# Final review

Aarti Singh

Machine Learning 10-315
Dec 2, 2019

## Machine Learning Tasks

- Supervised
  - Classification: Bayes optimal rule

    Naïve Bayes

    Logistic Regression

    Neural networks, Deep convolutional

    SVMs, kernels

    Decision tree

    Boosting

    k-NN
  - Regression: Bayes optimal rule

    Linear, regularized (ridge, lasso), kernelized

    Nonparametric kernel regression

## Machine Learning Tasks

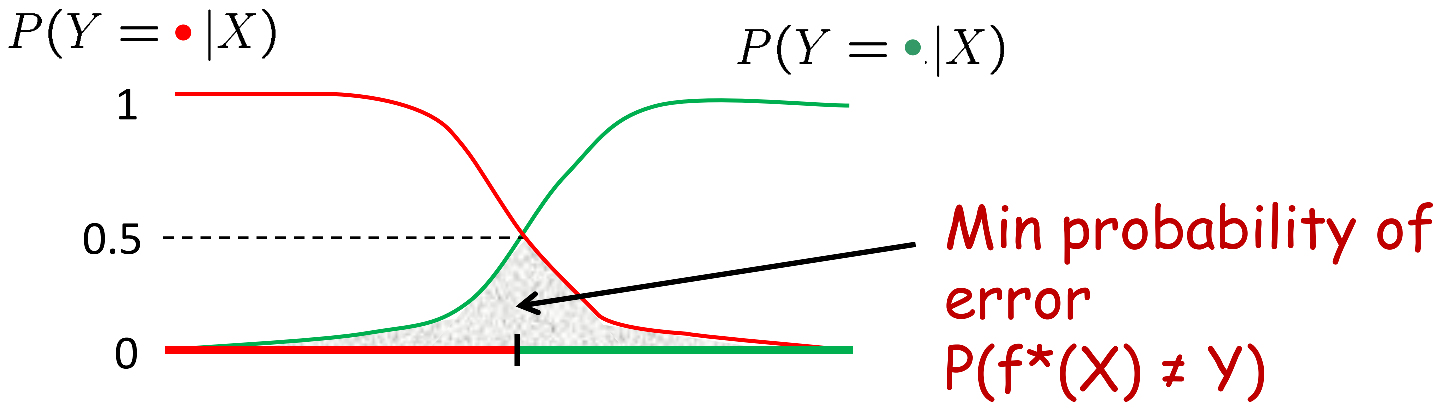
- Unsupervised:
  - Density estimation - MLE, MAP, nonparametric
  - Dimensionality reduction – PCA
  - Clustering – hierarchical, k-means, mixture models and EM algorithm

- Theory: PAC bounds (Haussler, Hoeffding, VC)

- General concepts: Overfitting, generalization, model selection (cross validation), bias-variance

# Bayes Optimal classifier

Optimal classifier: $f^*(x) = \arg\max_{Y=y} P(Y = y|X = x)$



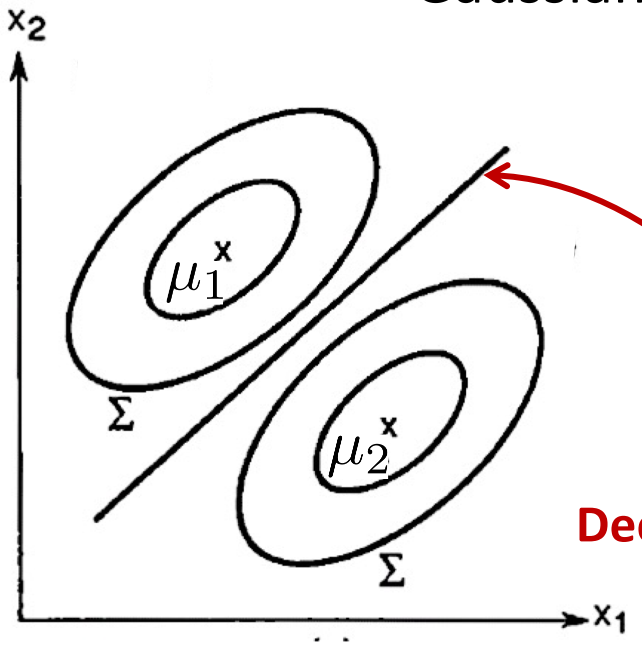
- Even the optimal classifier makes mistakes: min probability of error > 0

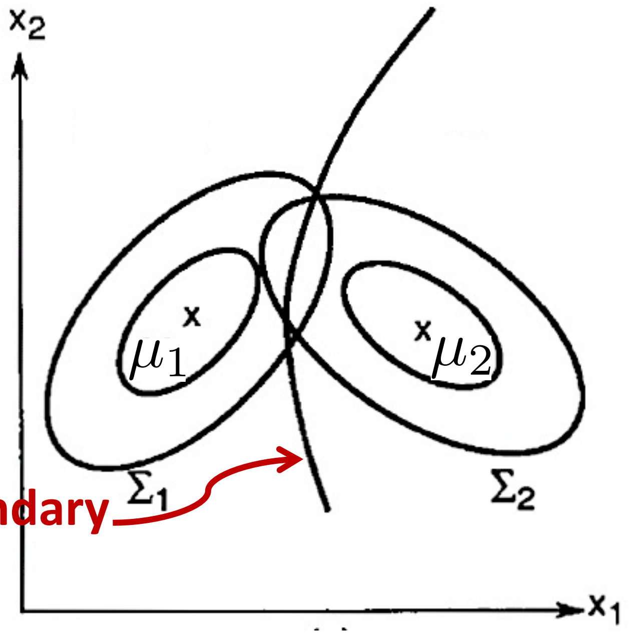
# d-dim Gaussian Bayes classifier

$$f^*(x) = \arg\max_{Y=y} \underbrace{P(X=x|Y=y)}_{\text{Class conditional density}} \underbrace{P(Y=y)}_{\text{Class probability}}$$

Class conditional density → Gaussian($\mu_y$,$\Sigma_y$)

Class probability → Bernoulli(θ)





Decision Boundary

# Naïve Bayes Classifier

- Bayes Classifier with additional "naïve" assumption:
  - Features are independent given class:

$$P(X_1 \ldots X_d | Y) = \prod_{i=1}^{d} P(X_i | Y)$$

$$\begin{aligned} f_{NB}(\mathbf{x}) &= \arg\max_y \; P(x_1, \ldots, x_d \mid y) P(y) \\ &= \arg\max_y \prod_{i=1}^{d} P(x_i | y) P(y) \end{aligned}$$

- Has fewer parameters, and hence requires fewer training data, even though assumption may be violated in practice

# MLE vs. MAP

- Maximum Likelihood estimation (MLE)

  Choose value that maximizes the probability of observed data

$$\widehat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta)$$

- Maximum *a posteriori* (MAP) estimation

  Choose value that is most probable given observed data and prior belief

$$\begin{aligned}\widehat{\theta}_{MAP} &= \arg\max_{\theta} P(\theta|D) \\ &= \arg\max_{\theta} P(D|\theta)P(\theta)\end{aligned}$$

When is MAP same as MLE?

