Midterm Review
CS 7301: Advanced Machine Learning

Vibhav Gogate
The University of Texas at Dallas

Supervised Learning

- Issues in supervised learning
  - What makes learning hard
- Point Estimation: MLE vs Bayesian estimation
- Linear models
  - Linear Regression, Logistic Regression, SVMs, Perceptron, Naïve Bayes under certain restrictions
- Non-linear models
  - Decision trees, Neural networks, Kernels
- Non-parametric algorithms
  - Nearest neighbor algorithms

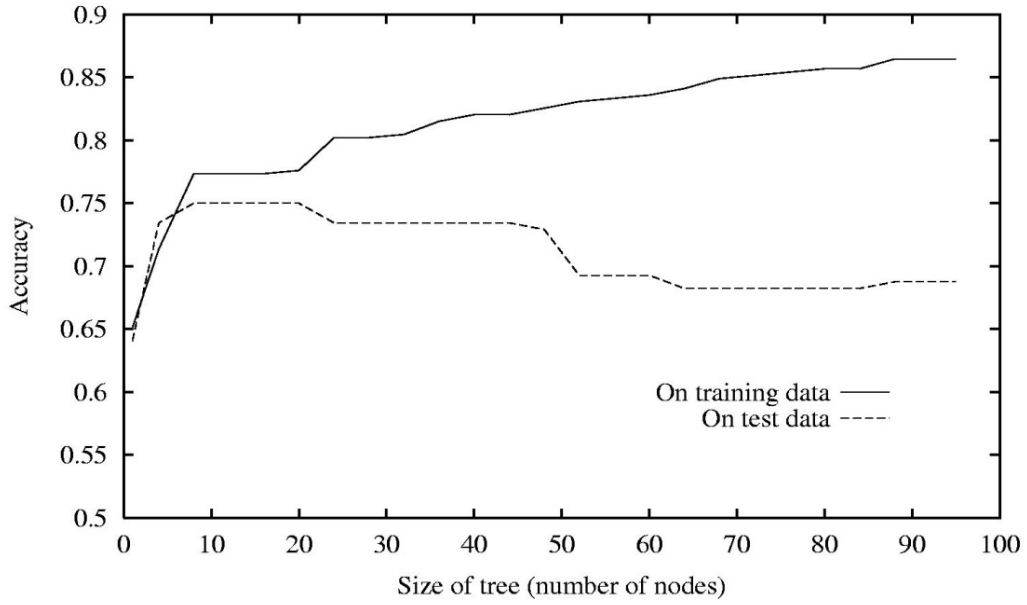
Key Perspective on Learning

- Representation
- Evaluation or Loss Function
  - Error + regularization
- Learning as Optimization
  - Closed form
  - Greedy search
  - Gradient ascent

What you should know in Decision Tree Learning?

- Representation
  - What it can represent and how
  - Size/Complexity of the representation
- Heuristics for selecting the next attribute
  - Information gain, one-step look ahead, gain ratio
  - What makes the heuristic good?
  - What are its cons?
  - Complexity analysis
  - Sample exam question: if I tweak the selection heuristic, how will that change the complexity and quality?
- Overfitting and Pruning
- Handling missing data
- Handling continuous attributes

Overfitting in Decision Tree Learning



- Noise
- Small number of examples associated with each leaf
  - What if only one example is associated with a leaf. Can you believe it?
  - Coincidental regularities

Probability Theory

- Be able to apply and understand
  - Axioms of probability
  - Distribution vs density
  - Conditional probability
  - Sum-rule, chain-rule
  - Bayes rule
- Sample question: If you know P(A|B), do you have enough information to compute P(B|A)?

Maximum Likelihood Estimation

- **Data:** Observed set D of α_H Heads and α_T Tails
- **Hypothesis:** Binomial distribution
- **Learning:** finding θ is an optimization problem
  - What's the objective function?

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

- **MLE:** Choose θ to maximize probability of *D*

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta} P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \ln P(\mathcal{D} \mid \theta) \end{aligned}$$

How to get a closed form solution?

$$\begin{aligned}\widehat{\theta} &= \arg\max_{\theta} \ln P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\end{aligned}$$

- Set derivative to zero, and solve!

