Administration

Questions

- Registration ☺
- Hw1 is due tomorrow night
- Hw2 will be out tomorrow night.
  - Please start working on it as soon as possible
  - Come to sections with questions
- No lectures next Week!!
  - Please watch the corresponding videos: check the schedule page across from the corresponding dates.
  - I will not have office hours this week.
  - Please go to the TAs office hours and discussion session.
- Extensions: you don't need to email me about extensions to the Hw. You have it – 96 hours of it.

Projects

- Projects proposals are due on Friday 3/10/17
- We will give you an approval to continue with your project, possibly, along with comments and/or a request to modify/augment/do a different project. There may also be a mechanism for peer comments.

- We encourage team projects – a team can be up to 3 people.

- Please start thinking and working on the project now.
- Your proposal is limited to 1-2 pages, but needs to include references and, ideally, some of the ideas you have developed in the direction of the project (maybe even some preliminary results).
- Any project that has a significant Machine Learning component is good.
- You can do experimental work, theoretical work, a combination of both or a critical survey of results in some specialized topic.
- The work *has* to include some reading. Even if you do not do a survey, you must read (at least) two related papers or book chapters and relate your work to it.
- Originality is not mandatory but is encouraged.
- Try to make it interesting!

Examples

- KDD Cup 2013:
  - "Author-Paper Identification": given an author and a small set of papers, we are asked to identify which papers are really written by the author.
    - https://www.kaggle.com/c/kdd-cup-2013-author-paper-identification-challenge
  - "Author Profiling": given a set of document, profile the author: identification, gender, native language, ...
- Caption Control: Is it gibberish? Spam? High quality text?
  - Adapt an NLP program to a new domain
- Work on making learned hypothesis (e.g., linear threshold functions, NN) more comprehensible
  - Explain the prediction
- Develop a (multi-modal) People Identifier
- Compare Regularization methods: e.g., Winnow vs. L1 Regularization
- Large scale clustering of documents + name the cluster
- Deep Networks: convert a state of the art NLP program to a deep network, efficient, architecture.
- Try to prove something

A Guide

Today:
Take a more general perspective and think more about learning, learning protocols, quantifying performance, etc.
This will motivate some of the ideas we will see next.

- Learning Algorithms
  - Search: (Stochastic) Gradient Descent with LMS
  - Decision Trees & Rules
- Importance of hypothesis space (representation)
- How are we doing?
  - Simplest: Quantification in terms of cumulative # of mistakes
  - More later
- Perceptron
  - How to deal better with large features spaces & sparsity?
    - Winnow
  - Variations of Perceptron
    - Dealing with overfitting
  - Closing the loop: Back to Gradient Descent
  - Dual Representations & Kernels
- Multilayer Perceptron
- Beyond Binary Classification?
  - Multi-class classification and Structured Prediction
- More general way to quantify learning performance (PAC)
  - New Algorithms (SVM, Boosting)

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.

Learning Conjunctions

- There is a hidden (monotone) conjunction the learner (you) is to learn

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- How many examples are needed to learn it ? How ?
  - Protocol I: The learner proposes instances as queries to the teacher
  - Protocol II: The teacher (who knows f) provides training examples
  - Protocol III: Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels (f(x))

Learning Conjunctions

- Protocol I: The learner proposes instances as queries to the teacher
- Since we know we are after a monotone conjunction:
- Is x_{100} in? <(1,1,1…,1,0), ?> f(x)=0 (conclusion: Yes)
- Is x_{99} in? <(1,1,…1,0,1), ?> f(x)=1 (conclusion: No)
- Is x_1 in ? <(0,1,…1,1,1), ?> f(x)=1 (conclusion: No)

- A straight forward algorithm requires n=100 queries, and will produce as a result the hidden conjunction (exactly).

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

What happens here if the conjunction is not known to be monotone?
If we know of a positive example, the same algorithm works.

Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples

Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples
- <(0,1,1,1,1,0,…,0,1), 1>

Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples
- <(0,1,1,1,1,0,…,0,1), 1> (We learned a superset of the good variables)

Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples
- <(0,1,1,1,1,0,…,0,1), 1> (We learned a superset of the good variables)
- To show you that all these variables are required…

Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples
- <(0,1,1,1,1,0,…,0,1), 1> (We learned a superset of the good variables)
- To show you that all these variables are required…
  - <(0,0,1,1,1,0,…,0,1), 0> need x_2
  - <(0,1,0,1,1,0,…,0,1), 0> need x_3
  - …..
  - <(0,1,1,1,1,0,…,0,0), 0> need x_{100}

Modeling Teaching Is tricky

- A straight forward algorithm requires k = 6 examples to produce the hidden conjunction (exactly).

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
- Teacher (Nature) provides the labels (f(x))
  - <(1,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0>
  - <(1,1,1,1,1,0,...0,1,1), 1>
  - <(1,0,1,1,1,0,...0,1,1), 0>
  - <(1,1,1,1,1,0,...0,0,1), 1>
  - <(1,0,1,0,0,0,...0,1,1), 0>
  - <(1,1,1,1,1,1,…,0,1), 1>
  - <(0,1,0,1,0,0,...0,1,1), 0>
- Skip

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$$

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - <(1,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0>

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$$

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - <(1,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0> ← learned nothing
  - <(1,1,1,1,1,0,...0,1,1), 1>

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$$

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - <(1,1,1,1,1,1,...,1,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$
  - <(1,1,1,0,0,0,...,0,0), 0> ← learned nothing
  - <(1,1,1,1,1,0,...0,1,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - <(1,0,1,1,0,0,...0,0,1), 0> ← learned nothing
  - <(1,1,1,1,1,0,...0,0,1), 1>

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - <(1,1,1,1,1,1,…,1,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$
  - <(1,1,1,0,0,0,…,0,0), 0> learned nothing
  - <(1,1,1,1,1,0,...0,1,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - <(1,0,1,1,0,0,...0,0,1), 0> learned nothing
  - <(1,1,1,1,1,0,...0,0,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
  - <(1,0,1,0,0,0,...0,1,1), 0>
  - <(1,1,1,1,1,1,…,0,1), 1>
  - <(0,1,0,1,0,0,...0,1,1), 0>

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - <(1,1,1,1,1,1,…,1,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$
  - <(1,1,1,0,0,0,…,0,0), 0> learned nothing
  - <(1,1,1,1,1,0,...0,1,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - <(1,0,1,1,0,0,...0,0,1), 0> learned nothing
  - <(1,1,1,1,1,0,...0,0,1), 1> $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
  - <(1,0,1,0,0,0,...0,1,1), 0> Final hypothesis:
  - <(1,1,1,1,1,1,…,0,1), 1> $h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
  - <(0,1,0,1,0,0,...0,1,1), 0>

Is that good ?
Performance ?
# of examples ?

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))
- Algorithm: …….
  - <(1,1,1,1,1,1,…,1,1), 1>
  - <(1,1,1,0,0,0,…,0,0), 0>
  - <(1,1,1,1,1,0,...0,1,1), 1>
  - <(1,0,1,1,0,0,...0,0,1), 0>
  - <(1,1,1,1,1,0,...0,0,1), 1>
  - <(1,0,1,0,0,0,...0,1,1), 0>
  - <(1,1,1,1,1,1,…,0,1), 1>
  - <(0,1,0,1,0,0,...0,1,1), 0>

> - Is it good
> - Performance ?
> - # of examples ?

Final hypothesis:

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

- With the given data, we only learned an “approximation” to the true concept

Two Directions

Two Directions

- Can continue to analyze the probabilistic intuition:
  - Never saw x_1=0 in positive examples, maybe we'll never see it?
  - And if we will, it will be with small probability, so the concepts we learn may be pretty good
  - Good: in terms of performance on future data
  - PAC framework
- Mistake Driven Learning algorithms
  - (Now, we can only reason about #(mistakes), not #(examples))
  - Update your hypothesis only when you make mistakes
  - Good: in terms of how many mistakes you make before you stop, happy with your hypothesis.
  - Note: not all on-line algorithms are mistake driven, so performance measure could be different.