# Logistic Regression

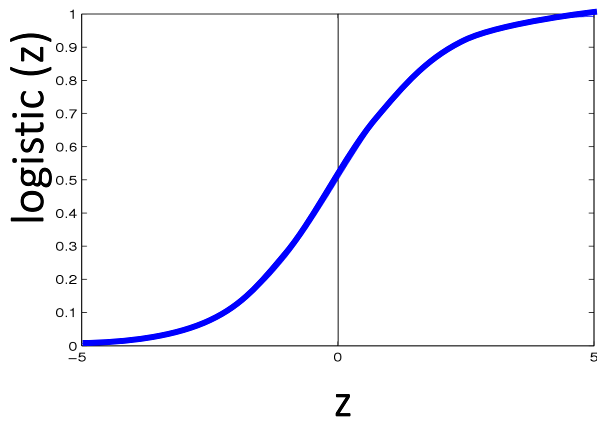
Not really regression

Assumes the following functional form for P(Y|X):

$$P(Y=0|X) = \frac{1}{1+\exp(w_0 + \sum_i w_i X_i)}$$

Logistic function applied to a linear function of the data

**Logistic function (or Sigmoid):** $\frac{1}{1+exp(-z)}$



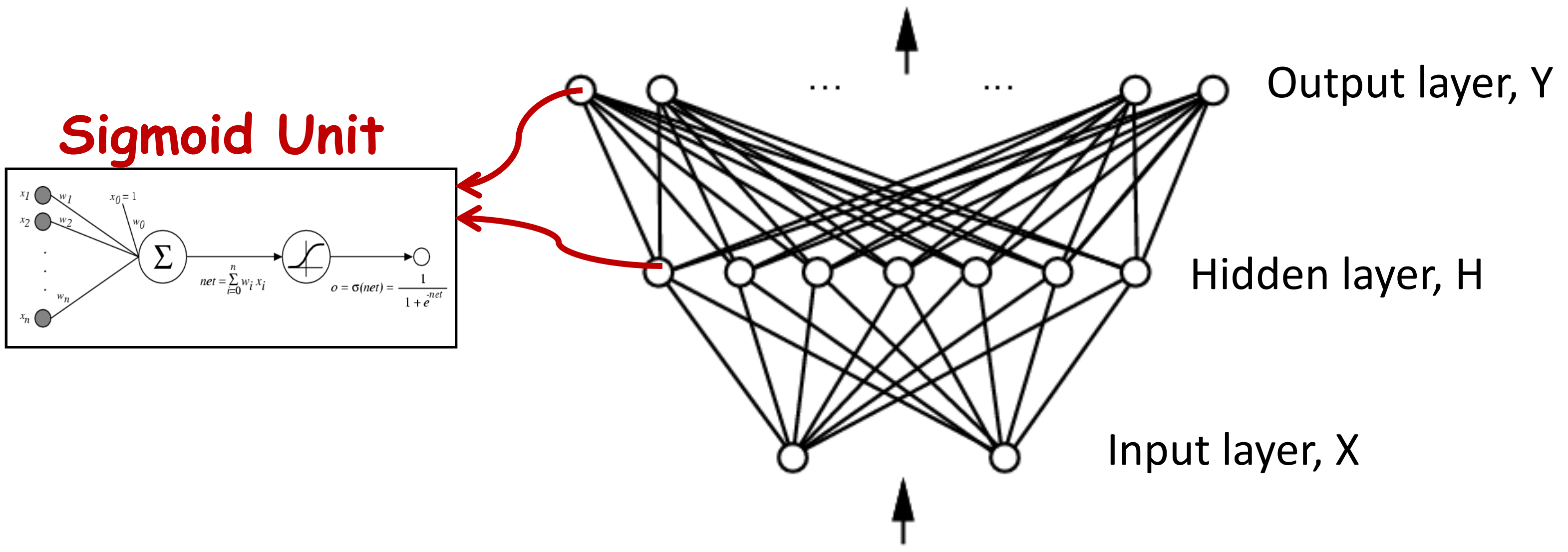
Training using maximum (conditional) likelihood – discriminative

$$\widehat{\mathbf{w}}_{MCLE} = \arg\max_{\mathbf{w}} \prod_{j=1}^{n} P(Y^{(j)} \mid X^{(j)}, \mathbf{w})$$

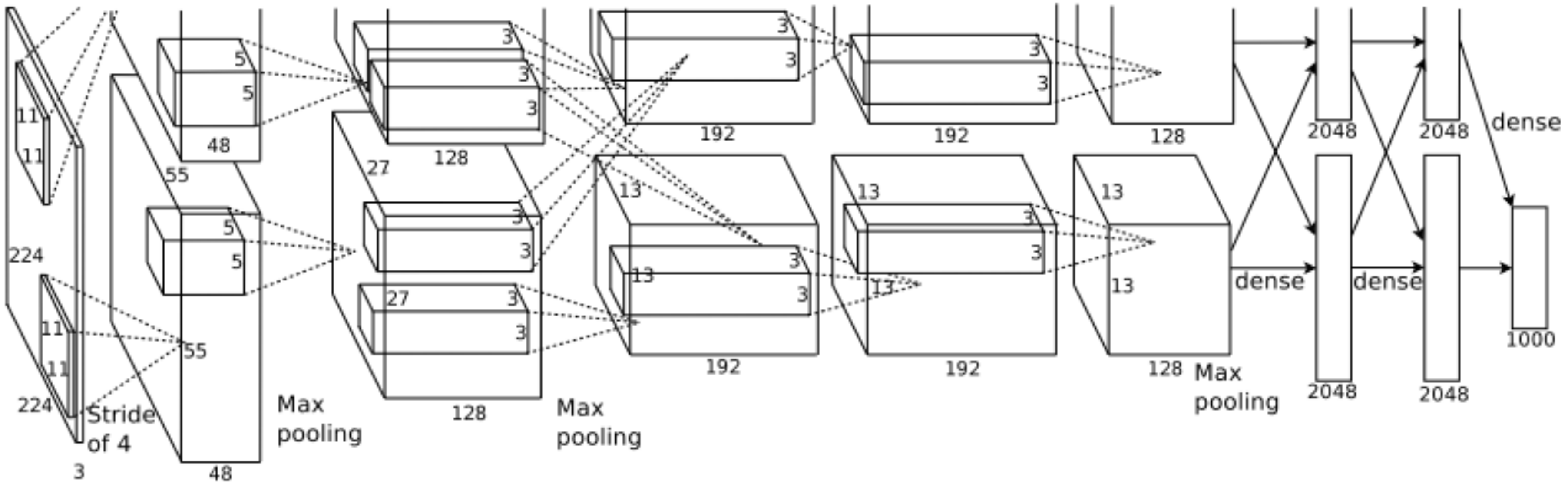
# Neural Networks to learn f: X → Y

- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

- Neural networks - Represent f by ***network*** of logistic/sigmoid units:

