Computer Science

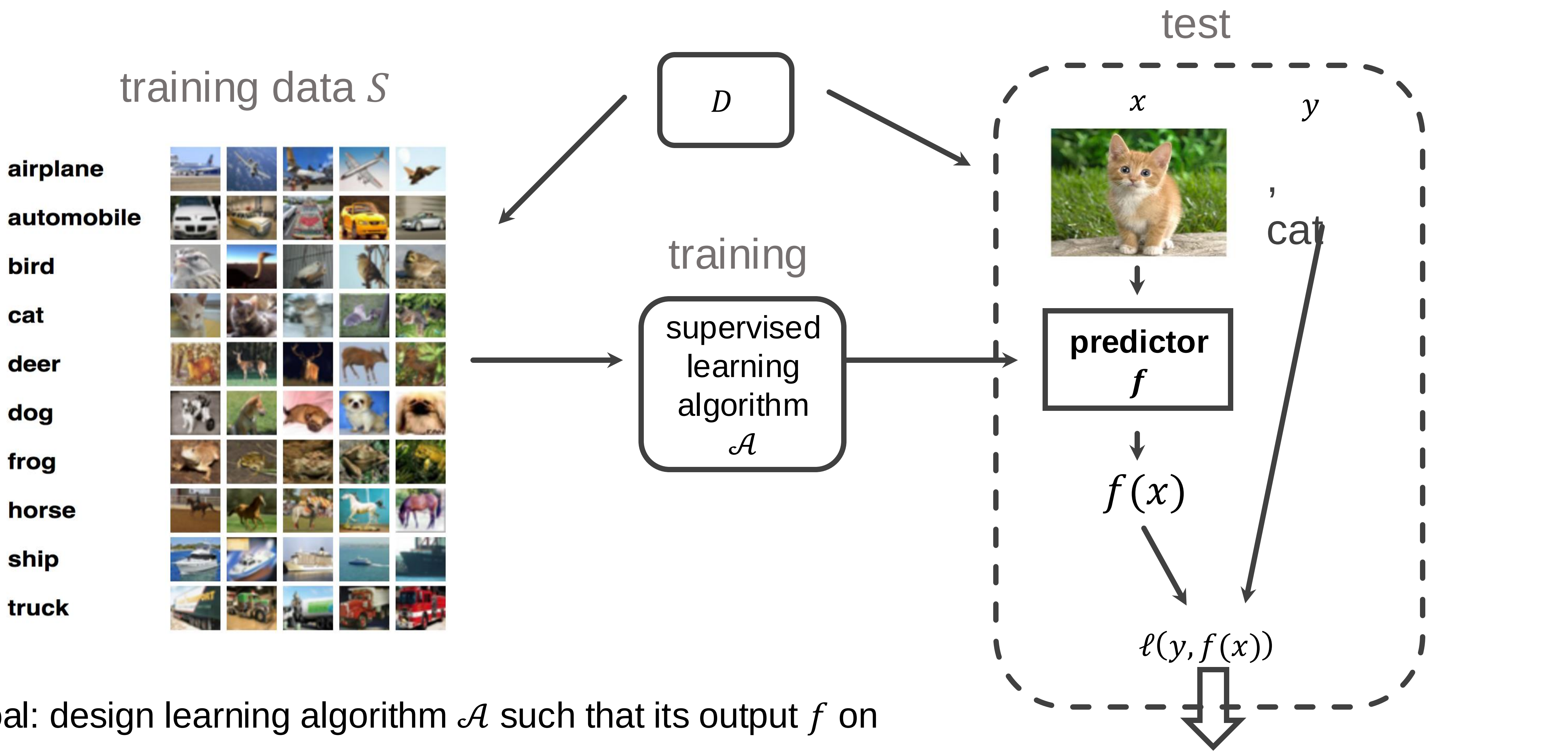
CSC580: Probabilistic Graphical Models

Midterm Review

Jason Pacheco

Supervised Learning

Supervised learning setup: putting it together



- Goal: design learning algorithm $\mathcal{A}$ such that its output f on iid training data S has low generalization error

Generalization error: $L_D(f) = \mathrm{E}_{(x,y)\sim D}\, \ell(y, f(x))$

Model: Decision Tree: Example

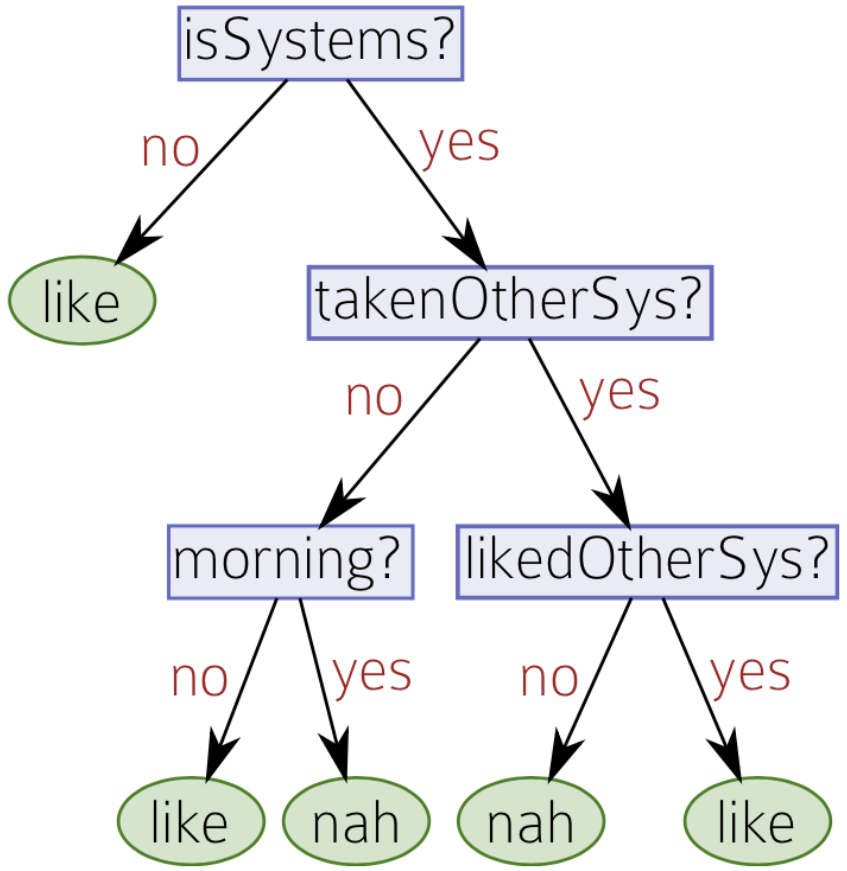


Figure 1.2: A decision tree for a course recommender system, from which the in-text "dialog" is drawn.

Input: the course & student info

Use questions to arrive at a conclusion.

Terminology:

- (Question, Answer) → (Feature, Feature Value)
- "Like" / "Nah" → Label
- {(A set of (Question & Answer)'s, Label)} → Train Data

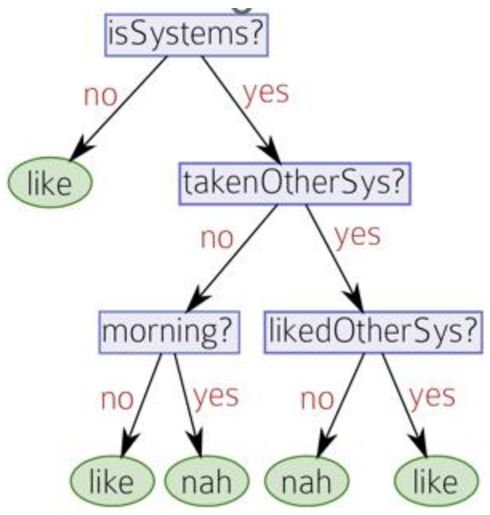
Prediction using a decision tree

- Test: predict using a decision tree:

test point: the data point to be classified
(vs train point: data point to be used for training)

Algorithm 2 DecisionTreeTest(*tree*, *test point*)

```
1: if tree is of the form Leaf(guess) then
2:     return guess
3: else if tree is of the form Node(f, left, right) then
4:     if $f = no$ in test point then
5:         return DecisionTreeTest(left, test point)
6:     else
7:         return DecisionTreeTest(right, test point)
8:     end if
9: end if
```



guess = prediction

left = no
right = yes

- Training: how to design a learning algorithm $\mathcal{A}$ that can build trees f from training data?

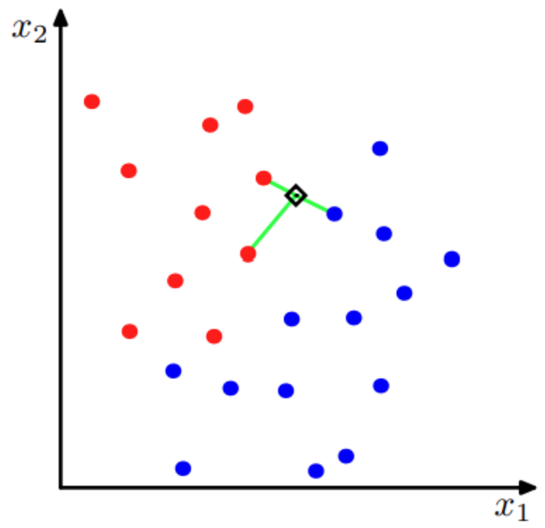
k-nearest neighbors (k-NN): main concept

Training set: $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$

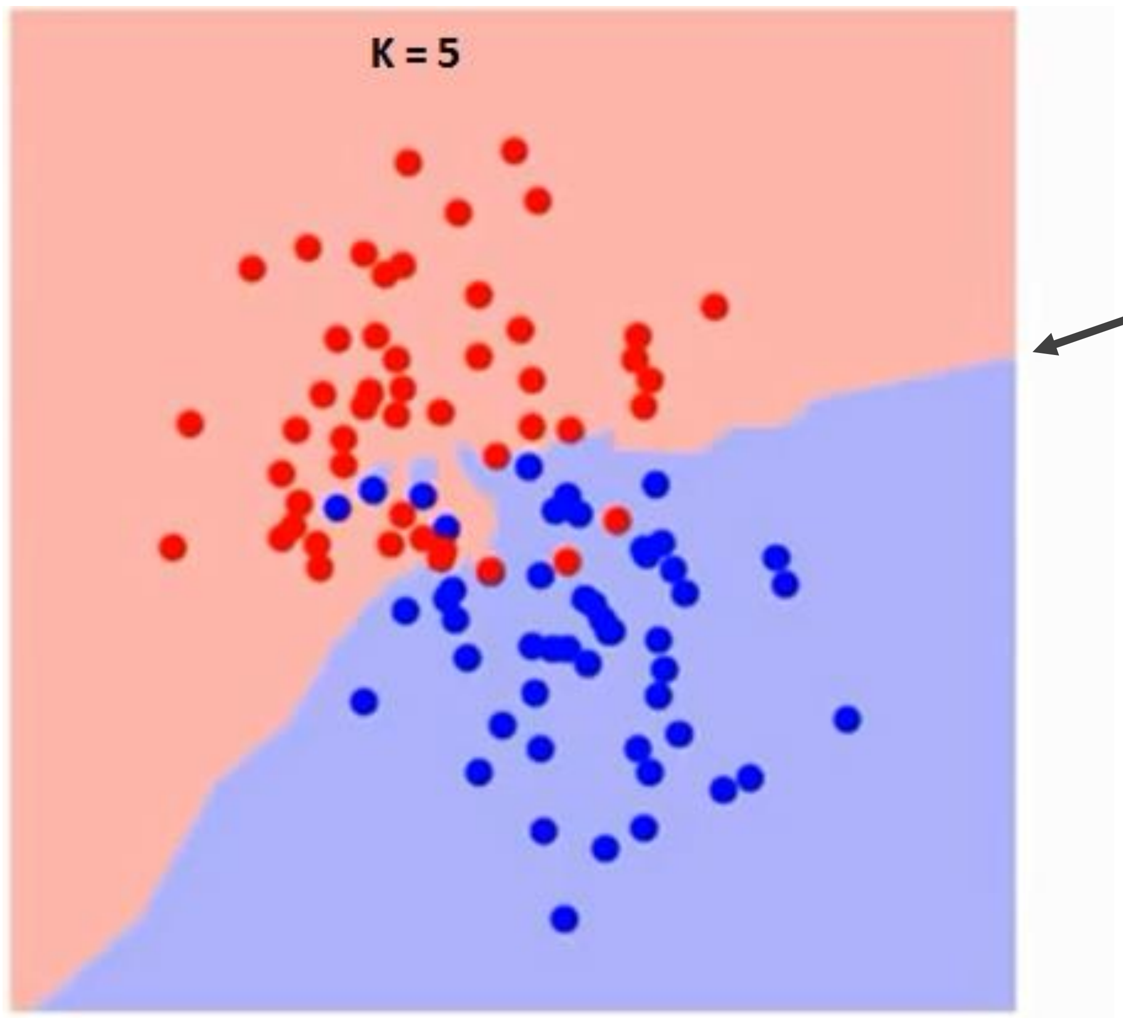
Inductive bias: given test example x, its label should resemble the labels of **nearby points**

