

On-Line Learning

- Not the most general setting for on-line learning.
- Not the most general metric
- (Regret: cumulative loss; Competitive analysis)

On Line Model

- **Model:**
 - Instance space: X (dimensionality – n)
 - Target: $f: X \rightarrow \{0,1\}$, $f \in C$, concept class (parameterized by n)
 - **Protocol:**
 - learner is given $x \in X$
 - learner predicts $h(x)$, and is then given $f(x)$ (feedback)
 - **Performance:** learner makes a mistake when $h(x) \neq f(x)$
 - number of mistakes algorithm A makes on sequence S of examples, for the target function f .
- $$M_A(C) = \max_{f \in C, S} M_A(f, S)$$
- A is a mistake bound algorithm for the concept class C , if $M_A(C)$ is a polynomial in n , the complexity parameter of the target concept.

Representation

Importance of
Representation

- Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?
 - **Theorem:** Given a sample on n attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the sample and has the minimum number of attributes.
 - [David Haussler, AIJ'88: "Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework"]
- Same holds for Disjunctions.
- Intuition: Reduction to minimum set cover problem.
 - Given a collection of sets that cover X , define a set of examples so that learning the best (dis/conj)junction implies a minimal cover.
- Consequently, we cannot learn the concept efficiently as a **(dis/con)junction**.
- But, we will see that we can do that, if we are willing to learn the concept as a Linear Threshold function.
- **In a more expressive class, the search for a good hypothesis sometimes becomes combinatorially easier.**

Linear Functions

$$f(x) = \begin{cases} 1 & \text{if } W_1 X_1 + W_2 X_2 + \dots + W_n X_n \geq \theta \\ 0 & \text{Otherwise} \end{cases}$$

■ Disjunctions $y = X_1 \vee X_3 \vee X_5$
 $y = (1 \cdot X_1 + 1 \cdot X_3 + 1 \cdot X_5 \geq 1)$



■ At least m of n: $y = \text{at least 2 of } \{X_1, X_3, X_5\}$
 $y = (1 \cdot X_1 + 1 \cdot X_3 + 1 \cdot X_5 \geq 2)$



■ Exclusive-OR: $y = (X_1 \wedge X_2 \vee) (X_1 \wedge X_2)$



■ Non-trivial DNF $y = (X_1 \wedge X_2) \vee (X_3 \wedge X_4)$



Linear Functions

Perceptron learning rule

- We learn $f: X \rightarrow \{-1, +1\}$ represented as $f = \text{sgn}\{w \bullet x\}$
- Where $X = \{0, 1\}^n$ or $X = \mathbb{R}^n$ and $w \in \mathbb{R}^n$
- Given Labeled examples: $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$

1. Initialize $w = 0 \in \mathbb{R}^n$
2. Cycle through all examples
 - a. Predict the label of instance x to be $y' = \text{sgn}\{w \bullet x\}$
 - b. If $y' \neq y$, **update** the weight vector:

$$\mathbf{w} = \mathbf{w} + r y \mathbf{x} \quad (r - \text{a constant, learning rate})$$

Otherwise, if $y' = y$, leave weights unchanged.

Perceptron

Perceptron Convergence

- **Perceptron Convergence Theorem:**
- If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge
 - How long would it take to converge ?
- **Perceptron Cycling Theorem:**
- If the training data is not linearly separable the perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop.
 - How to provide robustness, more expressivity ?

Perceptron: Mistake Bound Theorem

- Maintains a weight vector $w \in \mathcal{R}^N$, $w_0 = (0, \dots, 0)$.
- Upon receiving an example $x \in \mathcal{R}^N$
- Predicts according to the linear threshold function $w \bullet x \geq 0$.
- **Theorem [Novikoff, 1963]** *Let $(x_1; y_1), \dots, (x_t; y_t)$, be a sequence of labeled examples with $x_i \in \mathcal{R}^N$, $\|x_i\| \leq R$ and $y_i \in \{-1, 1\}$ for all i . Let $u \in \mathcal{R}^N$, $\gamma > 0$ be such that, $\|u\| = 1$ and $y_i u \bullet x_i \geq \gamma$ for all i .*

Complexity Parameter

Then Perceptron makes at most R^2 / γ^2 mistakes on this example sequence.

(see additional notes)

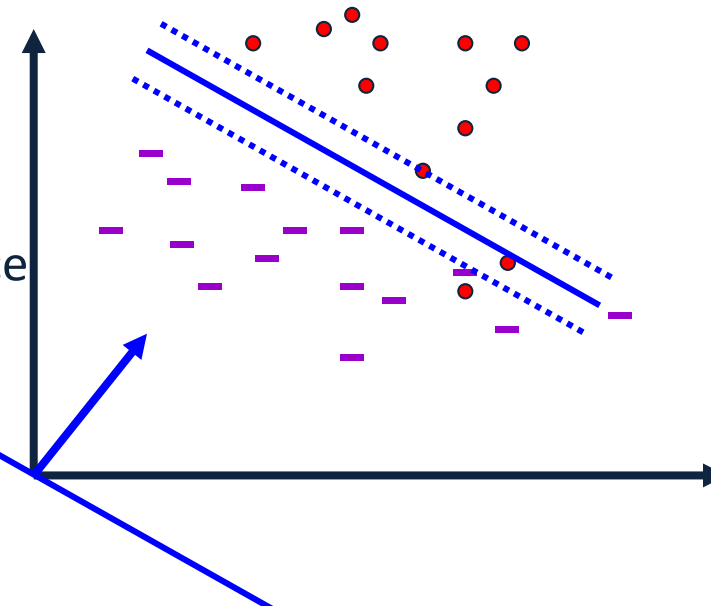
Analysis

Robustness to Noise

- In the case of non-separable data, the extent to which a data point fails to have margin γ via the hyperplane w can be quantified by a slack variable

$$\xi_i = \max(0, \gamma - y_i w \cdot x_i).$$

- Observe that when $\xi_i = 0$, the example x_i has margin at least γ . Otherwise, it grows linearly with $-\gamma - y_i w \cdot x_i$
- Denote: $D_2 = [\sum \{\xi_i^2\}]^{1/2}$
- **Theorem:** The perceptron is guaranteed to make no more than $((R+D_2)/\gamma)^2$ mistakes on any sequence of examples satisfying $\|x_i\|^2 < R$
- Perceptron is expected to have some robustness to noise.



Non separable case

Winnow Algorithm

Initialize : $\theta = n$; $w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ but $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ (if $x_i = 1$) (promotion)

If $f(x) = 0$ but $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ (if $x_i = 1$) (demotion)

- The Winnow Algorithm learns Linear Threshold Functions.
- For the class of disjunctions:
 - instead of demotion we can use elimination.

Winnow

Winnow – Mistake Bound

- Claim: Winnow makes $O(k \log n)$ mistakes on k -disjunctions

Initialize : $\theta = n$; $w_i = 1$

Prediction is 1 iff $w \cdot x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ but $w \cdot x < \theta$, $w_i \leftarrow 2w_i$ (if $x_i = 1$) (promotion)

If $f(x) = 0$ but $w \cdot x \geq \theta$, $w_i \leftarrow w_i/2$ (if $x_i = 1$) (demotion)

- u - # of mistakes on positive examples (promotions)
- v - # of mistakes on negative examples (demotions)

1. $u < k \log(2n)$

A weight that corresponds to a good variable is only promoted. When these weights get to n there will be no more mistakes on positives.

Analysis

Regularization Via Averaged Perceptron

- An **Averaged Perceptron** Algorithm is motivated by the following considerations:
 - Every Mistake-Bound Algorithm can be converted efficiently to a PAC algorithm – to yield **global guarantees** on performance.
 - In the mistake bound model:
 - We don't know when we will make the mistakes.
 - In the PAC model:
 - Dependence is on number of examples seen and not number of mistakes.
 - **Which hypothesis will you choose...??**
 - Being consistent with more examples is better
- To convert a given Mistake Bound algorithm (into a global guarantee algorithm):
 - Wait for a long stretch w/o mistakes (there must be one)
 - Use the hypothesis at the end of this stretch.
 - Its PAC behavior is relative to the length of the stretch.
- **Averaged Perceptron** returns a weighted average of a number of earlier hypotheses; the weights are a function of the length of no-mistakes stretch.

Averaged
Perceptron

Regularization Via Averaged Perceptron (or Winnow)

- **Training:**
[m : #(examples); k : #(mistakes) = #(hypotheses); c_i : consistency count for v_i]
- **Input:** a labeled training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$
- Number of epochs T
- **Output:** a list of weighted perceptrons $\{(v_1, c_1), \dots, (v_k, c_k)\}$
- Initialize: $k=0$; $v_1 = 0$, $c_1 = 0$
- Repeat T times:
 - For $i=1, \dots, m$:
 - Compute prediction $y' = \text{sign}(v_k \cdot x_i)$
 - If $y' = y$, then $c_k = c_k + 1$
else: $v_{k+1} = v_k + y_i x$; $c_{k+1} = 1$; $k = k+1$
- **Prediction:**
- **Given:** a list of weighted perceptrons $\{(v_1, c_1), \dots, (v_k, c_k)\}$; a new example x
Predict the label(x) as follows:
$$y(x) = \text{sign} \left[\sum_{i=1, k} c_i \text{sign}(v_i \cdot x) \right]$$

Averaged
Perceptron

II Perceptron with Margin

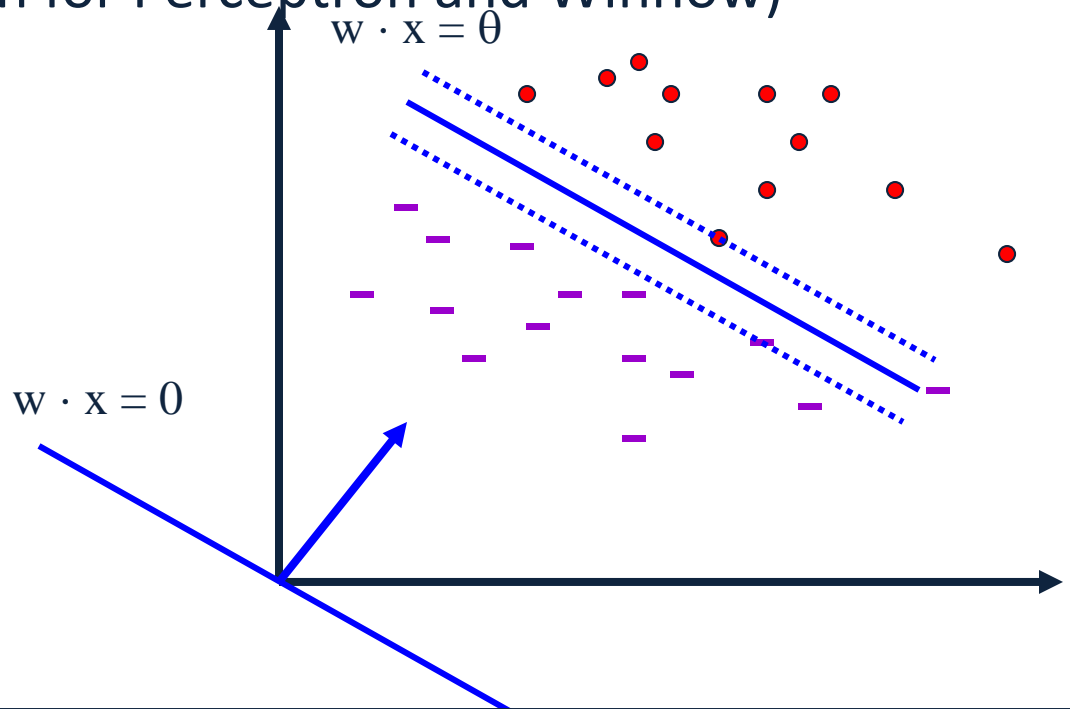
- Thick Separator (aka as Perceptron with Margin)
(Applies both for Perceptron and Winnow)

- Promote if:

- $w \cdot x - \theta < \gamma$

- Demote if:

- $w \cdot x - \theta > \gamma$



Note: γ is a functional margin. Its effect could disappear as w grows. Nevertheless, this has been shown to be a very effective algorithmic addition. (Grove & Roth 98,01; Karov et. al 97)

Winnow - Extensions

- This algorithm learns monotone functions
- For the general case:
 - Duplicate variables (down side?)
 - For the negation of variable x , introduce a new variable y .
 - Learn monotone functions over $2n$ variables
- Balanced version:
 - Keep two weights for each variable; effective weight is the difference

Update Rule :

If $f(x) = 1$ but $(w^+ - w^-) \cdot x \leq \theta$, $w_i^+ \leftarrow 2w_i^+$ $w_i^- \leftarrow \frac{1}{2}w_i^-$ where $x_i = 1$ (promotion)

If $f(x) = 0$ but $(w^+ - w^-) \cdot x \geq \theta$, $w_i^+ \leftarrow \frac{1}{2}w_i^+$ $w_i^- \leftarrow 2w_i^-$ where $x_i = 1$ (demotion)

- We'll come back to this idea when talking about multiclass.

Extensions

Winnow – A Robust Variation

- Modeling:
 - **Adversary's turn**: may change the target concept by adding or removing some variable from the target disjunction.
 - **Cost** of each addition move is 1.
 - **Learner's turn**: makes prediction on the examples given, and is then told the correct answer (according to current target function)
 - **Winnow-R**: Same as Winnow, only doesn't let weights go below $1/2$
 - **Claim**: Winnow-R makes $O(c \log n)$ mistakes, (c - cost of adversary) (generalization of previous claim)

Extensions

General Stochastic Gradient Algorithms

- Given examples $\{z=(x,y)\}_{1,m}$ from a distribution over $X \times Y$, we are trying to learn a linear function, parameterized by a weight vector w , so that we minimize the expected risk function

$$J(w) = E_z Q(z,w) \approx \frac{1}{m} \sum_{1,m} Q(z_i, w_i)$$

- In Stochastic Gradient Descent Algorithms we approximate this minimization by incrementally updating the weight vector w as follows:

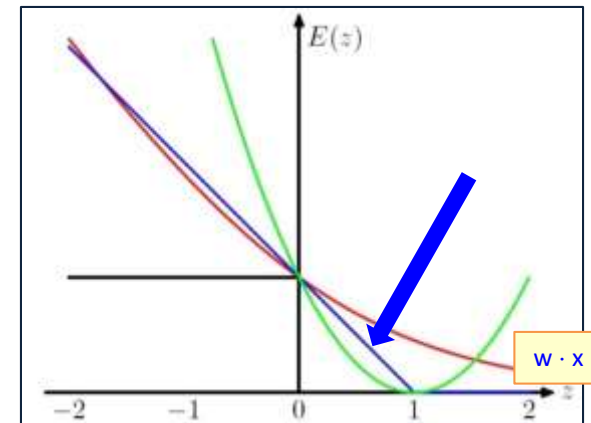
$$w_{t+1} = w_t - r_t g_w Q(z_t, w_t) = w_t - r_t g_t$$

- Where $g_t = g_w Q(z_t, w_t)$ is the gradient with respect to w at time t .
- The difference between algorithms now amounts to choosing a different loss function $Q(z, w)$

Stochastic Gradient Algorithms

$$w_{t+1} = w_t - r_t g_w Q(z_t, w_t) = w_t - r_t g_t$$

- **LMS:** $Q((x, y), w) = 1/2 (y - w \cdot x)^2$
- leads to the update rule (Also called Widrow's Adaline):
$$w_{t+1} = w_t + r (y_t - w_t \cdot x_t) x_t$$
- Here, even though we make binary predictions based on $\text{sign}(w \cdot x)$ we do not take the sign of the dot-product into account in the loss.
- Another common loss function is:
- **Hinge loss:**
$$Q((x, y), w) = \max(0, 1 - y w \cdot x)$$
- This leads to the **perceptron** update rule:
- If $y_i w_i \cdot x_i > 1$ (No mistake, by a margin): **No update**
- Otherwise (Mistake, relative to margin): $w_{t+1} = w_t + r y_t x_t$



New Stochastic Gradient Algorithms

$$w_{t+1} = w_t - r_t g_w \quad Q(z_t, w_t) = w_t - r_t g_t$$

(notice that this is a vector, each coordinate (feature) has its own $w_{t,j}$ and $g_{t,j}$)

- So far, we used fixed learning rates $r = r_t$, but this can change.
- **AdaGrad** alters the update to adapt based on historical information, so that frequently occurring features in the gradients get small learning rates and infrequent features get higher ones.
- The idea is to “learn slowly” from frequent features but “pay attention” to rare but informative features.
- Define a “per feature” learning rate for the feature j , as:
$$r_{t,j} = r / (G_{t,j})^{1/2}$$
- where $G_{t,j} = \sum_{k=1, t} g_{k,j}^2$ the sum of squares of gradients at feature j until time t .
- Overall, the update rule for Adagrad is:
$$w_{t+1,j} = w_{t,j} - g_{t,j} r / (G_{t,j})^{1/2}$$
- This algorithm is supposed to update weights faster than Perceptron or LMS when needed.

Regularization

- The more general formalism adds a **regularization** term to the risk function, and attempts to minimize:

$$J(w) = \sum_{1, m} Q(z_i, w_i) + \lambda R_i(w_i)$$

- Where R is used to enforce “simplicity” of the learned functions.

- LMS case: $Q((x, y), w) = (y - w \cdot x)^2$

- $R(w) = \|w\|_2^2$ gives the optimization problem called Ridge Regression.
- $R(w) = \|w\|_1$ gives a problem called the LASSO problem

- Hinge Loss case: $Q((x, y), w) = \max(0, 1 - y w \cdot x)$

- $R(w) = \|w\|_2^2$ gives the problem called Support Vector Machines

- Logistics Loss case: $Q((x, y), w) = \log(1 + \exp\{-y w \cdot x\})$

- $R(w) = \|w\|_2^2$ gives the problem called Logistics Regression

- These are convex optimization problems and, in principle, the same gradient descent mechanism can be used in all cases.

- We will see later why it makes sense to use the “size” of w as a way to control “simplicity”.

Generalization

- Dominated by the sparseness of the function space
 - Most features are irrelevant
- # of examples required by multiplicative algorithms depends mostly on # of relevant features
 - (Generalization bounds depend on the target $\|u\|$)
- # of examples required by additive algorithms depends heavily on sparseness of features space:
 - Advantage to additive. Generalization depend on input $\|x\|$
 - (Kivinen/Warmuth 95).

Generalization

Which Algorithm to Choose?

■ Generalization

The l_1 norm: $\|x\|_1 = \sum_i |x_i|$

The l_2 norm: $\|x\|_2 = (\sum_1^n |x_i|^2)^{1/2}$

The l_p norm: $\|x\|_p = (\sum_1^n |x_i|^p)^{1/p}$

The l_∞ norm: $\|x\|_\infty = \max_i |x_i|$

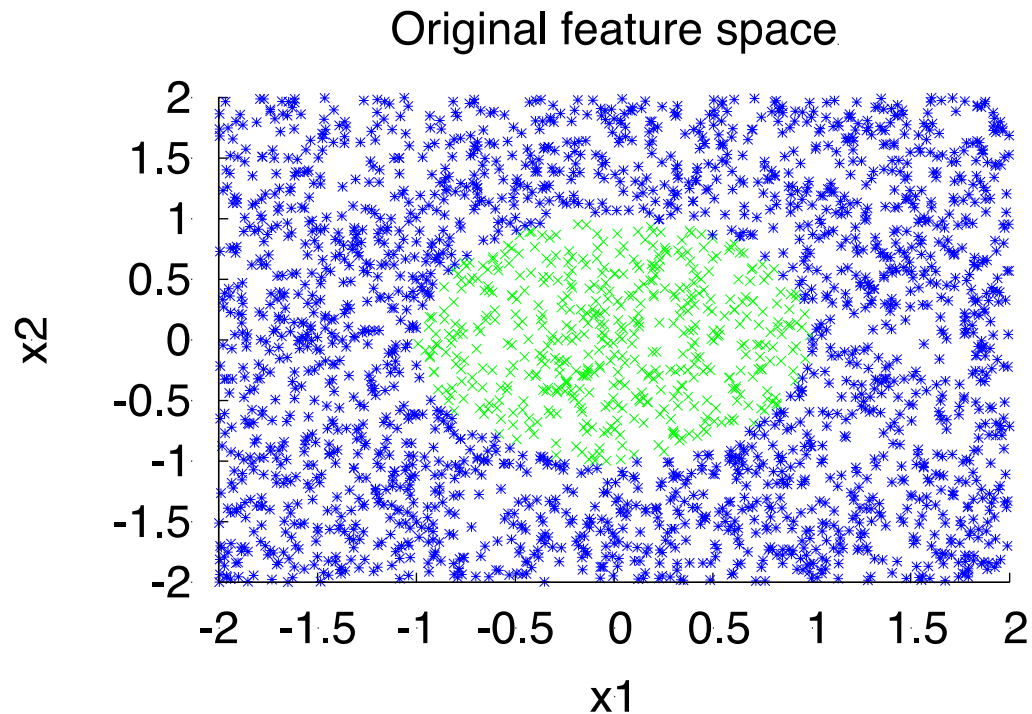
□ Multiplicative algorithms:

- Bounds depend on $\|u\|$, the separating hyperplane; i : example #)
- $M_w = 2 \ln n \|u\|_1^2 \frac{\max_i \|x^{(i)}\|_\infty^2}{\min_i (u \cdot x^{(i)})^2}$
- Do not care much about data; advantage with sparse target u

□ Additive algorithms:

- Bounds depend on $\|x\|$ (Kivinen / Warmuth, '95)
- $M_p = \|u\|_2^2 \frac{\max_i \|x^{(i)}\|_2^2}{\min_i (u \cdot x^{(i)})^2}$
- Advantage with few active features per example

Making data linearly separable



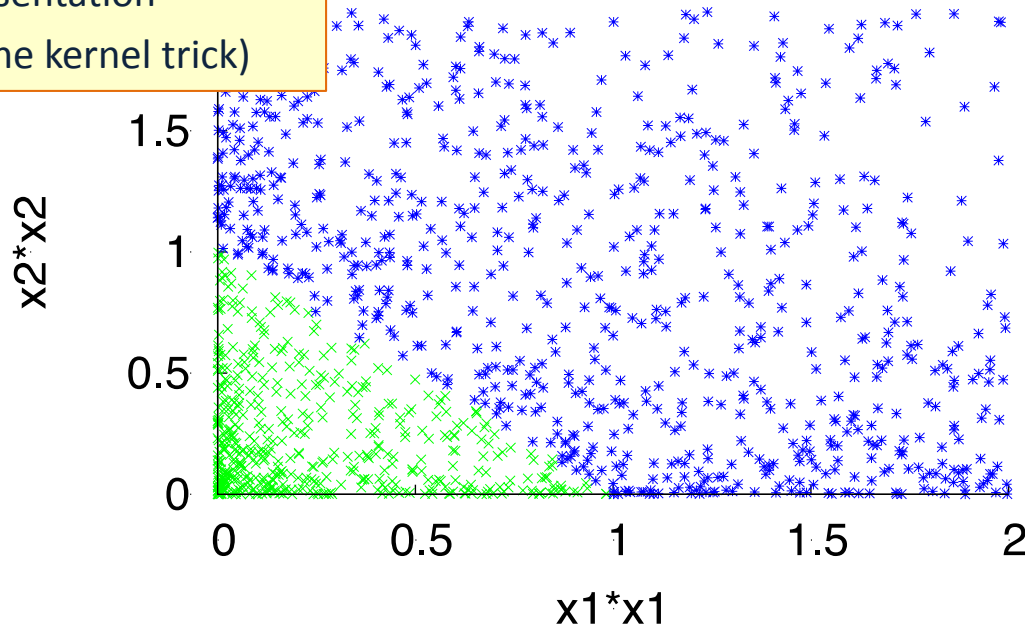
$$f(\mathbf{x}) = 1 \text{ iff } x_1^2 + x_2^2 \leq 1$$

Making data linearly separable

In order to deal with this, we introduce two new concepts:

- Dual Representation
- Kernel (& the kernel trick)

Transformed feature space



Transform data: $\mathbf{x} = (x_1, x_2) \Rightarrow \mathbf{x}' = (x_1^2, x_2^2)$
 $f(\mathbf{x}') = 1$ iff $x'_1 + x'_2 \leq 1$

Dual Representation

Examples : $\mathbf{x} \in \{0,1\}^n$;

Hypothesis : $\mathbf{w} \in \mathbb{R}^n$

$$f(\mathbf{x}) = \text{Th}_\theta \left(\sum_{i=1}^n \mathbf{w}_i x_i(\mathbf{x}) \right)$$

If Class = 1 but $\mathbf{w} \cdot \mathbf{x} \leq \theta$, $\mathbf{w}_i \leftarrow \mathbf{w}_i + 1$ (if $x_i = 1$) (promotion)

If Class = 0 but $\mathbf{w} \cdot \mathbf{x} \geq \theta$, $\mathbf{w}_i \leftarrow \mathbf{w}_i - 1$ (if $x_i = 1$) (demotion)

- Let \mathbf{w} be an initial weight vector for perceptron. Let $(\mathbf{x}^1,+)$, $(\mathbf{x}^2,+)$, $(\mathbf{x}^3,-)$, $(\mathbf{x}^4,-)$ be examples and assume mistakes are made on \mathbf{x}^1 , \mathbf{x}^2 and \mathbf{x}^4 .
- What is the resulting weight vector?

$$\mathbf{w} = \mathbf{w} + \mathbf{x}^1 + \mathbf{x}^2 - \mathbf{x}^4$$

- In general, the weight vector \mathbf{w} can be written as a linear combination of examples:

$$\mathbf{w} = \sum_{1,m} r \alpha_i y_i \mathbf{x}_i$$

- Where α_i is the **number of mistakes** made on \mathbf{x}_i .