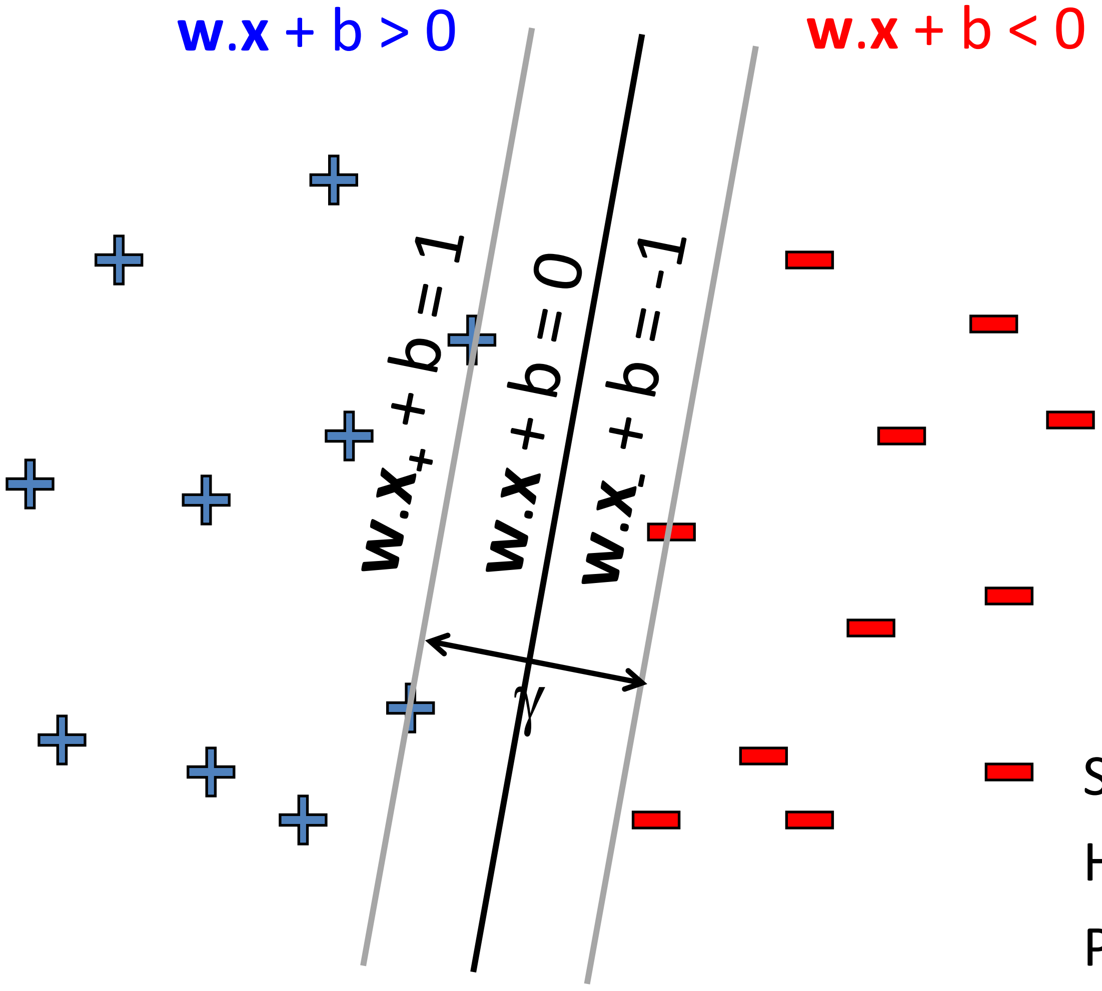


# Deep Convolutional Neural Networks



**Convolution layers (ReLU)**
**Max pooling layers** – nonlinear downsampling (max value of regions)
**Fully connected layers**
**Output softmax**

# Support Vector Machines



$$\min_{\mathbf{w},b} \ \mathbf{w.w}$$

$$\text{s.t. } (\mathbf{w.x}_j + b)\, y_j \geq 1 \quad \forall j$$

Solve efficiently by quadratic programming (QP)

Support vectors

Hard vs Soft

Primal vs Dual

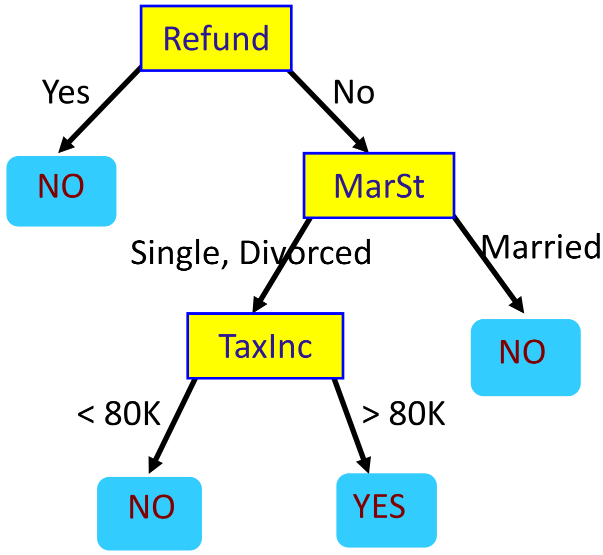
Kernel Trick

# Decision trees

- Top-down induction [ID3, C4.5, C5, …]

Main loop: C4.5

1. $X \leftarrow$ the "best" decision feature for next *node*
2. Assign $X$ as decision feature for *node*
3. For "best" split of $X$, create new descendants of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes
6. Prune back tree to reduce overfitting
7. Assign majority label to the leaf node



# AdaBoost [Freund & Schapire'95]

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$. Initially equal weights

For $t = 1, \ldots, T$:

- Train weak learner using distribution $D_t$. Naïve bayes, decision stump
- Get weak classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$. Magic (+ve)
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