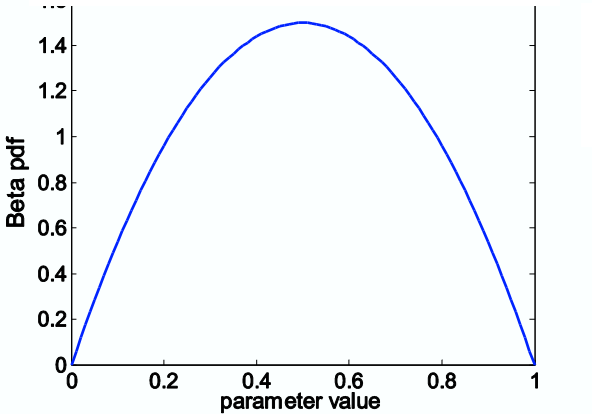
$$\begin{aligned}\frac{d}{d\theta} \ln P(\mathcal{D} \mid \theta) &= \frac{d}{d\theta}\left[\ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\right] \\ &= \frac{d}{d\theta}\left[\alpha_H \ln\theta + \alpha_T \ln(1-\theta)\right] \\ &= \alpha_H \frac{d}{d\theta}\ln\theta + \alpha_T \frac{d}{d\theta}\ln(1-\theta) \\ &= \frac{\alpha_H}{\theta} - \frac{\alpha_T}{1-\theta} = 0\end{aligned}$$

$$\boxed{\widehat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T}}$$

What if I have prior beliefs?

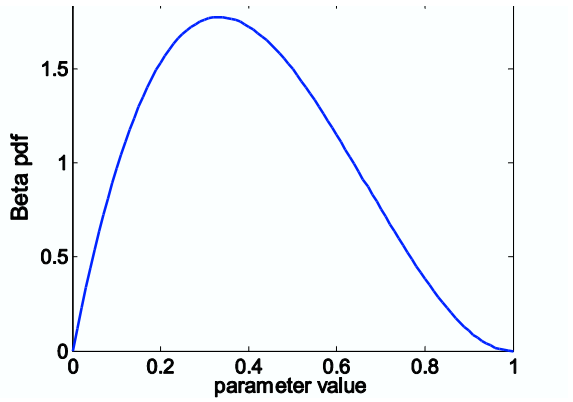
- Billionaire says: Wait, I know that the thumbtack is "close" to 50-50. What can you do for me now?
- **You say: I can learn it the Bayesian way…**
- Rather than estimating a single θ, we obtain a distribution over possible values of θ

In the beginning



Observe flips
e.g.: {tails, tails}

After observations

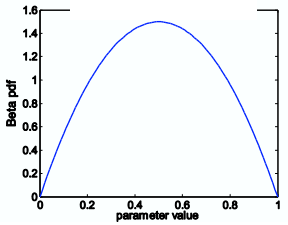


Bayesian Learning

Use Bayes rule:

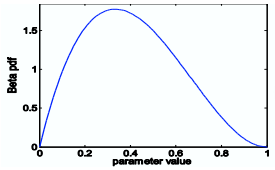
Data Likelihood

Prior



$$P(\theta \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \theta) P(\theta)}{P(\mathcal{D})}$$

Posterior



Normalization

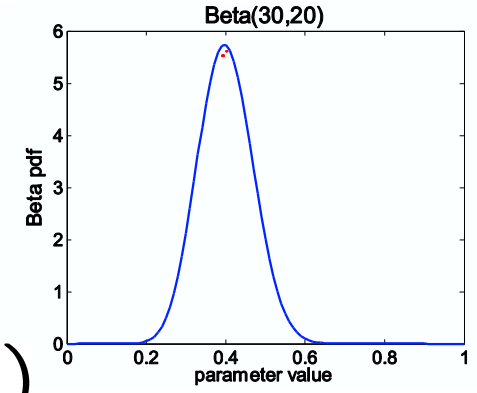
Or equivalently: $P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta) P(\theta)$

Also, for uniform priors:

→ reduces to MLE objective

$$P(\theta) \propto 1 \qquad P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta)$$

MAP: Maximum a Posteriori Approximation



$$P(\theta \mid \mathcal{D}) \sim Beta(\beta_H + \alpha_H, \beta_T + \alpha_T)$$

$$E[f(\theta)] = \int_0^1 f(\theta) P(\theta \mid \mathcal{D}) d\theta$$

- As more data is observed, Beta is more certain
- MAP: use most likely parameter to approximate the expectation

$$\hat{\theta} = \arg\max_{\theta} P(\theta \mid \mathcal{D})$$

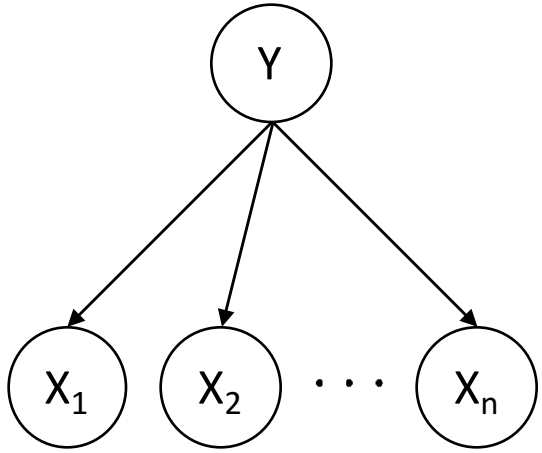
$$E[f(\theta)] \approx f(\hat{\theta})$$

What you should know?

