Exam 1 review

Jason G. Fleischer, Ph.D.
Asst. Teaching Professor
Department of Cognitive Science, UC San Diego

jfleischer@ucsd.edu

@jasongfleischer

https://jgfleischer.com

Slides in this presentation are from material kindly provided by Sebastian Rashka

Exam format

Monday Feb 13

- On paper worksheets we will provide.
- Bring pen and/or pencil! Bring your ID!
- Questions
  - ~ 25x bubble sheet style MCQ (1 min each)
  - ~ 10x math or written response questions (1 to 3 min each)
    - Text: “explain this” or “compare and contrast”. 2 - 4 sentences.
    - Math: show your work for partial credit
- You may bring a cheat sheet … max 1x single-sided sheet of 8.5 x 11”

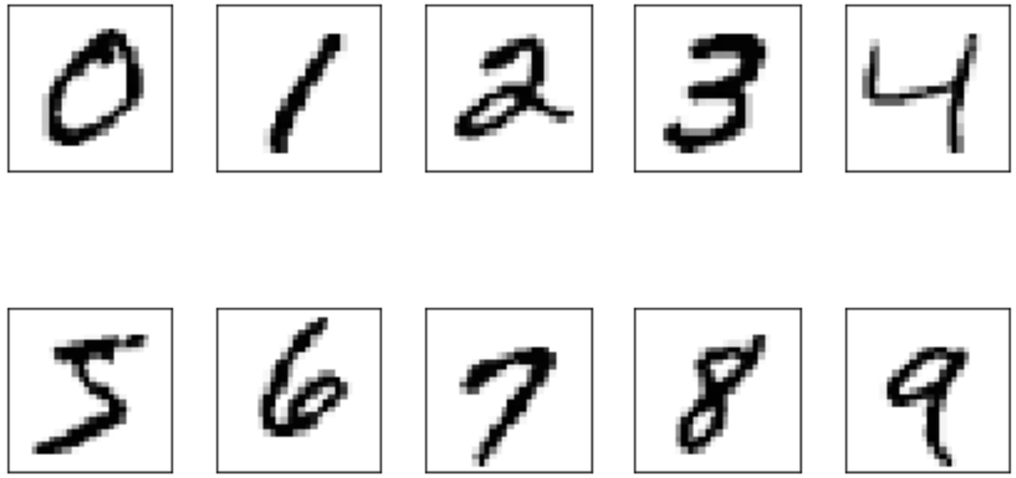
- If you are ill please wear a mask. I have them at the front if needed.
- If you have a question please raise your hand, we will come talk to you
- If its possible please give 1 seat space between you and neighbors (it won't work for everybody in this room, but try)
- Do not start exam until told to do so!
- Working after 2:50pm may earn you a points deduction
- Probably want to work on the "quicker questions" (multiple choice) at first, but time yourself knowing there are 4 "longer questions" that you can expect to take about 3-5 min each at the end
- On turning in: show your ID
- No electronic devices. No talking. Eyes on your own desk. Raise hands if you have a question.

Week 1

"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

— Tom Mitchell, Professor at Carnegie Mellon University

Handwriting Recognition Example:



- Task T: ?
- Performance measure P : ?
- Training experience E: ?

Supervised

- Labelled data
- Direct feedback
- Predict, classify, or fit a model

COGS 118A

Unsupervised

- No labels
- No feedback
- Find hidden structure using a model

COGS 118B

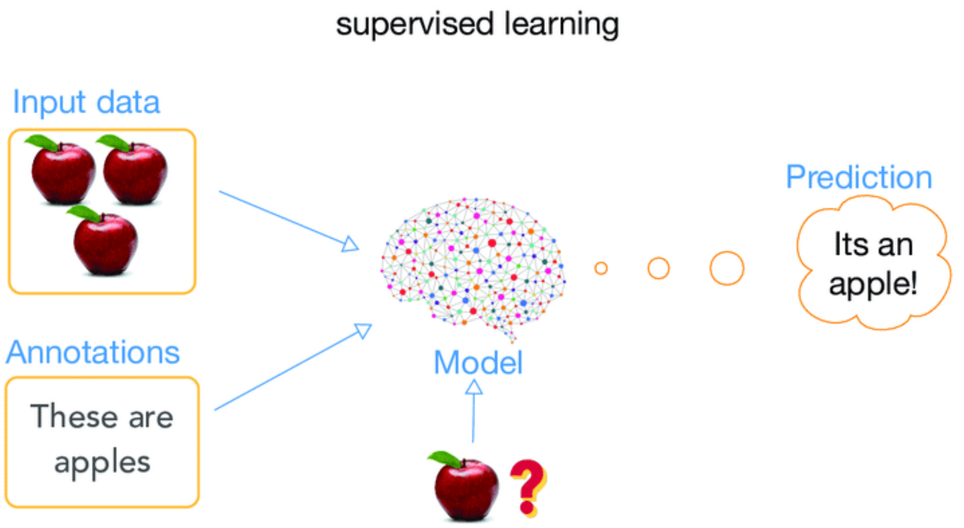
Reinforcement learning

- Feedback via reward (only label)
- Learns the series of actions that lead to reward

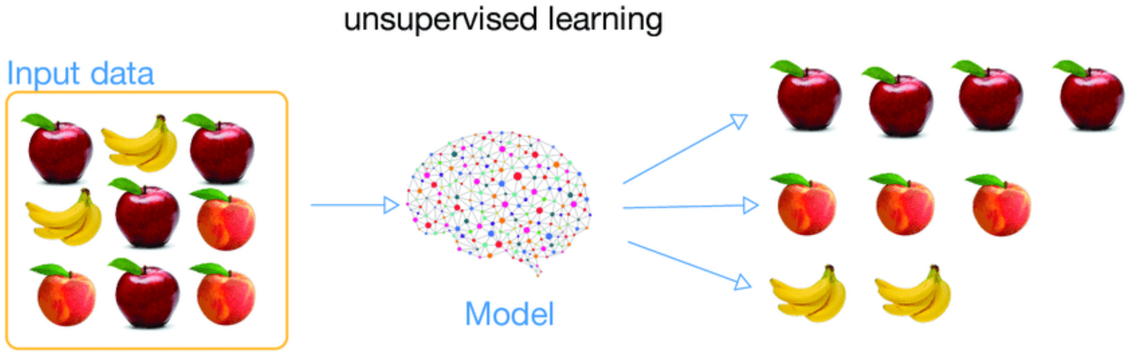
COGS 182

A false dichotomy? supervised vs unsupervised

COGS 118A

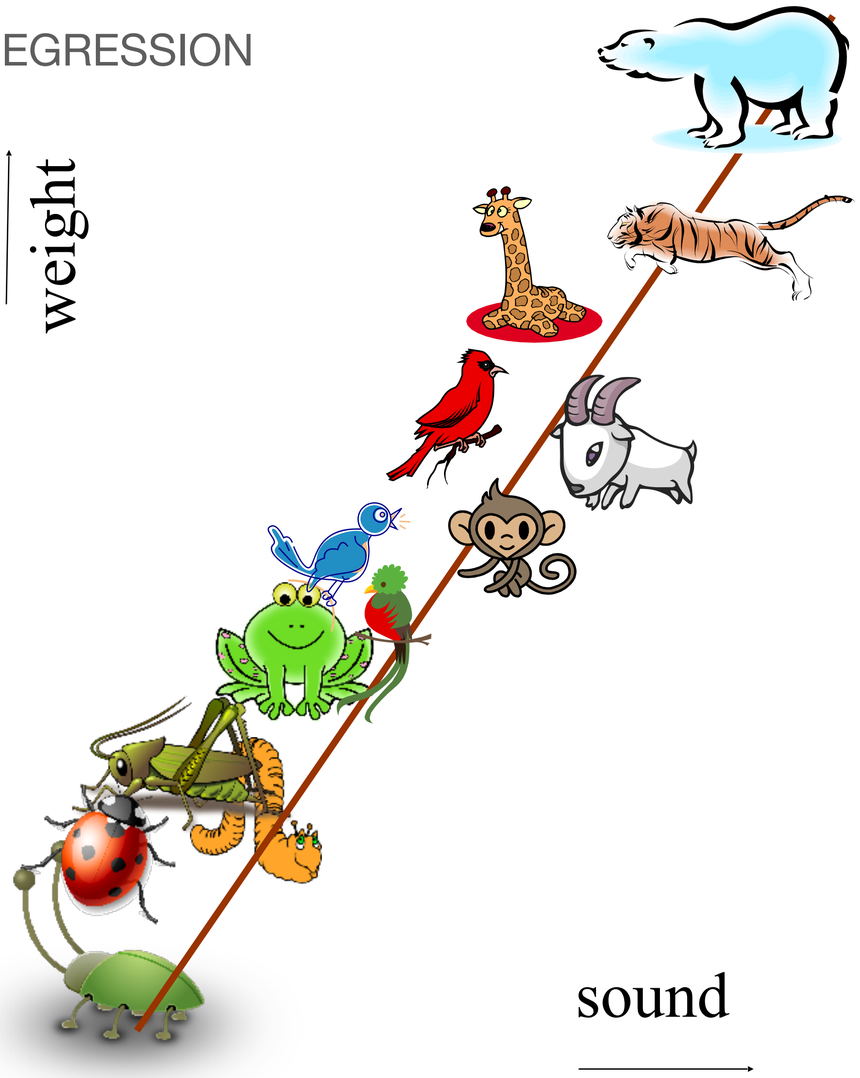


COGS 118B

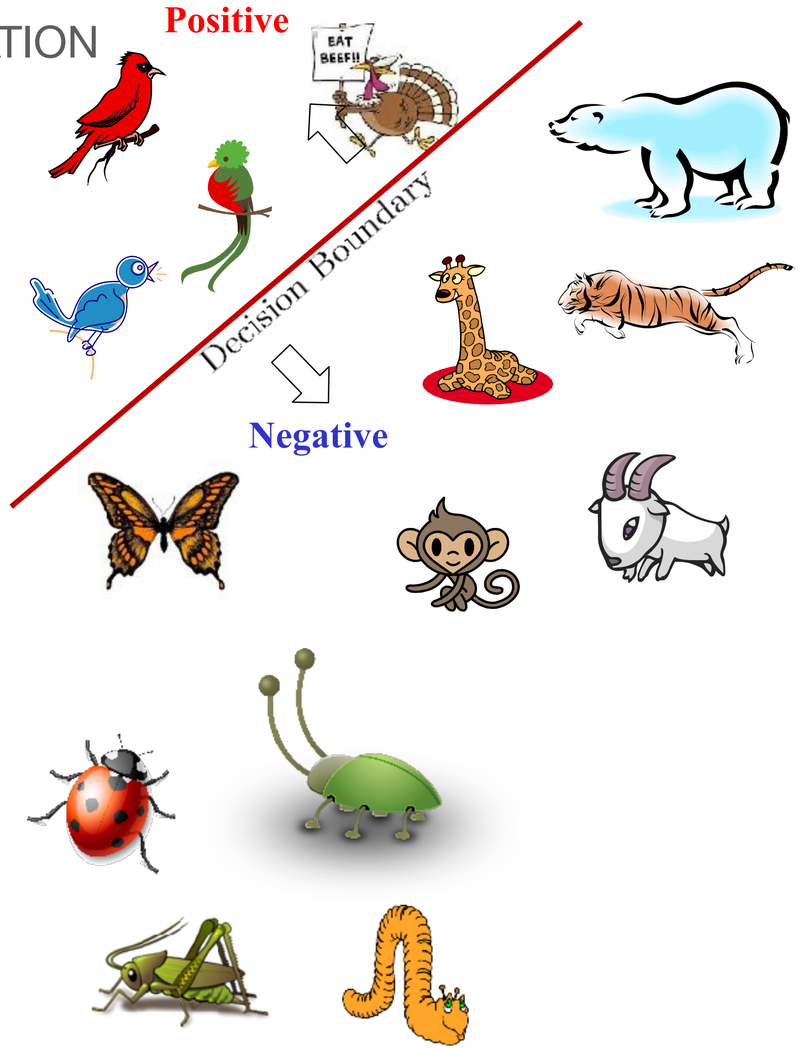


Another false dichotomy?

REGRESSION



CLASSIFICATION



Challenges of ML

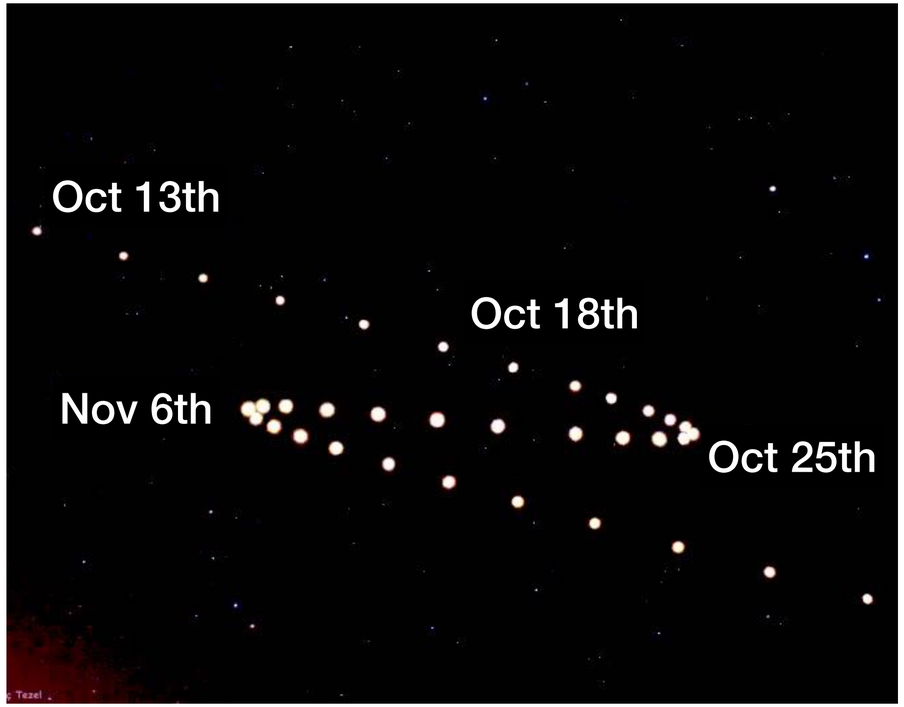
- Data
  - Insufficient
  - non-representative
  - poor quality
- Testing how good your model is
  - ML is too powerful - overfit!
  - Generalization error: the errors on new data not used in training
- Selecting a good enough/better/best ML
  - “Hyperparameter tuning”
  - “Model selection”
- How do you even measure performance? What’s the metric?

The process and art of machine learning
(At a small scale)

- Acquire data and knowledge about the subject
- Curate & clean the data
- Explore the data
- Make useful transformations of the data
- Split the data in different ways according to what you need to do
  - e.g., Train/test set split, Cross-validation, Nested cross-validation
  - More complexity of model-building, Less data —> more elaborate data splitting
- “DO ML”
- Evaluate and interpret your results

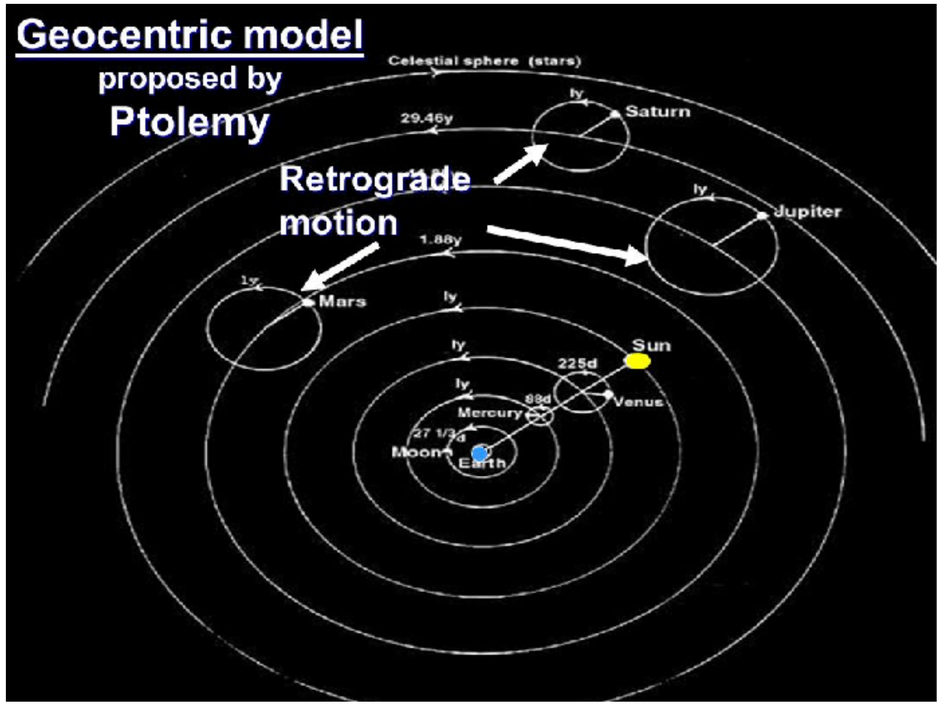
What is the goal?

Prediction



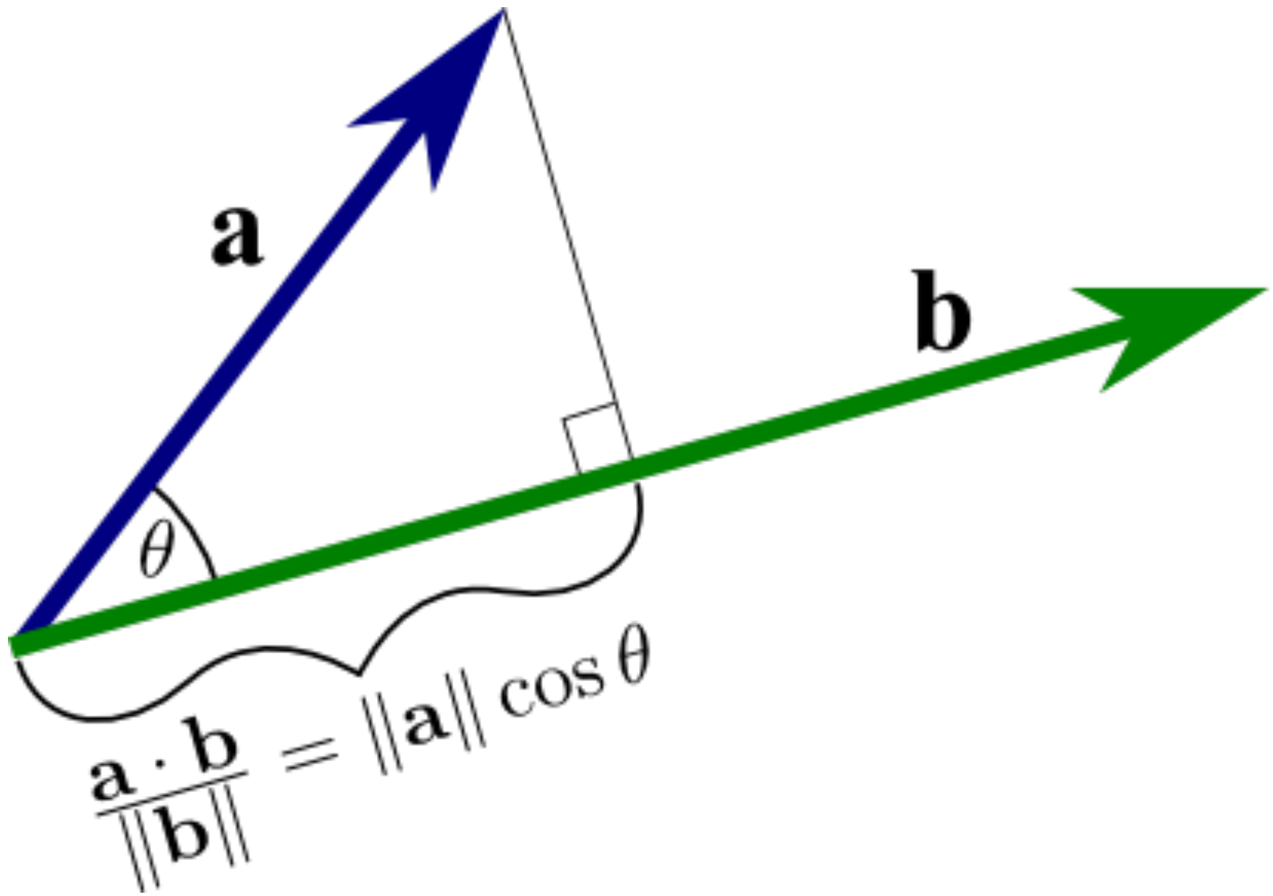
Only care that prediction y has low error

Modeling



We care that model w is an accurate representation of the real thing

Dot product - a scalar projection



$$\mathbf{a}\cdot\mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\,\cos\theta$$

$$\mathbf{a}\cdot\mathbf{b} \equiv \langle\mathbf{a},\mathbf{b}\rangle$$

$$\mathbf{a}\cdot\mathbf{b} \equiv \mathbf{a}^{\mathrm{T}}\mathbf{b}$$

Matrix multiplication

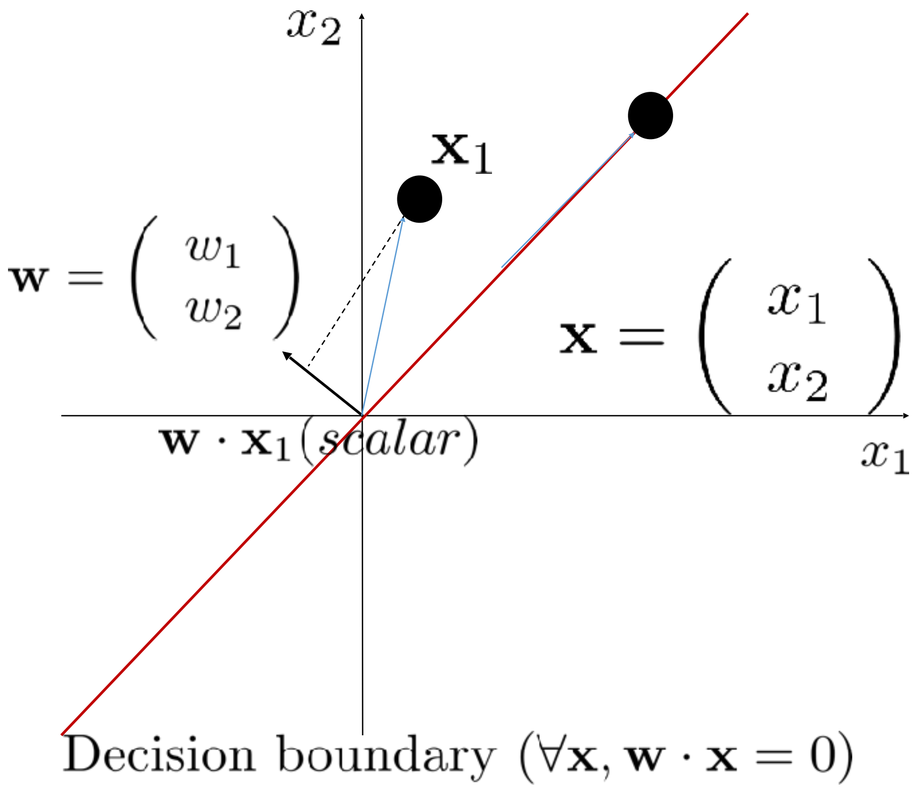
Matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix}$$

Line and vector



Any point $\mathbf{x}$ on the line satisfies:

$$\mathbf{w}^T\mathbf{x} \equiv < \mathbf{w}, \mathbf{x} > \equiv \mathbf{w} \cdot \mathbf{x} = 0$$

$\mathbf{w}$ is the normal direction of the line

Often: $||\mathbf{w}||_2 = 1$: a unit vector

Cosine similarity

	fly?	laying eggs?	weight (lb)
sparrow	yes	yes	0.087
chipmunk	no	no	0.19
bat	yes	no	0.09

Feature representation (one-hot encoded).

$$\text{sparrow} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0.087 \end{pmatrix} \qquad \text{chipmunk} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0.19 \end{pmatrix} \qquad \text{bat} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0.09 \end{pmatrix}$$

$$\frac{\text{sparrow} \cdot \text{chipmunk}}{||\text{sparrow}||_2 \times ||\text{chipmunk}||_2} = 0.0082$$

$$\frac{\text{sparrow} \cdot \text{bat}}{||\text{sparrow}||_2 \times ||\text{bat}||_2} = 0.502$$

$$\frac{\text{chipmunk} \cdot \text{bat}}{||\text{chipmunk}||_2 \times ||\text{bat}||_2} = 0.503$$

$\cdot$ refers to the dot product between two vectors;

$||\ ||_2$ refers to the L2 norm of a vector;

$\times$ refers to the multiplication of two scalar values.