Function

- input: x
- find the k nearest points to x from S; call their indice
- output: the majority vote of $\{y_i : i \in N(x)\}$
  - For regression, the average.



k-NN classification example



decision boundary

k-NN classification: pseudocode

- Training is trivial: store the training set
- Test:

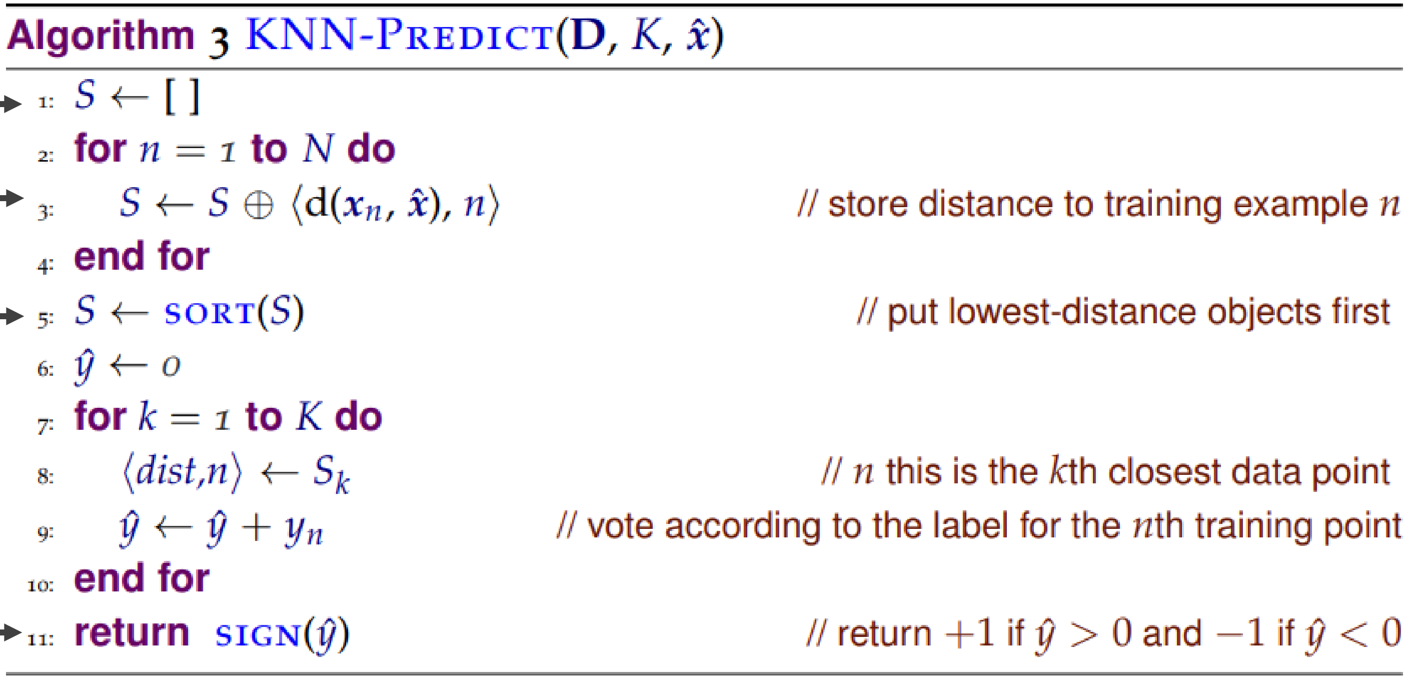
list

append to list

sort in first coordinate

Majority vote of $\{y_i : i \in N(x)\}$

-



Algorithm 3 KNN-PREDICT($\mathbf{D}$, K, $\hat{x}$)

```
S ← [ ]
for $n = 1$ to $N$ do
    $S \leftarrow S \oplus \langle d(x_n, \hat{x}), n \rangle$        // store distance to training example $n$
end for
$S \leftarrow$ SORT($S$)        // put lowest-distance objects first
$\hat{y} \leftarrow 0$
for $k = 1$ to $K$ do
    $\langle dist, n \rangle \leftarrow S_k$        // $n$ this is the $k$th closest data point
    $\hat{y} \leftarrow \hat{y} + y_n$        // vote according to the label for the $n$th training point
end for
return SIGN($\hat{y}$)        // return $+1$ if $\hat{y} > 0$ and $-1$ if $\hat{y} < 0$
```

- Time complexity (assuming distance calculation takes $O(d)$ time)
  - $O(m\,d + m \log m + k) = O\big(m(d + \log m)\big)$
- Faster nearest neighbor search: k-d trees, locality sensitive hashing

Background: Train set accuracy/error

- Suppose the ML algorithm has trained a function f using the dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$
- Train set accuracy:

$$\widehat{\mathrm{acc}}(f) := \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}\{f(x_i) = y_i\}$$

- Train set error: $\widehat{\mathrm{err}}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}\{f(x_i) \neq y_i\} = 1 - \widehat{\mathrm{acc}}(f)$

- Q: We have 100 train set (images) consisting of 5 cats, 80 dogs, and 15 lions. What is the train set accuracy of the majority vote classifier? What is the error?

Bayes optimal classifier

Theorem f_{BO} achieves the smallest 0-1 error among all classifiers.

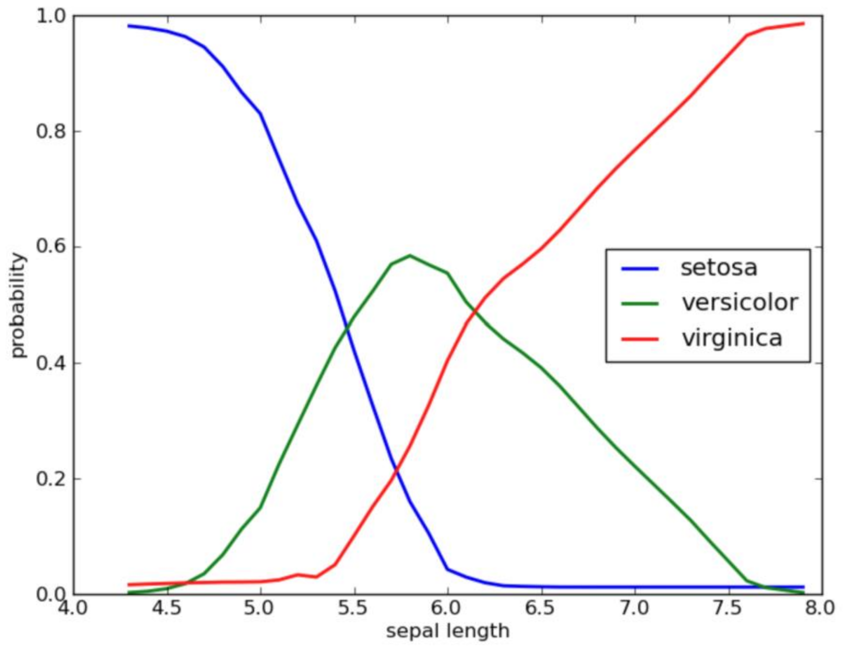
$$f_{BO}(x) = \arg\max_{y \in \mathcal{Y}} P_D(X = x, Y = y) = \arg\max_{y \in \mathcal{Y}} P_D(Y = y \,|X = x)\,, \forall x \in \mathcal{X}$$

Example Iris dataset classification:

Iris Setosa

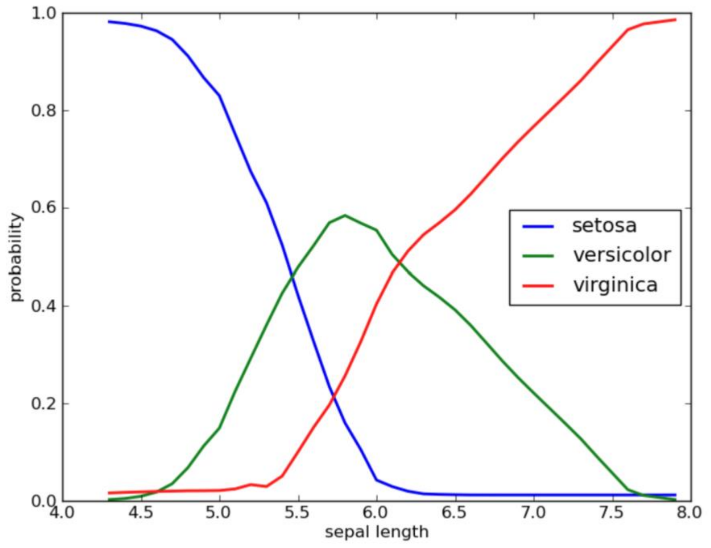
Iris Versicolor

Iris Virginica

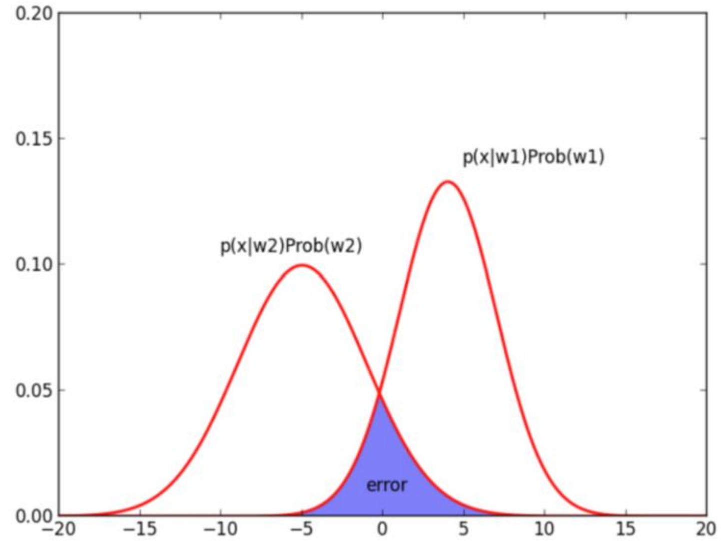


Bayes error rate: alternative form

$$\begin{aligned} L_D(f_{BO}) &= P_D\big(Y \neq f_{BO}(X)\big) \\ &= \textstyle\sum_x P_D(Y \neq f_{BO}(x) \mid X = x)\, P_D(X = x) \\ &= \textstyle\sum_x (1 - P_D(Y = f_{BO}(x) \mid X = x))\, P_D(X = x) \\ &= \textstyle\sum_x \left(1 - \max_y P_D(Y = y \mid X = x)\right) P_D(X = x) \\ &= \mathrm{E}\left[1 - \max_y P_D(Y = y \mid X)\right] \end{aligned}$$

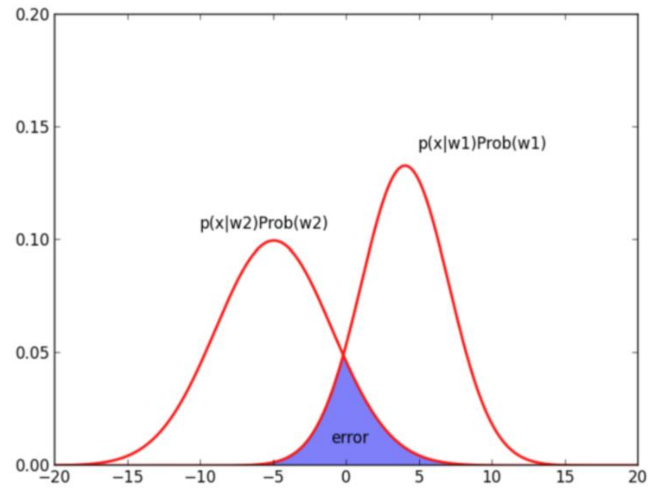


- Special case: binary classification
  - $L_D(f_{BO}) = \sum_x P_D(Y \neq f_{BO}(x), X = x)$
    $= \sum_x \min(\, P_D(Y = +1, X = x)\,, P_D(Y = -1, X = x))$

