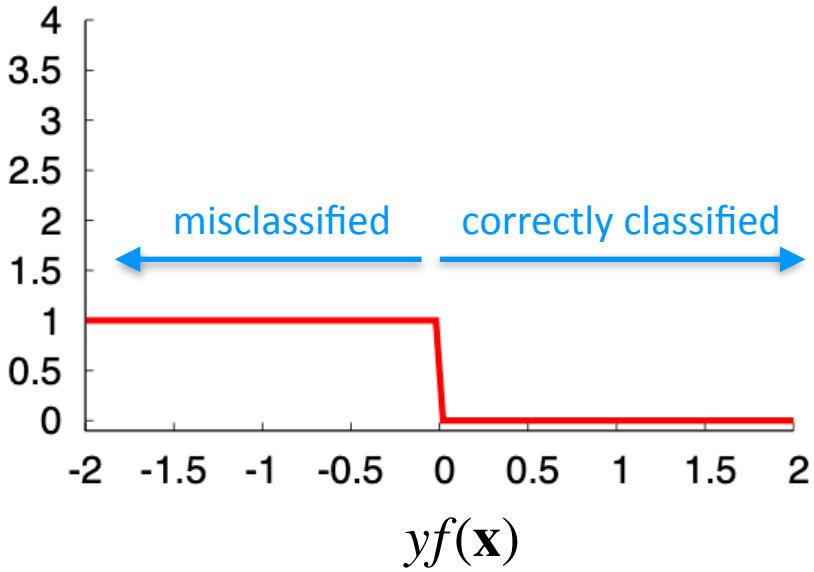
Input dependent loss

$$L(f(\mathbf{x}), y) = 0 \text{ if } f(\mathbf{x}) = y; \quad c(x) \text{ otherwise}$$

Misclassification error: 0-1 Loss

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

$$L(y, f(\mathbf{x})) = \frac{1}{2}(1 - sgn(yf(\mathbf{x})))$$



$$L(y, f(\mathbf{x})) = 0 \iff y = y'$$
$$= 1 \iff y \neq y'$$

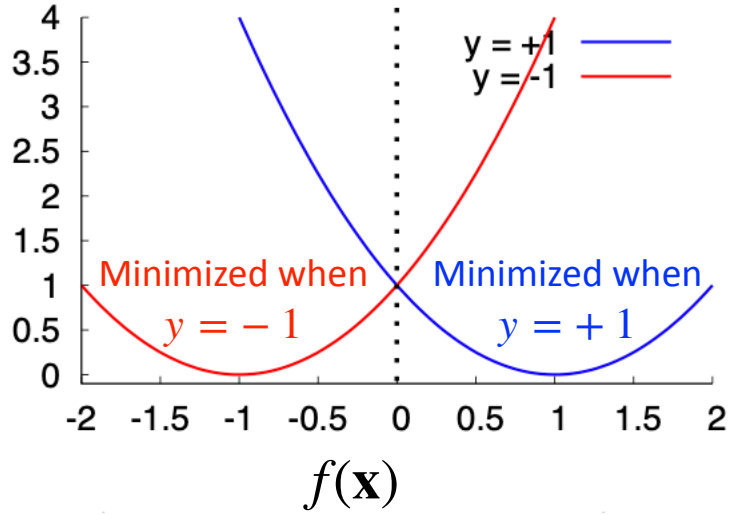
$$L(y, f(\mathbf{x})) = 0 \text{ if } f(\mathbf{x}) = y; \quad 1$$

$$L(yf(\mathbf{x})) = 0 \iff yf(\mathbf{x}) > 0 \quad \text{(correctly classified)}$$
$$= 1 \iff yf(\mathbf{x}) < 0 \quad \text{(misclassified)}$$

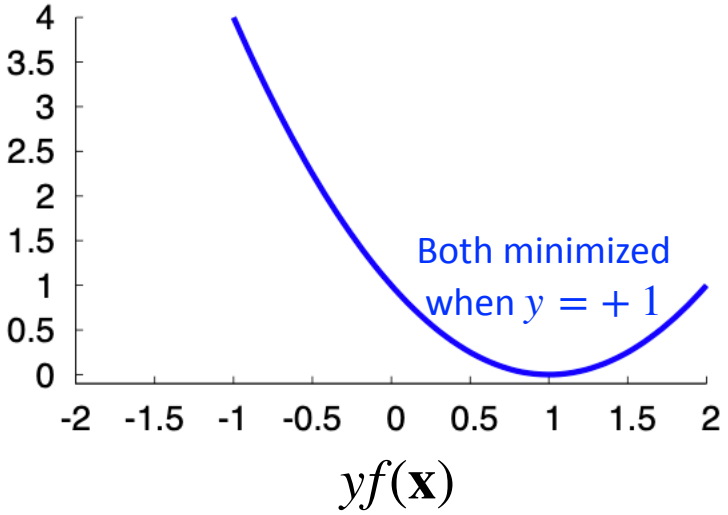
Square loss $(y - f(\mathbf{x}))^2$

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

Square loss as a function of $f(\mathbf{x})$



Square loss as a function of $yf(\mathbf{x})$



$$L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$$

What is our goal?

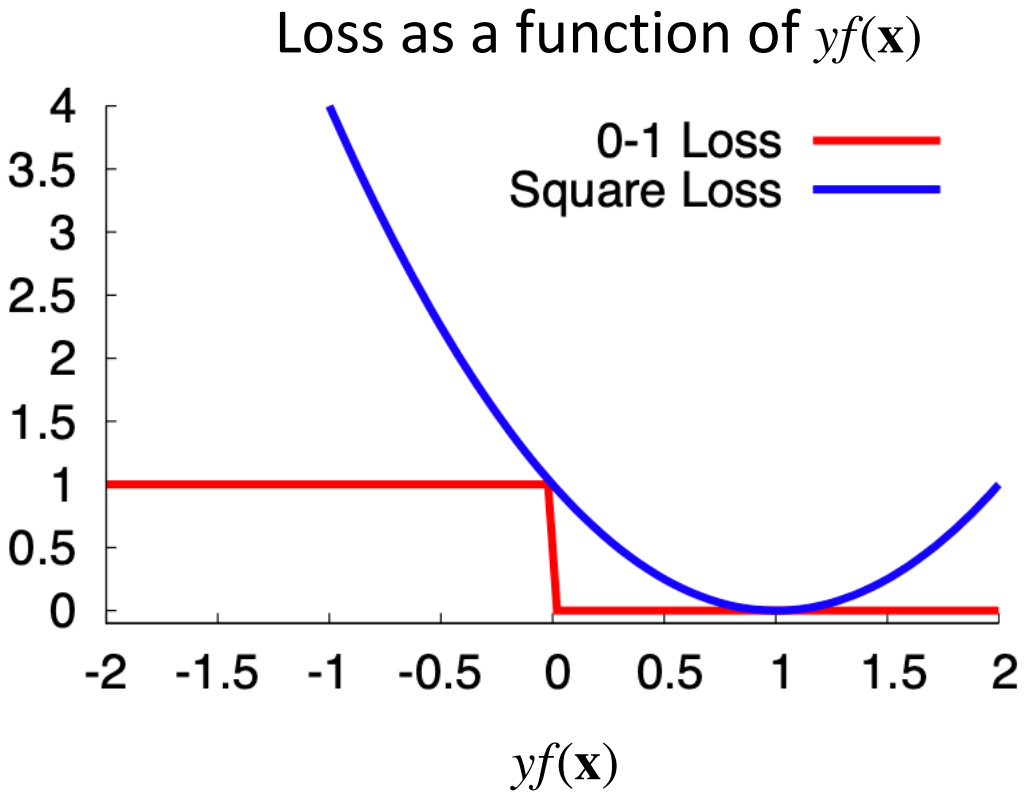
Our goal is to minimize the loss function, that is, find a set of parameter values for which the loss we get from the loss function is a minimum

Minimizing 0-1 loss function is typically NP-hard

To alleviate this computational problem, minimize a new function - a **convex** upper bound of the classification error