Feature scaling is another factor

Now we purposely stretch one particular feature dimension by a large factor. Let's see what will happen.

$$\text{sparrow} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 87 \end{pmatrix} \qquad \text{chipmunk} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 190 \end{pmatrix} \qquad \text{bat} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 90 \end{pmatrix}$$

$$\frac{\text{sparrow} \cdot \text{chipmunk}}{||\text{sparrow}||_2 \times ||\text{chipmunk}||_2} = 0.99984$$

$$\frac{\text{sparrow} \cdot \text{bat}}{||\text{sparrow}||_2 \times ||\text{bat}||_2} = 0.99987$$

$$\frac{\text{chipmunk} \cdot \text{bat}}{||\text{chipmunk}||_2 \times ||\text{bat}||_2} = 0.99990$$

Now, the concept of similarity diminishes.
Conclusion: The relative scaling of the individual features is also important.

In practice, we often normalize the individual features to [0, 1] to make them directly comparable.

One hot encoding

Hot or not?

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Orthogonal!
No ordinality!

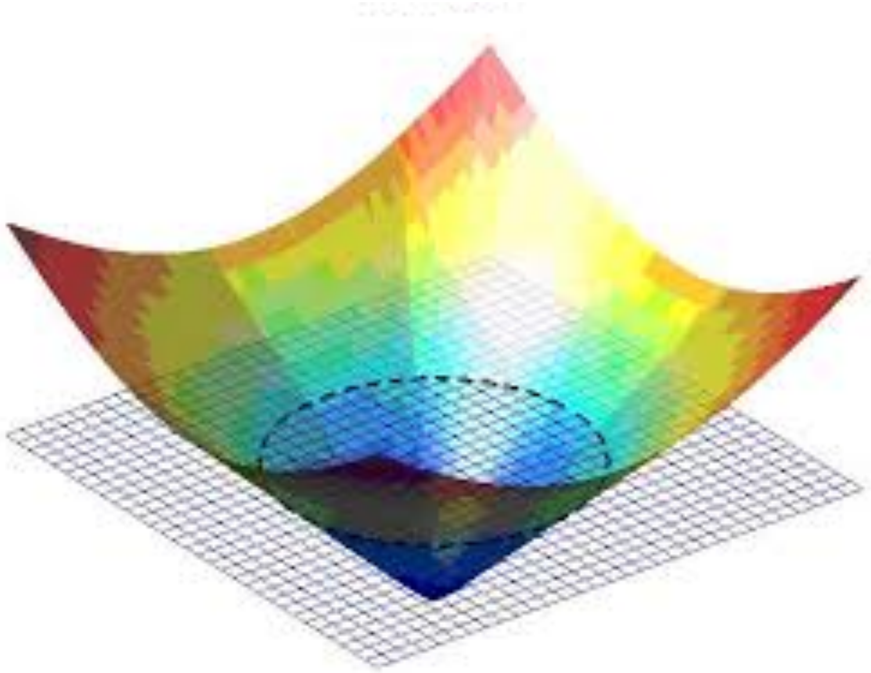
https://en.wikipedia.org/wiki/Matrix_calculus

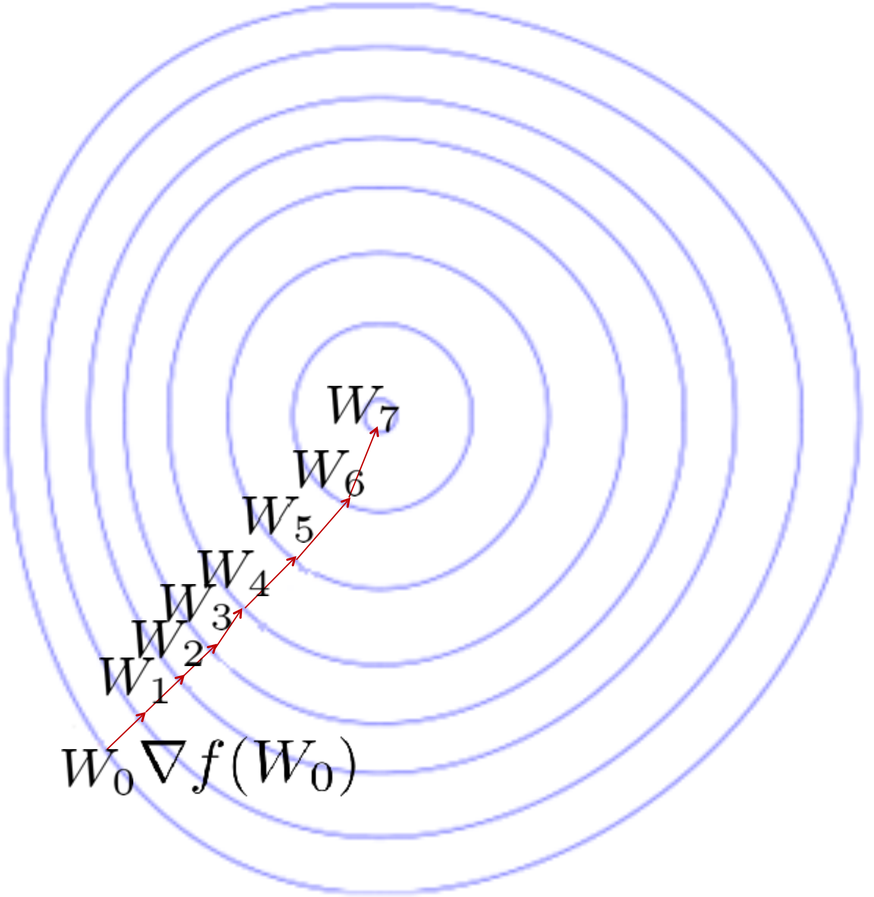
Identities: vector-by-vector $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

Condition	Expression	Numerator layout, i.e. by y and $\mathbf{x}^\mathsf{T}$	Denominator layout, i.e. by $\mathbf{y}^\mathsf{T}$ and x
a is not a function of **x**	$\frac{\partial \mathbf{a}}{\partial \mathbf{x}} =$	$\mathbf{0}$	
	$\frac{\partial \mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{I}$	
A is not a function of **x**	$\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{A}$	$\mathbf{A}^\top$
A is not a function of **x**	$\frac{\partial \mathbf{x}^\top \mathbf{A}}{\partial \mathbf{x}} =$	$\mathbf{A}^\top$	$\mathbf{A}$
a is not a function of **x**, **u** = **u**(**x**)	$\frac{\partial a\mathbf{u}}{\partial \mathbf{x}} =$	$a\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	
a = *a*(**x**), **u** = **u**(**x**)	$\frac{\partial a\mathbf{u}}{\partial \mathbf{x}} =$	$a\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \mathbf{u}\frac{\partial a}{\partial \mathbf{x}}$	$a\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial a}{\partial \mathbf{x}}\mathbf{u}^\top$
A is not a function of **x**, **u** = **u**(**x**)	$\frac{\partial \mathbf{A}\mathbf{u}}{\partial \mathbf{x}} =$	$\mathbf{A}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{A}^\top$
u = **u**(**x**), **v** = **v**(**x**)	$\frac{\partial (\mathbf{u}+\mathbf{v})}{\partial \mathbf{x}} =$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$	
u = **u**(**x**)	$\frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{x}} =$	$\frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{u}}$

a is not a function of **x**	$\frac{\partial (\mathbf{a}\cdot\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial (\mathbf{x}\cdot\mathbf{a})}{\partial \mathbf{x}} =$ $\frac{\partial \mathbf{a}^\top \mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^\top \mathbf{a}}{\partial \mathbf{x}} =$	$\mathbf{a}^\top$	$\mathbf{a}$
A is not a function of **x** **b** is not a function of **x**	$\frac{\partial \mathbf{b}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{b}^\top \mathbf{A}$	$\mathbf{A}^\top \mathbf{b}$
A is not a function of **x**	$\frac{\partial \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{x}^\top(\mathbf{A}+\mathbf{A}^\top)$	$(\mathbf{A}+\mathbf{A}^\top)\mathbf{x}$
A is not a function of **x** **A** is symmetric	$\frac{\partial \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$2\mathbf{x}^\top \mathbf{A}$	$2\mathbf{A}\mathbf{x}$
A is not a function of **x**	$\frac{\partial^2 \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}^2} =$	$\mathbf{A}+\mathbf{A}^\top$	
A is not a function of **x** **A** is symmetric	$\frac{\partial^2 \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}^2} =$	$2\mathbf{A}$	

$$\mathscr{L}(\mathbf{w}) = \mathbf{y} - f(\mathbf{X}; \mathbf{w})$$



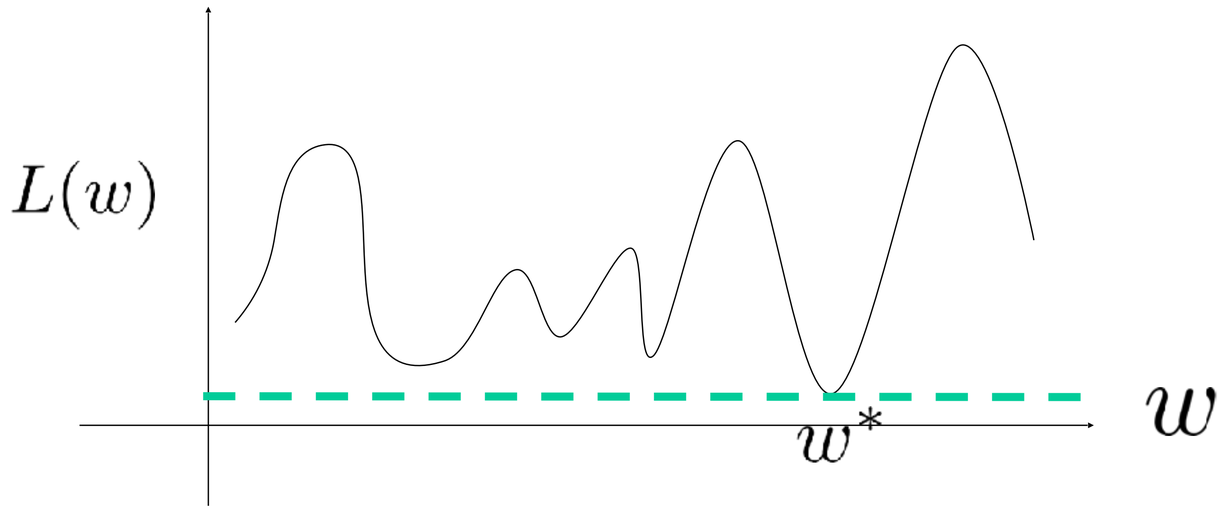


Optimization: argmin

$$w^* = \arg\min_w L(w)$$

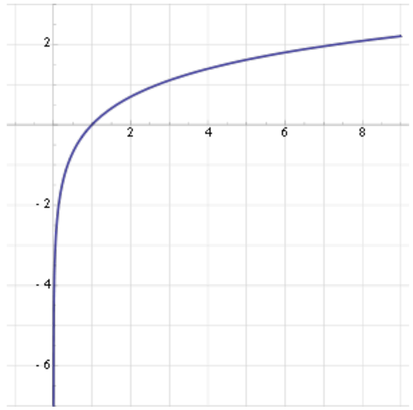
The operator $\arg\min$ defines the optimal value (in the argument of function $L()$) w^* that minimizes $L(w)$

$\arg\min L(w)$ doesn't return the value of $L(w)$



argmin and argmax

The function $\ln(v)$ is monotonically increasing, e.g. $\forall v_1 > v_2$ it is always true that $\ln(v_1) > \ln(v_2)$, then:



$$w^* = \arg\max_w G(w)$$

$$= \arg\max_w \ln(G(w))$$

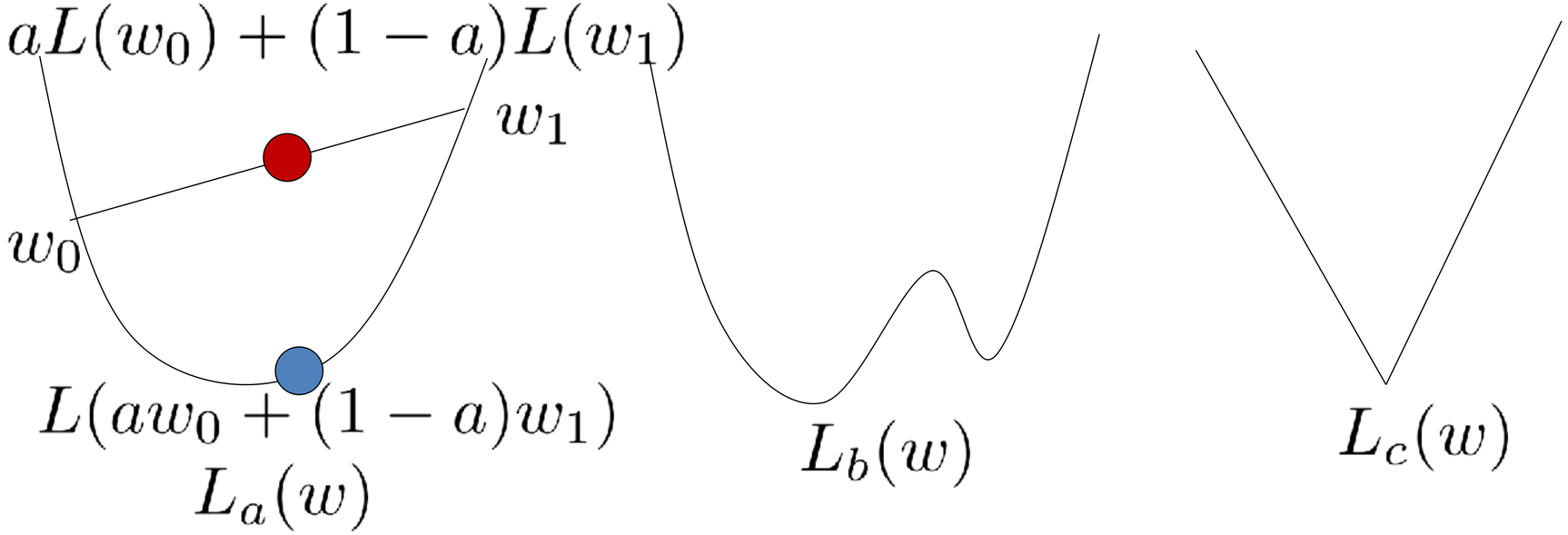
$$= \arg\min_w -\ln(G(w))$$

Applying a monotonic function to another function does not change its arg min or arg max!

Adding a constant to a function does not change its arg min or arg max!

Convex functions

$$w^* = \arg min_w \quad L(w)$$

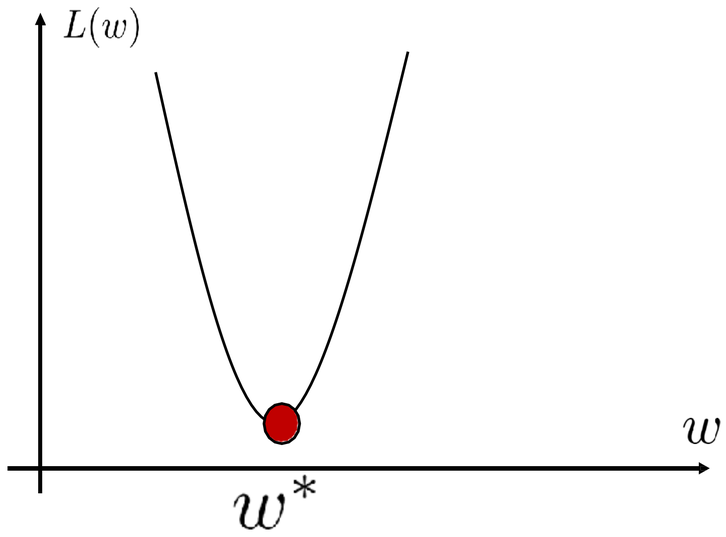


Definition:

$$\forall w_0, w_1, a \in [0, 1]$$

$$aL(w_0) + (1-a)L(w_1) \geq L(aw_0 + (1-a)w_1)$$

Convex functions that are differentiable



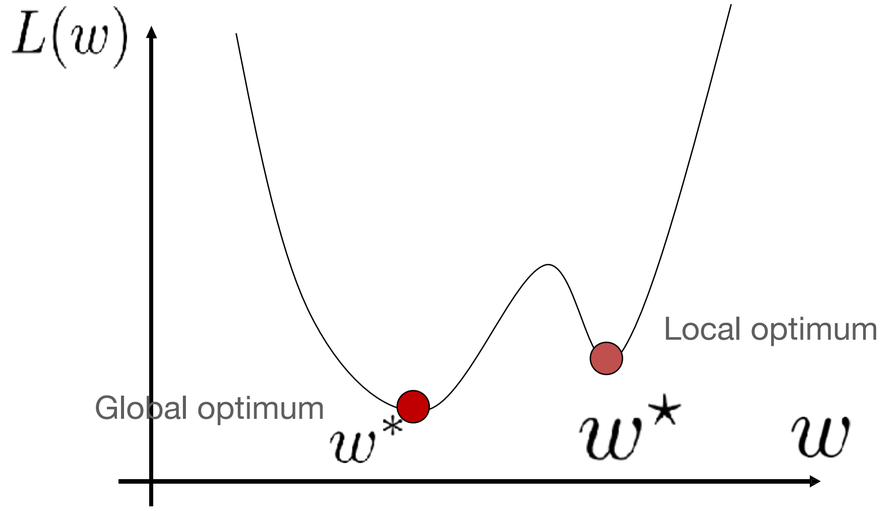
For a convex and differentiable function $L(w)$, it's global optimal is achieved at w^*,

$$\text{when } \frac{\partial L(w)}{\partial w}\Big|_{w^*} = 0$$

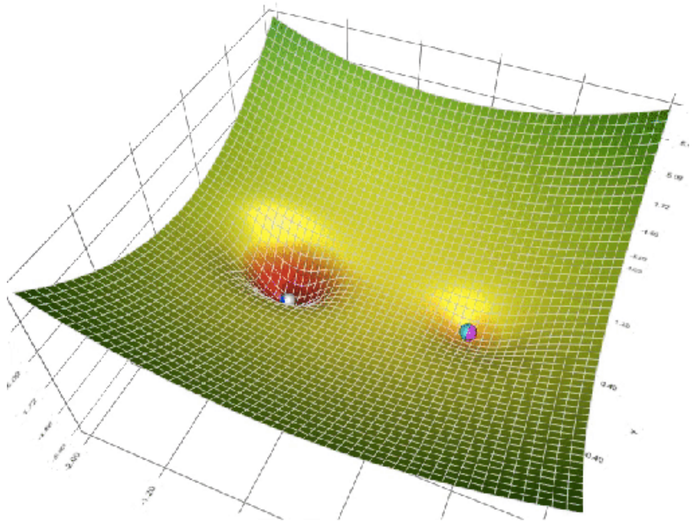
To find w^*, we simply solve for the equation:

$$\frac{\partial L(w)}{\partial w} = 0$$

Non-convex functions that are differentiable



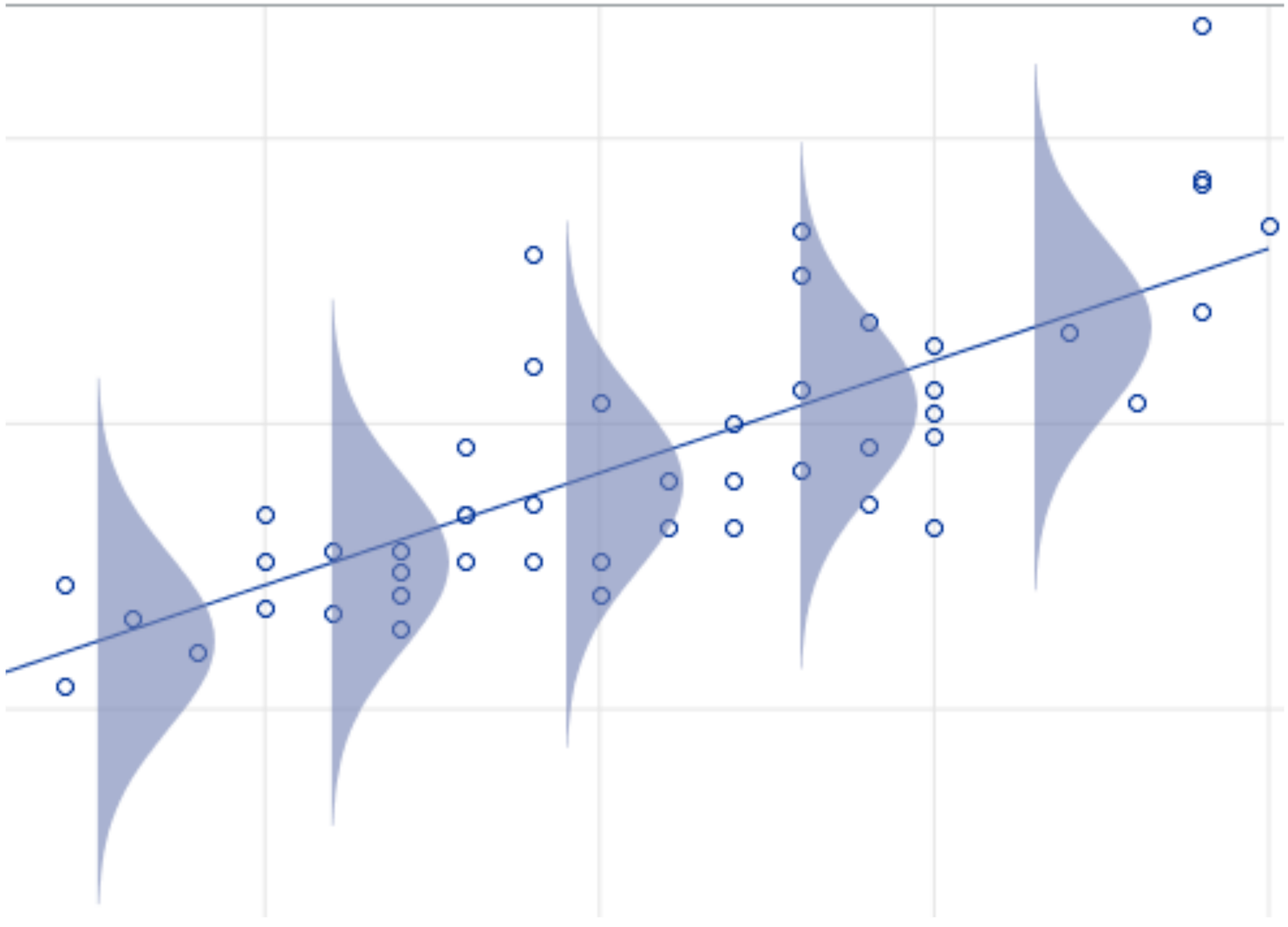
Compute $\frac{\partial L(w)}{\partial w}$ to solve the problem iteratively through gradient descent



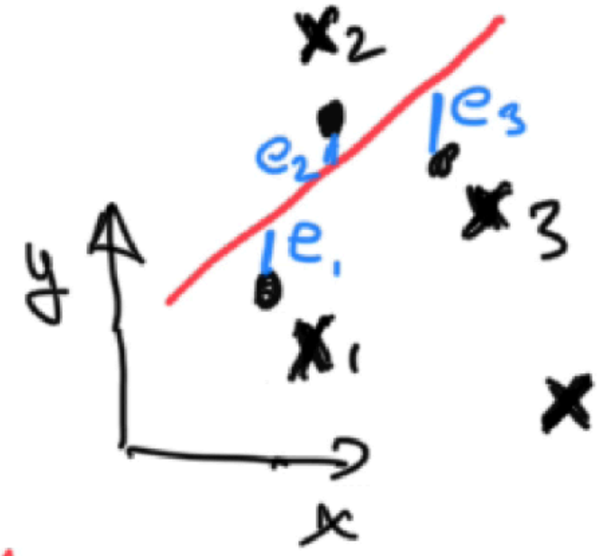
Week 2

Regression

The intuition



Derive OLS



$$\mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} b \\ m \end{bmatrix}$$

$$\hat{y}_n = b + m x_n = \mathbf{x}_n^T \mathbf{w}$$

$$e_n = y_n - \hat{y}_n = y_n - \mathbf{x}_n^T \mathbf{w}$$

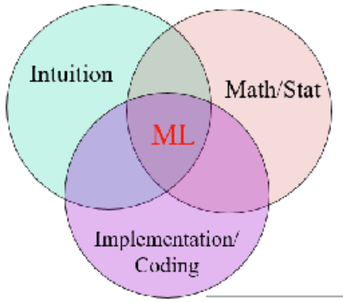
$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \end{bmatrix}$$

$$\hat{\mathbf{y}} = X\mathbf{w} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{bmatrix}$$

$$\mathbf{e} = \mathbf{y} - X\mathbf{w} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \end{bmatrix}$$