- MLE vs MAP and the relationship between the two
- MLE learning and Bayesian learning
  - Thumbtack example
  - Gaussians

The Naïve Bayes Classifier

- **Given:**
  - Prior P(Y)
  - *n* conditionally independent features **X** given the class Y
  - For each X_i, we have likelihood $P(X_i|Y)$



- **Decision rule:**

$$y^* = h_{NB}(\mathbf{x}) = \arg\max_y P(y)P(x_1, \ldots, x_n \mid y)$$
$$= \arg\max_y P(y)\prod_i P(x_i|y)$$

Subtleties of Naïve Bayes

- What is the hypothesis space?
- What kind of functions can it learn?
- When does it work and when it does not?
  - Correlated features
- MLE vs Bayesian learning of Naïve Bayes
- Gaussian Naïve Bayes

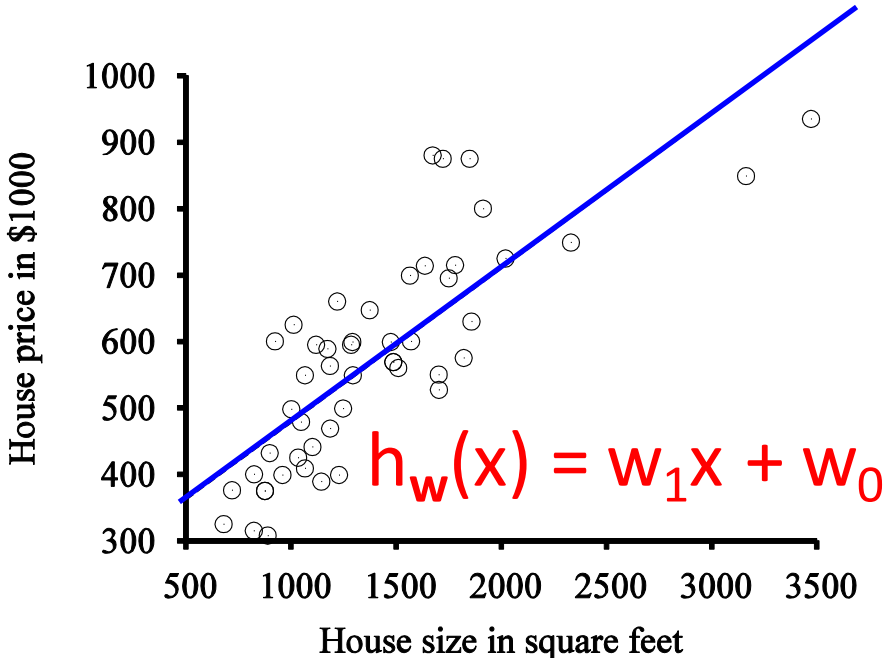
Generative *vs.* **Discriminative** Classifiers

- **Want to Learn**: h:**X** $\mapsto$ Y
  - **X** – features
  - Y – target classes
- **Generative classifier**, e.g., Naïve Bayes:

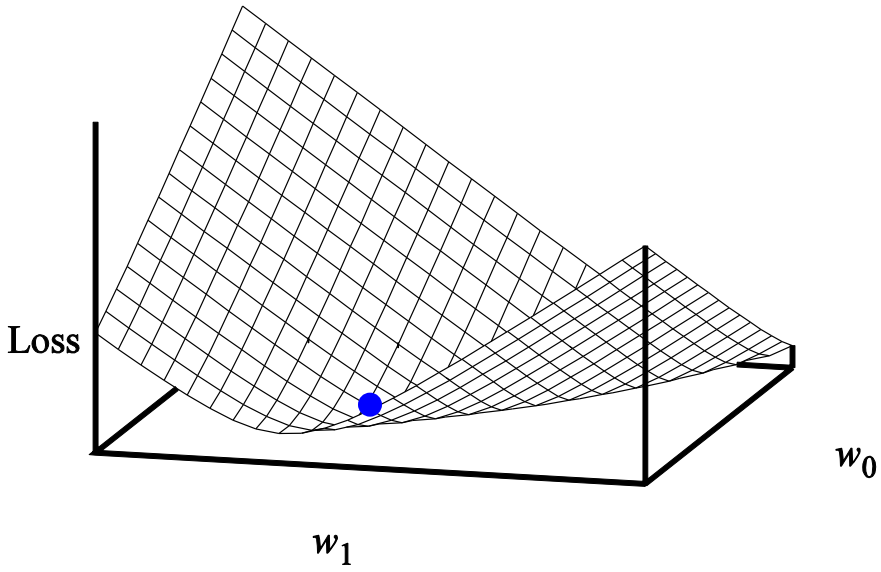
$$P(Y \mid \mathbf{X}) \propto P(\mathbf{X} \mid Y)\, P(Y)$$