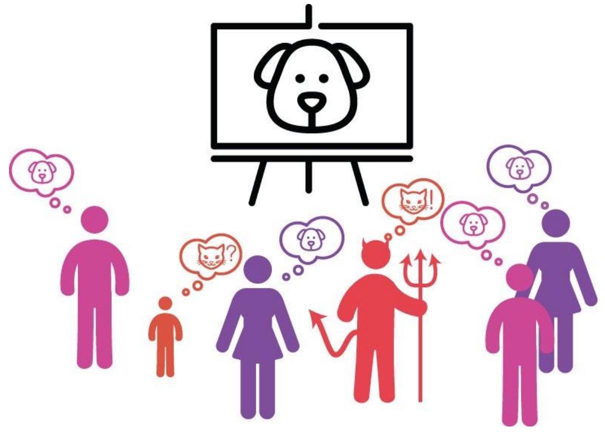


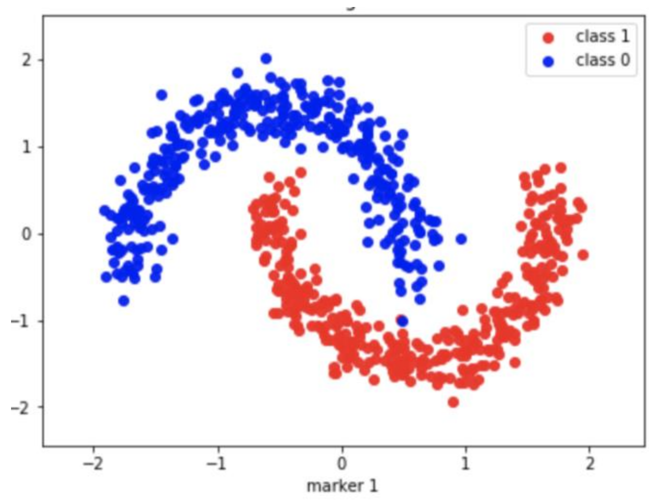
When is the Bayes error rate nonzero?

$$L_D(f_{BO}) = \sum_x \min(\, P_D(Y=+1, X=x)\,, P_D(Y=-1, X=x))$$



- Limited feature representation
- Noise in the training data
  - Feature noise
  - Label noise
  - Sensor failure
  - Typo in reviews for sentiment classification
- May not be a single “correct” answer
- Inductive bias of the model / learning algorithm

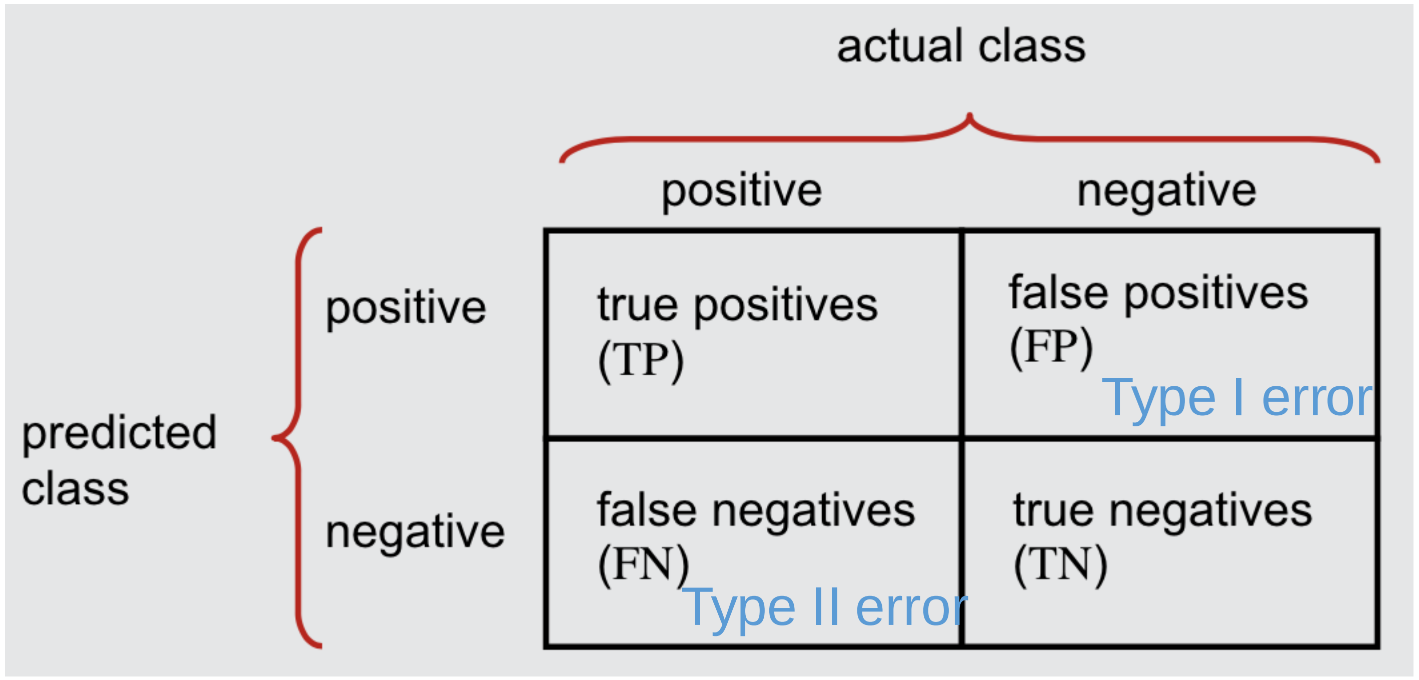




Model Validation and Selection

New measures of classification performance

- True positive rate (TPR) $= \frac{\mathrm{TP}}{\mathrm{P}} = \frac{P(\hat{y}=+1, y=+1)}{P(y=+1)}$
  (aka recall, sensitivity)
- True negative rate (TNR) = $\frac{\mathrm{TN}}{\mathrm{N}}$
  (specificity)
- False positive rate (FPR) = $\frac{\mathrm{FP}}{\mathrm{N}}$
- False negative rate (FNR) = $\frac{\mathrm{FN}}{\mathrm{P}}$

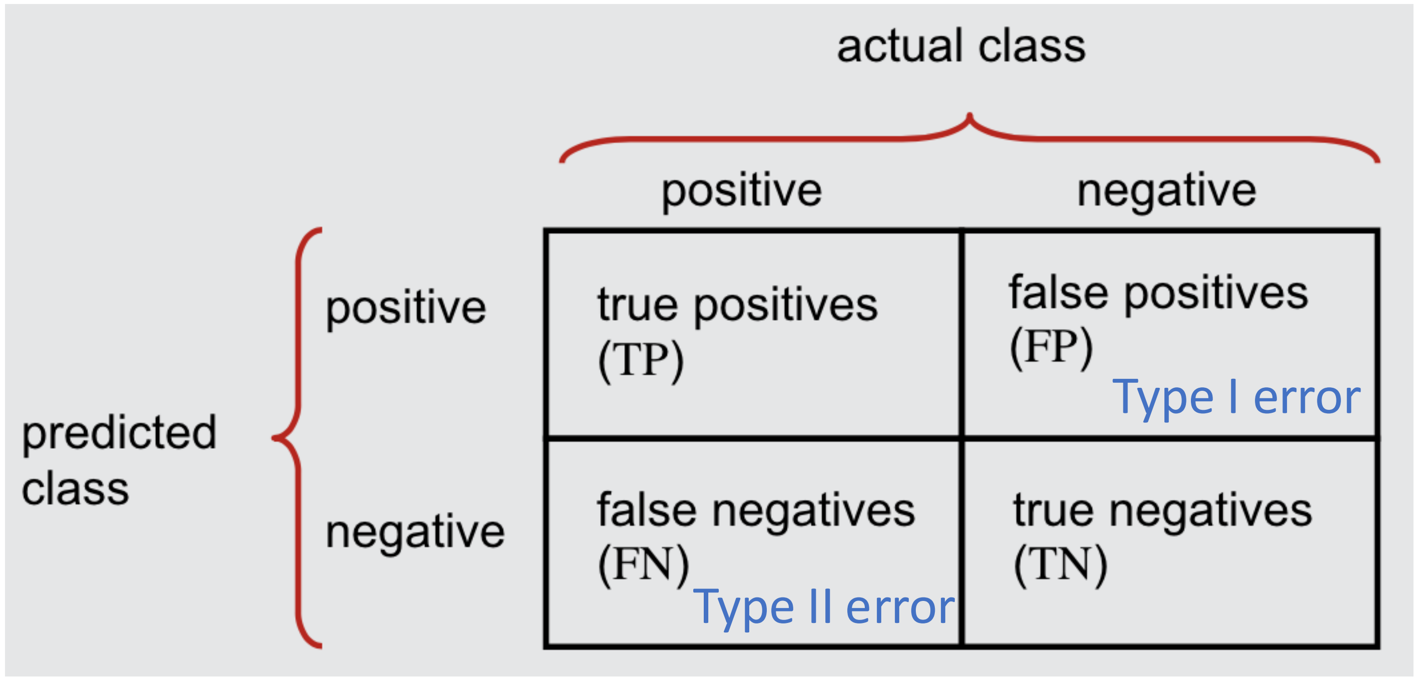


P = TP + FN   N = FP + TN

- Precision = $\frac{\mathrm{TP}}{\mathrm{P}-called} = \frac{P(\hat{y}=+1, y=+1)}{P(\hat{y}=+1)}$, $\mathrm{P} - called = \mathrm{TP} + \mathrm{FP}$

New measures of classification performance

- True positive rate (TPR) $= \frac{\text{TP}}{\text{P}} = \frac{P(\hat{y}=+1, y=+1)}{P(y=+1)}$ (aka recall, sensitivity)
- True negative rate (TNR) = $\frac{\text{TN}}{\text{N}}$ (specificity)
- False positive rate (FPR) = $\frac{\text{FP}}{\text{N}}$
- False negative rate (FNR) = $\frac{\text{FN}}{\text{P}}$
- Precision = $\frac{\text{TP}}{\text{P}-called} = \frac{P(\hat{y}=+1, y=+1)}{P(\hat{y}=+1)}$, $\text{P} - called = \text{TP} + \text{FP}$



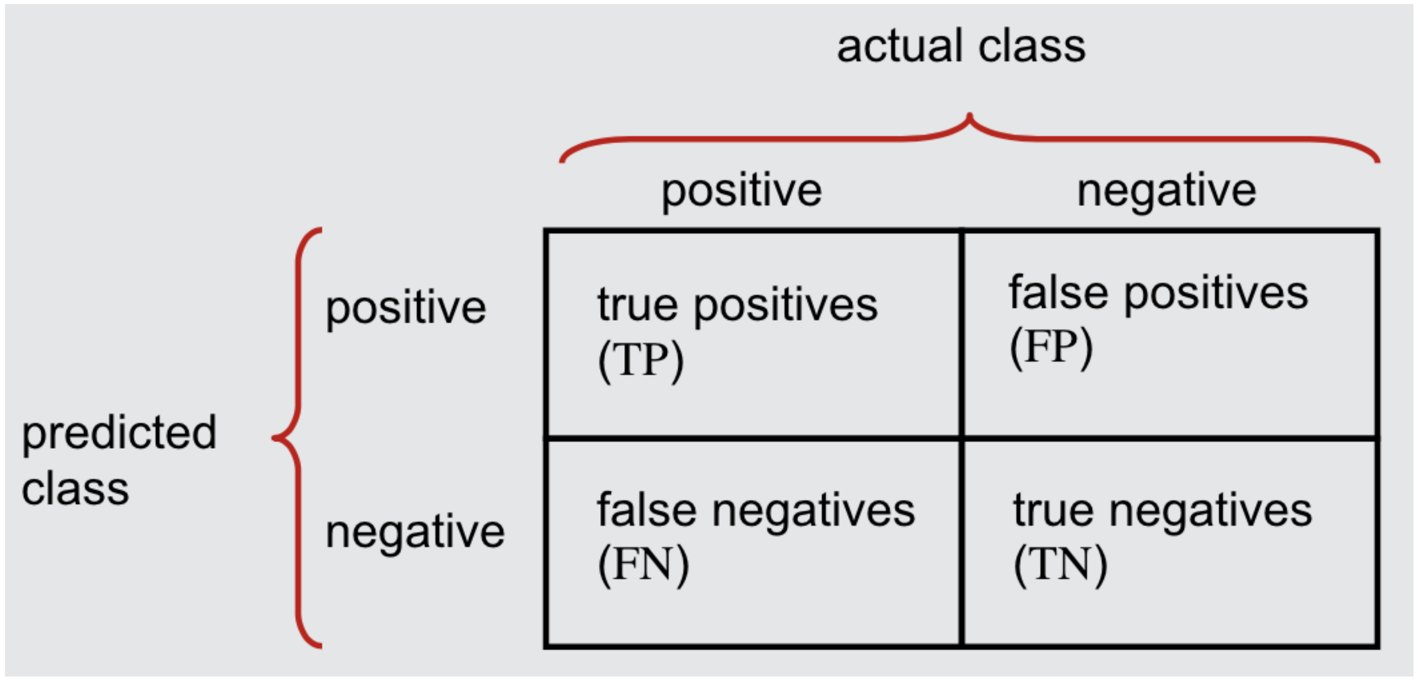
		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP) Type I error
	negative	false negatives (FN) Type II error	true negatives (TN)