A continuous convex loss function allows a simpler optimization algorithm

The square loss is a convex upper bound on 0-1 Loss



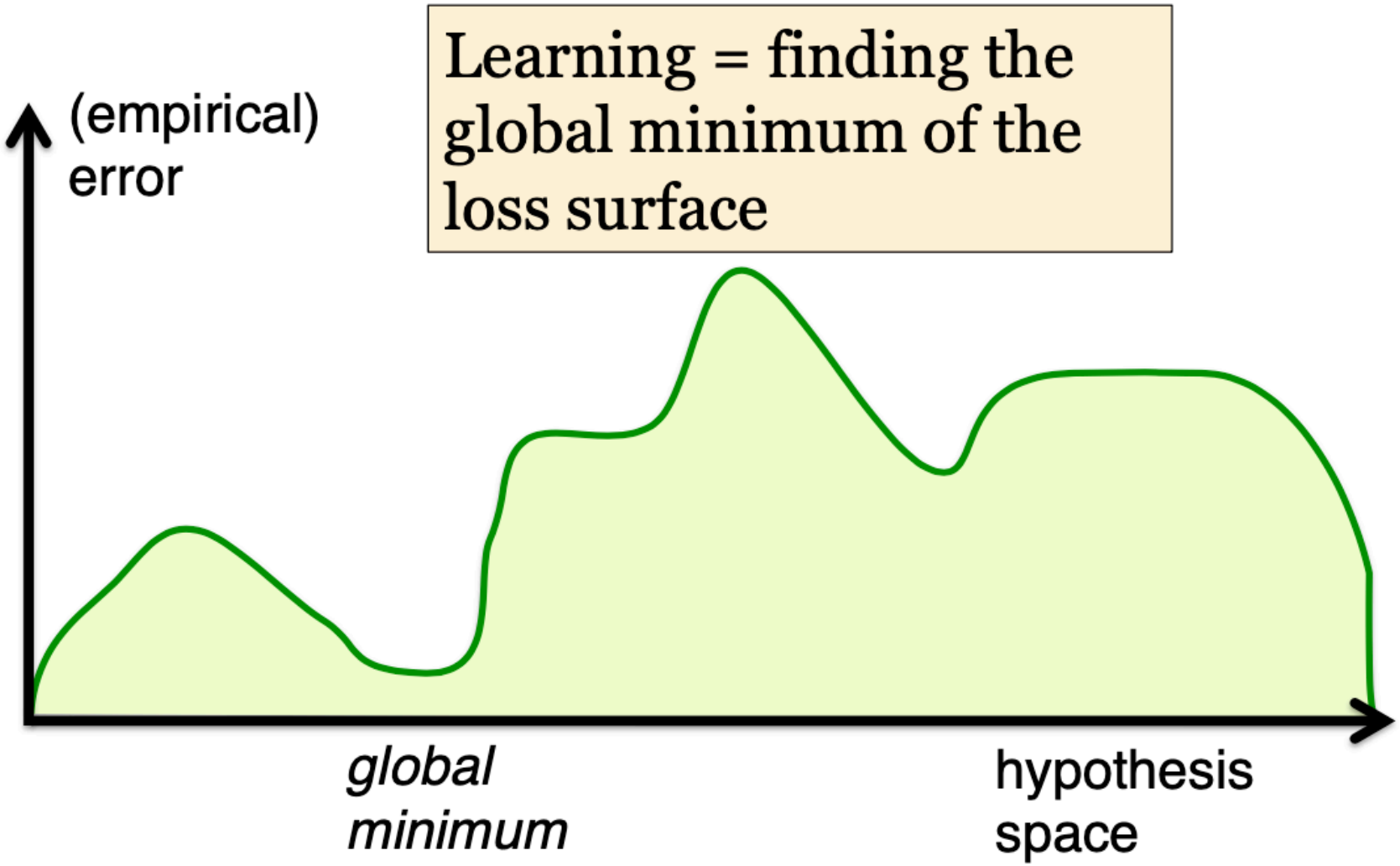
The loss surface

Linear classification

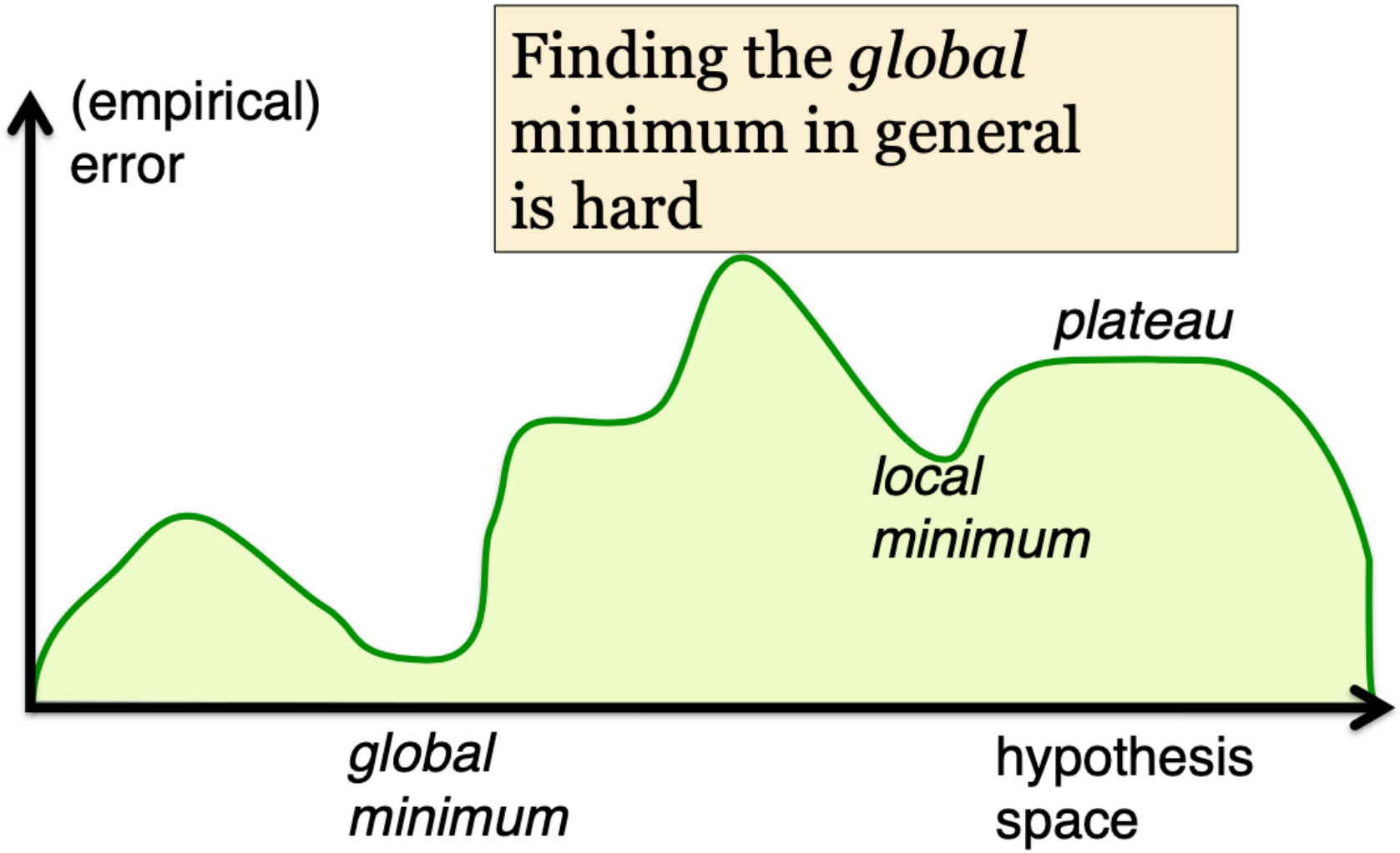
- Hypothesis space is parameterized by $\mathbf{w}$
- Plain English: each $\mathbf{w}$ yields a different classifier

Error/Loss/Risk are all functions of $\mathbf{w}$

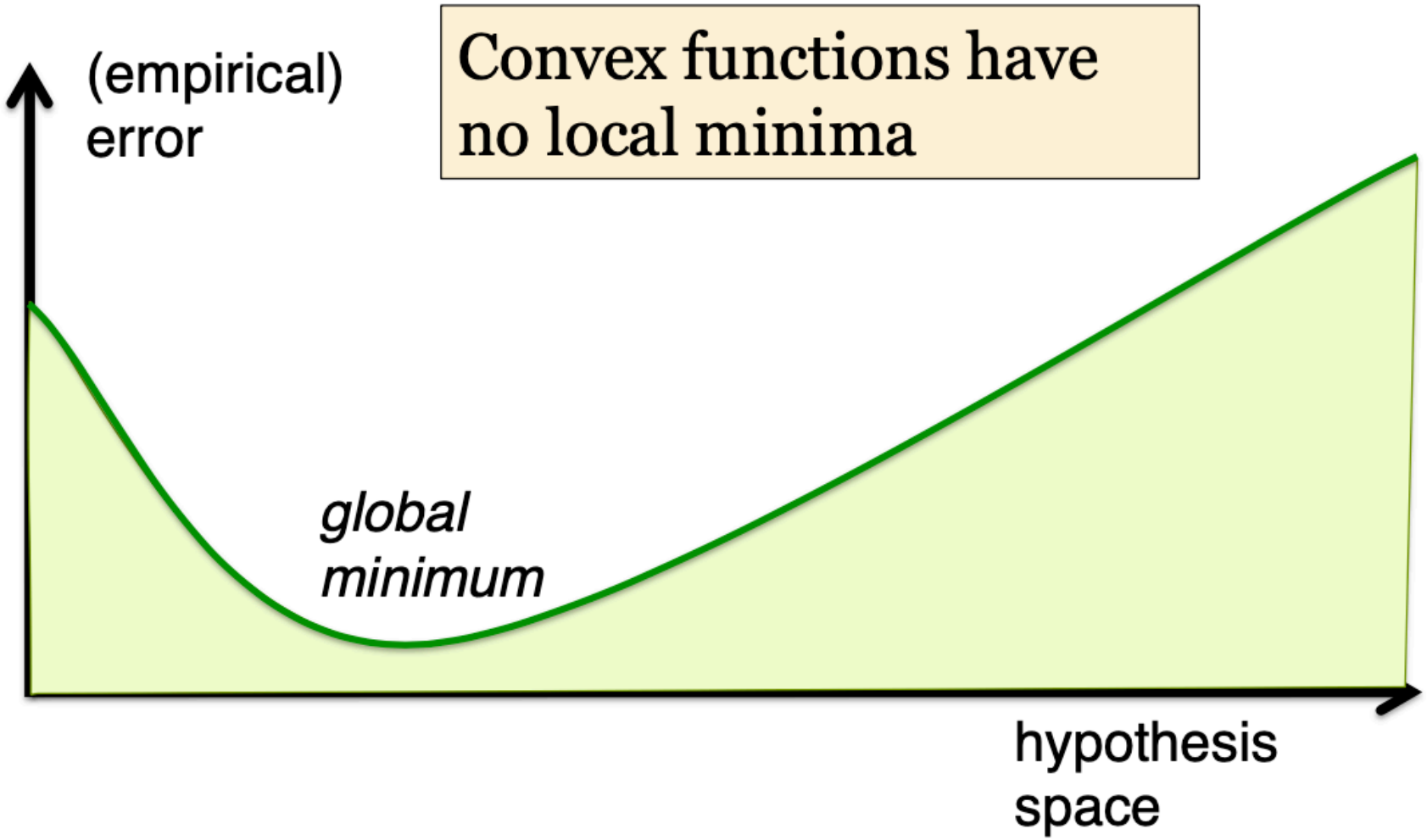
The loss surface



The loss surface



The loss surface



The risk of a classifier $R(f)$

The risk (aka generalization error) of a classifier $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ is its expected loss: (= loss, averaged over all possible data sets):

$$R(f) = \int L(y, f(\mathbf{x}))P(\mathbf{x}, y)d\mathbf{x}, y$$

Ideal learning objective: Find an f that minimizes risk

The i.i.d. assumption

We always assume that training and test items are **independently and identically distributed (i.i.d.):**

There is a distribution $P(X, Y)$ from which the data $D = \{(\mathbf{x}, y)\}$ is generated. Usually $P(X, Y)$ is unknown to us (we just know it exists)

Training and test data are samples drawn from the same $P(X, Y)$: they are **identically distributed**

Each (x, y) is drawn **independently** from $P(X, Y)$

The empirical risk of $f(\mathbf{x})$

The empirical risk of a classifier $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ on data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_d, y_d)\}$ is its average loss on the items in D

$$R_D(f) = \frac{1}{d}\sum_{i=1}^{d} L(y_i, f(\mathbf{x}_i))$$

The empirical risk of $f(\mathbf{x})$

The empirical risk of a classifier $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ on data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_d, y_d)\}$ is its average loss on the items in D

$$R_D(f) = \frac{1}{d}\sum_{i=1}^{d} L(y_i, f(\mathbf{x}_i))$$

Realistic learning objective: find an f that minimizes empirical risk

Note that the learner can ignore the constant $\frac{1}{d}$

Empirical risk minimization

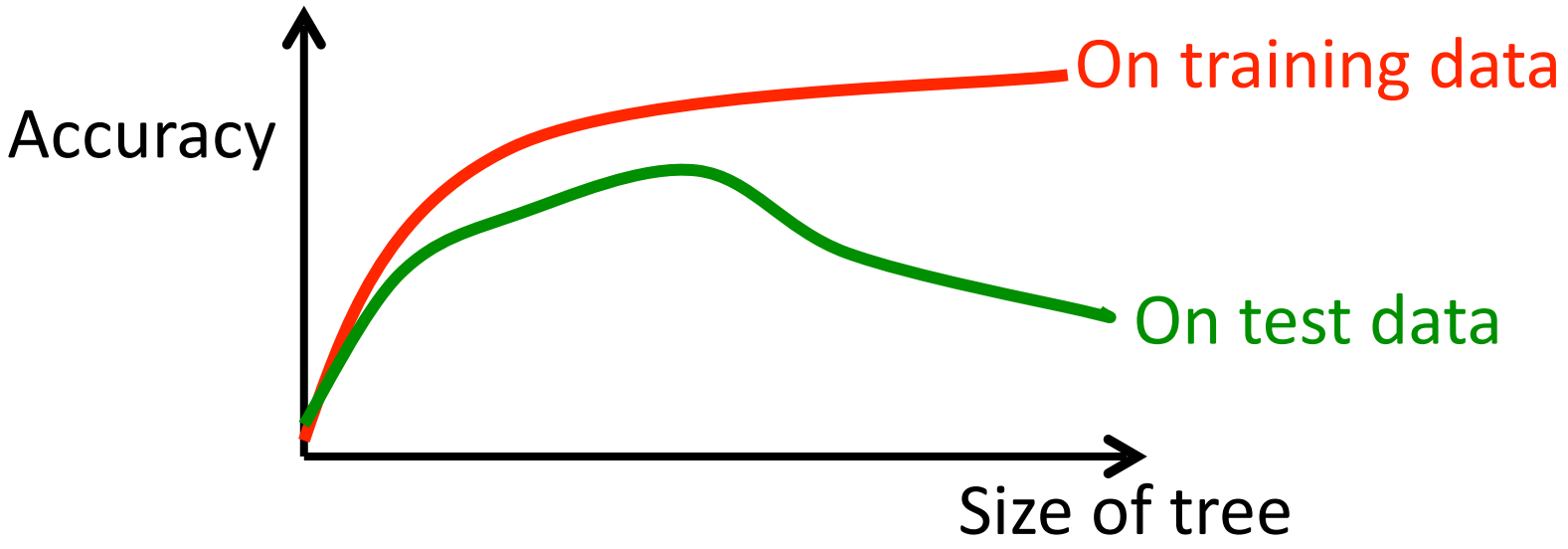
Learning:

Given training data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_d, y_d)\}$, return the classifier $f(\mathbf{x})$ that minimizes the empirical risk $R_D(f)$