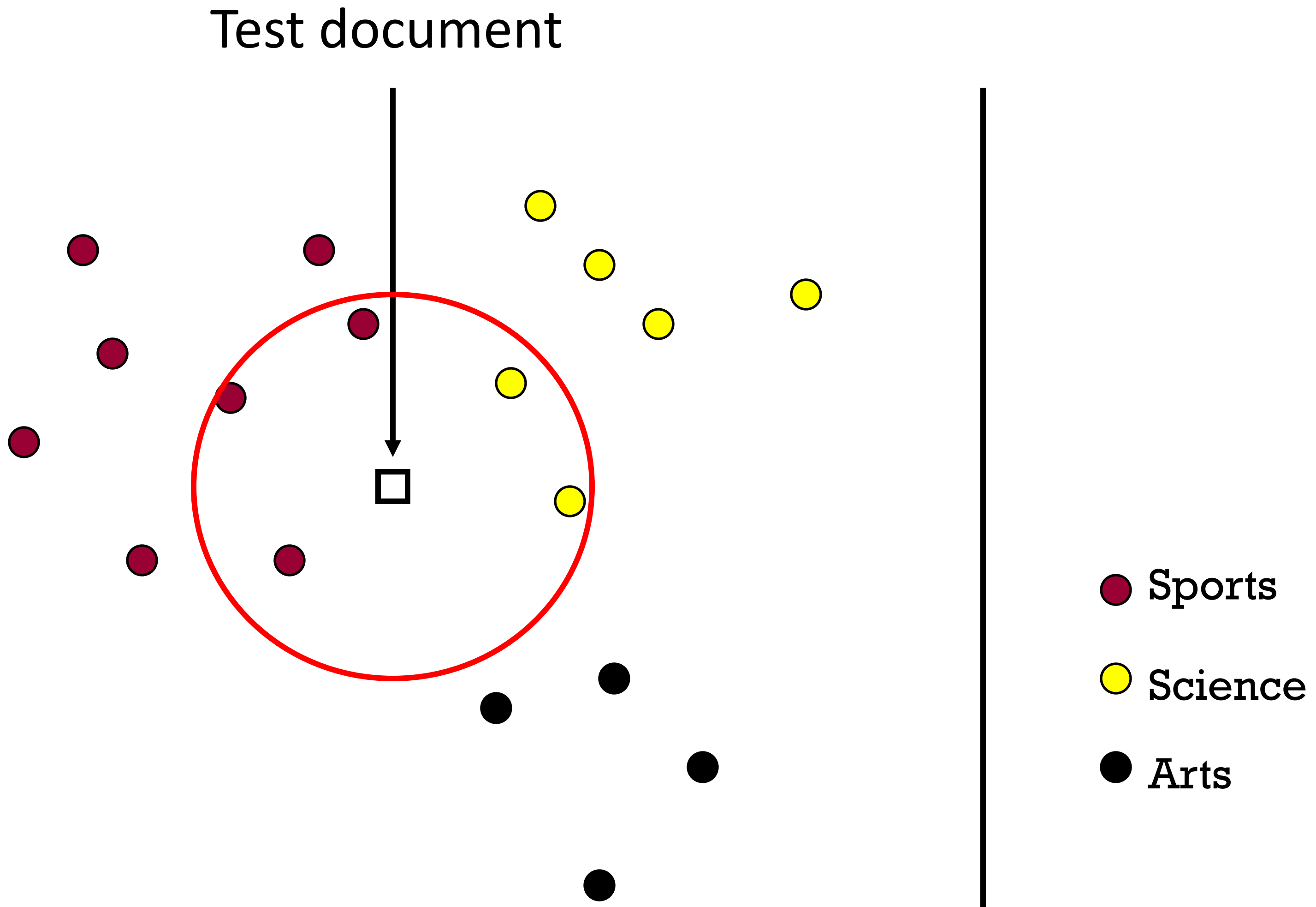
Increase weight if wrong on pt i $y_i h_t(x_i) = -1 < 0$

where $Z_t$ is a normalization factor

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

# k-NN classifier (k=5)



What should we predict? … Average? Majority? Why?

# Machine Learning Tasks

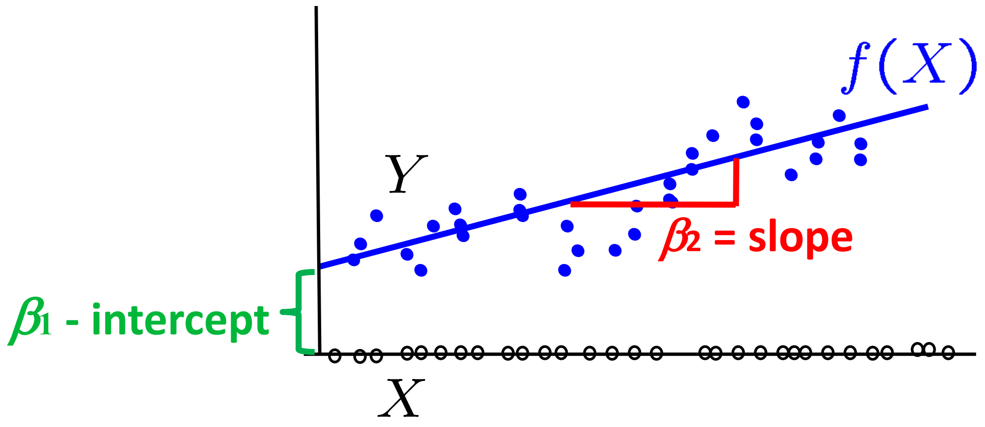
- Supervised
  - Classification: Bayes optimal rule
    - Naïve Bayes
    - Logistic Regression
    - Neural networks, Deep convolutional
    - SVMs, kernels
    - Decision tree
    - Boosting
    - k-NN
  - Regression: Bayes optimal rule
    - Linear, regularized (ridge, lasso), kernelized
    - Nonparametric kernel regression

# Linear Regression

$$\widehat{f}_n^L = \arg\min_{\boxed{f \in \mathcal{F}_L}} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$$

Least Squares Estimator

$\mathcal{F}_L$ - Class of Linear functions



Uni-variate case:

$$f(X) = \beta_1 + \beta_2 X$$

Multi-variate case:

$$f(X) = f(X^{(1)}, \dots, X^{(p)}) = \beta_1 \overset{1}{\cancel{X^{(1)}}} + \beta_2 X^{(2)} + \dots + \beta_p X^{(p)}$$

$$= X\beta \qquad \text{where} \quad X = [X^{(1)} \dots X^{(p)}], \quad \beta = [\beta_1 \dots \beta_p]^T$$

# Regularized Least Squares

What if $(\mathbf{A}^T\mathbf{A})$ is not invertible ?