Note: We care about the dot product: $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} =$

$$\begin{aligned} &= \left(\sum_{1,m} r \alpha_i y_i \mathbf{x}_i \right) \cdot \mathbf{x} \\ &= \sum_{1,m} r \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) \end{aligned}$$

Kernel Based Methods

$$f(\mathbf{x}) = \mathbf{T} \mathbf{h}_\theta \left(\sum_{\mathbf{z} \in M} \mathbf{S}(\mathbf{z}) \mathbf{K}(\mathbf{x}, \mathbf{z}) \right)$$

- A method to run Perceptron on a very large feature set, without incurring the cost of keeping a very large weight vector.
- Computing the dot product can be done in the original feature space.
- **Notice:** this pertains only to efficiency: The classifier is identical to the one you get by blowing up the feature space.
- Generalization is still relative to the real dimensionality (or, related properties).
- **Kernels** were popularized by SVMs, but many other algorithms can make use of them (== run in the dual).
 - Linear Kernels: no kernels; stay in the original space. A lot of applications actually use linear kernels.

General

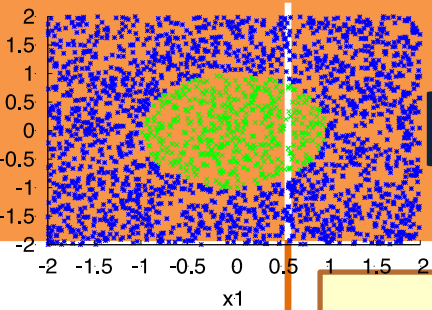
Implementation

$$f(\mathbf{x}) = \mathbf{T}h_{\theta} \left(\sum_{\mathbf{z} \in M} \mathbf{S}(\mathbf{z})\mathbf{K}(\mathbf{x}, \mathbf{z}) \right)$$

$$\mathbf{K}(\mathbf{x}, \mathbf{z}) = \sum_{i \in I} \mathbf{t}_i(\mathbf{z})\mathbf{t}_i(\mathbf{x})$$

- Simply run Perceptron in an on-line mode, but keep track of the set M .
- Keeping the set M allows us to keep track of $S(\mathbf{z})$.
- Rather than remembering the weight vector \mathbf{w} , **remember the set M** (P and D) – all those examples on which we made mistakes.
- Dual Representation

Kernels – General Conditions



- **Kernel Trick:** You want to work with degree 2 polynomial features, (x) . Then, your dot product will be in a space of dimensionality $n(n+1)/2$. The kernel trick allows you to save and compute dot products in an n dimensional space.

- **Can we use any $K(.,.)$?** $f(\mathbf{x}) = \mathbf{T} \mathbf{h}_\theta \left(\sum_{\mathbf{z} \in \mathcal{M}} \mathbf{S}(\mathbf{z}) \mathbf{K}(\mathbf{x}, \mathbf{z}) \right)$
 - A function $K(x,z)$ is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space. $\mathbf{K}(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{i} \in \mathcal{I}} \mathbf{t}_i(\mathbf{z}) \mathbf{t}_i(\mathbf{x})$

- Take the **quadratic kernel:** $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$

- **Example: Direct construction (2 dimensional, for simplicity):**

- $\mathbf{K}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}_1 \mathbf{z}_1 + \mathbf{x}_2 \mathbf{z}_2)^2 = \mathbf{x}_1^2 \mathbf{z}_1^2 + 2\mathbf{x}_1 \mathbf{z}_1 \mathbf{x}_2 \mathbf{z}_2 + \mathbf{x}_2^2 \mathbf{z}_2^2$

- $= (\mathbf{x}_1^2, \sqrt{2} \mathbf{x}_1 \mathbf{x}_2, \mathbf{x}_2^2) (\mathbf{z}_1^2, \sqrt{2} \mathbf{z}_1 \mathbf{z}_2, \mathbf{z}_2^2)$

- $= (\mathbf{x})^T (\mathbf{z}) \rightarrow$ A dot product in an expanded space.

- It is not necessary to explicitly show the feature function .

- **General condition:** construct the Gram matrix $\{k(x_i, z_j)\}$; check that it's positive semi definite.

Kernel: Example

The Kernel Matrix

- The **Gram matrix** of a set of n vectors $S = \{\mathbf{x}_1 \dots \mathbf{x}_n\}$ is the $n \times n$ matrix \mathbf{G} with $\mathbf{G}_{ij} = \mathbf{x}_i \mathbf{x}_j$
 - The kernel matrix is the Gram matrix of $\{\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n)\}$
 - (size depends on the # of examples, not dimensionality)
- Direct option:
 - If you have the $\phi(\mathbf{x}_i)$, you have the Gram matrix (and it's easy to see that it will be positive semi-definite)
- Indirect:
 - If you have the Kernel, write down the Kernel matrix K_{ij} , and show that it is a legitimate kernel, without an explicit construction of $\phi(\mathbf{x}_i)$

Constructing New Kernels

- You can construct new kernels $k'(\mathbf{x}, \mathbf{x}')$ from existing ones:
 - Multiplying $k(\mathbf{x}, \mathbf{x}')$ by a constant c :
 $k'(\mathbf{x}, \mathbf{x}') = ck(\mathbf{x}, \mathbf{x}')$
 - Multiplying $k(\mathbf{x}, \mathbf{x}')$ by a function f applied to \mathbf{x} and \mathbf{x}' :
 $k'(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$
 - Applying a polynomial (with non-negative coefficients) to $k(\mathbf{x}, \mathbf{x}')$:
 $k'(\mathbf{x}, \mathbf{x}') = P(k(\mathbf{x}, \mathbf{x}'))$ with $P(z) = \sum_i a_i z^i$ and $a_i \geq 0$
 - Exponentiating $k(\mathbf{x}, \mathbf{x}')$:
 $k'(\mathbf{x}, \mathbf{x}') = \exp(k(\mathbf{x}, \mathbf{x}'))$

Summary – Kernel Based Methods

$$f(\mathbf{x}) = \mathbf{T}h_{\theta} \left(\sum_{\mathbf{z} \in \mathbf{M}} \mathbf{S}(\mathbf{z})\mathbf{K}(\mathbf{x}, \mathbf{z}) \right)$$

- A method to run Perceptron on a very large feature set, without incurring the cost of keeping a very large weight vector.
- Computing the weight vector can be done in the original feature space.
- **Notice:** this pertains only to **efficiency**: the classifier is identical to the one you get by blowing up the feature space.
- **Generalization** is still relative to the real dimensionality (or, related properties).
- Kernels were popularized by SVMs but apply to a range of models, Perceptron, Gaussian Models, PCAs, etc.

Efficiency-Generalization Tradeoff

- There is a **tradeoff between the computational efficiency** with which these kernels can be computed and the **generalization ability** of the classifier.
- For example, using such kernels the **Perceptron algorithm can make an exponential number of mistakes** even when learning simple functions. [Kharon,Roth,Servedio,NIPS'01; Ben David et al.]
- In addition, computing with kernels depends strongly on the number of examples. It turns out that **sometimes working in the blown up space is more efficient than using kernels.** [Cumby,Roth,ICML'03]

Explicit & Implicit Kernels: Complexity

- Is it always worthwhile to define kernels and work in the dual space?
- Computationally: [Cumby,Roth 2003]
 - Dual space – $t_1 m^2$ vs, Primal Space – $t_2 m$
 - Where m is # of examples, t_1, t_2 are the sizes of the (Dual, Primal) feature spaces, respectively.
 - Typically, $t_1 \ll t_2$, so it boils down to the **number of examples** one needs to consider relative to the growth in dimensionality.
- Rule of thumb: a lot of examples \rightarrow use Primal space
- Most applications today: People use **explicit** kernels. That is, they blow up the feature space explicitly.

Kernels: Generalization

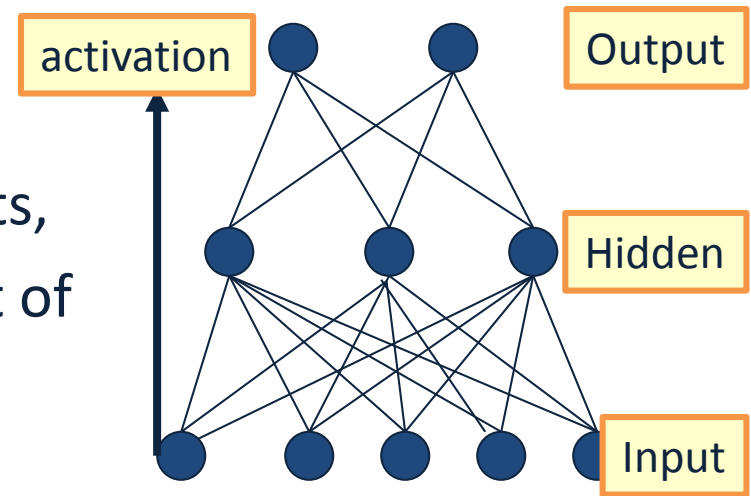
- Do we want to use the most expressive kernels we can?
 - (e.g., when you want to add quadratic terms, do you really want to add all of them?)
- No; this is equivalent to working in a larger feature space, and will lead to overfitting.
- Here is a simple argument that shows that simply adding irrelevant features does not help.

Kernels: Generalization(2)

- Given: A linearly separable set of points $S = \{x_1, \dots, x_n\} \in \mathbb{R}^n$ with separator $w \in \mathbb{R}^n$
- Embed S into a higher dimensional space $n' > n$, by adding zero-mean random noise e to the additional dimensions.
- Then $w' \cdot x = (w, 0) \cdot (x, e) = w \cdot x$
- So $w' \in \mathbb{R}^{n'}$ still separates S .
- We will now look at $\gamma / ||x||$ which we have shown to be inversely proportional to generalization (and mistake bound) ?
$$\gamma(S, w') / ||x'|| = \min_S w'^T x' / ||w'|| ||x'|| =$$
$$\min_S w^T x / ||w|| ||x'|| < \gamma(S, w') / ||x||$$
- Since $||x'|| = ||(x, e)|| > ||x||$
- The new ratio is larger, which implies generalization suffers.
- **Intuition:** adding a lot of noisy/irrelevant features cannot help

Multi-Layer Neural Network

- Multi-layer networks were designed to overcome the computational (expressivity) limitation of a single threshold element.
- The idea is to stack several layers of threshold elements, each layer using the output of the previous layer as input.
- Multi-layer networks can represent arbitrary functions, but building effective learning methods for such networks was [thought to be] difficult.

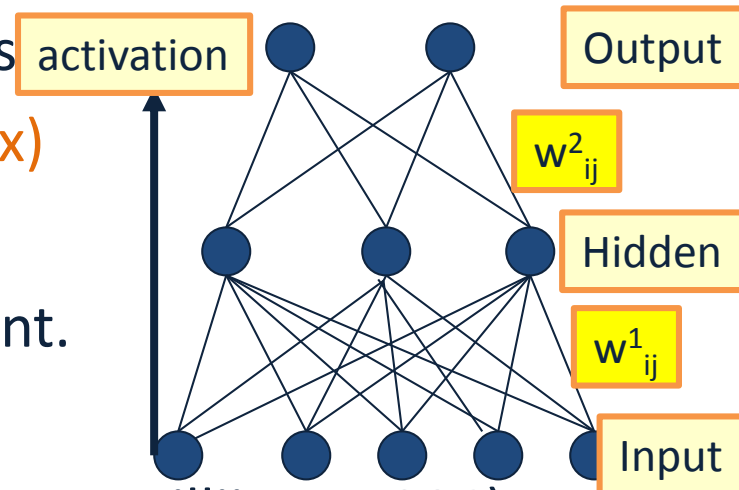


Basic Units

- **Linear Unit:** Multiple layers of linear functions $o_j = w \cdot x$ produce linear functions. We want to represent nonlinear functions

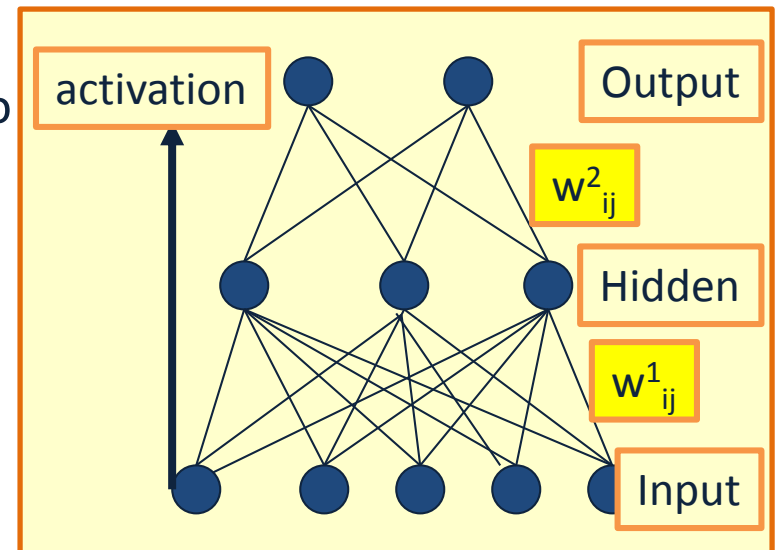
- **Threshold units:** $o_j = \text{sgn}(w \cdot x)$ are not differentiable, hence unsuitable for gradient descent.

- The key idea (Rumelhart, Hinton, William, 1986) was to notice that the discontinuity of the threshold element can be represented by a smooth non-linear approximation: $o_j = [1 + \exp\{-w \cdot x\}]^{-1}$



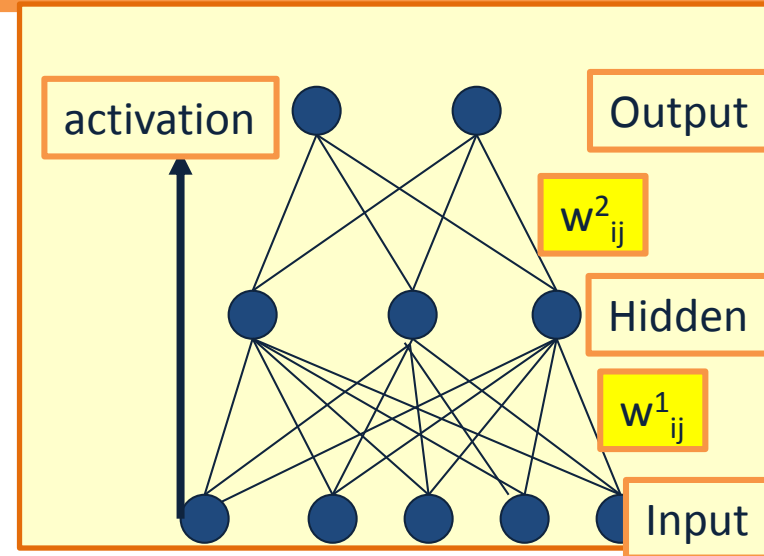
Learning with a Multi-Layer Perceptron

- It's easy to learn the top layer – it's just a linear unit.
- Given feedback (truth) at the top layer, and the activation at the layer below it, you can use the Perceptron update rule (more generally, gradient descent) to updated these weights.
- The problem is what to do with the other set of weights – we do not get feedback in the intermediate layer(s).



Learning with a Multi-Layer Perceptron

- The problem is what to do with the other set of weights – we do not get feedback in the intermediate layer(s).
- **Solution:** If all the activation functions are differentiable, then the output of the network is also a differentiable function of the input and weights in the network.
- Define an **error function** (e.g., sum of squares) that is a differentiable function of the output, that this error function is also a differentiable function of the weights.
- We can then evaluate the derivatives of the error with respect to the weights, and use these derivatives to find weight values that minimize this error function. This can be done, for example, using gradient descent (or other optimization methods).
- This results in an algorithm called back-propagation.



Computational Learning Theory

- What general laws constrain inductive learning ?
 - What learning problems can be solved ?
 - When can we trust the output of a learning algorithm ?
- We seek theory to relate
 - Probability of successful Learning
 - Number of training examples
 - Complexity of hypothesis space
 - Accuracy to which target concept is approximated
 - Manner in which training examples are presented

Recall what we did earlier:

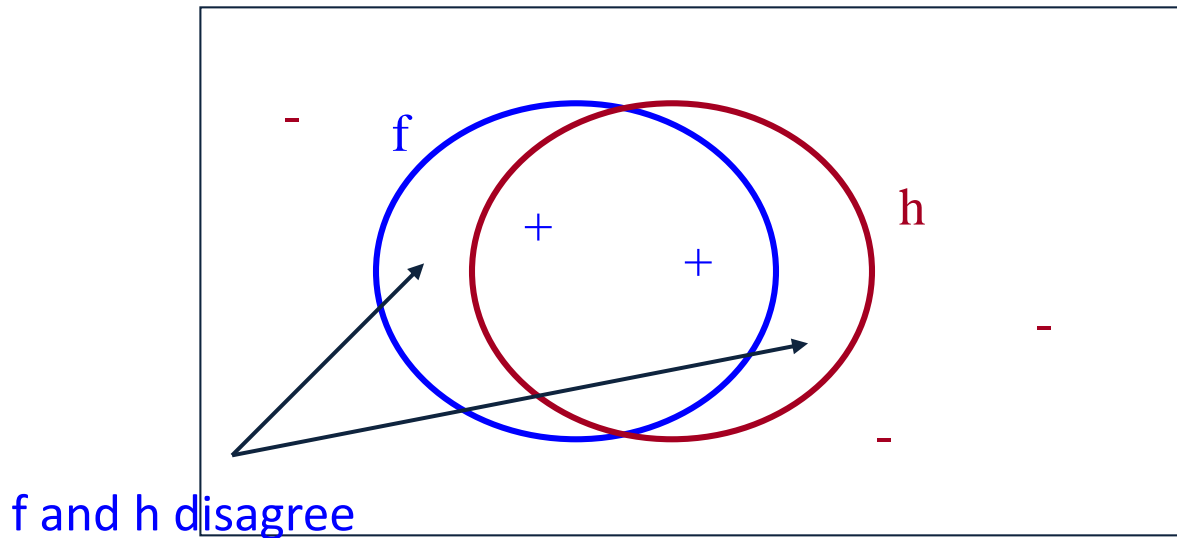
Quantifying Performance

- We want to be able to say something rigorous about the performance of our learning algorithm.
- We will concentrate on discussing the number of examples one needs to **see** before we can say that our learned hypothesis is good.

PAC Learning – Intuition

- We have seen many examples (drawn according to D)
- Since in all the positive examples x_1 was active, it is **very likely** that it will be active in future positive examples
- If not, in any case, x_1 is active only in a small percentage of the examples so our error will be small

$$\text{Error}_D = \Pr_{x \in D} [f(x) \neq h(x)]$$



$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Formulating Prediction Theory

- Instance Space X , Input to the Classifier; Output Space $Y = \{-1, +1\}$
- Making predictions with: $h: X \rightarrow Y$
- D : An unknown distribution over $X \times Y$
- S : A set of examples drawn independently from D ; $m = |S|$, size of sample.

Now we can define:

- True Error: $\text{Error}_D = \Pr_{(x,y) \in D} [h(x) \neq y]$
- Empirical Error: $\text{Error}_S = \Pr_{(x,y) \in S} [h(x) \neq y] = \frac{1}{m} \sum_{i=1}^m [h(x_i) \neq y_i]$
 - (Empirical Error (Observed Error, or Test/Train error, depending on S))

This will allow us to ask: (1) Can we describe/bound Error_D given Error_S ?

- Function Space: C – A set of possible target concepts; target is: $f: X \rightarrow Y$
- Hypothesis Space: H – A set of possible hypotheses
- This will allow us to ask: (2) Is C learnable?
 - Is it possible to learn a given function in C using functions in H , given the supervised protocol?

Probably Approximately Correct

- Cannot expect a learner to learn a concept **exactly**.
- Cannot always expect to learn a **close approximation** to the target concept
- Therefore, the only realistic expectation of a good learner is that **with high probability** it will learn a **close approximation** to the target concept.
- In **Probably Approximately Correct (PAC)** learning, one requires that given small parameters ϵ and δ , with probability at least $(1 - \delta)$ a learner produces a hypothesis with **error at most ϵ**
- The reason we can hope for that is the **Consistent Distribution** assumption.

PAC Learnability

- Consider a concept class C defined over an instance space X (containing instances of length n), and a learner L using a hypothesis space H .
- C is PAC learnable by L using H if
 - for all $f \in C$,
 - for all distributions D over X , and fixed $0 < \epsilon, \delta < 1$,
- L , given a collection of m examples sampled independently according to D produces
 - with probability at least $(1 - \delta)$ a hypothesis $h \in H$ with error at most ϵ , ($\text{Error}_D = \Pr_D[f(x) \neq h(x)]$)
- where m is polynomial in $1/\epsilon$, $1/\delta$, n and $\text{size}(H)$
- C is efficiently learnable if L can produce the hypothesis in time polynomial in $1/\epsilon$, $1/\delta$, n and $\text{size}(H)$

Definition

PAC Learnability

- We impose two limitations:
- Polynomial **sample complexity** (information theoretic constraint)
 - Is there enough information in the sample to distinguish a hypothesis h that approximate f ?
- Polynomial **time complexity** (computational complexity)
 - Is there an efficient algorithm that can process the sample and produce a good hypothesis h ?
- To be PAC learnable, there must be a hypothesis $h \in H$ with arbitrary small error **for every** $f \in C$. We generally assume $H \supseteq C$. (Properly PAC learnable if $H=C$)
- **Worst Case definition**: the algorithm must meet its accuracy
 - for **every** distribution (The distribution free assumption)
 - for **every** target function f in the class C

Comments

Occam's Razor (1)

Claim: The probability that there exists a hypothesis $h \in H$ that
(1) is consistent with m examples and
(2) satisfies $\text{error}(h) > \varepsilon$ ($\text{Error}_D(h) = \Pr_{x \in D} [f(x) \neq h(x)]$)
is less than $|H|(1 - \varepsilon)^m$.

Proof: Let h be such a bad hypothesis.

- The probability that h is consistent with one example of f is

$$\Pr_{x \in D} [f(x) = h(x)] < 1 - \varepsilon$$

- Since the m examples are drawn independently of each other,
The probability that h is consistent with m example of f is less than $(1 - \varepsilon)^m$
- The probability that *some* hypothesis in H is consistent with m examples
is less than $|H| (1 - \varepsilon)^m$

Note that we don't need a true f for this argument; it can be done with h , relative to a distribution over $X \times Y$.

Occam's Razor (1)

We want this probability to be smaller than δ , that is:

$$|H|(1-\epsilon)^m < \delta$$

$$\ln(|H|) + m \ln(1-\epsilon) < \ln(\delta)$$

What do we know now about the Consistent Learner scheme?

(with $e^{-x} = 1-x+x^2/2+\dots$; $e^{-x} > 1-x$; $\ln(1-\epsilon) < -\epsilon$; gives a safer δ)

$$m > \frac{1}{\epsilon} \{ \ln(|H|) + \ln(1/\delta) \}$$

We showed that a m -consistent hypothesis generalizes well ($\text{err} < \epsilon$)
(Appropriate m is a function of $|H|, \epsilon, \delta$)

(gross over estimate)

It is called Occam's razor, because it indicates a preference towards small hypothesis spaces

What kind of hypothesis spaces do we want? Large? Small?

To guarantee consistency we need $H \supseteq C$. But do we want the smallest H possible?

Consistent Learners

- Immediately from the definition, we get the following general scheme for PAC learning:

- Given a sample D of m examples
 - Find some $h \in H$ that is consistent with all m examples
 - We showed that if m is large enough, a consistent hypothesis must be close enough to f
 - Check that m is not too large (polynomial in the relevant parameters) : we showed that the “closeness” guarantee requires that
$$m > 1/(\ln |H| + \ln 1/\delta)$$
 - Show that the consistent hypothesis $h \in H$ can be computed efficiently

- In the case of conjunctions

- We used the Elimination algorithm to find a hypothesis h that is consistent with the training set (easy to compute)
- We showed directly that if we have sufficiently many examples (polynomial in the parameters), then h is close to the target function.

We did not need to show it directly.
See above.

Computational Complexity

- Determining whether there is a **2-term DNF** consistent with a set of training data is NP-Hard
- Therefore the class of **k-term-DNF** is **not** efficiently (properly) PAC learnable due to computational complexity
- We have seen an algorithm for learning **k-CNF**.
- And, **k-CNF** is a superset of **k-term-DNF**
 - (That is, every k-term-DNF can be written as a k-CNF)
- Therefore, **C=k-term-DNF** can be learned as using **H=k-CNF** as the hypothesis Space
- **Importance of representation:**
 - **Concepts that cannot be learned using one representation can be learned using another (more expressive) representation.**

This result is analogous to an earlier observation that it's better to learn linear separators than conjunctions.

Negative Results – Examples

- Two types of nonlearnability results:
- **Complexity Theoretic**
 - Showing that various concepts classes cannot be learned, based on well-accepted assumptions from computational complexity theory.
 - E.g. : C cannot be learned unless $P=NP$
- **Information Theoretic**
 - The concept class is sufficiently rich that a polynomial number of examples may not be sufficient to distinguish a particular target concept.
 - Both type involve “representation dependent” arguments.
 - The proof shows that a given class cannot be learned by algorithms using hypotheses from the same class. (So?)
- Usually proofs are for EXACT learning, but apply for the distribution free case.