Evaluation

BIAS AND VARIANCE

Overfitting



A decision tree overfits the training data when its accuracy on the training data goes up but its accuracy on unseen data goes down

Reasons for overfitting

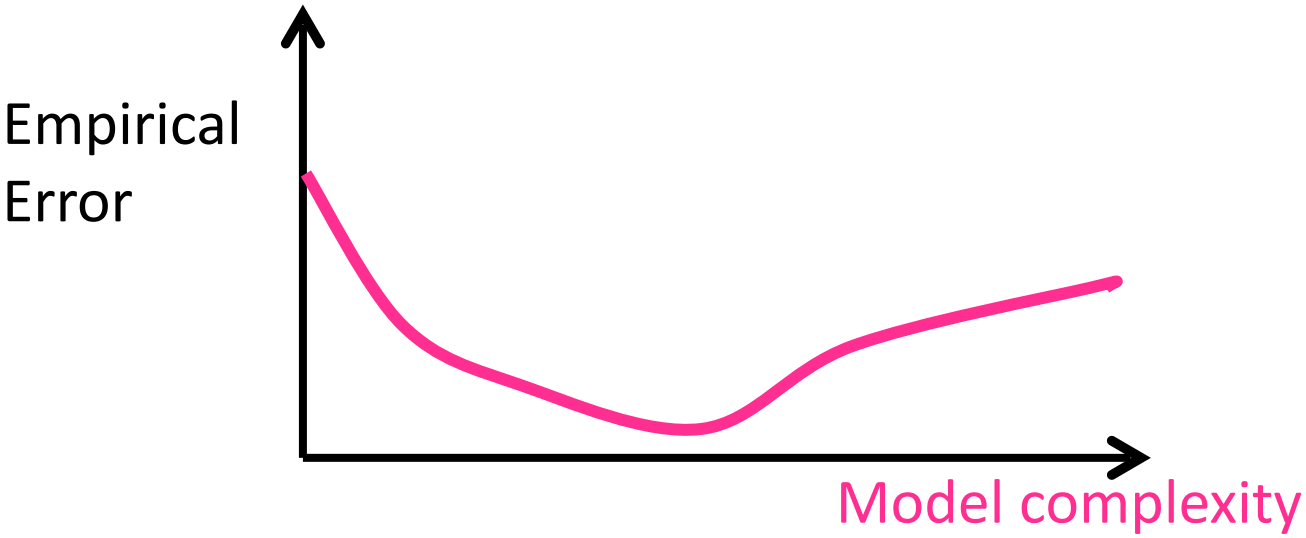
Too much variance in the training data

- Training data is not a representative sample of instance space
- We split tree on features that are actually irrelevant

Too much noise in the training data

- Noise = some feature values or class labels are incorrect
- We learn to predict the noise

Overfitting



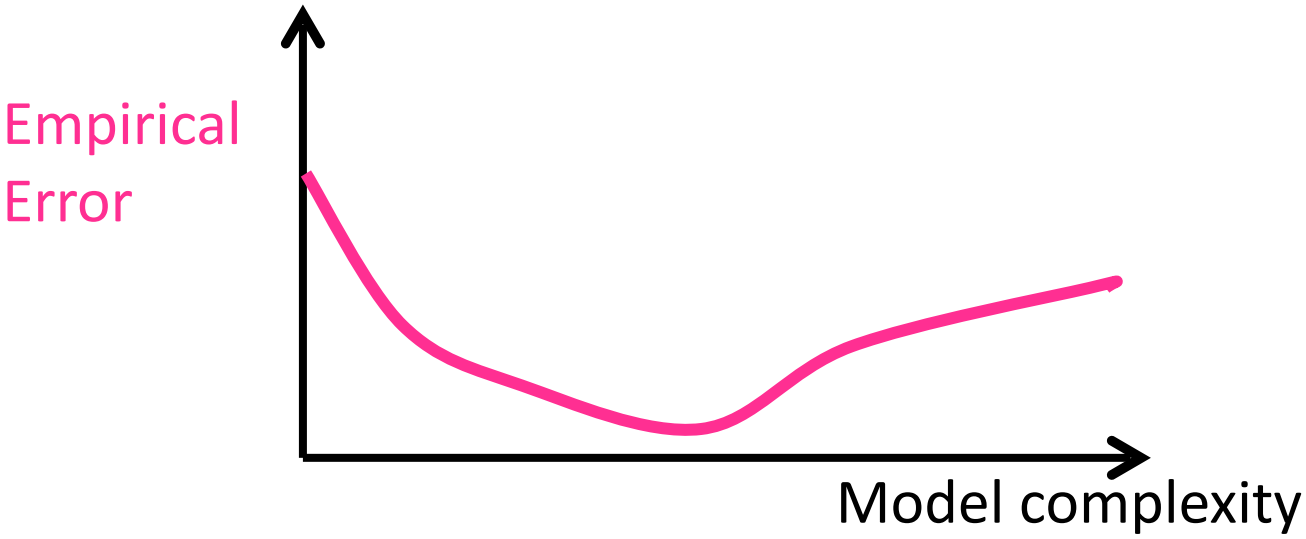
Model complexity (informally):

How many parameters do we have to learn?

Decision trees: complexity = # of nodes

Perceptron: number of weights

Overfitting



Empirical error (= on a given data set):

The percentage of items in this data set that are misclassified by the classifier f

The i.i.d. assumption

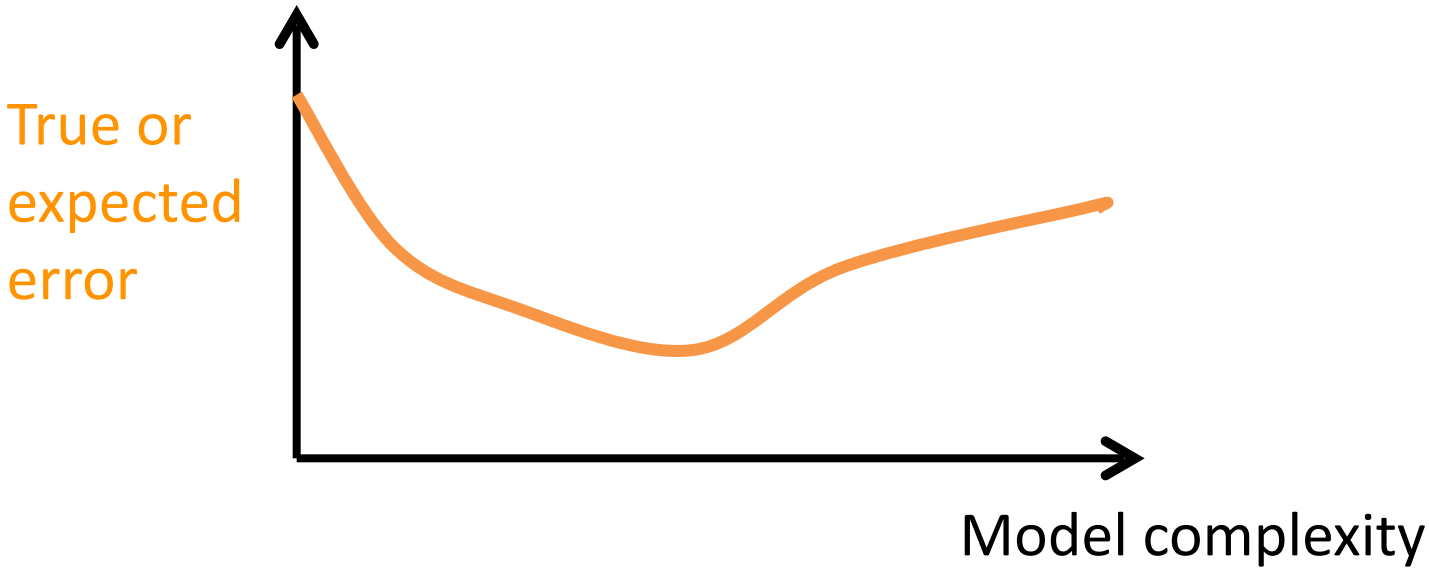
We always assume that training and test items are **independently and identically distributed (i.i.d.):**

There is a distribution $P(X, Y)$ from which the data $D = \{(\mathbf{x}, y)\}$ is generated. Usually $P(X, Y)$ is unknown to us (we just know it exists)

Training and test data are samples drawn from the same $P(X, Y)$: they are **identically distributed**

Each (x, y) is drawn **independently** from $P(X, Y)$

Overfitting

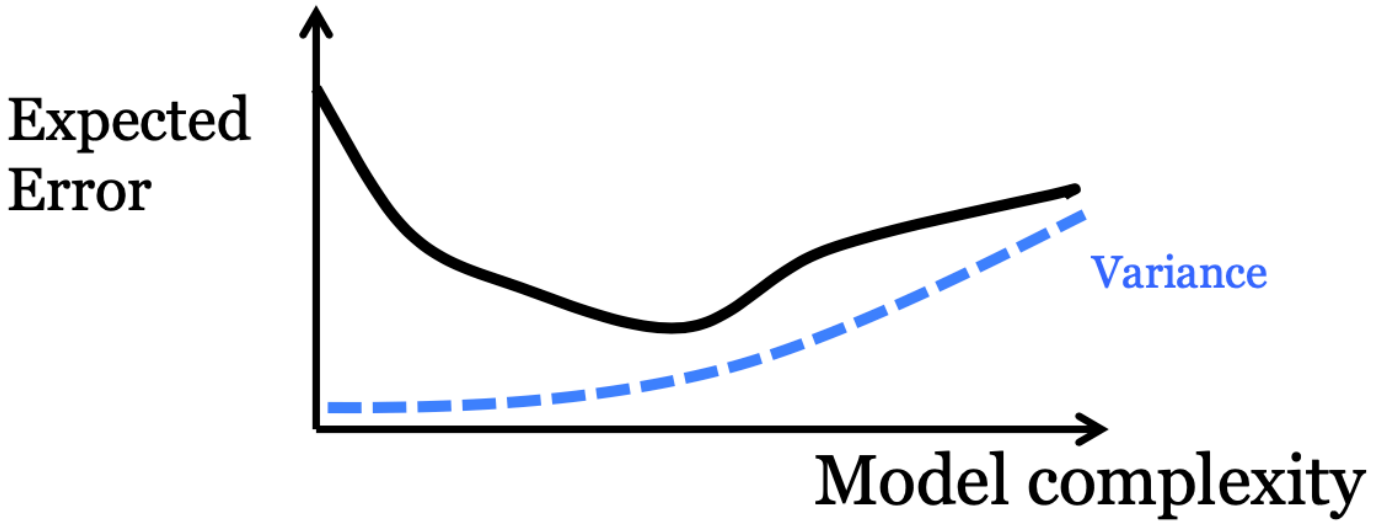


True or expected error:

What percentage of items drawn from $P(X, Y)$ do we expect to be misclassified by f?

