Administration

- Midterm exam on Tuesday 10/25
  - Closed books; in class; ~4 questions
  - All the material covered before the midterm
  - Go over practice midterms

Questions?

Midterm Review

- Today:
- Quick run-through of the material we've covered so far

- The selection of slides in today's lecture doesn't mean that you don't need to look at the rest when prepping for the exam!

- Slides are from previous lectures
  - I'll not go in to the details
  - Slides chosen might be not completely coherent
  - The goal is to remind you what we did and solicit questions

Midterm

- Closed book exam

- All lectures until today
- Intro. to ML / Decision Trees / Online learning / COLT /NN/ Boosting/SVM
  - Lectures / Problem sets

- Cheating?
  - No.

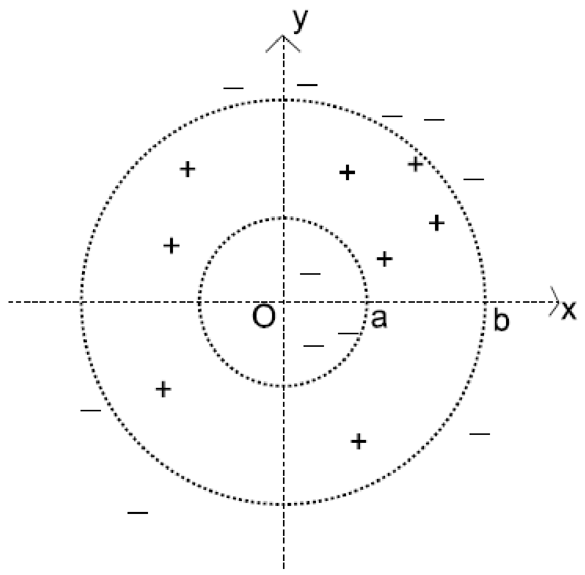
Sample Questions

- Question types:
  4~5 question sets including a set of short questions

- Previous midterm exams / solutions:
- http://l2r.cs.illinois.edu/~danr/Teaching/CS446-16/handout.html

Sample of short Question

(a) [6 points] Consider a concept space **H** of two nested circles centered on the origin (see figure below). Formally, a concept $h \in \mathbf{H}$ is defined by 2 non-negative real parameters $a, b \in \mathbb{R}^+$ such that $a < b$. An example $(x, y) \in \mathbb{R}^2$ is labeled $+1$ if and only if $a^2 < x^2 + y^2 < b^2$, i.e. (x, y) is within the band of the two nested circles of radius a and b respectively.



State the VC-dimension of **H**. Prove that your answer is correct.

Sample Question set

Perceptrons [25 points]

In this question, we will be asking you about Perceptrons and their variants.

Let $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$, where the j-th example $\mathbf{x}^{(j)}$ is associated with the label $y^{(j)} \in \{-1, +1\}$. Each example $\mathbf{x}^{(j)}$ is a bit-vector of length n, i.e. $\mathbf{x}^{(j)} \in \{0,1\}^n$, with the interpretation that the i-th bit of the vector $(x_i^{(j)})$ is 1 if the element described by $\mathbf{x}^{(j)}$ has the i-th attribute on.

(a) [7 points] Let us first consider a Perceptron where the positive example $\mathbf{x}$ satisfies $\mathbf{w} \cdot \mathbf{x} \geq \theta$, where $\mathbf{w} \in \mathbb{R}^n$, $\theta \in \mathbb{R}$ and $\mathbf{x}$ is some example $\mathbf{x}^{(j)}$ from D.

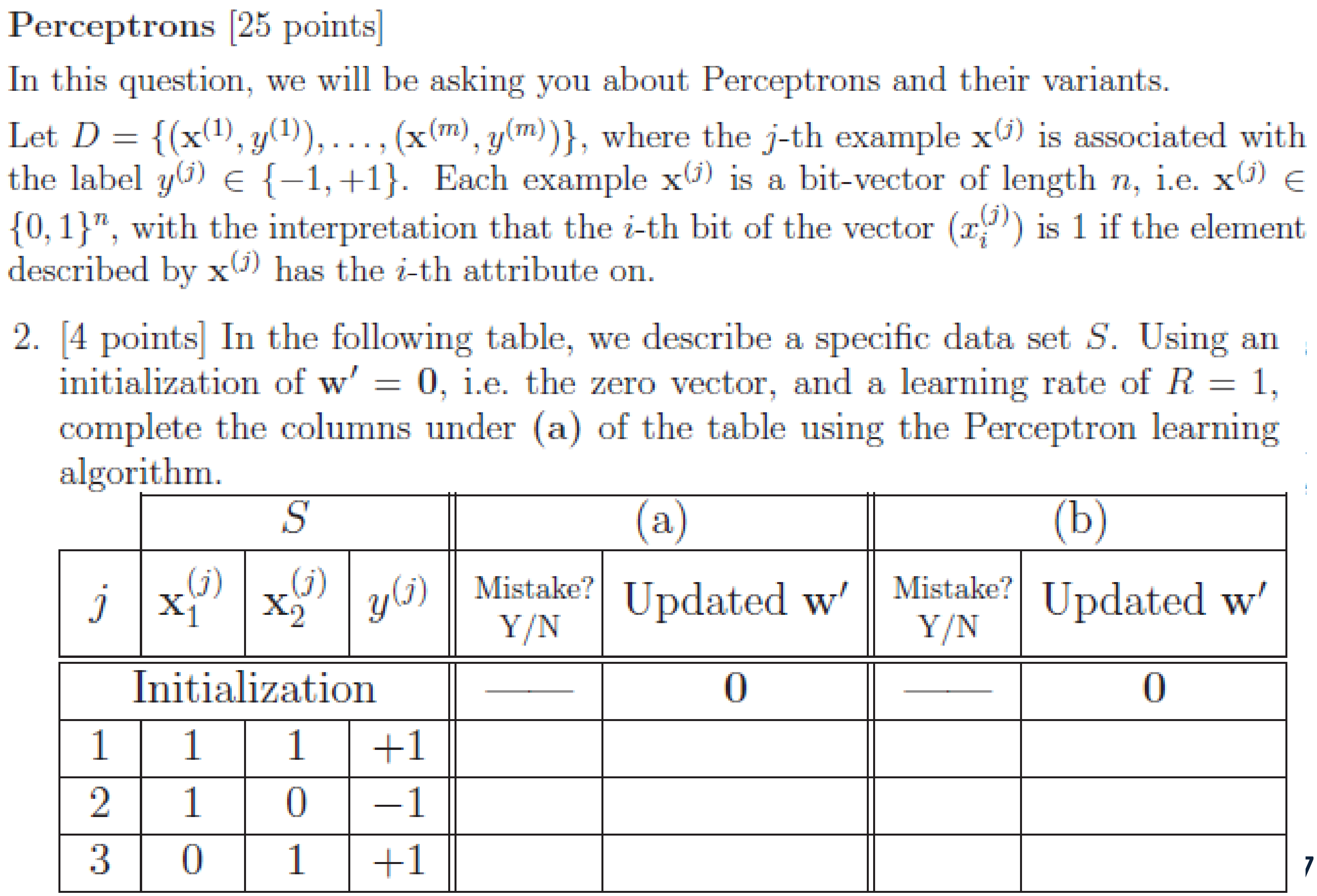
1. [3 points] Suggest an equivalent representation of this Perceptron in the form of $\underline{\mathbf{w}' \cdot \mathbf{x}' \geq 0}$ given an example $\mathbf{x}^{(j)}$, where $\mathbf{x}' \in \{0,1\}^{n'}$ for some suitable integer n'.

   Define $n' =$ ________________

   Define $\mathbf{w}' =$ ________________

   Define $\mathbf{x}' =$ ________________

Sample Question set



Perceptrons [25 points]

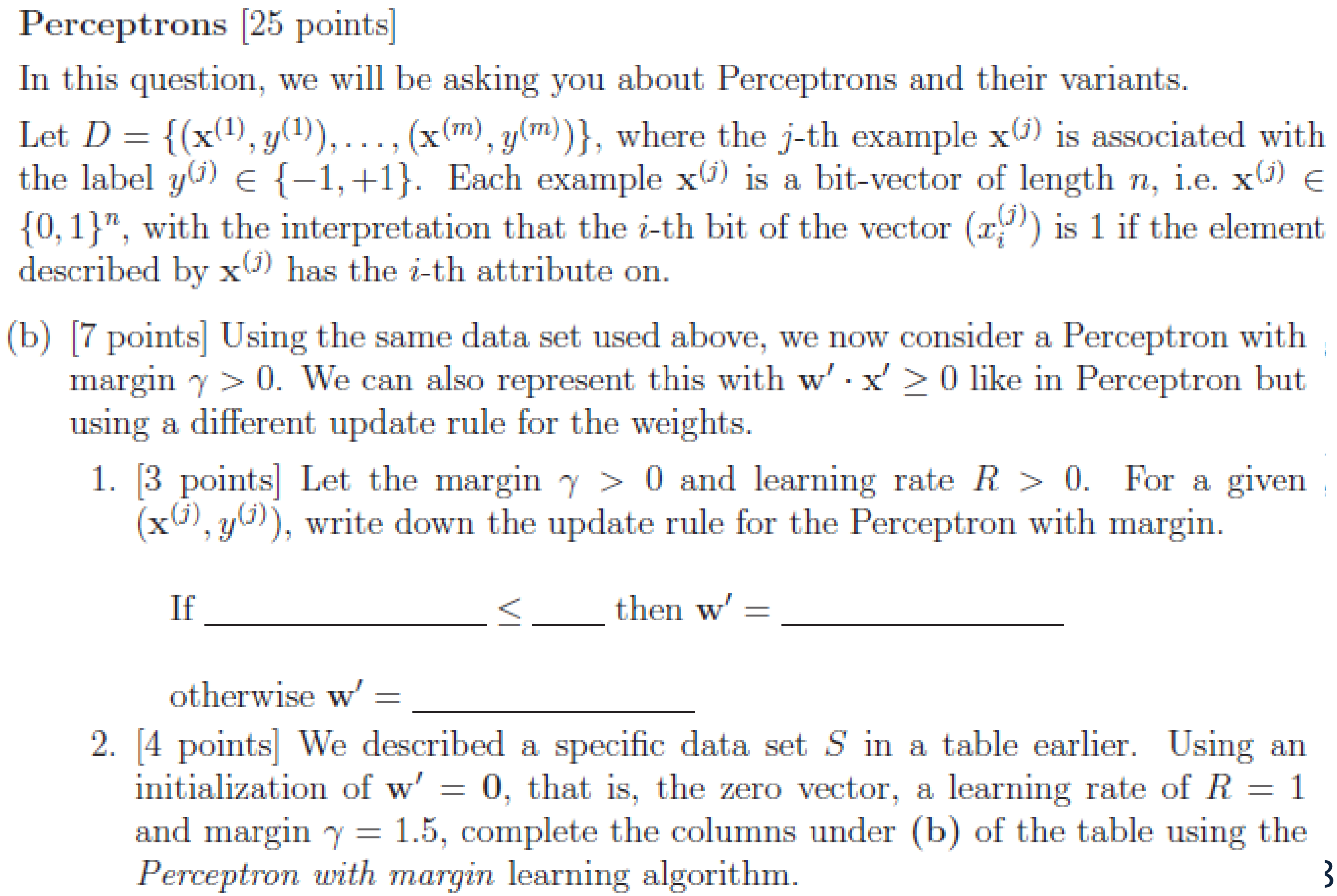
In this question, we will be asking you about Perceptrons and their variants.