On-Line Learning

- Two new learning algorithms
  (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

Motivation

- Consider a learning problem in a very high dimensional space $\{x_1, x_2, x_3, \ldots\ldots, x_{1000000}\}$
- And assume that the function space is very sparse (every function of interest depends on a small number of attributes.)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge .x_{100}$$

Middle Eastern deserts are known for their sweetness

- Can we develop an algorithm that depends only weakly on the space dimensionality and mostly on the number of relevant attributes ?
- How should we represent the hypothesis?

On-Line Learning

- Of general interest; simple and intuitive model;
- Robot in an assembly line, language learning,…

- Important in the case of very large data sets, when the data cannot fit memory – Streaming data

- Evaluation: We will try to make the smallest number of mistakes in the long run.
  - What is the relation to the “real” goal?
  - Generate a hypothesis that does well on previously unseen data

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.

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

- A is a mistake bound algorithm for the concept class C, if MA(c) is a polynomial in n, the complexity parameter of the target concept.

On-Line/Mistake Bound Learning

- We could ask: how many mistakes to get to ϵ-δ (PAC) behavior?
  - Instead, looking for exact learning. (easier to analyze)
- No notion of distribution; a worst case model
- Memory: get example, update hypothesis, get rid of it (??)

On-Line/Mistake Bound Learning

- We could ask: how many mistakes to get to ϵ-δ (PAC) behavior
  - Instead, looking for exact learning. (easier to analyze)
- No notion of distribution; a worst case model
- Memory: get example, update hypothesis, get rid of it (??)
- Drawbacks:
  - Too simple
  - Global behavior: not clear when will the mistakes be made

On-Line/Mistake Bound Learning

- We could ask: how many mistakes to get to ϵ-δ (PAC) behavior
  - Instead, looking for exact learning. (easier to analyze)
- No notion of distribution; a worst case model
- Memory: get example, update hypothesis, get rid of it (??)
- Drawbacks:
  - Too simple
  - Global behavior: not clear when will the mistakes be made
- Advantages:
  - Simple
  - Many issues arise already in this setting
  - Generic conversion to other learning models
  - "Equivalent" to PAC for "natural" problems (?)

Generic Mistake Bound Algorithms

- Is it clear that we can bound the number of mistakes ?
- Let C be a finite concept class. Learn $f \in C$
- CON:
  - In the ith stage of the algorithm:
  - C_i all concepts in C consistent with all i-1 previously seen examples
  - Choose randomly $f \in C_i$ and use to predict the next example
  - Clearly, $C_{i+1} \subseteq C_i$ and, if a mistake is made on the ith example, then $|C_{i+1}| < |C_i|$ so progress is made.
- The CON algorithm makes at most $|C|$-1 mistakes
- Can we do better ?

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.

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 $|\{f_j \in C_i; f_j(e_i) = 0\}| < |\{f_j \in C_i; f_j(e_i) = 1\}|$

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

The Halving Algorithm

- Hard to compute
- In some cases Halving is optimal (C - class of all Boolean functions)
- In general, to be optimal, instead of guessing in accordance with the majority of the valid concepts, we should guess according to the concept group that gives the least number of expected mistakes (even harder to compute)

Learning Conjunctions

What's possible?

- There is a hidden conjunctions the learner is to learn

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- The number of conjunctions: 3^n
- log(|C|) = n
- The …… algorithm makes n mistakes
  - Learn …..
- k-conjunctions:
  - Assume that only k<<n attributes occur in the disjunction
- The number of k-conjunctions: $2^k C(n,k) \approx 2^k n^k$
  - log(|C|) = $k \log n$
  - Can we learn efficiently with this number of mistakes ?

Can mistakes be bounded in the non-finite case?

Can this bound be achieved?

