Learning Conjunctions

What's possible?

Can mistakes be bounded in the non-finite case?

Can this bound be achieved?

- 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 numb
- log(|C|) =
- The elimina istakes
  - Learn fro active literals.

Last time:
- Learning Protocols
  - Exact (vs. in exact) Learning
  - On Line Learning
- # of examples needed to learn
- # of mistakes needed to learn
- Developed ideas on what might be possible (finite hypothesis classes)

- 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 ?

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.**

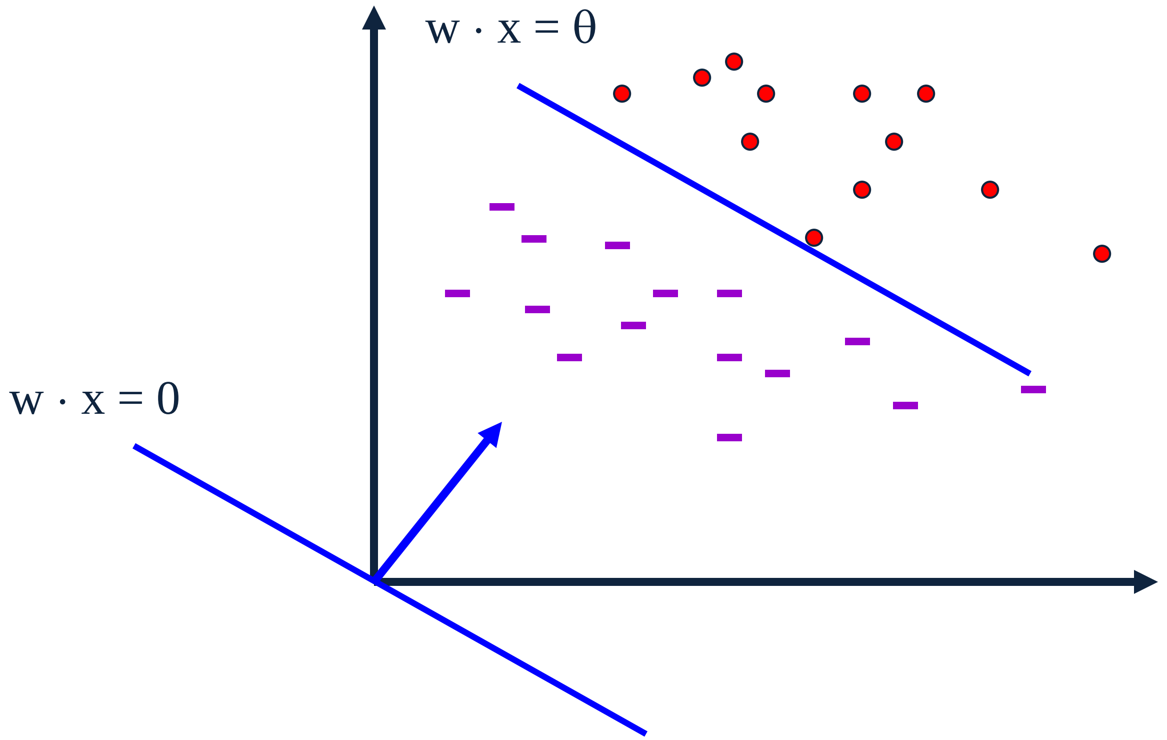
Linear Functions

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

- 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 $\{x_1, x_3, x_5\}$
  $y = (1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 >= 2)$ ✓