Let $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$, where the j-th example $\mathbf{x}^{(j)}$ is associated with the label $y^{(j)} \in \{-1, +1\}$. Each example $\mathbf{x}^{(j)}$ is a bit-vector of length n, i.e. $\mathbf{x}^{(j)} \in \{0,1\}^n$, with the interpretation that the i-th bit of the vector $(x_i^{(j)})$ is 1 if the element described by $\mathbf{x}^{(j)}$ has the i-th attribute on.

2. [4 points] In the following table, we describe a specific data set S. Using an initialization of $\mathbf{w}' = \mathbf{0}$, i.e. the zero vector, and a learning rate of $R = 1$, complete the columns under **(a)** of the table using the Perceptron learning algorithm.

	S			(a)		(b)	
j	$\mathbf{x}_1^{(j)}$	$\mathbf{x}_2^{(j)}$	$y^{(j)}$	Mistake? Y/N	Updated $\mathbf{w}'$	Mistake? Y/N	Updated $\mathbf{w}'$
Initialization				——	$\mathbf{0}$	——	$\mathbf{0}$
1	1	1	+1				
2	1	0	−1				
3	0	1	+1				

Sample Question set



Perceptrons [25 points]

In this question, we will be asking you about Perceptrons and their variants.

Let $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$, where the j-th example $\mathbf{x}^{(j)}$ is associated with the label $y^{(j)} \in \{-1, +1\}$. Each example $\mathbf{x}^{(j)}$ is a bit-vector of length n, i.e. $\mathbf{x}^{(j)} \in \{0, 1\}^n$, with the interpretation that the i-th bit of the vector $(x_i^{(j)})$ is 1 if the element described by $\mathbf{x}^{(j)}$ has the i-th attribute on.

(b) [7 points] Using the same data set used above, we now consider a Perceptron with margin $\gamma > 0$. We can also represent this with $\mathbf{w}' \cdot \mathbf{x}' \geq 0$ like in Perceptron but using a different update rule for the weights.

1. [3 points] Let the margin $\gamma > 0$ and learning rate $R > 0$. For a given $(\mathbf{x}^{(j)}, y^{(j)})$, write down the update rule for the Perceptron with margin.

   If $\underline{\qquad\qquad\qquad} \leq \underline{\qquad}$ then $\mathbf{w}' = \underline{\qquad\qquad\qquad}$

   otherwise $\mathbf{w}' = \underline{\qquad\qquad\qquad}$

2. [4 points] We described a specific data set S in a table earlier. Using an initialization of $\mathbf{w}' = \mathbf{0}$, that is, the zero vector, a learning rate of $R = 1$ and margin $\gamma = 1.5$, complete the columns under (b) of the table using the *Perceptron with margin* learning algorithm.

Course Overview

- Introduction: Basic problems and questions
- A detailed example: Linear threshold units
- Two Basic Paradigms:
  - PAC (Risk Minimization)
  - Bayesian theory
- Learning Protocols:
  - Supervised; Unsupervised; Semi-supervised
- Algorithms
  - Decision Trees (C4.5)
  - [Rules and ILP (Ripper, Foil)]
  - Linear Threshold Units (Winnow; Perceptron; Boosting; SVMs; Kernels)
    - Gradient Descent
  - Neural Networks (Backpropagation)
  - Probabilistic Representations (naïve Bayes; Bayesian trees; Densities)
  - Unsupervised /Semi supervised: EM
- Clustering; Dimensionality Reduction

Key Issues in Machine Learning

- Modeling
  - How to formulate application problems as machine learning problems ? How to represent the data?
  - Learning Protocols (where is the data & labels coming from?)
- Representation
  - What are good hypothesis spaces ?
  - Any rigorous way to find these? Any general approach?
- Algorithms
  - What are good algorithms?
  - How do we define success?
  - Generalization Vs. over fitting
  - The computational problem

Using supervised learning

- What is our instance space?
  - Gloss: What kind of features are we using?
- What is our label space?
  - Gloss: What kind of learning task are we dealing with?
- What is our hypothesis space?
  - Gloss: What kind of model are we learning?
- What learning algorithm do we use?
  - Gloss: How do we learn the model from the labeled data?

(What is our loss function/evaluation metric?)
  - Gloss: How do we measure success?

Terminology

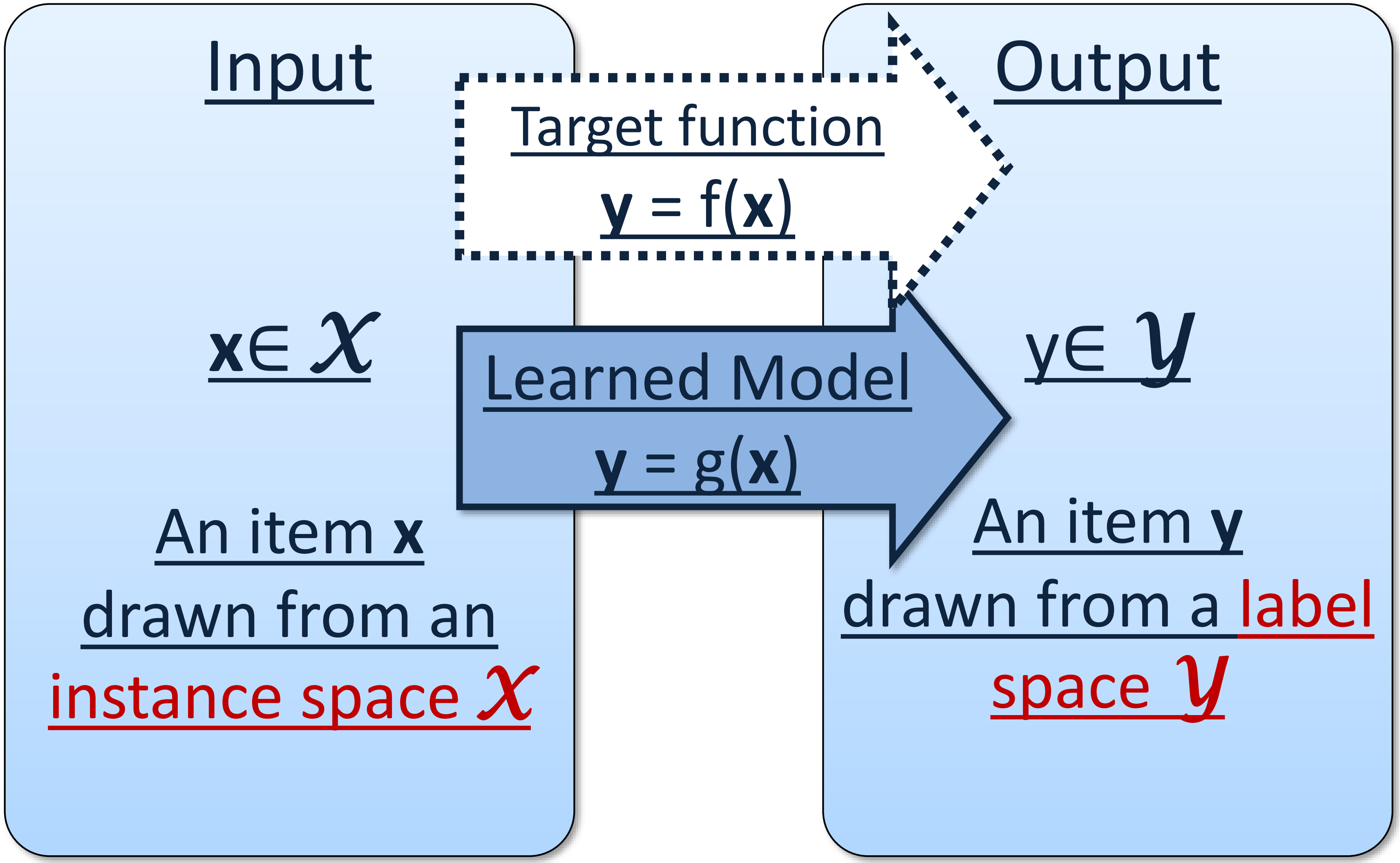
- Target function (concept): The true function f :X →{…Labels…}
- Concept: Boolean function. Example for which f (x)= 1 are positive examples; those for which f (x)= 0 are negative examples (instances)

- Hypothesis: A proposed function h, believed to be similar to f. The output of our learning algorithm.
- Hypothesis space: The space of all hypotheses that can, in principle, be output by the learning algorithm.

- Classifier: A discrete valued function produced by the learning algorithm. The possible value of f: {1,2,…K} are the classes or class labels. (In most algorithms the classifier will actually return a real valued function that we'll have to interpret).