Agnostic Learning

- Assume we are trying to learn a concept f using hypotheses in H , but $f \notin H$
- In this case, our goal should be to find a hypothesis $h \in H$, with a small training error:

$$Err_{TR}(h) = \frac{1}{m} |\{x \in \text{training _ examples}; f(x) \neq h(x)\}|$$

- We want a guarantee that a hypothesis with a small training error will have a good accuracy on unseen examples

- $Err_D(h) = \Pr_{x \in D}[f(x) \neq h(x)]$

- **Hoeffding bounds** characterize the deviation between the true probability of some event and its observed frequency over m independent trials. $\Pr[p > \hat{p} + \varepsilon] < e^{-2m\varepsilon^2}$

- (p is the underlying probability of the binary variable (e.g., toss is Head) being 1)

Agnostic Learning

- Therefore, the probability that an element in H will have training error which is off by more than ε can be bounded as follows:

$$\Pr[Err_D(h) > Err_{TR}(h) + \varepsilon] < e^{-2m\varepsilon^2}$$

- Doing the same union bound game as before, with $\delta = |H|e^{-2m\varepsilon^2}$
- We get a **generalization bound** – a bound on how much will the true error E_D deviate from the observed (training) error E_{TR} .
- For any distribution D generating training and test instances, with probability at least $1-\delta$ over the choice of the training set of size m , (drawn IID), for all $h \in H$

$$Error_D(h) < Error_{TR}(h) + \sqrt{\frac{\log|H| + \log(1/\delta)}{2m}}$$

Agnostic Learning

- An agnostic learner which makes no commitment to whether f is in H and returns the hypothesis with least training error over at least the following number of examples m can guarantee with probability at least $(1-\delta)$ that its training error is not off by more than ϵ from the true error.

$$m > \frac{1}{2\epsilon^2} \{ \ln(|H|) + \ln(1/\delta) \}$$

Learnability depends on the log of the size of the hypothesis space

Infinite Hypothesis Space

- The previous analysis was restricted to finite hypothesis spaces
- Some infinite hypothesis spaces are more expressive than others
 - E.g., Rectangles, vs. 17- sides convex polygons vs. general convex polygons
 - Linear threshold function vs. a conjunction of LTUs
- Need a measure of the expressiveness of an infinite hypothesis space other than its size
- The Vapnik-Chervonenkis dimension (**VC dimension**) provides such a measure.
- Analogous to $|H|$, there are bounds for sample complexity using **VC(H)**

Shattering

- We say that a set S of examples is **shattered** by a set of functions H if for every partition of the examples in S into positive and negative examples there is a function in H that gives exactly these labels to the examples
(Intuition: A rich set of functions shatters large sets of points)

Left bounded intervals on the real axis: $[0, a)$, for some real number $a > 0$



Sets of **two** points cannot be shattered

(we mean: given two points, you can label them in such a way that no concept in this class will be consistent with their labeling)

VC Dimension

- We say that a set S of examples is shattered by a set of functions H if for every partition of the examples in S into positive and negative examples there is a function in H that gives exactly these labels to the examples
- The VC dimension of hypothesis space H over instance space X is the size of the largest finite subset of X that is shattered by H .

Even if only one subset of this size does it!
- If there exists a subset of size d that can be shattered, then $VC(H) \geq d$
- If no subset of size d can be shattered, then $VC(H) < d$

$$\underline{VC(\text{Half intervals}) = 1}$$

(no subset of size 2 can be shattered)

$$\underline{VC(\text{Intervals}) = 2}$$

(no subset of size 3 can be shattered)

$$\underline{VC(\text{Half-spaces in the plane}) = 3}$$

(no subset of size 4 can be shattered)

Some are shattered, but some are

Sample Complexity & VC Dimension

- Using $VC(H)$ as a measure of expressiveness we have an Occam algorithm for infinite hypothesis spaces.

- Given a sample D of m examples
- Find some $h \in H$ that is consistent with all m examples

• If

$$m > \frac{1}{\epsilon} \left\{ 8VC(H) \log \frac{13}{\epsilon} + 4 \log \left(\frac{2}{\delta} \right) \right\}$$

- Then with probability at least $(1-\delta)$, h has error less than ϵ .

(that is, if m is polynomial we have a PAC learning algorithm; to be efficient, we need to produce the hypothesis h efficiently.

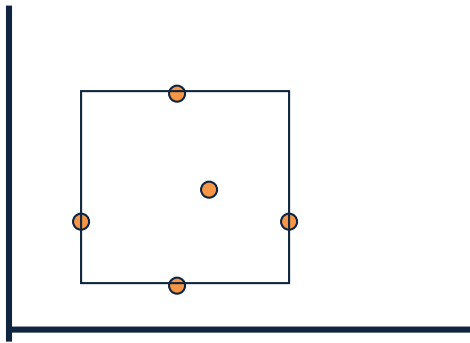
What if H is finite?

- Notice that to shatter m examples it must be that: $|H| > 2^m$, so $\log(|H|) \geq VC(H)$

Learning Rectangles

- Consider axis parallel rectangles in the real plan
- Can we PAC learn it ?
 - (1) What is the VC dimension ?

- But, no five instances can be shattered



Therefore $VC(H) = 4$

There can be at most 4 distinct extreme points (smallest or largest along some dimension) and these cannot be included (labeled +) without including the 5th point.

As far as sample complexity, this guarantees PAC learnability.

Sample Complexity Lower Bound

- There is also a general lower bound on the minimum number of examples necessary for PAC learning in the general case.
- Consider any concept class C such that $VC(C) > 2$, any learner L and small enough ϵ, δ .

Then, there exists a distribution D and a target function in C such that if L observes less than

$$m = \max\left[\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right), \frac{VC(C) - 1}{32\epsilon}\right]$$

examples, then with probability at least δ , L outputs a hypothesis having $\text{error}(h) > \epsilon$.

Ignoring constant factors, the lower bound is the same as the upper bound, except for the extra $\log(1/\epsilon)$ factor in the upper bound.

Boosting

- Boosting is (today) a general learning paradigm for putting together a Strong Learner, given a collection (possibly infinite) of Weak Learners.
- The original Boosting Algorithm was proposed as an answer to a theoretical question in PAC learning. [The Strength of Weak Learnability; Schapire, 89]
- Consequently, Boosting has interesting theoretical implications, e.g., on the relations between PAC learnability and compression.
 - If a concept class is efficiently PAC learnable then it is efficiently PAC learnable by an algorithm whose required memory is bounded by a polynomial in n , size c and $\log(1/\epsilon)$.
 - There is no concept class for which efficient PAC learnability requires that the entire sample be contained in memory at one time – there is always another algorithm that “forgets” most of the sample.

The Boosting Approach

■ Algorithm

- ❑ Select a small subset of examples
- ❑ Derive a rough rule of thumb
- ❑ Examine 2nd set of examples
- ❑ Derive 2nd rule of thumb
- ❑ Repeat T times
- ❑ Combine the learned rules into a single hypothesis

■ Questions:

- ❑ How to choose subsets of examples to examine on each round?
- ❑ How to combine all the rules of thumb into single prediction rule?

■ Boosting

- ❑ General method of converting rough rules of thumb into highly accurate prediction rule

A Formal View of Boosting

- Given **training set** $(x_1, y_1), \dots (x_m, y_m)$
- $y_i \in \{-1, +1\}$ is the correct label of instance $x_i \in X$
- For $t = 1, \dots, T$
 - Construct a **distribution** D_t on $\{1, \dots, m\}$
 - Find **weak hypothesis** (“rule of thumb”)
$$h_t : X \rightarrow \{-1, +1\}$$
with small error ϵ_t on D_t :
$$\epsilon_t = \Pr_D [h_t(x_i) \neq y_i]$$
- Output: **final hypothesis** H_{final}

Adaboost

■ Constructing D_t on $\{1, \dots, m\}$:

□ $D_1(i) = 1/m$

□ Given D_t and h_t :

□ $D_{t+1}(i) = D_t(i)/z_t e^{-\alpha_t}$

$D_t(i)/z_t e^{+\alpha_t}$

$= D_t(i)/z_t \exp(-\alpha_t y_i h_t(x_i))$

where $z_t =$ normalization constant

and

$\alpha_t = \frac{1}{2} \ln\left\{ \frac{1 - \epsilon_t}{\epsilon_t} \right\}$

Think about unwrapping it all the way to $1/m$

$$Z_t = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

if $y_i = h_t(x_i)$

< 1 ; smaller weight

if $y_i \neq h_t(x_i)$

> 1 ; larger weight

Notes about α_t :

$e^{+\alpha_t} = \sqrt{(1 - \epsilon_t)/\epsilon_t} > 1$

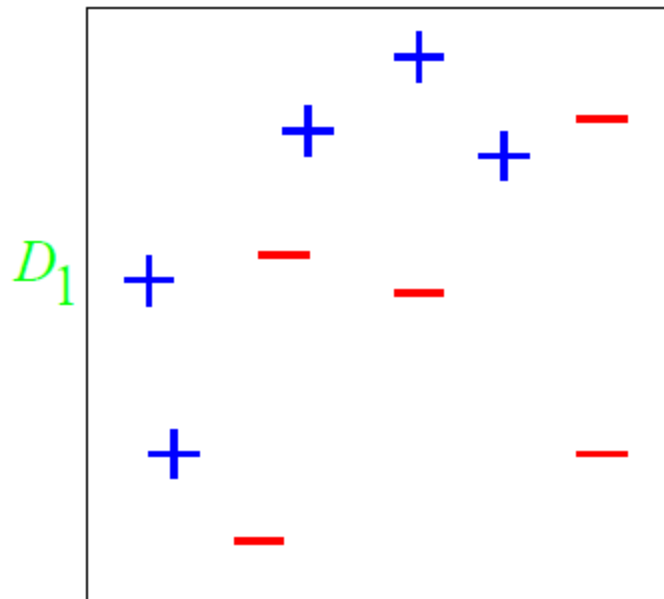
□ Positive due to the weak learning assumption

□ Examples that we predicted correctly are demoted, others promoted

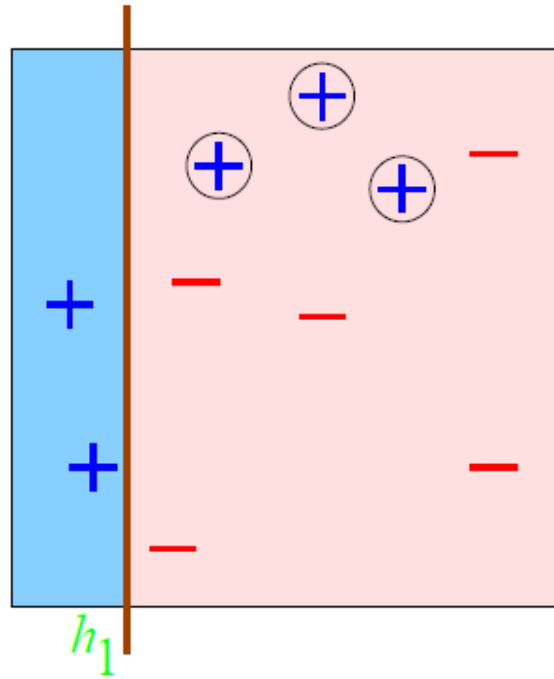
□ Sensible weighting scheme: better hypothesis (smaller error) \rightarrow larger weight

■ Final hypothesis: $H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t h_t(x)\right)$

A Toy Example

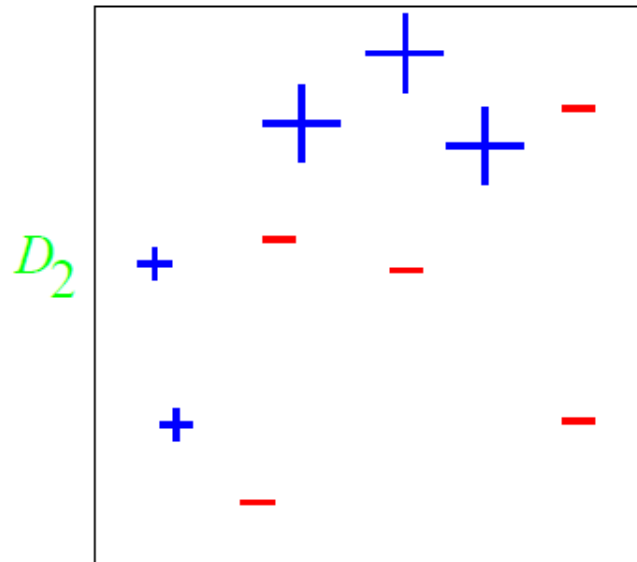


Round 1

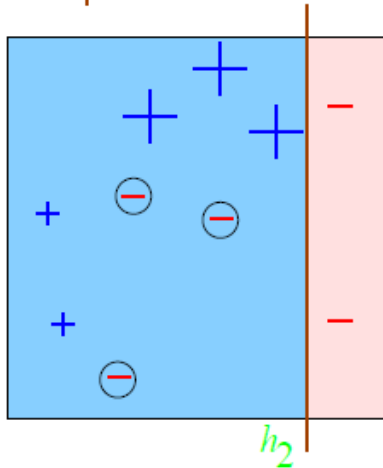
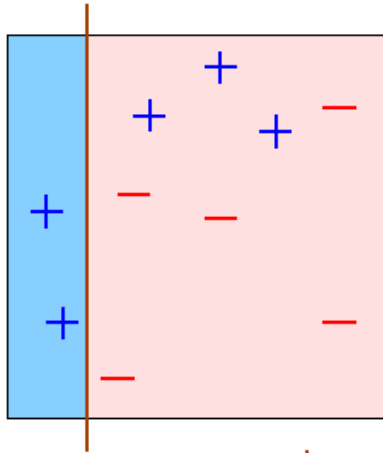


$$\epsilon_1 = 0.30$$

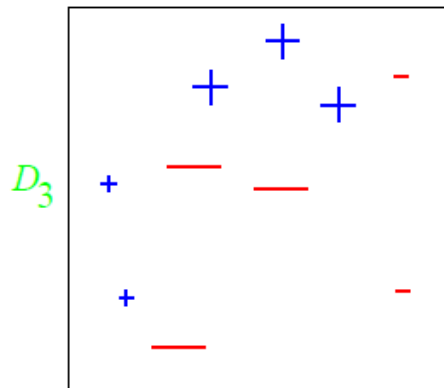
$$\alpha_1 = 0.42$$



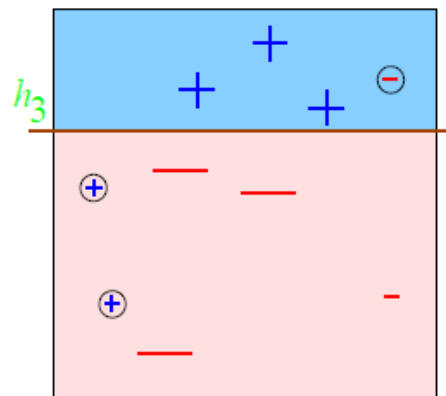
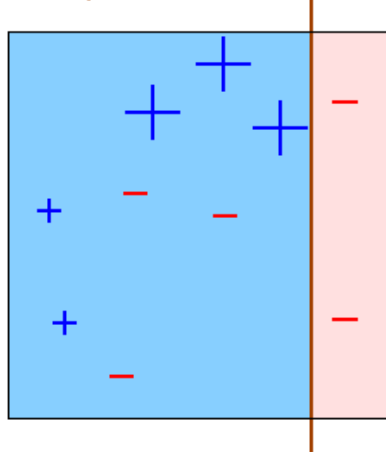
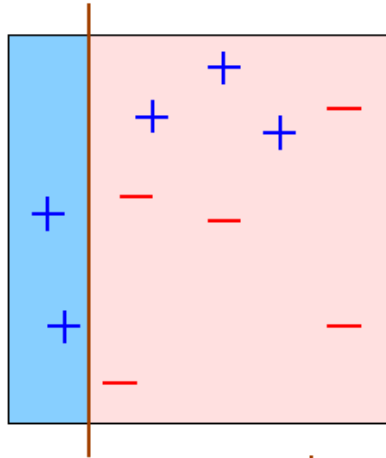
Round 2



$\epsilon_2=0.21$
 $\alpha_2=0.65$



Round 3



$\epsilon_3=0.14$

$\alpha_3=0.92$

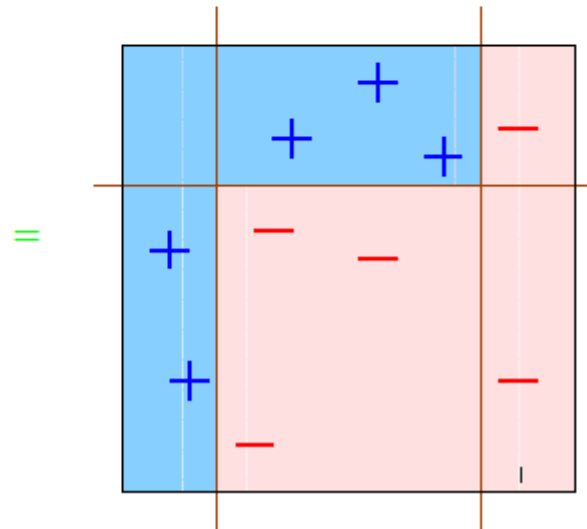
A Toy Example

A cool and important note about the final hypothesis: it is possible that the combined hypothesis makes no mistakes on the training data, but boosting can still learn, by adding more weak hypotheses.

Final Hypothesis

H_{final}

$$= \text{sign} \left(0.42 \left[\begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \\ \hline \end{array} \right] + 0.65 \left[\begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \\ \hline \end{array} \right] + 0.92 \left[\begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \\ \hline \end{array} \right] \right)$$



Summary of Ensemble Methods

- Boosting
- Bagging
- Random Forests

Boosting

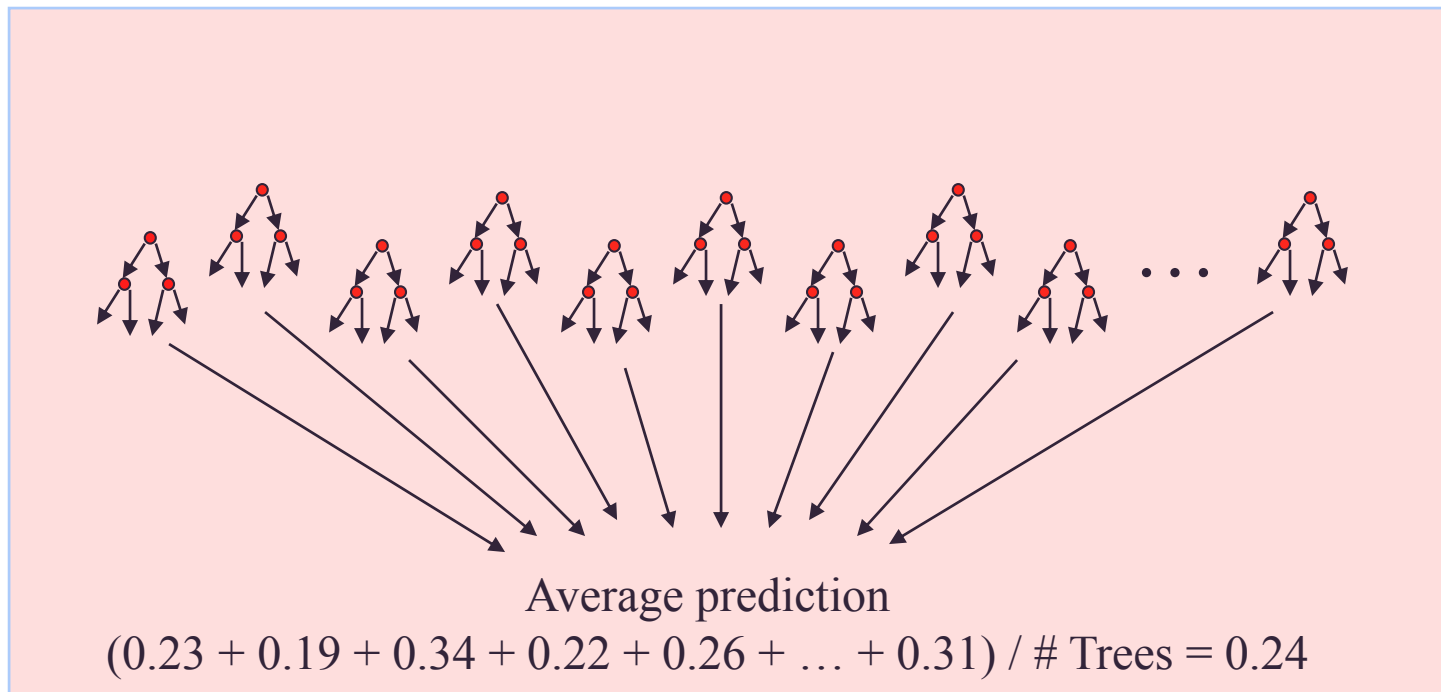
- Initialization:
 - Weigh all training samples equally
- Iteration Step:
 - Train model on (weighted) train set
 - Compute error of model on train set
 - Increase weights on training cases model gets wrong!!!
- Typically requires 100's to 1000's of iterations
- Return final model:
 - Carefully weighted prediction of each model

Bagging

- Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor.
- The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class.
- The **multiple versions** are formed by making **bootstrap replicates** of the learning set and using these as new learning sets.
 - That is, use samples of the data, with repetition
- Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy.
- The vital element is the **instability of the prediction** method. If perturbing the learning set can cause significant changes in the predictor constructed then bagging can improve accuracy.

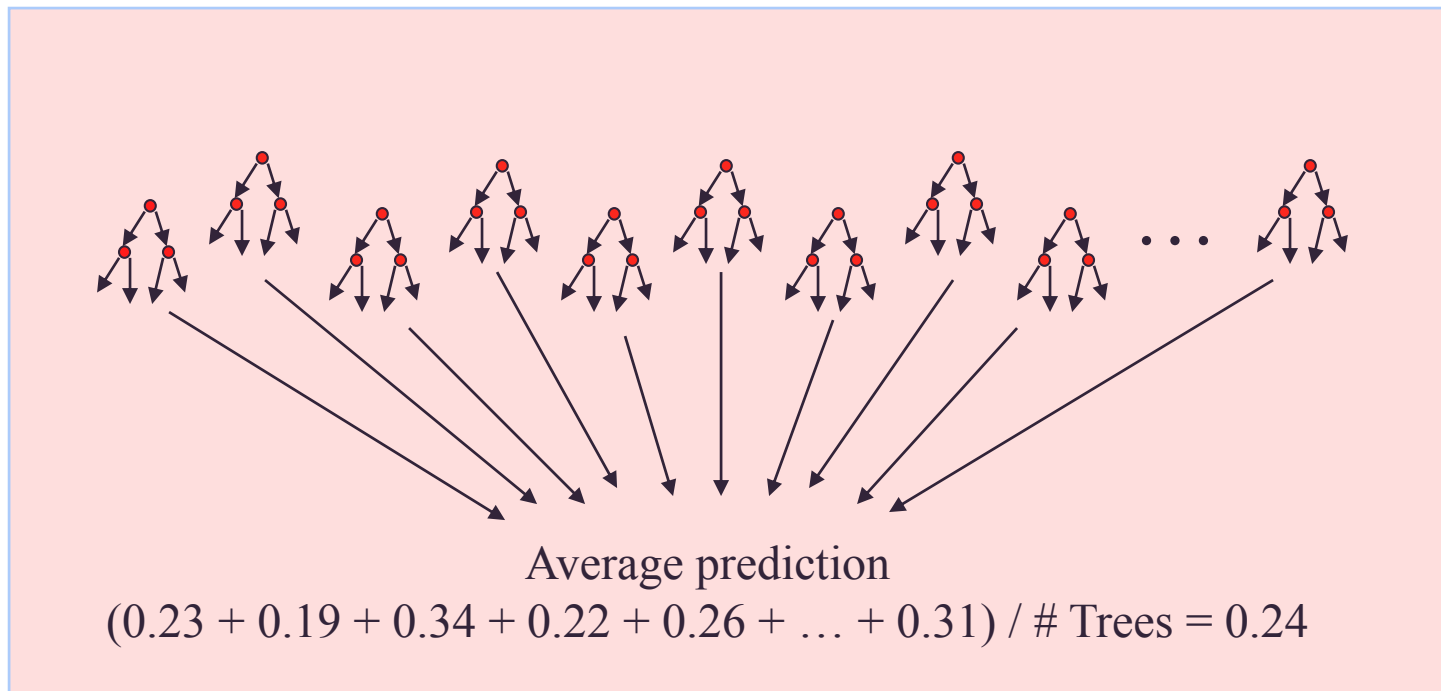
Example: Bagged Decision Trees

- Draw 100 bootstrap samples of data
- Train trees on each sample \rightarrow 100 trees
- Average prediction of trees on out-of-bag samples



Random Forests (Bagged Trees++)


- Draw **1000+** bootstrap samples of data
- **Draw sample of available attributes at each split**
- Train trees on each sample/attribute set → **1000+** trees
- Average prediction of trees on out-of-bag samples



Classification

- So far we focused on Binary Classification
- For linear models:
 - Perceptron, Winnow, SVM, GD, SGD
- The prediction is simple:
 - Given an example x ,
 - Prediction = $\text{sgn}(w^T x)$
 - Where w is the learned model
- The output is a single bit

Multi-Categorical Output Tasks

- 
- Multi-class Classification ($y \in \{1, \dots, K\}$)
 - character recognition ('6')
 - document classification ('homepage')
 - Multi-label Classification ($y \subseteq \{1, \dots, K\}$)
 - document classification ('(homepage, facultypage)')
 - Category Ranking ($y \in \pi K$)
 - user preference ('(love > like > hate)')
 - document classification ('hompager > facultypage > sports')
 - Hierarchical Classification ($y \subseteq \{1, \dots, K\}$)
 - cohere with class hierarchy
 - place document into index where 'soccer' is-a 'sport'

Setting

- Learning:
 - Given a data set $D = \{(x_i, y_i)\}_1^m$
 - Where $x_i \in \mathbb{R}^n$, $y_i \in \{1, 2, \dots, k\}$.
- Prediction (inference):
 - Given an example x , and a learned function (model),
 - Output a single class labels y .

Binary to Multiclass

- Most schemes for multiclass classification work by reducing the problem to that of binary classification.
- There are multiple ways to decompose the multiclass prediction into multiple binary decisions
 - One-vs-all
 - All-vs-all
 - Error correcting codes
- We will then talk about a more general scheme:
 - Constraint Classification
- It can be used to model other non-binary classification and leads to **Structured Prediction**.