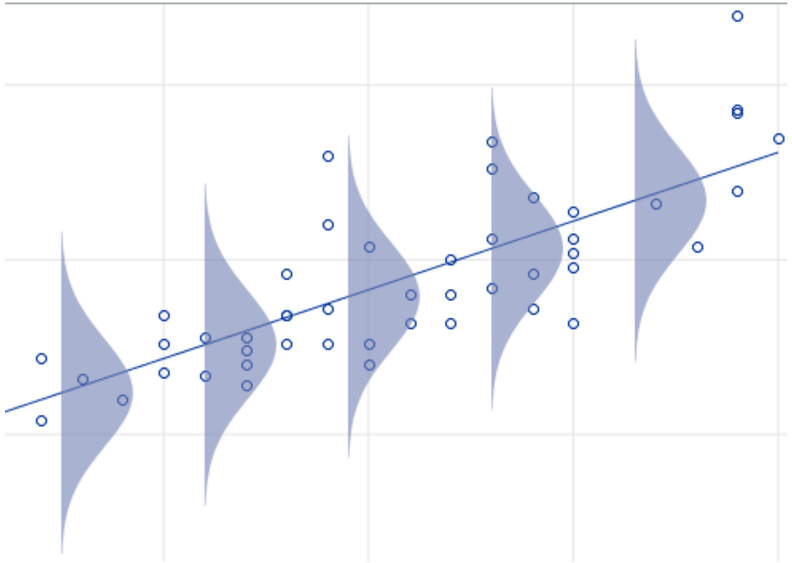
RSS - residual sum of squares

$$L(\mathbf{w}) = \mathbf{e}^T\mathbf{e} = \sum_{n=1}^{N} e_n^2$$

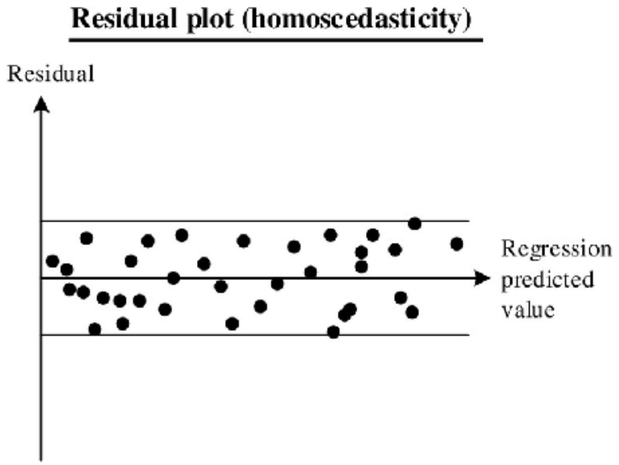


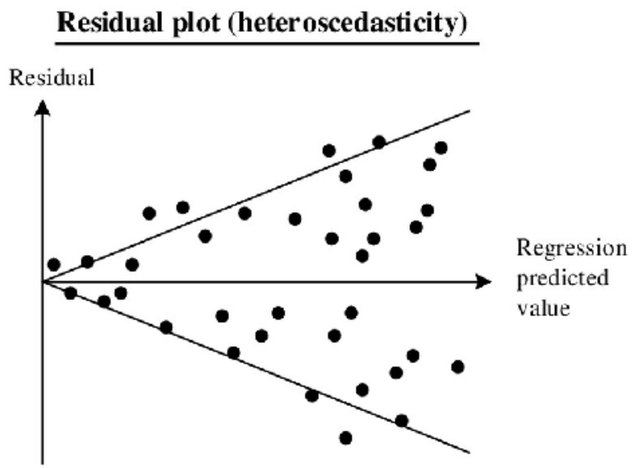
ASS-U-ME D properties

Linear Regression using Ordinary Least Squares

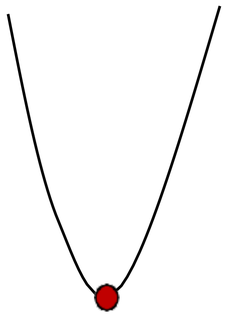


A1. The linear regression model is “linear in parameters.”
A2. There is a random sampling of observations.
A3. The conditional mean should be zero.
A4. No multi-collinearity (or perfect collinearity).
A5. Homoscedasticity
A6: Error terms should be normally distributed





Least squares estimation



Obtain/train: $f(x, W) = w_0 + w_1 x$

$W^* = \arg min_W \sum_i (\mathbf{x}_i^T \cdot W - y_i)^2$

$$W = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \quad \mathbf{x}_i = \begin{pmatrix} 1 \\ x_i \end{pmatrix}$$

$$W^* = \arg min_W = \arg min_W L(W) = (XW - Y)^T (XW - Y)$$

$$L(W) = W^T X^T XW - W^T X^T Y - Y^T XW + Y^T Y$$

$$\frac{dL(W)}{dW} = 2X^T XW - 2X^T Y = 0$$

$$W^* = (X^T X)^{-1} X^T Y$$

In [ ]:

For simplification $A = (X^T X)$

```
import numpy as np
from numpy.linalg import inv
# Comput X^T X and denote it as A
A = np.dot(np.transpose(X), X)
# Obtain optimal W
W = np.dot(inv(A),np.dot(np.transpose(X),Y))
```

Conclusions for linear regression with the least squares estimation

Linear regression:

- Univariate linear regression

Output: $y = w_0 + w_1 x_1$

- Polynomial linear regression

Output: $y = w_0 + w_1 x_1 + w_2 x_1^2 + ... + w_q x_m^q$

- Multivariate linear regression

Output: $y = w_0 + w_1 x_1 + w_2 x_2 + ... + w_m x_m$

They all share a general form (when generalizing X):

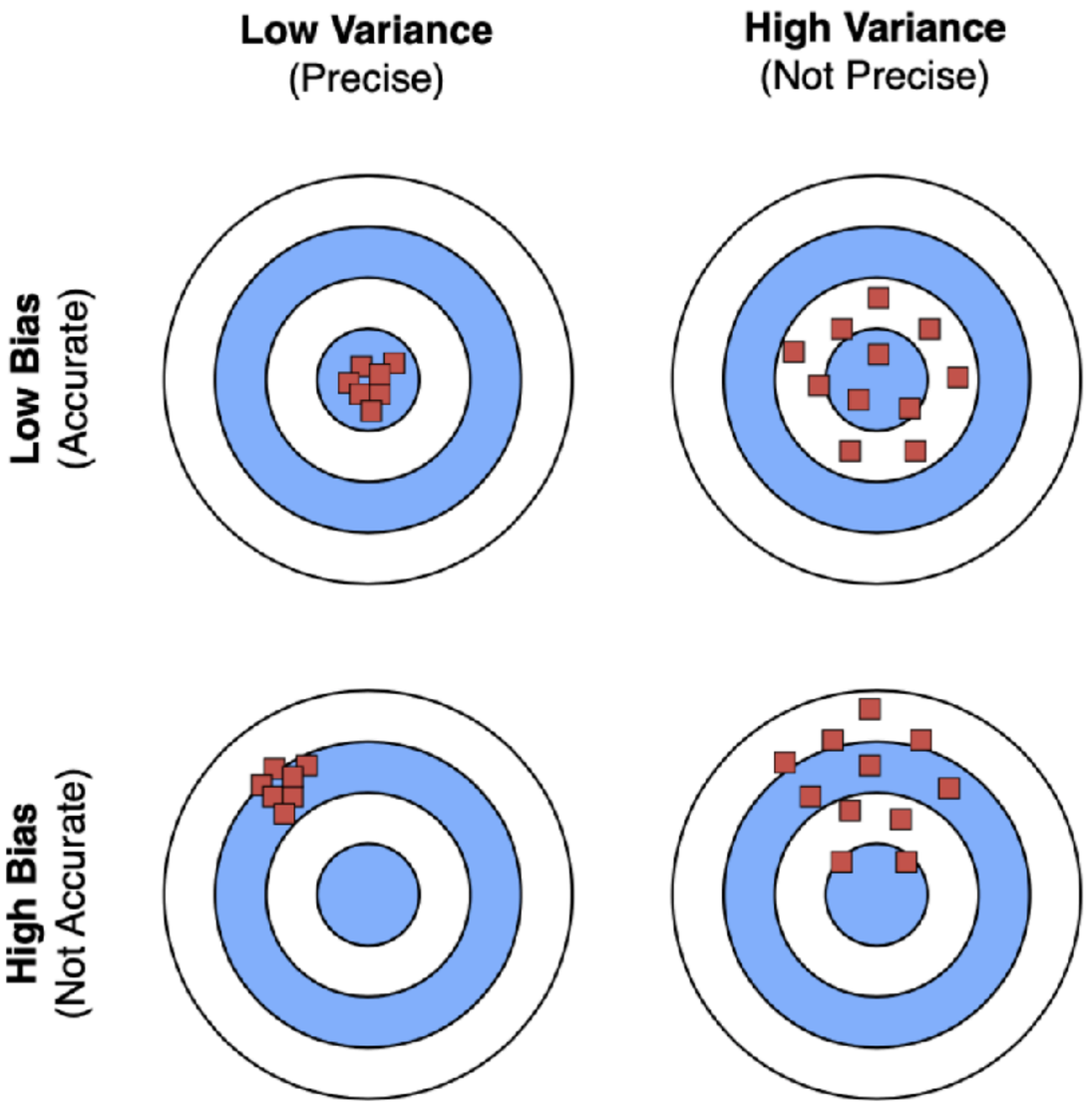
$$Y = X W$$

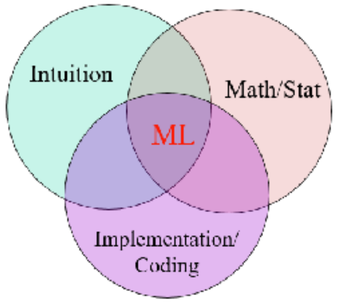
linear w.r.t. the W!

With an analytical solution:

$$W^* = (X^T X)^{-1} X^T Y$$

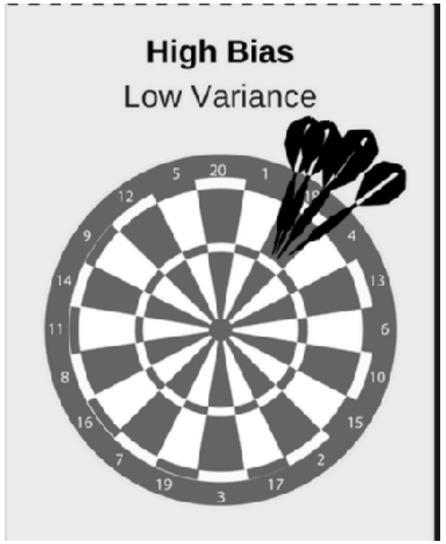
Bias Variance Tradeoff

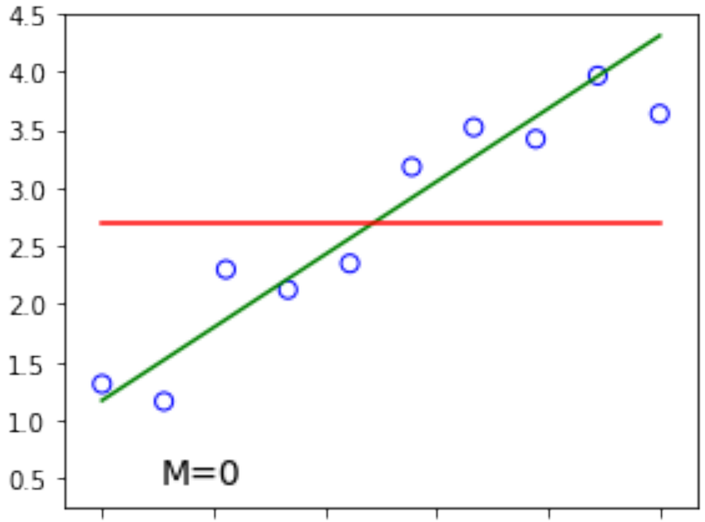




Intuition

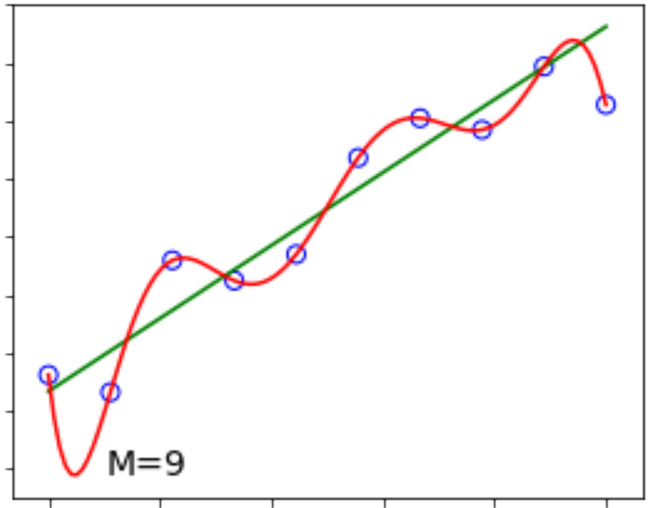
Bias-variance tradeoff in model complexity





Underfitting

(Model is too simple!)

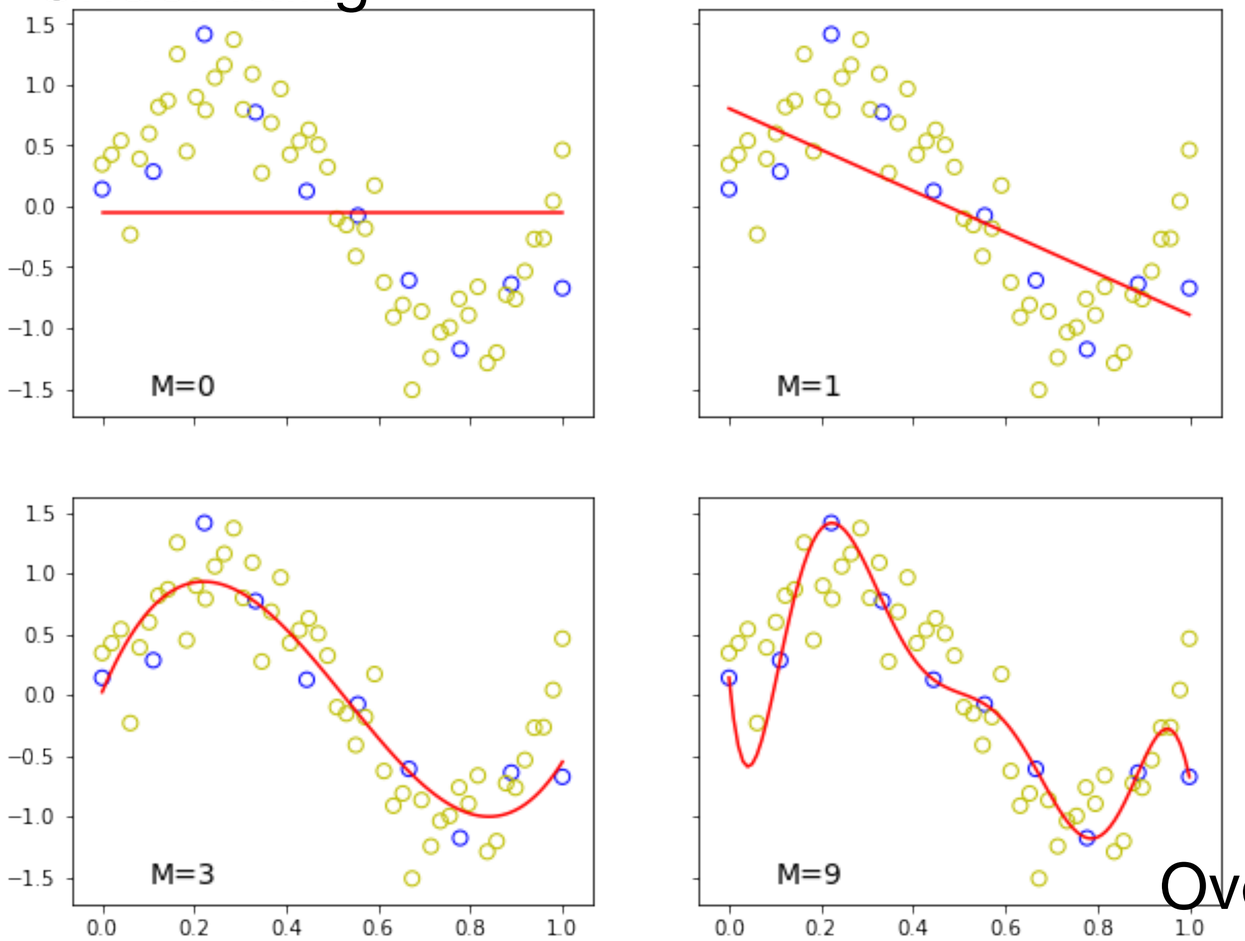


Overfitting

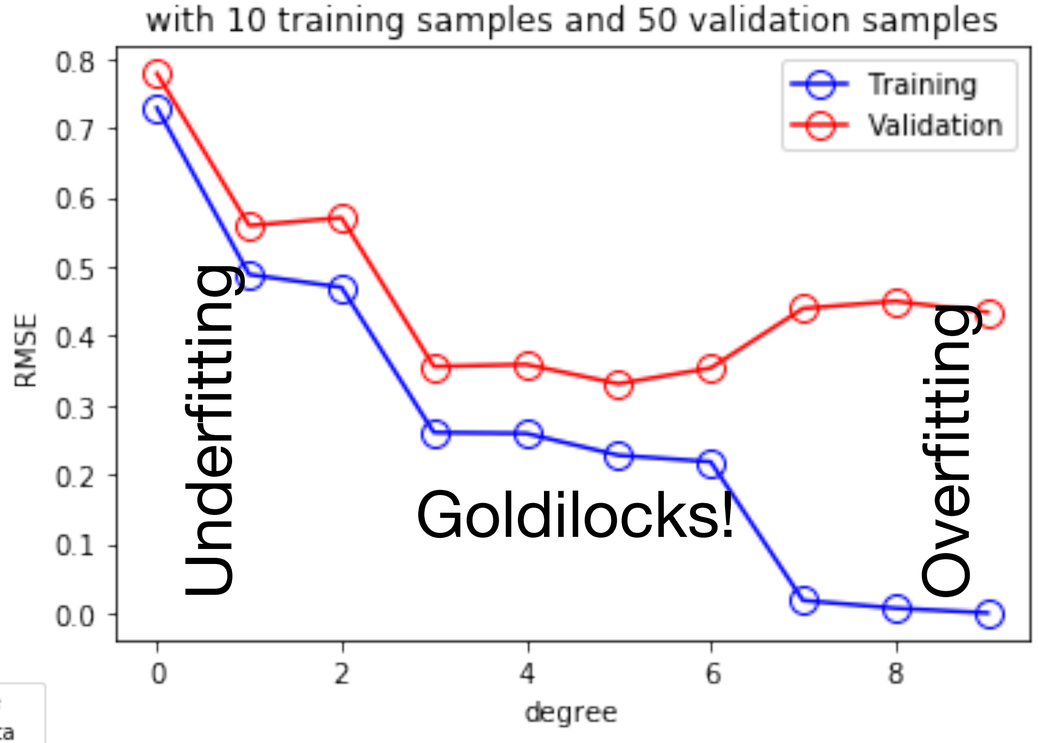
(Model is too complex!)

Training set -> set the parameters
Validation set -> try different models, select best
Test set -> how good is your chosen model

Underfitting



Overfitting



A “validation curve”

https://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html#plotting-validation-curves

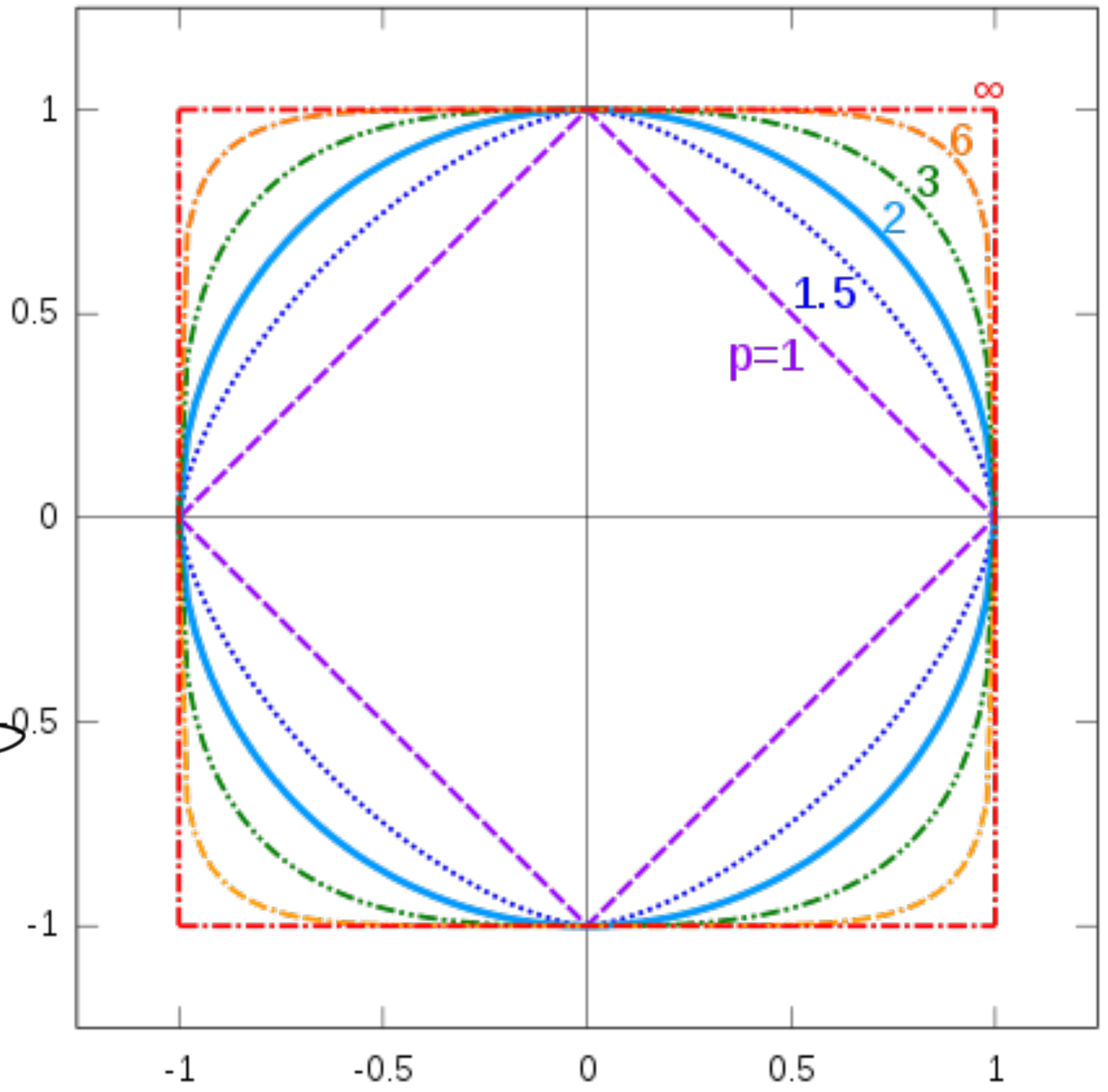
Week 3

Vector norms

Lp

$$\|x\|_p = \left(x_1^p + x_2^p + \dots x_n^p \right)^{1/p}$$

for the $x \in \mathbb{R}^2$ where x_1, x_2 are in range $[0, 1]$



Regularizations reduce overfitting to random noise

Penalize solutions that are too complex

One technique that is often used to control the over-fitting phenomenon in such cases is that of **regularization**, which involves adding a penalty term to the error function below in order to discourage the coefficients from reaching large values. By preventing the sum of our weights from growing large, we are preventing complex fitting... the total amount of weight allowed will be preferentially allocated to the most important features, preventing features that have less effect on the answer from getting too much love from the algorithm.