r equations , p unknowns – underdetermined system of linear equations
many feasible solutions
Need to constrain solution further

e.g. bias solution to "small" values of β (small changes in input don't translate to large changes in output)

$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda||\beta||_2^2$$

Ridge Regression
(l2 penalty)

$$\lambda \geq 0$$

$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda||\beta||_1$$

Lasso
(l1 penalty)

Many β can be zero – many inputs are irrelevant to prediction in high-dimensional settings (typically intercept term not penalized)

# Kernelized ridge regression

$$\widehat{\beta} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{Y} \qquad \widehat{f}_n(X) = \mathbf{X}\widehat{\beta}$$

Using dual, can re-write solution as:

$$\widehat{\beta} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda\mathbf{I})^{-1}\mathbf{Y}$$

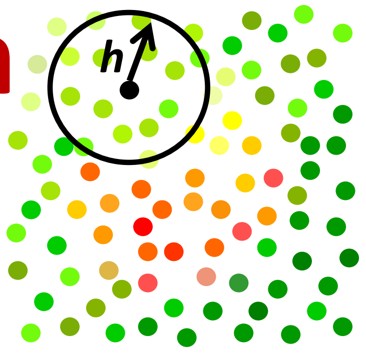
How does this help?

- Only need to invert n x n matrix (instead of p x p or m x m)
- More importantly, kernel trick!

$$\widehat{f}_n(X) = \mathbf{K}_X(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y} \quad \text{where} \quad \begin{aligned}\mathbf{K}_X(i) &= \boldsymbol{\phi}(X) \cdot \boldsymbol{\phi}(X_i) \\ \mathbf{K}(i,j) &= \boldsymbol{\phi}(X_i) \cdot \boldsymbol{\phi}(X_j)\end{aligned}$$

Work with kernels, never need to write out the high-dim vectors

# Local Kernel Regression



- Nonparametric estimator akin to kNN
- Nadaraya-Watson Kernel Estimator

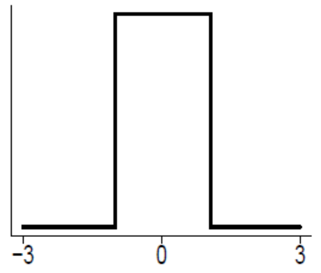
$$\widehat{f}_n(X) = \sum_{i=1}^{n} w_i Y_i \quad \text{Where} \quad w_i(X) = \frac{K\left(\frac{X-X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{X-X_i}{h}\right)}$$

- Weight each training point based on distance to test point
- Boxcar kernel yields

  local average

boxcar kernel :

$$K(x) = \frac{1}{2} I(x),$$



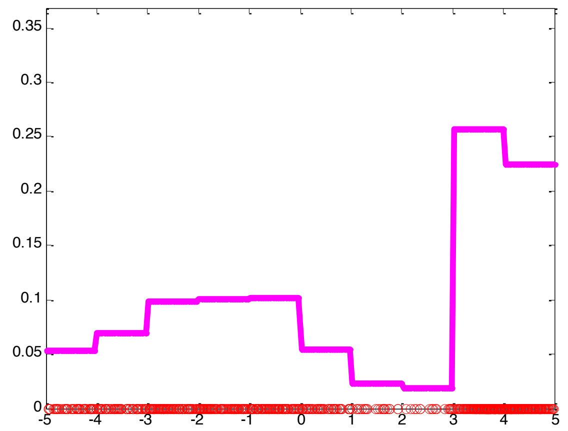
# Machine Learning Tasks

- Unsupervised:
  - Density estimation - MLE, MAP, nonparametric
  - Dimensionality reduction – PCA
  - Clustering – hierarchical, k-means, mixture models and EM algorithm
- Theory: PAC bounds (Haussler, Hoeffding, VC)
- General concepts: Overfitting, generalization, model selection (cross validation), bias-variance

# Kernel density estimate

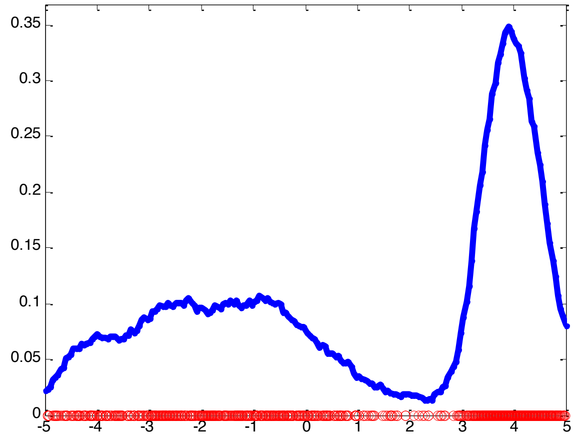
- Histogram – blocky estimate

$$\widehat{p}(x) = \frac{1}{\Delta} \frac{\sum_{j=1}^{n} \mathbf{1}_{X_j \in \mathrm{Bin}_x}}{n}$$

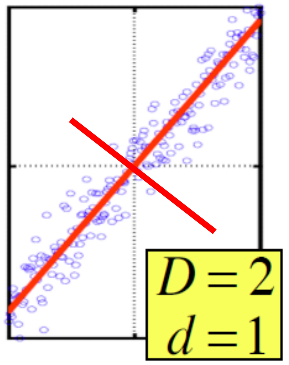


- Kernel density estimate aka "Parzen/moving window method"

$$\widehat{p}(x) = \frac{1}{\Delta} \frac{\sum_{j=1}^{n} \mathbf{1}_{||X_j - x|| \leq \Delta}}{n}$$



# Principal Component Analysis (PCA)



$$(\mathbf{X}\mathbf{X}^T)\mathbf{v} = \lambda\mathbf{v}$$

**Therefore, v is the eigenvector of sample covariance matrix $XX^T$**

Sample variance of projection = $\mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v}^T\mathbf{v} = \lambda$

**Thus, the eigenvalue λ denotes the amount of variability captured along that dimension (aka amount of energy along that dimension).**

Eigenvalues $\lambda_1 > \lambda_2 > \lambda_3 > \ldots$

The 1st Principal component $v_1$ is the eigenvector of the sample covariance matrix $XX^T$ associated with the largest eigenvalue $\lambda_1$

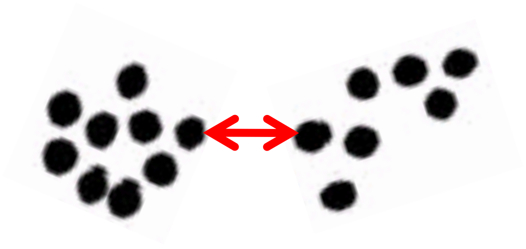
The 2nd Principal component $v_2$ is the eigenvector of the sample covariance matrix $XX^T$ associated with the second largest eigenvalue $\lambda_2$