  - Assume some **functional form for P(X|Y), P(Y)**
  - Estimate parameters of P(X|Y), P(Y) directly from training data
  - Use Bayes rule to calculate P(Y|X= x)
  - This is a **'*generative*' model**
    - **Indirect** computation of P(Y|X) through Bayes rule
    - As a result, **can also generate a sample of the data**, $P(X) = \sum_y P(y)\, P(X|y)$
- **Discriminative classifiers**, e.g., Logistic Regression:
  - Assume some **functional form for P(Y|X)**
  - Estimate parameters of P(Y|X) directly from training data
  - This is the **'*discriminative*' model**
    - Directly learn P(Y|X)
    - But **cannot obtain a sample of the data**, because P(X) is not available

Linear Regression



$\text{Argmin}_w \text{Loss}(h_w)$

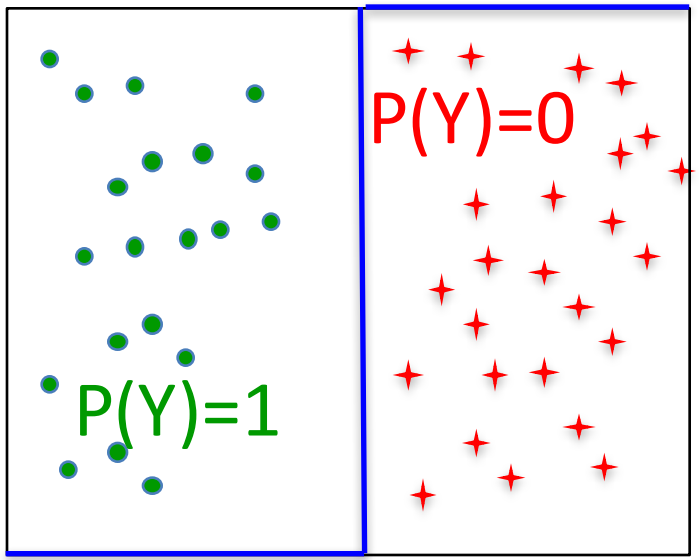


$$w_1 = \frac{N\sum(x_j y_j) - (\sum x_j)(\sum y_j)}{N\sum(x_j^2) - (\sum x_j)^2}$$

$$w_0 = (\sum(y_j) - w_1(\sum x_j)/N$$

Logistic Regression

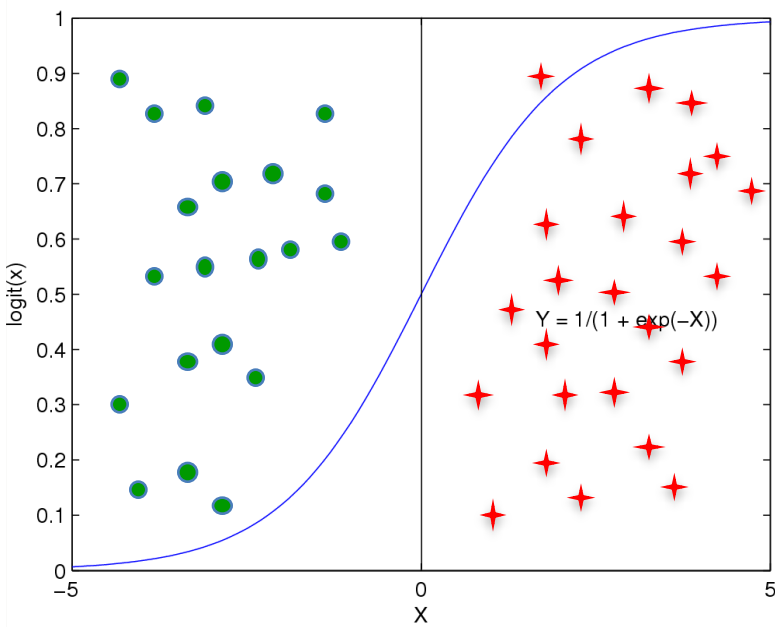
- Learn P(Y|**X**) directly!
  - Assume a particular functional form
  - ***Not differentiable…***