The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified loss/error function of the form

$$L(\mathbf{w}) = (X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\|\mathbf{w}\|_2$$

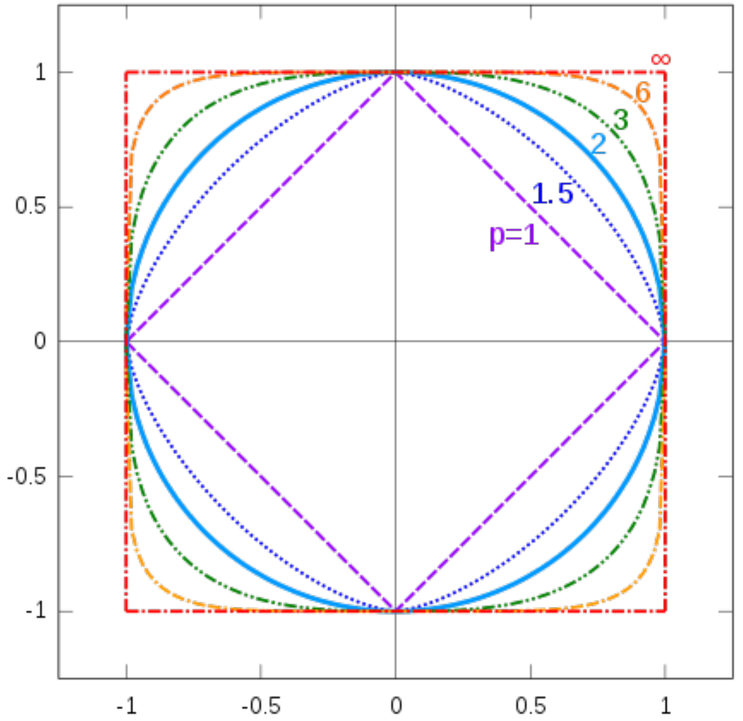
where the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term and $\mathbf{X}$ is the (nxm) design matrix and $\mathbf{w}$ is an m long column vector and $\mathbf{y}$ is an n long column vector.

There is a closed-form solution below:

$$\mathbf{w}^* = (X^T X + \lambda I)^{-1} X^T y$$

Regularizations reduce overfitting to random noise

Penalize solutions that are too complex



You can also do L1 regularization which is often called LASSO regression (least absolute shrinkage and selection operator)

$$L(\mathbf{w}) = (X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\|\mathbf{w}\|_1$$

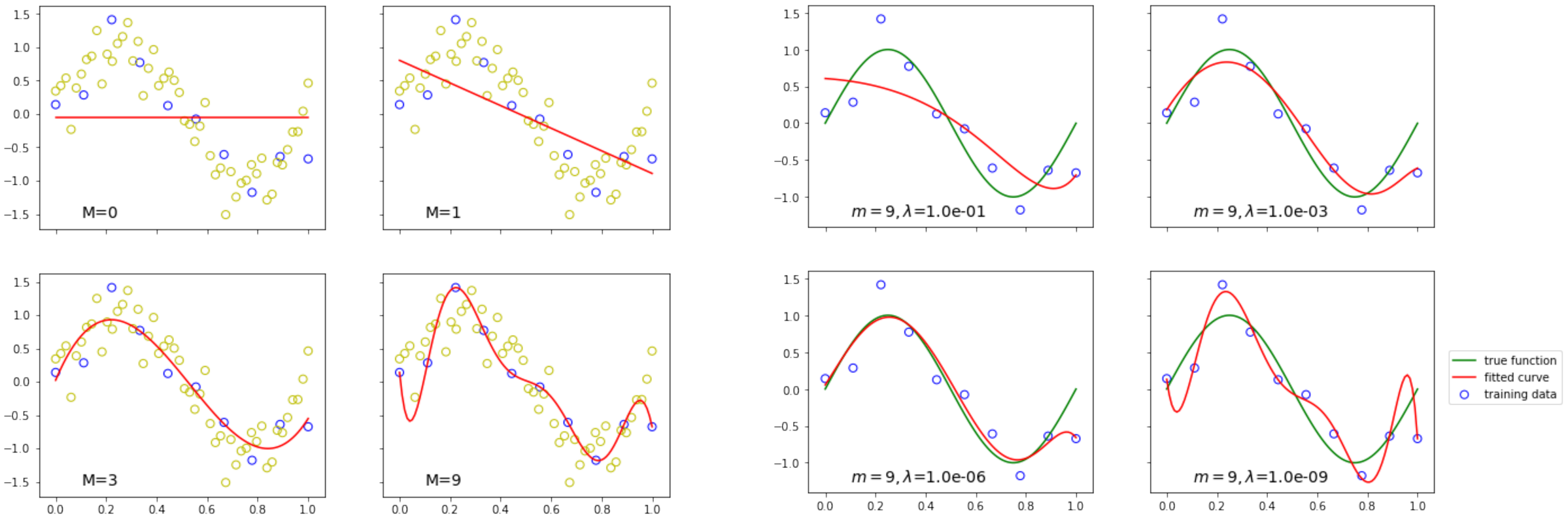
And you can combine the two together in a technique called ElasticNet

$$L(\mathbf{w}) = (X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}(\alpha\|\mathbf{w}\|_1 + (1-\alpha)\|\mathbf{w}\|_2)$$

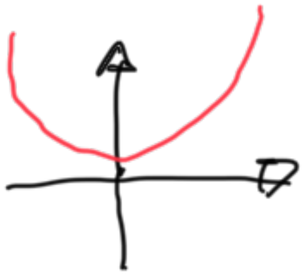
where $\alpha \in [0, 1]$ is a parameter dictating the proportion of L1 to L2 regularization.

Why all these different kinds of regularization? Well L1 tends to produce a *sparse* solution... many weights that are not important are driven towards 0. You use this technique when it seems appropriate to you that less-important factors have no influence on the solution. Whereas L2 limits the total amount of weight evenly, so less-important factors can continue to have less-influence-but-still-some-influence.

One application of L1 regularization: feature selection. Let's say you have data about the expression levels of ~27,000 protein coding genes across a few thousand humans, and you want to dermine if any of the genes have an effect on a disease. Since almost NONE of them will, it's good to use something like a heavy L1 penalty, which will prevent you from picking up too much on random chance associations that may exist in the data.


9th order polynomial regression + L2 regularization
M=0
M=1
M=3
M=9
$m=9, \lambda$=1.0e-01
$m=9, \lambda$=1.0e-03
$m=9, \lambda$=1.0e-06
$m=9, \lambda$=1.0e-09
true function
fitted curve
training data

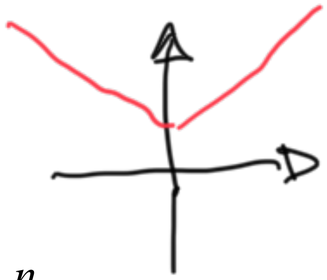
Robust regression with L1 error term



L2

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

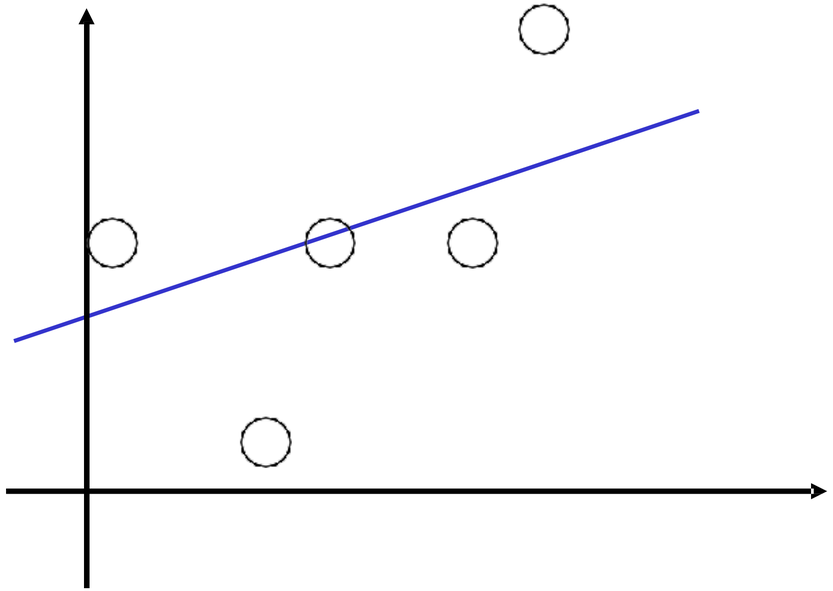
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w}; \mathbf{x}; \mathbf{y})$$

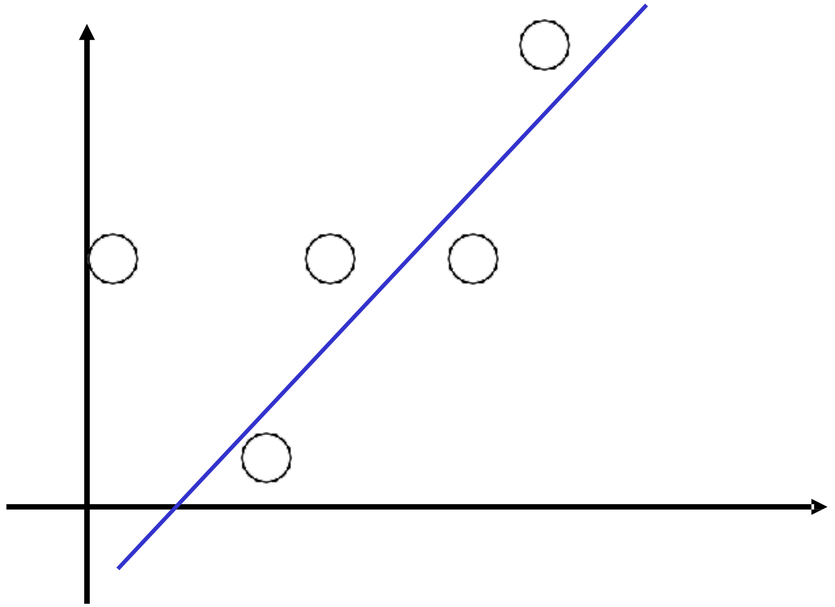


L1

$$\mathscr{L}_{\text{OLS}}(\mathbf{w}; \mathbf{x}; \mathbf{y}) = \big(\mathbf{y} - f(\mathbf{x}; \mathbf{w})\big)^{\mathrm{T}} \big(\mathbf{y} - f(\mathbf{x}; \mathbf{w})\big)$$
$$= \|\mathbf{y} - f(\mathbf{x}; \mathbf{w})\|_2^2$$

$$\mathscr{L}_{\text{robust}}(\mathbf{w}; \mathbf{x}; \mathbf{y}) = \sum_{i=1}^{n} \big|y_i - f\big(\mathbf{x}_i; \mathbf{w}\big)\big|$$
$$= \|\mathbf{y} - f(\mathbf{x}; \mathbf{w})\|_1$$





2 extreme cases lead to same testing error

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \qquad S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}$$

$$\epsilon_{\text{testing}} = \epsilon_{\text{training}} + \epsilon_{\text{generalization}}$$

Case 1:

1. $e_{testing} = 0.5 + 0.0$

- We make a completely random guess even after training (didn't learn anything and attained an error of 0.5).
- A random guess is however highly generalizable, which doesn't incur any generalization error.

Case 2:

2. $e_{testing} = 0.0 + 0.5$

- We make perfect classifications on the training data (memorizing the labels for every training sample).
- Merely memorizing the training samples with their corresponding classification labels is however highly non-generalizable, incurring the largest generalization error (no identical training sample will appear in testing).

Both are extreme cases that lead to trivial ML models.

Testing error

$$\epsilon_{\text{testing}} = \epsilon_{\text{training}} + \epsilon_{\text{generalization}}$$

Case 1:

1. $e_{testing} = 0.5 + 0.0$

Case 2:

2. $e_{testing} = 0.0 + 0.5$

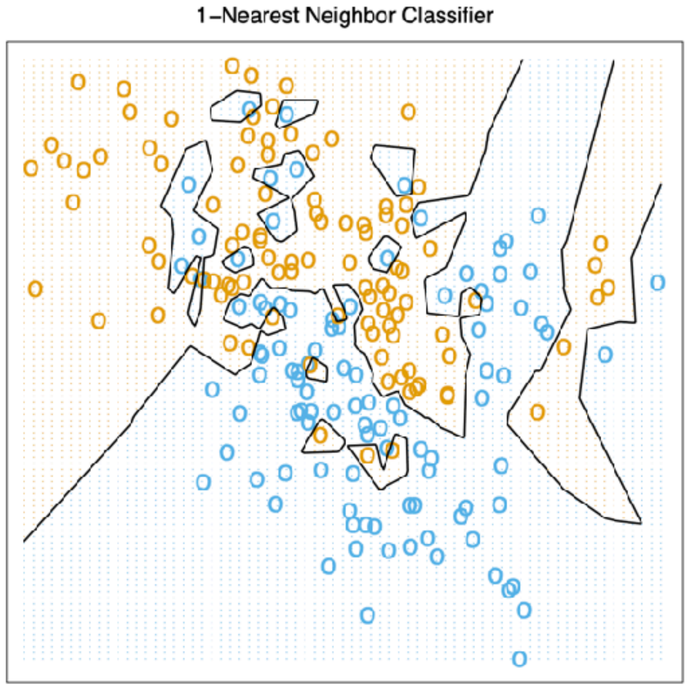
Both are extreme cases that lead to trivial models that we should avoid.

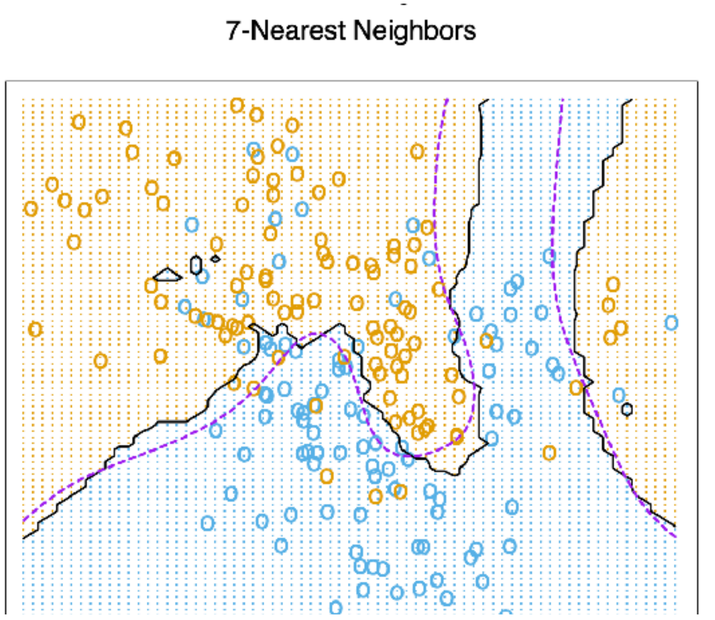
Ideally: $e_{testing} = 0.0 + 0.0$

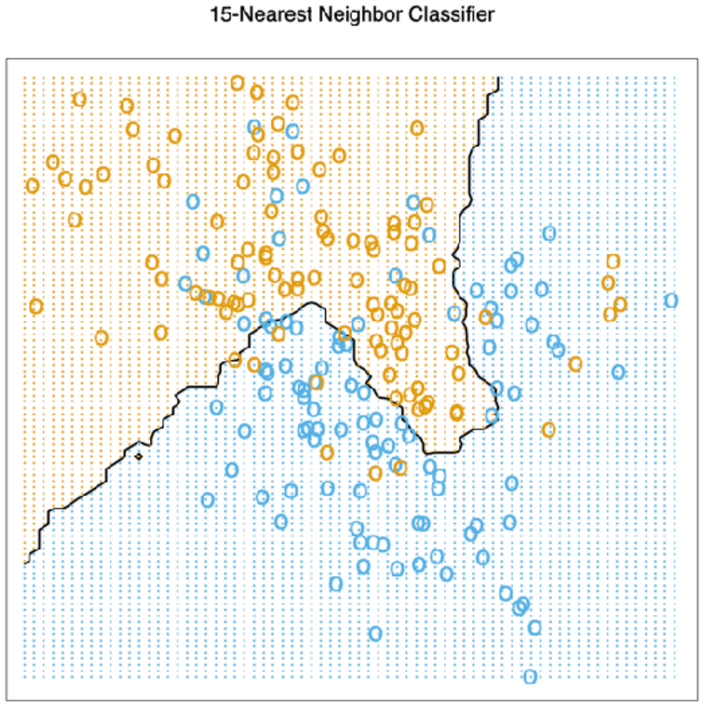
A classification model that is perfect after training and makes no error in testing.

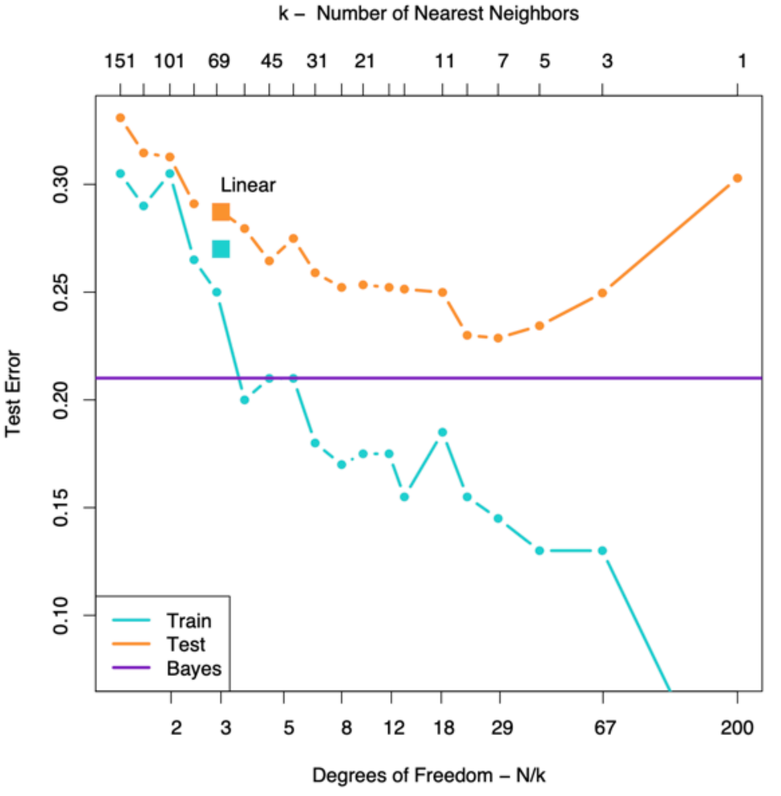
In practice: $e_{testing} = 0.05 + 0.1$

A classification model that does well after training and generalizes reasonably well in testing.









Figures from Hastie, Tibshirani, & Freidman; ESL

Nearest Neighbor

When to Consider

- Instance map to points in R^n
- Less than 20 attributes per instance
- Lots of training data

Advantages

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages

- Slow at query time
- Easily fooled by irrelevant attributes

Training
Time O(1)
Memory O(n)

Testing
Time O(n)
Memory O(n)

Issues

- Distance measure
  - Most common: Euclidean
- Choosing k
  - Increasing k reduces variance, increases bias
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.

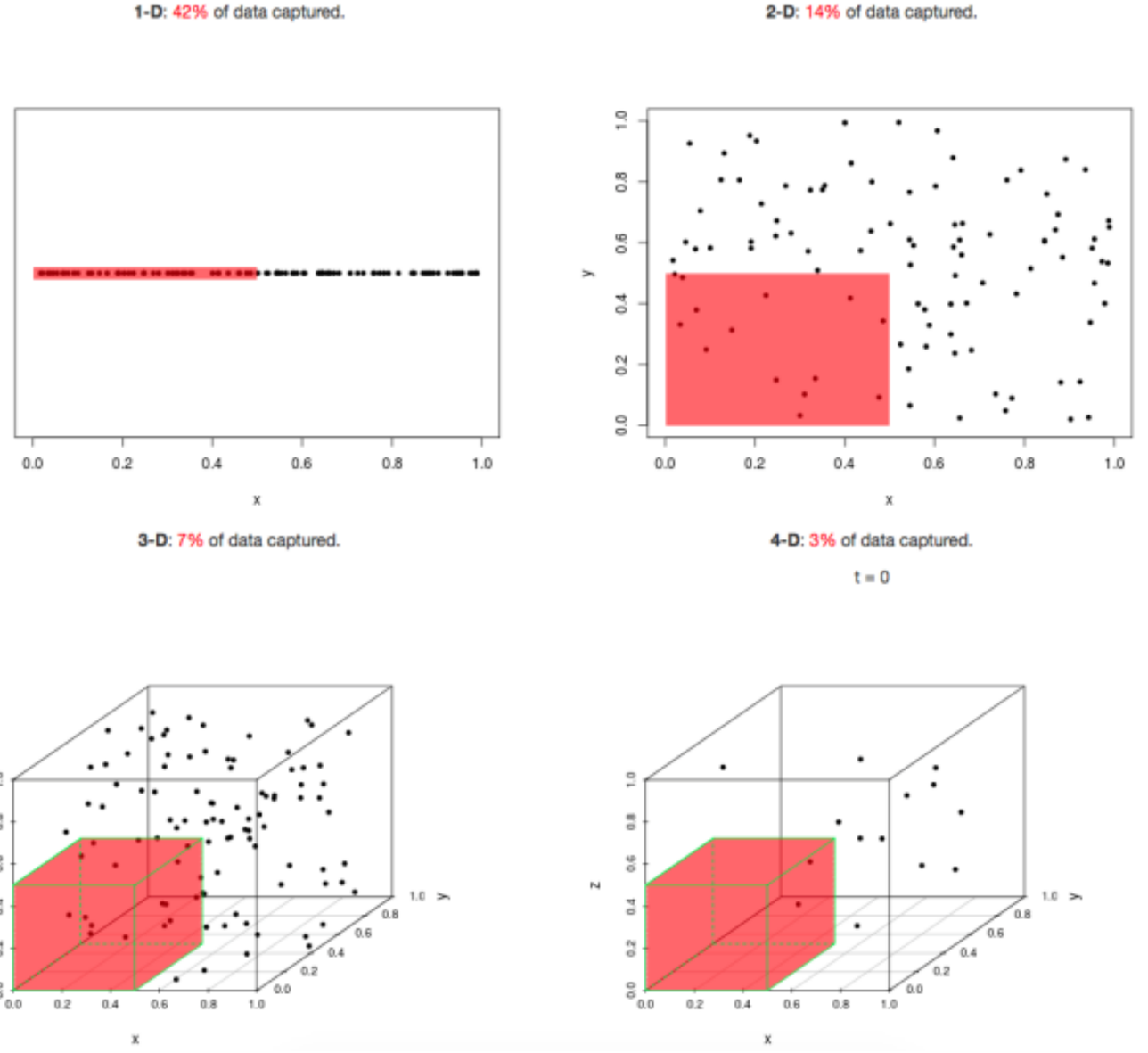
NEED TO HAVE VARIABLES STANDARDIZED!!!! (ITS A METRIC SPACE)

NEED TO HAVE ORDERS OF MAGNITUDE MORE DATAPOINTS THAN VARIABLES!!!! (CURSE OF DIMENSIONALITY)