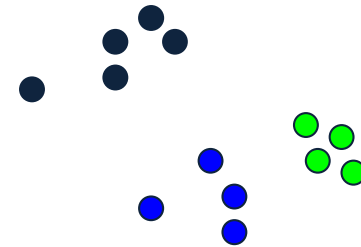
One-Vs-All

- **Assumption:** Each class can be separated from **all the rest** using a binary classifier in the hypothesis space.
- **Learning:** Decomposed to learning **k** independent binary classifiers, one for each class label.
- **Learning:**
 - Let **D** be the set of training examples.
 - \forall label **l**, construct a binary classification problem as follows:
 - Positive examples: Elements of **D** with label **l**
 - Negative examples: All other elements of **D**
 - This is a binary learning problem that we can solve, producing **k** binary classifiers w_1, w_2, \dots, w_k
- **Decision:** Winner Takes All (WTA):
 - $$f(x) = \operatorname{argmax}_i w_i \cdot x$$

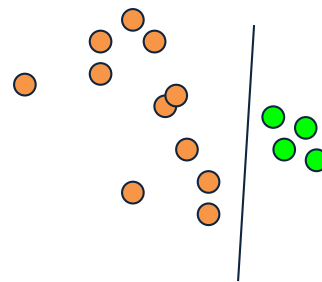
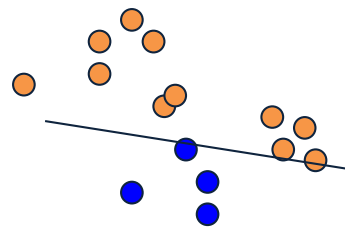
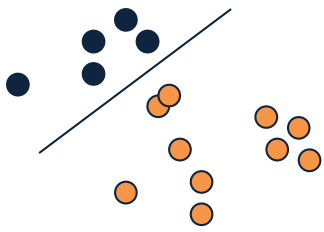
Solving MultiClass with 1vs All learning

- MultiClass classifier

 - Function $f : \mathbb{R}^n \rightarrow \{1,2,3,\dots,k\}$



- Decompose into binary problems



- Not always possible to learn

- No theoretical justification

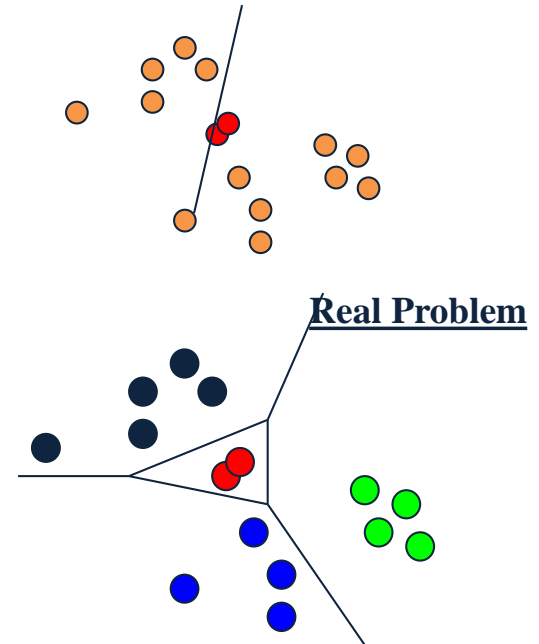
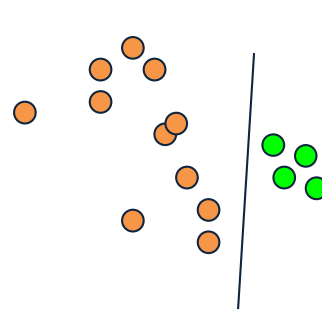
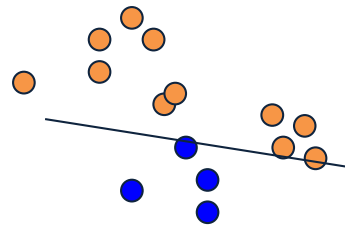
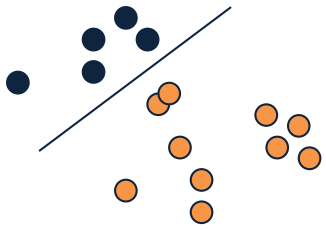
 - Need to make sure the range of all classifiers is the same

- (unless the problem is easy)

Learning via One-Versus-All (OvA) Assumption

- Find $v_r, v_b, v_g, v_y \in \mathbb{R}^n$ such that
 - $v_r \cdot x > 0$ iff $y = \text{red}$ \otimes
 - $v_b \cdot x > 0$ iff $y = \text{blue}$ \checkmark
 - $v_g \cdot x > 0$ iff $y = \text{green}$ \checkmark
 - $v_y \cdot x > 0$ iff $y = \text{yellow}$ \checkmark
- Classification: $f(x) = \text{argmax}_i v_i \cdot x$

$$\underline{H = \mathbb{R}^{nk}}$$



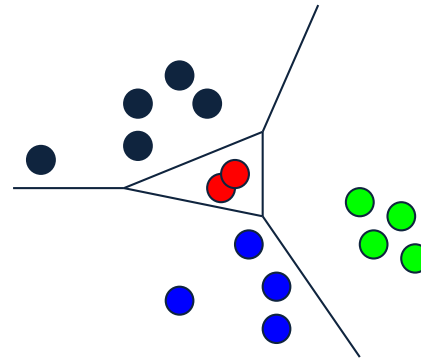
All-Vs-All

- **Assumption:** There is a separation between **every pair** of classes using a binary classifier in the hypothesis space.
- **Learning:** Decomposed to learning $\binom{k}{2} \sim k^2$ independent binary classifiers, one corresponding to each pair of class labels. For the pair (i, j) :
 - **Positive example:** all examples with label i
 - **Negative examples:** all examples with label j
- **Decision:** More involved, since output of binary classifier may not cohere. Each label gets $k-1$ votes.
- **Decision Options:**
 - **Majority:** classify example x to take label i if i wins on x more often than j ($j=1, \dots, k$)
 - **A tournament:** start with $n/2$ pairs; continue with winners .

Learning via All-Verses-All (AvA) Assumption

■ Find $v_{rb}, v_{rg}, v_{ry}, v_{bg}, v_{by}, v_{gy} \in \mathbb{R}^d$ such that

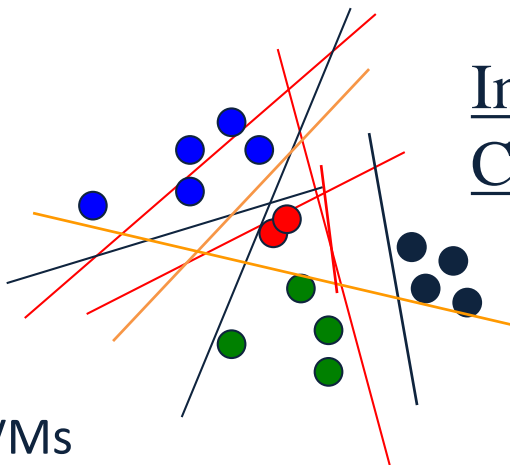
- $v_{rb} \cdot x > 0$ if $y = \text{red}$
 < 0 if $y = \text{blue}$
- $v_{rg} \cdot x > 0$ if $y = \text{red}$
 < 0 if $y = \text{green}$
- ... (for all pairs)



It is possible to separate all k classes with the $O(k^2)$ classifiers

$$\underline{H = R^{kkn}}$$

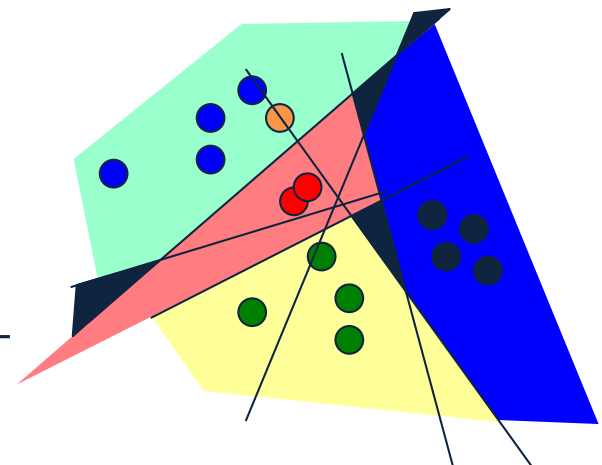
How to classify?



Individual Classifiers

SVMs

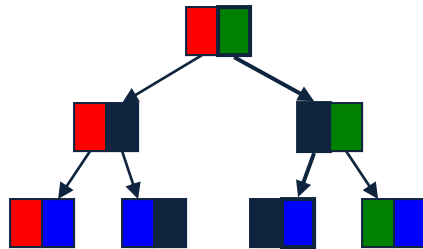
Decision Regions



CS446 Fall '16

Classifying with AvA

Tournament



Majority Vote



1 red, 2 yellow, 2 green

→ ?

All are post-learning and *might* cause weird stuff

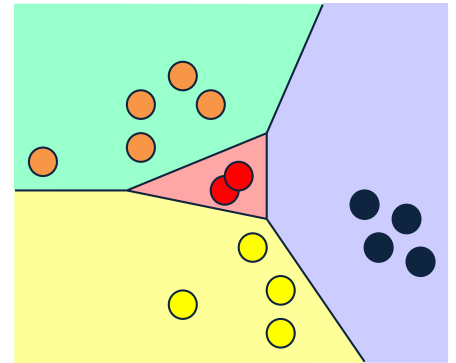
One-vs-All vs. All vs. All

- Assume m examples, k class labels.
 - For simplicity, say, m/k in each.
- One vs. All:
 - classifier f_i : m/k (+) and $(k-1)m/k$ (-)
 - Decision:
 - Evaluate k linear classifiers and do Winner Takes All (WTA):
 - $$f(x) = \operatorname{argmax}_i f_i(x) = \operatorname{argmax}_i w_i^T x$$
- All vs. All:
 - Classifier f_{ij} : m/k (+) and m/k (-)
 - More expressivity, but less examples to learn from.
 - Decision:
 - Evaluate k^2 linear classifiers; decision sometimes unstable.
- What type of learning methods would prefer All vs. All (efficiency-wise)?

(Think about Dual/Primal)

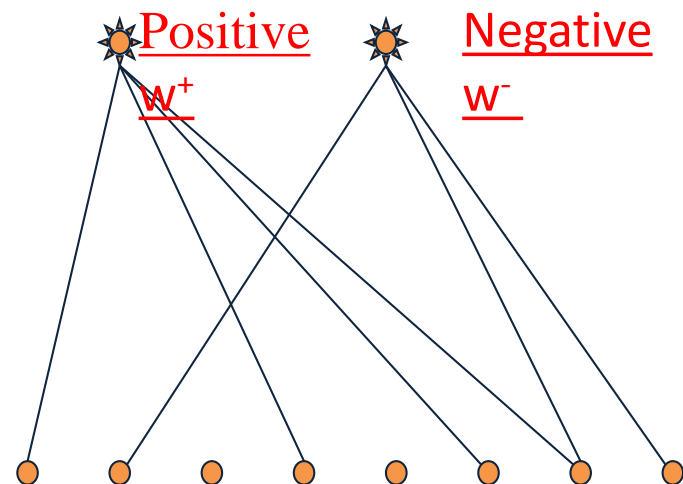
Problems with Decompositions

- Learning optimizes over *local* metrics
 - Does not guarantee good *global* performance
 - We don't care about the performance of the *local* classifiers
- Poor decomposition \Rightarrow poor performance
 - Difficult local problems
 - Irrelevant local problems
- Especially true for Error Correcting Output Codes
 - Another (class of) decomposition
 - Difficulty: how to make sure that the resulting problems are separable.
- Efficiency: e.g., All vs. All vs. One vs. All
- Former has advantage when working with the dual space.
- Not clear how to generalize multi-class to problems with a very large # of output.



Recall: Winnow's Extensions

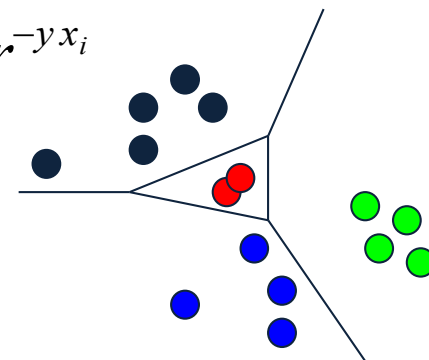
- Winnow learns **monotone Boolean functions**
- We extended to general Boolean functions via
- “Balanced Winnow”
 - 2 weights per variable;
 - **Decision**: using the “effective weight”, the difference between w^+ and w^-
 - This is equivalent to the Winner take all decision
 - **Learning**: In principle, it is possible to use the 1-vs-all rule and update each set of n weights **separately**, but we suggested the “balanced” Update rule that takes into account how both sets of n weights predict on example \mathbf{x}



$$\text{If } [(\mathbf{w}^+ - \mathbf{w}^-) \bullet \mathbf{x} \geq \theta] \neq y, \quad w_i^+ \leftarrow w_i^+ r^{y x_i}, \quad w_i^- \leftarrow w_i^- r^{-y x_i}$$

Can this be generalized to the case of k labels, $k > 2$?

We need a “global” learning approach



Extending Balanced

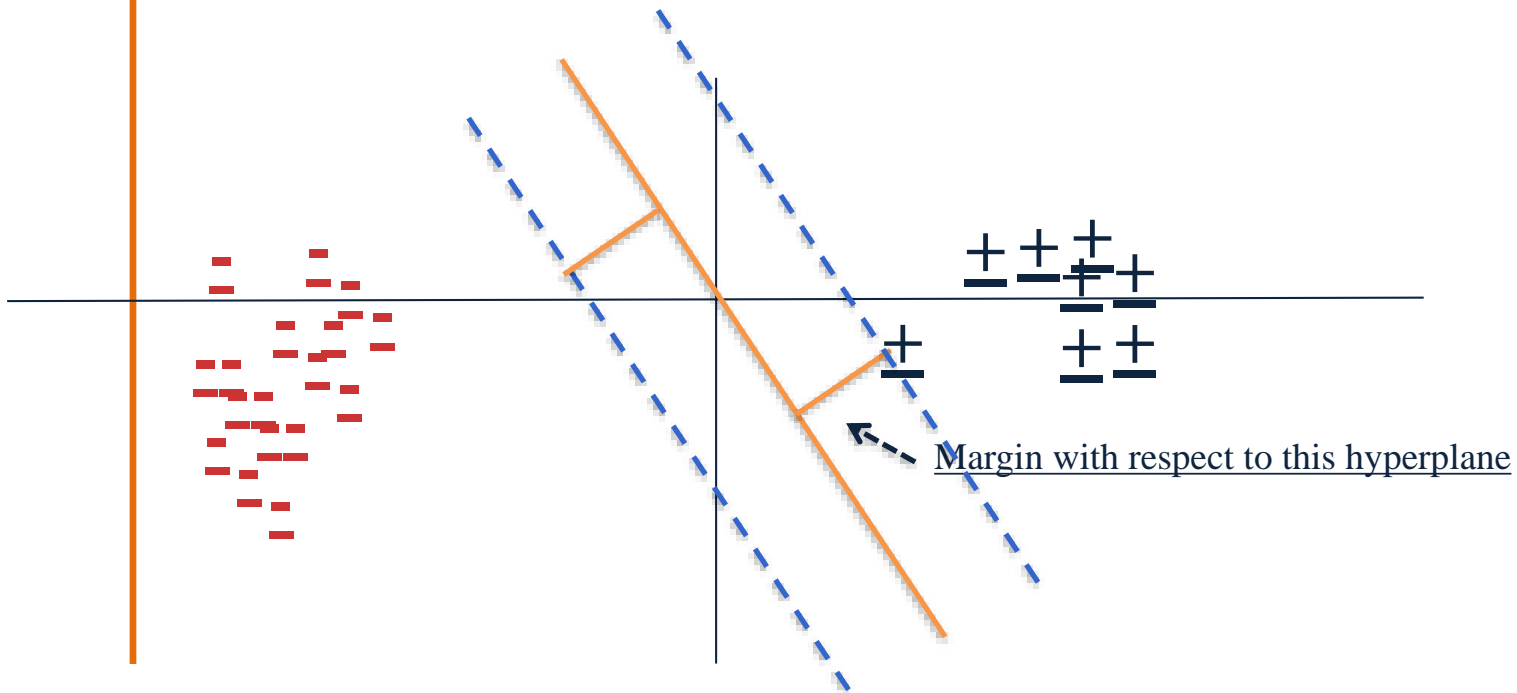
- In a 1-vs-all training you have a target node that represents **positive** examples and target node that represents **negative** examples.
- Typically, we train each node separately (mine/not-mine example).
- Rather, given an example we could say: this is more a **+** example than a **-** example.

$$\text{If } [(\mathbf{w}^+ - \mathbf{w}^-) \bullet \mathbf{x} \geq \theta] \neq y, \quad w_i^+ \leftarrow w_i^+ r^{yx_i}, \quad w_i^- \leftarrow w_i^- r^{-yx_i}$$

- We compared the activation of the different target nodes (classifiers) on a given example. (This example is more class **+** than class **-**)
- Can this be generalized to the case of **k** labels, **k > 2**?

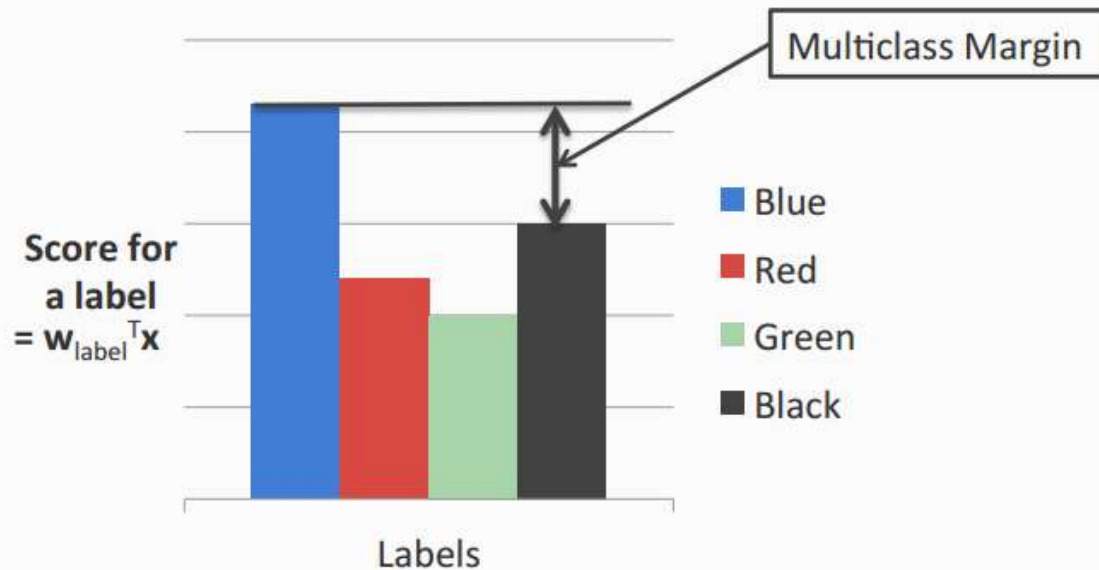
Recall: Margin for binary classifiers

- The **margin** of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Multiclass Margin

Defined as the score difference between the highest scoring label and the second one



Multiclass SVM (Intuition)

■ Recall: Binary SVM

- Maximize margin

- Equivalently,

Minimize norm of weights such that the closest points to the hyperplane have a score 1

■ Multiclass SVM

- Each label has a different weight vector (like one-vs-all)

- Maximize multiclass margin

- Equivalently,

Minimize total norm of the weights such that the true label is scored at least 1 more than the second best one

Multiclass SVM in the separable case

Recall hard binary SVM

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{s.t. } \forall i, \quad & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{aligned}$$

Size of the weights. Effectively,
regularizer

$$\begin{aligned} \min_{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K} \quad & \frac{1}{2} \sum_k \mathbf{w}_k^T \mathbf{w}_k \\ \text{s.t.} \quad & \mathbf{w}_{y_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1 \end{aligned}$$

$$\begin{aligned} \forall (\mathbf{x}_i, \mathbf{y}_i) \in D, \\ k \in \{1, 2, \dots, K\}, k \neq \mathbf{y}_i, \end{aligned}$$

The score for the true label is higher than the score
for **any** other label by 1

Multiclass SVM: General case

Size of the weights. Effectively, regularizer

Total slack. Effectively, don't allow too many examples to violate the margin constraint

$$\min_{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K, \xi} \quad \frac{1}{2} \sum_k \mathbf{w}_k^T \mathbf{w}_k + C \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in D} \xi_i$$

s.t. $\mathbf{w}_{\mathbf{y}_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1 - \xi_i, \quad \forall (\mathbf{x}_i, \mathbf{y}_i) \in D,$
 $k \in \{1, 2, \dots, K\}, k \neq \mathbf{y}_i,$
 $\forall i.$

$\xi_i \geq 0,$

The score for the true label is higher than the score for **any** other label by $1 - \xi_i$

Slack variables. Not all examples need to satisfy the margin constraint.

Slack variables can only be positive

Multiclass SVM: Summary

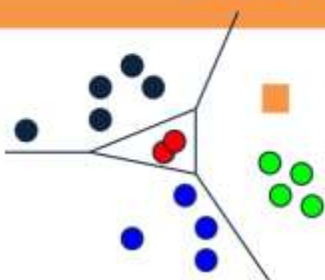
- **Training:**
 - Optimize the “global” SVM objective
- **Prediction:**
 - Winner takes all
 $\operatorname{argmax}_i \mathbf{w}_i^T \mathbf{x}$
- With K labels and inputs in \mathfrak{R}^n , we have nK weights in all
 - Same as one-vs-all
- **Why does it work?**
 - Why is this the “right” definition of multiclass margin?
- A theoretical justification, along with extensions to other algorithms beyond SVM is given by “Constraint Classification”
 - Applies also to multi-label problems, ranking problems, etc.
 - [Dav Zimak; with D. Roth and S. Har-Peled]

Constraint Classification

- The examples we give the learner are pairs (x, y) , $y \in \{1, \dots, k\}$
- The “black box learner” (1 vs. all) we described might be thought of as a function of x only but, actually, we made use of the labels y
- How is y being used?
 - y decides what to do with the example x ; that is, which of the k classifiers should take the example as a positive example (making it a negative to all the others).
- How do we predict?
 - Let: $f_y(x) = w_y^T \cdot x$
 - Then, we predict using: $y^* = \operatorname{argmax}_{y=1, \dots, k} f_y(x)$
- Equivalently, we can say that we predict as follows:
 - Predict y iff
 - $\forall y' \in \{1, \dots, k\}, y' \neq y \quad (w_y^T - w_{y'}^T) \cdot x \geq 0 \quad (**)$
- So far, we did not say how we learn the k weight vectors w_y ($y = 1, \dots, k$)
 - Can we train in a way that better fits the way we predict?
 - What does it mean?

Is it better in any well defined way?

Linear Separability for Multiclass



- We are learning k **n-dimensional** weight vectors, so we can concatenate the k weight vectors into

$$w = (w_1, w_2, \dots, w_k) \in \mathbb{R}^{kn}$$

Notice: This is just a representational trick. We did not say how to learn the weight vectors.

- **Key Construction:** (Kesler Construction; Zimak's Constraint Classification)

- We will represent each example (x, y) , as an nk -dimensional vector, x_y , with x embedded in the y -th part of it ($y=1, 2, \dots, k$) and the other coordinates are 0.

- **E.g.,** $x_y = (0, x, 0, 0) \in \mathbb{R}^{kn}$ (here $k=4, y=2$)

- Now we can understand the n -dimensional decision rule:

- Predict y iff $\forall y' \in \{1, \dots, k\}, y' \neq y \quad (w_{y'}^T - w_y^T) \cdot x \geq 0$ (**)

- Equivalently, in the nk -dimensional space.

- Predict y iff $\forall y' \in \{1, \dots, k\}, y' \neq y \quad w^T \cdot (x_y - x_{y'}) \equiv w^T \cdot x_{yy'} \geq 0$

- **Conclusion:** The set $(x_{yy'}, +) \equiv (x_y - x_{y'}, +)$ is **linearly separable** from the set $(-x_{yy'}, -)$ using the linear separator $w \in \mathbb{R}^{kn}$.
- **We solved** the voroni diagram challenge.

Constraint Classification

■ Training:

- [We first explain via Kesler's construction; then show we don't need it]
- Given a data set $\{(x,y)\}$, (m examples) with $x \in \mathbb{R}^n$, $y \in \{1,2,\dots,k\}$ create a binary classification task:
 $(x_y - x_{y'}, +)$, $(x_{y'} - x_y, -)$, for all $y' \neq y$ ($2m(k-1)$ examples)
Here $x_y \in \mathbb{R}^{kn}$
- Use your favorite linear learning algorithm to train a binary classifier.

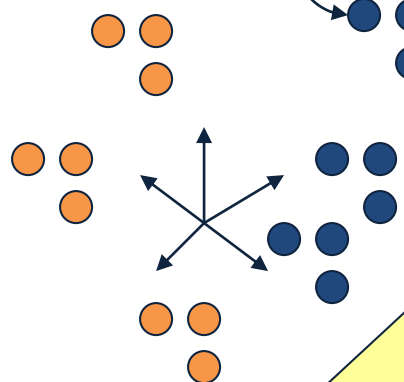
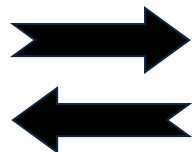
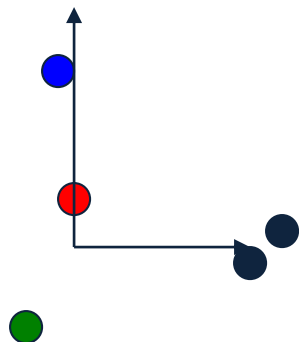
■ Prediction:

- Given an nk dimensional weight vector w and a new example x , predict:
$$\operatorname{argmax}_y w^T x_y$$

Details: Kesler Construction & Multi-Class Separability

Transform Examples

$\frac{2 > 1}{2 > 3}$
 $\frac{2 > 4}{2 > 4}$



If (x, i) was a given n -dimensional example (that is, x has is labeled i , then $x_{ij}, \forall j=1, \dots, k, j \neq i$, are positive examples in the nk -dimensional space. $-x_{ij}$ are negative examples.