- Training examples: A set of examples of the form {(x, f (x))}

Protocol: Supervised learning

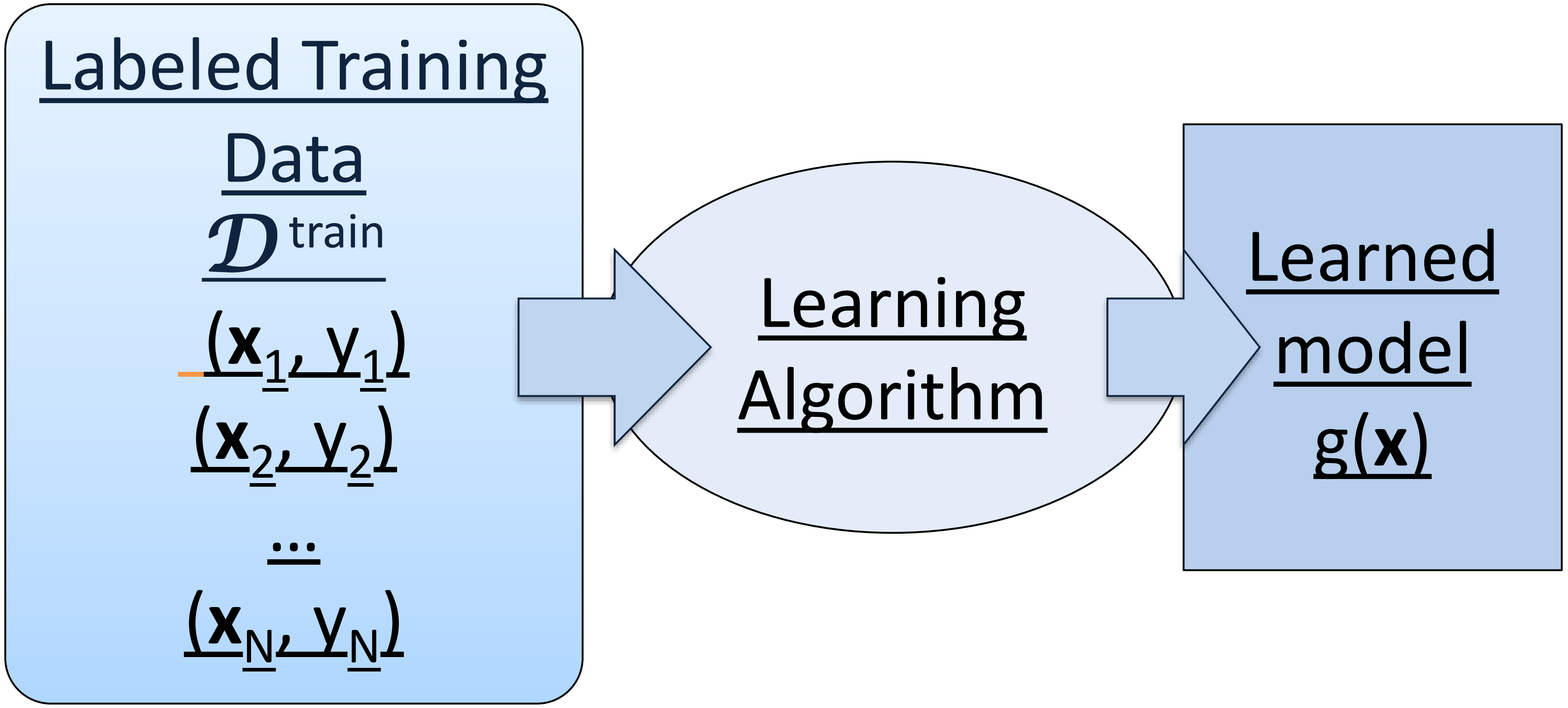


Input
Target function
$\mathbf{y} = f(\mathbf{x})$
Output
$\mathbf{x} \in \mathcal{X}$
Learned Model
$\mathbf{y} = g(\mathbf{x})$
$y \in \mathcal{Y}$
An item **x** drawn from an instance space $\mathcal{X}$
An item **y** drawn from a label space $\mathcal{Y}$

The i.i.d. assumption

- Training and test items are independently and identically distributed (i.i.d.):
  - There is a distribution $P(\mathbf{X}, Y)$ from which the data $\mathcal{D} = \{(\mathbf{x}, y)\}$ is generated.
    - Sometimes it's useful to rewrite $P(\mathbf{X}, Y)$ as $P(\mathbf{X})P(Y|\mathbf{X})$
      Usually $P(\mathbf{X}, Y)$ is unknown to us (we just know it exists)
  - Training and test data are samples drawn from the *same* $P(\mathbf{X}, Y)$: they are identically distributed
  - Each $(\mathbf{x}, y)$ is drawn independently from $P(\mathbf{X}, Y)$

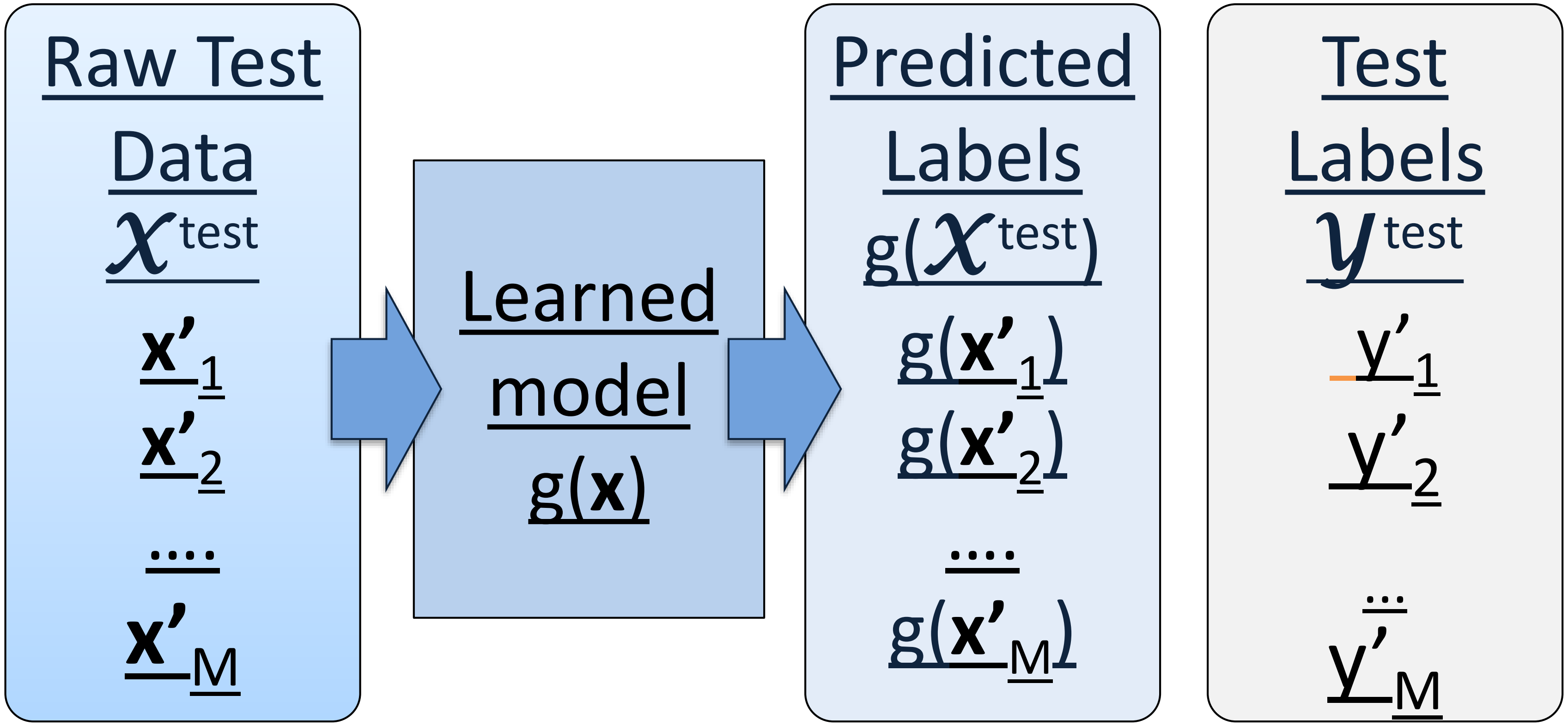
Supervised learning: Training



- Give the learner examples in $\mathcal{D}^{\text{train}}$
- The learner returns a model $g(\mathbf{x})$

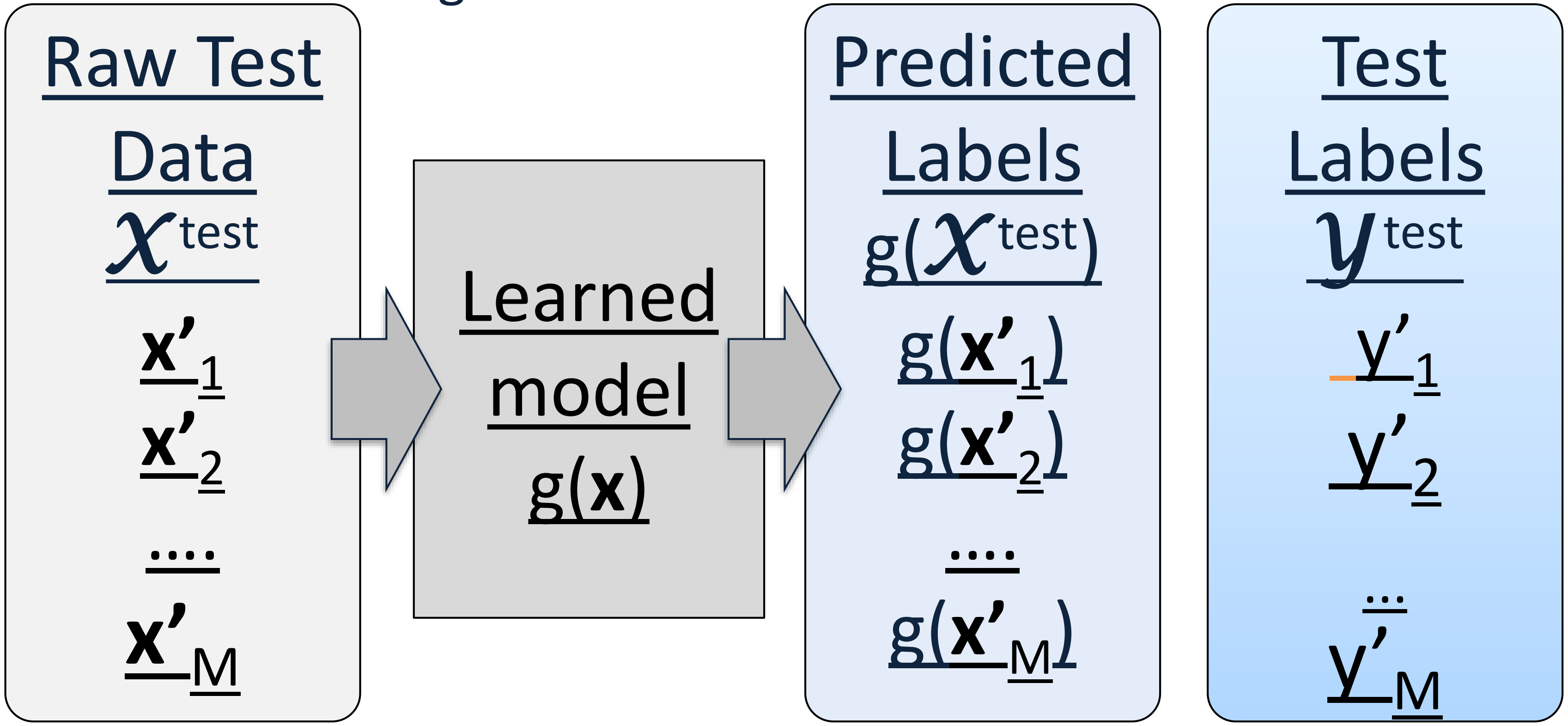
Supervised learning: Testing

- Apply the model to the raw test data



Supervised learning: Testing

- Evaluate the model by comparing the predicted labels against the test labels



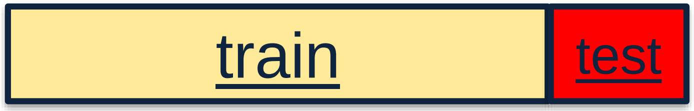
Raw Test Data $\mathcal{X}^{\text{test}}$
$\mathbf{x}'_1$
$\mathbf{x}'_2$
….
$\mathbf{x}'_M$
Learned model $g(\mathbf{x})$
Predicted Labels $g(\mathcal{X}^{\text{test}})$
$g(\mathbf{x}'_1)$
$g(\mathbf{x}'_2)$
….
$g(\mathbf{x}'_M)$
Test Labels $\mathcal{Y}^{\text{test}}$
y'_1
y'_2
…
y'_M