Slides from David Sonntag, NYU

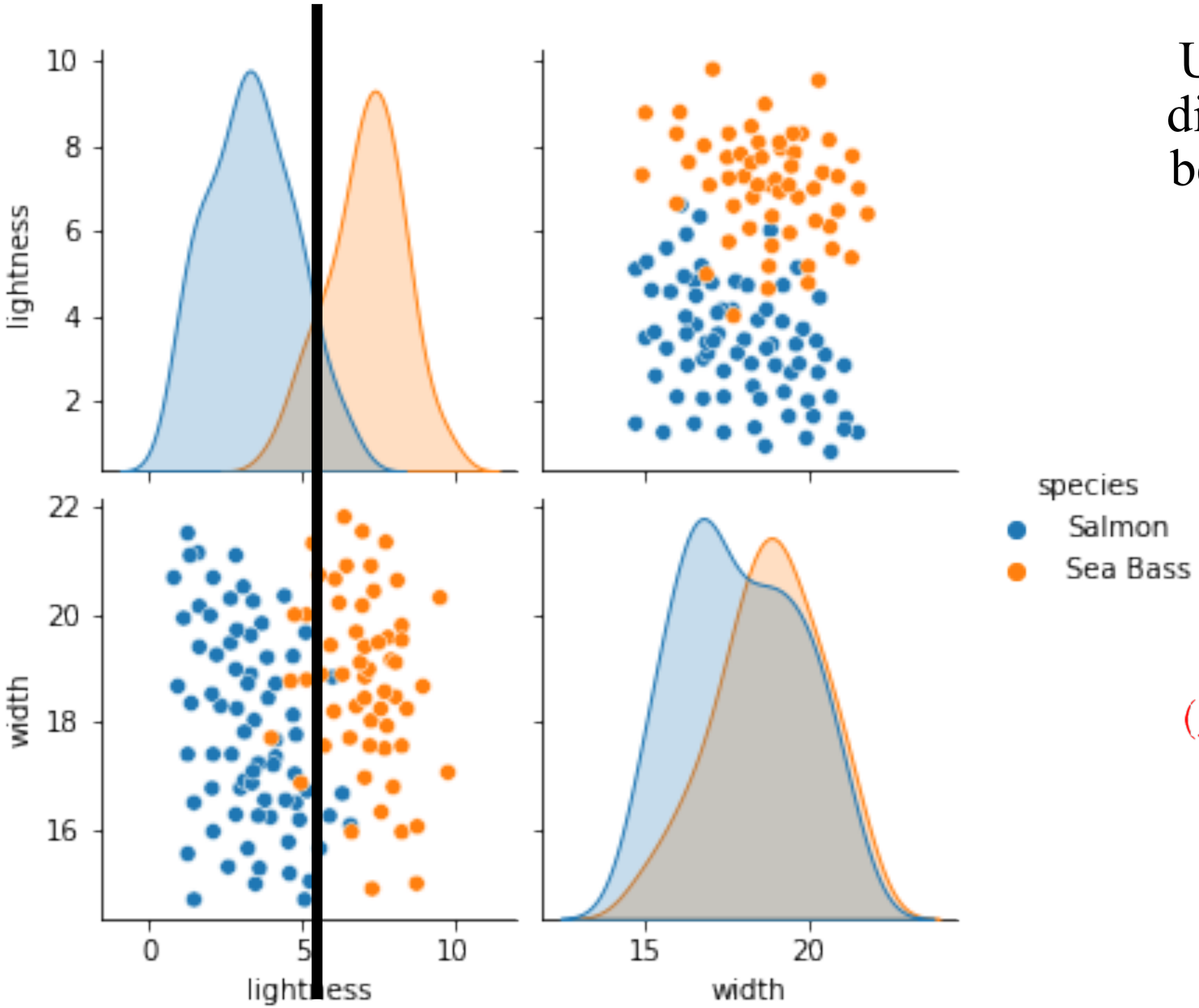
The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces because the “neighborhood” becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm.
- Suppose our query point is at the origin.
  - 1D –
    - On a one dimensional line, we must go a distance of 5/5000 = 0.001 on average to capture the 5 nearest neighbors
  - 2D –
    - In two dimensions, we must go sqrt(0.001) to get a square that contains 0.001 of the volume
  - D –
    - In D dimensions, we must go $(0.001)^{1/D}$



Slides from David Sonntag, NYU

Seaborn: sns.pairplot()



Decision stump

Use exhaustive search to find which dimension and which threshold value best separates the positive class from the negative class

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th)$$

$$= \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

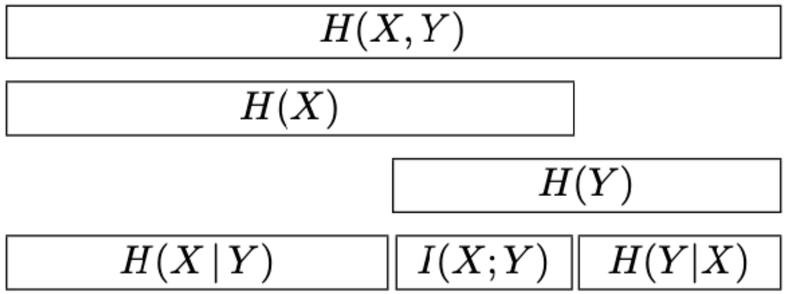


Figure 8.1. The relationship between joint information, marginal entropy, conditional entropy and mutual entropy.

From David MacKay

Entropy of a single random variable: how uncertain are we about what the value x will be before we measure it?

$$H(X) = -\sum_i P(X = x_i) \log P(X = x_i)$$

Joint entropy:

$$H(X,Y) = -\sum_{i,j} P(X = x_i, Y = y_j) \log P(X = x_i, Y = y_j)$$

Conditional entropy: how much info you've gained about Y if you know X

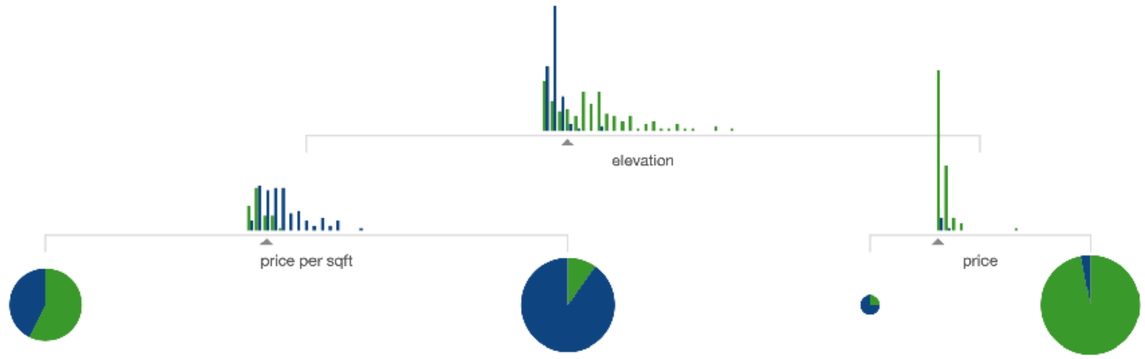
$$H(Y|X) = \sum_i P(X = x_i) H(Y|X = x_i)$$
$$= -\sum_i P(X = x_i)[\sum_j P(Y = y_j|X = x_i) \log P(Y = y_j|X = x_i)]$$

Mutual information: the distance between the joint and the product

$$I(X;Y) = \sum_i \sum_j P(X = x_i, Y = y_j) \log \frac{P(X=x_i, Y=y_j)}{P(X=x_i)P(Y=y_j)}$$

Generic Decision Tree

A recursive algorithm

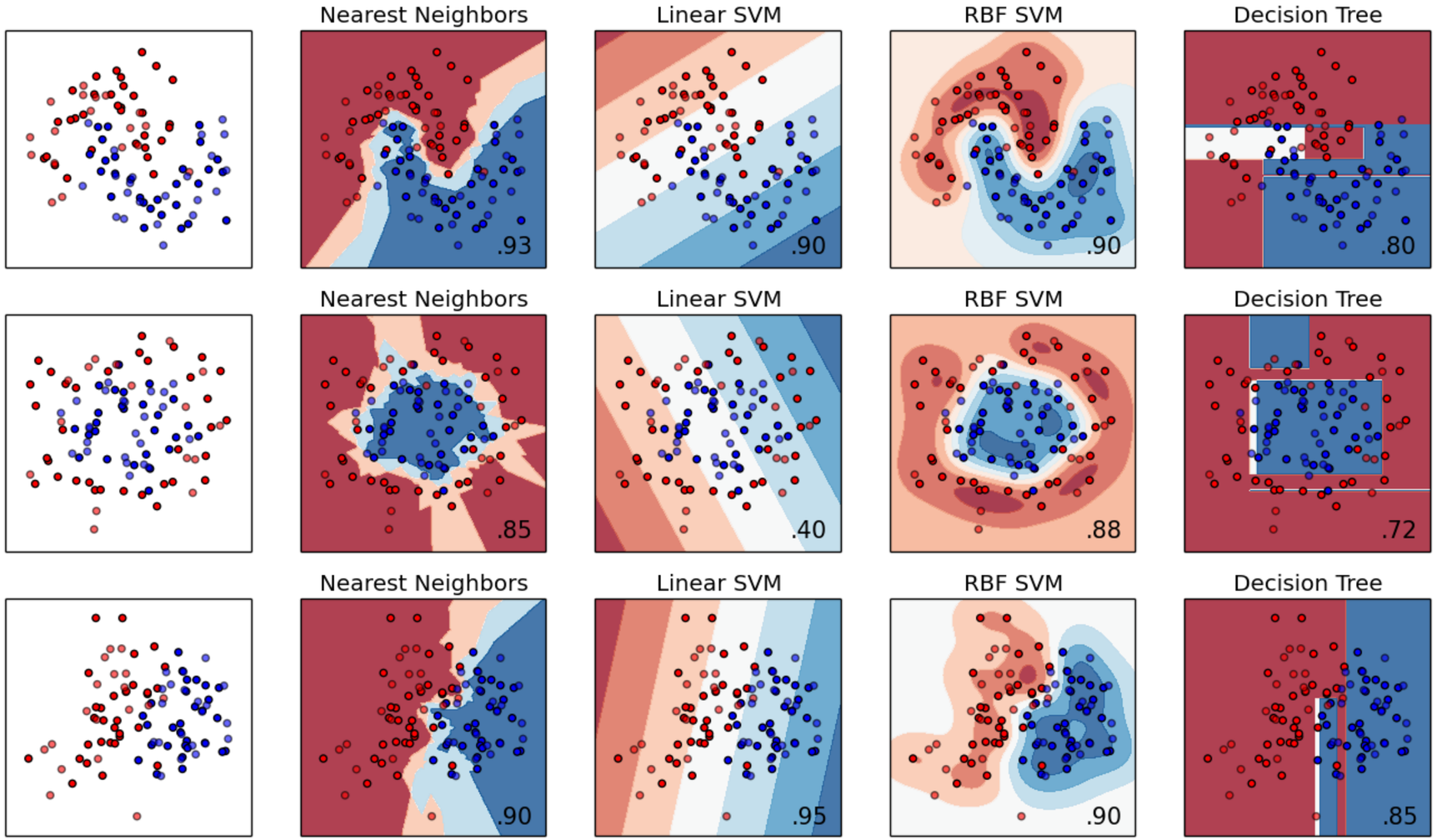



```
def build_tree(S):
    # inputs: dataset S
    # outputs: a (sub)tree
    if stop_criteria(S): # typically S is pure OR len(S) < some value
        return leaf_node(S)
    else:
        # typically optimal_test() picks a
        # test `t_star` (e.g., a feature and a decision threshold)
        # that creates `partitions` of S such that
        # those partitions maximize an information metric
        # i.e., each partition is mostly just a single class
        t_star, partitions = optimal_test(S)

        children=[]
        for child_S in partitions:
            children.append( build_tree(child_S) )
        return (t_star + children)
```

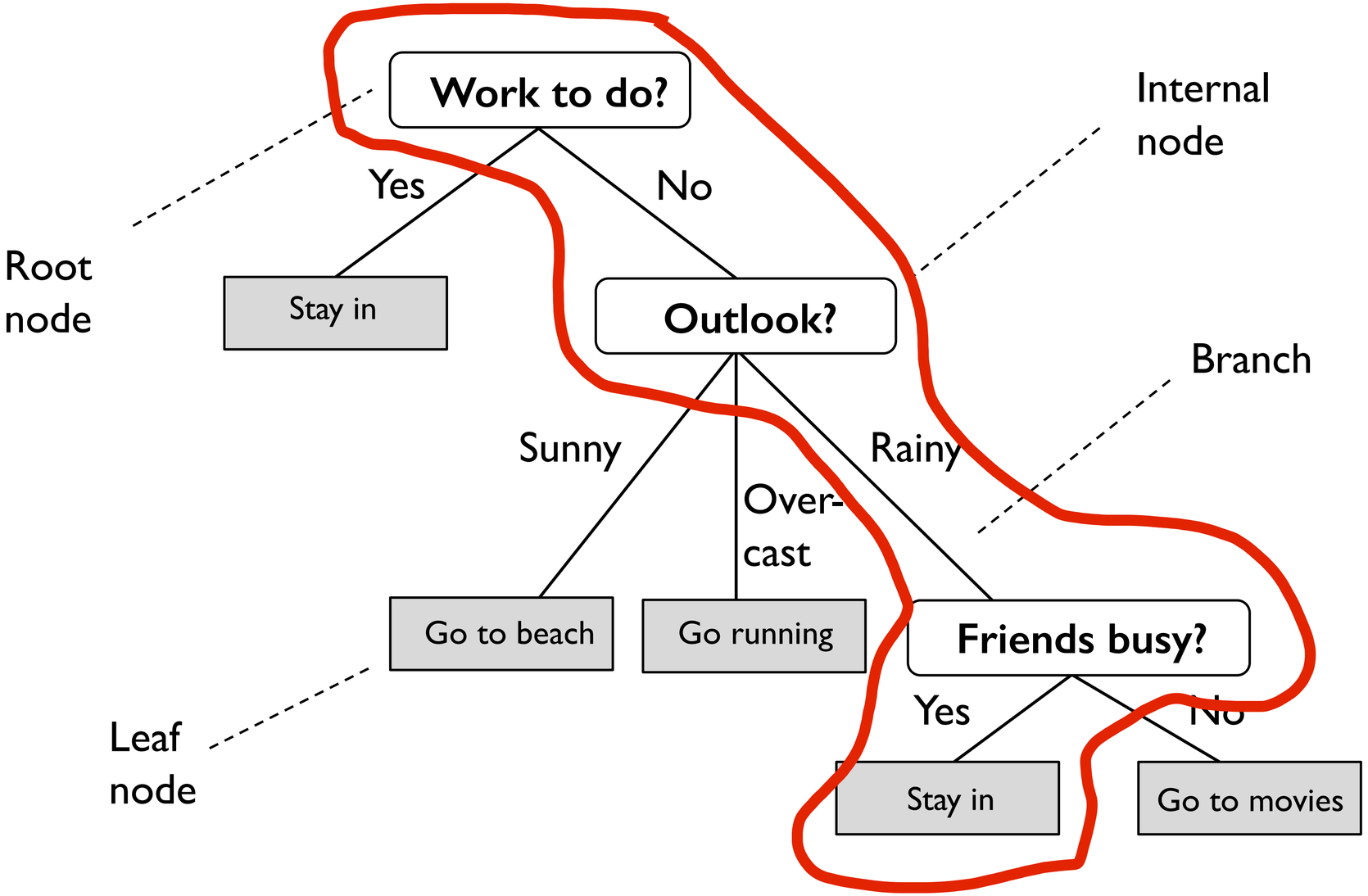
Decision trees can learn any boundary

… as long as its piecewise linear



Each leaf of a tree corresponds to an if-then rule

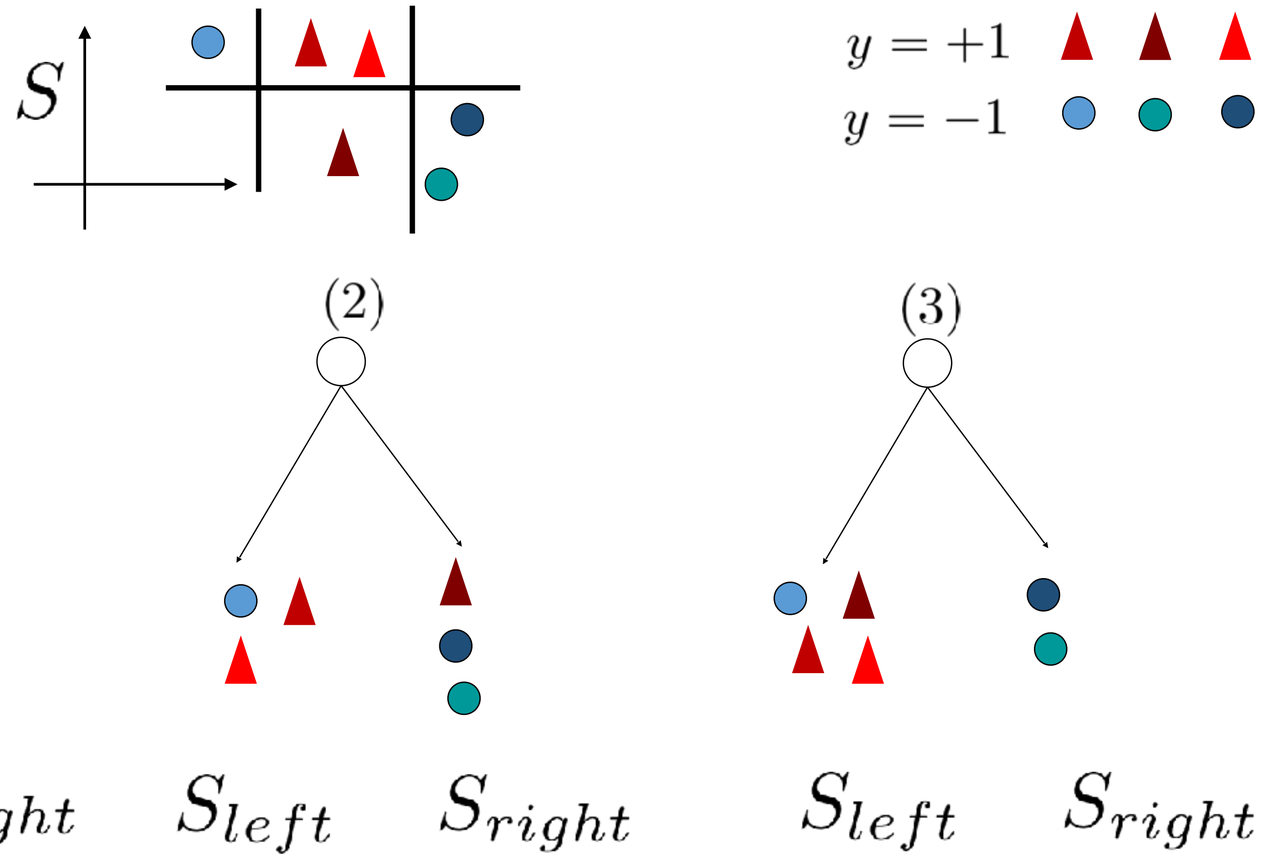
(Work to do? = False)∩(Outlook = Rainy?)∩(Friends busy? = Yes)∪(Work to do? = True)



Tree construction (J. Quinlan)

Recursively construct a tree, selecting a feature and a threshold value that maximizes the gain at each recursion.

So which split should we use first?



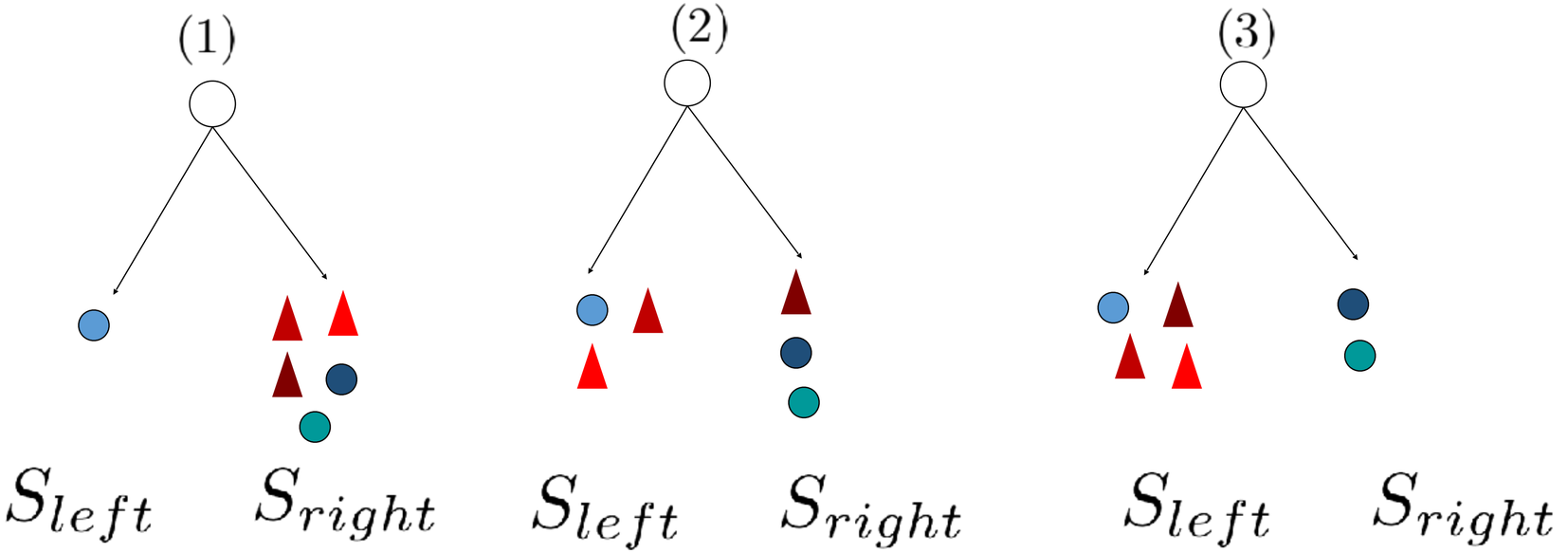
ID3 tree construction

$$\arg\max_{B} G(S,B) = H(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} H(S_i)$$

(1) $G(S_{left}) = -\frac{1}{6}(1\log_2 1 + 0\log_2 0) = 0, \quad G(S_{right}) = -\frac{5}{6}(0.4\log_2 0.4 + 0.6\log_2 0.6) = 0.81, \quad \therefore G(S,B) = H(S) - 0.81$

(2) $G(S_{left}) = -\frac{3}{6}(0.33\log_2 0.33 + 0.67\log_2 0.67) = 0.46, \quad G(S_{right}) = -\frac{3}{6}(0.67\log_2 0.67 + 0.33\log_2 0.33) = 0.46, \quad \therefore G(S,B) = H(S) - 0.92$

(3) $G(S_{left}) = -\frac{4}{6}(0.25\log_2 0.25 + 0.75\log_2 0.75) = 0.54, \quad G(S_{right}) = -\frac{2}{6}(0\log_2 0 + 1\log_2 1) = 0, \quad \therefore G(S,B) = H(S) - 0.54$



C4.5

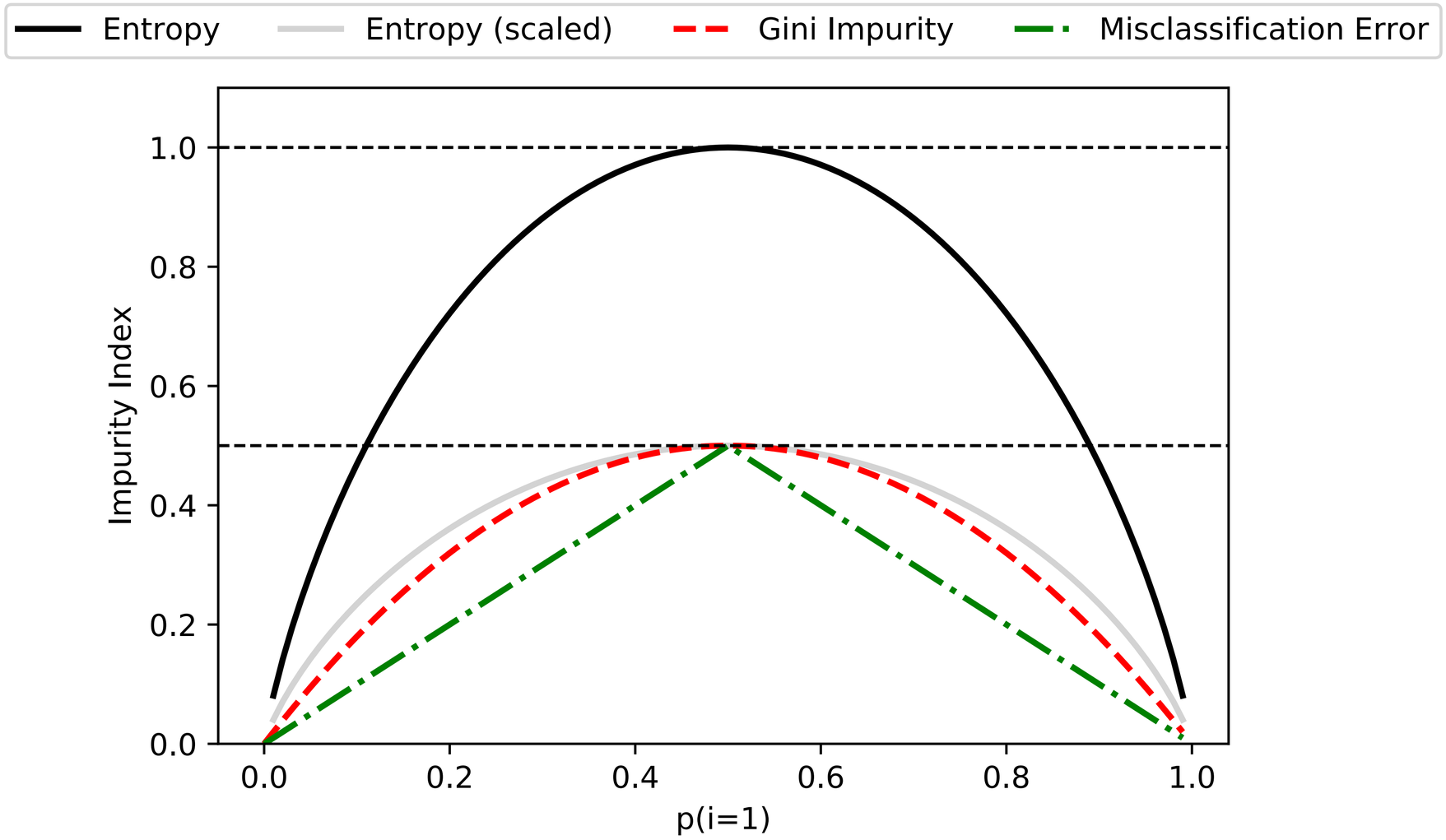
- continuous and discrete features
- Ross Quinlan 1993, Quinlan, J. R. (1993). C4. 5: Programming for machine learning. *Morgan Kauffmann*, *38*, 48.
- continuous is very expensive, because must consider all possible ranges
- handles missing attributes (ignores them in gain compute)
- post-pruning (bottom-up pruning)
- Gain Ratio $G(S,B)/P(S,B)$

$$G(S,B) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I(S_i).$$

$$P(S,B) = -\sum_{i=1}^{t} \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right).$$