Logistic Regression

- Learn P(Y | **X**) directly!
  - Assume a particular functional form
  - Logistic Function
    - Aka Sigmoid

$$\frac{1}{1 + exp(-z)}$$



Issues in Linear and Logistic Regression

- Overfitting avoidance: Regularization
  - L1 vs L2 regularization

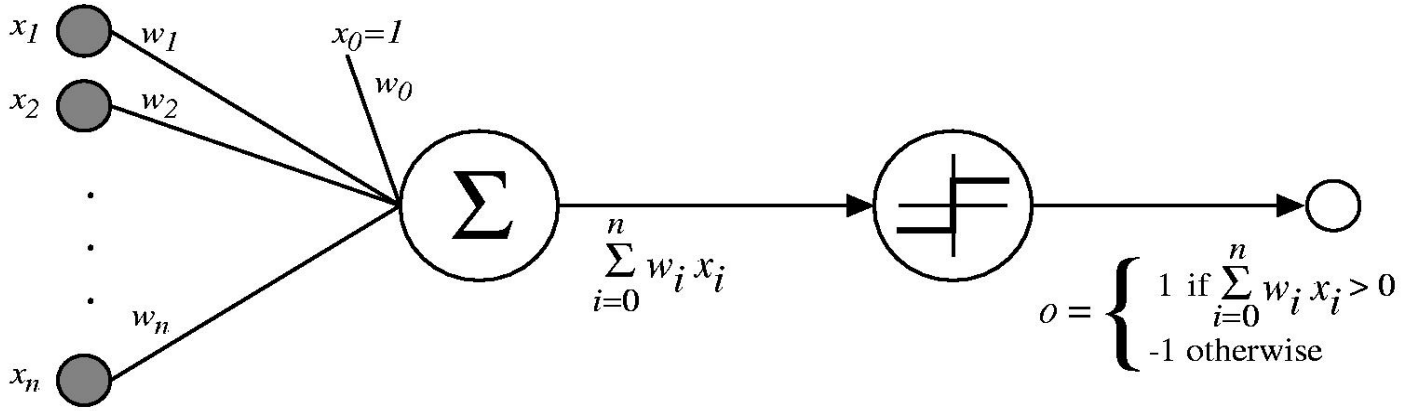
$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})] \right\}$$

What you should know about Logistic Regression (LR)

- Gaussian Naïve Bayes with class-independent variances representationally equivalent to LR
  - Solution differs because of objective (loss) function
- In general, NB and LR make different assumptions
  - NB: Features independent given class ! assumption on P(**X**|Y)
  - LR: Functional form of P(Y|**X**), no assumption on P(**X**|Y)
- LR is a linear classifier
  - decision rule is a hyperplane
- LR optimized by conditional likelihood
  - no closed-form solution
  - concave ! global optimum with gradient ascent
  - Maximum conditional a posteriori corresponds to regularization
- Convergence rates
  - GNB (usually) needs less data
  - LR (usually) gets to better solutions in the limit

Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

From Logistic Regression to the Perceptron: 2 easy steps!

- Logistic Regression: (in vector notation): y is {0,1}

$$w = w + \eta \sum_j [y_j^* - p(y_j^*|x_j, w)] f(x_j)$$

- Perceptron: y is {0,1}, y(x;w) is prediction given w

$$w = w + [y^* - y(x;w)] f(x)$$

Differences?

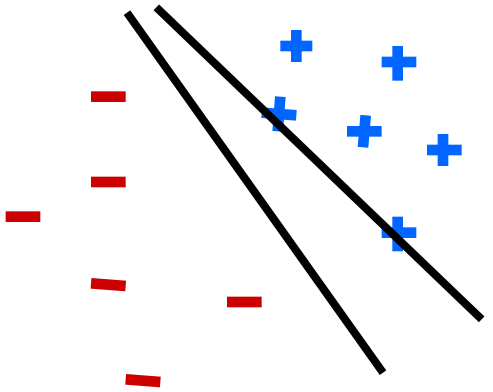
- Drop the Σ_j over training examples: online vs. batch learning
- Drop the dist'n: probabilistic vs. error driven learning

Properties of Perceptrons

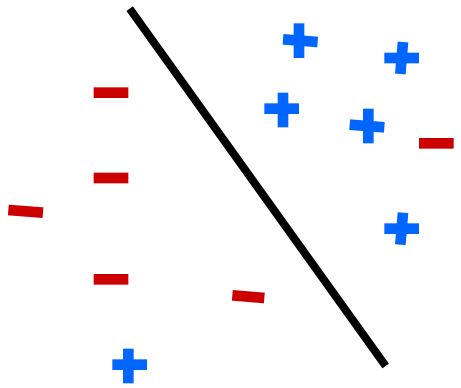
- Separability: some parameters get the training set perfectly correct