(That's what we really care about – **generalization**)

Variance of a learner (informally)

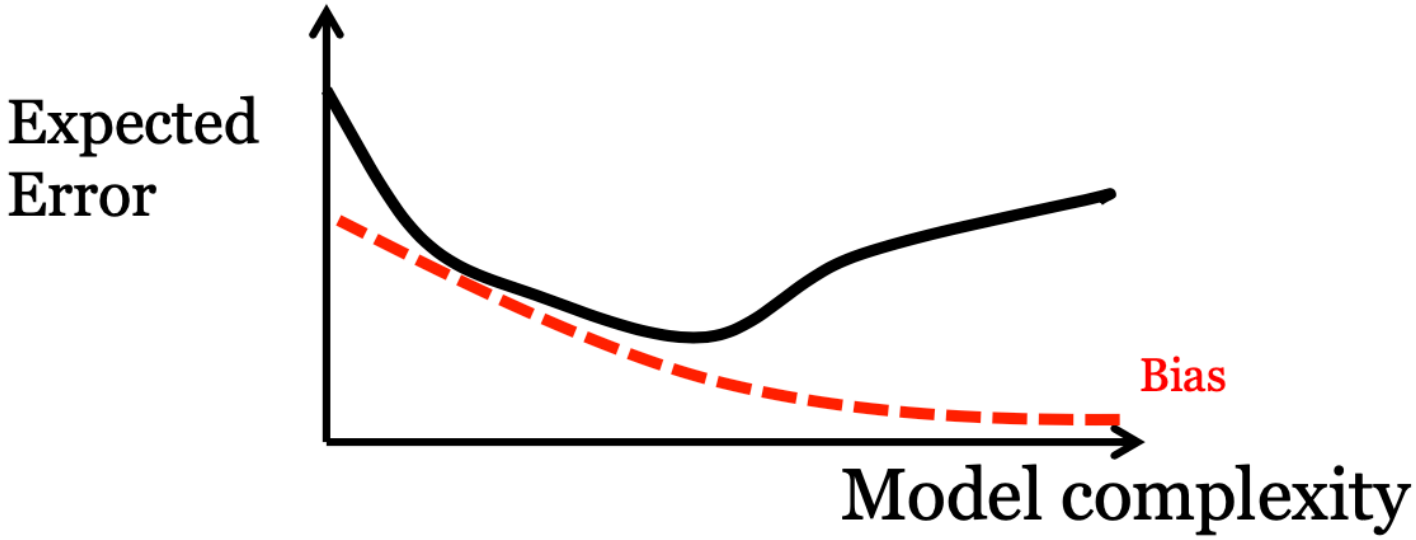


Variance measures how susceptible the learner is to **minor changes in the training data**

- i.e., to different samples from $P(X, Y)$

Variance increases with model complexity

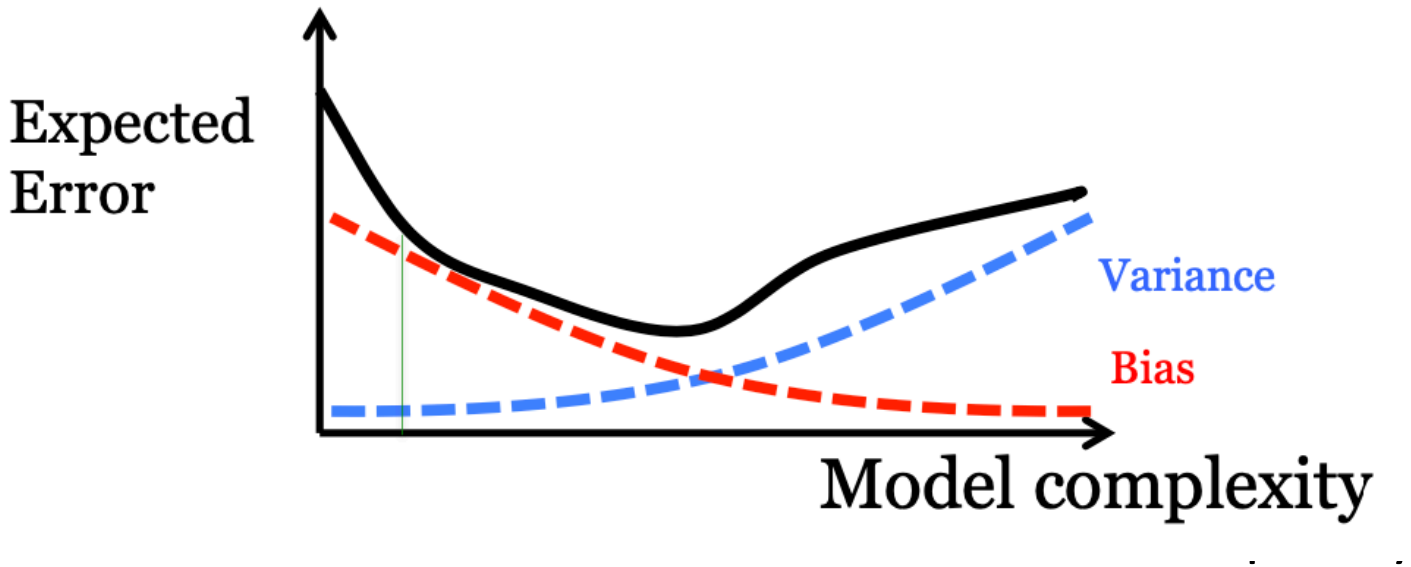
Bias of a learner (informally)



How likely is the learner to identify the **target** hypothesis?

- Bias is low when the model is expressive (complex)
- Bias is high when the model is (too) simple

Impact of bias and variance



Risk = Expected error of a learner ≈ bias^2 + variance
(+ noise)

(Computed using squared error loss)

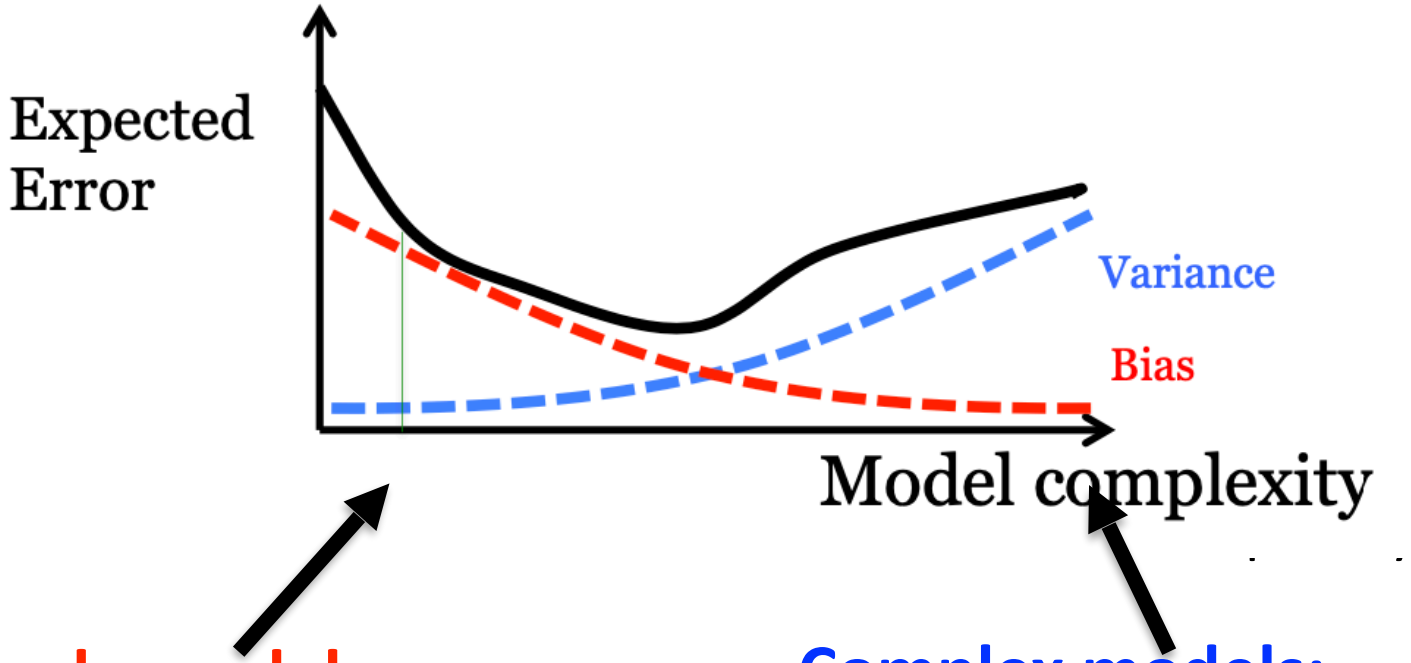
Empirical risk minimization

The empirical risk of a classifier $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ on data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_d, y_d)\}$ is its average loss on the items in D

$$R_D(f) = \frac{1}{d}\sum_{i=1}^{d} L(y_i, f(\mathbf{x}_i))$$

Given training data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_d, y_d)\}$, return the classifier $f(\mathbf{x})$ that minimizes the empirical risk $R_D(f)$

Impact of bias and variance



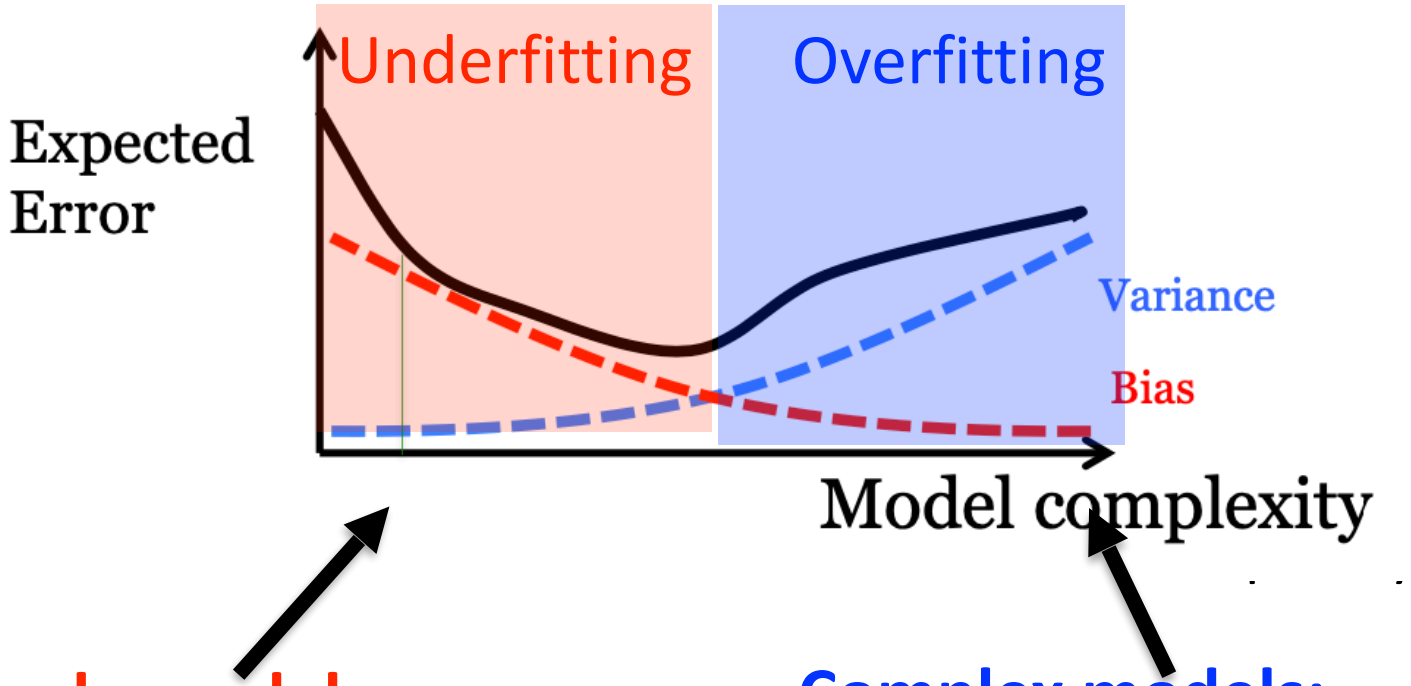
Simple models:

High bias and low variance

Complex models:

High variance and low bias

Impact of bias and variance



Simple models:

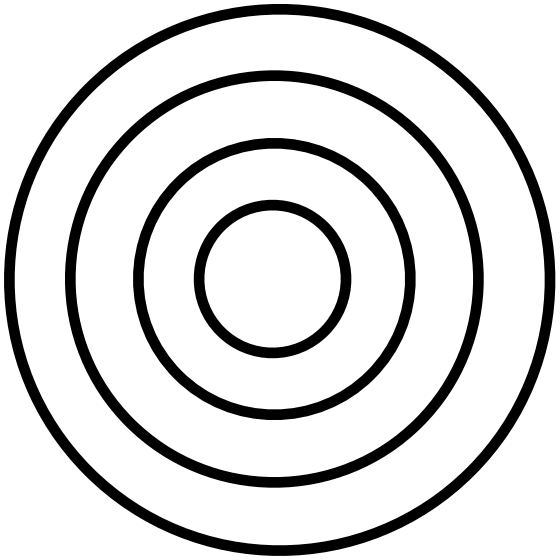
High bias and low variance

Complex models:

High variance and low bias

Let’s play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models



High bias

Hypothesis cannot represent target function

Low bias

Hypothesis can represent target function

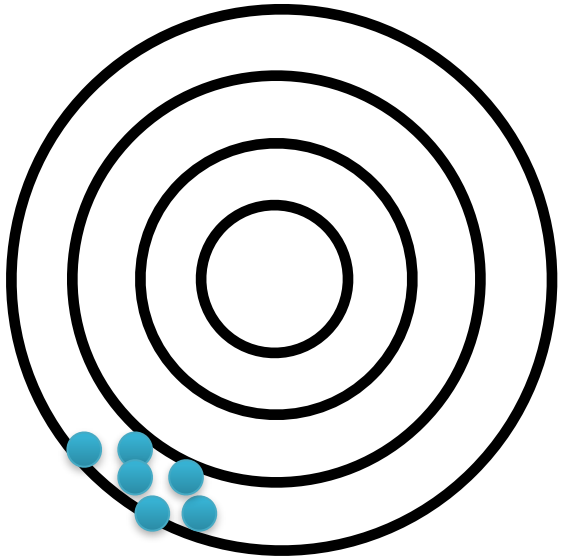
Low variance

High variance

How much does classifier depend on training data?

Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models



High bias

Hypothesis cannot represent target function

Low bias

Hypothesis can represent target function

Each dot is a model that is learned from a a different dataset

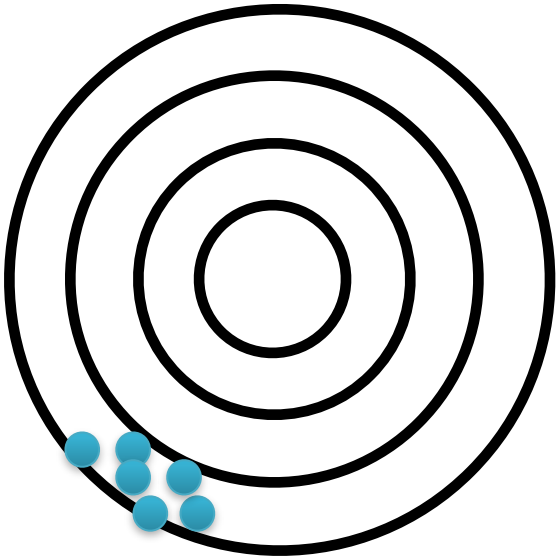
Low variance

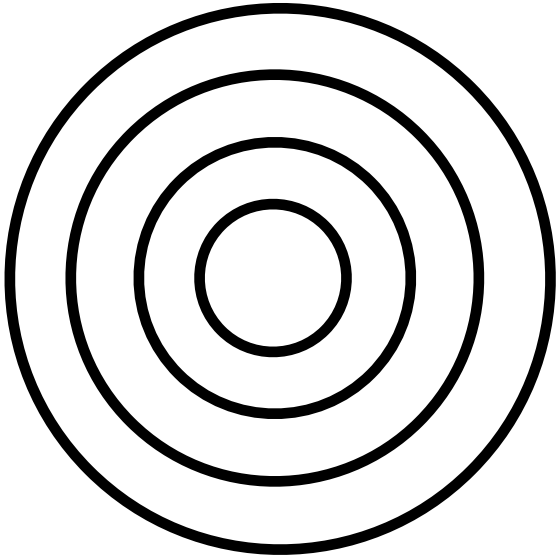
High variance

How much does classifier depend on training data?

Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models





High bias

Hypothesis cannot represent target function

Low bias

Hypothesis can represent target function

Each dot is a model that is learned from a a different dataset

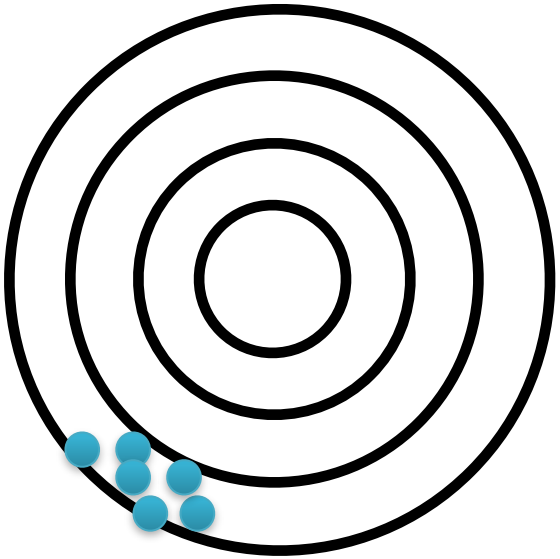
Low variance

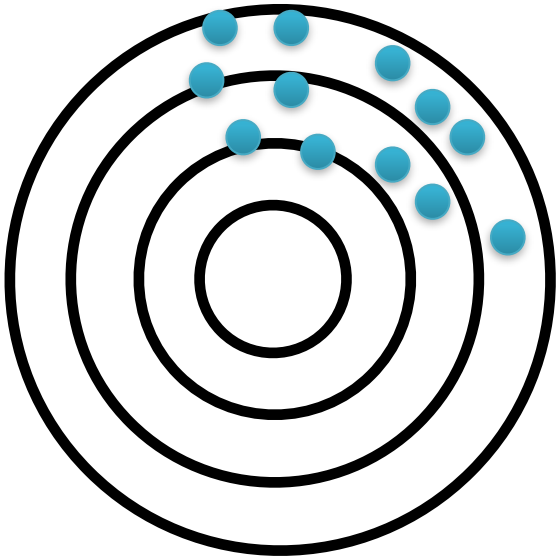
High variance

How much does classifier depend on training data?

Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models





High bias

Hypothesis cannot represent target function

Low bias

Hypothesis can represent target function

Each dot is a model that is learned from a a different dataset

Low variance

High variance

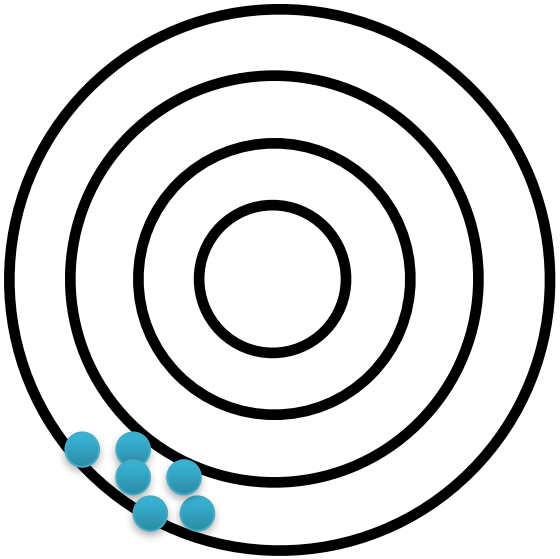
How much does classifier depend on training data?

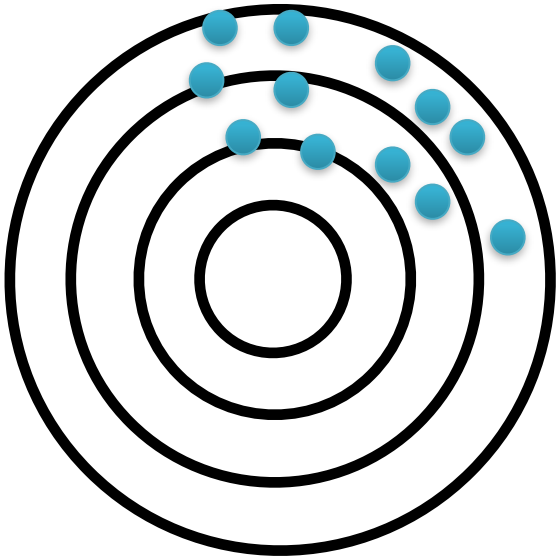
Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

High bias

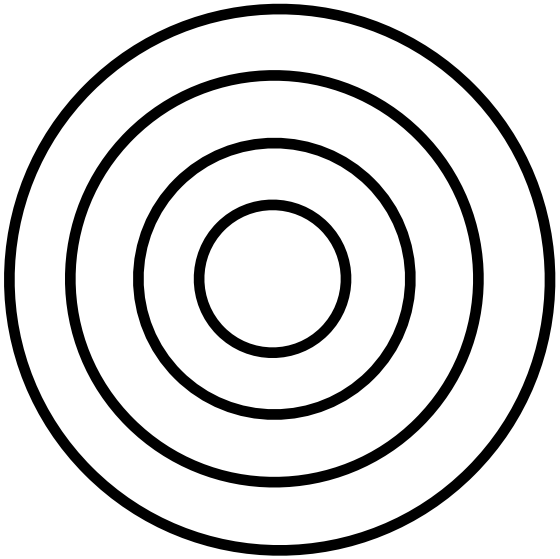
Hypothesis cannot represent target function





Low bias

Hypothesis can represent target function



Each dot is a model that is learned from a a different dataset

Low variance

High variance

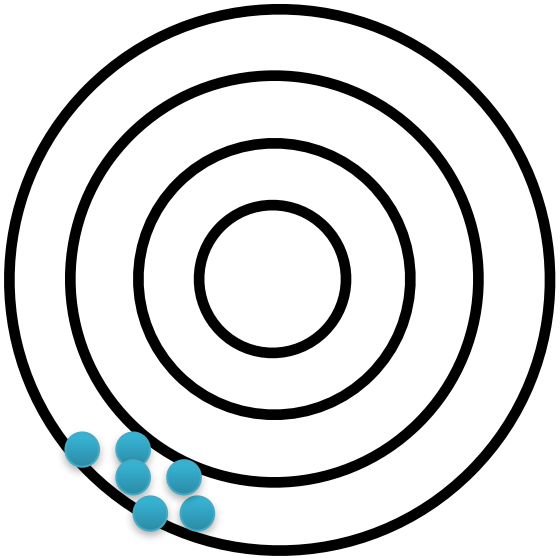
How much does classifier depend on training data?

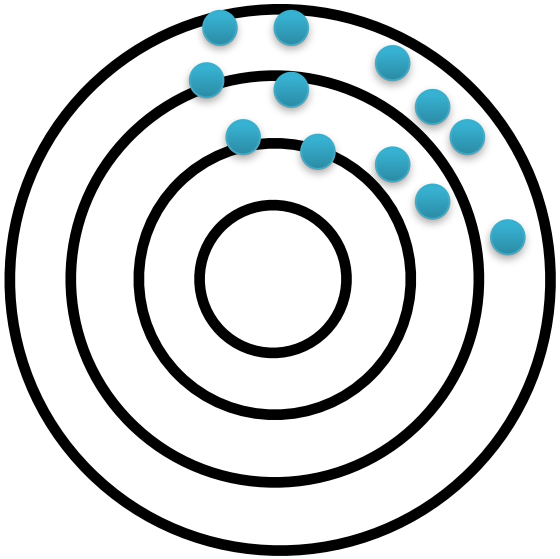
Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

High bias

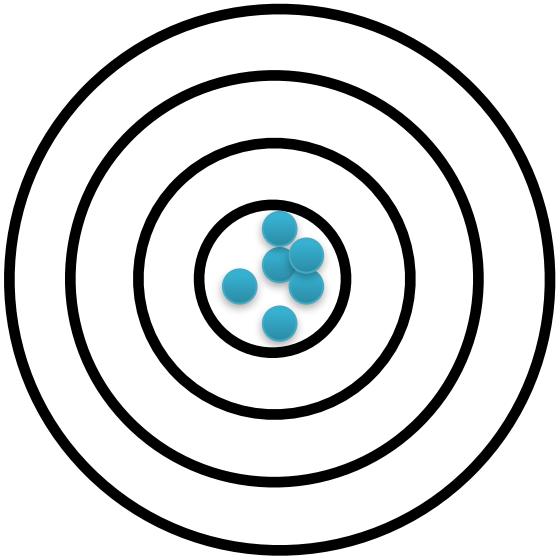
Hypothesis cannot represent target function





Low bias

Hypothesis can represent target function



Each dot is a model that is learned from a a different dataset

Low variance

High variance

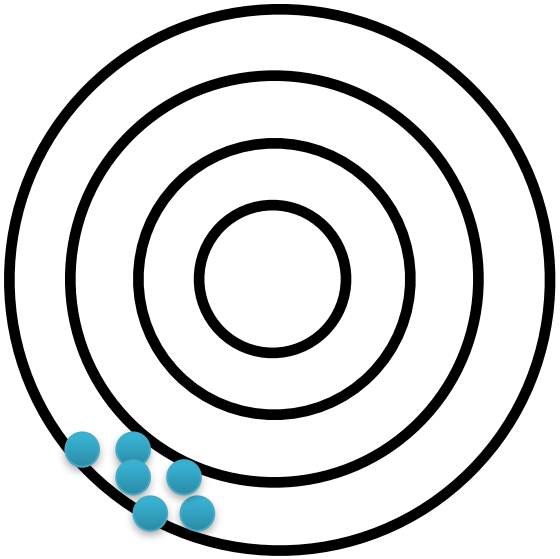
How much does classifier depend on training data?