Experimental Machine Learning

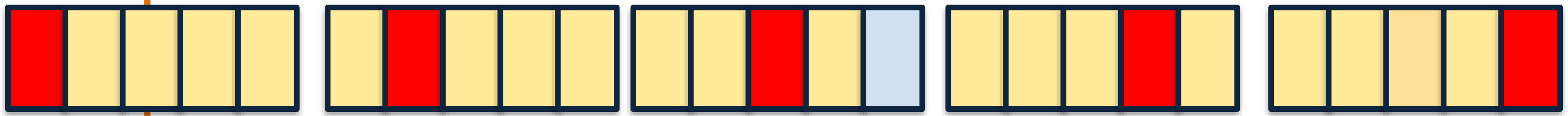
- Machine Learning is an Experimental Field and we will spend some time (in Problem sets) learning how to run experiments and evaluate results
  - First hint: be organized; write scripts
- Basics:
  - Split your data into two (or three) sets:
    - Training data (often 70-90%)
    - Test data (often 10-20%)
    - Development data (10-20%)
- You need to report performance on test data, but you are not allowed to look at it.
  - You are allowed to look at the development data (and use it to tweak parameters)

N-fold cross validation

- Instead of a single test-training split:



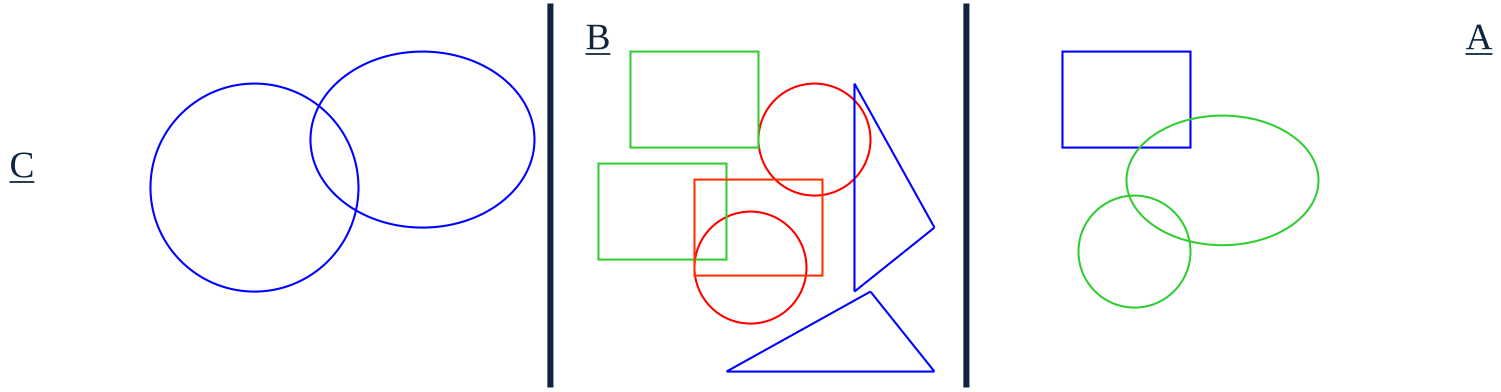
- Split data into N equal-sized parts



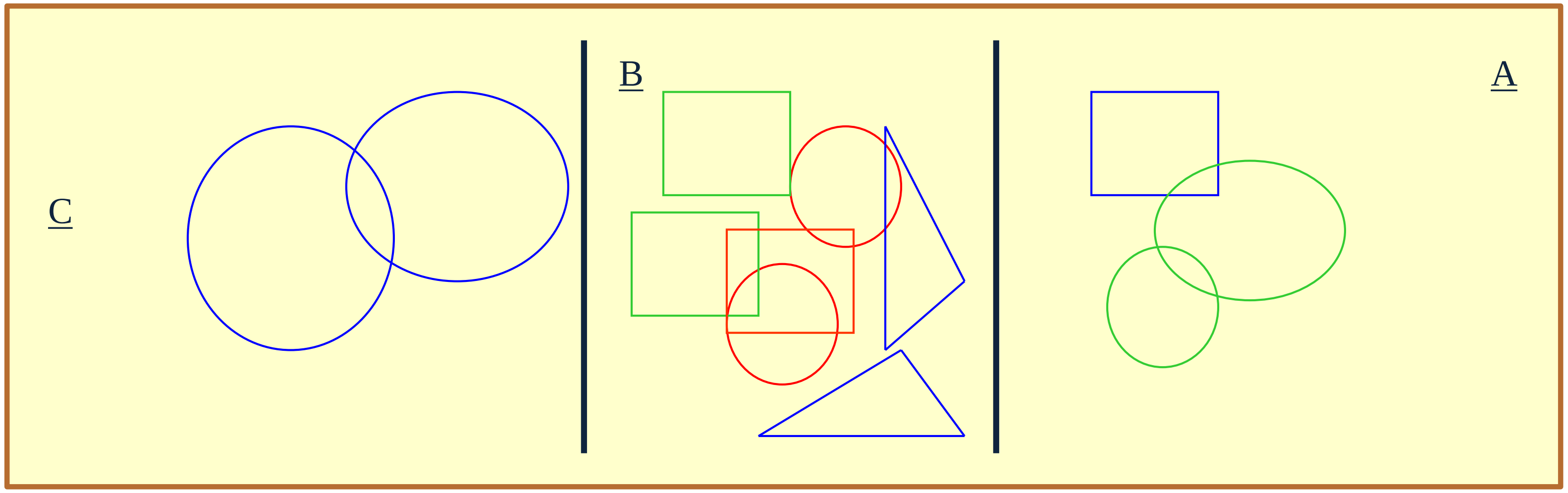
- Train and test N different classifiers
- Report average accuracy and standard deviation of the accuracy

Decision Trees

- A hierarchical data structure that represents data by implementing a divide and conquer strategy
- Can be used as a non-parametric classification and regression method
- Given a collection of examples, **learn a decision tree that represents it.**
- Use this representation to **classify new examples**

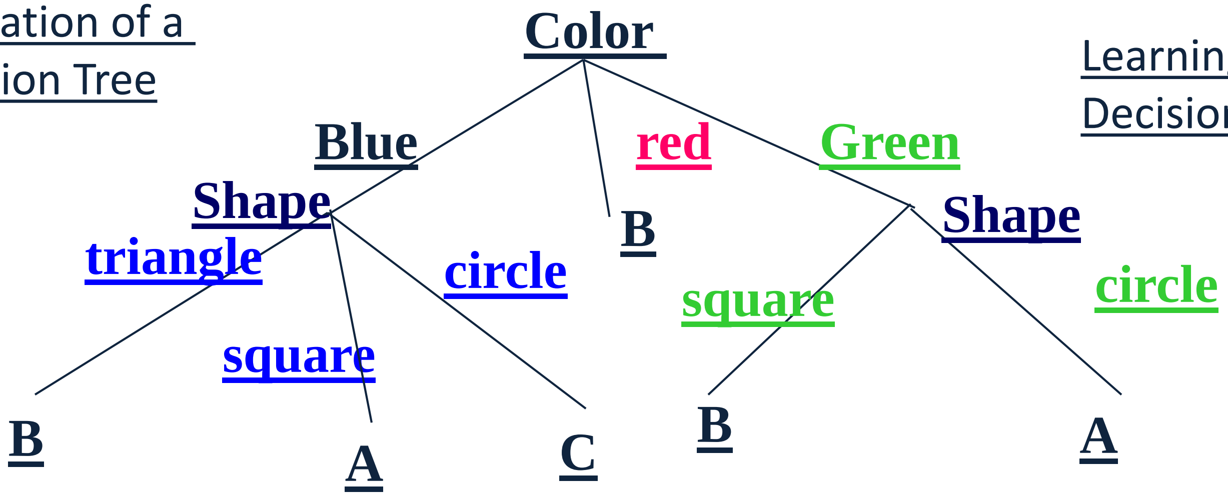


The Representation



Evaluation of a Decision Tree

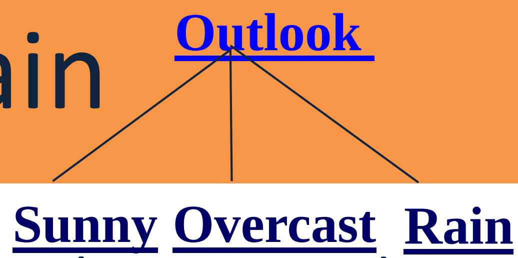
Learning a Decision Tree



High Entropy – High level of Uncertainty

Low Entropy – No Uncertainty.

Information Gain



- The information gain of an attribute a is the expected reduction in entropy caused by partitioning on this attribute

$$\mathrm{Gain}(S,a) = \mathrm{Entropy}(S) - \sum_{v \in \mathrm{values}(a)} \frac{|S_v|}{|S|} \mathrm{Entropy}(S_v)$$

- where S_v is the subset of S for which attribute a has value v, and the entropy of partitioning the data is calculated by weighing the entropy of each partition by its size relative to the original set
  - Partitions of low entropy (imbalanced splits) lead to high gain
- Go back to check which of the A, B splits is better

On-Line Learning

- Not the most general setting for on-line learning.
- Note: online learning protocol v.s. online learning algorithm

On Line Model

- Model:protocol
  - Instance space: X (dimensionality – n)
  - Target: f: X $\to$ {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.

$$\boxed{M_A(C) = \max_{f \in C, S} M_A(f, S)}$$

Quantifying Performance

- We want to be able to say something rigorous about the performance of our learning algorithm.