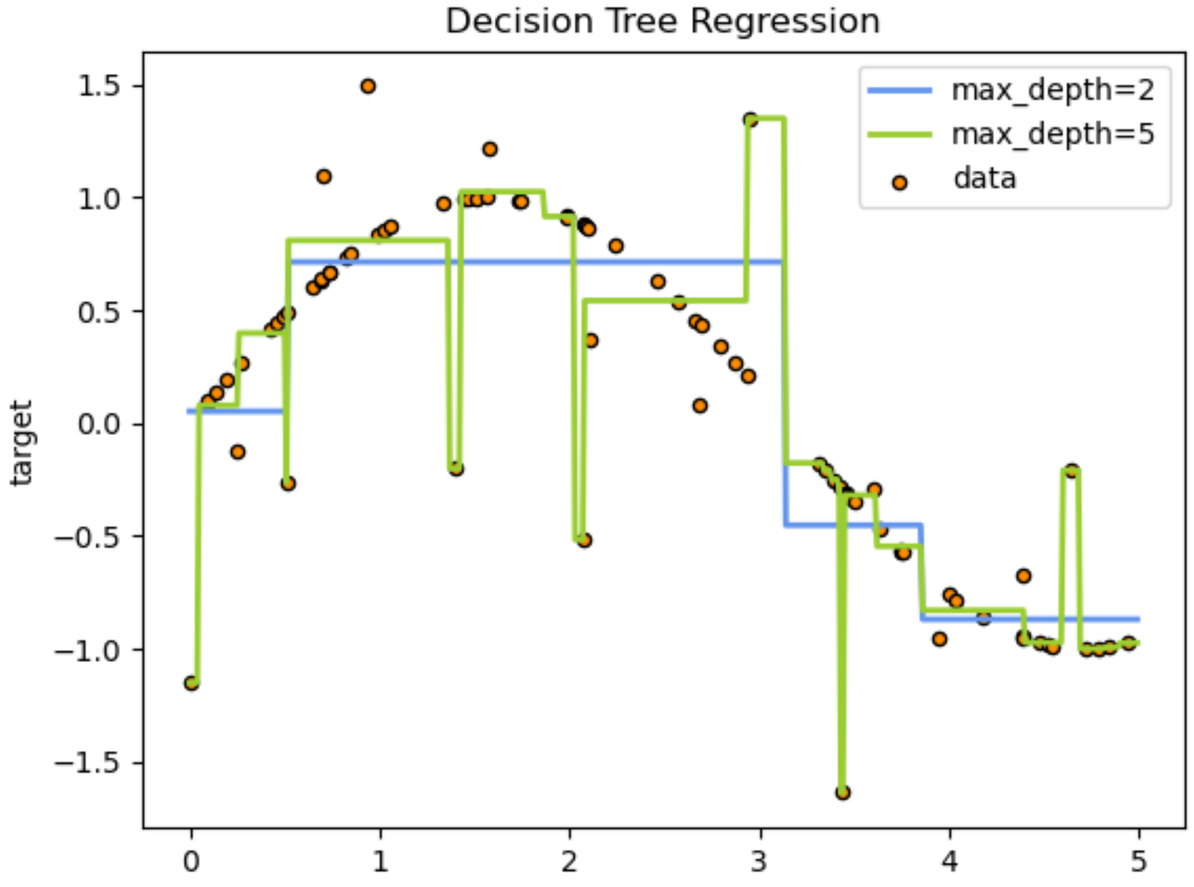
CART

- Breiman, L. (1984). *Classification and regression trees*. Belmont, Calif: Wadsworth International Group.
- continuous and discrete features
- strictly binary splits (taller trees than ID3, C4.5)
- binary splits can generate better trees than C4.5, but tend to be larger and harder to interpret; k-attributes has a ways to create a binary partitioning
- variance reduction in regression trees
- Gini impurity, $G = \sum_{i=1}^{C} p(i) * (1 - p(i))$
- cost complexity pruning


Entropy
Entropy (scaled)
Gini Impurity
Misclassification Error
Impurity Index
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
p(i=1)

CART Regression

Use MSE as the impurity measurement



Decision trees

Advantages

- Interpretable
- Non-parametric method
- Able to fit arbitrary decision boundaries (not just linear!)
- Don’t need to scale features to match each other
- Can be combined with techniques to make it better (like bagging and boosting)

Disadvantages

- Easy to overfit
- Needs some kind of pruning and tree-growth limits to avoid overfitting
- For regression trees, the output is bounded by the limits of the training samples

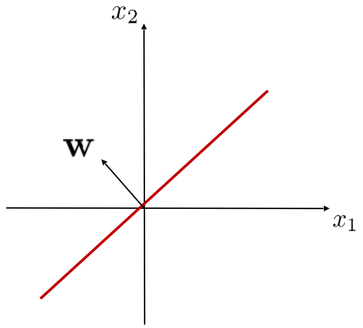
Week 4

Decision boundary

When b=0:

$\{\mathbf{x}; \forall \mathbf{x}\ such\ that\ \mathbf{w}^T\mathbf{x} = 0\}$

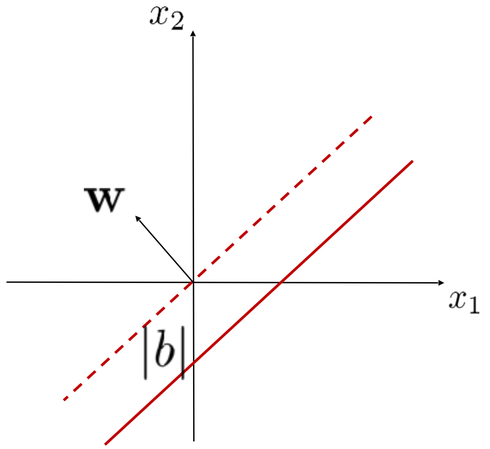
The decision boundary always goes through the origin.



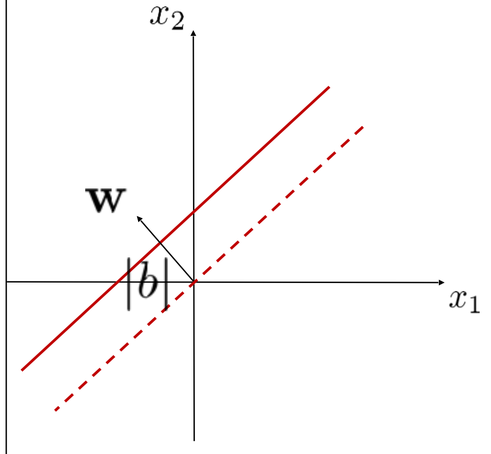
Decision boudary:

$\{\mathbf{x}; \forall \mathbf{x}\ such\ that\ \mathbf{w}^T\mathbf{x} + b = 0\}$

When b>0 the decision boundary is moved along the opposite direction of w.



When b<0 the decision boundary is moved along the same direction of w.



Decision boundary

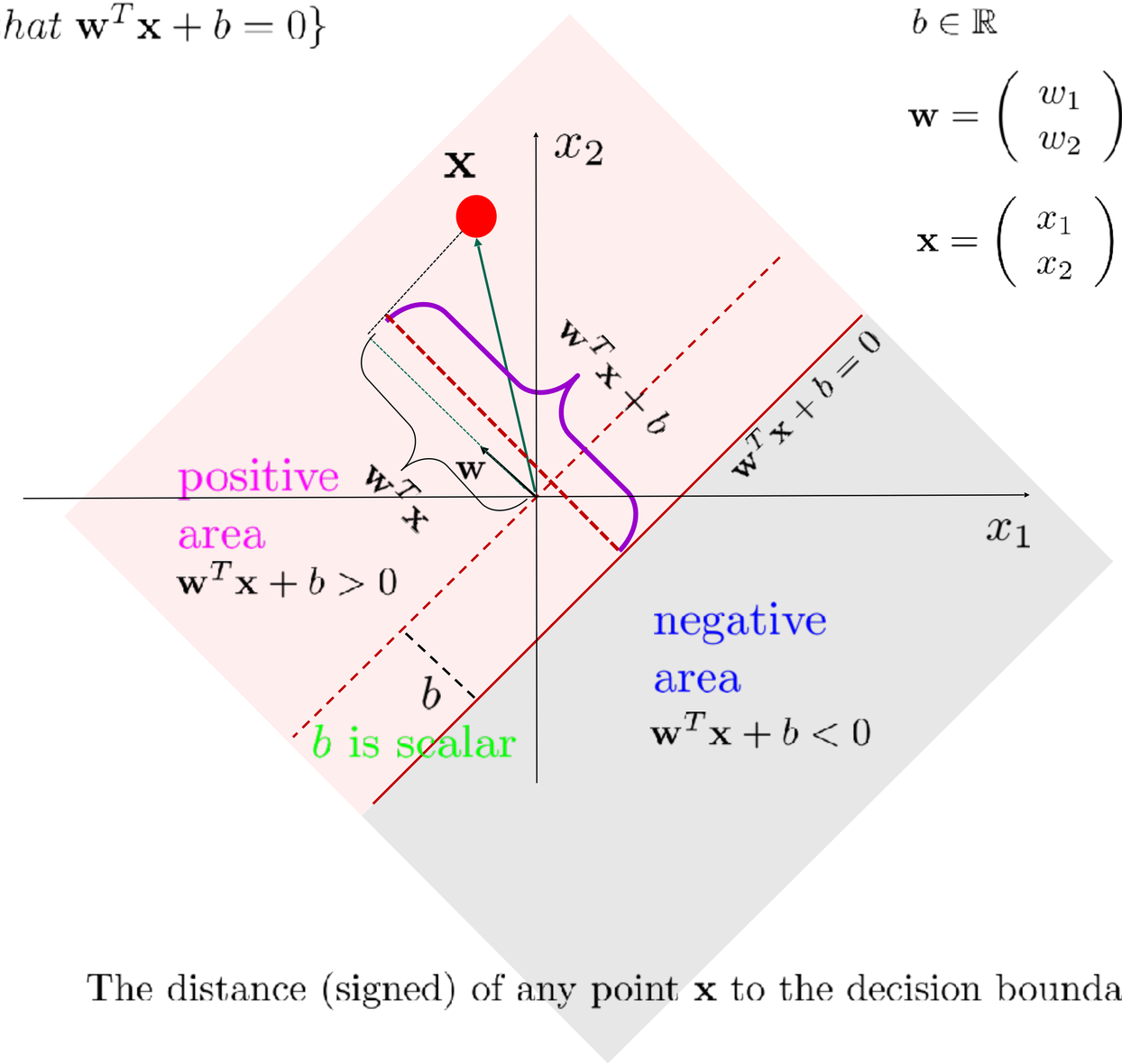
Decision boudary:

$\{\mathbf{x}; \forall \mathbf{x}\ such\ that\ \mathbf{w}^T\mathbf{x} + b = 0\}$

$b \in \mathbb{R}$

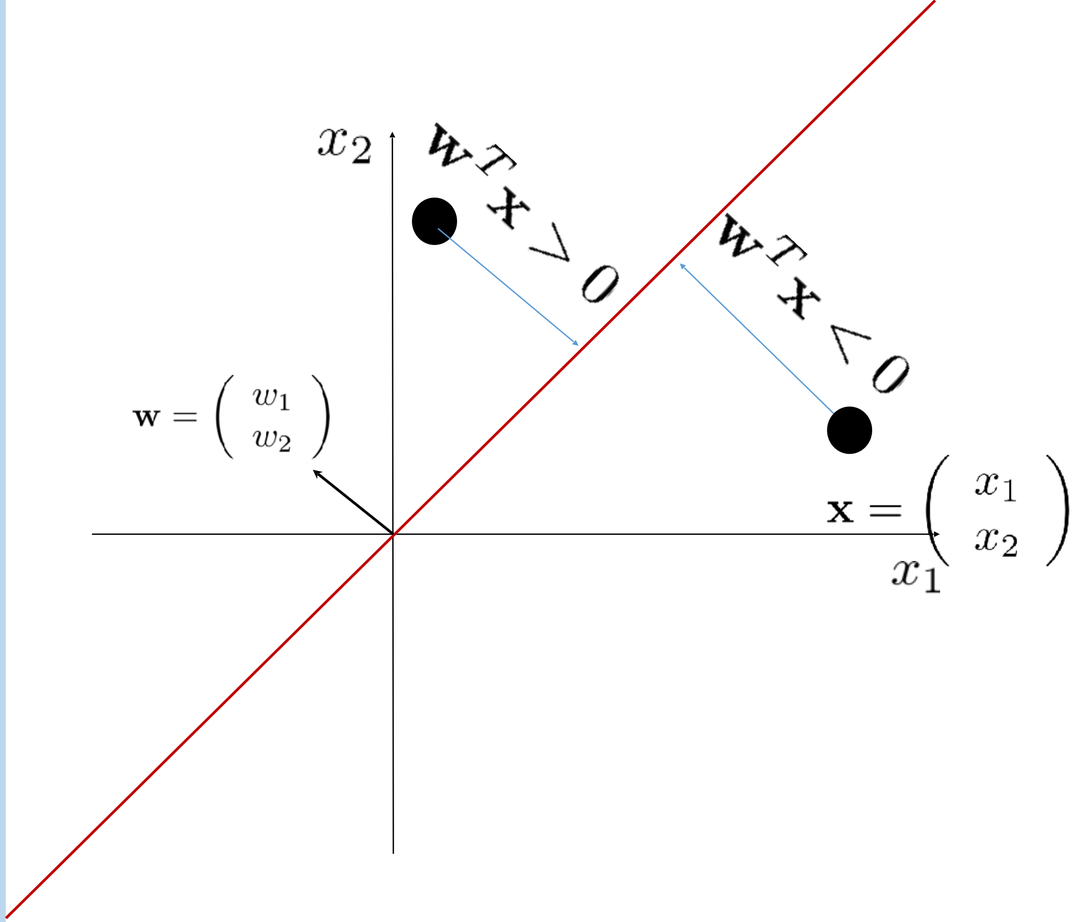
$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$



The distance (signed) of any point $\mathbf{x}$ to the decision boundary is: $\mathbf{w}^T\mathbf{x} + b$

Distance to the decision boundary



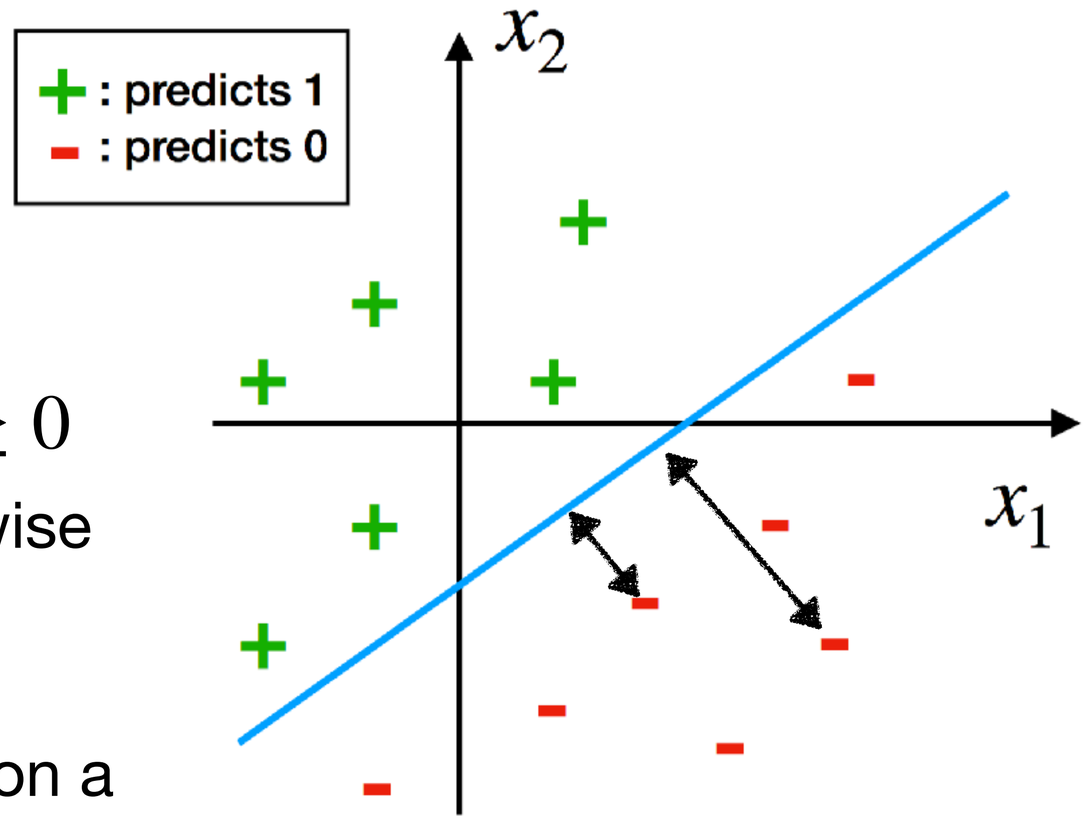
The distance (signed) of any point $\mathbf{x}$ to the line is:

$$\mathbf{w}^T\mathbf{x} \equiv < \mathbf{w}, \mathbf{x} >$$

$\mathbf{w}^T\mathbf{x} > 0$: above the line

$\mathbf{w}^T\mathbf{x} < 0$: below the line

Distance as Probability

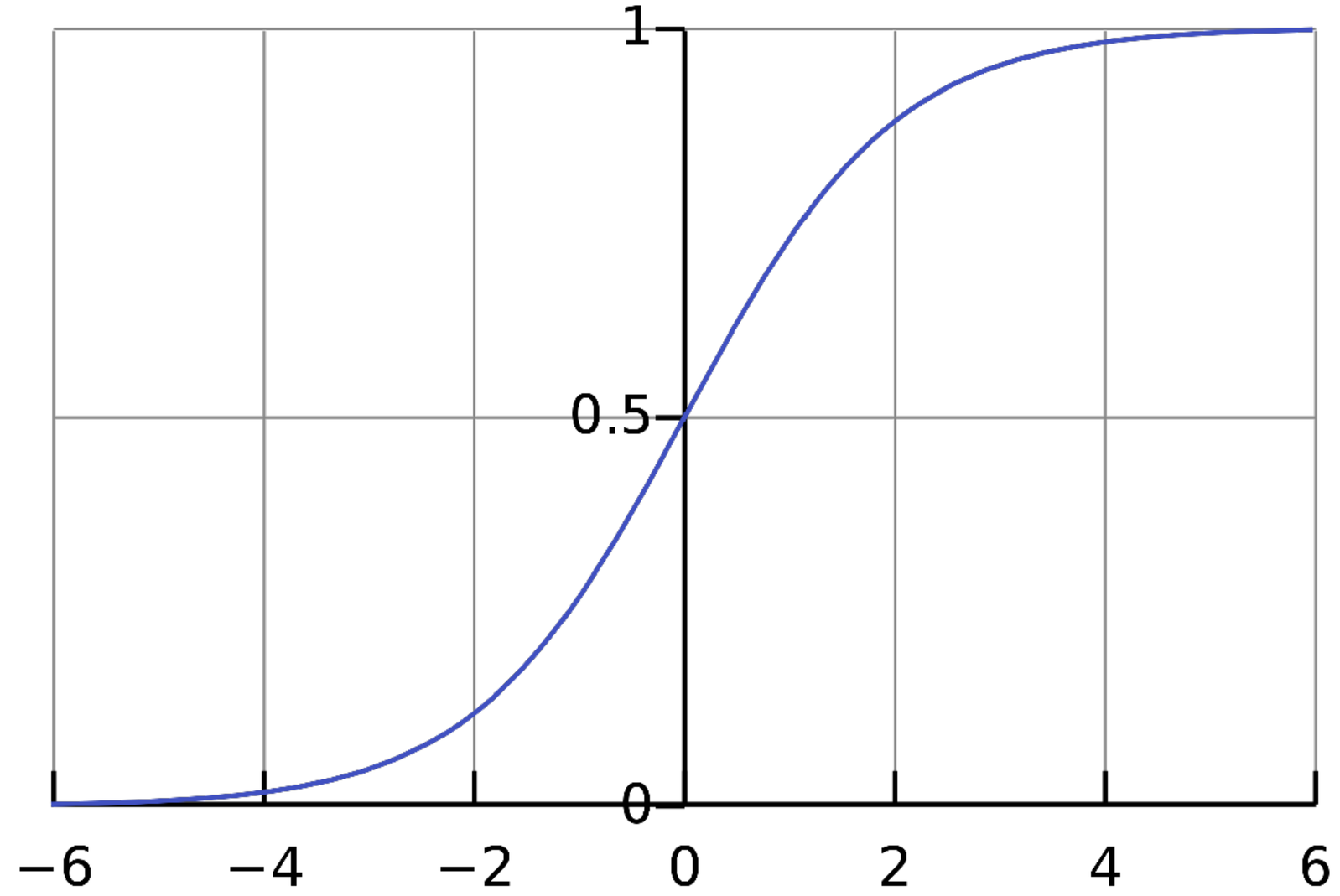


- Classification $f(x; w) = \begin{cases} 1 \text{ if } w^T x \geq 0 \\ -1 \text{ otherwise} \end{cases}$
- Our predictions are only +1 or -1
- What if we want to make our prediction a probability? Turn distance above/below line into a number between 0 and 1
- $f(x; w) = p(y = +1 \,|\, x; w)$
  or equivalently
  $f(x; w) = -p(y = -1 \,|\, x; w)$

Logistic function a.k.a. logit

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

Always in range 0 - 1 => probability

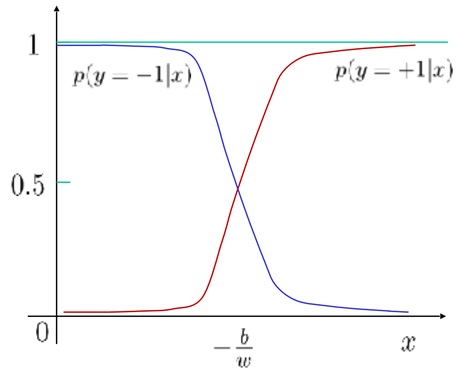


Training a logistic regression classifier

$$S_{training} = \{(-1.1, -1), (3.2, +1), (2.5, -1), (5.0, +1), (4.3, +1)\}$$

$$p(y = +1|\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

$$p(y = -1|\mathbf{x}) = \frac{1}{1+e^{(\mathbf{w}^T\mathbf{x}+b)}}$$



$$p(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

Train a logistic regression classifier $f(\mathbf{x}) = \begin{cases} +1 & if \ \frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}} \geq 0.5 \\ -1 & otherwise \end{cases}$:

Intuition: find the best parameters $(\mathbf{w}, b)^*$ to maximize the probabilities of fitting the ground-truth label y_i for each $\mathbf{x}_i$.

Math: $(\mathbf{w}, b)^* = \arg\max_{(\mathbf{w},b)} \prod_{i=1}^{n} \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$

Training a logistic regression classifier

Intuition: find the best parameters $(\mathbf{w}, b)^*$ to maximize the probabilities of fitting the ground-truth label y_i for each $\mathbf{x}_i$.