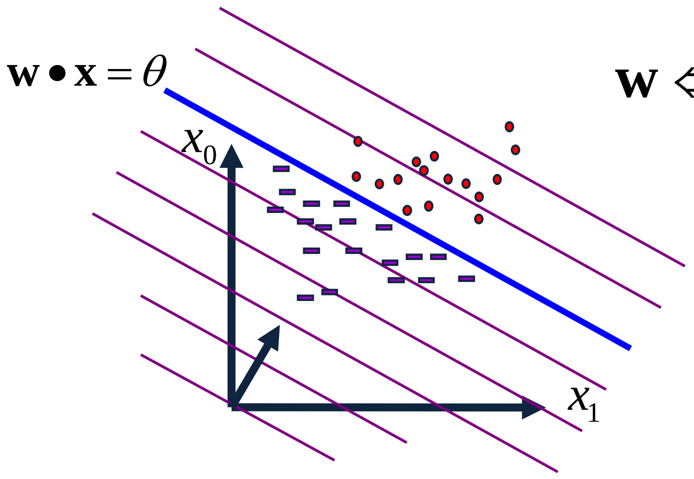
Linear Functions

- Exclusive-OR: $y = (x_1 \wedge x_2 \vee)(x_1 \wedge x_2)$ ✗
- Non-trivial DNF $y = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ ✗


$w \cdot x = \theta$
$w \cdot x = 0$

Footnote About the Threshold

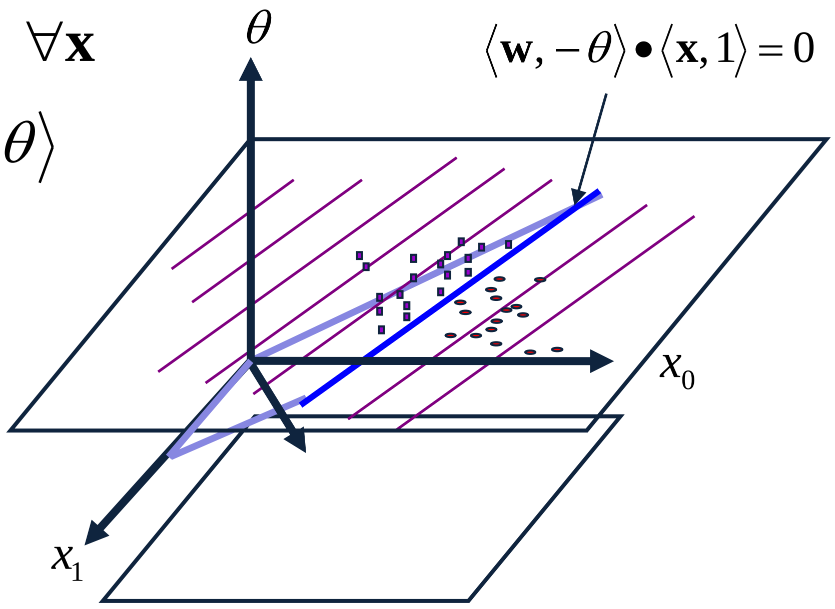
- On previous slide, Perceptron has no threshold
- But we don't lose generality:



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

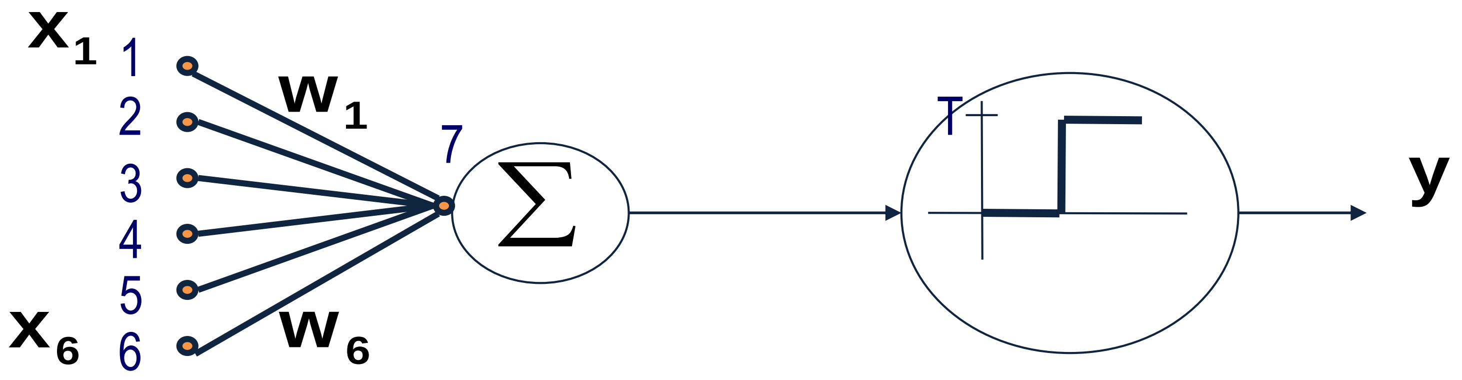
$$\mathbf{w} \Leftrightarrow \langle \mathbf{w}, -\theta \rangle$$