- Evaluating a learning algorithm:
  - Experiments
  - COLT
    - E.g, PAC theory, VC theory, Mistake bound

Mistake Driven Learning Algorithm

- learn a linear function over the feature space
  - Perceptron (+ many variations)
  - Winnow
  - General Gradient Descent view

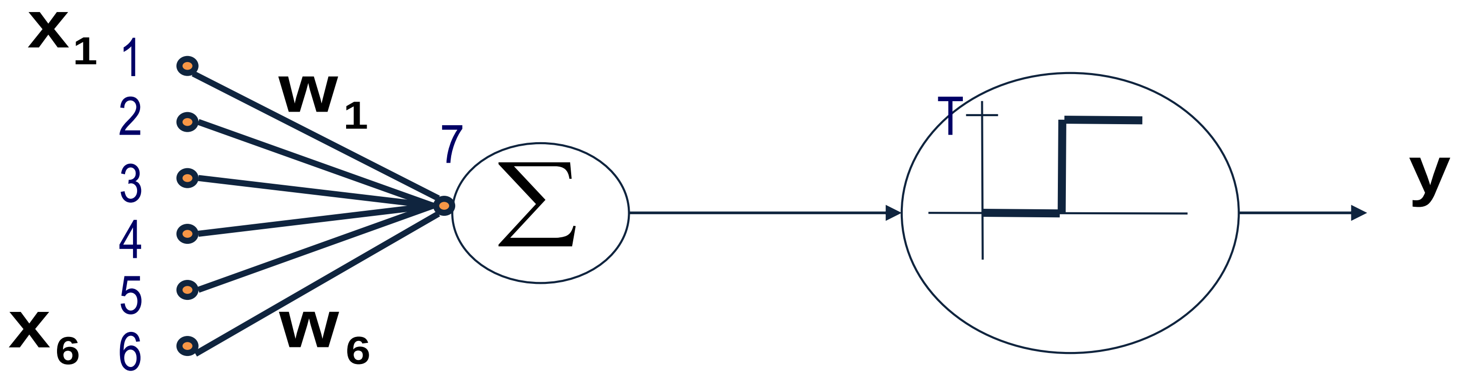
- Issues:
  - Importance of Representation
  - Complexity of Learning
  - Idea of Kernel Based Methods
  - More about features

The Halving Algorithm

- Let C be a concept class. Learn f ϵ C
- Halving:
- In the ith stage of the algorithm:
  - C_i all concepts in C consistent with all i-1 previously seen examples
- Given an example e_i consider the value $f_j(e_i)$for all $f_j \in C_i$ and predict by majority.
- Predict 1 if $|\{f_j \in C_i; f_j(e_i)=0\}| < |\{f_j \in C_i; f_j(e_i)=1\}|$
- Clearly $C_{i+1} \subseteq C_i$ and if a mistake is made in the ith example, then $|C_{i+1}| < \frac{1}{2}|C_i|$
- The Halving algorithm makes at most log(|C|) mistakes

Perceptron learning rule

- On-line, mistake driven algorithm.
- Rosenblatt (1959) suggested that when a target output value is provided for a single neuron with fixed input, it can incrementally change weights and learn to produce the output using the Perceptron learning rule
- (Perceptron == Linear Threshold Unit)



Perceptron learning rule

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

```
1. Initialize $w = 0 \in \mathbf{R}^n$
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:
        $w = w + r\, y\, x$     (r - a constant, learning rate)
        Otherwise, if $y' = y$, leave weights unchanged.
```

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* $x_i \in \Re^N$, $||x_i|| \leq R$ *and* $y_i \in \{-1,1\}$ *for all i. Let* $u \in \Re^N$, $\gamma > 0$ *be such that,* $||\mathbf{u}|| = 1$ *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

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 – Mistake Bound

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

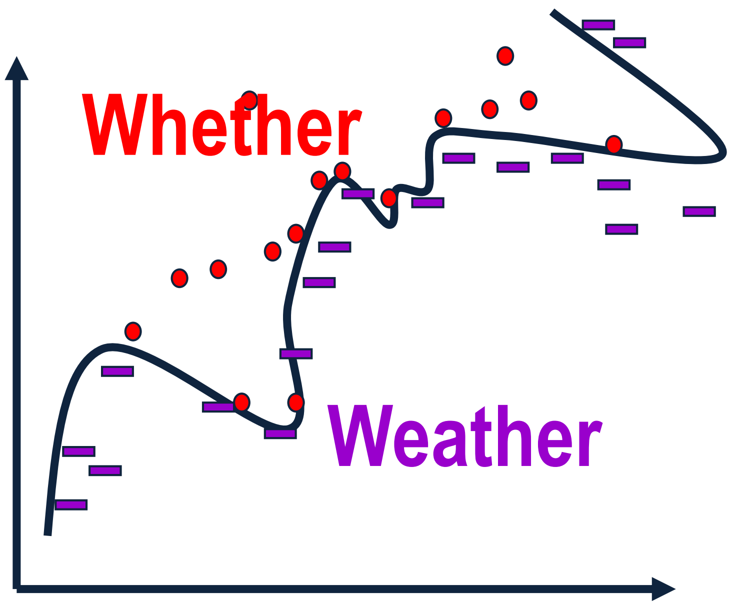
```
Initialize : $\theta = n$;  $w_i = 1$
Prediction is 1 iff $w \bullet x \geq \theta$
If no mistake : do nothing
If $f(x) = 1$ but $w \bullet x < \theta$ ,  $w_i \leftarrow 2w_i$  (if $x_i = 1$) (promotion)
If $f(x) = 0$ but $w \bullet x \geq \theta$ ,  $w_i \leftarrow w_i/2$  (if $x_i = 1$) (demotion)
```

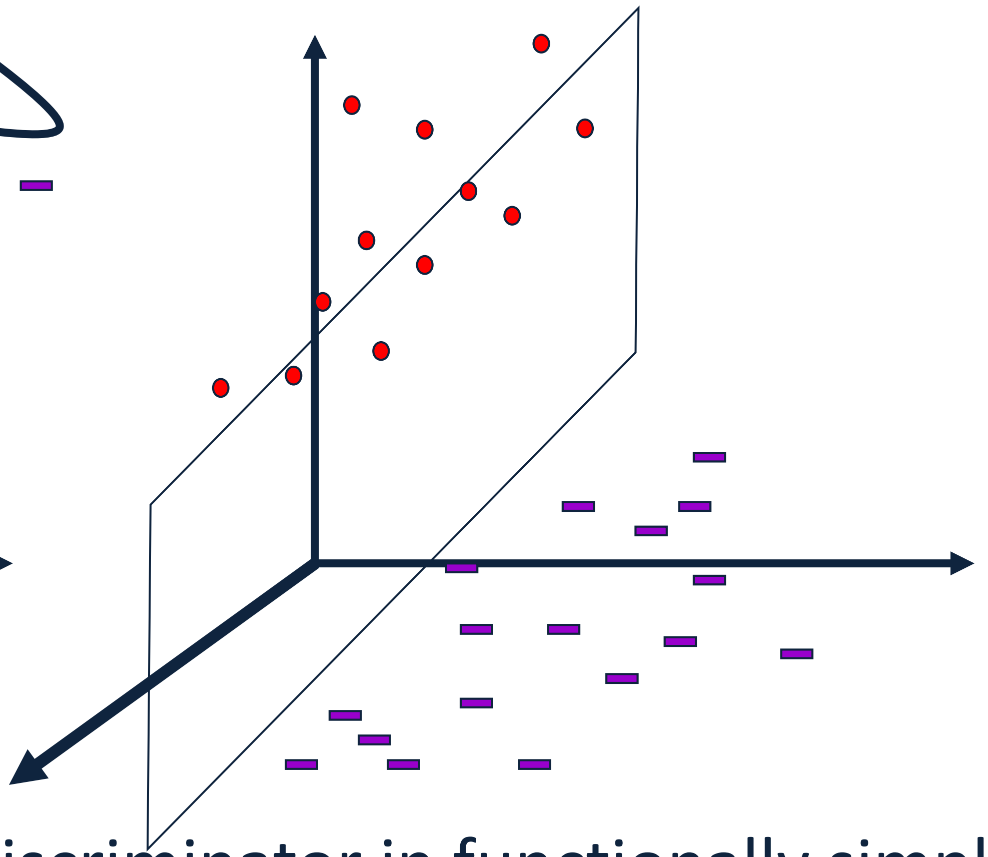
- u - # of mistakes on positive examples (promotions)
- v - # of mistakes on negative examples (demotions)

Winnow

# of mistakes: $u + v < 3u + 2 = O(k \log n)$

Transforming Feature Spaces





New discriminator in functionally simpler

$x_1 x_2 \overline{x}_3 \vee \overline{x}_1 x_4 \overline{x}_3 \vee x_3 \overline{x}_2 x_5$

$y_1 \vee y_4 \vee y_5$

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 expected risk function

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

- 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 $Q(z, w)$

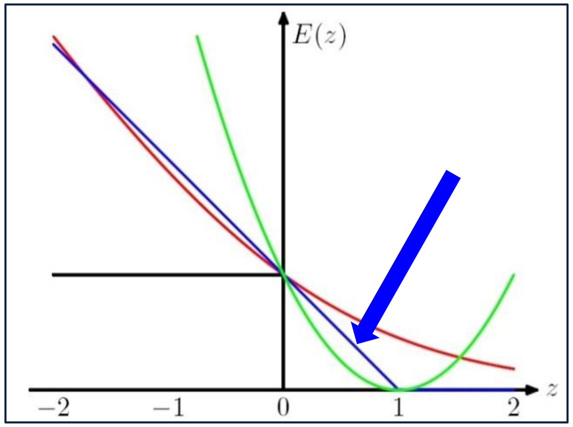
Stochastic Gradient Algorithms

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

- 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 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

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

(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 $G_{t,j} = \sum_{k=1, t} g^2_{k,j}$ 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 the problem call the LASSO problem

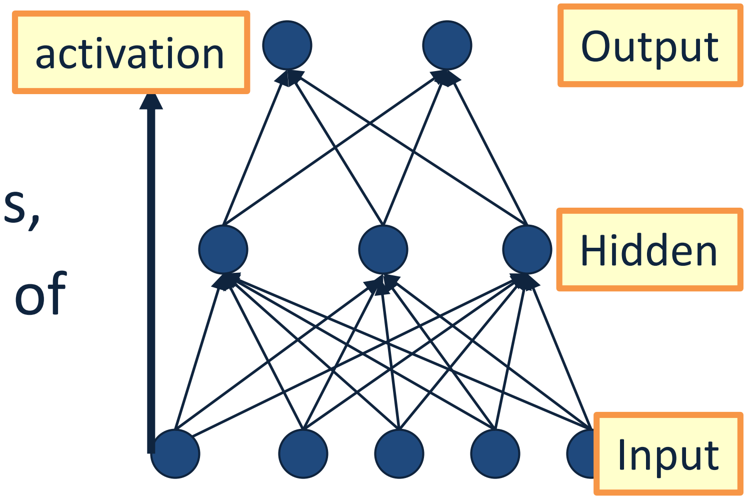
- Hinge Loss case: $Q((x, y), w) = \max(0, 1 - y\, w \cdot x)$
  - $R(w) = ||w||^2_2$ gives the problem called Support Vector Machines

- 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".

Multi-Layer Neural Networks

- Multi-layer network 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 network was [thought to be] difficult.