Math: $(\mathbf{w}, b)^* = \arg\max_{(\mathbf{w},b)} \prod_{i=1}^{n} \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$

$$(\mathbf{w}, b)^* = \arg\max_{(\mathbf{w},b)} \prod_{i=1}^{n} \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

$$= \arg\max_{(\mathbf{w},b)} \ln\left(\prod_{i=1}^{n} \frac{1}{1+e^{-y_i(\mathbf{w}^T x_i+b)}}\right)$$

$$= \arg\min_{(\mathbf{w},b)} \sum_{i=1}^{n} -\ln\left(\frac{1}{1+e^{-y_i\times(\mathbf{w}^T\mathbf{x}_i+b)}}\right)$$

Training a MULTIVARIATE logistic regression classifier

$$\mathbf{x}_i \in \mathbb{R}^m, i = 1..n \qquad y_i \in \{-1, +1\}, i = 1..n$$

Model parameters: $\mathbf{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$

$$p(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

Intuition: find the best parameters $(\mathbf{w}, b)^*$ to maximize the probabilities of fitting the ground-truth label y_i for each x_i.

Math:

$$(\mathbf{w}, b)^* = \arg\max_{(\mathbf{w},b)} \prod_{i=1}^{n} \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

$$(\mathbf{w}, b)^* = \arg\min_{(\mathbf{w},b)} \sum_{i=1}^{n} -\ln\left(\frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}\right)$$

$$= \arg\min_{\mathbf{w},b} \mathcal{L}(\mathbf{w}, b)$$

Derivative for the logistic regression classifier

$$\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} \ln(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i + b)}) \qquad p(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

$$\frac{\partial \mathcal{L}(\mathbf{w},b)}{\partial \mathbf{w}} = \sum_{i=1}^{n} \frac{\partial \ln(1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)})}{\partial \mathbf{w}}$$

$$= \sum_{i=1}^{n} \frac{\frac{\partial(1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)})}{\partial \mathbf{w}}}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

Ln' = 1/x; chain rule

$$= \sum_{i=1}^{n} \frac{e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}(-y_i\mathbf{x}_i)}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

Exp' = exp; chain rule

$$= \sum_{i=1}^{n} \frac{(1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}-1)(-y_i\mathbf{x}_i)}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}}$$

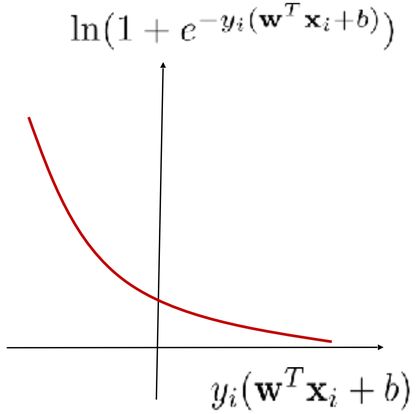
+1 - 1 to allow factoring below

$$= \sum_{i=1}^{n} (1 - \frac{1}{1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)}})(-y_i\mathbf{x}_i)$$

$$= \sum_i -y_i\mathbf{x}_i(1 - p(y_i|\mathbf{x}_i))$$

Multivariate input

$$\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} \ln(1 + e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)})$$



$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b) = \sum_i \frac{-y_i \mathbf{x}_i e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{1 + e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)}} = \sum_i -y_i \mathbf{x}_i (1 - p(y_i | \mathbf{x}_i))$$

$$\nabla_b \mathcal{L}(\mathbf{w}, b) = \sum_i \frac{-y_i e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{1 + e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)}} = \sum_i -y_i (1 - p(y_i | \mathbf{x}_i))$$

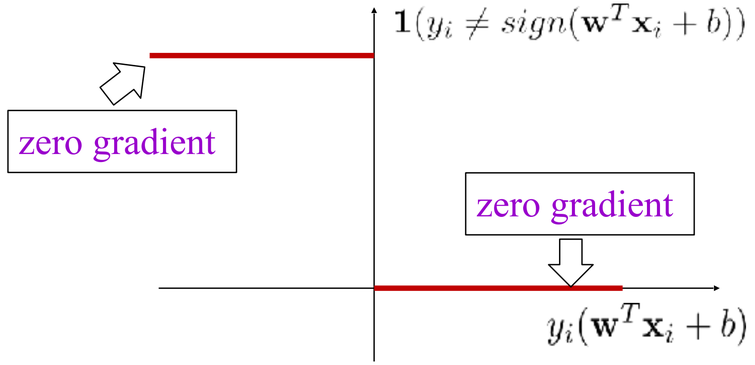
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t, b_t)$$

$$b_{t+1} = b_t - \lambda_t \times \nabla_b \mathcal{L}(\mathbf{w}_t, b_t)$$

Hard loss (error) function

Standard 0/1 loss (gradient 0 nearly everywhere, no gradient feedback):

Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_i \mathbf{1}(y_i \neq sign(\mathbf{w}^T\mathbf{x}_i + b))$



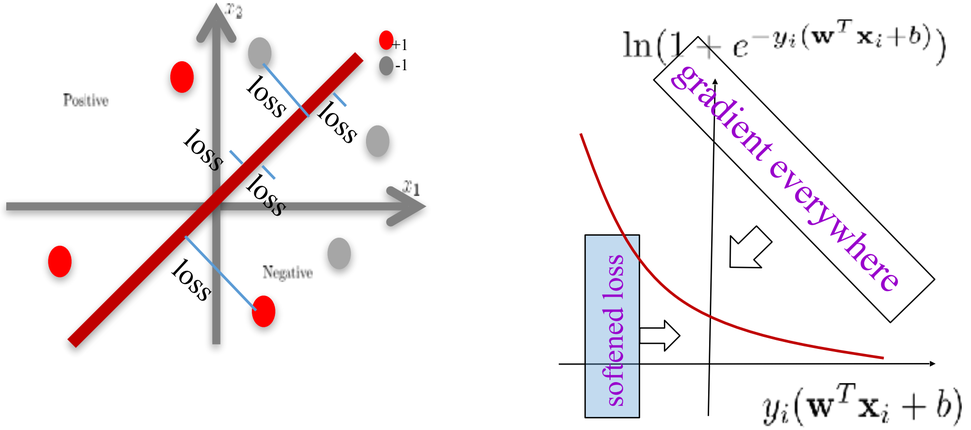
It is the most direct loss, but is also the hardest to minimize.

Zero gradient everywhere!

Soft loss (error) function

Loss used in logistic regression.

Training: minimize $\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} \ln(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i + b)})$



Every data point receives a loss (gradient everywhere).

A loss based on the distance to the decision boundary for wrong classification (has a gradient).

Used in logistic regression classifier.

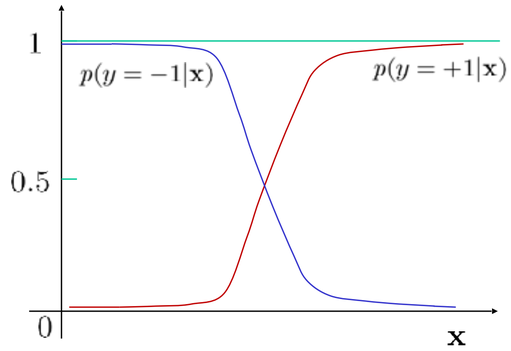
Logistic regression classifier

$$p(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(\mathbf{w}\cdot\mathbf{x}_i+b)}}$$

$$\mathbf{x} \in \mathbb{R}^m$$

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

$$f(\mathbf{x}) = \begin{cases} +1 & if \ \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}} \geq 0.5 \\ -1 & otherwise \end{cases}$$



Pros:

1. It is well-normalized.
2. Easy to turn into probability.
3. Easy to implement.

Cons:

1. Indirect loss function.
2. Dependent on good feature set.
3. Weak on feature selection.

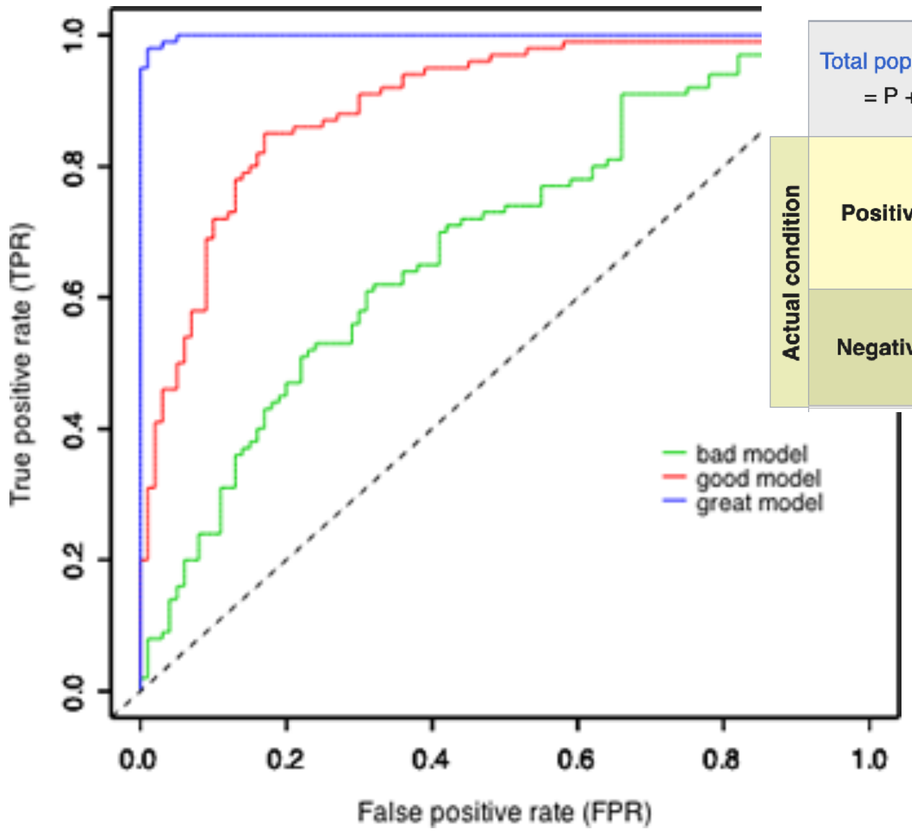
Error Metrics and Evaluation

		Predicted condition		Sources: [6][7][8][9][10][11][12][13][14] view·talk·edit	
	Total population $= P + N$	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)**	**True positive (TP),** hit	**False negative (FN),** type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	Negative (N)	**False positive (FP),** type I error, false alarm, overestimation	**True negative (TN),** correct rejection	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{PN}$ $= 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

https://en.wikipedia.org/wiki/Precision_and_recall

Receiver Operating Characteristic

ROC-AUC

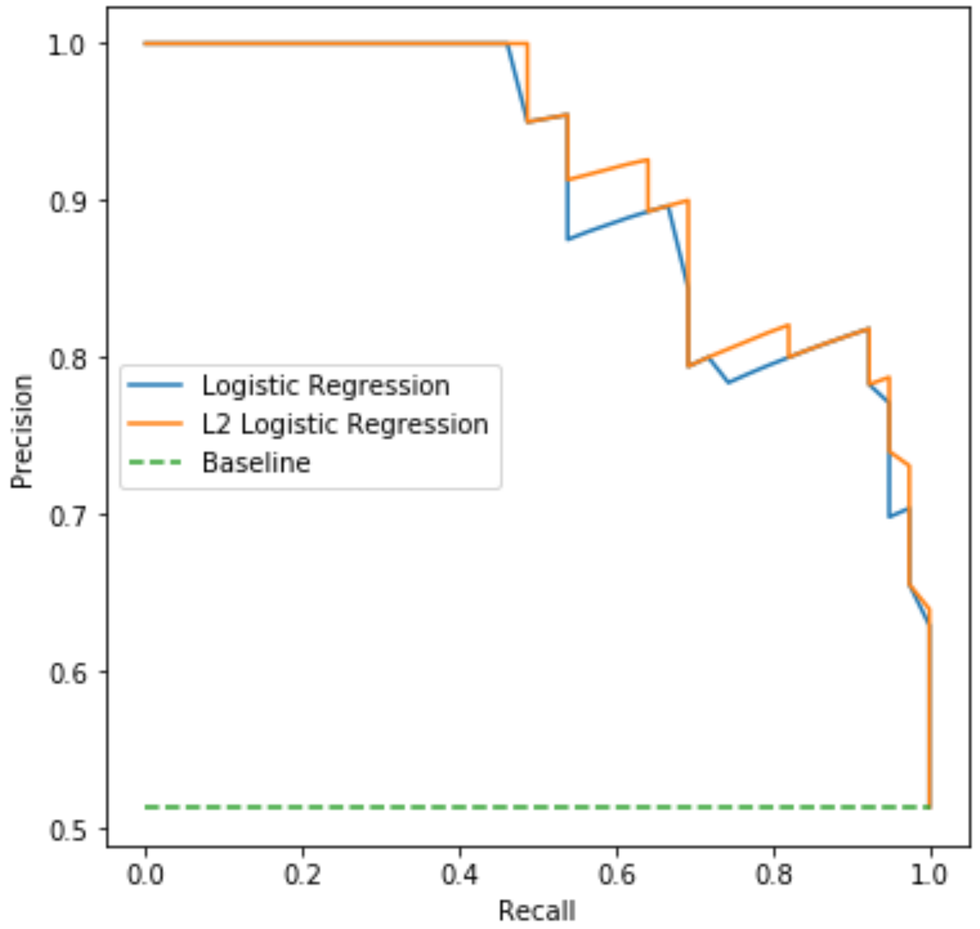


		Predicted condition				
	Total population = P + N	Positive (PP)	Negative (PN)			
Actual condition	Positive (P)	**True positive** (TP), hit	**False negative** (FN), type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$	
	Negative (N)	**False positive** (FP), type I error, false alarm, overestimation	**True negative** (TN), correct rejection	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$	

The value can range from 0 to 1. However AUC score of a random classifier for balanced data is 0.5

Courtesy of Alvira Swalin

Precision - Recall curves



		Predicted condition		
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	
Actual condition	**Positive (P)**	**True positive (TP),** hit	**False negative (FN),** type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$
	Negative (N)	**False positive (FP),** type I error, false alarm, overestimation	**True negative (TN),** correct rejection	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$
	Prevalence $= \frac{P}{P+N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	

Whadda bout regression??

A gentle intro to R-squared

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

The most general definition of the coefficient of determination is

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

In the best case, the modeled values exactly match the observed values, which results in $SS_{\text{res}} = 0$ and $R^2 = 1$. A baseline model, which always predicts $\bar{y}$, will have $R^2 = 0$.

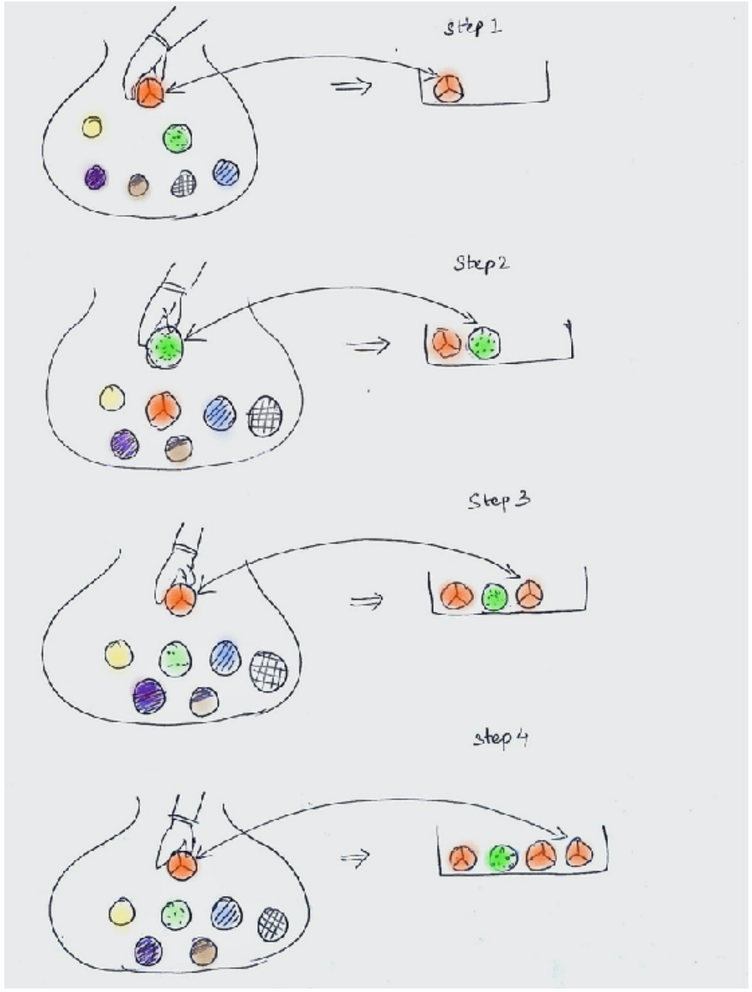
A problem… adding more features/model complexity always increases R-squared…
so there's a penalized version called Adjusted R-squared, also AIC/BIC, etc
OTHER common regression scores: MSE / RMSE, MAE

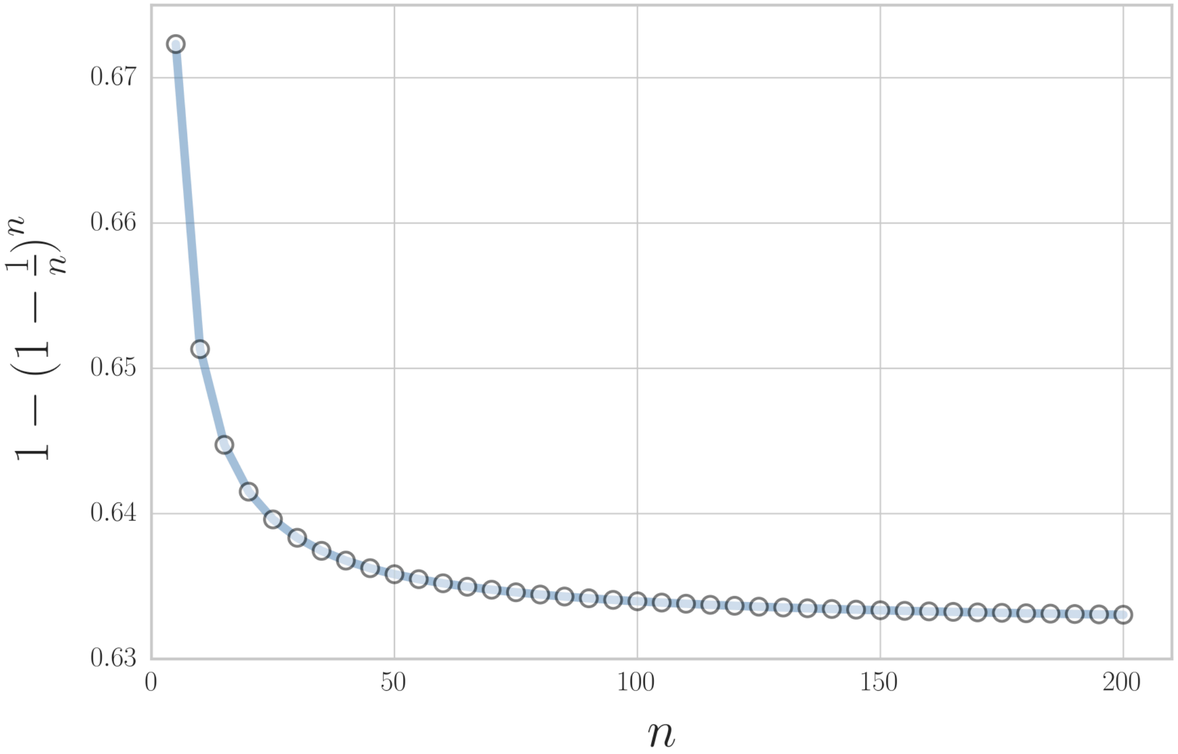
Bootstrap sampling

$$P(\textbf{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$

$$\frac{1}{e} \approx 0.368, \quad n \to \infty\,.$$

$$P(\textbf{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632$$





Evaluating generalization via train - test

Huge data technique

All the data

Randomly assign samples to sets

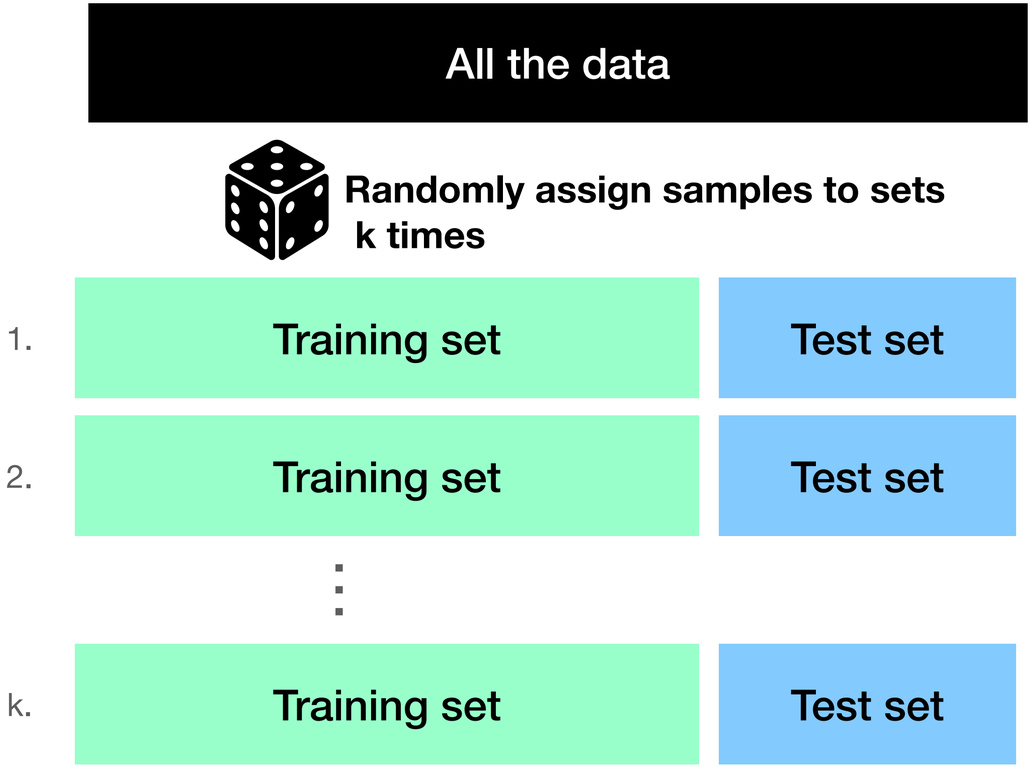
Training set

Test set

$$\epsilon_{\text{testing}} = \epsilon_{\text{training}} + \epsilon_{\text{generalization}}$$