P = TP + FN N = FP + TN

Applications:
- Search engine: precision & recall
- Cancer classification: FNR vs. FPR

Adjust TP, FP, TN, FN

- Decision values
  - E.g., the predicted $P(Y = 1|X = x)$
  - Some classifiers just have a real-value where positive value indicates positive prediction.
    (e.g, supper vector machine – will be covered later)



		actual class: positive	actual class: negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

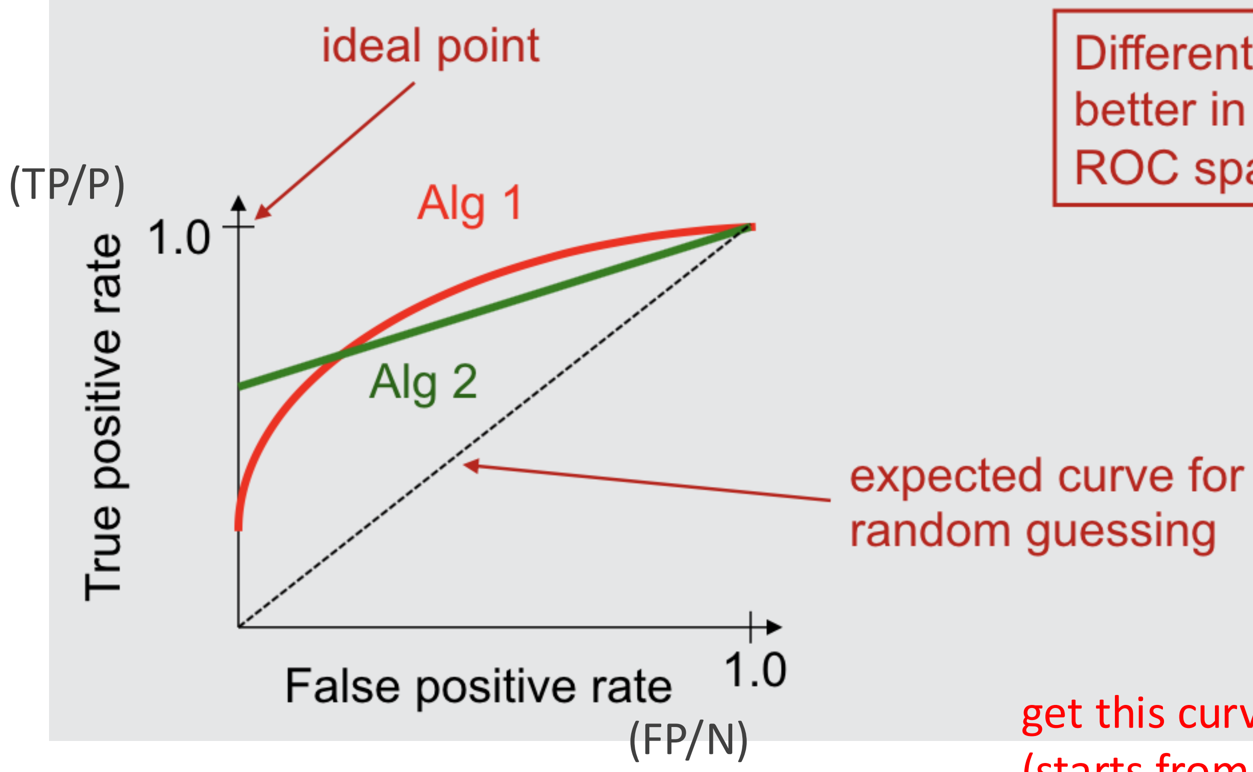
P = TP + FN N = FP + TN

- Default: $P(Y = 1|X = x) \geq .5$ then call it positive
  - Threshold to $1.1 \Rightarrow$ always predict neg.
  - Threshold to $0 \Rightarrow$ always predict pos.

TPR	FPR
0 bad	0
1	1 bad

ROC curve

A *Receiver Operating Characteristic* (*ROC*) curve plots the TP-rate vs. the FP-rate as a threshold on the confidence of an instance being positive is varied



Different methods can work better in different parts of ROC space.

get this curve by varying the threshold from large to small (starts from (0,0) then goes to left and downwards to (1,1)) (the green curve is misleading)

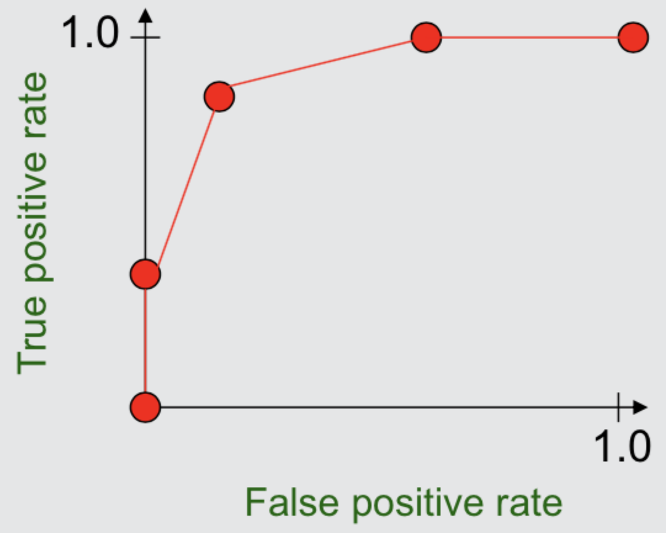
ROC curve

- **Conceptually**, consider every possible threshold, put a dot for each, and connect them.
- **In practice**, just need to care about when the 'correct class' changes from + to – or from – to +.
  - results in staircase shape, but diagonal line can still happen.
- A popular alternative: just plot when going from + to -. (what's shown here)

decision value; sorted in decreasing order

TPR=0, FPR=0

instance	confidence positive		correct class
Ex 9	.99		+
Ex 7	.98	TPR= 2/5, FPR= 0/5	+
Ex 1	.72		-
Ex 2	.70		+
Ex 6	.65	TPR= 4/5, FPR= 1/5	+
Ex 10	.51		-
Ex 3	.39		-
Ex 5	.24	TPR= 5/5, FPR= 3/5	+
Ex 4	.11		-
Ex 8	.01	TPR= 5/5, FPR= 5/5	-

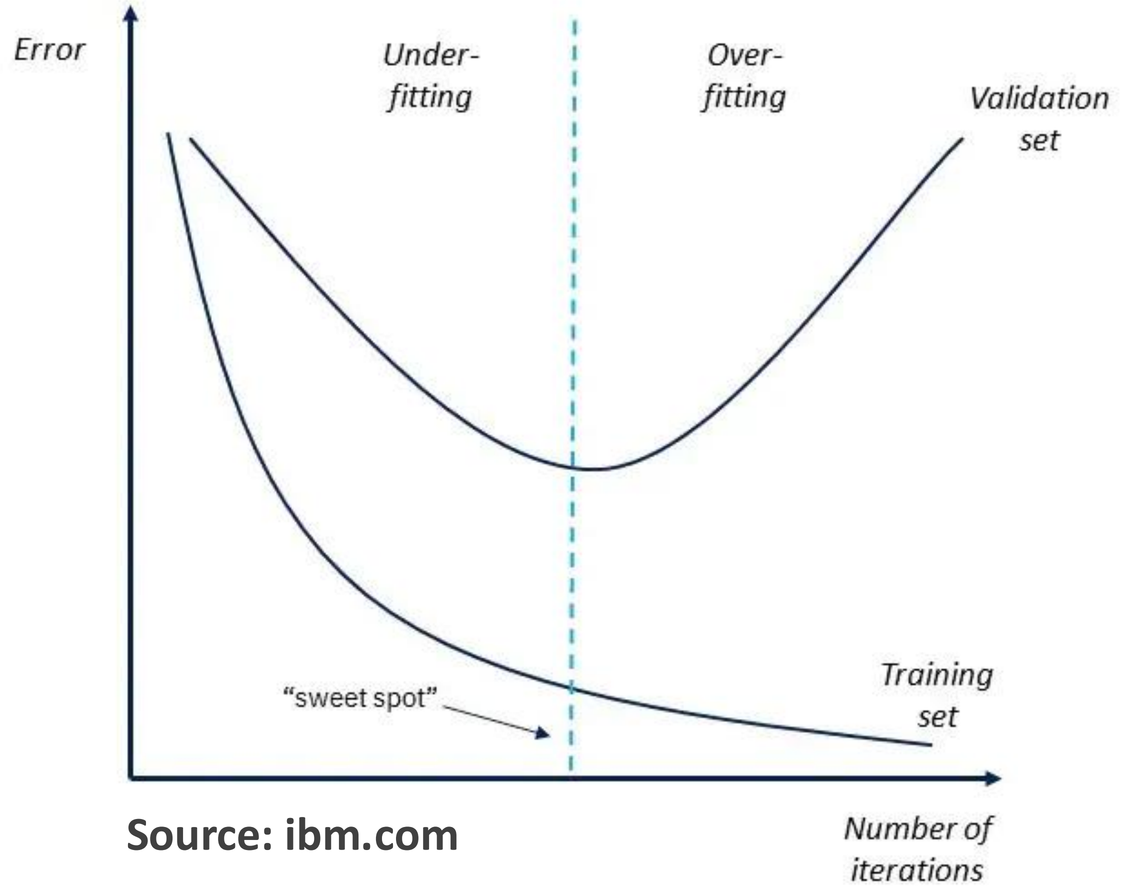


ROC curve algorithm

```
let $\left(\left(y^{(1)}, c^{(1)}\right) \ldots \left(y^{(m)}, c^{(m)}\right)\right)$ be the test-set instances sorted according to predicted confidence
    $c^{(i)}$ that each instance is positive
let $num\_neg$, $num\_pos$ be the number of negative/positive instances in the test set
$TP = 0,\ FP = 0$
$last\_TP = 0$
for $i = 1$ to $m$
    // find thresholds where there is a pos instance on high side, neg instance on low side
    if ($i > 1$) and ( $c^{(i)} \neq c^{(i-1)}$ ) and ( $y^{(i)}$ == neg ) and ( $TP > last\_TP$ )
        $\leftarrow$ $FPR = FP / num\_neg,\ \ TPR = TP / num\_pos$
        output ($FPR, TPR$) coordinate
        $last\_TP = TP$
    if $y^{(i)}$ == pos
        $++TP$
    else
        $++FP$
$FPR = FP / num\_neg,\ \ TPR = TP / num\_pos$
output ($FPR, TPR$) coordinate
```

Overfitting vs Underfitting

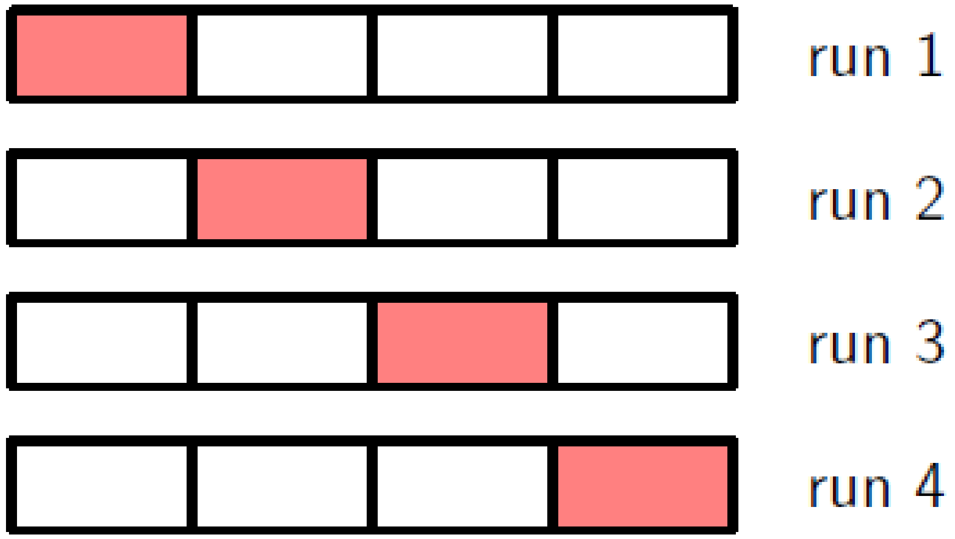
Underfitting performs poorly on *both* training and validation…



Source: ibm.com

…overfitting performs well on training but not on validation

Cross-Validation



N-fold Cross Validation Partition training data into N “chunks” and for each run select one chunk to be validation data

For each run, fit to training data (N-1 chunks) and measure accuracy on validation set. Average model error across all runs.