Model Neuron (Logistic)

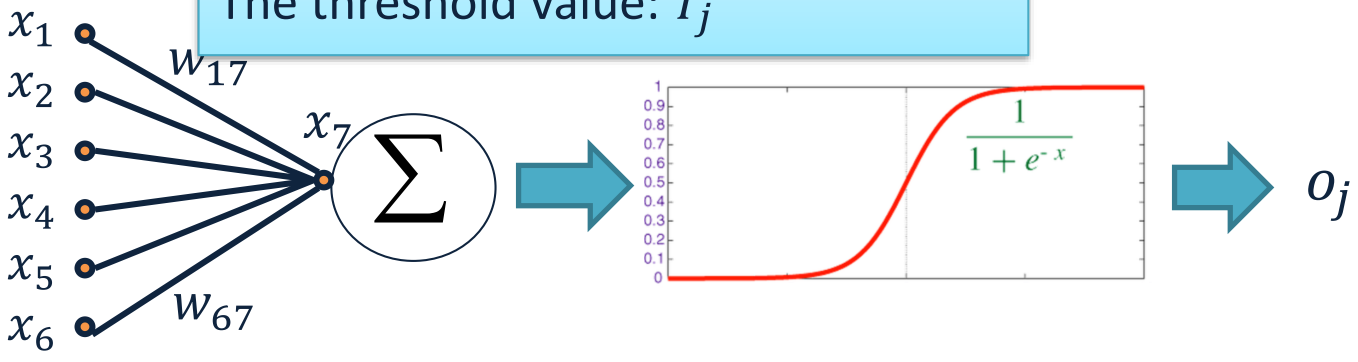
Neuron Definition

- Neuron is modeled by a unit j connected by weighted

The parameters so far?

The set of connective weights: w_{ij}

The threshold value: T_j

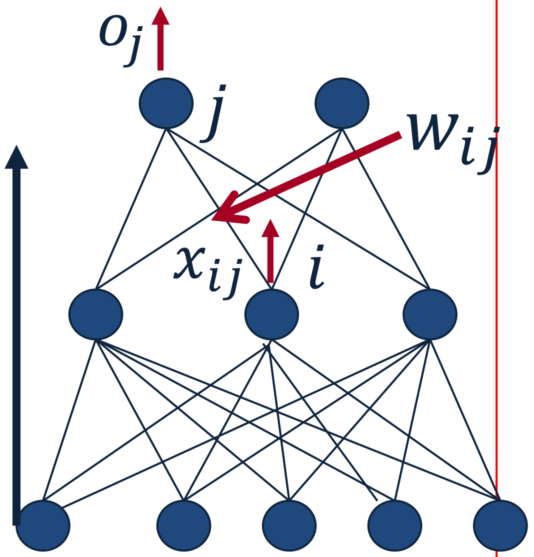


- Use a non-linear, differentiable output function such as the sigmoid or logistic function
- Net input to a unit is defined as: $\text{net}_j = \sum w_{ij} . x_i$
- Output of a unit is defined as: $o_j = \dfrac{1}{1 + \exp\left(-(\text{net}_j - T_j)\right)}$

Derivation of Learning Rule (3)

- Weights of output units:
  - w_{ij} is changed by:

$$\Delta w_{ij} = R(t_j - o_j)o_j(1 - o_j)x_{ij}$$
$$= R\delta_j x_{ij}$$



where

$$\delta_j = (t_j - o_j)o_j(1 - o_j)$$

The Backpropagation Algorithm

- Create a fully connected three layer network. Initialize weights.
- Until all examples produce the correct output within ϵ (or other criteria)

```
For each example in the training set do:
    1.  Compute the network output for this example
    2.  Compute the error between the output and target value
            $\delta_k = (t_k - o_k) o_k (1 - o_k)$
    1.  For each output unit k, compute error term
            $\delta_j = o_j(1 - o_j). \sum_{k \in downstream(j)} -\delta_k \; w_{jk}$
    1.  For each hidden unit, compute error term:
            $\Delta w_{ij} = R \delta_j x_{ij}$
    1.  Update network weights
End epoch
```

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

Computational Issues

- Assume the data is linearly separable.
- Sample complexity:
  - Suppose we want to ensure that our LTU has an error rate (on new examples) of less than ε with high probability (at least $(1-\delta)$)
  - How large does m (the number of examples) must be in order to achieve this ? It can be shown that for n dimensional problems

$$m = O(1/\varepsilon\ [\ln(1/\delta) + (n+1)\ln(1/\varepsilon)\ ].$$

- Computational complexity: What can be said?
  - It can be shown that there exists a polynomial time algorithm for finding consistent LTU (by reduction from linear programming).
  - [Contrast with the NP hardness for 0-1 loss optimization]
  - (On-line algorithms have inverse quadratic dependence on the margin)

PAC Learnability

Definition

- 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 distribution D over X, and fixed $0< \varepsilon, \delta < 1$,
- 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, (ErrorD = PrD[f(x) : = h(x)])
- where m is polynomial in $1/ \varepsilon$, $1/ \delta$, n and size(H)
- C is efficiently learnable if L can produce the hypothesis in time polynomial in $1/ \varepsilon$, $1/ \delta$, n and size(H)

Occam's Razor (1)

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

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

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

What do we know now about the Consistent Learner scheme?

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

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

We showed that a m-consistent hypothesis generalizes well (err< ϵ) (Appropriate m is a function of |H|, ϵ, δ)

(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

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

  - Show that the consistent hypoth $\in$ H can be computed efficiently

We need to show that m is polynomial in n when |H| is a function of n. That is, showing ln|H| is polynomial in n

- 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), than h is close to the target function.

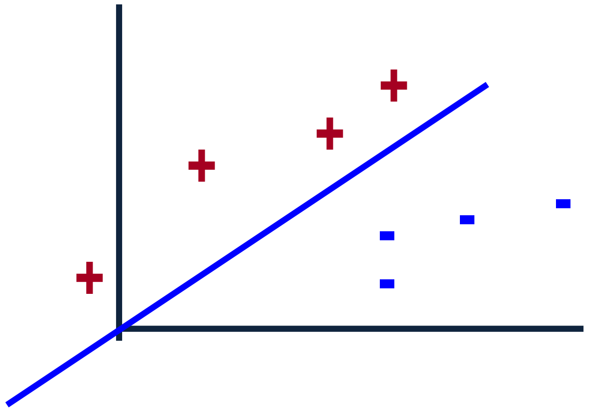
Infinite Hypothesis Space

- 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)

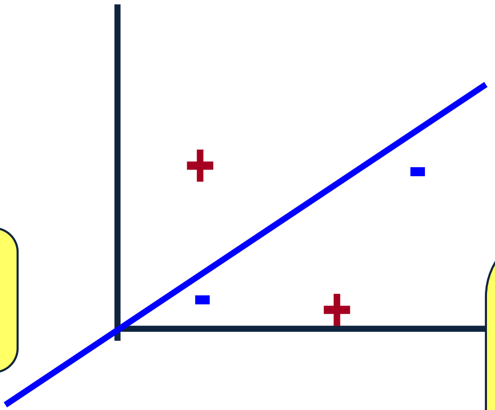
Shattering

- 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

<u>Half-spaces in the plane:</u>



All sets of three?



1. If the 4 points form a convex polygon… (if not?)
2. If one point is inside the convex hull defined by the other three… (if not?)

sets of one, two or three points can be shattered
but there is no set of four points that can be shattered

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) >=d
- If no subset of size d can be shattered, then VC(H) < d

<u>VC(Half intervals) = 1</u> (no subset of size 2 can be shattered)
<u>VC(Intervals) = 2</u> (no subset of size 3 can be shattered)
<u>VC(Half-spaces in the plane) = 3</u> (no subset of size 4 can be shattered)

Some are shattered, but some are not

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
- $$\boxed{m > \frac{1}{\varepsilon}\{8VC(H)\log\frac{13}{\varepsilon} + 4\log(\frac{2}{\delta})\}}$$
- Then with probability at least $(1-\delta)$, h has error less than ε.

What if H is finite?

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

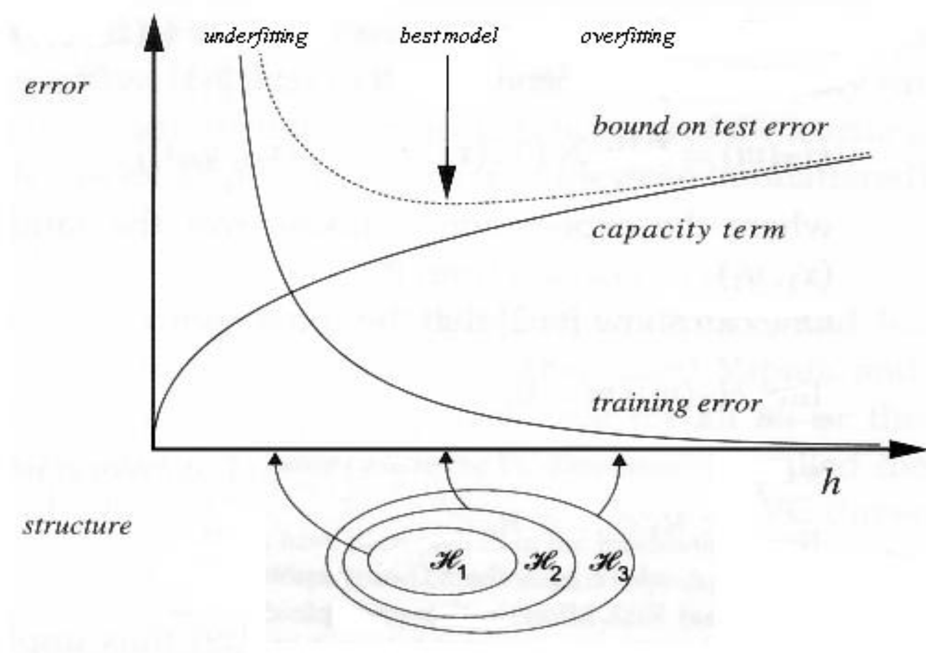
- Notice that to shatter m examples it must be that: $|H|>2^m$, so $\log(|H|) \geq VC(H)$

COLT approach to explaining Learning

- No Distributional Assumption
- Training Distribution is the same as the Test Distribution

- Generalization bounds depend on this view and affects model selection.

$$Err_D(h) < Err_{TR}(h) + P(VC(H), \log(1/\delta), 1/m)$$



- This is also called the "Structural Risk Minimization" principle.