$i > j$	$f_i(x) - f_j(x) > 0$
	$w_i \cdot x - w_j \cdot x > 0$
	$W \cdot X_i - W \cdot X_j > 0$
	$W \cdot (X_i - X_j) > 0$
	$W \cdot X_{ij} > 0$

$$X_i = (0, x, 0, 0) \in \mathbf{R}^{kd}$$

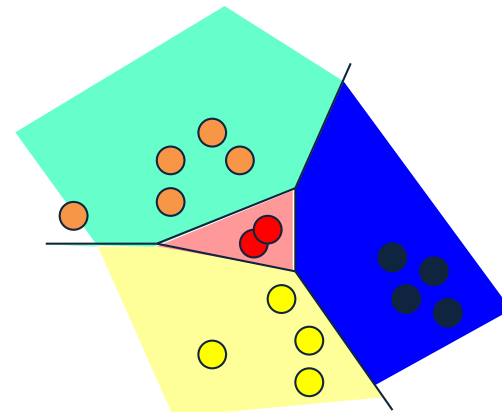
$$X_j = (0, 0, 0, x) \in \mathbf{R}^{kd}$$

$$X_{ij} = X_i - X_j = (0, x, 0, -x)$$

$$W = (w_1, w_2, w_3, w_4) \in \mathbf{R}^{kd}$$

Learning via Kesler's Construction

- Given $(x_1, y_1), \dots, (x_N, y_N) \in \mathbf{R}^n \times \{1, \dots, k\}$
- Create
 - $\mathbf{P}^+ = \cup \mathbf{P}^+(x_i, y_i)$
 - $\mathbf{P}^- = \cup \mathbf{P}^-(x_i, y_i)$
- Find $\mathbf{w} = (w_1, \dots, w_k) \in \mathbf{R}^{kn}$, such that
 - $\mathbf{w} \cdot \mathbf{x}$ separates \mathbf{P}^+ from \mathbf{P}^-
- One can use any algorithm in this space: Perceptron, Winnow, SVM, etc.
- To understand how to update the weight vector in the n-dimensional space, we note that
 - $\mathbf{w}^T \cdot \mathbf{x}_{yy'} \geq 0$ (in the nk-dimensional space)
 - is equivalent to:
 - $(\mathbf{w}_y^T - \mathbf{w}_{y'}^T) \cdot \mathbf{x} \geq 0$ (in the n-dimensional space)



Perceptron in Kesler Construction

- A perceptron update rule applied in the **nk-dimensional space** due to a mistake in $w^T \cdot x_{ij} \geq 0$
- Or, equivalently to $(w_i^T - w_j^T) \cdot x \geq 0$ (in the **n-dimensional space**)
- Implies the following update:

- Given example (x, i) (example $x \in \mathbb{R}^n$, labeled i)
 - $\forall (i, j), i, j = 1, \dots, k, i \neq j$ (***)
 - If $(w_i^T - w_j^T) \cdot x < 0$ (mistaken prediction; equivalent to $w^T \cdot x_{ij} < 0$)
 - $w_i \leftarrow w_i + x$ (promotion) and $w_j \leftarrow w_j - x$ (demotion)

- Note that this is a generalization of balanced Winnow rule.

- Note that we promote w_i and demote **k-1** weight vectors w_j

Conservative update

- The general scheme suggests:
- Given example (x, i) (example $x \in \mathbb{R}^n$, labeled i)
 - $\forall (i, j), i, j = 1, \dots, k, i \neq j$ (***)
 - If $(w_i^T - w_j^T) \cdot x < 0$ (mistaken prediction; equivalent to $w^T \cdot x_{ij} < 0$)
 - $w_i \leftarrow w_i + x$ (promotion) and $w_j \leftarrow w_j - x$ (demotion)
- Promote w_i and demote $k-1$ weight vectors w_j
- A conservative update: (SNoW and LBJava's implementation):
 - In case of a mistake: only the weights corresponding to the target node i and that **closest** node j are updated.
 - Let: $j^* = \operatorname{argmax}_{j=1, \dots, k} w_j^T \cdot x$ (highest activation among competing labels)
 - If $(w_i^T - w_{j^*}^T) \cdot x < 0$ (mistaken prediction)
 - $w_i \leftarrow w_i + x$ (promotion) and $w_{j^*} \leftarrow w_{j^*} - x$ (demotion)
 - Other weight vectors are not being updated.

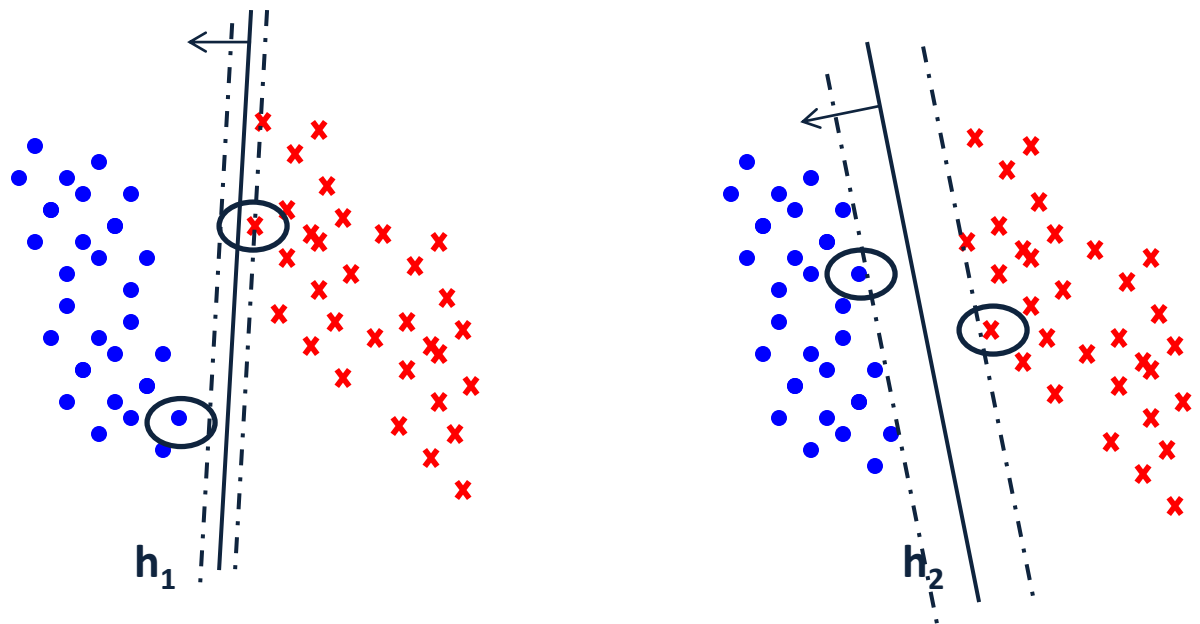
Data Dependent VC dimension

- So far we discussed VC dimension in the context of a **fixed** class of functions.
- We can also parameterize the class of functions in interesting ways.
- Recall the VC based generalization bound:

$$\text{Err}(h) \leq \text{err}_{\text{TR}}(h) + \text{Poly}\{\text{VC}(H), 1/m, \log(1/\delta)\}$$

Linear Classification

- Although both classifiers separate the data, the distance with which the separation is achieved is different:



Concept of Margin

- The margin γ_i of a point $x_i \in \mathbb{R}^n$ with respect to a linear classifier $h(x) = \text{sign}(w \cdot x + b)$ is defined as the distance of x_i from the hyperplane $w \cdot x + b = 0$:
- $\gamma_i = |(w \cdot x_i + b) / \|w\| |$
- The **margin** of a set of points $\{x_1, \dots, x_m\}$ with respect to a **hyperplane** w , is defined as the margin of the point closest to the hyperplane:
- $$\gamma = \min_i \gamma_i = \min_i |(w \cdot x_i + b) / \|w\| |$$

VC and Linear Classification

- If H_γ is the space of all linear classifiers in \mathcal{R}^n that separate the training data with margin at least γ , then:

$$VC(H_\gamma) \leq \min(R^2/\gamma^2, n) + 1,$$

- Where R is the radius of the smallest sphere (in \mathcal{R}^n) that contains the data.
- Thus, for such classifiers, we have a bound of the form:

$$\text{Err}(h) \leq \text{err}_{\text{TR}}(h) + \{ (O(R^2/\gamma^2) + \log(4/\delta))/m \}^{1/2}$$

Data Dependent VC dimension

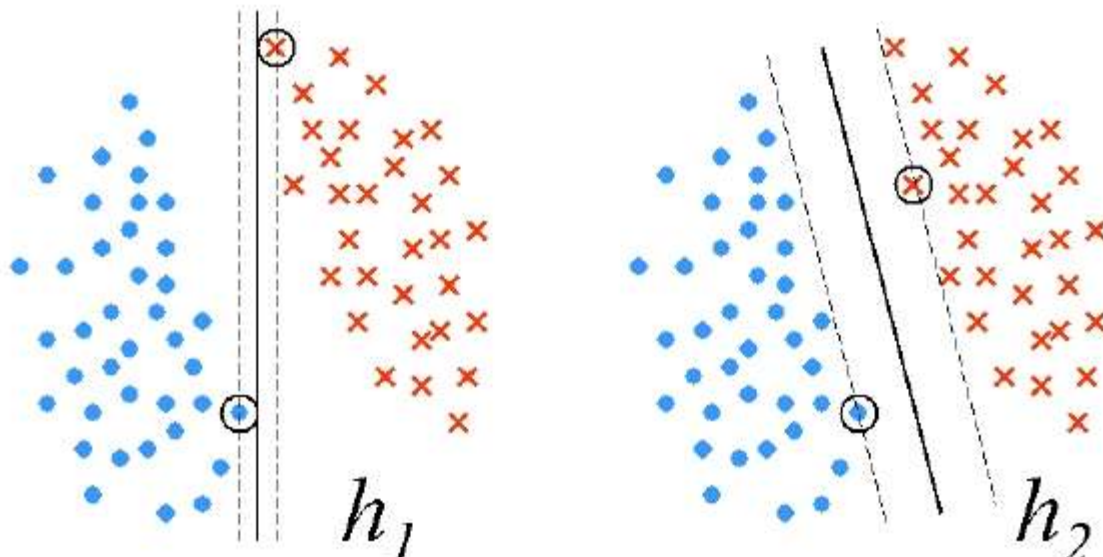
- Namely, when we consider the class H_γ of linear hypotheses that separate a given data set with a margin γ ,
- We see that
 - Large Margin $\gamma \rightarrow$ Small VC dimension of H_γ
- Consequently, our goal could be to find a separating hyperplane w that **maximizes the margin** of the set S of examples.
- A **second observation** that drives an algorithmic approach is that:

Small $\|w\| \rightarrow$ Large Margin
- **This leads to an algorithm:** from among all those w 's that agree with the data, find the one with the **minimal size $\|w\|$**

Maximal Margin

- This discussion motivates the notion of a maximal margin.
- The **maximal margin** of a data set S is define as:

$$\gamma(S) = \max_{\|w\|=1} \min_{(x,y) \in S} |y w^T x|$$



1. For a given w : Find the closest point.
2. Then, find the one the gives the **maximal** margin value across all w 's (of size 1).

Note: the selection of the point is in the **min** and therefore the **max** does not change if we scale w , so it's okay to only deal with normalized w 's.

How does it help us to derive these h 's?

$$\operatorname{argmax}_{\|w\|=1} \min_{(x,y) \in S} |y w^T x|$$

Hard SVM Optimization

- We have shown that the sought after weight vector w is the solution of the following optimization problem:

SVM Optimization: (***)

- Minimize: $\frac{1}{2} ||w||^2$

Subject to: $\forall (x,y) \in S: y w^T x \geq 1$

- This is an optimization problem in $(n+1)$ variables, with $|S|=m$ inequality constraints.

Support Vector Machines

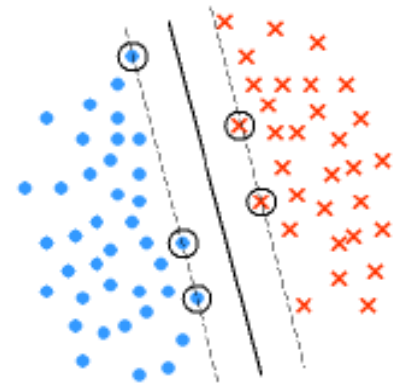
- The name “Support Vector Machine” stems from the fact that w^* is **supported** by (i.e. is the linear span of) the examples that are exactly at a distance $1/||w^*||$ from the separating hyperplane. These vectors are therefore called **support vectors**.

- **Theorem:** Let w^* be the minimizer of the SVM optimization problem (***) for $S = \{(x_i, y_i)\}$. Let $I = \{i: w^{*T}x_i = 1\}$.

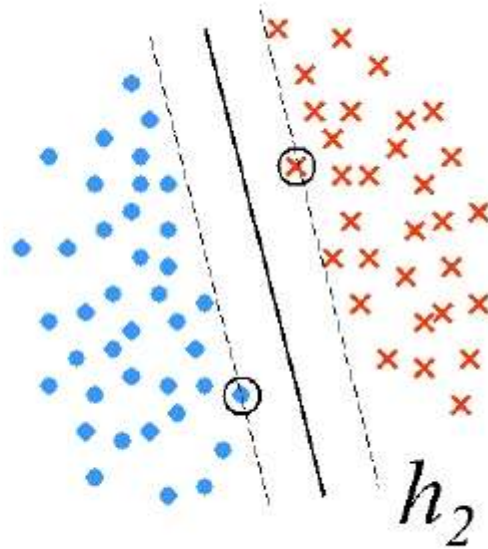
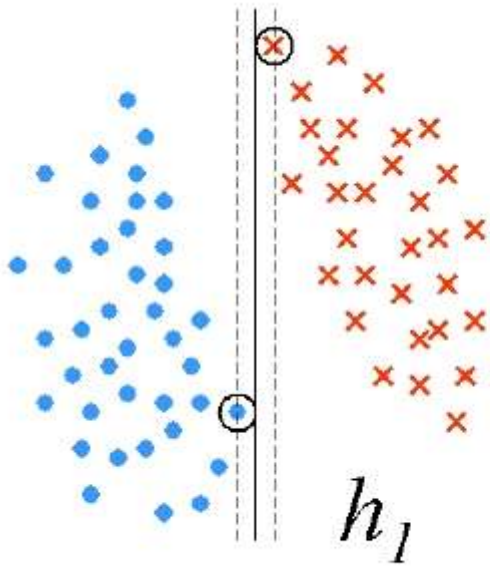
Then there exists coefficients $\alpha_i > 0$ such that:

$$w^* = \sum_{i \in I} \alpha_i y_i x_i$$

This representation should ring a bell...



Maximal Margin



The margin of a linear separator $w^T x + b = 0$ is $2 / \|w\|$

$$\max 2 / \|w\| = \min \|w\| \\ = \min \frac{1}{2} w^T w$$

$$\min_{w, b} \frac{1}{2} w^T w$$

$$\text{s.t. } y_i (w^T x_i + b) \geq 1, \forall (x_i, y_i) \in S$$

Duality

- This, and other properties of Support Vector Machines are shown by moving to the [dual problem](#).

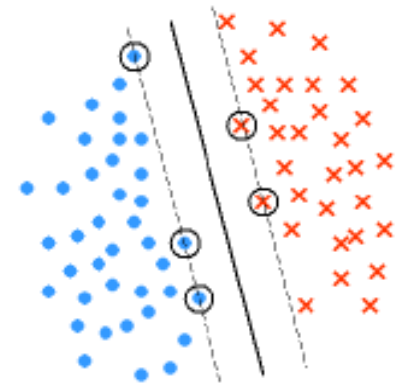
- **Theorem:** Let w^* be the minimizer of the SVM optimization problem (***)

for $S = \{(x_i, y_i)\}$.

Let $I = \{i: y_i (w^{*\top} x_i + b) = 1\}$.

Then there exists coefficients $\alpha_i > 0$ such that:

$$w^* = \sum_{i \in I} \alpha_i y_i x_i$$



Soft SVM

- Notice that the relaxation of the constraint:

$$y_i w^T x_i \geq 1$$

- Can be done by introducing a **slack variable** ξ_i (per example) and requiring:

$$y_i w^T x_i \geq 1 - \xi_i ; \xi_i \geq 0$$

- Now, we want to solve:

$$\min_{w, \xi_i} \frac{1}{2} w^T w + C \sum_i \xi_i$$

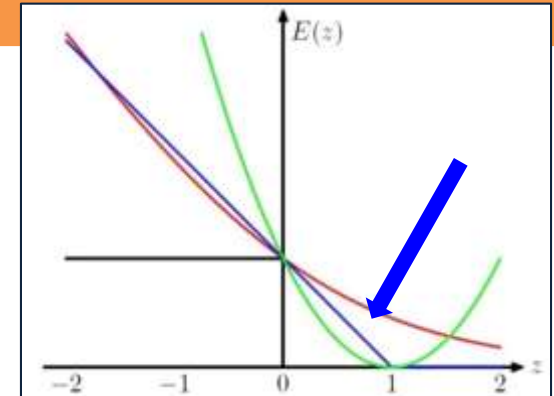
$$\text{s.t. } y_i w^T x_i \geq 1 - \xi_i ; \xi_i \geq 0 \quad \forall i$$

Soft SVM (2)

- Now, we want to solve:

$$\min_{w, \xi_i} \frac{1}{2} w^T w + C \sum_i \xi_i$$

$$\text{s.t. } \xi_i \geq 1 - y_i w^T x_i; \xi_i \geq 0 \quad \forall i$$



In optimum, $\xi_i = \max(0, 1 - y_i w^T x_i)$

- Which can be written as:

$$\min_w \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i w^T x_i).$$

- What is the interpretation of this?

SVM Objective Function

- The problem we solved is:

$$\text{Min } \frac{1}{2} ||w||^2 + c \sum \xi_i$$

- Where $\xi_i > 0$ is called a **slack variable**, and is defined by:

- $\xi_i = \max(0, 1 - y_i w^t x_i)$

- Equivalently, we can say that: $y_i w^t x_i \geq 1 - \xi; \xi \geq 0$

- And this can be written as:

$$\text{Min } \frac{1}{2} ||w||^2$$

Regularization term

Can be replaced by other **regularization functions**

$$+ c \sum \xi_i$$

Empirical loss

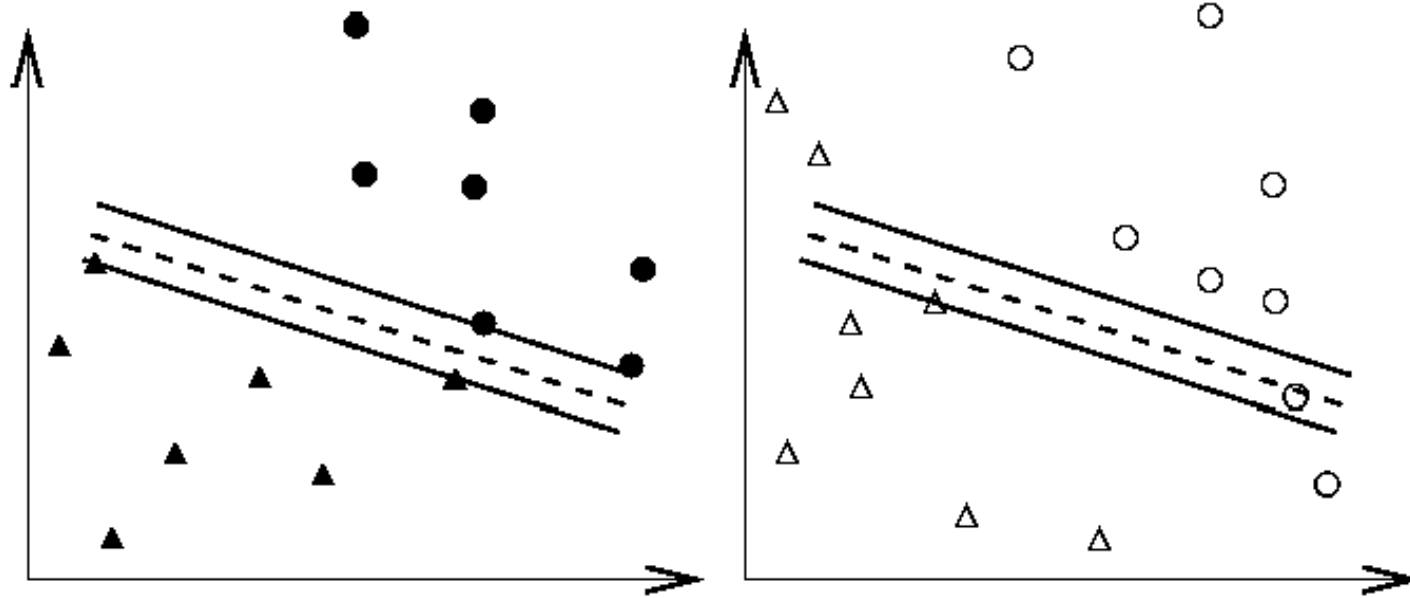
Can be replaced by other **loss functions**

- General Form of a learning algorithm:

- Minimize empirical loss, and Regularize (to avoid over fitting)

- Theoretically motivated improvement over the original algorithm we've see at the beginning of the semester.

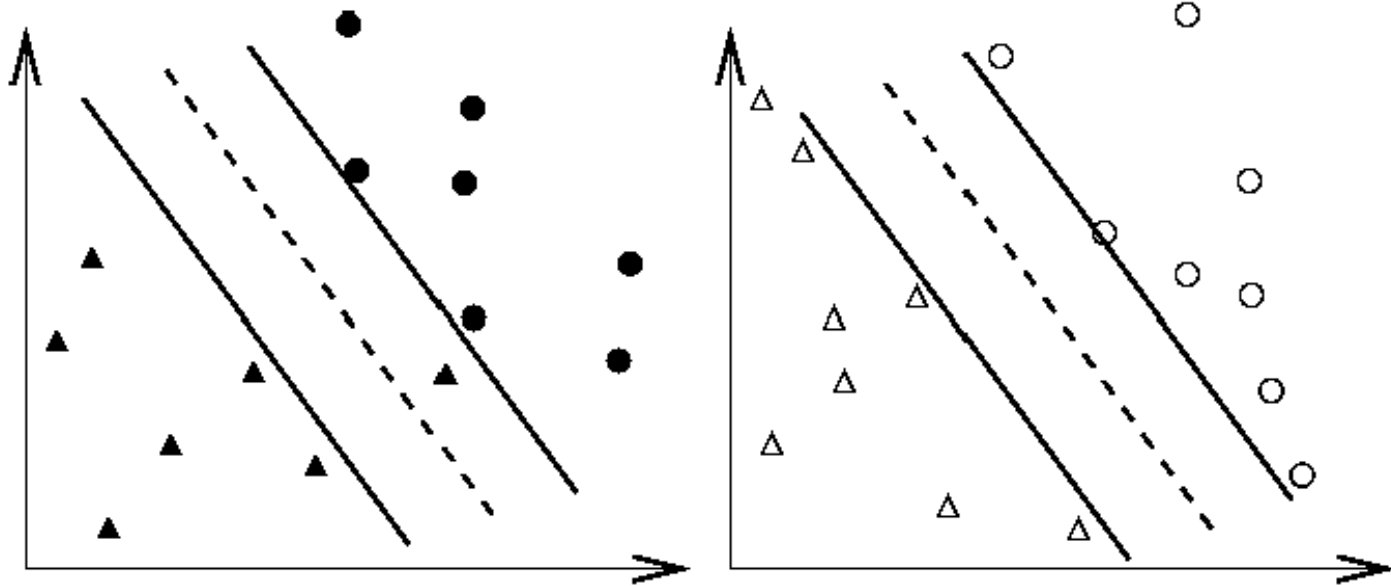
Balance between regularization and empirical loss



(a) Training data and an over-fitting classifier

(b) Testing data and an over-fitting classifier

Balance between regularization and empirical loss

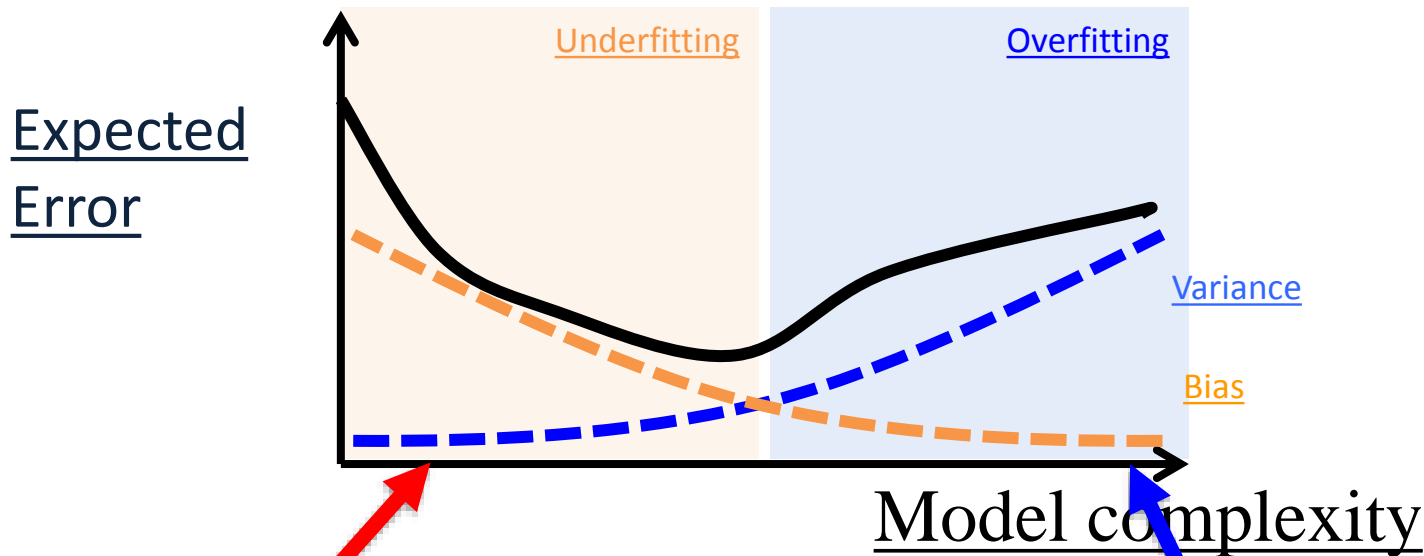


(c) Training data and a better classifier

(d) Testing data and a better classifier

[\(DEMO\)](#)

Underfitting and Overfitting



■ **Simple models:**
High bias and low variance

Complex models:
High variance and low bias

■ **Smaller C**

Larger C

What Do We Optimize?