And so on …

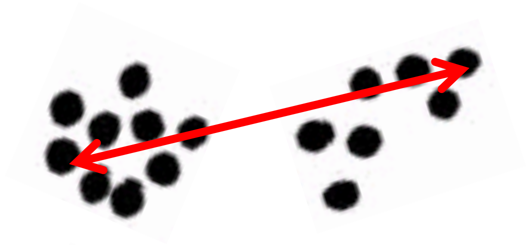
# Bottom-up Agglomerative clustering

Different algorithms differ in how the similarities are defined (and hence updated) between two clusters

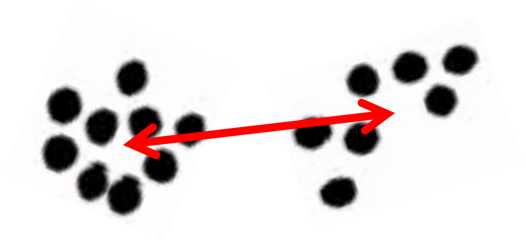
- Single-Linkage
  - Nearest Neighbor: similarity between their closest members.



- Complete-Linkage
  - Furthest Neighbor: similarity between their furthest members.



- Centroid
  - Similarity between the centers of gravity



- Average-Linkage
  - Average similarity of all cross-cluster pairs.

## K-Means

### Algorithm

Input – Desired number of clusters, *k*

Initialize – the *k* cluster centers (randomly if necessary)

Iterate –

1. Assign points to the nearest cluster centers
2. Re-estimate the *k* cluster centers (aka the centroid or mean), by assuming the memberships found above are correct.

$$\vec{\mu}_k = \frac{1}{\mathcal{C}_k} \sum_{i \in \mathcal{C}_k} \vec{x}_i$$

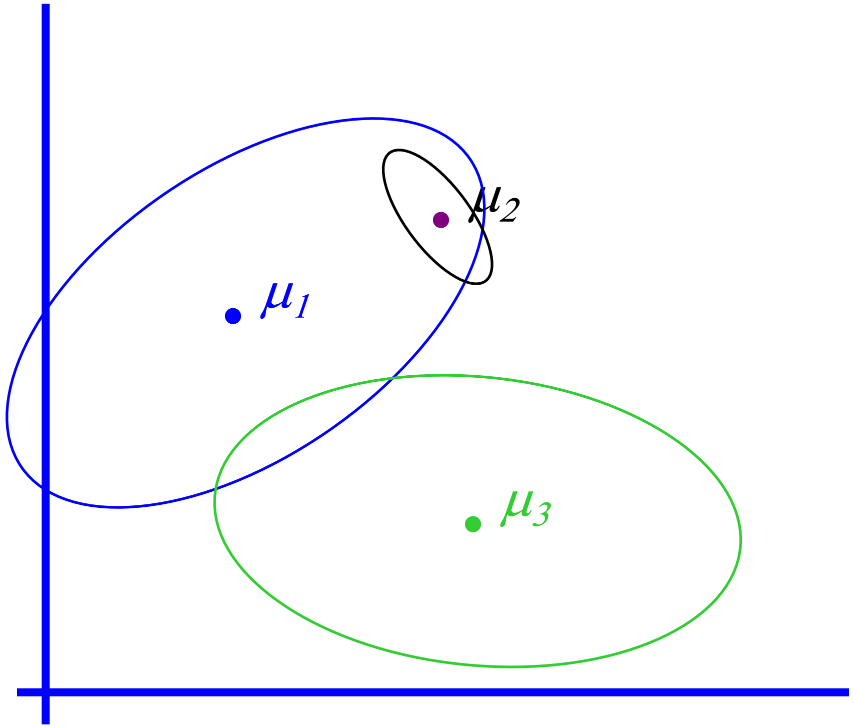
Termination –

If none of the objects changed membership in the last iteration, exit. Otherwise go to 1.

# General GMM

GMM – Gaussian Mixture Model (Multi-modal distribution)

- There are k components
- Component *i* has an associated mean vector $\mu_i$
- Each component generates data from a Gaussian with mean $\mu_i$ and covariance matrix $\Sigma_i$



Each data point is generated according to the following recipe:

1) Pick a component at random: Choose component i with probability *P(y=i)*
2) Datapoint x $\sim \mathrm{N}(\mu_i, \Sigma_i)$

# EM for general GMMs

Iterate. On iteration t let our estimates be

$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \ldots \mu_k^{(t)}, \Sigma_1^{(t)}, \Sigma_2^{(t)} \ldots \Sigma_k^{(t)}, p_1^{(t)}, p_2^{(t)} \ldots p_k^{(t)} \}$

$p_i^{(t)}$ is shorthand for estimate of $P(y=i)$ on t'th iteration

**E-step**

Compute "expected" classes of all datapoints for each class

$$P(y=i \mid x_j, \lambda_t) \propto p_i^{(t)} p(x_j \mid \mu_i^{(t)}, \Sigma_i^{(t)})$$

*Just evaluate a Gaussian at $x_j$*

**M-step**

Compute MLEs given our data's class membership distributions (weights)

$$\mu_i^{(t+1)} = \frac{\sum_j P(y=i \mid x_j, \lambda_t) x_j}{\sum_j P(y=i \mid x_j, \lambda_t)} \qquad \Sigma_i^{(t+1)} = \frac{\sum_j P(y=i \mid x_j, \lambda_t)\left(x_j - \mu_i^{(t+1)}\right)\left(x_j - \mu_i^{(t+1)}\right)^T}{\sum_j P(y=i \mid x_j, \lambda_t)}$$

$$p_i^{(t+1)} = \frac{\sum_j P(y=i \mid x_j, \lambda_t)}{m}$$

$m$ = #data points

# Summary of PAC bounds

With probability ≥ 1-$\delta$,

1) for all h $\in$ H s.t. $\text{error}_{\text{train}}(h) = 0$,

$$\text{error}_{\text{true}}(h) \leq \varepsilon = \frac{\ln|H| + \ln\frac{1}{\delta}}{m}$$

2) for all h $\in$ H,

$$|\text{error}_{\text{true}}(h) - \text{error}_{\text{train}}(h)| \leq \varepsilon = \sqrt{\frac{\ln|H| + \ln\frac{1}{\delta}}{2m}}$$