Theoretical Motivation of Boosting

- "Strong" PAC algorithm:
  - for any distribution
  - $\forall\ \epsilon, \delta > 0$
  - Given polynomially many random examples
  - Finds hypothesis with error $\leq \epsilon$ with probability $\geq$ (1-δ)

- "Weak" PAC algorithm
  - Same, but only for $\epsilon \leq$ ½ - γ

- [Kearns & Valiant '88]:
  - Does weak learnability imply strong learnability?
  - Anecdote: the importance of the distribution free assumption
    - It does not hold if PAC is restricted to only the uniform distribution, say

A Formal View of Boosting

- Given training set $(x_1, y_1), \ldots (x_m, y_m)$
- $y_i \in \{-1, +1\}$ is the correct label of instance $x_i \in X$
- For $t = 1, \ldots, T$
  - Construct a distribution D_t on $\{1,\ldots m\}$
  - Find weak hypothesis ("rule of thumb")

    $$h_t : X \to \{-1, +1\}$$

    with small error ϵ_t on D_t:

    $$\epsilon_t = \Pr_D [h_t (x_i) \neg= y_i]$$
- Output: final hypothesis H_{final}

Adaboost

Think about unwrapping it all the way to 1/m

- Constructing D_t on {1,…m}:
  - $D_1(i) = 1/m$
  - Given D_t and h_t :
  - $D_{t+1} = D_t(i)/z_t \times e^{-\alpha_t}$ if $y_i = h_t(xi)$

    $D_t(i)/z_t \times e^{+\alpha_t}$ if $y_i \neg= h_t(xi)$

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

  where z_t = normalization constant

  and

  $\alpha_t = ½ \ln\{ (1- \epsilon_t)/\epsilon_t \}$

< 1; smaller weight

> 1; larger weight

Notes about α_t:

- Positive due to the weak learning assumption
- Examples that we predicted correctly are demoted, others promoted
- Sensible weighting scheme: better hypothesis (smaller error) → larger weight

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

Margin of a Separating Hyperplane

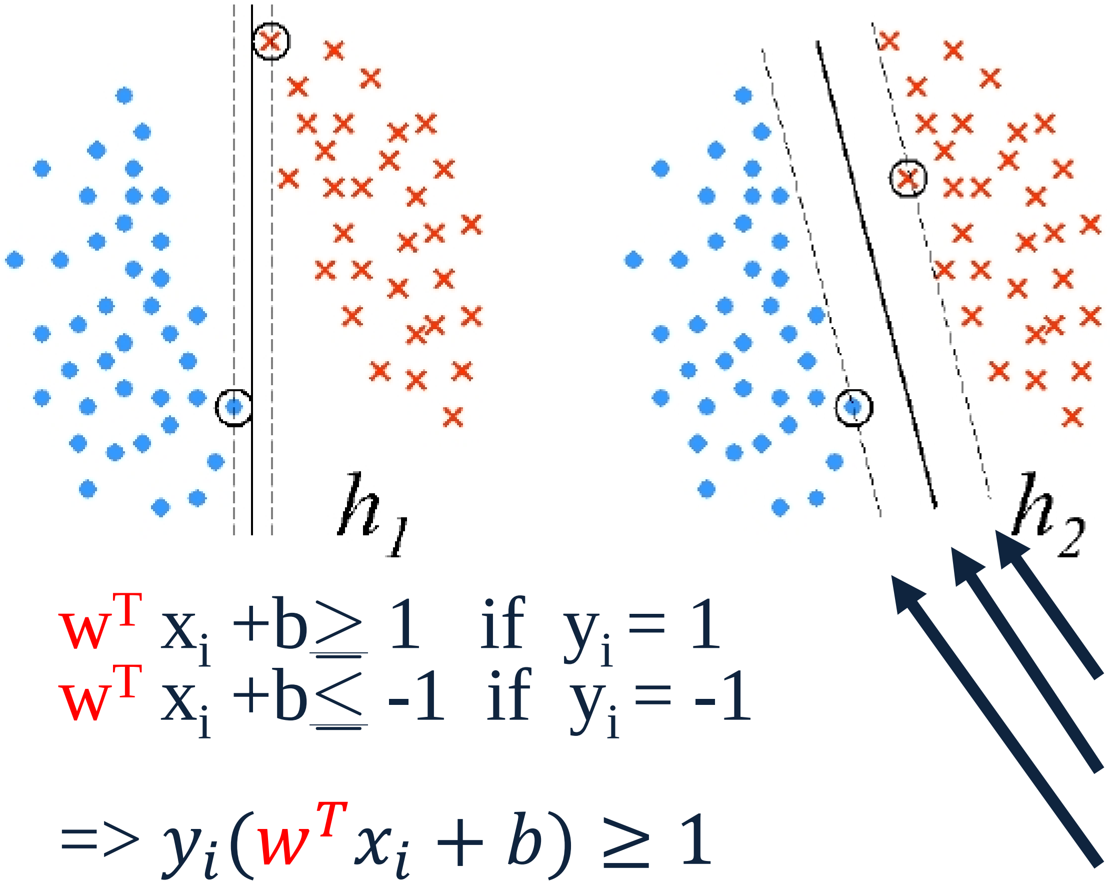
- A separating hyperplane: $w^T x + b = 0$

Assumption: data is linear separable

Distance between $w^T x + b = +1$ and -1 is $2 / \|w\|$

Idea:

1. Consider all possible w with different angles
2. Scale w such that the constraints are tight
3. Pick the one with largest margin



$$w^T x_i + b \geq 1 \quad \text{if} \quad y_i = 1$$
$$w^T x_i + b \leq -1 \quad \text{if} \quad y_i = -1$$

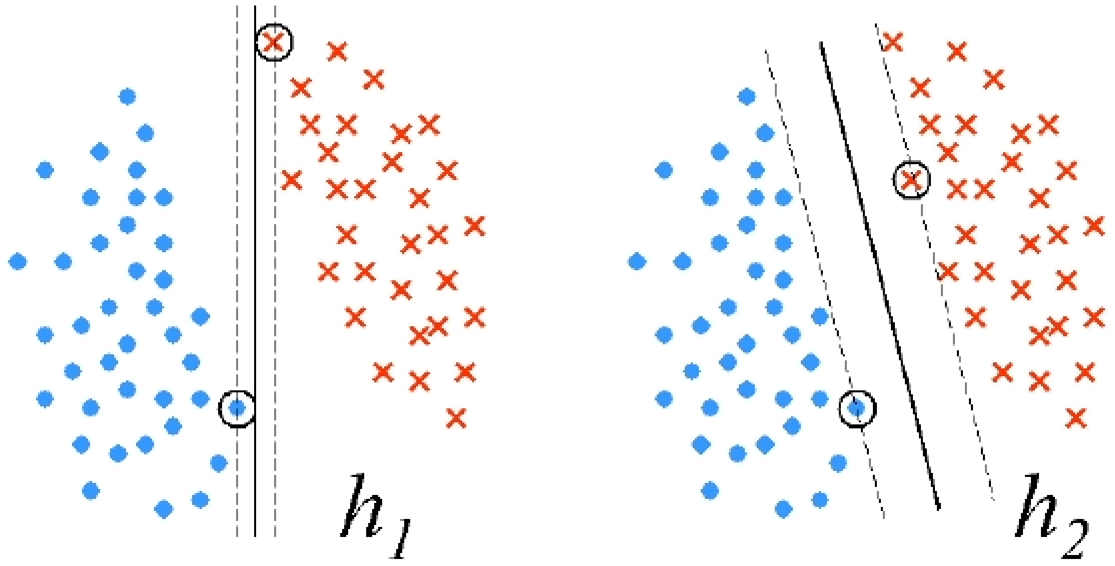
$$=> y_i(w^T x_i + b) \geq 1$$

$w^T x+b = 1$

$w^T x+b = 0$

$w^T x+b = -1$

Maximal Margin



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

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

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

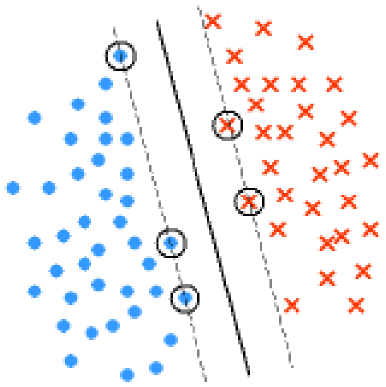
$$\text{s.t} \quad y_i(w^T x_i + b) \geq 1, \forall (x_i, y_i) \in S$$

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 $I = \{i: y_i (w^{*T} x_i + b) = 1\}$.
  Then there exists coefficients $\alpha_i > 0$ such that:

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



(recap) Kernel Perceptron

$\textbf{Examples}: \mathbf{x} \in \{0,1\}^{n};\quad \textbf{Nonlinear mapping}: \mathbf{x} \to \mathbf{t(x)}, \mathbf{t(x)} \in \mathbf{R}^{n'}$

$\textbf{Hypothesis}: \mathbf{w} \in \mathbf{R}^{n'}; \textbf{Decision function}: \mathbf{f(x)} = \mathbf{sgn}(\sum_{i=1}^{n'} \mathbf{w}_i \mathbf{t(x)}_i) = \mathbf{sgn}(\mathbf{w} \bullet \mathbf{t(x)})$

$$\boxed{\textbf{If}\ \ \mathbf{f(x^{(k)})} \neq \mathbf{y^{(k)}},\quad \mathbf{w} \leftarrow \mathbf{w} + \mathbf{r}\, \mathbf{y^{(k)}} t(\mathbf{x^{(k)}})}$$

- If n’ is large, we cannot represent **w** explicitly. However, the weight vector **w** can be written as a linear combination of examples:

$$\boxed{\mathbf{w} = \sum_{j=1}^{m} \mathbf{r} \alpha_j \mathbf{y^{(j)}} \mathbf{t(x^{(j)})}}$$

- Where α_j is the number of mistakes made on $x^{(j)}$
- Then we can compute f(x) based on $\{x^{(j)}\}$ and $\boldsymbol{\alpha}$

$$\boxed{\mathbf{f(x)} = \mathbf{sgn}(\mathbf{w} \bullet \mathbf{t(x)}) = \mathbf{sgn}(\sum_{j=1}^{m} \mathbf{r} \alpha_j \mathbf{y^{(j)}} \mathbf{t(x^{(j)})} \bullet \mathbf{t(x)}) = \mathbf{sgn}(\sum_{j=1}^{m} \mathbf{r} \alpha_j \mathbf{y^{(j)}} K(\mathbf{x^{(j)}}, \mathbf{x}))}$$

(recap) Kernel Perceptron

$$\textbf{Examples}: x \in \{0,1\}^n; \quad \textbf{Nonlinear mapping}: x \to t(x),\ t(x) \in R^{n'}$$

$$\textbf{Hypothesis}: w \in R^{n'}; \textbf{Decision function}: f(x) = \text{sgn}(w \bullet t(x))$$

- In the training phase, we initialize $\boldsymbol{\alpha}$ to be an all-zeros vector.
- For training sample $(x^{(k)}, y^{(k)})$, instead of using the original Perceptron update rule in the $R^{n'}$ space

$$\boxed{\text{If } f(x^{(k)}) \neq y^{(k)}, \quad w \leftarrow w + r\, y^{(k)} t(x^{(k)})}$$