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.**

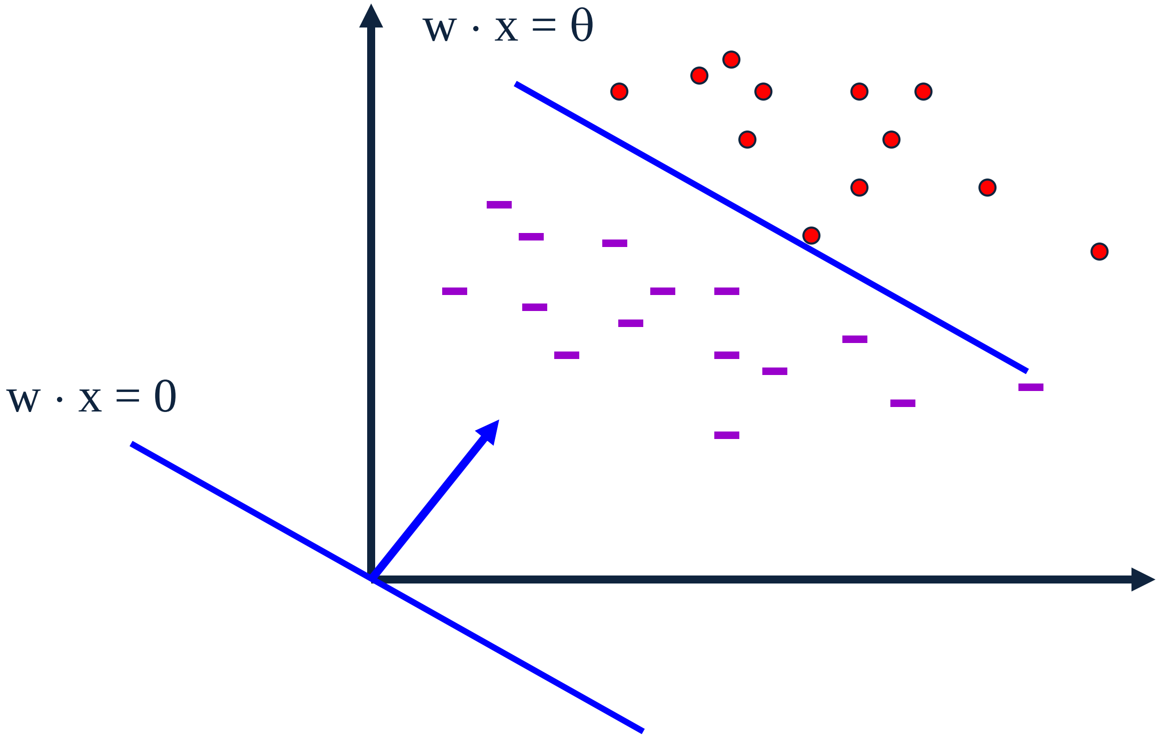
Linear Functions

$$f(x) = \begin{cases} 1 & \text{if } \; w_1 x_1 + w_2 x_2 + \ldots w_n x_n >= \theta \\ 0 & \text{Otherwise} \end{cases}$$

- Disjunctions $y = x_1 \vee x_3 \vee x_5$
  $y = (1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 >= 1)$
- At least m of n: $y =$ at least 2 of $\{x_1 , x_3 , x_5\}$
  $y = (1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 >= 2)$