- Convergence: if the training is separable, perceptron will eventually converge (binary case)

Separable

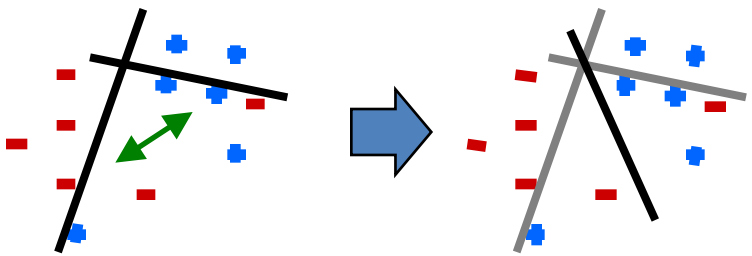


Non-Separable

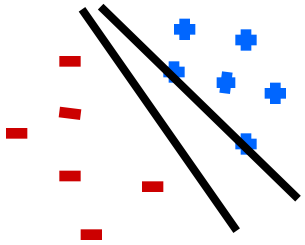


Problems with the Perceptron

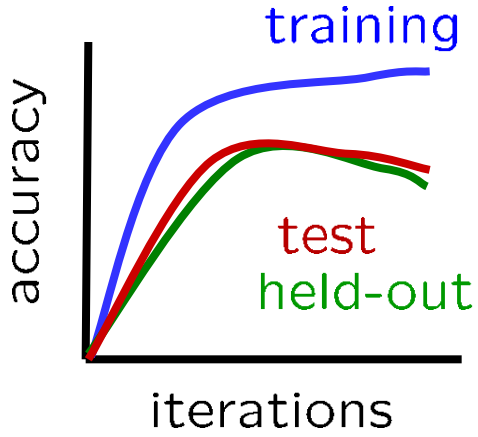
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



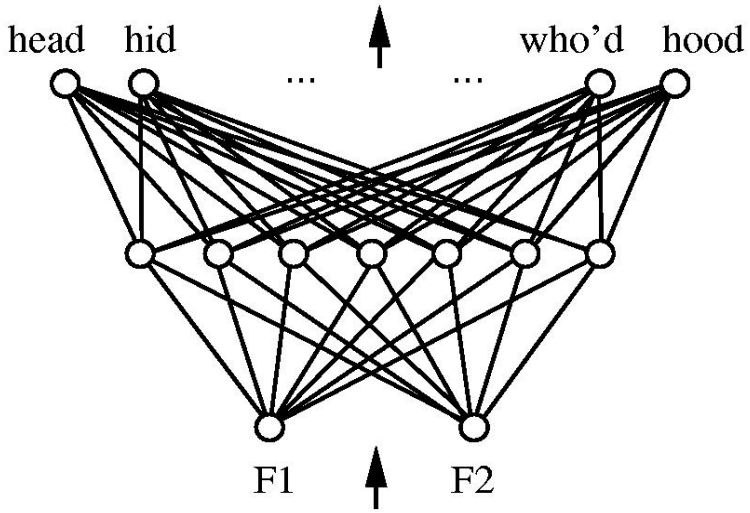
- Mediocre generalization: finds a "barely" separating solution



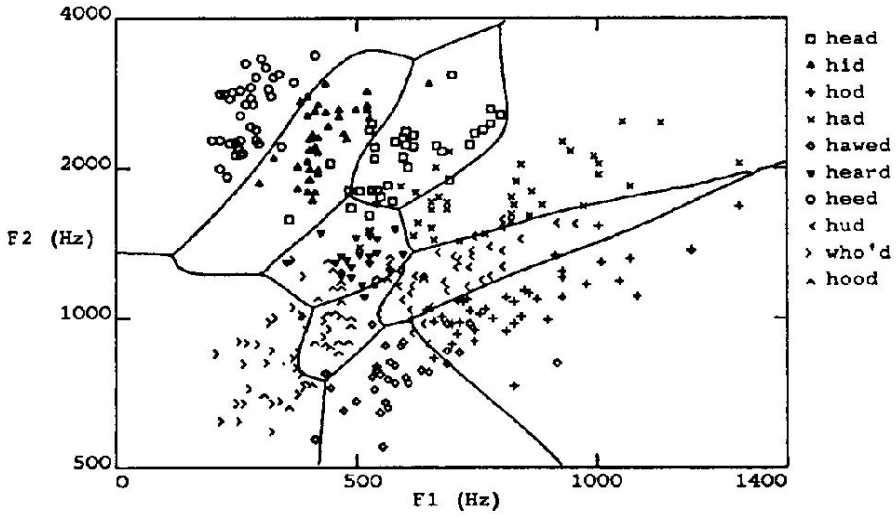
- Overtraining: test / validation accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



Multilayer Networks of Sigmoid Units



$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$



Backpropagation Algorithm

Initialize all weights to small random numbers

Until convergence, Do

For each training example, Do

1. Input it to network and compute network outputs
2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

Neural networks: What you should know?