Drawback Need a lot of training data to partition.

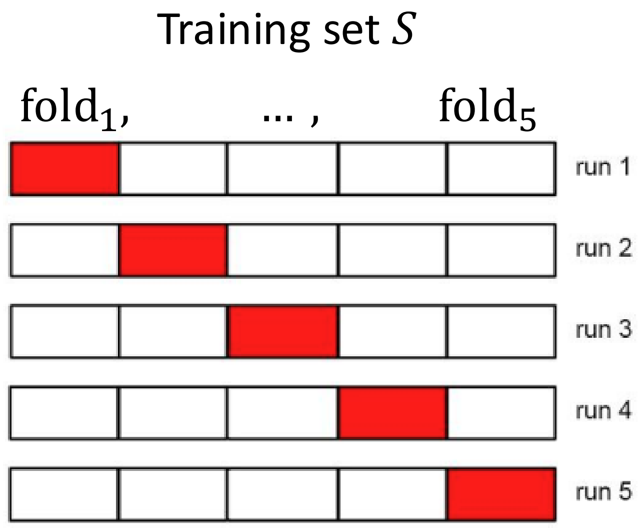
Source: Bishop, C. PRML

Hyperparameter tuning: cross-validation

- Main idea: split the training / validation data in multiple ways

- For hyperparameter $h \in \{1, \dots, H\}$
    - For $k \in \{1, \dots, K\}$
        - train $\hat{f}_k^h$ with $S \setminus \mathrm{fold}_k$
        - measure error rate $e_{h,k}$ of $\hat{f}_k^h$ on fold_k
    - Compute the average error of the above: $\widehat{\mathrm{err}}^h = \frac{1}{K} \sum_{k=1}^{K} e_{h,k}$
- Choose $\hat{h} = \arg\min_h \widehat{\mathrm{err}}^h$
- Train $\hat{f}$ using S (all the training points) with hyperparameter $\hat{h}$

- $k = |S|$: leave one out cross validation (LOOCV)



Interval Estimation / Hypothesis Testing

Motivation: evaluating & comparing ML models

Example

- Your ML model f has test set error = 6.9%
- Your nemesis, Gabe's, ML model g has test set error = 6.8%
- How confident are we to conclude that g has smaller generalization error than that of f?

- Intuition: We should be more confident if the test set is larger, less if it's smaller
- Our uncertainty can be quantified with a *confidence interval*
- Determining the best model can be done rigorously with *hypothesis testing*

Disclaimer: we only focus on the key ideas (standard stats courses spend >= 5 lectures on this)

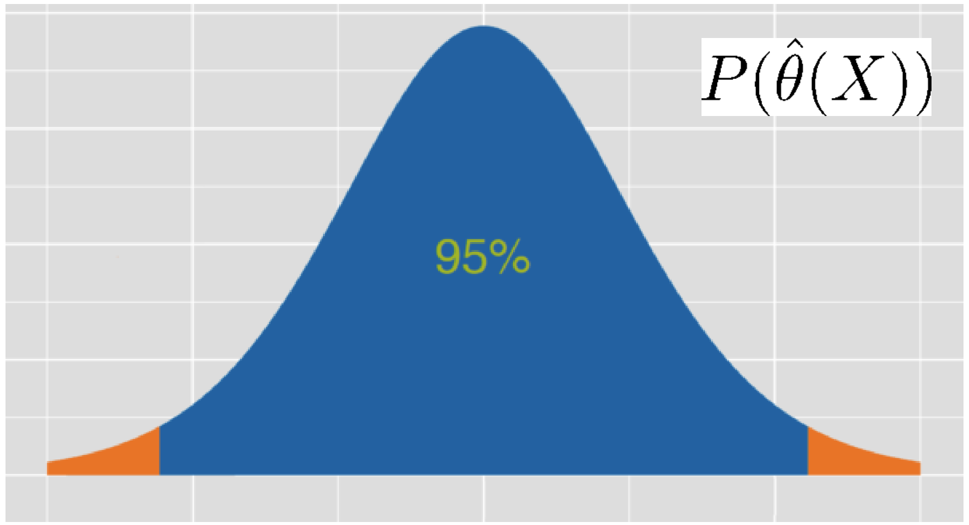
Confidence Intervals

Intuition Find an interval such that we are *pretty sure* it encompasses the true parameter value (e.g. algorithm accuracy).

Given data $X_1, \ldots, X_n$ and confidence $\alpha \in (0,1)$ find interval (a,b) such that,

$$P(\theta \in (a,b)) \geq 1 - \alpha$$

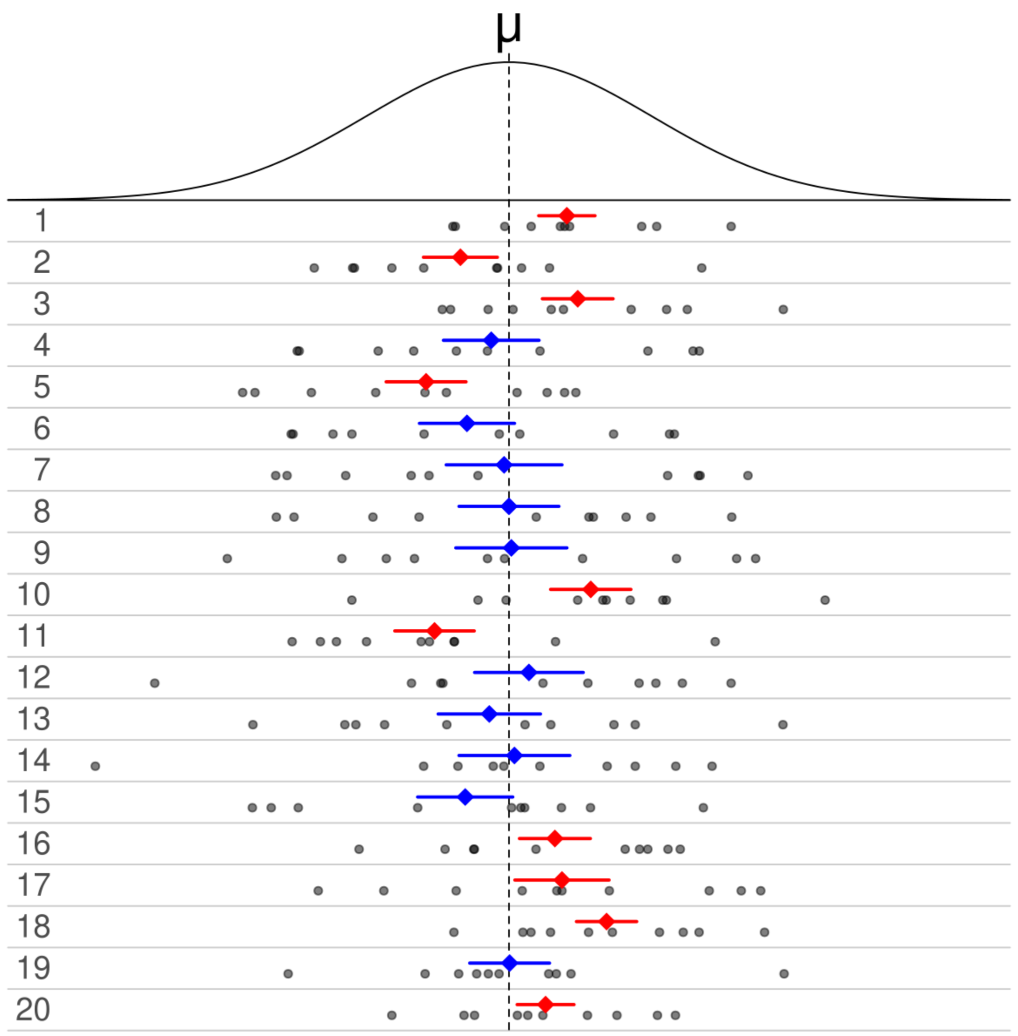
In English the interval (a,b) contains the true parameter value θ with probability **at least** $1 - \alpha$



- Intervals must be computed from data $a(X_1, \ldots, X_n)$ and $b(X_1, \ldots, X_n)$
- Interval (a,b) is **random**, parameter θ is **not random** (it is fixed)
- Requires that we know the distribution of the estimator $\hat{\theta}$

Knowledge Check

What is the confidence level of this estimator?



CI construction

A standard recipe:

- Construct an estimator for θ based on S -- call it $\hat{\theta}_S$
- Let $I(S) \coloneqq [\hat{\theta}_S - w, \hat{\theta}_S + w]$, where w is chosen such that for any θ,

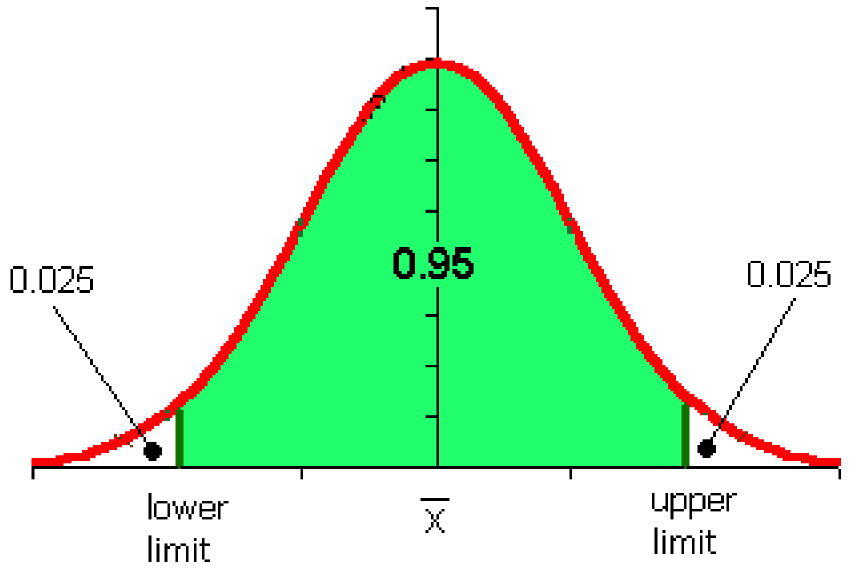
$$P_{S\sim D_\theta^n}\left(\theta \in [\hat{\theta}_S - w, \hat{\theta}_S + w]\right) \geq 1 - \alpha$$

Important example: confidence interval for normal mean

- $D_\mu = N(\mu, 1), S = (X_1, \dots, X_n) \sim D_\mu^n$
- Define $\hat{\mu}_S = \frac{1}{n}\sum_{i=1}^n X_i$
- $\hat{\mu}_S - \mu \sim N\left(0, \frac{1}{n}\right)$
- How to choose w such that $P(|\hat{\mu}_S - \mu| \leq w) \geq 1 - \alpha$?