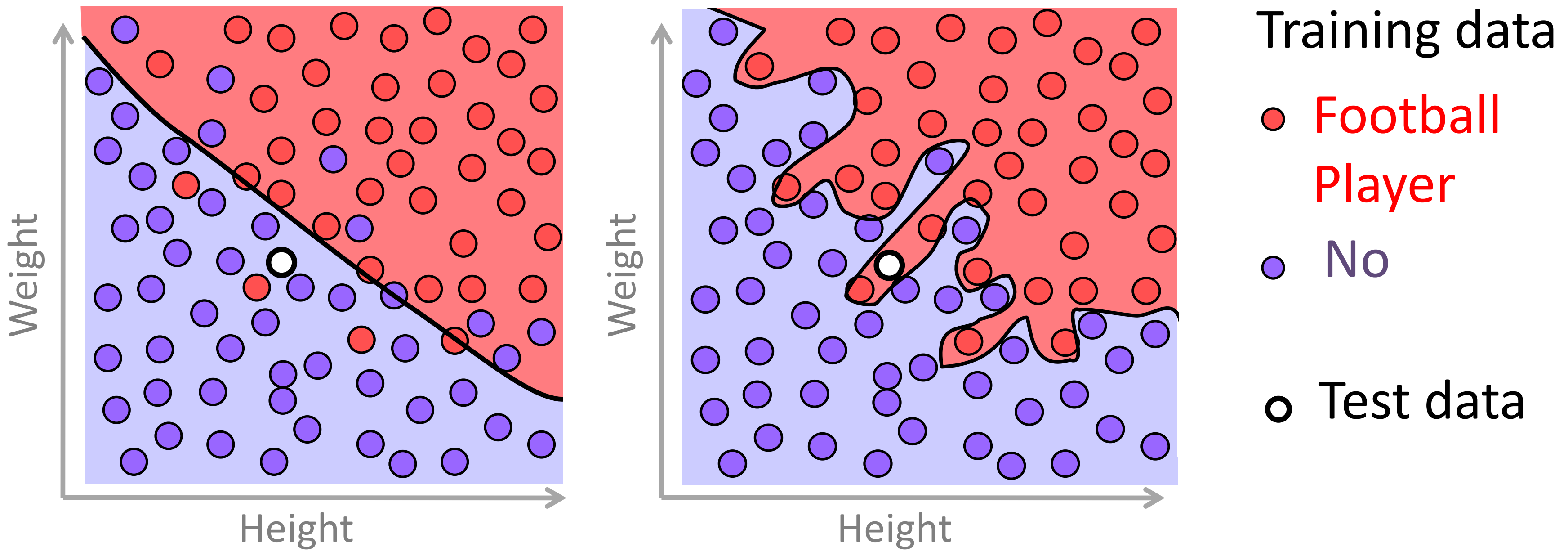
Finite hypothesis space

3) for all h $\in$ H,

$$|\text{error}_{\text{true}}(h) - \text{error}_{\text{train}}(h)| \leq \varepsilon = 8\sqrt{\frac{VC(H)\left(\ln\frac{m}{VC(H)} + 1\right) + \ln\frac{8}{\delta}}{2m}}$$

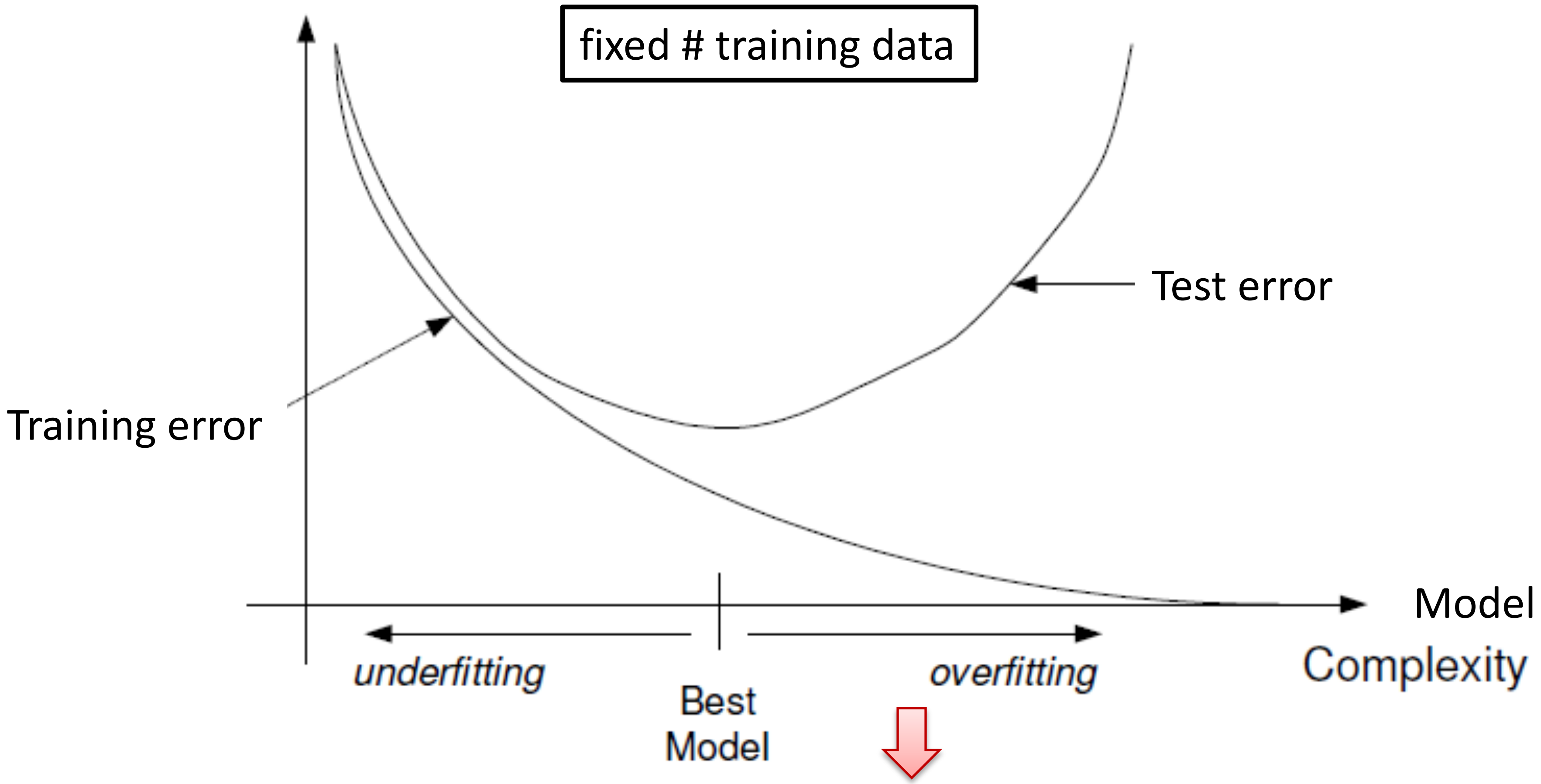
Infinite hypothesis space

# Training Data vs. Test Data



- A good machine learning algorithm
  - Does not **overfit** training data
  - **Generalizes** well to test data