$$\Leftrightarrow$$


θ
$\langle \mathbf{w}, -\theta \rangle \bullet \langle \mathbf{x}, 1 \rangle = 0$
x_0
x_1

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)



Mistake driven algorithms

Analysis via mistake bound

Can mistakes be bounded in the non-finite case?

Can we achieve good bounds?

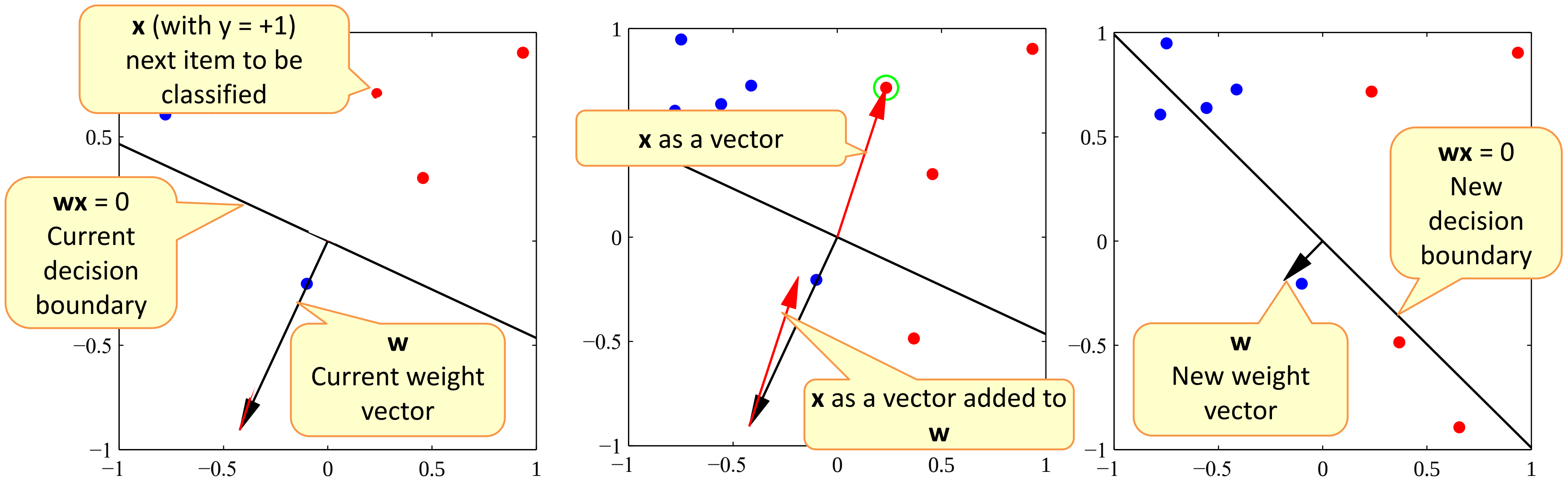
Perceptron

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

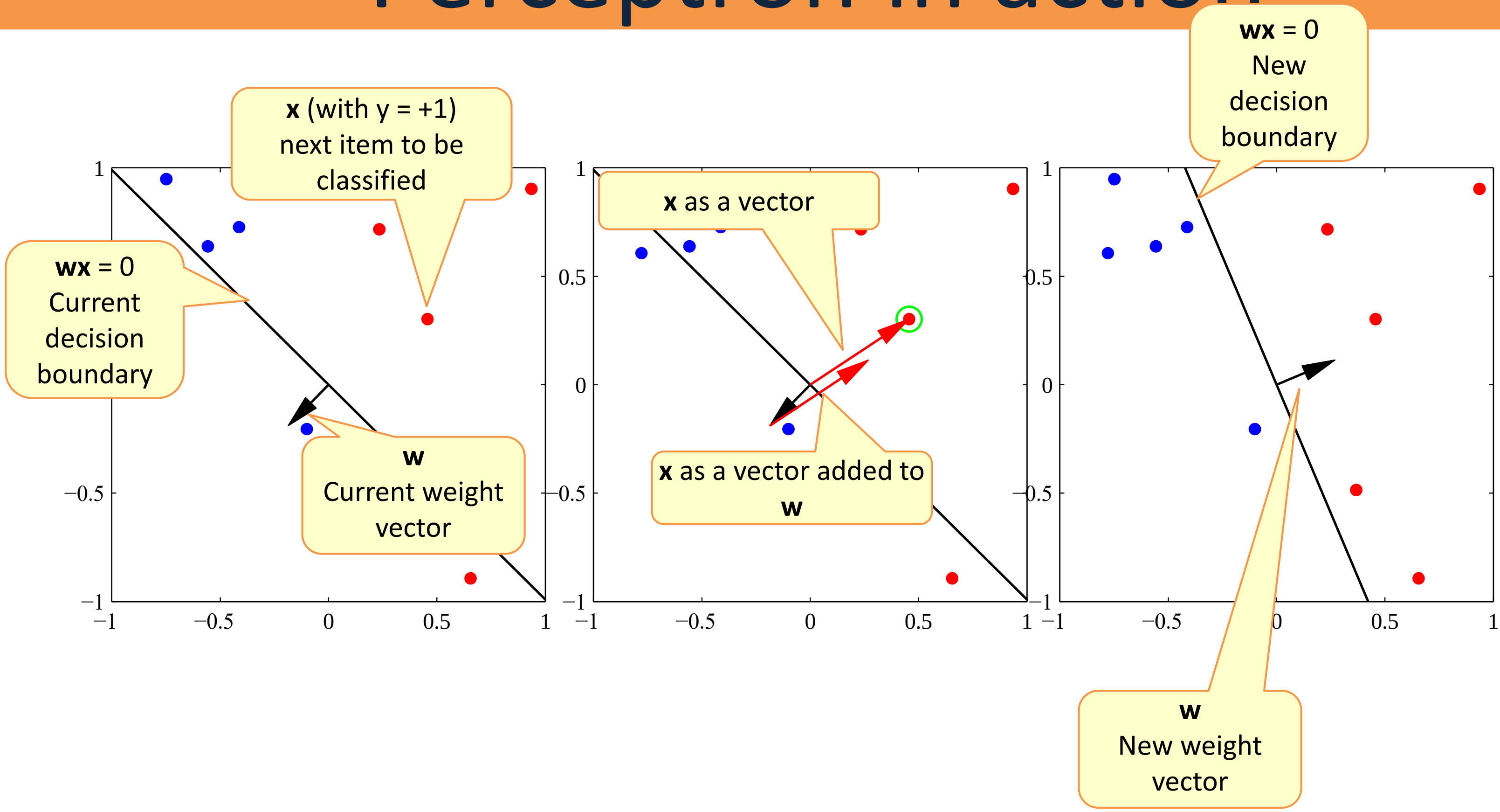
Perceptron in action



(Figures from Bishop 2006)