- 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

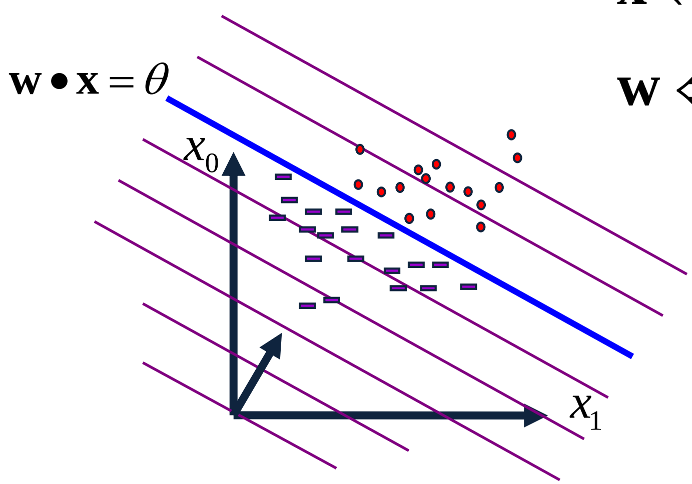

w · x = θ
w · x = 0

Footnote About the Threshold

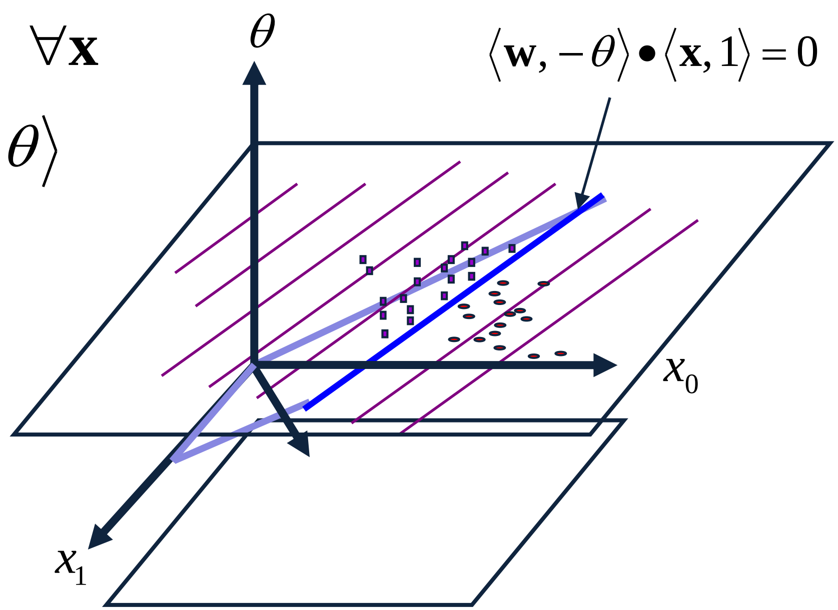
- On previous slide, Perceptron has no threshold
- But we don’t lose generality:

$$\mathbf{x} \Leftrightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

$$\mathbf{w} \Leftrightarrow \langle \mathbf{w}, -\theta \rangle$$



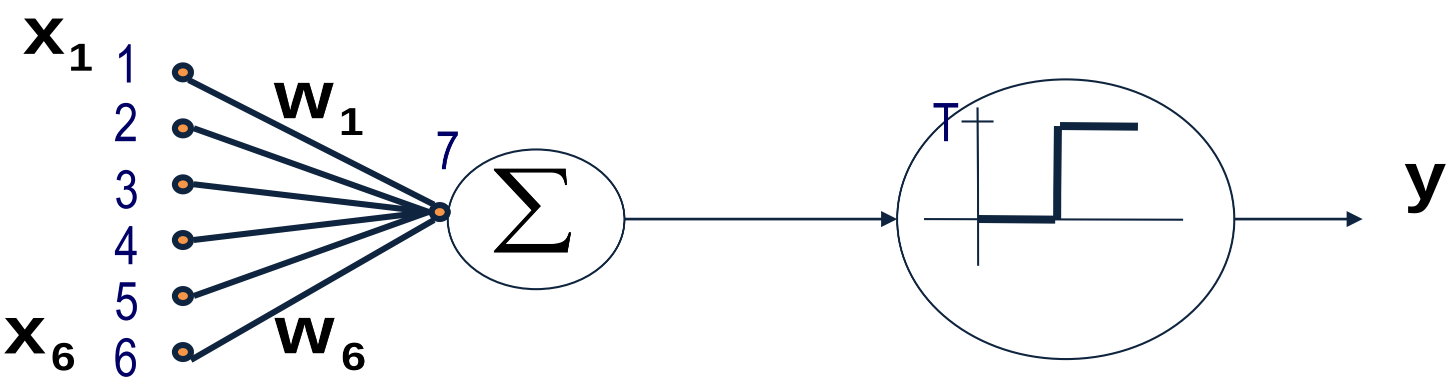
$$\Leftrightarrow$$



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



Perceptron learning rule

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

Perceptron

1. Initialize $w = 0 \in \mathbf{R}^n$
2. Cycle through all examples

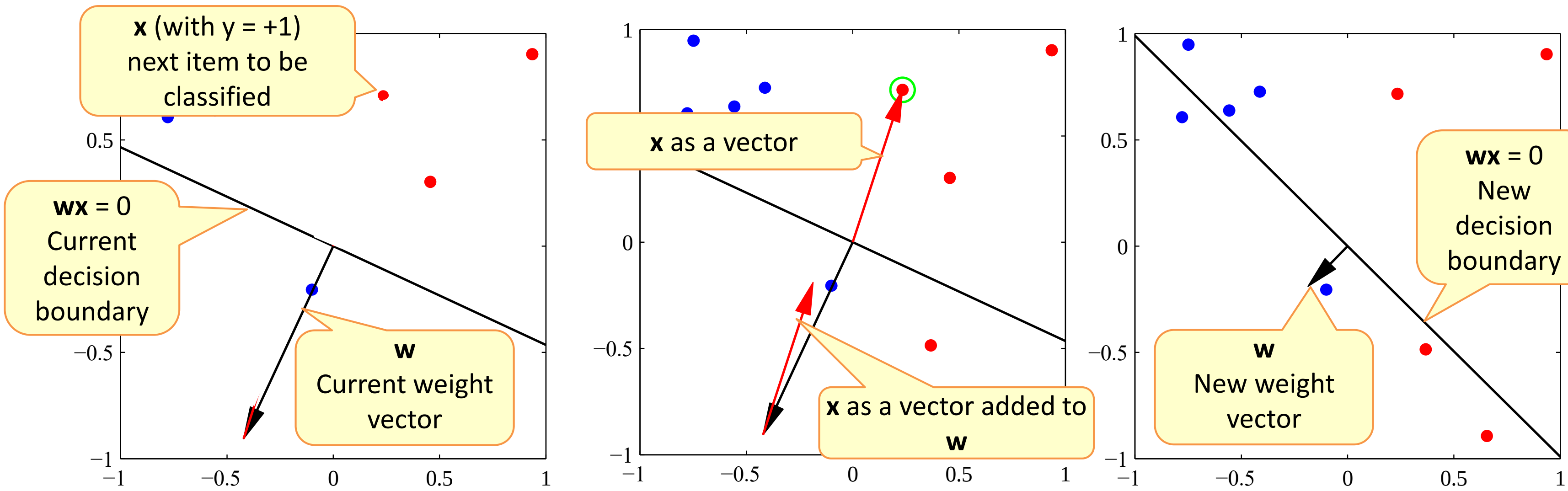
   a. Predict the label of instance x to be $y' = \mathrm{sgn}\{w \bullet x\}$

   b. If $y' \neq y$, update the weight vector:

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

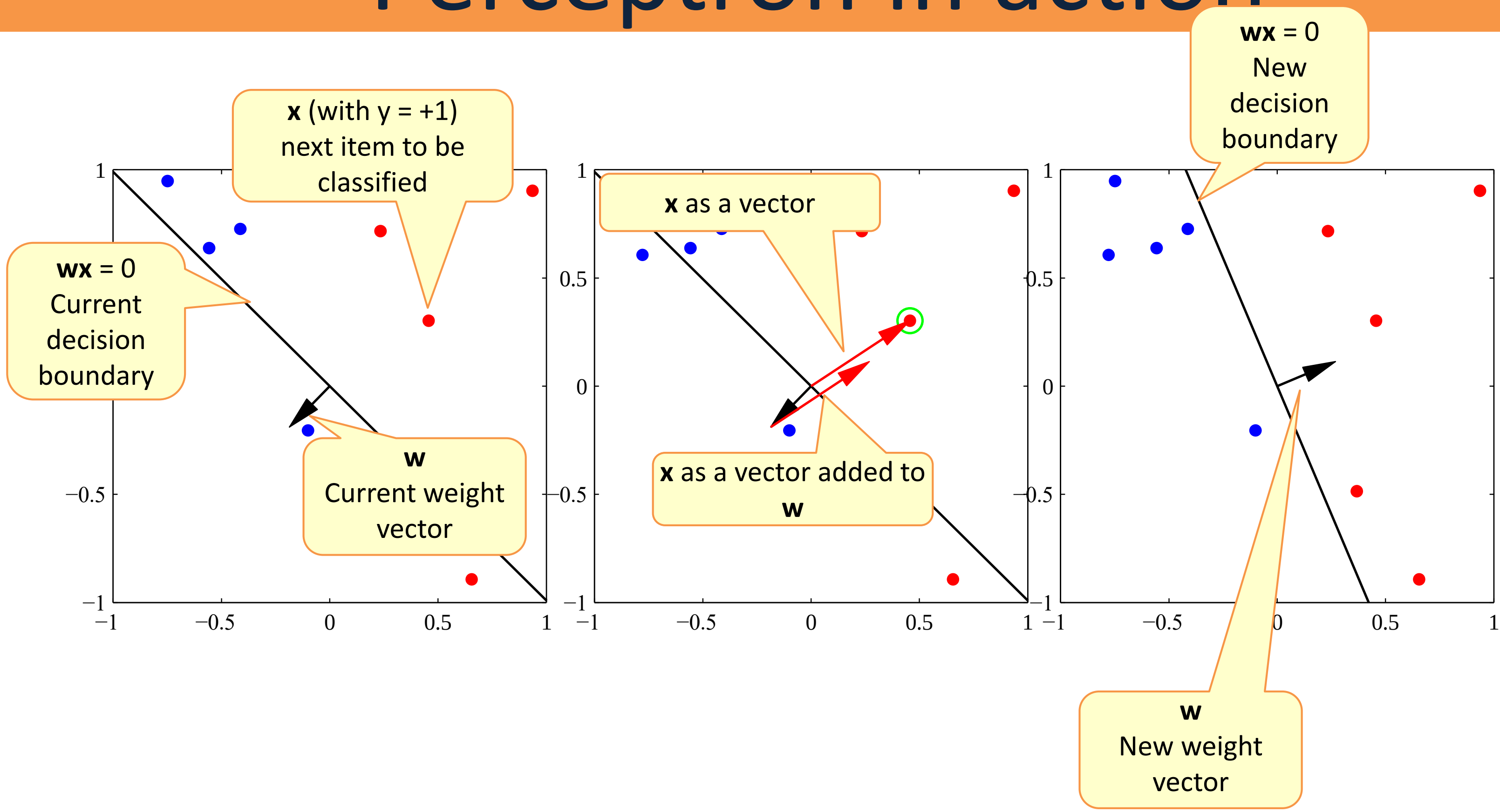
   Otherwise, if $y' = y$, leave weights unchanged.

Perceptron in action



(Figures from Bishop 2006)

Perceptron in action



(Figures from Bishop 2006)

- Positive
- Negative

Perceptron learning rule

- If x is Boolean, only weights of active features are updated
- Why is this important?

$$\mathbf{w}^{i+1} = \mathbf{w}^{i} + \mathbf{x}$$

$$\begin{pmatrix} w_1+1 \\ w_2 \\ w_3-1 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

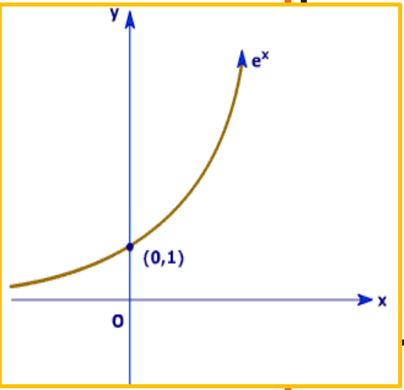
1. Initialize $w=0 \in \mathbf{R}^n$
2. Cycle through all examples

   a. Predict the label of instance x to be y' = sgn{w•x)

   b. If $y' \neq y$, update the weight vector to

   w = w + r y x (r - a constant, learning rate)