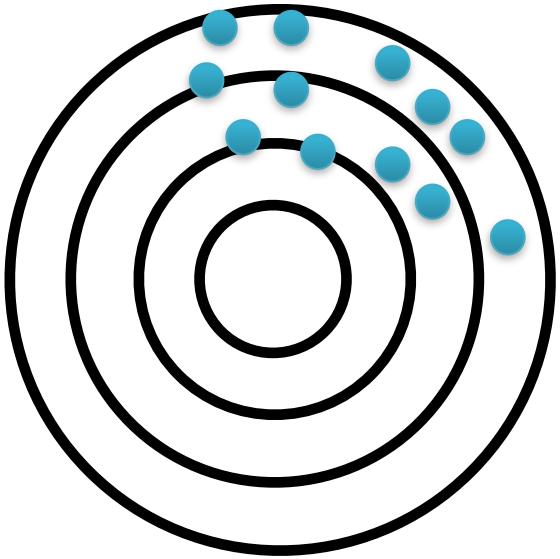
Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

High bias

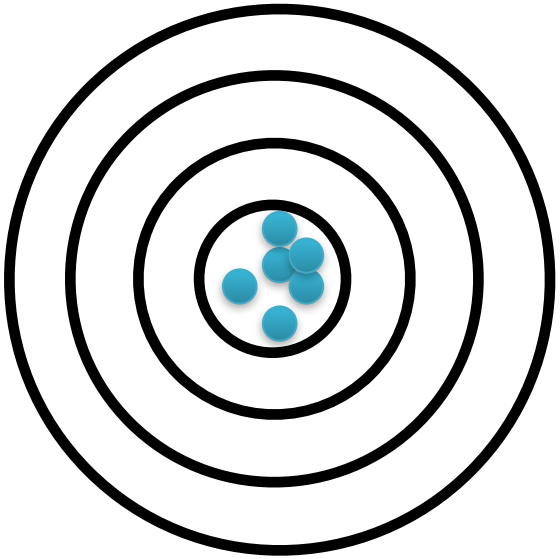
Hypothesis cannot represent target function

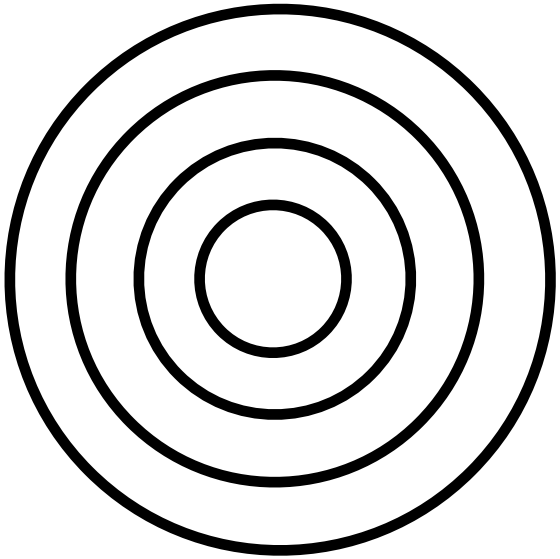




Low bias

Hypothesis can represent target function





Each dot is a model that is learned from a a different dataset

Low variance

High variance

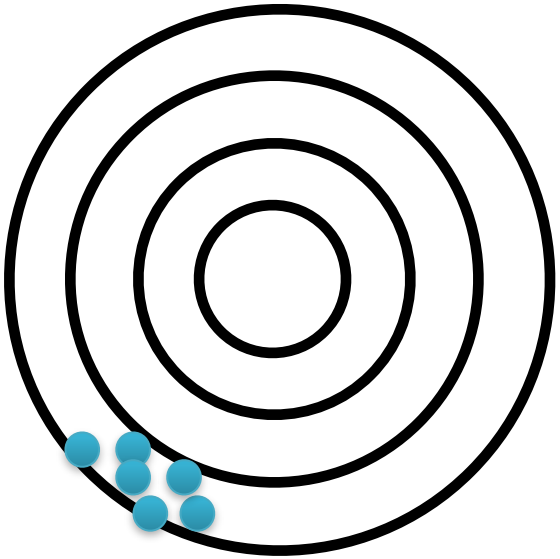
How much does classifier depend on training data?

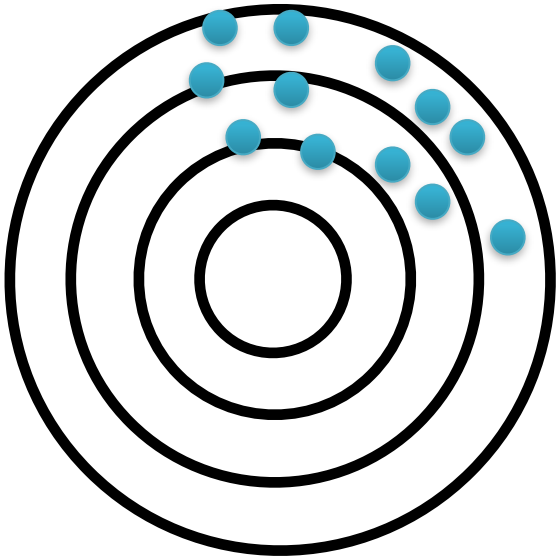
Let's play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

High bias

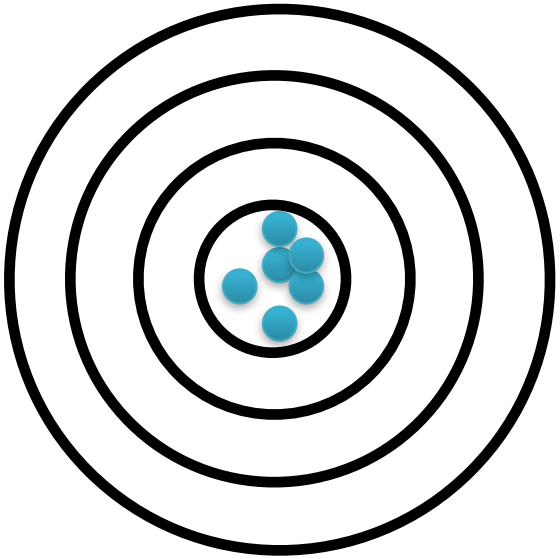
Hypothesis cannot represent target function

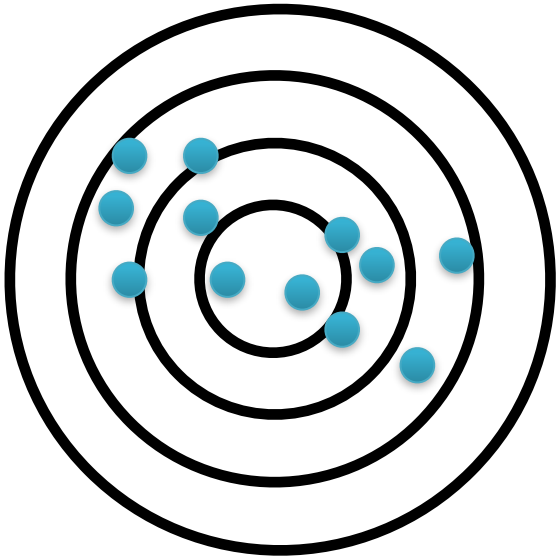




Low bias

Hypothesis can represent target function





Each dot is a model that is learned from a a different dataset

Low variance

High variance

How much does classifier depend on training data?

Managing of bias and variance

Ensemble methods reduce variance

- Multiple classifiers are combined
- E.g., bagging, boosting

Decision trees of a given depth

- Increasing depth decreases bias, increases variance

Neural networks (aka multi-layer perceptron)

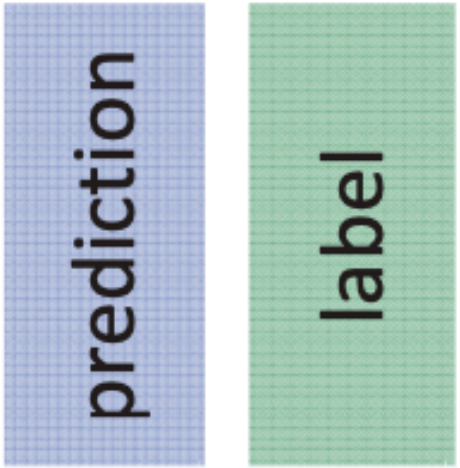
- Deeper models (more layers) decrease bias but can increase variance

Evaluation

OTHER METRICS

Evaluation metrics

To evaluate model, compare predicted labels to actual

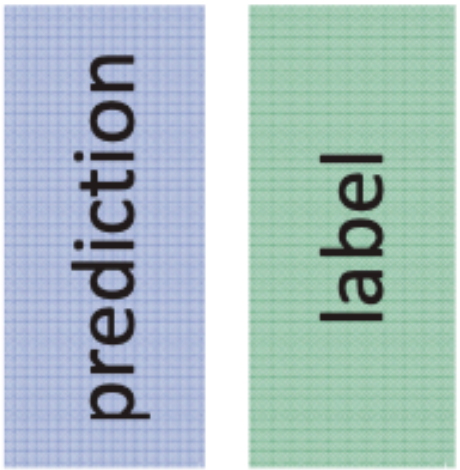


Accuracy: proportion of examples where we predicted correct label

$$\text{accuracy} = \frac{\#\text{ correct predictions}}{\#\text{ test instances}}$$

Evaluation metrics

To evaluate model, compare predicted labels to actual



Accuracy: proportion of examples where we predicted correct label

$$\text{accuracy} = \frac{\#\text{ correct predictions}}{\#\text{ test instances}}$$

Error: proportion of examples where we predicted incorrect label

$$\text{error} = 1 - \text{accuracy}$$

$$= \frac{\#\text{ incorrect predictions}}{\#\text{ test instances}}$$

Problems

What if the number of observations for one class is different than the number of observations for another class?

What if you have more than 2 classes? How do you know whether all classes are being predicted equally well?

E.g., are you predicting +1 labels with the same accuracy as you predict -1 labels?