Evaluating generalization via k shuffle splits

Limited data technique



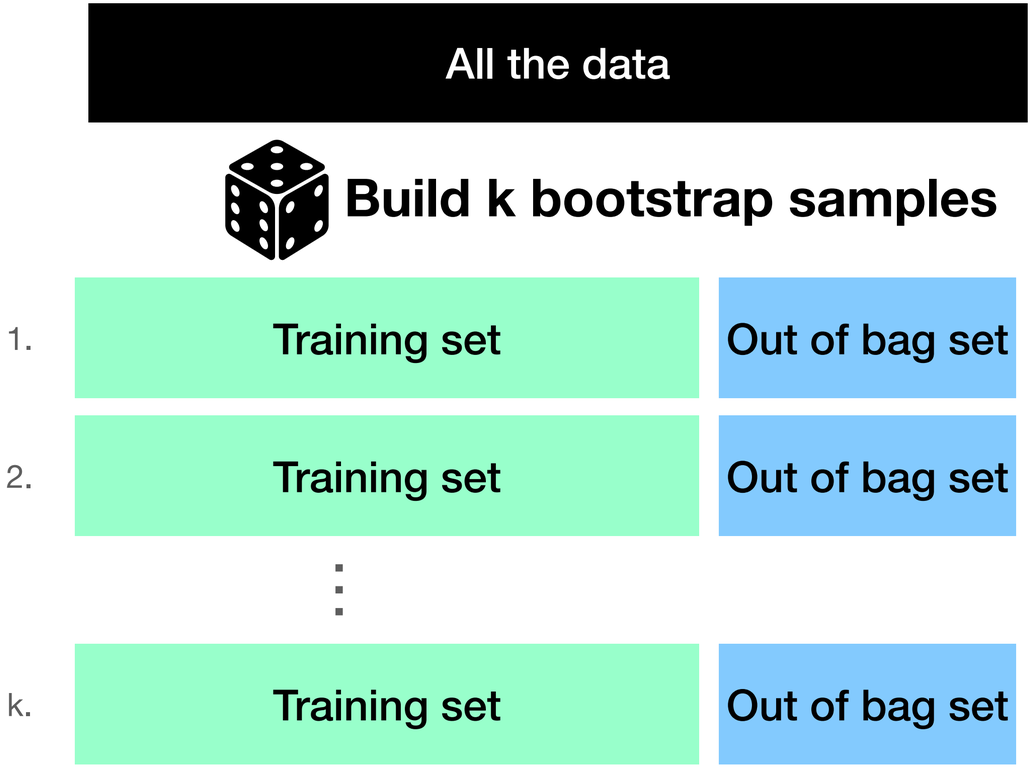
Use the mean of the k test set performances

$$\bar{\epsilon}\text{testing} = \epsilon\text{training} + \epsilon\text{generalization}$$

$$\bar{\epsilon}\text{testing} = 1/k \sum_{i \in [0,k]} \epsilon\text{test}_i$$

Evaluating generalization via bootstrap sampling

Limited data technique



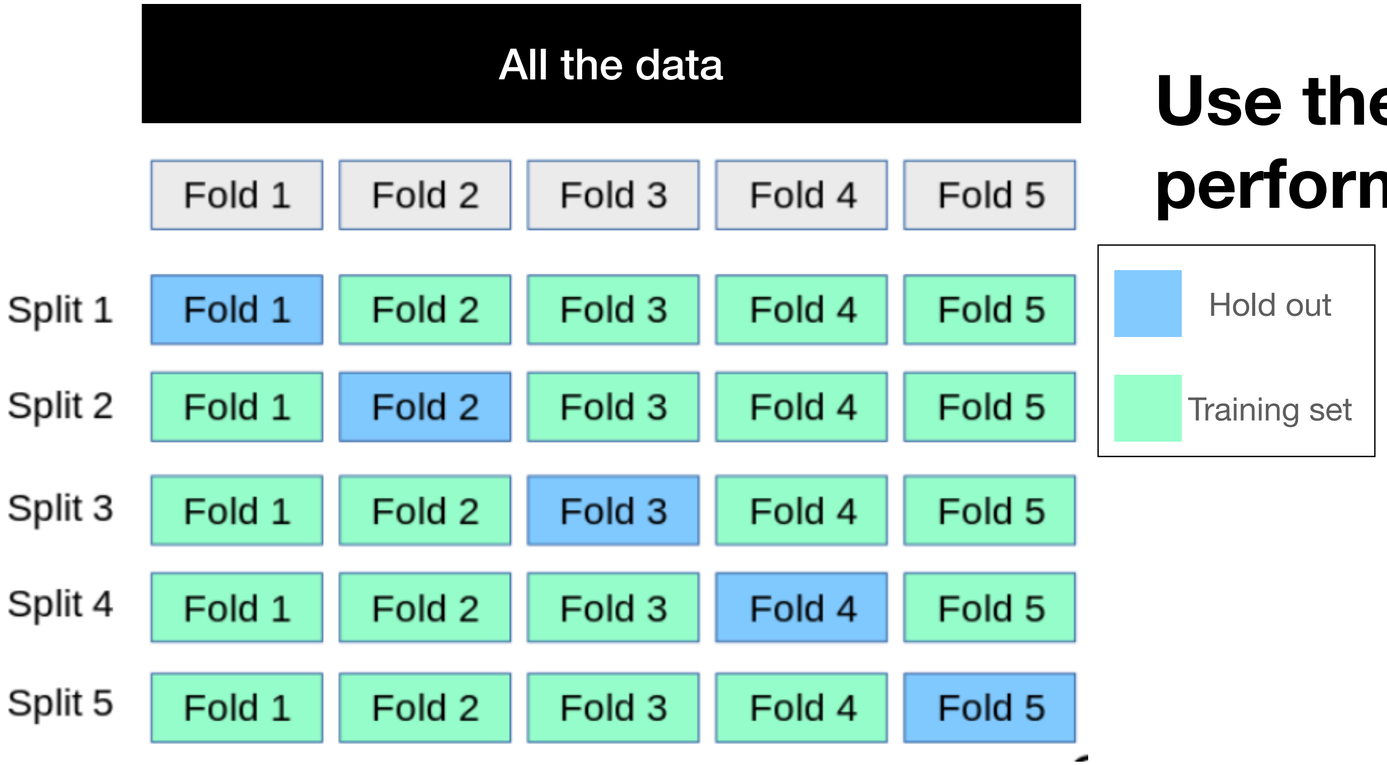
Use the a corrected mean of the n performances, combining a known overestimate with a known underestimate

$$\bar{\epsilon}.632+ = 1/k \sum_{i \in [0,k]} \left(\omega * \epsilon \text{OOB}_i + (1 - \omega) * \epsilon \text{training}_i \right)$$

$$\omega = \frac{.632}{(1 - .368)R}, R = - \frac{\epsilon \text{OOB}_i - \epsilon \text{training}_i}{\gamma - (1 - \epsilon \text{OOB}_i)}$$, where γ is a constant calulated on the dataset

Evaluating generalization via k-folds cross-validation

Limited data technique

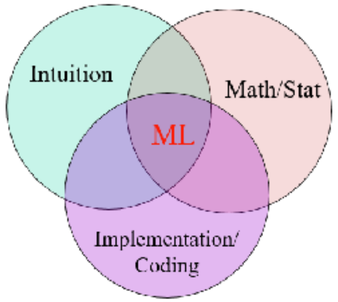


Use the mean of the k hold out set performances

$$\bar{\epsilon}\text{testing} = \epsilon\text{training} + \epsilon\text{generalization}$$

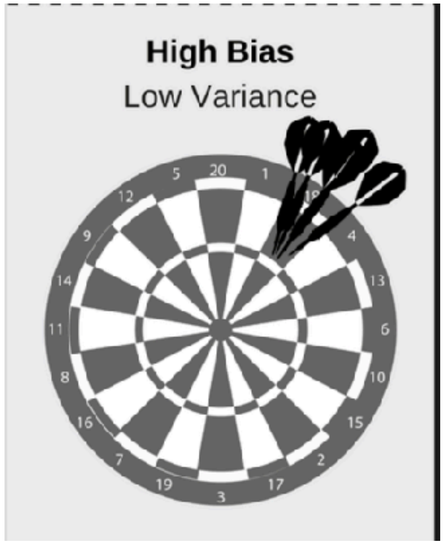
$$\bar{\epsilon}\text{testing} = 1/k \sum_{i \in [0,k]} \epsilon\text{hold out}_i$$

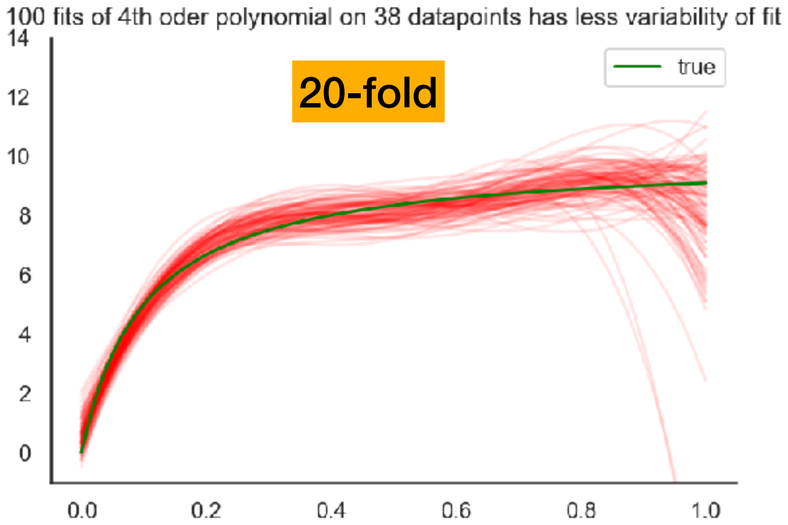
When k = # samples, this is Leave One Out Cross Validation



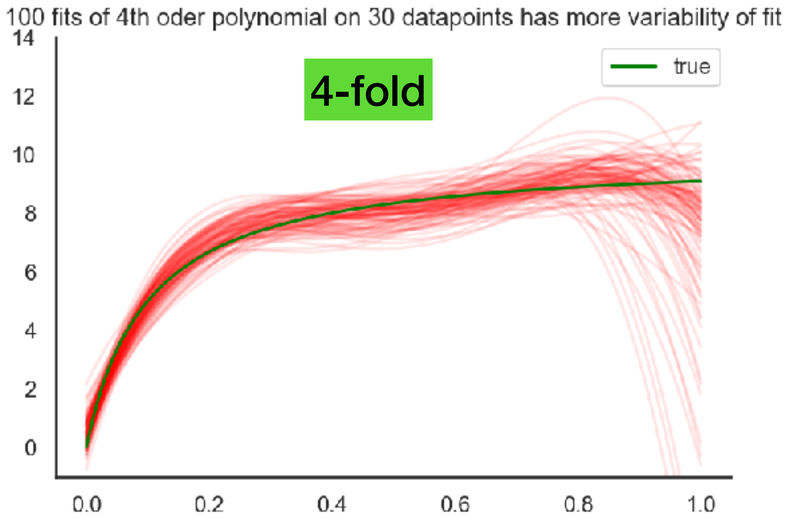
Bias-variance tradeoff in cross validation

In terms of sample size effects on fit

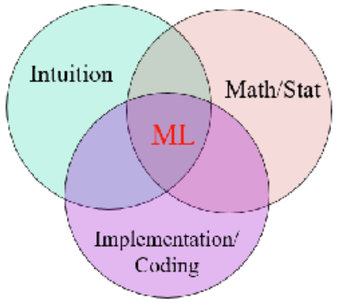




Larger training set size
(K-folds as k gets large -> LOOCV)
more uniformity of fit

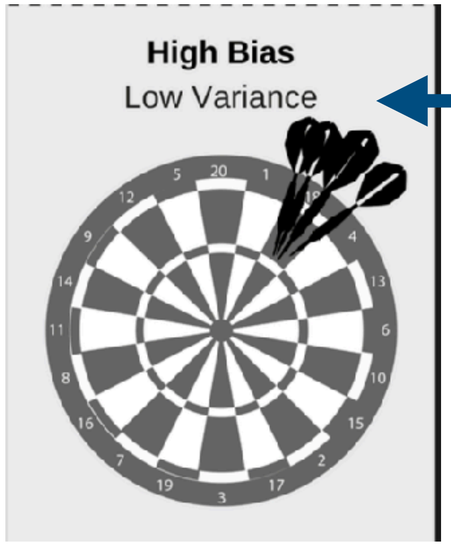


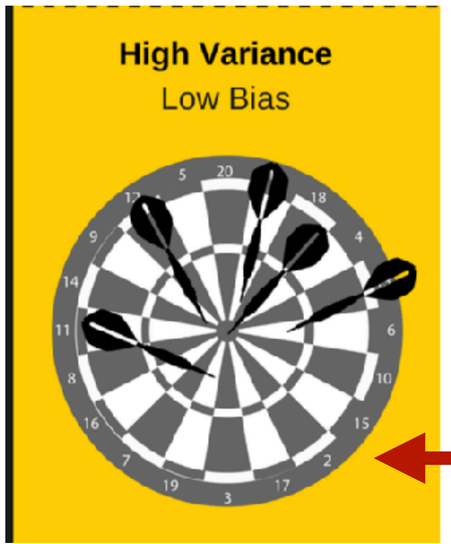
Smaller training set size
(K-folds as k gets small)
more variety of fit

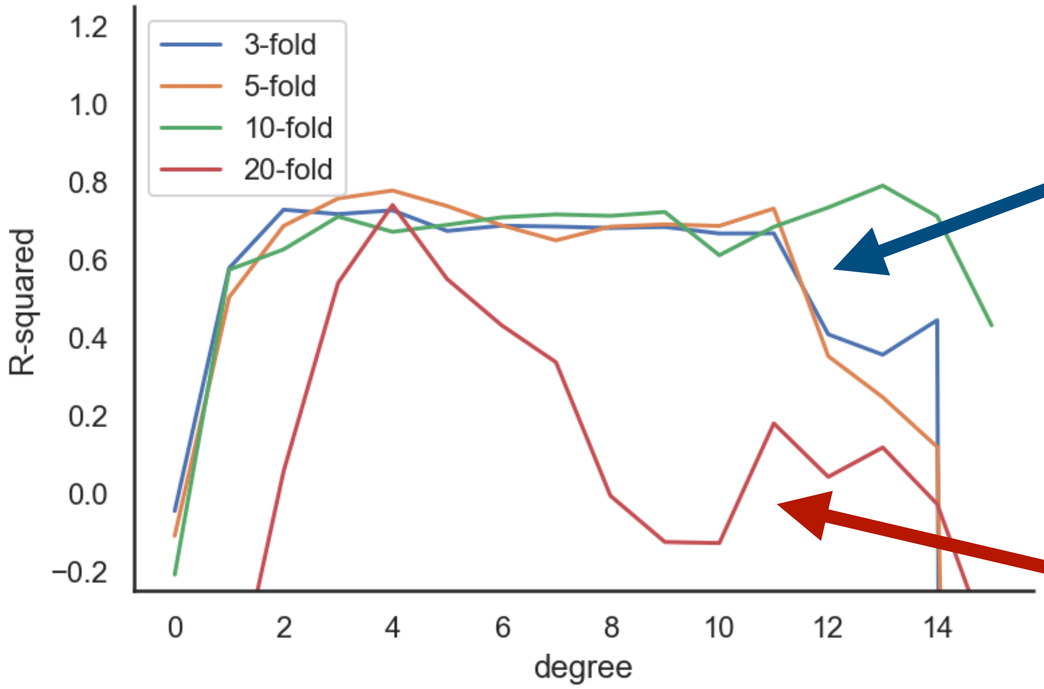


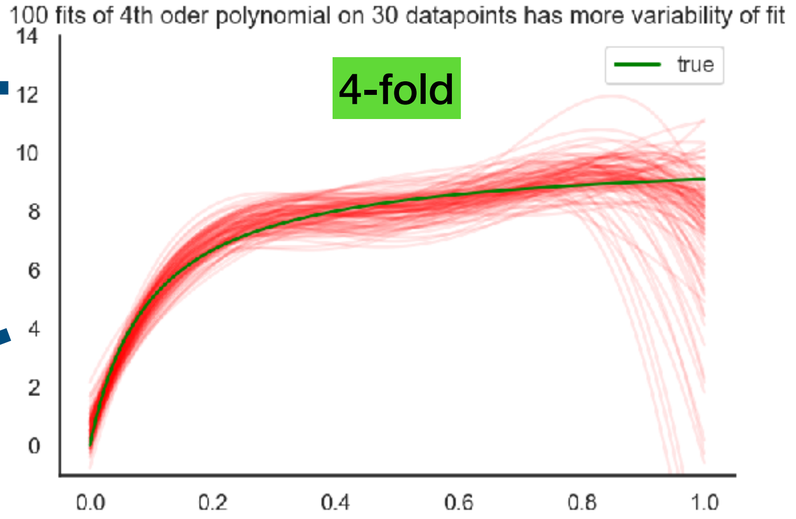
Bias-variance tradeoff in cross validation

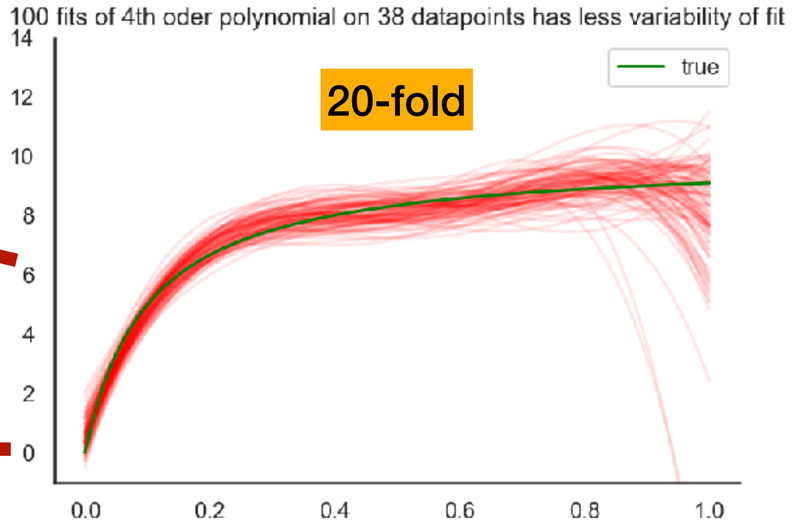
In terms of error estimate its the opposite, ⬆️ fit variability == ⬇️ estimate var



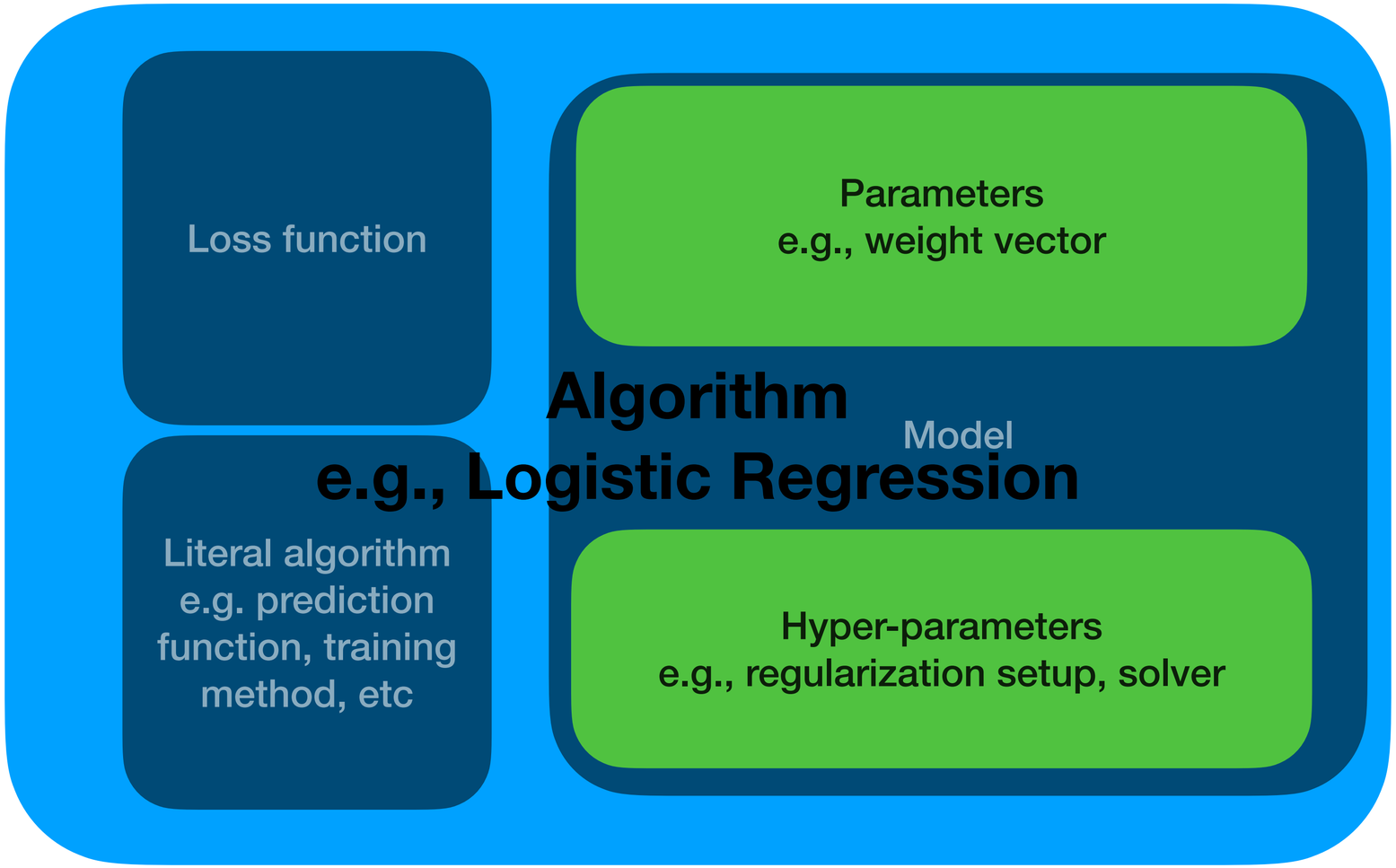








Week 5


Loss function
Parameters
e.g., weight vector
Algorithm
Model
e.g., Logistic Regression
Literal algorithm
e.g. prediction
function, training
method, etc
Hyper-parameters
e.g., regularization setup, solver

Model or Algorithm selection

Method #0 - Internet sized datasets

- Split data into train, validate, test
- [OPTIONAL] Outer loop… do this T times:
  - For each model in the hyper-parameter space or each algorithm-model combination:
    - Train it on training
    - Predict on the validation set
- Pick the best model or algorithm based on its performance on [the mean across trials] of the validation set
- Train the best version on the whole of training set + validation set
- Test it on the test set to estimate its ability to generalize

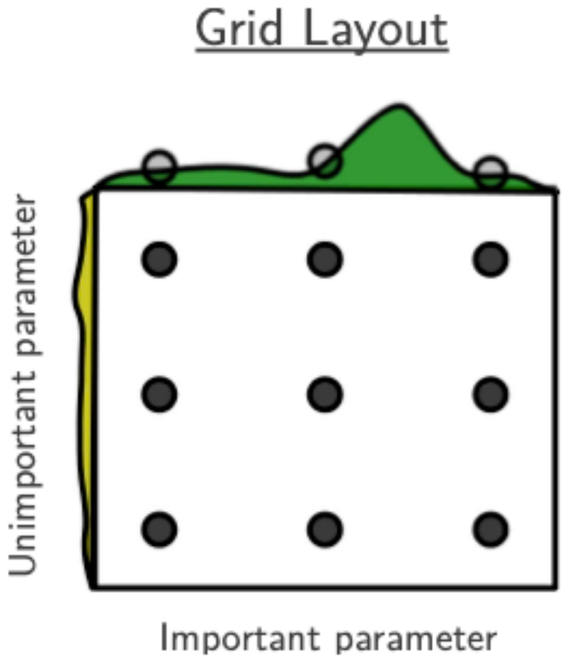
Model or Algorithm selection

Method #1 - Medium sized datasets

- Split data into cross-validation and test sets
- [OPTIONAL] Outer loop… do this T times:
  - For each model in the hyper-parameter space or each algorithm-model combination:
    - Use k-fold cross validation to estimate validation error
- Pick the best model or algorithm based on its performance on [the mean across trials] of the validation sets
- Train the best version on the whole of cross validation set
- Test it on the test set to estimate its ability to generalize

Grid Search

- Exhaustive search
- Thorough but expensive
- Specify grid for parameter search
- Can be run in parallel
- Can suffer from poor coverage
- Often run with multiple resolutions



Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, *13*(1), 281-305.

Randomized Search

- Search based on a time budget
- Preferred if there are many hyperparameters (e.g. > 3 distinct ones)
- specify distribution for parameter search
- can be run in parallel

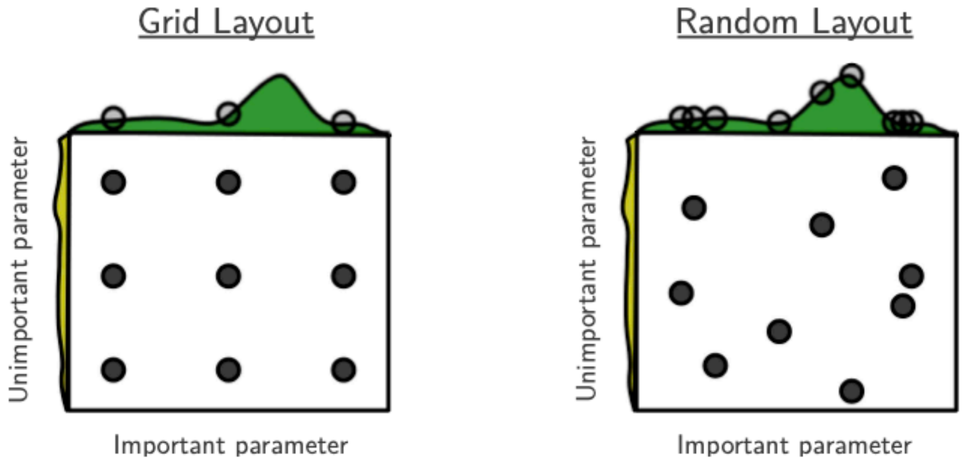


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, *13*(1), 281-305.

Parsimony Principle

Choose the simplest w/in 1 std error of optimal

Which parameter would you select?