● Positive
● Negative

Perceptron in action



(Figures from Bishop 2006)

- ● Positive
- ● Negative

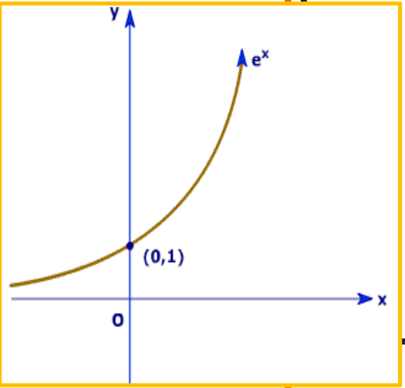
Perceptron learning rule

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

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

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

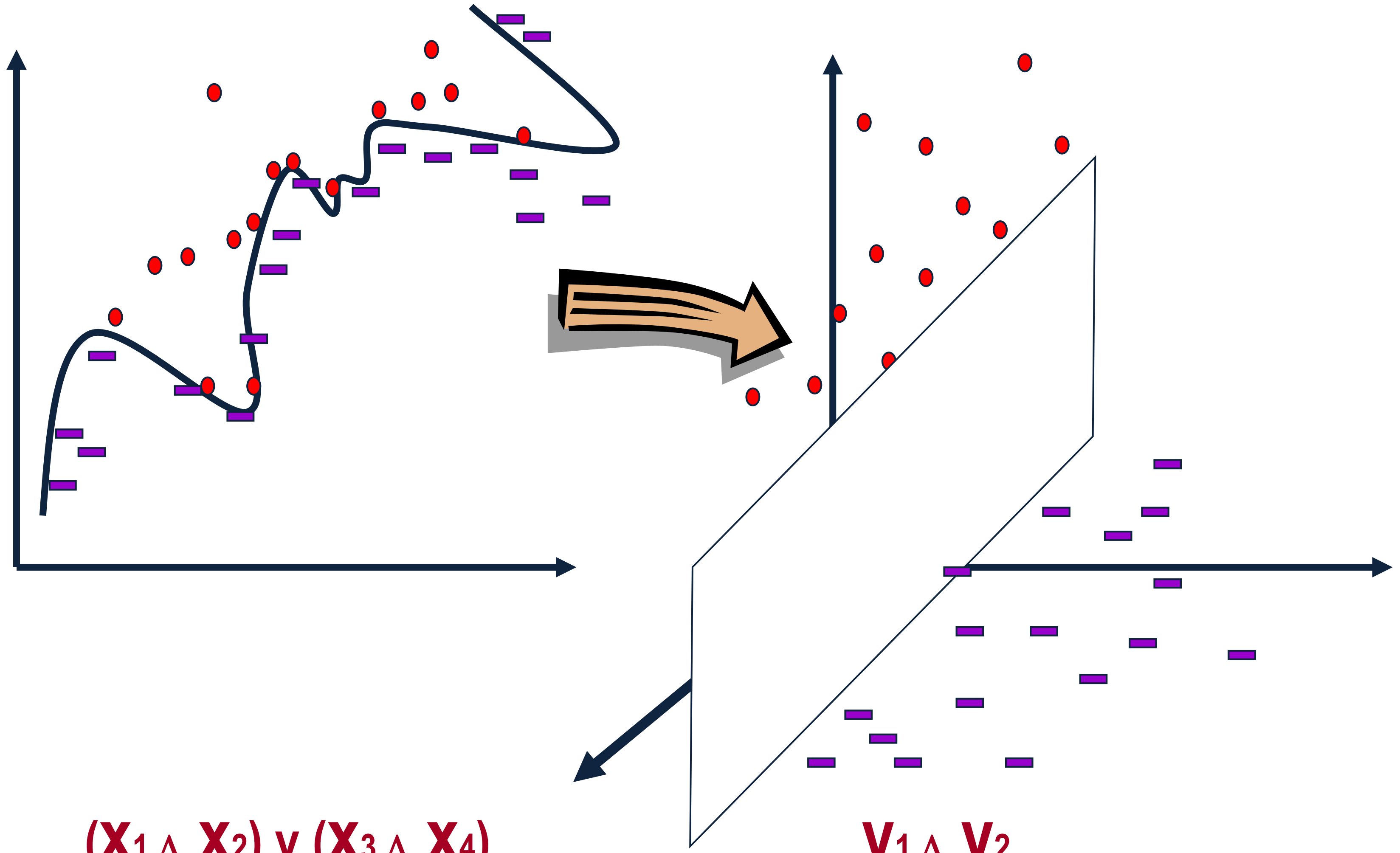
```
1. Initialize $w=0 \in \mathbf{R}^n$
2. Cycle through all examples
   a. Predict the label of instance x to be y' = sgn{w•x)
   b. If $y' \neq y$, update the weight vector to
      w = w + r y x        (r - a constant, learning rate)
      Otherwise, if y'=y, leave weights unchanged.
```