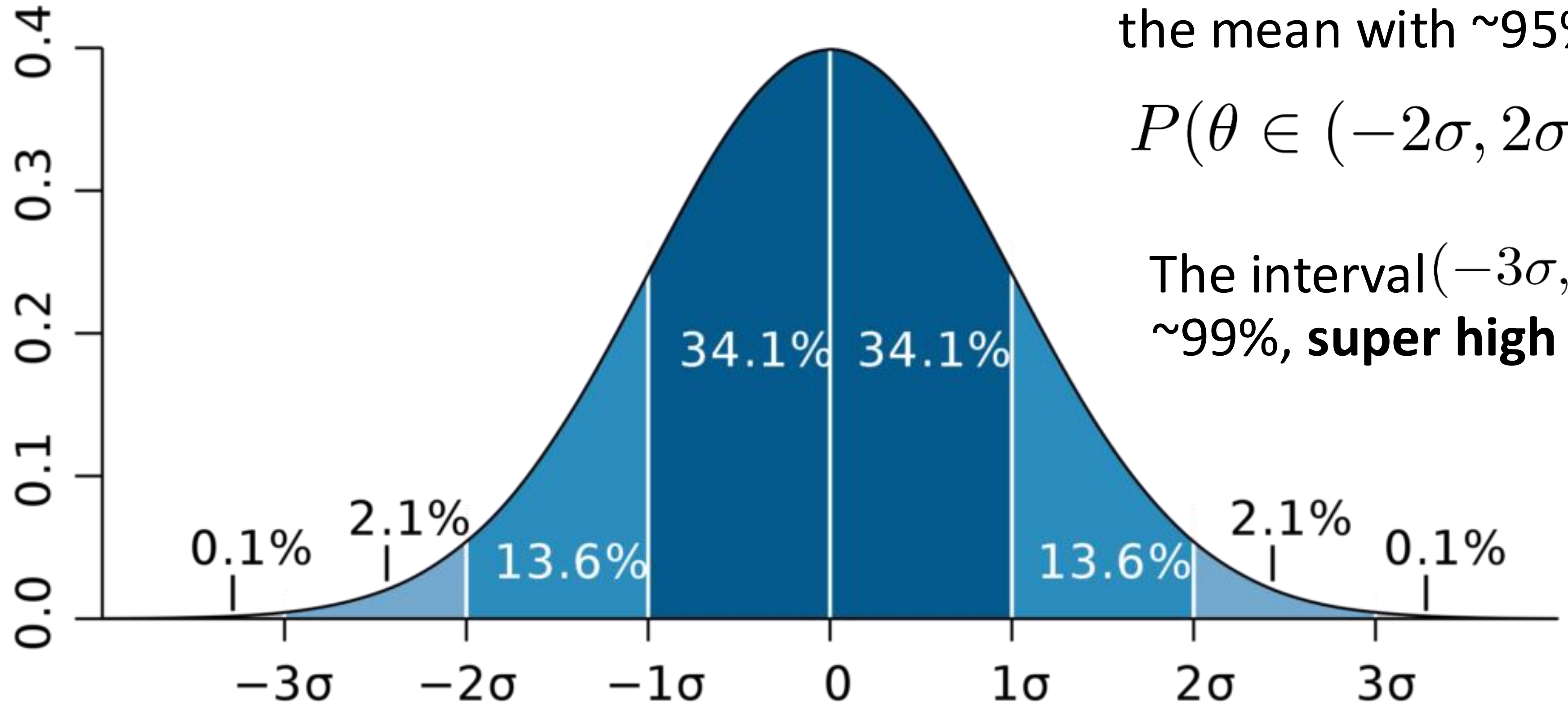
Known variance

Central limit theorem



Confidence Intervals of the Normal Distribution

Given enough data many estimators follow a Normal distribution (central limit theorem)



A Normal RV falls within 2σ of the mean with ~95% probability

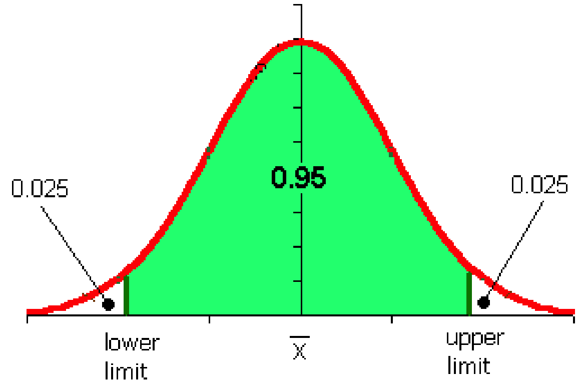
$$P(\theta \in (-2\sigma, 2\sigma)) \geq 0.95$$

The interval $(-3\sigma, 3\sigma)$ covers ~99%, **super high confidence**

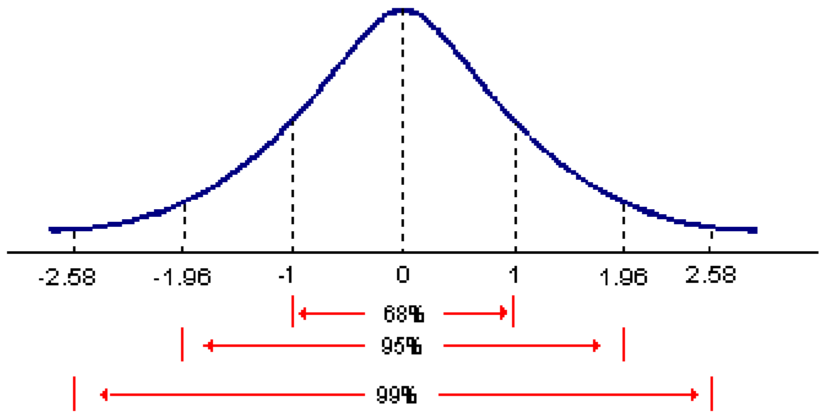
For various reasons, 95% has become standard confidence level

CI for normal mean (cont'd)

- $\hat{\mu}_S - \mu \sim N\left(0, \frac{1}{n}\right)$ Central limit theorem
- How to choose w such that $P(|\hat{\mu}_S - \mu| \le w) \ge 1 - \alpha$?



- Note: $Z = \sqrt{n}\,(\hat{\mu}_S - \mu) \sim N(0,1)$
- Suffices to find z_α such that $P(|Z| \le z_\alpha) \ge 1 - \alpha$, and let $w = \frac{z_\alpha}{\sqrt{n}}$



- Final $(1-\alpha)$-confidence interval construction for μ: $I(S) = \left[\hat{\mu}_S - \frac{z_\alpha}{\sqrt{n}}, \hat{\mu}_S + \frac{z_\alpha}{\sqrt{n}}\right]$
- E.g. 95%-confidence interval for μ: $I(S) = \left[\hat{\mu}_S - \frac{1.96}{\sqrt{n}}, \hat{\mu}_S + \frac{1.96}{\sqrt{n}}\right]$

CI for means of general distributions, *unknown* variance

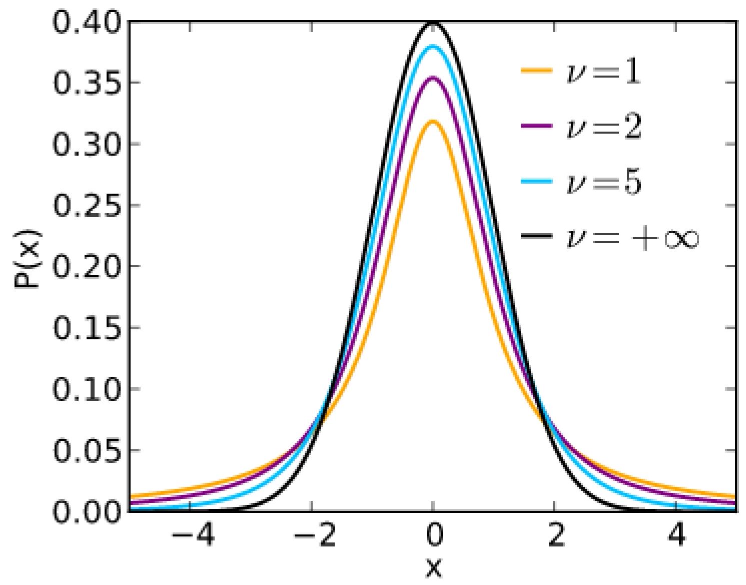
- Given D_θ with mean parameter θ with *unknown* variance
- $\hat{\sigma}_n^2 := \frac{\sum_{i=1}^n (X_i - \hat{\mu}_n)^2}{n-1} \Longrightarrow$ unbiased estimator of $\mathrm{var}(D_\theta)$
- *Theorem:* Let $X_1, \dots, X_n \sim N(\mu, \sigma^2)$, and $\hat{\mu}_n := \frac{1}{n} \sum_{i=1}^n X_i$

$$\sqrt{n}\frac{\hat{\mu}_n - \mu}{\hat{\sigma}_n} \sim \text{student-t (mean 0, scale 1, degrees of freedom} = n - 1)$$

- CI: $\left[\hat{\mu}_n \pm \frac{\hat{\sigma}_n \cdot t_\alpha}{\sqrt{n}}\right]$

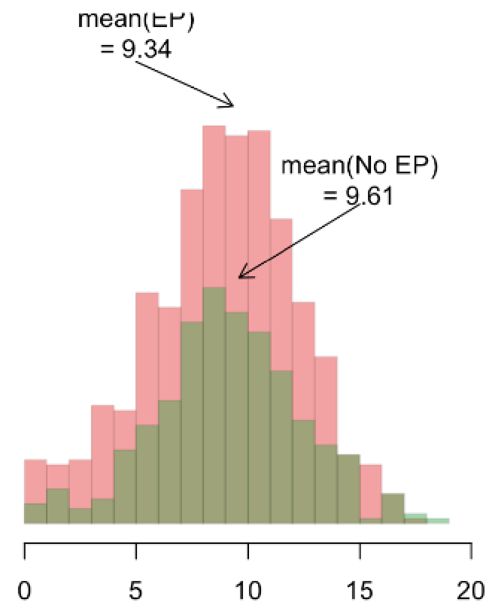
```
import scipy.stats as st
alpha = 0.05
st.t.ppf(1-alpha/2,df=2)
=> 4.302652729911275

st.t.ppf(1-alpha/2,df=5)
=> 2.5705818366147395

st.t.ppf(1-alpha/2,df=10)
=> 2.2281388519649385

st.t.ppf(1-alpha/2,df=30)
=> 2.0422724563012373

st.t.ppf(1-alpha/2,df=100)
=> 1.9839715184496334
```

How do we estimate variance of algorithm performance?