Confusion matrices

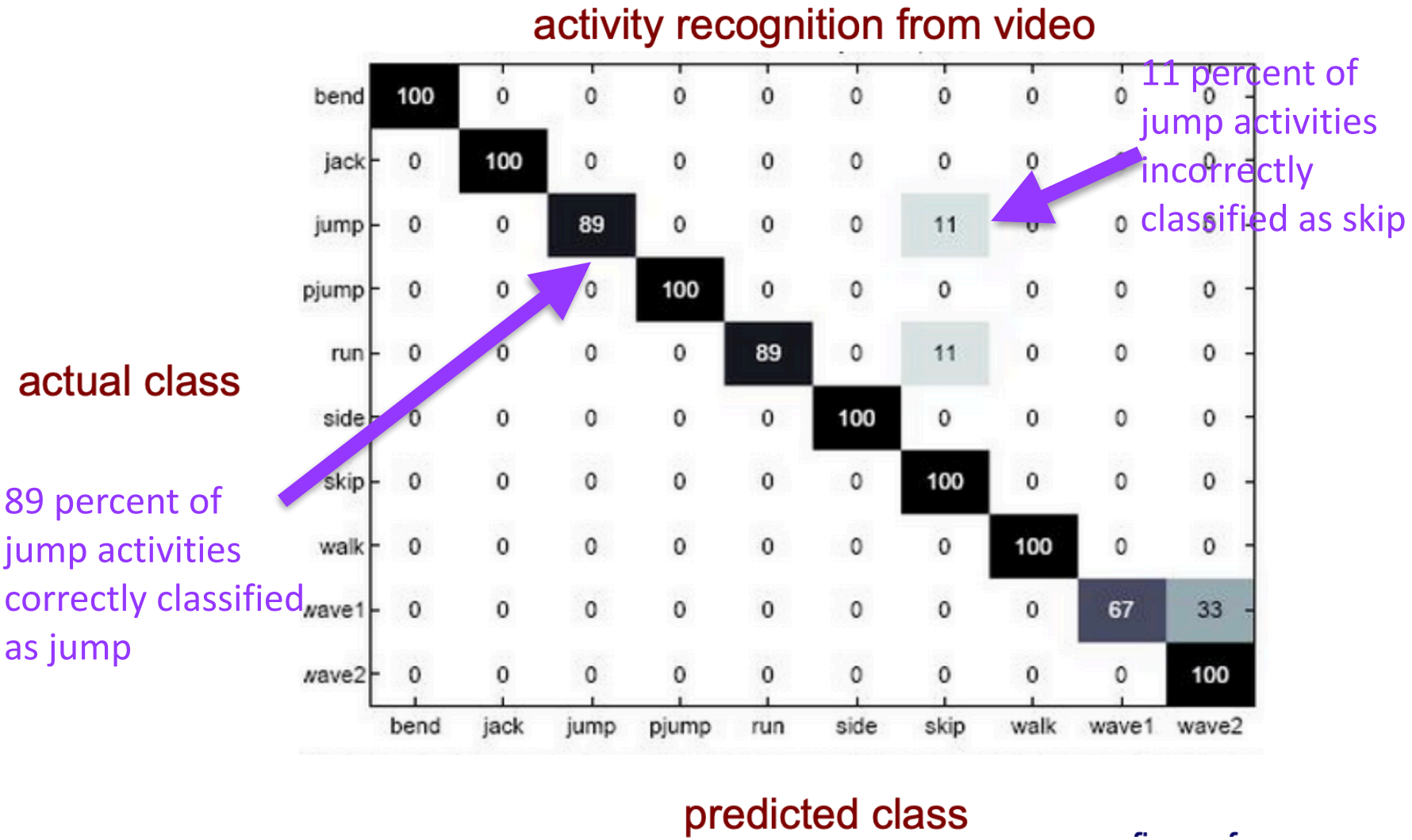
Help us understand what types of mistakes are made by a learned model

Confusion matrix

- Summarizes performance of a classification model
- Shows ways in which your model is confused (types of errors made) when model makes predictions

Confusion matrices

Table contains counts of correct and incorrect classifications



from http://vision.jhu.edu/

Confusion matrix for 2-class problems

Imagine a classifier that identifies presence of disease

		predicted class: Yes	predicted class: No
actual class	Yes	TP	FN
actual class	No	FP	TN

True Positive
TP = person tests positive and really has disease

True Negative
TN = person tests negative and really does not have disease

False Positive
FP = person tests positive and does not have disease

False Negative
FN = person tests negative and has disease

Confusion matrix for 2-class problems

Imagine a classifier that identifies presence of disease

actual class \ predicted class	Yes	No
Yes	TP	FN
No	FP	TN

True Positive
TP = person tests positive and really has disease

True Negative
TN = person tests negative and really does not have disease

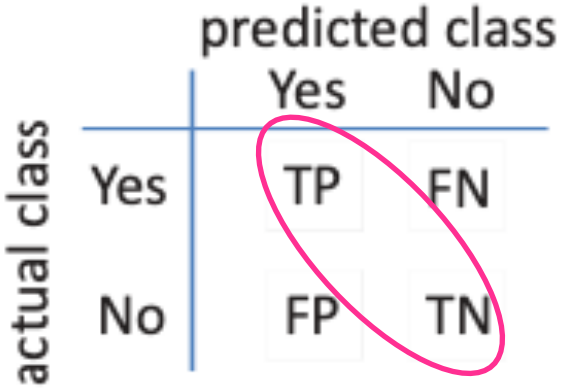
False Positive
FP = person tests positive and does not have disease

False Negative
FN = person tests negative and has disease

How do we compute accuracy?

Confusion matrix for 2-class problems

Imagine a classifier that identifies presence of disease



$$accuracy = \frac{TP + TN}{P + N}$$

True Positive
TP = person tests positive and really has disease

True Negative
TN = person tests negative and really does not have disease

False Positive
FP = person tests positive and does not have disease

False Negative
FN = person tests negative and has disease

P = actual class is positive = TP + FN

N = actual class is negative = TN + FP

FN: has disease

FP: does not have disease

Is accuracy an adequate measure of performance?

Accuracy may not be useful measure in cases where …

There is a large class skew

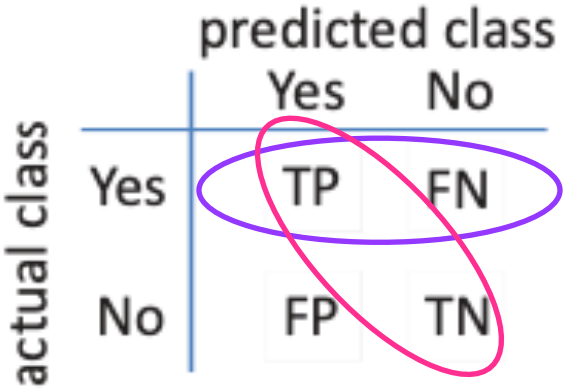
- Is 98% accuracy good if 97% of the instances are negative?
- If we just always guessed negative that would give us 97% accuracy!

There are differential misclassification costs – say, getting a positive wrong costs more than getting a negative wrong

- Consider a medical domain in which a false positive results in an extraneous test but a false negative results in a failure to treat a disease

Confusion matrix for 2-class problems

Given a dataset of P positive instance and N negative instances



$$accuracy = \frac{TP + TN}{P + N}$$

Imagine a classifier that identifies presence of disease

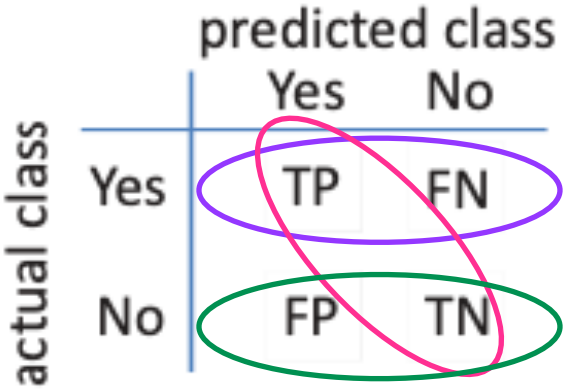
$$sensitivity = \frac{TP}{TP + FN}$$

(true positive rate) = probability of positive test given person has disease

How good is model at detecting positive cases?

Confusion matrix for 2-class problems

Given a dataset of P positive instance and N negative instances



$$accuracy = \frac{TP + TN}{P + N}$$

Imagine a classifier that identifies presence of disease

$$sensitivity = \frac{TP}{TP + FN}$$

(true positive rate) = probability of positive test given person has disease

How good is model at detecting positive cases?

$$specificity = \frac{TN}{TN + FP}$$

(true negative rate) = probability of negative test given person does not have disease

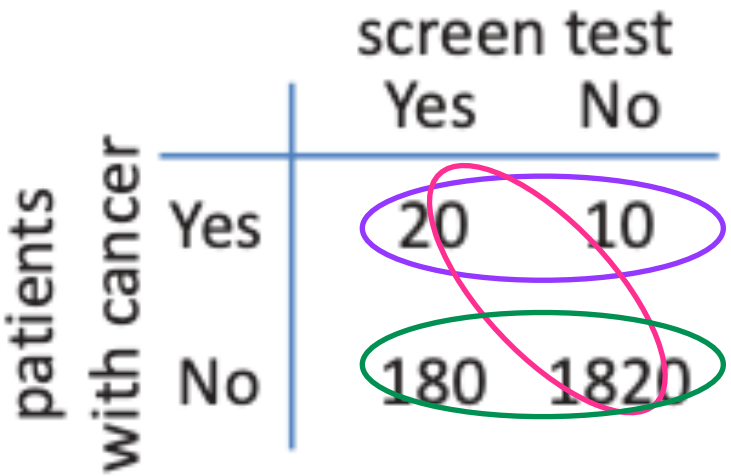
How good is model at detecting negative cases?