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

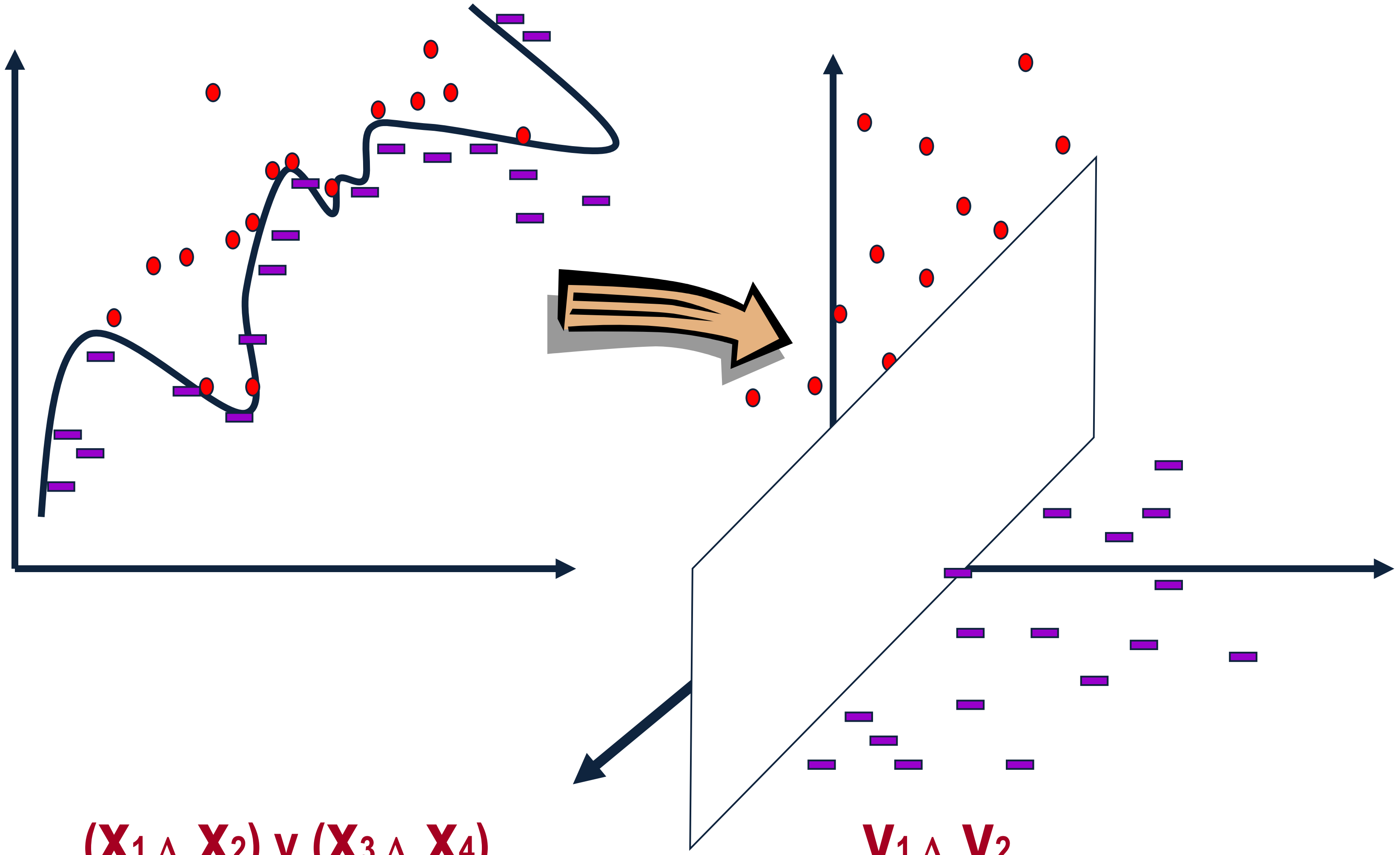


$$\mathbf{w} \bullet \mathbf{x} > 0 \text{ is equivalent to } \frac{1}{1+\exp\{-(\mathbf{w} \bullet \mathbf{x})\}} > 1/2$$

Perceptron Learnability

Perceptron

- Obviously can't learn what it can't represent (???)
  - Only linearly separable functions
- Minsky and Papert (1969) wrote an influential book demonstrating Perceptron's representational limitations
  - Parity functions can't be learned (XOR)
  - In vision, if patterns are represented with local features, can't represent symmetry, connectivity
- Research on Neural Networks stopped for years
- Rosenblatt himself (1959) asked,
  - *"What pattern recognition problems can be transformed so as to become linearly separable?"*



$(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ $\quad$ $y_1 \wedge y_2$

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 ?