Two-sample hypothesis testing: definition

- Given D_θ with parameter θ
- Samples $S_X = (X_1, \dots, X_n)$ and $S_Y = (Y_1, \dots, Y_n)$ drawn iid from distribution D_{θ_X} and D_{θ_Y}, respectively



- Equality test version:
  - Null hypothesis $H_0\colon \theta_X = \theta_Y$
  - Alternative hypothesis $H_1\colon \theta_X \neq \theta_Y$

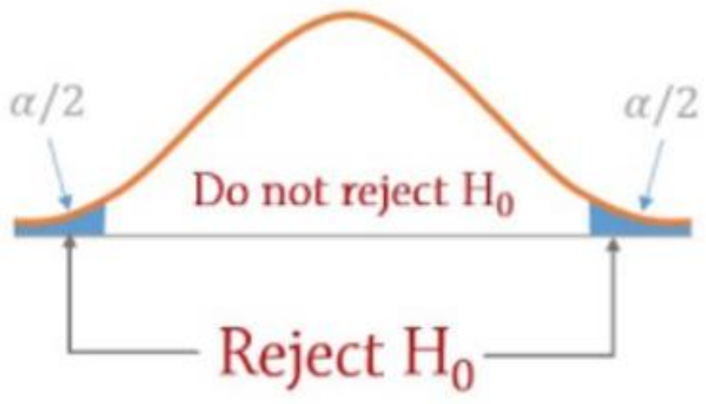
- E.g. $D_\mu = \mathrm{Ber}(\mu)$, $H_0\colon \mu_X = \mu_Y$
- Design hypothesis tester T such that the two types of errors are controlled

Paired t-test

- $S_X = (X_1, \dots, X_n)$ and $S_Y = (Y_1, \dots, Y_n)$ drawn iid from distribution $D_{\theta_X} = N(\mu_X, \sigma_X^2)$ and $D_{\theta_Y} = N(\mu_Y, \sigma_Y^2)$, respectively
  - $H_0: \mu_X = \mu_Y$
  - $H_1: \mu_X \neq \mu_Y$

- Let $\delta_i := X_i - Y_i$, for all $i = 1, \dots, n$
- Let $\bar{\delta}_n := \frac{1}{n}\sum_{i=1}^{n} \delta_i$

- Design hypothesis test T so that $P_{H_0}(T(S) = 0) \geq 1 - \alpha$
- Intuition: reasonable to reject if $|\bar{\delta}_n|$ is large

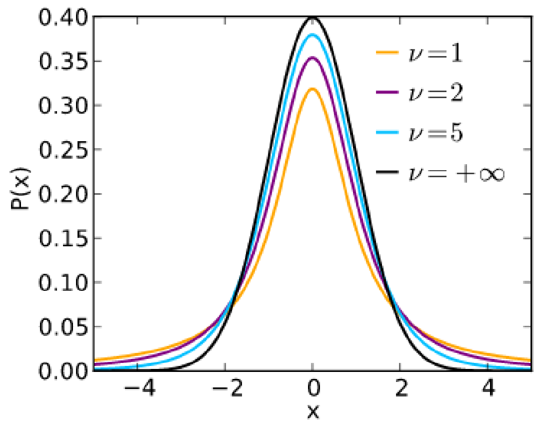


Paired t-test

- Under H_0, $\delta_i \sim N(0, \sigma^2), i = 1, \dots, n$, where $\sigma^2 = \sigma_X^2 + \sigma_Y^2$

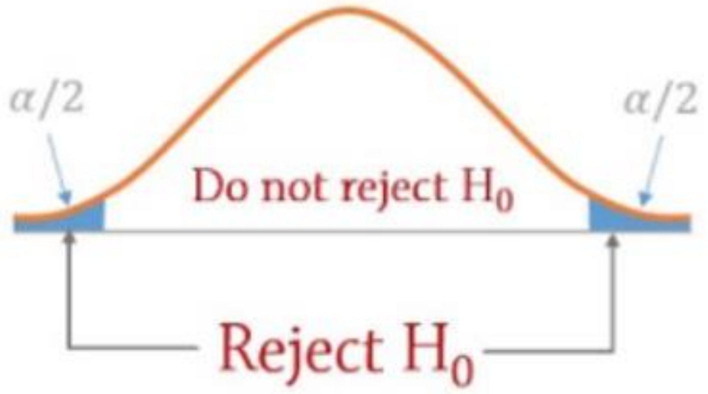
- Recall Thm: Let $\delta_1, \dots, \delta_n \sim N(0, \sigma^2)$, and $\bar{\delta}_n := \frac{1}{n}\sum_{i=1}^n \delta_i$, $\hat{\sigma}_n^2 := \frac{\sum_{i=1}^n (\delta_i - \bar{\delta}_n)^2}{n-1}$

$$Z = \sqrt{n}\frac{\bar{\delta}_n}{\hat{\sigma}_n} \sim \text{student-t (mean 0, scale 1, degrees of freedom} = n-1)$$



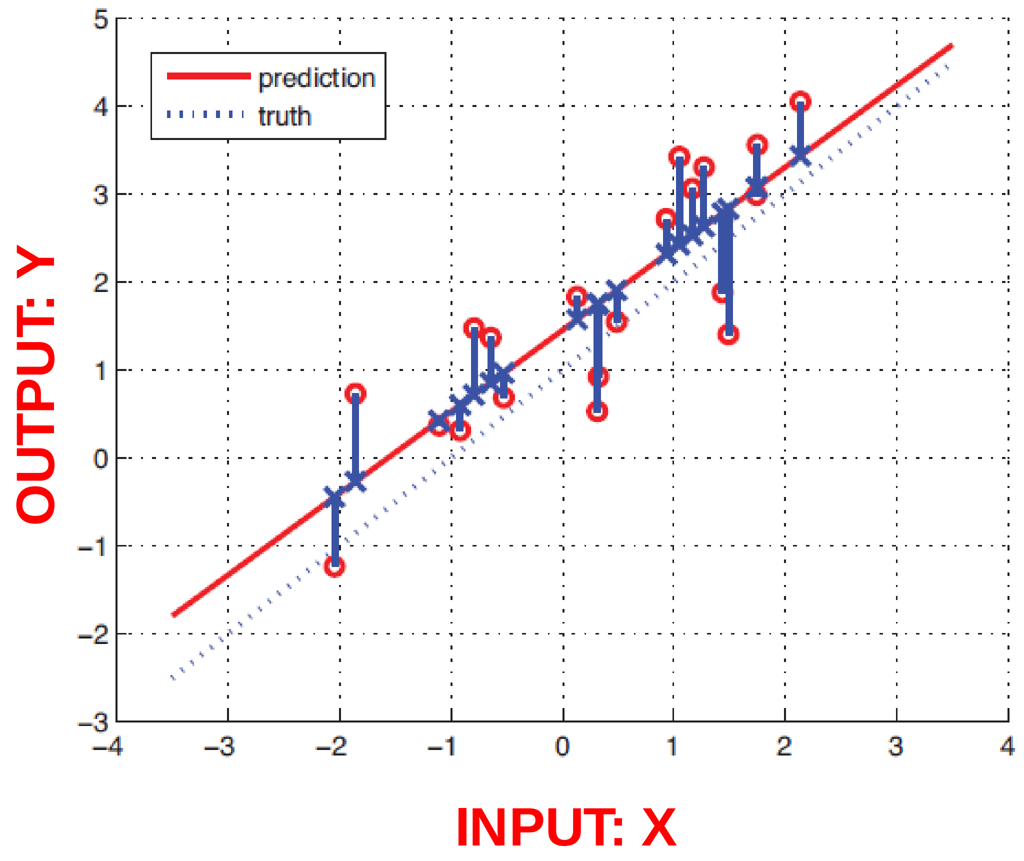
- Let's ask "under H_0, what is a plausible range of values of Z with failure rate $\alpha = 0.05$?"
  - Find the 0.025, 0.975-quantiles of Z => $t_{0.025}, t_{0.975}$
  - Hypothesis tester

$$T(S) = I(Z \notin [t_{0.025}, t_{0.975}]) = I\left(\sqrt{n}\frac{\bar{\delta}_n}{\hat{\sigma}_n} \notin [t_{0.025}, t_{0.975}]\right)$$



Linear Models

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

Outputs y are real-valued

Linear Regression As the name suggests, uses a *linear function*:

$$y = w^T x + b$$

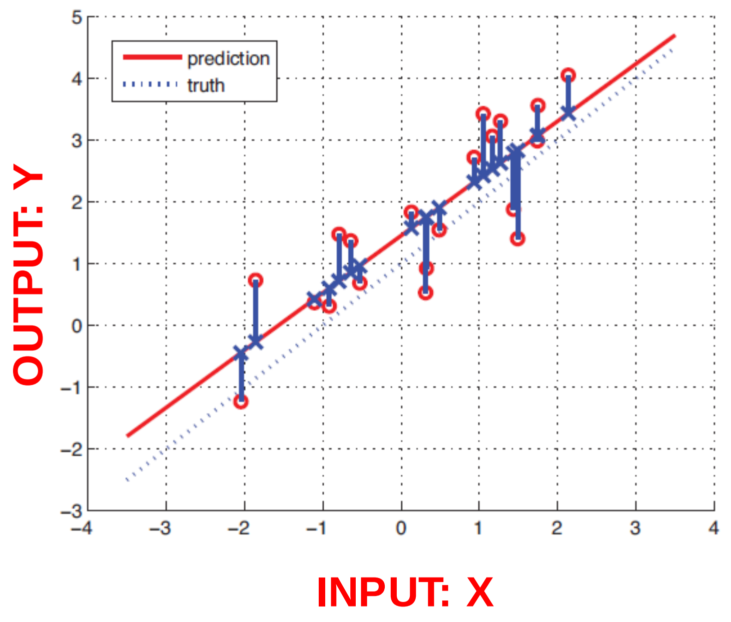
We will add noise later…

Linear Regression

Input-output mapping is not exact, so we will add zero-mean Gaussian noise,

Multivariate Normal (uncorrelated)

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$



This is equivalent to the likelihood function,

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2)$$

Because Adding a constant to a Normal RV is still a Normal RV,

$$z \sim \mathcal{N}(m, P) \qquad z + c \sim \mathcal{N}(m + c, P)$$

In the case of linear regression $z \to \epsilon$ and $c \to w^T x$

Great, we're done right?

We need to fit it to data by learning the regression weights

How to do this? What makes *good* weights?

Data – We have this

$$y = w^T x + \epsilon$$

Random; Can't do anything about it

Don't know these; need to learn them

Learning Linear Regression Models

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

Learning Linear Regression Models

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

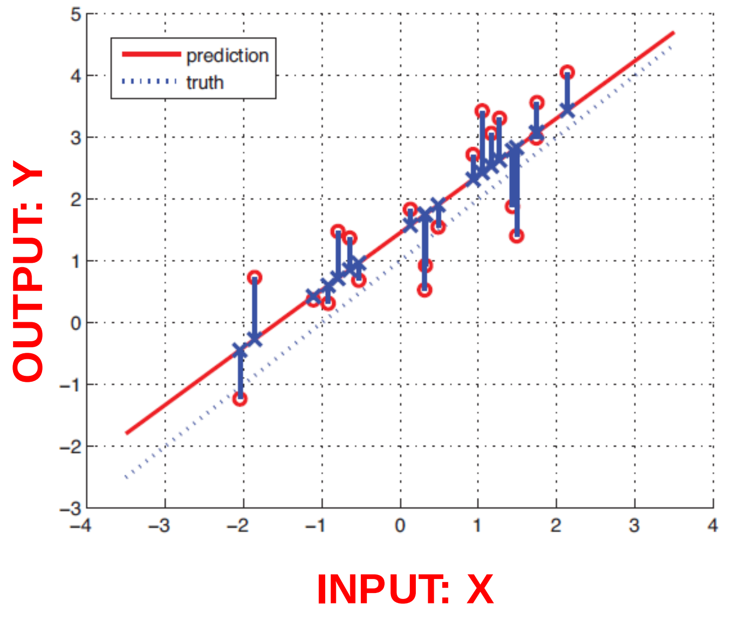
MLE for Linear Regression

Given training data $\{(x_i, y_i)\}_{i=1}^N$ likelihood function is given by,

$$\log \prod_{i=1}^N p(y_i \mid x_i, w) = \sum_{i=1}^N \log p(y_i \mid x_i, w)$$

Recall that the likelihood is Gaussian:

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2)$$



So MLE maximizes the log-likelihood over the whole data as,

$$w^{\text{MLE}} = \arg\max_w \sum_{i=1}^N \log \mathcal{N}(y_i \mid w^T x_i, \sigma^2)$$

MLE of Gaussian Mean

Assume data are i.i.d. univariate Gaussian,

Variance is known

$$p(\mathcal{Y} \mid \mu) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mu, \sigma^2)$$

Log-likelihood function:

$$\mathcal{L}(\mu) = \sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{1}{2}(y_i - \mu)^2 \sigma^{-2} \right) \right)$$