# Training vs. Test Error



Training error is no longer a good indicator of test error

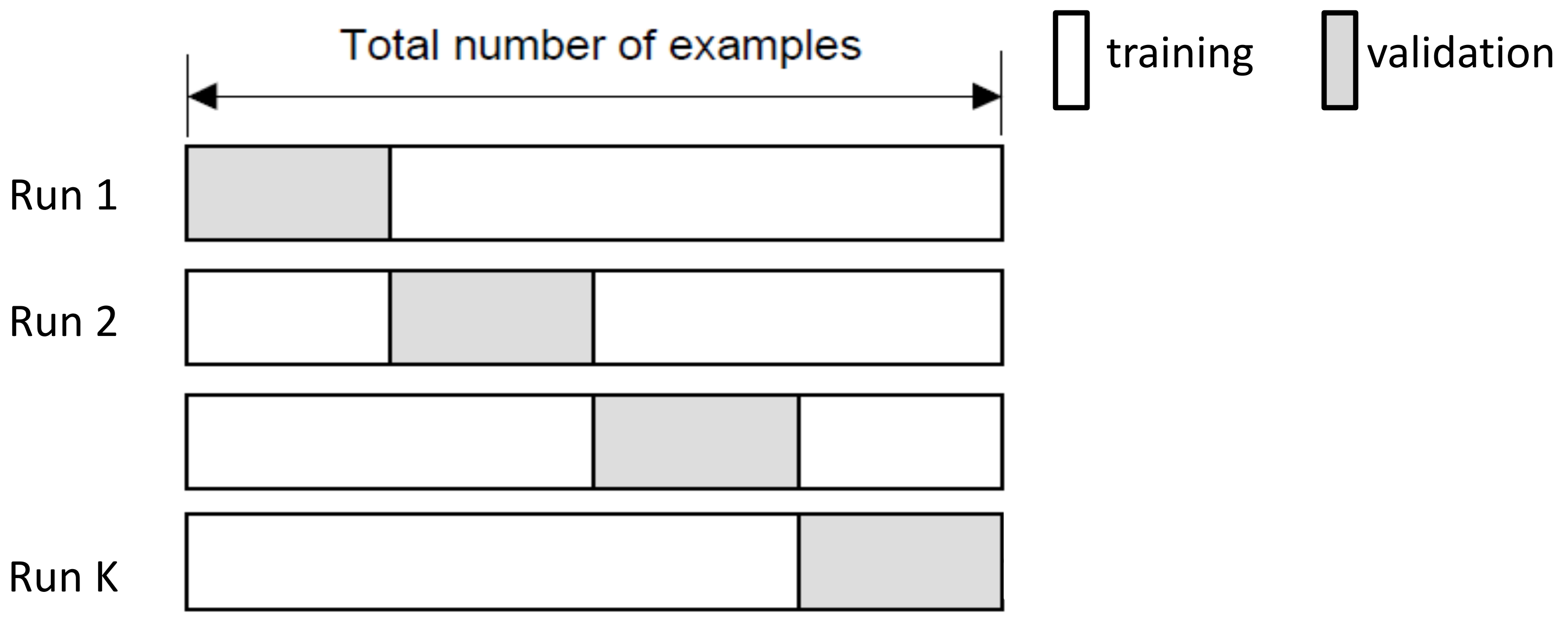
# Cross-validation

## K-fold cross-validation

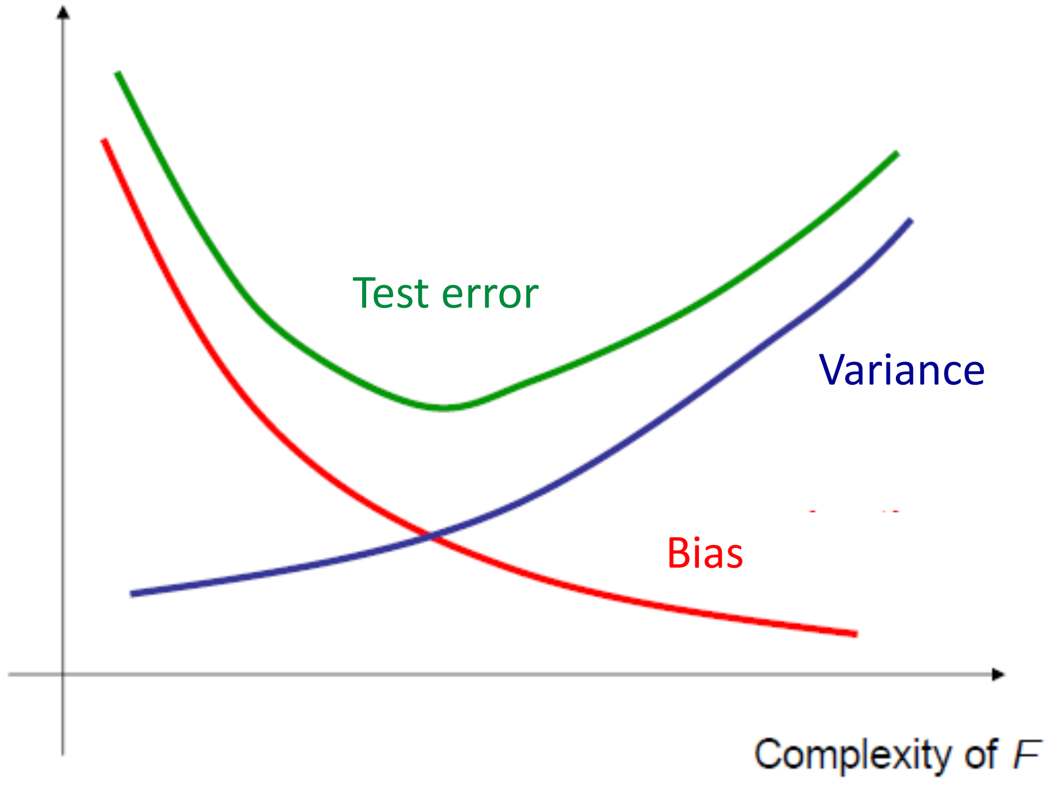
Create K-fold partition of the dataset.

Do K runs: train using K-1 partitions and calculate validation error on remaining partition (rotating validation partition on each run).

Report average validation error



# Bias-Variance tradeoff



Bias = E[f(X)] – f*(X) How far is the model from best model on average

Variance = $E[(f(X) - E[f(X)])^2]$ How variable is the model