- Logistic Regression

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^l \log(1 + e^{-y_i(w^T x_i)})$$

- L1-loss SVM

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^l \max(0, 1 - y_i w^T x_i)$$

- L2-loss SVM

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^l \max(0, 1 - y_i w^T x_i)^2$$

Optimization: How to Solve

- 1. Earlier methods used Quadratic Programming. Very slow.
- 2. The soft SVM problem is an unconstrained optimization problems. It is possible to use the **gradient descent algorithm**! Still, it is quite slow.
- Many options within this category:
 - Iterative scaling; non-linear conjugate gradient; quasi-Newton methods; truncated Newton methods; trust-region newton method.
 - All methods are iterative methods, that **generate a sequence w_k** that converges to the optimal solution of the optimization problem above.
 - Currently: **Limited memory BFGS** is very popular
- 3. 3rd generation algorithms are based on Stochastic Gradient Decent
 - The runtime does not depend on n =#(examples); advantage when n is very large.
 - Stopping criteria is a problem: method tends to be too aggressive at the beginning and reaches a moderate accuracy quite fast, but it's convergence becomes slow if we are interested in more accurate solutions.
- 4. Dual Coordinated Descent (& Stochastic Version)

SGD for SVM

- Goal: $\min_w f(w) \equiv \frac{1}{2} w^T w + \frac{c}{m} \sum_i \max(0, 1 - y_i w^T x_i)$. m : data size

m is here for mathematical correctness, it doesn't matter in the view of modeling.

- Compute sub-gradient of $f(w)$:

$$\nabla f(w) = w - C y_i x_i \text{ if } 1 - y_i w^T x_i \geq 0 ; \text{ otherwise } \nabla f(w) = w$$

1. Initialize $w = 0 \in R^n$

2. For every example $(x_i, y_i) \in D$

If $y_i w^T x_i \leq 1$ **update** the weight vector to

$$w \leftarrow (1 - \gamma)w + \gamma C y_i x_i \quad (\gamma - \text{learning rate})$$

Otherwise $w \leftarrow (1 - \gamma)w$

3. Continue until convergence is achieved

Convergence can be proved for a slightly complicated version of SGD (e.g, Pegasos)

This algorithm should ring a bell...

Nonlinear SVM

- We can map data to a high dimensional space: $x \rightarrow \phi(x)$ [\(DEMO\)](#)
- Then use Kernel trick: $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ [\(DEMO2\)](#)

Primal:

$$\min_{w, \xi_i} \frac{1}{2} w^T w + C \sum_i \xi_i$$

$$\text{s.t. } y_i w^T \phi(x_i) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \quad \forall i$$

Dual:

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

$$\text{s.t. } 0 \leq \alpha \leq C \quad \forall i$$

$$Q_{ij} = y_i y_j K(x_i, x_j)$$

Theorem: Let w^* be the minimizer of the primal problem, α^* be the minimizer of the dual problem.

Then $w^* = \sum_i \alpha^* y_i x_i$

1: Direct Learning

- Model the problem of text correction as a problem of **learning from examples**.
- Goal: learn directly how to make predictions.

PARADIGM

- Look at many (positive/negative) examples.
- Discover some regularities in the data.
- Use these to construct a **prediction policy**.
- A policy (a function, a predictor) needs to be specific.
 [it/in] rule: if **the** occurs after the target \Rightarrow in
- **Assumptions** comes in the form of a **hypothesis class**.

Bottom line: approximating $h : X \rightarrow Y$, is estimating **$P(Y|X)$** .

Direct Learning (2)

- Consider a distribution D over space $X \times Y$
- X - the instance space; Y - set of labels. (e.g. ± 1)
- Given a sample $\{(x, y)\}_1^m$, and a loss function $L(x, y)$
- Find $h \in H$ that minimizes

$$\sum_{i=1, m} D(x_i, y_i) L(h(x_i), y_i) + \text{Reg}$$

- L can be: $L(h(x), y) = 1, h(x) \neq y$, o/w $L(h(x), y) = 0$ (0-1 loss)

$$L(h(x), y) = (h(x) - y)^2, \quad (L_2)$$

$$L(h(x), y) = \max\{0, 1 - y h(x)\} \quad (\text{hinge loss})$$

$$L(h(x), y) = \exp\{-y h(x)\} \quad (\text{exponential loss})$$

- **Guarantees:** If we find an algorithm that minimizes loss on the observed data. Then, learning theory guarantees good future behavior (as a function of H).

2: Generative Model

The model is called “generative” since it assumes how data X is generated given y

- Model the problem of text correction as that of **generating correct sentences**.
- Goal: learn a **model of the language**; use it to predict.

PARADIGM

- Learn a probability distribution over sentences
 - In practice: make assumptions on the distribution’s type
- Use it to estimate which sentence is more likely.
 - $\Pr(\text{I saw the girl } \mathbf{it} \text{ the park}) \leftrightarrow \Pr(\text{I saw the girl } \mathbf{in} \text{ the park})$
 - In practice: a decision policy depends on the assumptions

Bottom line: the generating paradigm approximates

$$P(X,Y) = P(X|Y) P(Y).$$

- **Guarantees:** We need to assume the “right” probability distribution

Probabilistic Learning

- There are actually two different notions.
 - Learning probabilistic concepts
 - The learned concept is a function $c:X\rightarrow[0,1]$
 - $c(x)$ may be interpreted as the probability that the label 1 is assigned to x
 - The learning theory that we have studied before is applicable (with some extensions).
 - Bayesian Learning: Use of a probabilistic criterion in selecting a hypothesis
 - The hypothesis can be deterministic, a Boolean function.
- It's not the hypothesis – it's the process.

Basics of Bayesian Learning

- **Goal:** find the best hypothesis from some space H of hypotheses, **given** the observed data D .
- Define best to be: most probable hypothesis in H
- In order to do that, we need to assume a probability distribution **over the class H** .
- In addition, we need to know something about the relation between the data observed and the hypotheses (E.g., a coin problem.)
 - As we will see, we will be Bayesian about other things, e.g., the parameters of the model

Basics of Bayesian Learning

- $P(h)$ - the prior probability of a hypothesis h
Reflects background knowledge; before data is observed. If no information - uniform distribution.
- $P(D)$ - The probability that this sample of the Data is observed.
(No knowledge of the hypothesis)
- $P(D|h)$: The probability of observing the sample D , given that hypothesis h is the target
- $P(h|D)$: The posterior probability of h . The probability that h is the target, given that D has been observed.

Bayes Theorem

$$\mathbf{P(\mathbf{h} | \mathbf{D}) = P(\mathbf{D} | \mathbf{h}) \frac{P(\mathbf{h})}{P(\mathbf{D})}}$$

- $P(h|D)$ increases with $P(h)$ and with $P(D|h)$
- $P(h|D)$ decreases with $P(D)$

Learning Scenario

- $P(h|D) = P(D|h) P(h)/P(D)$
- The learner considers a set of candidate hypotheses H (models), and attempts to find the most probable one $h \in H$, given the observed data.
- Such maximally probable hypothesis is called maximum a posteriori hypothesis (MAP); Bayes theorem is used to compute it:

$$\begin{aligned} h_{\text{MAP}} &= \operatorname{argmax}_{h \in \mathcal{H}} P(h|D) = \operatorname{argmax}_{h \in \mathcal{H}} P(D|h) P(h)/P(D) \\ &= \operatorname{argmax}_{h \in \mathcal{H}} P(D|h) P(h) \end{aligned}$$

Learning Scenario (2)

$$h_{\text{MAP}} = \operatorname{argmax}_{h \in \mathcal{H}} P(h|D) = \operatorname{argmax}_{h \in \mathcal{H}} P(D|h) P(h)$$

- We may assume that a priori, hypotheses are equally probable:
$$P(h_i) = P(h_j) \quad \forall h_i, h_j \in H$$

- We get the **Maximum Likelihood hypothesis**:

$$h_{\text{ML}} = \operatorname{argmax}_{h \in \mathcal{H}} P(D|h)$$

- Here we just look for the hypothesis that best explains the data

Bayes Optimal Classifier

- How should we use the general formalism?
- What should H be?
- H can be a collection of functions. Given the training data, choose an optimal function. Then, given new data, evaluate the selected function on it.
- H can be a collection of possible predictions. Given the data, try to directly choose the optimal prediction.
- Could be different!

Bayes Optimal Classifier

- The first formalism suggests to learn a good hypothesis and use it.
- (Language modeling, grammar learning, etc. are here)

$$\mathbf{h}_{\text{MAP}} = \operatorname{argmax}_{\mathbf{h} \in \mathbf{H}} \mathbf{P}(\mathbf{h} \mid \mathbf{D}) = \operatorname{argmax}_{\mathbf{h} \in \mathbf{H}} \mathbf{P}(\mathbf{D} \mid \mathbf{h})\mathbf{P}(\mathbf{h})$$

- The second one suggests to directly choose a decision. [\[it/in\]](#):
- This is the issue of “thresholding” vs. entertaining all options until the last minute. (Computational Issues)

Justification: Bayesian Approach

- The Bayes optimal function is

$$f_B(x) = \operatorname{argmax}_y D(x; y)$$

- That is, given input x , return the most likely label
- It can be shown that f_B has the lowest possible value for $\operatorname{Err}(f)$
- Caveat: we can never construct this function: it is a function of D , which is unknown.
- But, it is a useful theoretical construct, and drives attempts to make assumptions on D

Maximum-Likelihood Estimates

- We attempt to model the underlying distribution

$$D(x, y) \text{ or } D(y | x)$$

- To do that, we assume a model

$$P(x, y | \theta) \text{ or } P(y | x, \theta),$$

where θ is the set of parameters of the model

- Example: **Probabilistic Language Model (Markov Model):**

- We assume a model of language generation. Therefore, $P(x, y | \theta)$ was written as a function of symbol & state probabilities (the parameters).

- We typically look at the log-likelihood

- Given training samples $(x_i; y_i)$, maximize the log-likelihood

- $L(\theta) = \sum_i \log P(x_i; y_i | \theta)$ or $L(\theta) = \sum_i \log P(y_i | x_i, \theta)$

Justification: Bayesian Approach

- Assumption: Our selection of the model is good; there is some parameter setting θ^* such that the true distribution is really represented by our model

$$D(x, y) = P(x, y \mid \theta^*)$$

- Define the maximum-likelihood estimates:

$$\theta_{ML} = \operatorname{argmax}_{\theta} L(\theta)$$

- As the training sample size goes to ∞ , then

$$P(x, y \mid \theta_{ML}) \text{ converges to } D(x, y)$$

Given the [assumption](#) above, and the availability of [enough data](#)

$$\operatorname{argmax}_y P(x, y \mid \theta_{ML})$$

converges to the Bayes-optimal function

$$f_B(x) = \operatorname{argmax}_y D(x; y)$$

Bayesian Classifier

- $f: X \rightarrow V$, finite set of values
- Instances $x \in X$ can be described as a collection of features

$$x = (x_1, x_2, \dots, x_n) \quad x_i \in \{0, 1\}$$

- Given an example, assign it the most probable value in V
- Bayes Rule:

$$\mathbf{v}_{\text{MAP}} = \mathbf{argmax}_{v_j \in V} \mathbf{P}(v_j | \mathbf{x}) = \mathbf{argmax}_{v_j \in V} \mathbf{P}(v_j | \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$$

$$\begin{aligned} \mathbf{v}_{\text{MAP}} &= \mathbf{argmax}_{v_j \in V} \frac{\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n | v_j) \mathbf{P}(v_j)}{\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)} \\ &= \mathbf{argmax}_{v_j \in V} \mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n | v_j) \mathbf{P}(v_j) \end{aligned}$$

- Notational convention: $P(y)$ means $P(Y=y)$

Bayesian Classifier

$$V_{\text{MAP}} = \operatorname{argmax}_v P(x_1, x_2, \dots, x_n | v)P(v)$$

- Given training data we can estimate the two terms.
- Estimating $P(v)$ is easy. E.g., under the binomial distribution assumption, count the number of times v appears in the training data.
- However, it is not feasible to estimate $P(x_1, x_2, \dots, x_n | v)$
- In this case we have to estimate, for each target value, the probability of each instance (most of which will not occur).
- In order to use a Bayesian classifiers in practice, we need to make assumptions that will allow us to estimate these quantities.

Naive Bayes

$$V_{\text{MAP}} = \operatorname{argmax}_v P(x_1, x_2, \dots, x_n | v)P(v)$$

$$P(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n | v_j) =$$

$$= P(\mathbf{x}_1 | \mathbf{x}_2, \dots, \mathbf{x}_n, v_j)P(\mathbf{x}_2, \dots, \mathbf{x}_n | v_j)$$

$$= P(\mathbf{x}_1 | \mathbf{x}_2, \dots, \mathbf{x}_n, v_j)P(\mathbf{x}_2 | \mathbf{x}_3, \dots, \mathbf{x}_n, v_j)P(\mathbf{x}_3, \dots, \mathbf{x}_n | v_j)$$

=

$$= P(\mathbf{x}_1 | \mathbf{x}_2, \dots, \mathbf{x}_n, v_j)P(\mathbf{x}_2 | \mathbf{x}_3, \dots, \mathbf{x}_n, v_j)P(\mathbf{x}_3 | \mathbf{x}_4, \dots, \mathbf{x}_n, v_j) \dots P(\mathbf{x}_n | v_j)$$

- Assumption: feature values are independent given the target value

Naive Bayes (2)

$$V_{\text{MAP}} = \operatorname{argmax}_v P(x_1, x_2, \dots, x_n | v) P(v)$$

- Assumption: feature values are independent given the target value

$$P(x_1 = b_1, x_2 = b_2, \dots, x_n = b_n | v = v_j) = \prod_{k=1}^n P(x_k = b_k | v = v_j)$$

- Generative model:
- First choose a value $v_j \in V$ according to $P(v)$
- For each v_j : choose x_1, x_2, \dots, x_n according to $P(x_k | v_j)$

Naive Bayes (3)

$$V_{\text{MAP}} = \operatorname{argmax}_v P(x_1, x_2, \dots, x_n | v) P(v)$$

- Assumption: feature values are independent given the target value

$$P(x_1 = b_1, x_2 = b_2, \dots, x_n = b_n | v = v_j) = \prod_{i=1}^n P(x_i = b_i | v = v_j)$$

- **Learning method:** Estimate $n|V| + |V|$ parameters and use them to make a prediction. (How to estimate?)
- Notice that this is **learning without search**. Given a collection of training examples, you just compute the best hypothesis (given the assumptions).
- This is learning **without trying to achieve consistency** or even approximate consistency.

Lecture 10: EM

- EM is a **class of algorithms** that is used to estimate a probability distribution in the presence of missing attributes.
- Using it requires an assumption on the underlying probability distribution.
- The algorithm can be very sensitive to this assumption and to the starting point (that is, the initial guess of parameters).
- In general, known to converge to a local maximum of the maximum likelihood function.

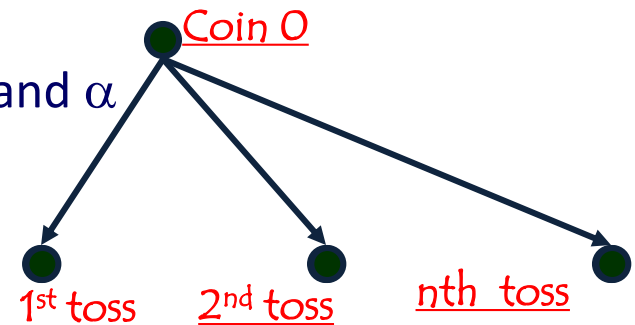
Three Coin Example

- We observe a series of coin tosses generated in the following way:
- A person has three coins.
 - Coin 0: probability of Head is α
 - Coin 1: probability of Head p
 - Coin 2: probability of Head q
- Consider the following coin-tossing scenarios:

Estimation Problems

- Scenario I: Toss one of the coins four times.
Observing HHTH
Question: Which coin is more likely to produce this sequence ?
- Scenario II: Toss coin 0. If Head – toss coin 1; o/w – toss coin 2
Observing the sequence HHHHT, THTHT, HHHHT, HHTTH
produced by Coin 0 , Coin1 and Coin2
Question: Estimate most likely values for p , q (the probability of H in each coin) and the probability to use each of the coins (α)
- Scenario III: Toss coin 0. If Head – toss coin 1; o/w – toss coin 2
Observing the sequence HHHT, HTHT, HHHT, HTTH
produced by Coin 1 and/or Coin 2
Question: Estimate most likely values for p , q and α

There is no known analytical solution to this problem (general setting). That is, it is not known how to compute the values of the parameters so as to maximize the likelihood of the data.



Key Intuition (1)

- If we knew which of the data points (HHHT), (HTHT), (HTTH) came from Coin1 and which from Coin2, there was no problem.
- Recall that the “simple” estimation is the **ML estimation**:
- Assume that you toss a $(p, 1-p)$ coin m times and get k Heads $m-k$ Tails.

$$\log[P(D|p)] = \log [p^k (1-p)^{m-k}] = k \log p + (m-k) \log (1-p)$$

- To maximize, set the derivative w.r.t. p equal to 0:

$$d \log P(D|p)/dp = k/p - (m-k)/(1-p) = 0$$

- Solving this for p , gives: $p=k/m$

Key Intuition (2)

- If we knew which of the data points (HHHT), (HTHT), (HTTH) came from Coin1 and which from Coin2, there was no problem.
- Instead, use an iterative approach for estimating the parameters:
 - Guess the probability that a given data point came from Coin 1 or 2; Generate fictional labels, weighted according to this probability.
 - Now, compute the most likely value of the parameters. [recall NB example]
 - Compute the likelihood of the data given this model.
 - Re-estimate the initial parameter setting: set them to maximize the likelihood of the data.
(Labels \leftrightarrow Model Parameters) \leftrightarrow Likelihood of the data
- This process can be iterated and can be shown to converge to a local maximum of the likelihood function

EM Algorithm (Coins) - I

- We will assume (for a minute) that we know the parameters $\tilde{p}, \tilde{q}, \tilde{\alpha}$ and use it to estimate which Coin it is (Problem 1)
- Then, we will use this “label” estimation of the observed tosses, to estimate the **most likely** parameters
 - and so on...
- Notation: n data points; in each one: m tosses, h_i heads.
- What is the probability that the i th data point came from **Coin1** ?
- **STEP 1 (Expectation Step):** (Here $h=h_i$)

$$\begin{aligned} P_1^i = P(\text{Coin1} | D^i) &= \frac{P(D^i | \text{Coin1}) P(\text{Coin1})}{P(D^i)} = \\ &= \frac{\tilde{\alpha} \tilde{p}^h (1 - \tilde{p})^{m-h}}{\tilde{\alpha} \tilde{p}^h (1 - \tilde{p})^{m-h} + (1 - \tilde{\alpha}) \tilde{q}^h (1 - \tilde{q})^{m-h}} \end{aligned}$$

EM Algorithm

- Now, we would like to compute parameters that maximize it.
- We will maximize the log likelihood of (over n data points)
 - $LL = \sum_{i=1,n} \log P(D^i | p, q, \alpha)$
- But, one of the variables – the coin’s name – is hidden. We can marginalize:
 - $LL = \sum_{i=1,n} \log \sum_{y=0,1} P(D^i, y | p, q, \alpha)$
- However, the sum is inside the log, making the solution difficult.
- Since the latent variable y is not observed, we cannot use the complete-data log likelihood. **Instead**, we use the expectation of the complete-data log likelihood under the posterior distribution of the latent variable to approximate $\log p(D^i | p', q', \alpha')$
- We think of the likelihood $\log P(D^i | p', q', \alpha')$ as a random variable that depends on the value y of the coin in the i^{th} toss. Therefore, instead of maximizing the LL we will maximize the expectation of this random variable (over the coin’s name). [Justified using Jensen’s Inequality; later & above]

$$\begin{aligned}
 LL &= \sum_{i=1,n} \log \sum_{y=0,1} P(D^i, y | p, q, \alpha) = \\
 &= \sum_{i=1,n} \log \sum_{y=0,1} P(D^i | p, q, \alpha) P(y | D^i, p, q, \alpha) = \\
 &= \sum_{i=1,n} \log E_y P(D^i | p, q, \alpha) > \\
 &> \sum_{i=1,n} E_y \log P(D^i | p, q, \alpha)
 \end{aligned}$$

Where the inequality is due to Jensen’s Inequality.
We maximize a lower bound on the Likelihood.

EM Algorithm (Coins) - III

- We maximize the expectation of this random variable (over the coin name).

$$\begin{aligned} E[LL] &= E[\sum_{i=1,n} \log P(D^i | p, q, \alpha)] = \underline{\sum_{i=1,n} E[\log P(D^i | p, q, \alpha)]} = \\ &= \sum_{i=1,n} P_1^i \log P(D^i, 1 | p, q, \alpha) + (1-P_1^i) \log P(D^i, 0 | p, q, \alpha) \end{aligned}$$

- This is due to the linearity of the expectation and the random variable definition:

$$\begin{aligned} \log P(D^i, y | p, q, \alpha) &= \log P(D^i, 1 | p, q, \alpha) \quad \text{with Probability } P_1^i \\ &\quad \log P(D^i, 0 | p, q, \alpha) \quad \text{with Probability } (1-P_1^i) \end{aligned}$$

EM Algorithm (Coins) - IV

- Explicitly, we get:

$$\begin{aligned} E\left(\sum_i \log P(D^i | \tilde{p}, \tilde{q}, \tilde{\alpha})\right) &= \\ &= \sum_i P_1^i \log P(1, D^i | \tilde{p}, \tilde{q}, \tilde{\alpha}) + (1 - P_1^i) \log P(0, D^i | \tilde{p}, \tilde{q}, \tilde{\alpha}) = \\ &= \sum_i P_1^i \log(\tilde{\alpha} \tilde{p}^{h_i} (1 - \tilde{p})^{m - h_i}) + (1 - P_1^i) \log((1 - \tilde{\alpha}) \tilde{q}^{h_i} (1 - \tilde{q})^{m - h_i}) = \\ &= \sum_i P_1^i (\log \tilde{\alpha} + h_i \log \tilde{p} + (m - h_i) \log(1 - \tilde{p})) + \\ &\quad (1 - P_1^i) (\log(1 - \tilde{\alpha}) + h_i \log \tilde{q} + (m - h_i) \log(1 - \tilde{q})) \end{aligned}$$

EM Algorithm (Cont.)

When computing the derivatives, notice P_1^i here is a constant; it was computed using the current parameters in the E step

- Finally, to find the most likely parameters we maximize the derivatives with respect to \tilde{p}
- STEP 2: Maximization Step**
- (Sanity check: Think of the weighted fictional points)

$$\frac{dE}{d\tilde{\alpha}} = \sum_{i=1}^n \frac{P_1^i}{\tilde{\alpha}} - \frac{1-P_1^i}{1-\tilde{\alpha}} = 0 \quad \Rightarrow \quad \tilde{\alpha} = \frac{\sum P_1^i}{n}$$

$$\frac{dE}{d\tilde{p}} = \sum_{i=1}^n P_1^i \left(\frac{h_i}{\tilde{p}} - \frac{m-h_i}{1-\tilde{p}} \right) = 0 \quad \Rightarrow \quad \tilde{p} = \frac{\sum P_1^i \frac{h_i}{m}}{\sum P_1^i}$$

$$\frac{dE}{d\tilde{q}} = \sum_{i=1}^n (1-P_1^i) \left(\frac{h_i}{\tilde{q}} - \frac{m-h_i}{1-\tilde{q}} \right) = 0 \quad \Rightarrow \quad \tilde{q} = \frac{\sum (1-P_1^i) \frac{h_i}{m}}{\sum (1-P_1^i)}$$

The General EM Procedure

- Initially, the parameter θ is set as θ_0
- In E step
 - We use the current parameter values θ^{old} to find the posterior distribution of the latent variables given by $p(Z|X, \theta^{\text{old}})$
 - Use $p(Z|X, \theta^{\text{old}})$ to compute the expectation of the complete-data log likelihood $\ln p(X, Z|\theta)$ under $p(Z|X, \theta^{\text{old}})$

$$Q(\theta, \theta^{\text{old}}) = \sum_Z p(Z|X, \theta^{\text{old}}) \ln p(X, Z|\theta)$$

E

- In M step, we need to compute θ^{new} which maximizes $Q(\theta, \theta^{\text{old}})$

$$\theta^{\text{new}} = \arg \max_{\theta} Q(\theta, \theta^{\text{old}})$$

M

Summary: EM

- EM is a general procedure for learning in the presence of unobserved variables.
- We have shown how to use it in order to estimate the most likely density function for a mixture of probability distributions.
- EM is an iterative algorithm that can be shown to converge to a local maximum of the likelihood function. Thus, might requires many restarts.
- It depends on assuming a family of probability distributions.
- It has been shown to be quite useful in practice, when the assumptions made on the probability distribution are correct, but can fail otherwise.