- How does it learn non-linear functions?
- Can it learn, for example an XOR function?
  - Draw a neural network for it with appropriate weights
- Backprop
- Overfitting
- What kind of functions can it learn?
- Tradeoff
  - number of hidden units
  - number of layers

Linear SVM

- Aim: Learn a large margin classifier
- Mathematical Formulation:

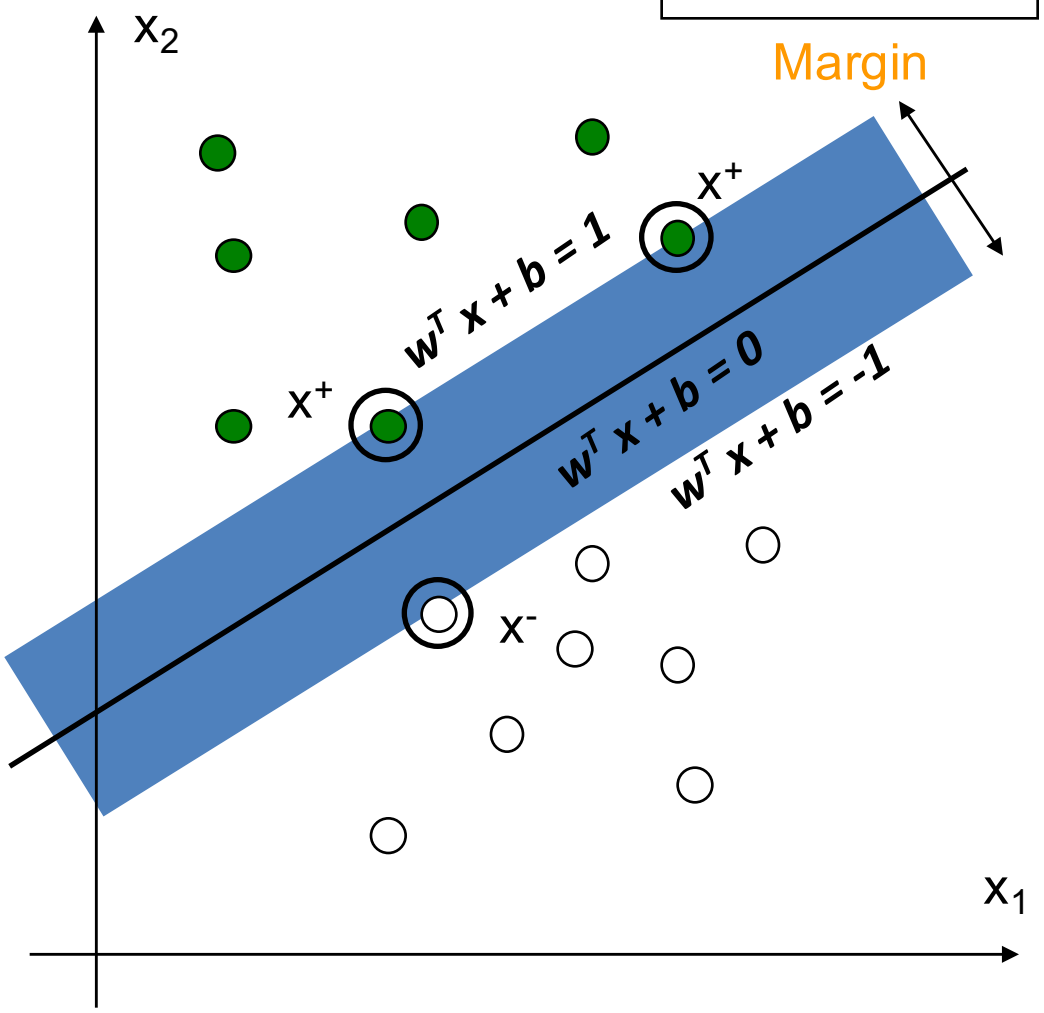
$$\text{maximize} \quad \frac{2}{\|\mathbf{w}\|}$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$