Constant doesn't depend on mean

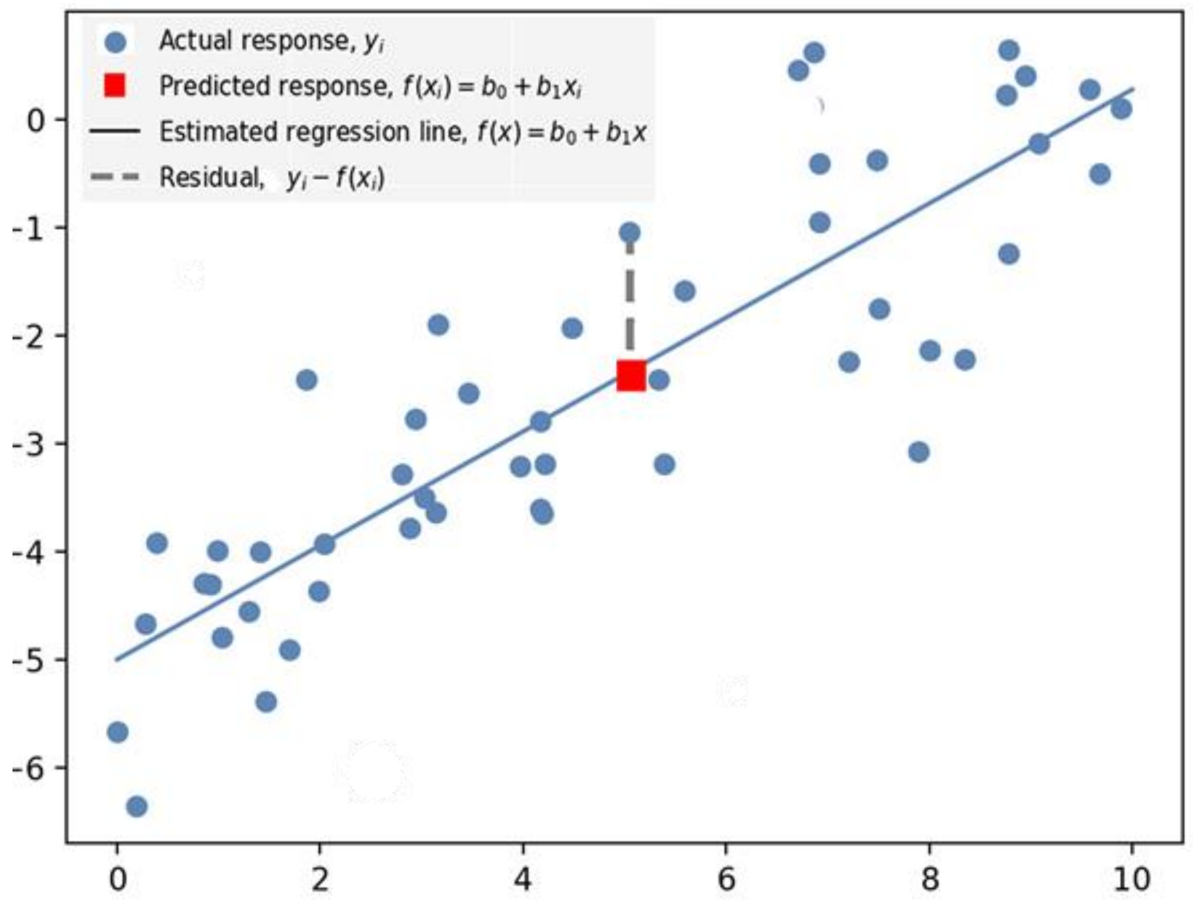
$$= \text{const.} - \frac{1}{2} \sum_{i=1}^{N} \left((y_i - \mu)^2 \sigma^{-2} \right)$$

MLE doesn't change when we:
1) Drop constant terms (in μ)
2) Minimize negative log-likelihood

MLE estimate is *least squares estimator*:

$$\mu^{\text{MLE}} = -\frac{1}{2\sigma^2} \arg\max_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2 = \arg\min_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2$$

MLE of Linear Regression



Substitute linear regression prediction into MLE solution and we have,

$$\min_w \sum_{i=1}^{N} (y_i - wx_i)^2$$

So for Linear Regression, MLE = Least Squares Estimation

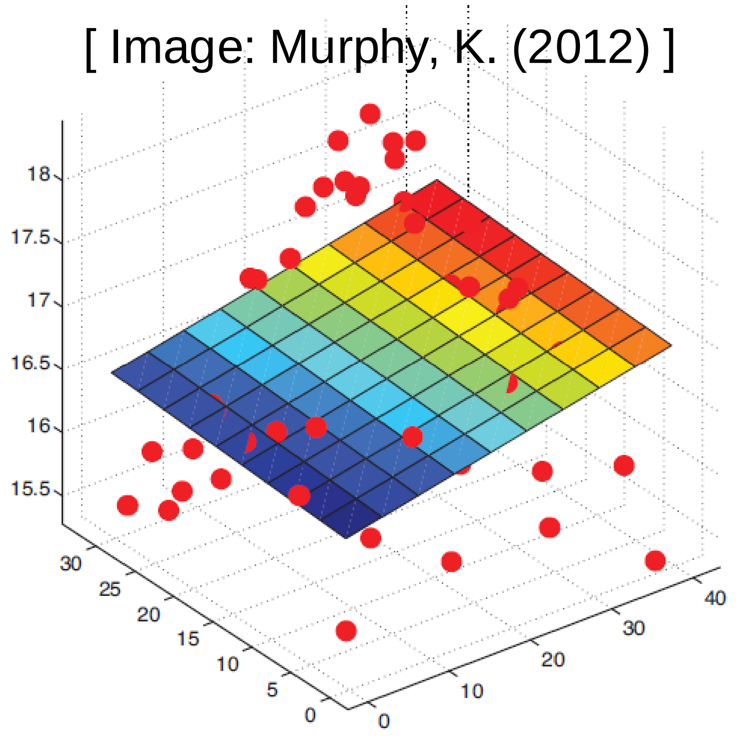
https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

MLE of Linear Regression

Using previous results, MLE is equivalent to minimizing squared residuals,

$$\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

[Image: Murphy, K. (2012)]



Some slightly more advanced linear algebra gives us a solution,

$$w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

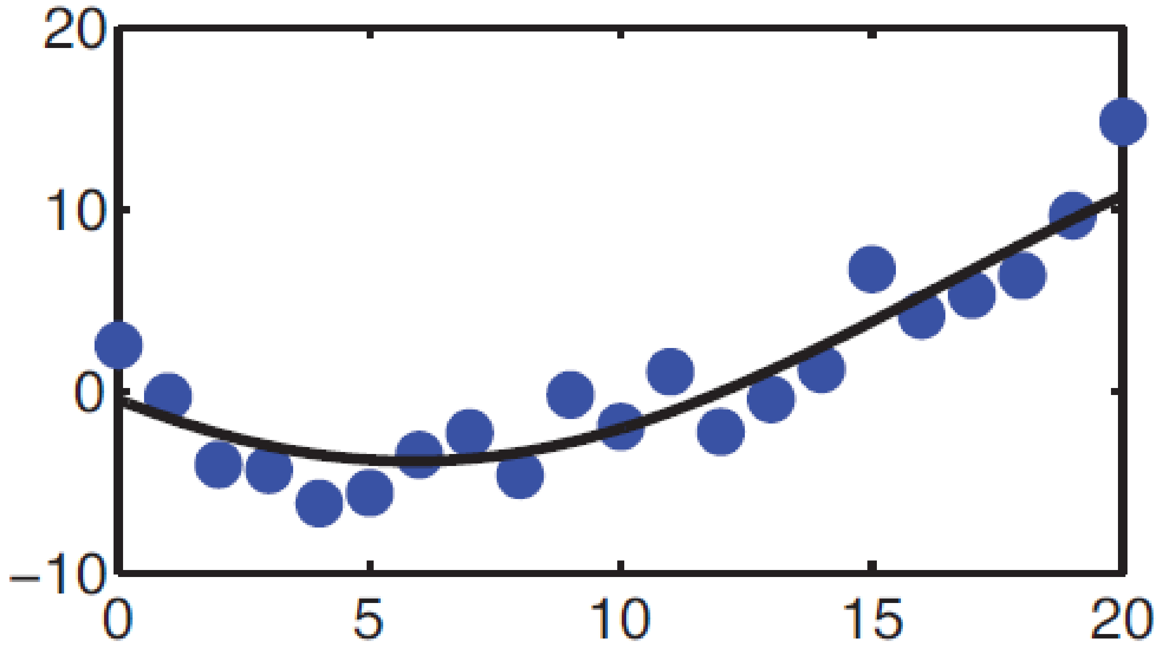
Ordinary Least Squares (OLS) solution

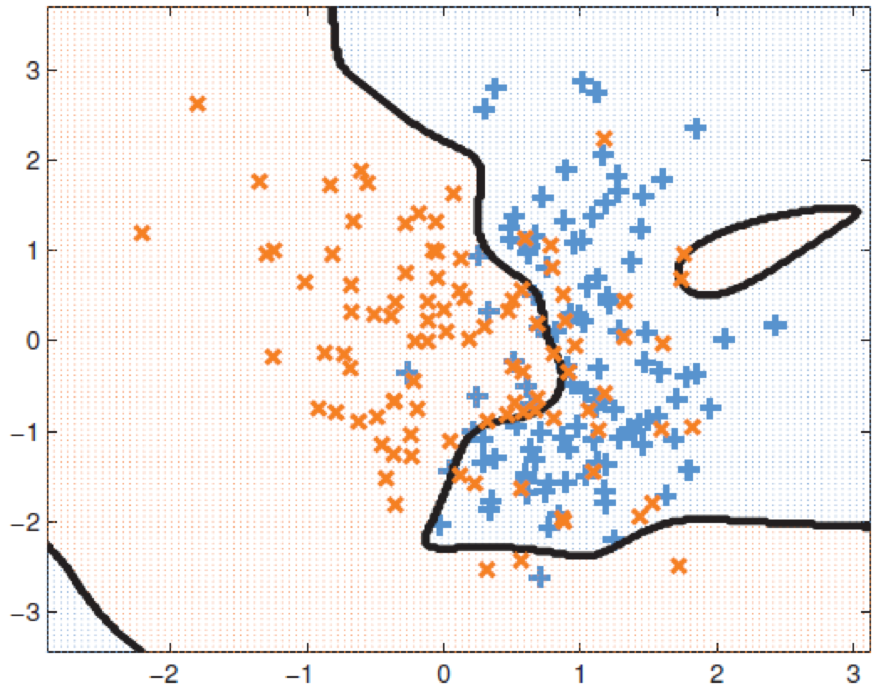
Derivation a bit involved for lecture but…

- We know it has a closed-form and why
- We can evaluate it
- Generally know where it comes from

Nonlinear Models

Nonlinear Data





What if our data are *not* well-described by a linear function?

What if classes are *not* *linearly-separable*?

[Source: Murphy, K. (2012)]

Basis Functions

- A **basis function** can be any function of the input features X
- Define a set of m basis functions $\phi_1(x), \ldots, \phi_m(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{i=1}^{m} w_i \phi_i(x) = w^T \phi(x)$$

- Regression model is *linear in the basis transformations*
- Model is *nonlinear in the data X*

Kernel Functions

*A **kernel function** is an inner-product of some basis function computed on two inputs*

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$

A consequence is that kernel functions are non-negative real-valued functions over a pair of inputs,

$$\kappa(x, x') \in \mathbb{R} \qquad \kappa(x, x') \geq 0$$

Kernel functions can be interpreted as a measure of distance between two inputs

Kernel Functions

Example The *linear basis* $\phi(x) = x$ produces the kernel,

$$\kappa(x, x') = \phi(x)^T \phi(x') = x^T x'$$

It is often easier to directly specify the kernel rather than the basis function…

Example Gaussian kernel models similarity according to an unnormalized Gaussian distribution,

$$\kappa(x, x') = \exp\left(-\frac{1}{2\sigma^2}(x - x')^2\right)$$

Note Despite the name, this is **not** a Gaussian probability density.

Also called a *radial basis function* (RBF)

Kernel Functions

Given *any* set of data $\{x_i\}_{i=1}^n$ a necessary and sufficient condition of a valid kernel function is that the nxn **gram matrix**,

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \dots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \dots & \kappa(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \dots & \kappa(x_n, x_n) \end{pmatrix}$$

Is a *symmetric positive semidefinite matrix.*

Kernel Ridge Regression

Kernel representation requires inversion of NxN matrix

Primal

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \dots & \phi_M(x_N) \end{pmatrix}$$

$$w = \underbrace{(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}}_{\substack{\textbf{MxM Matrix Inversion} \\ \mathbf{O(M^3)}}}\mathbf{\Phi}^T\mathbf{y}$$

Dual

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1,x_1) & \kappa(x_1,x_2) & \dots & \kappa(x_1,x_n) \\ \kappa(x_2,x_1) & \kappa(x_2,x_2) & \dots & \kappa(x_2,x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n,x_1) & \kappa(x_n,x_2) & \dots & \kappa(x_n,x_n) \end{pmatrix}$$

$$y(x) = \mathbf{k}(\mathbf{x})^T\underbrace{(\mathbf{K} + \lambda I)^{-1}}_{\substack{\textbf{NxN Matrix Inversion} \\ \mathbf{O(N^3)}}}\mathbf{y}$$

Number of training data N greater than basis functions M