Lecture 11: Representing Probability Distribution

- **Goal:** To represent all joint probability distributions over a set of random variables X_1, X_2, \dots, X_n
- There are many ways to represent distributions.
- A table, listing the probability of each instance in $\{0,1\}^n$
 - We will need $2^n - 1$ numbers

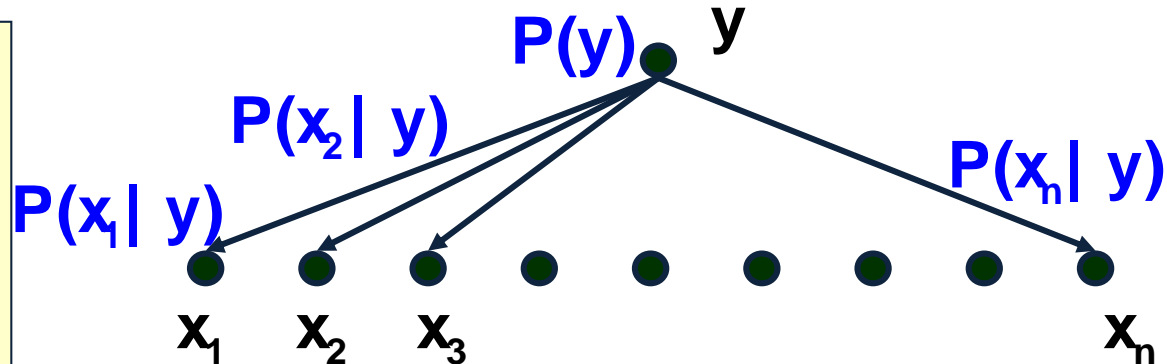
What can we do? Make Independence Assumptions

- Multi-linear polynomials
 - Polynomials over variables (Samdani & Roth'09, Poon & Domingos'11)
- ➔ Bayesian Networks
 - Directed acyclic graphs
- Markov Networks
 - Undirected graphs

Unsupervised Learning

- In general, the problem is very hard. But, under some assumptions on the distribution we have shown that we can do it. (exercise: show it's the most likely distribution)

We can compute the probability of any event or conditional event over the $n+1$ variables.

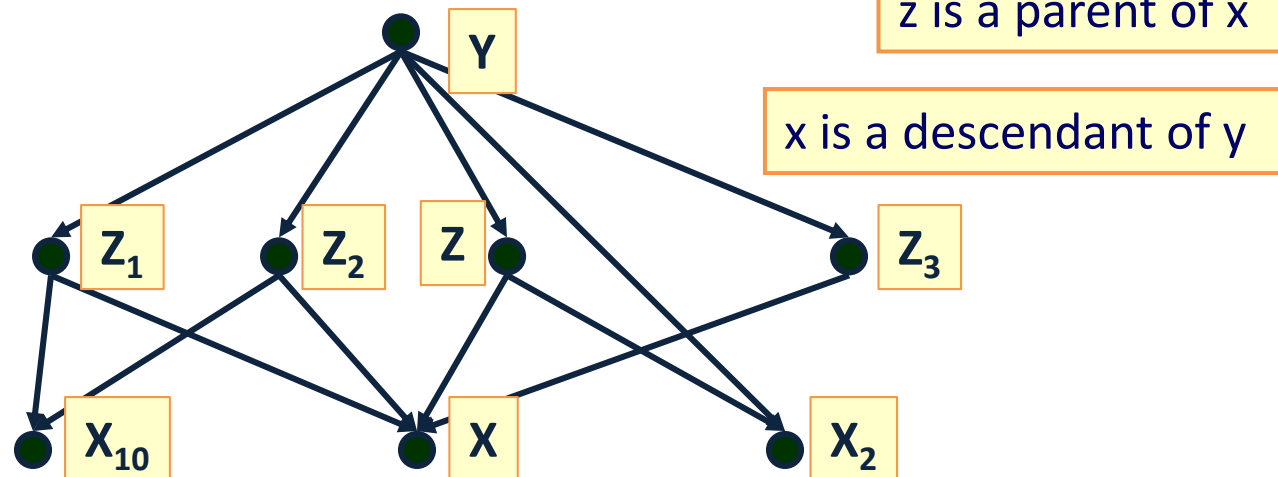


- Assumptions: (conditional independence given y)
 - $P(x_i | x_j, y) = P(x_i | y) \forall i, j$
- Can these assumptions be relaxed ?
- Can we learn more general probability distributions ?
 - (These are essential in many applications: language, vision.)

Graphical Models of Probability Distributions

- Bayesian Networks represent the joint probability distribution over a set of variables.
- Independence Assumption: $\forall x$, x is independent of its non-descendants given its parents

This is a theorem. To prove it, order the nodes from leaves up, and use the product rule. The terms are called **CPTs (Conditional Probability tables)** and they completely define the probability distribution.



- With these conventions, the joint probability distribution is given by:

$$P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$$

Bayesian Network

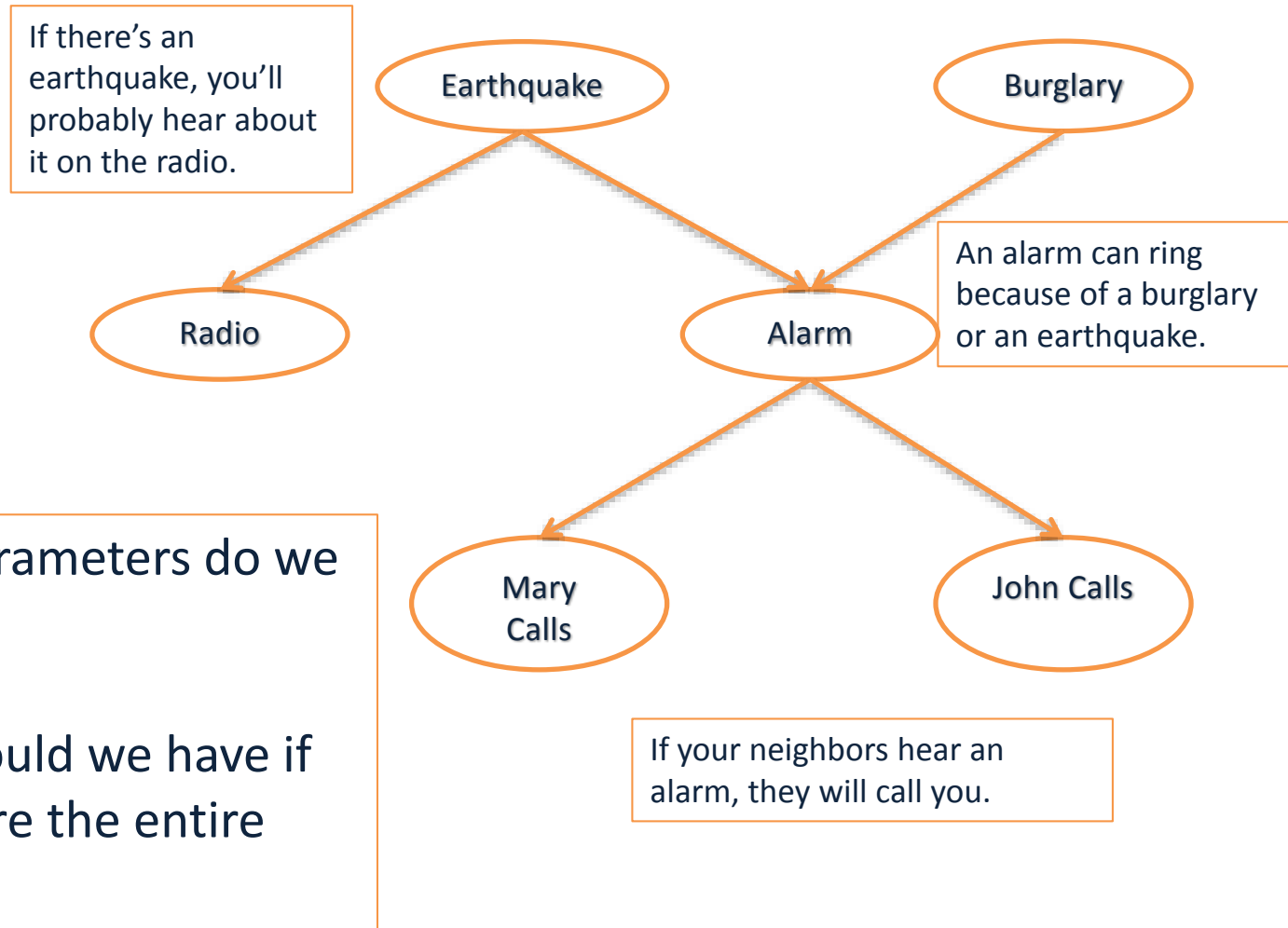
- Semantics of the DAG
 - Nodes are random variables
 - Edges represent causal influences
 - Each node is associated with a conditional probability distribution
- Two equivalent viewpoints
 - A data structure that represents the joint distribution compactly
 - A representation for a set of conditional independence assumptions about a distribution

Bayesian Network: Example

The burglar alarm in your house rings when there is a burglary or an earthquake. An earthquake will be reported on the radio. If an alarm rings and your neighbors hear it, they will call you.

What are the random variables?

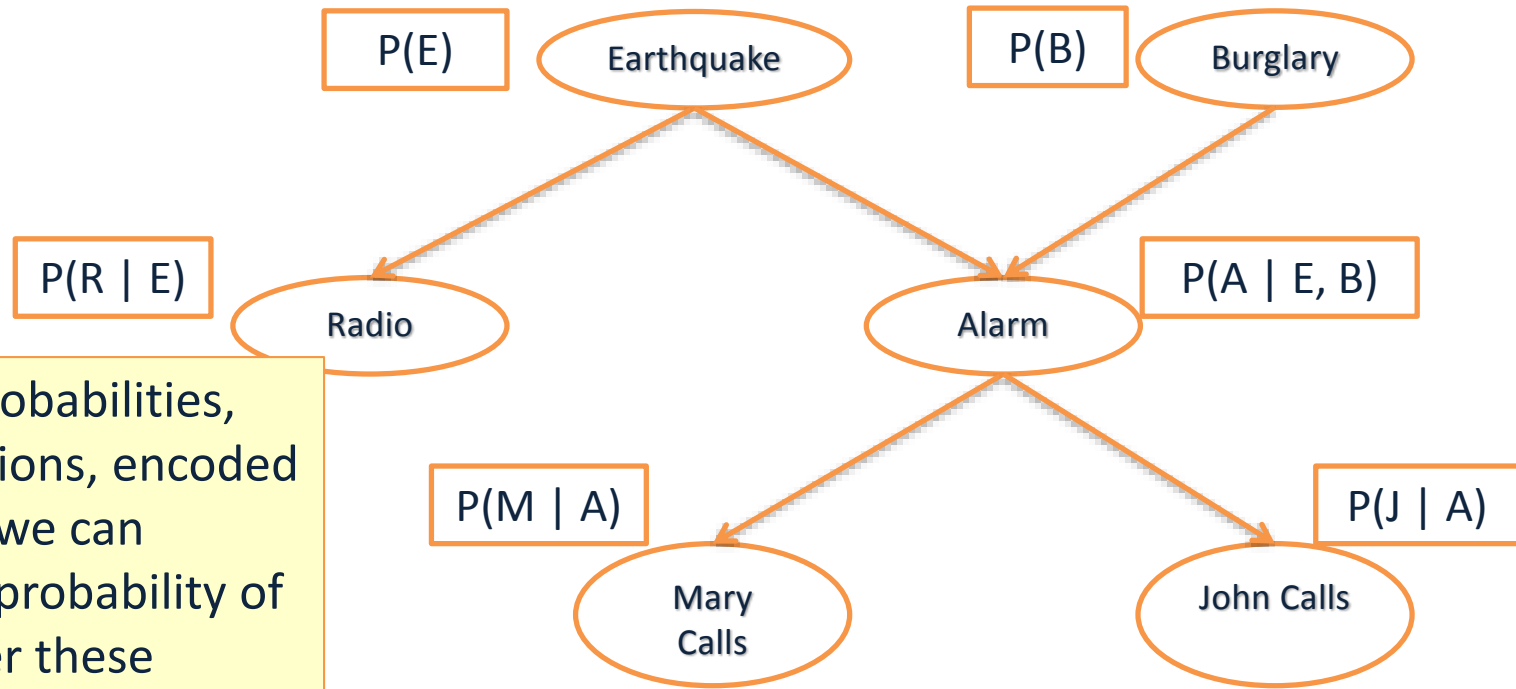
Bayesian Network: Example



How many parameters do we have?

How many would we have if we had to store the entire joint?

Bayesian Network: Example



With these probabilities, (and assumptions, encoded in the graph) we can compute the probability of any event over these variables.

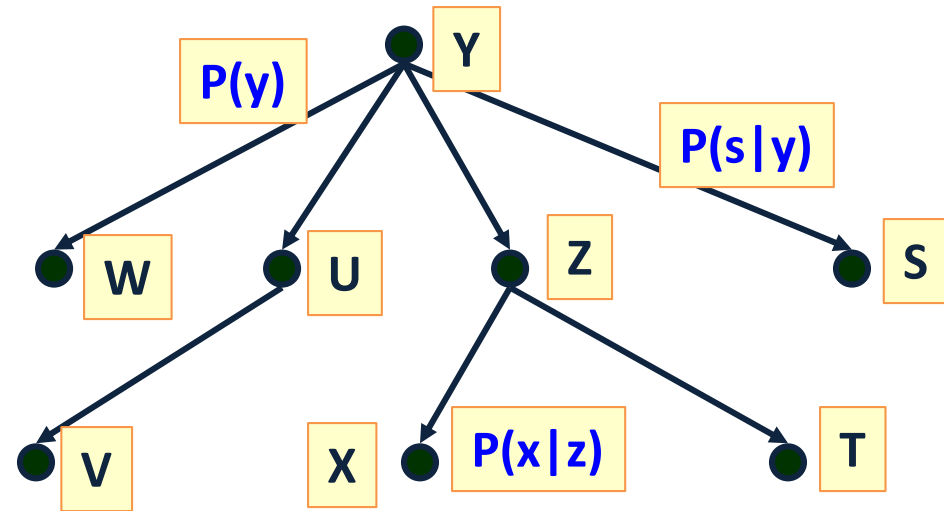
$$\begin{aligned} P(E, B, A, R, M, J) &= P(E | B, A, R, M, J)P(B, A, R, M, J) \\ &= P(E) \times P(B) \times P(R | E) \times P(A | E, B) \times P(M | A) \times P(J | A) \end{aligned}$$

Computational Problems

- Learning the **structure** of the Bayes net
 - (What would be the guiding principle?)
- Learning the **parameters**
 - Supervised? Unsupervised?
- **Inference:**
 - Computing the probability of an event: [#P Complete, Roth'93, '96]
 - Given structure and parameters
 - Given an observation E , what is the probability of Y ? $P(Y=y \mid E=e)$
 - (E, Y are sets of instantiated variables)
 - Most likely explanation (Maximum A Posteriori assignment, MAP, MPE) [NP-Hard; Shimony'94]
 - Given structure and parameters
 - Given an observation E , what is the most likely assignment to Y ?
 - $\text{Argmax}_y P(Y=y \mid E=e)$
 - (E, Y are sets of instantiated variables)

Tree Dependent Distributions

- Directed Acyclic graph
 - Each node has at most one parent
- Independence Assumption:
 - x is independent of its non-descendants given its parents
- (x is independent of other nodes given z ; v is independent of w given u ;

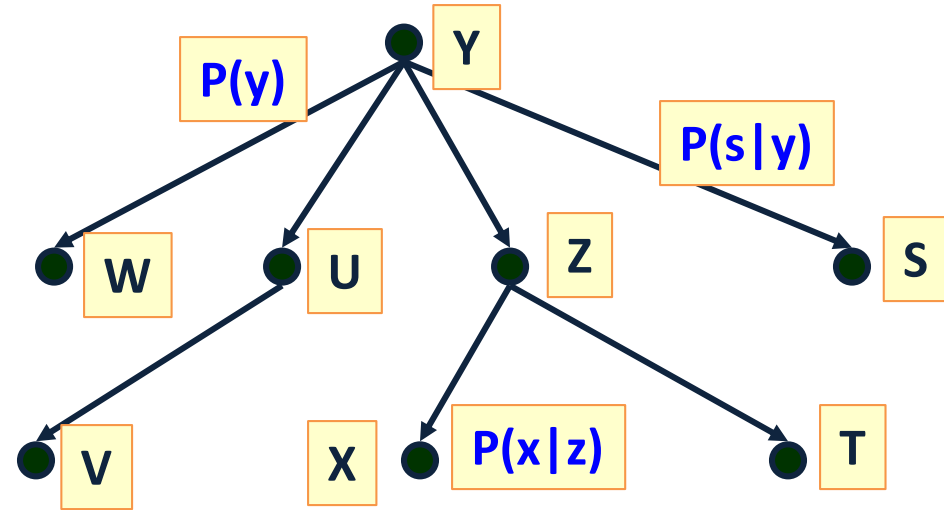


$$P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$$

- Need to know two numbers for each link: $p(x|z)$, and a prior for the root $p(y)$

Tree Dependent Distributions

- This is a generalization of naïve Bayes.
- Inference Problem:
 - Given the Tree with all the associated probabilities, evaluate the probability of an event $p(x)$?



- $P(x=1) = P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$
 $= P(x=1 | z=1)P(z=1) + P(x=1 | z=0)P(z=0)$

- Recursively, go up the tree:

$$P(z=1) = P(z=1 | y=1)P(y=1) + P(z=1 | y=0)P(y=0)$$

$$P(z=0) = P(z=0 | y=1)P(y=1) + P(z=0 | y=0)P(y=0)$$

Linear Time Algorithm

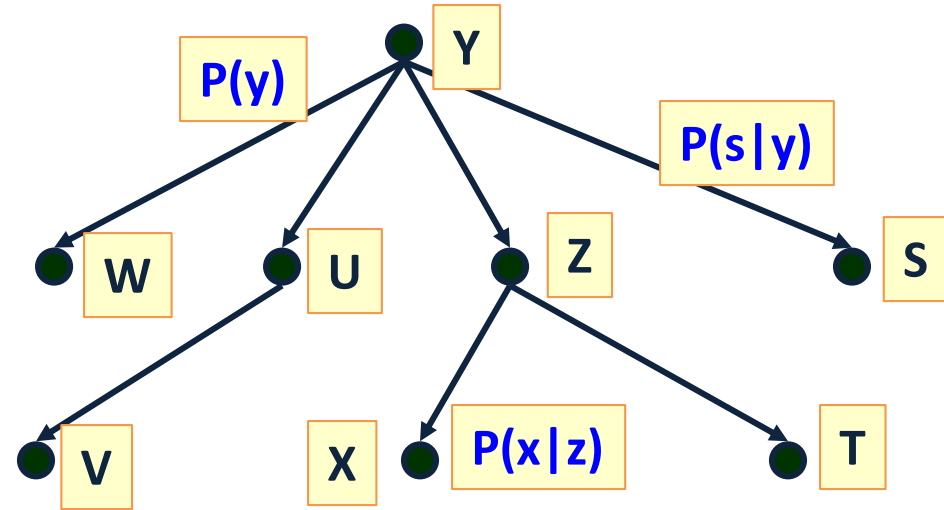
Now we have everything in terms of the CPTs (conditional probability tables)

Tree Dependent Distributions

- This is a generalization of naïve Bayes.

- **Inference Problem:**

- Given the Tree with all the associated probabilities, evaluate the probability of an event $p(x,y)$?



- $P(x=1, y=0) = P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$
 $= P(x=1 | y=0)P(y=0)$

- Recursively, go up the tree along the path from x to y :

$$P(x=1 | y=0) = \sum_{z=0,1} P(x=1 | y=0, z)P(z | y=0) = \sum_{z=0,1} P(x=1 | z)P(z | y=0)$$

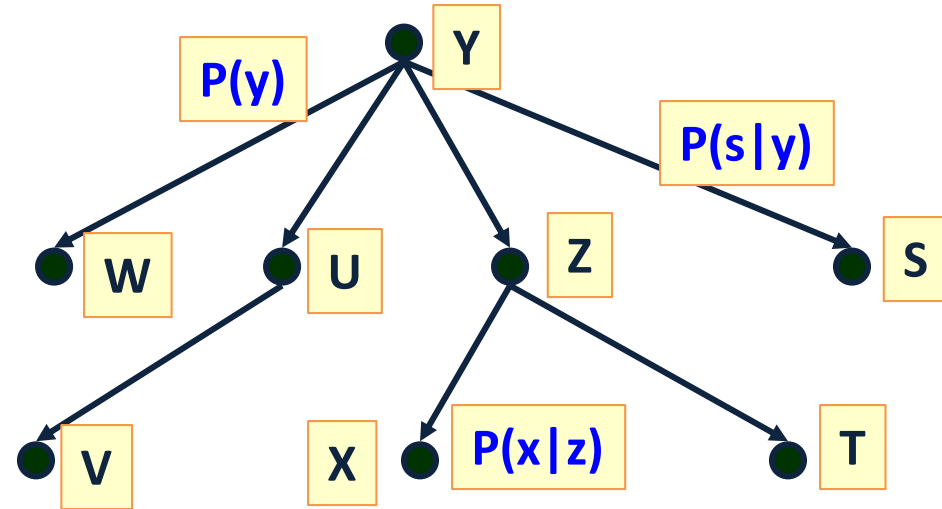
Now we have everything in terms of the CPTs (conditional probability tables)

Tree Dependent Distributions

- This is a generalization of naïve Bayes.

- **Inference Problem:**

- Given the Tree with all the associated probabilities, evaluate the probability of an event $p(x,u)$?



- (No direct path from x to u) $P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$

- $P(x=1, u=0) = P(x=1 | u=0)P(u=0)$

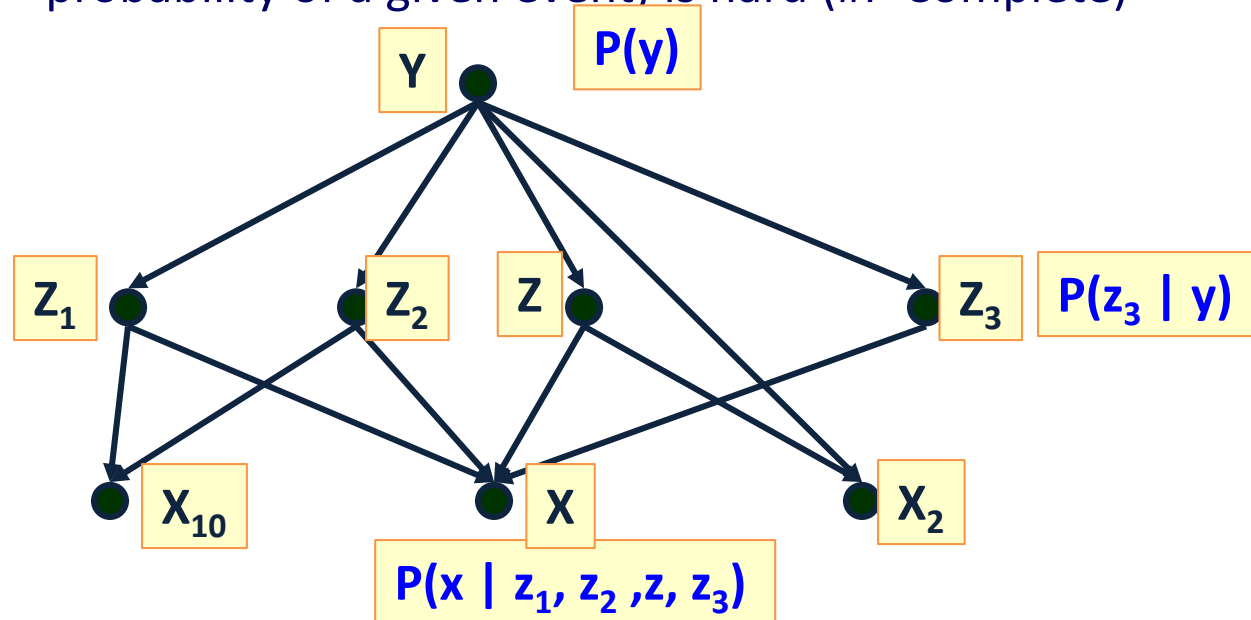
- Let y be a parent of x and u (we always have one)

$$\begin{aligned}
 P(x=1 | u=0) &= \sum_{y=0,1} P(x=1 | u=0, y)P(y | u=0) = \\
 &= \sum_{y=0,1} P(x=1 | y)P(y | u=0) =
 \end{aligned}$$

Now we have reduced it to cases we have seen

Graphical Models of Probability Distributions

- For general Bayesian Networks
 - The learning problem is hard
 - The inference problem (given the network, evaluate the probability of a given event) is hard (#P Complete)



$$P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$$

Tree Dependent Distributions

- Learning Problem:

- Given data (n tuples) assumed to be sampled from a tree-dependent distribution

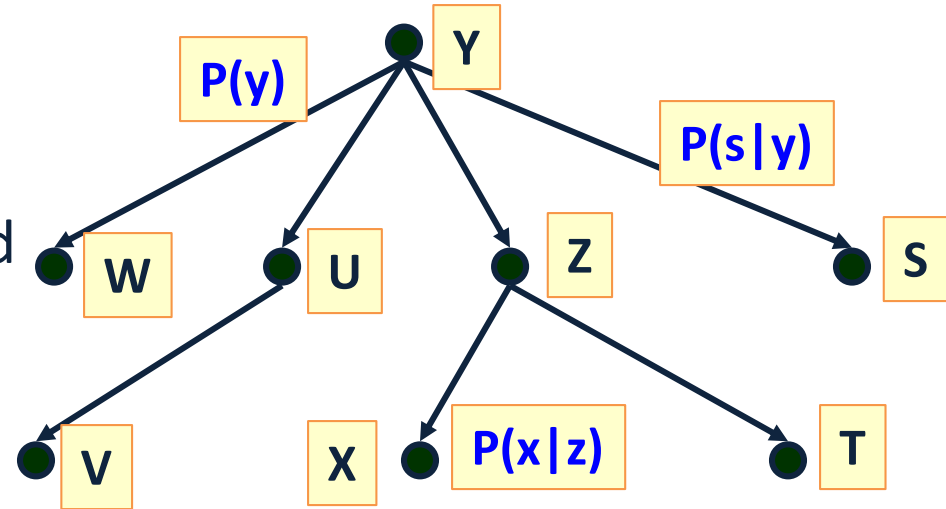
- What does that mean?
- Generative model

- Find the tree representation of the distribution.

- What does that mean?