Common theme in machine learning:
LEARNING IS OPTIMIZATION

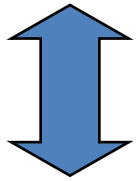


Solving the Optimization Problem

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

Lagrangian Dual Problem

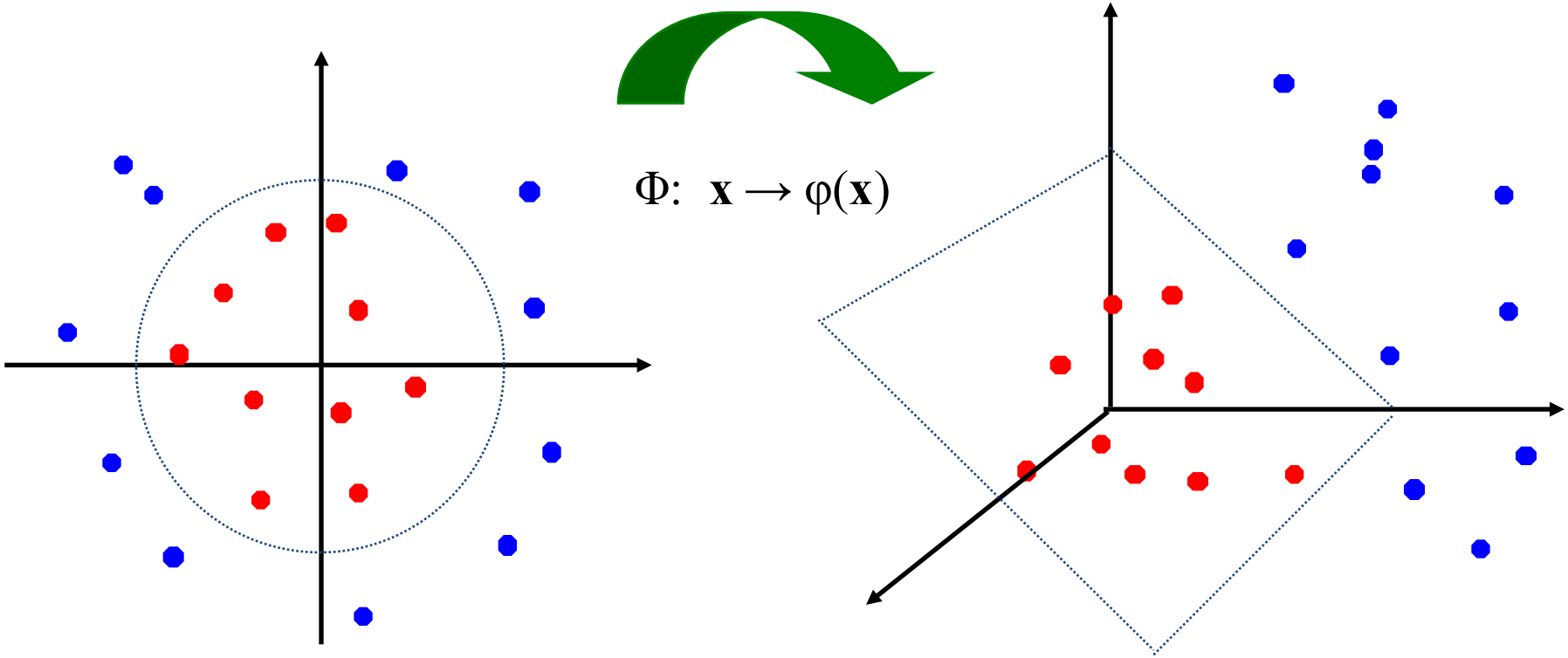


$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{s.t.} \quad \alpha_i \geq 0 \text{ , and } \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Non-linear SVMs: Feature Space

- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:



This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.

- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

  - Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

- In general, functions that satisfy *Mercer's condition* can be kernel functions: Kernel matrix should be positive semidefinite.

K-nearest Neighbor

- 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.

Nearest Neighbor

- Advantages
  - variable-sized hypothesis space
  - Learning is extremely efficient
    - however growing a good kd-tree can be expensive
  - Very flexible decision boundaries
- Disadvantages
  - distance function must be carefully chosen
  - Irrelevant or correlated features must be eliminated
  - Typically cannot handle more than 30 features
  - Computational costs: Memory and classification-time computation

Locally Weighted Linear Regression: LWLR

- Idea:
  - k-NN forms local approximation for each query point x_q
  - Why not form an explicit approximation $\hat{f}$ for region surrounding x_q
    - Fit linear function to k nearest neighbors
    - Fit quadratic, ...
    - Thus producing ``piecewise approximation'' to $\hat{f}$
      - Minimize error over k nearest neighbors of x_q
      - Minimize error entire set of examples, weighting by distances
      - Combine two above