Confusion matrix: cancer dataset

patients with cancer	screen test: Yes	screen test: No
Yes	20	10
No	180	1820

Compute accuracy, sensitivity, and specificity

Confusion matrix: cancer dataset



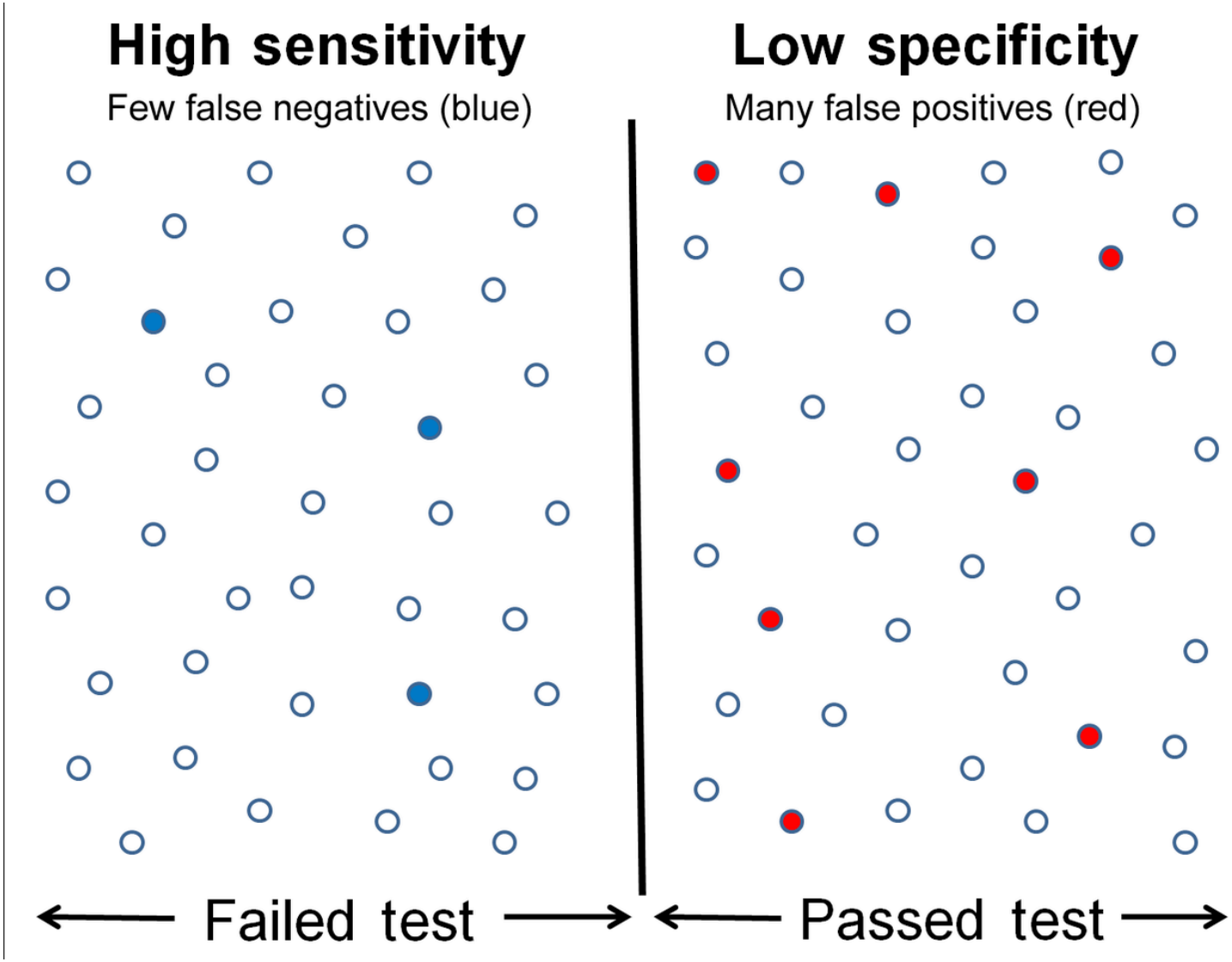
$$accuracy = \frac{TP + TN}{P + N}$$

$$sensitivity = \frac{TP}{TP + FN}$$

$$specificity = \frac{TN}{TN + FP}$$

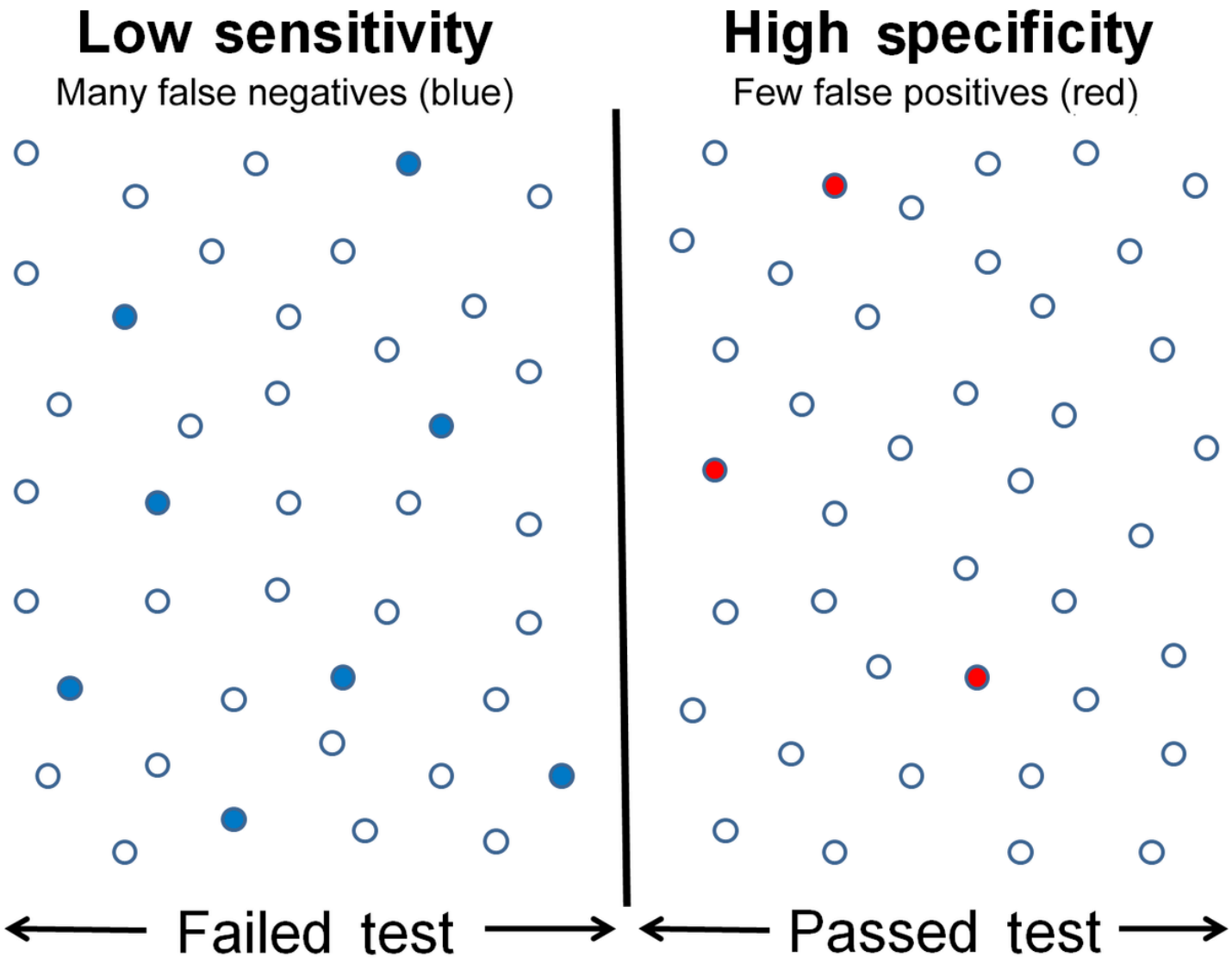
Compute accuracy, sensitivity, and specificity

Sensitivity vs. specificity



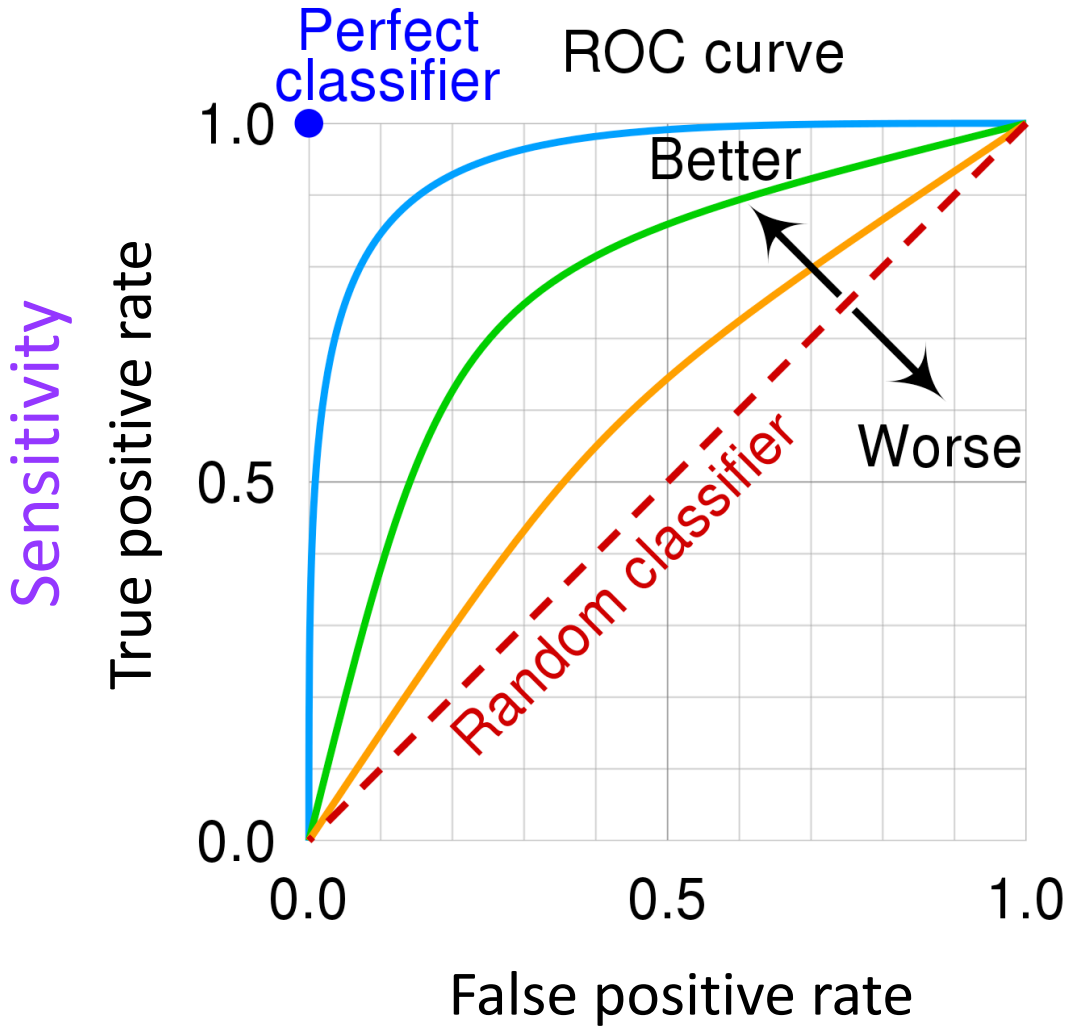
from https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Sensitivity vs. specificity



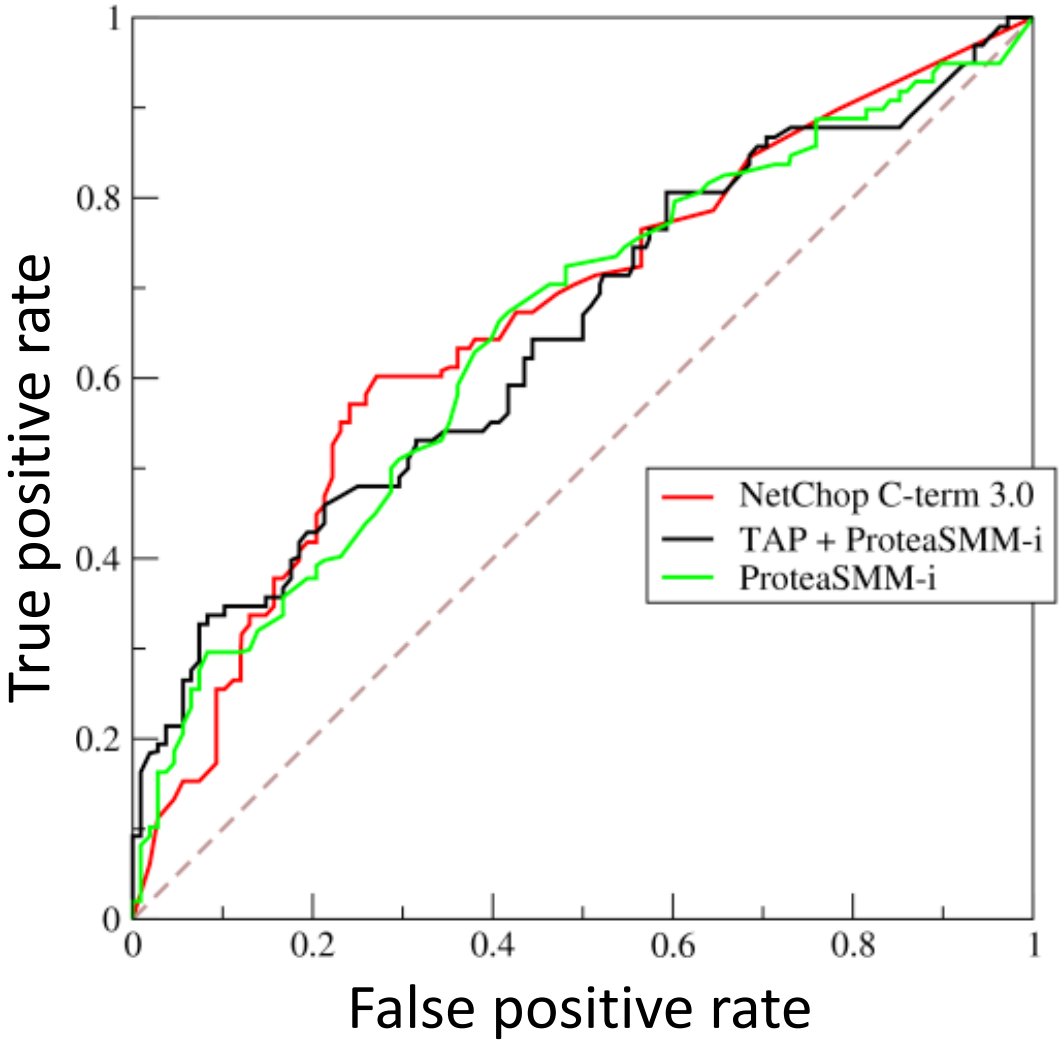
from https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Receiver Operating Characteristic (ROC) curve



from https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Receiver Operating Characteristic (ROC) curve



from https://en.wikipedia.org/wiki/Receiver_operating_characteristic