- Among all trees, find the most likely one, given the data:

$$P(T|D) = P(D|T) P(T)/P(D)$$



$$P(y, x_1, x_2, \dots, x_n) = p(y) \prod_i P(x_i | \text{Parents}(x_i))$$

Tree Dependent Distributions

- **Learning Problem:**

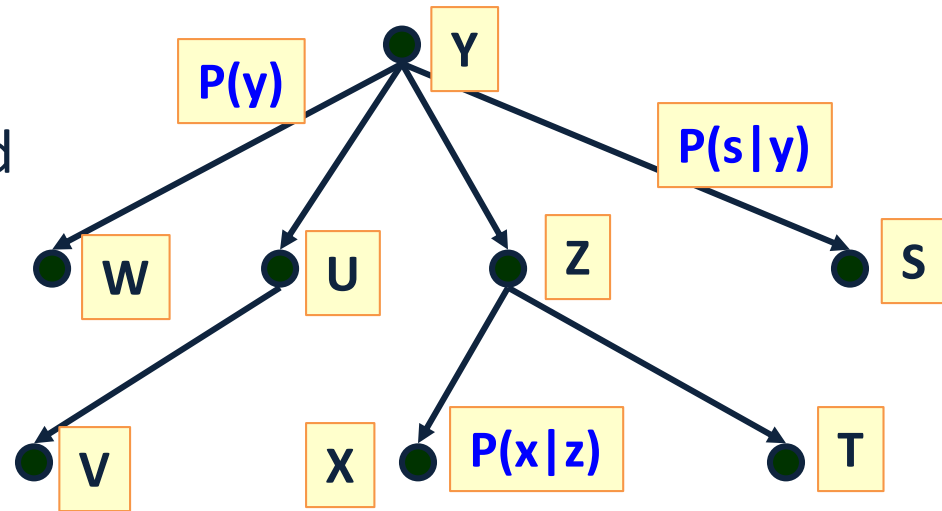
- Given data (n tuples) assumed to be sampled from a tree-dependent distribution

- Find the tree representation of the distribution.

- Assuming uniform prior on trees, the **Maximum Likelihood** approach is to maximize $P(D|T)$,

$$T_{ML} = \operatorname{argmax}_T P(D|T) = \operatorname{argmax}_{T \{x\}} P_T(x_1, x_2, \dots, x_n)$$

- Now we can see why we had to solve the inference problem first; it is required for learning.

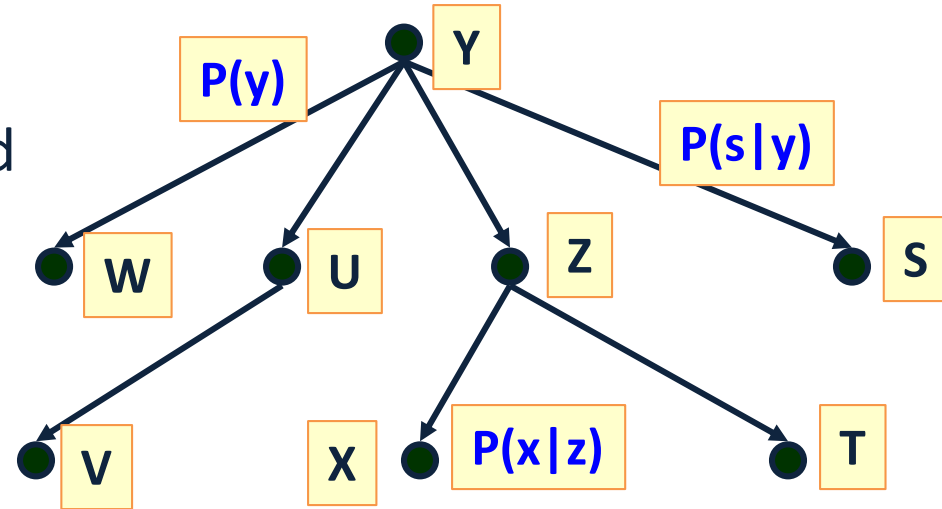


Tree Dependent Distributions

- **Learning Problem:**

- Given data (n tuples) assumed to be sampled from a tree-dependent distribution

- Find the tree representation of the distribution.



- Assuming uniform prior on trees, the **Maximum Likelihood** approach is to maximize $P(D|T)$,

$$T_{ML} = \operatorname{argmax}_T P(D|T) = \operatorname{argmax}_T \prod_{\{x\}} P_T(x_1, x_2, \dots, x_n) =$$

$$= \operatorname{argmax}_T \prod_{\{x\}} \prod_i P_T(x_i | \text{Parents}(x_i))$$

- Try this for naïve Bayes

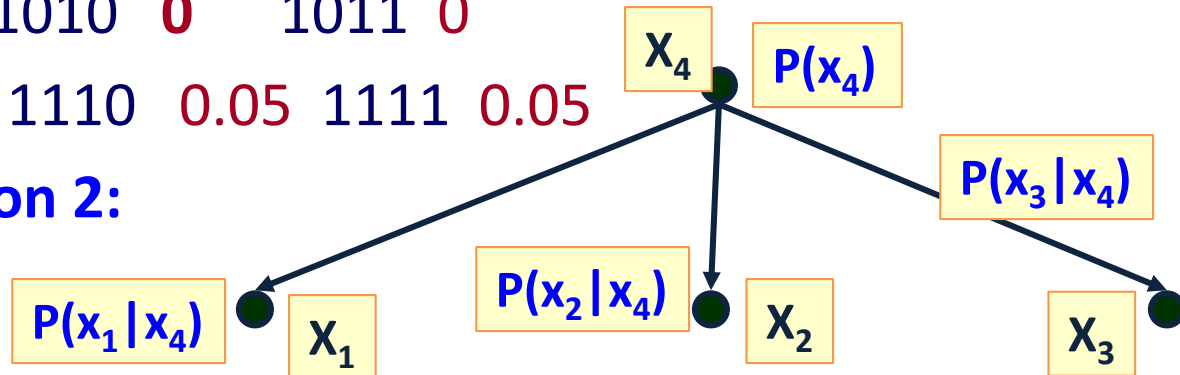
Example: Learning Distributions

Probability Distribution 1:

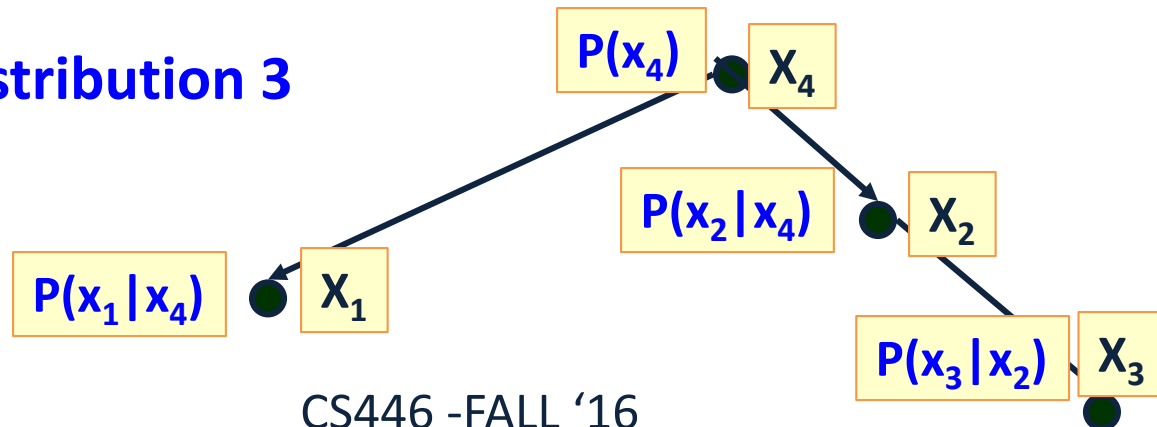
0000	0.1	0001	0.1	0010	0.1	0011	0.1
0100	0.1	0101	0.1	0110	0.1	0111	0.1
1000	0	1001	0	1010	0	1011	0
1100	0.05	1101	0.05	1110	0.05	1111	0.05

Are these representations of the same distribution?
Given a sample, which of these generated it?

Probability Distribution 2:



Probability Distribution 3



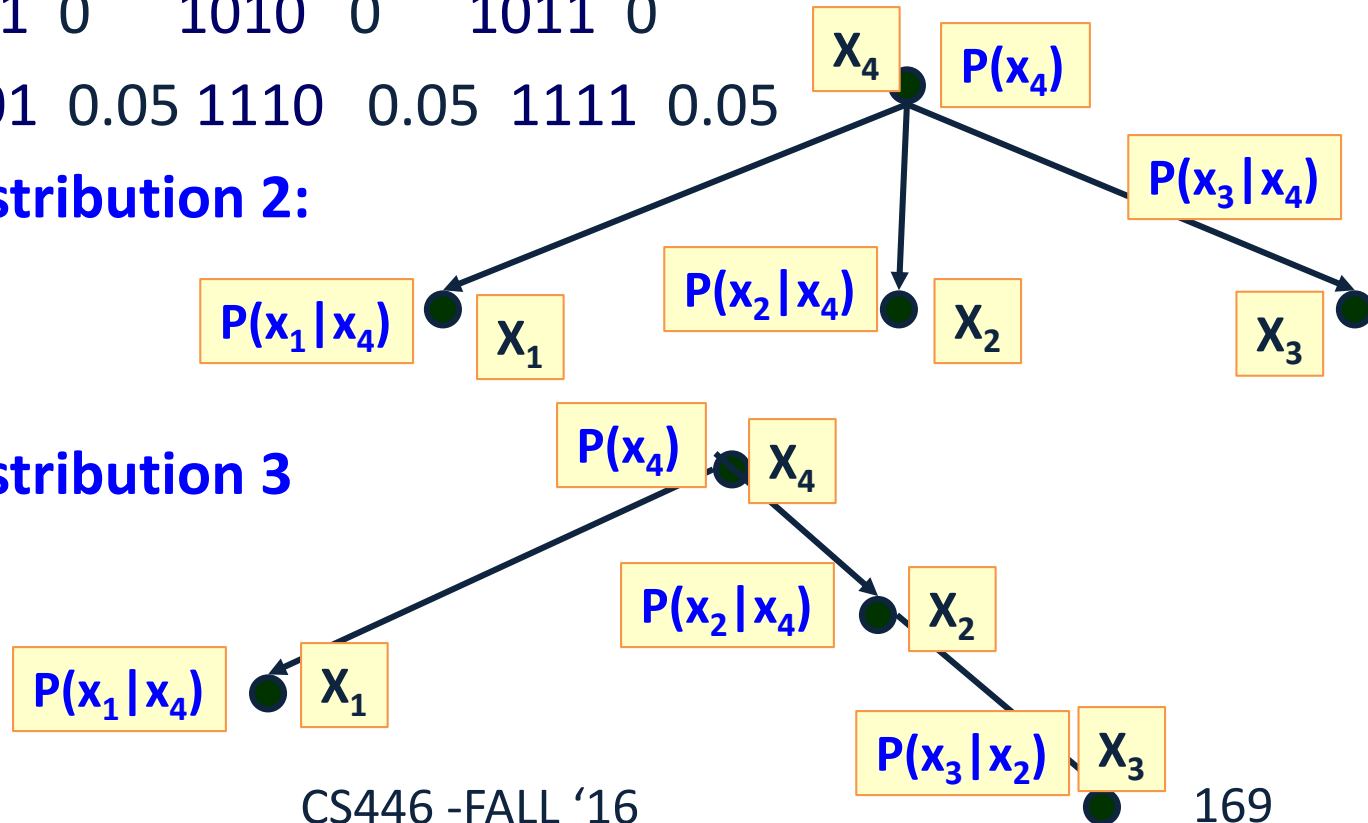
Example: Learning Distributions

Probability Distribution 1:

0000	0.1	0001	0.1	0010	0.1	0011	0.1
0100	0.1	0101	0.1	0110	0.1	0111	0.1
1000	0	1001	0	1010	0	1011	0
1100	0.05	1101	0.05	1110	0.05	1111	0.05

We are given 3 data points: 1011; 1001; 0100
Which one is the target distribution?

Probability Distribution 2:



Probability Distribution 3

Example: Learning Distributions

■ Probability Distribution 1:

0000	0.1	0001	0.1	0010	0.1	0011	0.1
0100	0.1	0101	0.1	0110	0.1	0111	0.1
1000	0	1001	0	1010	0	1011	0
1100	0.05	1101	0.05	1110	0.05	1111	0.05

We are given 3 data points: 1011; 1001; 0100
Which one is the target distribution?

■ What is the likelihood that this table generated the data?

$$P(T|D) = P(D|T) P(T)/P(D)$$

■ Likelihood(T) $\approx P(D|T) \approx P(1011|T) P(1001|T)P(0100|T)$

- $P(1011|T) = 0$
- $P(1001|T) = 0.1$
- $P(0100|T) = 0.1$

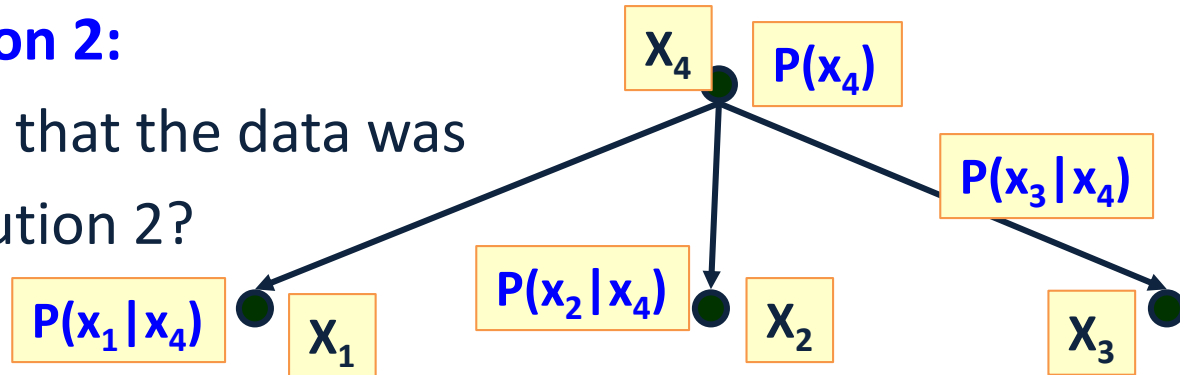
■ $P(\text{Data} | \text{Table}) = 0$

Example: Learning Distributions

Probability Distribution 2:

What is the likelihood that the data was sampled from Distribution 2?

Need to define it:



- $P(x_4=1)=1/2$

- $p(x_1=1 | x_4=0)=1/2$

- $p(x_2=1 | x_4=0)=1/3$

- $p(x_3=1 | x_4=0)=1/6$

- $p(x_1=1 | x_4=1)=1/2$

- $p(x_2=1 | x_4=1)=1/3$

- $p(x_3=1 | x_4=1)=5/6$

Likelihood(T) \sim P(D | T) \sim P(1011 | T) P(1001 | T) P(0100 | T)

- $P(1011 | T) = p(x_4=1)p(x_1=1 | x_4=1)p(x_2=0 | x_4=1)p(x_3=1 | x_4=1) = 1/2 \cdot 1/2 \cdot 2/3 \cdot 5/6 = 10/72$

- $P(1001 | T) = 1/2 \cdot 1/2 \cdot 2/3 \cdot 5/6 = 10/72$

- $P(0100 | T) = 1/2 \cdot 1/2 \cdot 2/3 \cdot 5/6 = 10/72$

- $P(\text{Data} | \text{Tree}) = 125/4 \cdot 3^6$

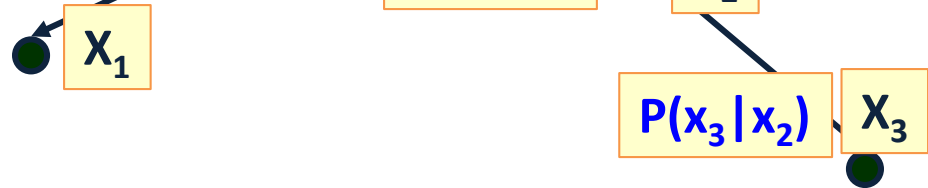
Example: Learning Distributions

Probability Distribution 3:

What is the likelihood that the data was sampled from Distribution 2?

Need to define it:

$$P(x_1 | x_4)$$



$$P(x_4=1)=2/3$$

$$p(x_1=1 | x_4=0)=1/3$$

$$p(x_2=1 | x_4=0)=1$$

$$p(x_3=1 | x_2=0)=2/3$$

$$p(x_1=1 | x_4=1)=1$$

$$p(x_2=1 | x_4=1)=1/2$$

$$p(x_3=1 | x_2=1)=1/6$$

Likelihood(T) \sim P(D | T) \sim P(1011 | T) P(1001 | T) P(0100 | T)

$$P(1011 | T) = p(x_4=1)p(x_1=1 | x_4=1)p(x_2=0 | x_4=1)p(x_3=1 | x_2=1) = 2/3 \cdot 1 \cdot 1/2 \cdot 2/3 = 2/9$$

$$P(1001 | T) = 2/3 \cdot 1 \cdot 1/2 \cdot 1/3 = 1/9$$

$$P(0100 | T) = 1/3 \cdot 2/3 \cdot 1 \cdot 5/6 = 10/54$$

$$P(\text{Data} | \text{Tree}) = 10/3^7$$

Distribution 2 is the most likely distribution to have produced the data.

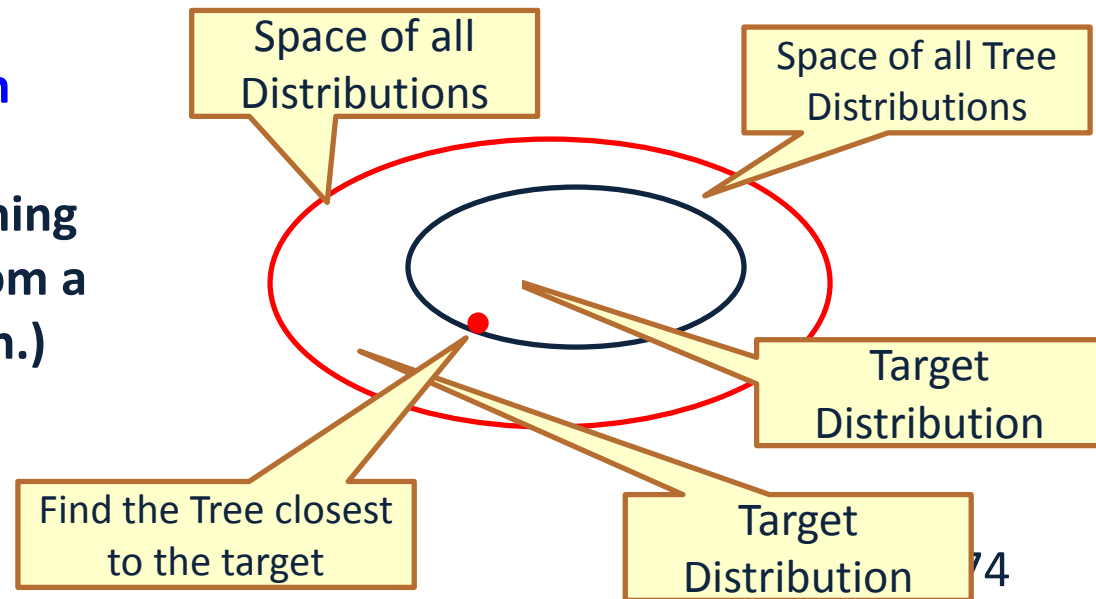
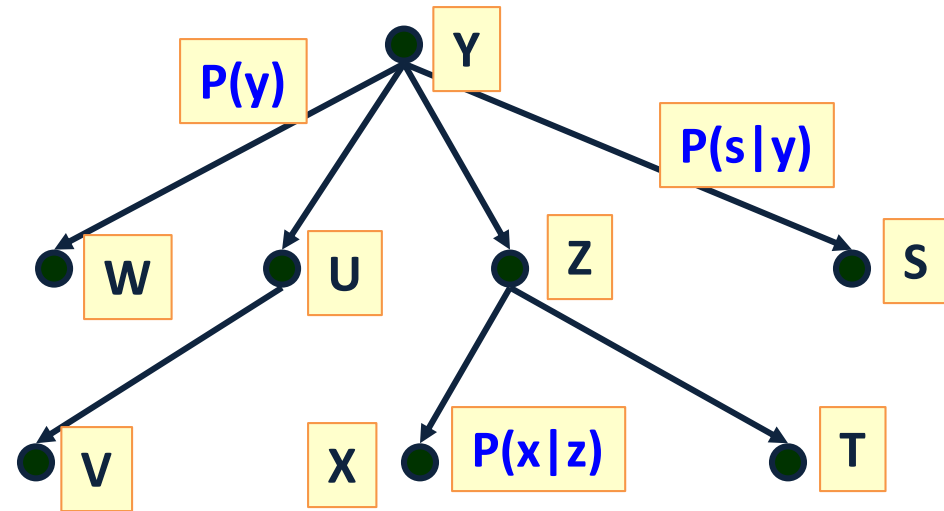
Example: Summary

- We are now in the same situation we were when we decided which of two coins, fair (0.5,0.5) or biased (0.7,0.3) generated the data.
- But, this isn't the most interesting case.
- In general, we will not have a small number of possible distributions to choose from, but rather a parameterized family of distributions. (analogous to a coin with $p \in [0,1]$)
- We need a systematic way to search this family of distributions.

Learning Tree Dependent Distributions

Learning Problem:

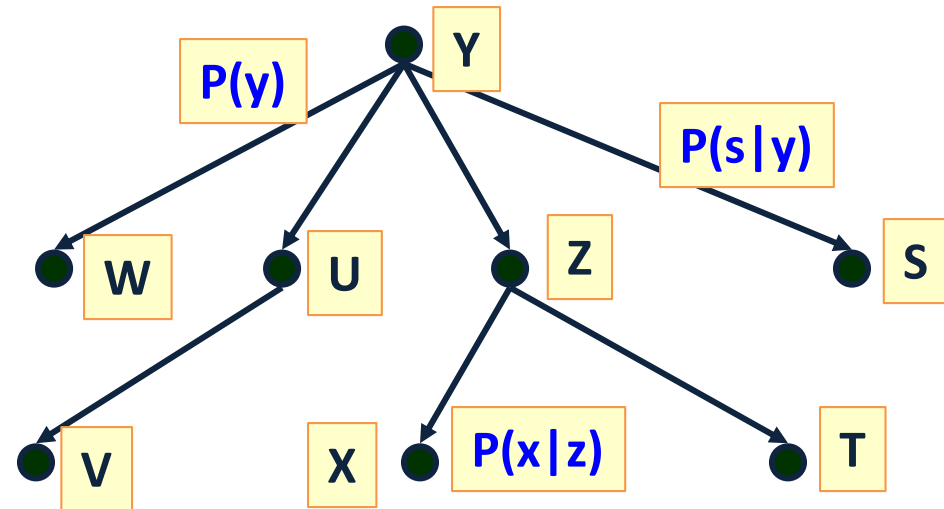
- 1. Given data (n tuples) assumed to be sampled from a tree-dependent distribution
- find the **most probable tree representation** of the distribution.
- 2. Given data (n tuples)
- find the **tree representation that best approximates the distribution** (without assuming that the data is sampled from a tree-dependent distribution.)



Learning Tree Dependent Distributions

Learning Problem:

- 1. Given data (n tuples) assumed to be sampled from a tree-dependent distribution
- find the **most probable tree representation** of the distribution.
- 2. Given data (n tuples)
- find the **tree representation that best approximates the distribution** (without assuming that the data is sampled from a tree-dependent distribution.)



The simple minded algorithm for learning a tree dependent distribution requires

- (1) for each tree, compute its likelihood

$$\begin{aligned}
 L(T) &= P(D|T) = \\
 &= \prod_{\{x\}} P_T(x_1, x_2, \dots, x_n) = \\
 &= \prod_{\{x\}} \prod_i P_T(x_i | \text{Parents}(x_i))
 \end{aligned}$$

- (2) Find the maximal one

1. Distance Measure

- To measure how well a probability distribution P is approximated by probability distribution T we use here the Kullback-Leibler cross entropy measure (KL-divergence):

$$D(P, T) = \sum_x P(\mathbf{x}) \log \frac{P(\mathbf{x})}{T(\mathbf{x})}$$

- Non negative.
- $D(P, T) = 0$ iff P and T are identical
- Non symmetric. Measures how much P differs from T .

2. Ranking Dependencies

- Intuitively, the important edges to keep in the tree are edges $(x---y)$ for x, y which depend on each other.
- Given that the distance between the distribution is measured using the KL divergence, the corresponding measure of dependence is the **mutual information between x and y** , (measuring the information x gives about y)

$$\mathbf{I}(\mathbf{x}, \mathbf{y}) = \sum_{x, y} \mathbf{P}(\mathbf{x}, \mathbf{y}) \log \frac{\mathbf{P}(\mathbf{x}, \mathbf{y})}{\mathbf{P}(\mathbf{x})\mathbf{P}(\mathbf{y})}$$

- which we can estimate with respect to the empirical distribution (that is, the given data).

Learning Tree Dependent Distributions

- The algorithm is given m independent measurements from P .
- For each variable x , estimate $P(x)$ (Binary variables – n numbers)
- For each pair of variables x, y , estimate $P(x,y)$ ($O(n^2)$ numbers)
- For each pair of variables compute the mutual information
- Build a complete undirected graph with all the variables as vertices.
- Let $I(x,y)$ be the weights of the edge (x,y)
- Build a maximum weighted spanning tree

Learning Tree Dependent Distributions

- The algorithm is given m independent measurements from P .
 - For each variable x , estimate $P(x)$ (Binary variables – n numbers)
 - For each pair of variables x, y , estimate $P(x,y)$ ($O(n^2)$ numbers)
 - For each pair of variables compute the mutual information
 - Build a complete undirected graph with all the variables as vertices.
- (2) ■ Let $I(x,y)$ be the weights of the edge (x,y)
- Build a maximum weighted spanning tree
- (3) ■ Transform the resulting undirected tree to a directed tree.
- Choose a root variable and set the direction of all the edges away from it.
- (1) ■ Place the corresponding conditional probabilities on the edges.