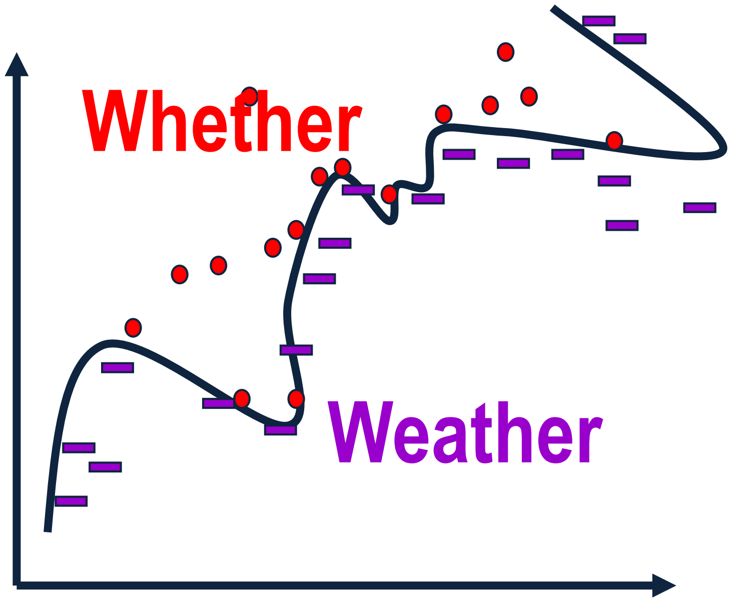
we maintain $\boldsymbol{\alpha}$ by

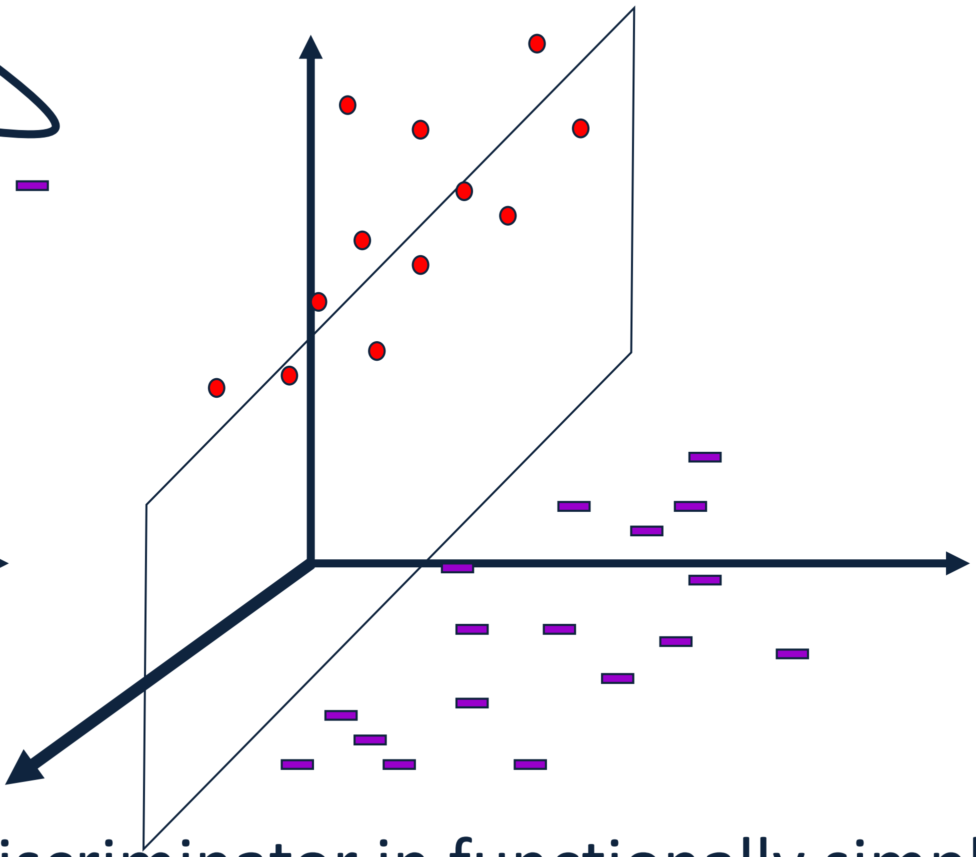
$$\boxed{\text{if } f(x^{(k)}) = \text{sgn}\left(\sum_{j=1}^{m} r\alpha_j y^{(j)} K(x^{(j)}, x^{(k)})\right) \neq y^{(k)} \quad \text{then } \alpha_k \leftarrow \alpha_k + 1}$$

based on the relationship between w and $\boldsymbol{\alpha}$:

$$\boxed{w = \sum_{j=1}^{m} r\alpha_j y^{(j)} t(x^{(j)})}$$

Embedding



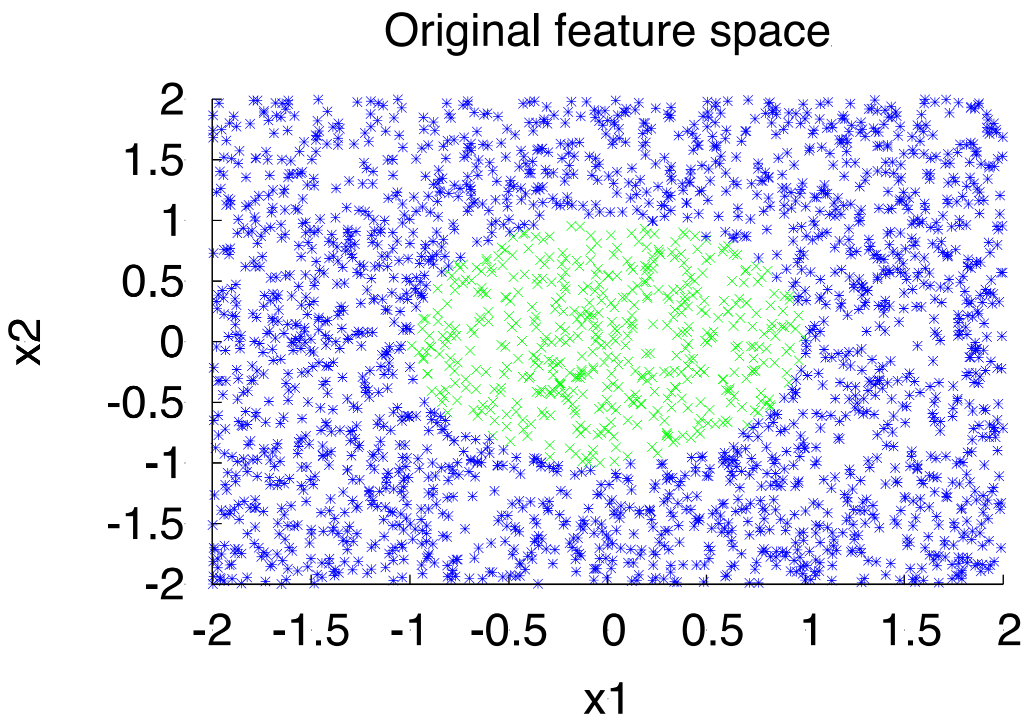


New discriminator in functionally simpler

$\mathbf{x_1 x_2 \overline{x}_3 \vee \overline{x}_1 x_4 \overline{x}_3 \vee x_3 \overline{x}_2 x_5}$

$\mathbf{y_1 \vee y_4 \vee y_5}$

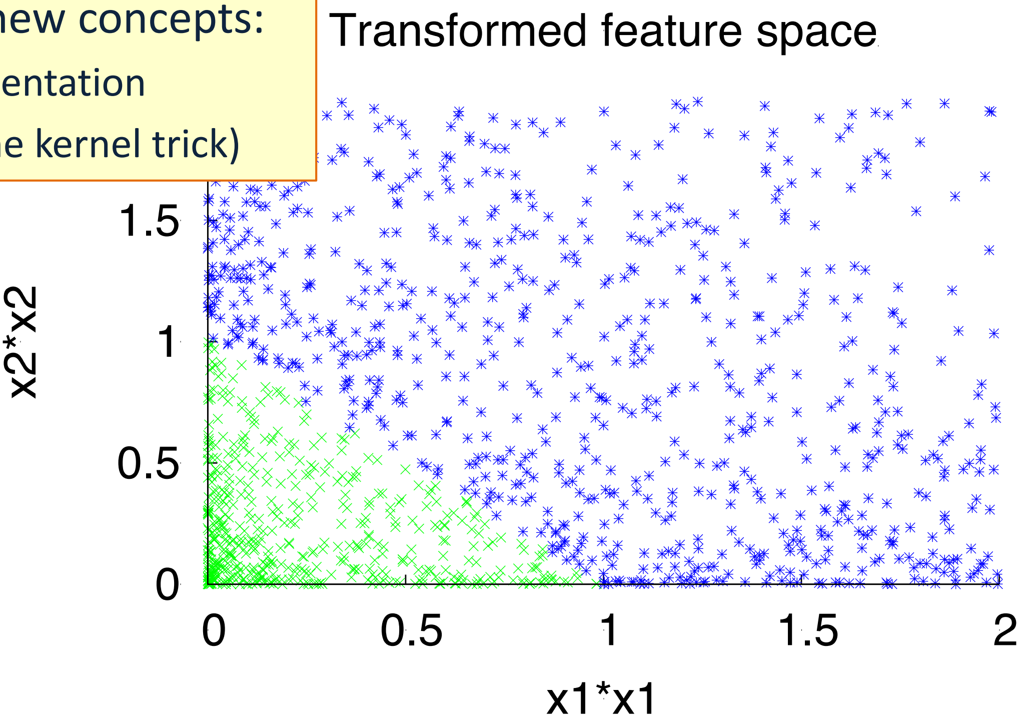
Making data linearly separable



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

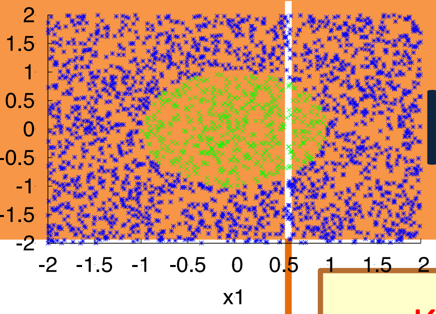
Making data linearly separable

- In order to deal with this, we introduce two new concepts:
  - Dual Representation
  - Kernel (& the kernel trick)



Transform data: $\mathbf{x} = (x_1, x_2) \Rightarrow \mathbf{x'} = (x_1^2, x_2^2)$

$f(\mathbf{x'}) = 1$ iff $x'_1 + x'_2 \leq 1$



Kernels – General Conditions

Kernel: Example

- Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. 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(.,.)?
  - A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.
- Take the quadratic kernel: $k(x,z) = (x^T z)^2$
- Example: Direct construction (2 dimensional, for simplicity):
- $\mathbf{K(x,z) = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2}$
- $\mathbf{= (x_1^2, \text{sqrt}\{2\}\, x_1x_2, x_2^2)\ (z_1^2, \text{sqrt}\{2\}\, z_1z_2, z_2^2)}$
- $\mathbf{= \Phi(x)^T \Phi(z) \rightarrow}$ **A dot product in an expanded space.**
- It is not necessary to explicitly show the feature function ϕ.
- General condition: construct the Gram matrix $\{k(x_i, z_j)\}$; check that it's positive semi definite.

Good Luck ☺ !!