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

Perceptron Learnability

- 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?”*



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

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 ?

Perceptron

```
Input set of examples and their labels $\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, $\eta$, $\theta_{Init}$

• Initialize $\mathbf{w} \leftarrow \mathbf{0}$ and $\theta \leftarrow \theta_{Init}$
• for every training epoch:
• for every $x_j \in \mathcal{X}$:
    – $\hat{y} \leftarrow sign(\langle \mathbf{w}, x_j \rangle - \theta)$
    – if $(\hat{y} \neq y_j)$
        * $\mathbf{w} \leftarrow \mathbf{w} + \eta y_j x_j$
        * $\theta \leftarrow \theta + \eta y_j \theta_{Init}$
```

Just to make sure we understand that we learn both w and θ

Perceptron: Mistake Bound Theorem

- Maintains a weight vector $w \in \mathcal{R}^N$, $w_0 = (0,\ldots,0)$.
- 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* $(x_1; y_1),\ldots,: (x_t; y_t)$, *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

Perceptron-Mistake Bound

Proof: Let v_k be the hypothesis before the k-th mistake. Assume that the k-th mistake occurs on the input example (x_i, y_i).

Assumptions

$v_1 = \mathbf{0}$

$||\boldsymbol{u}|| = 1$

$y_i\, u \bullet \mathrm{x}_i \geq \gamma$

$$\therefore\ y_i(\vec{v_k} \cdot \vec{x_i}) \leq 0.$$

$$\begin{aligned} \vec{v_{k+1}} &= \vec{v_k} + y_i\vec{x_i} \\ \vec{v_{k+1}} \cdot \vec{u} &= \vec{v_k} \cdot \vec{u} + y_i(\vec{u} \cdot \vec{x_i}) \\ &\geq \vec{v_k} \cdot \vec{u} + \gamma \\ \therefore\ \vec{v_{k+1}} \cdot \vec{u} &\geq k\gamma \end{aligned}$$

1. Note that the bound does not depend on the dimensionality nor on the number of examples.
2. Note that we place weight vectors and examples in the same space.

$$\begin{aligned} ||\vec{v_{k+1}}||^2 &= ||\vec{v_k}||^2 + 2y_i(\vec{v_k} \cdot \vec{x_i}) + ||\vec{x_i}||^2 \\ &\leq ||\vec{v_k}||^2 + R^2 \\ \therefore\ ||\vec{v_{k+1}}||^2 &\leq kR^2 \end{aligned}$$

Therefore,

$$\underline{\sqrt{k}R} \geq ||\vec{v_{k+1}}|| \geq \vec{v_{k+1}} \cdot \vec{u} \geq \underline{k\gamma}. \quad \Longrightarrow \quad k < R^2 / \gamma^2$$

The second inequality follows because $||\vec{u}|| \leq 1$.

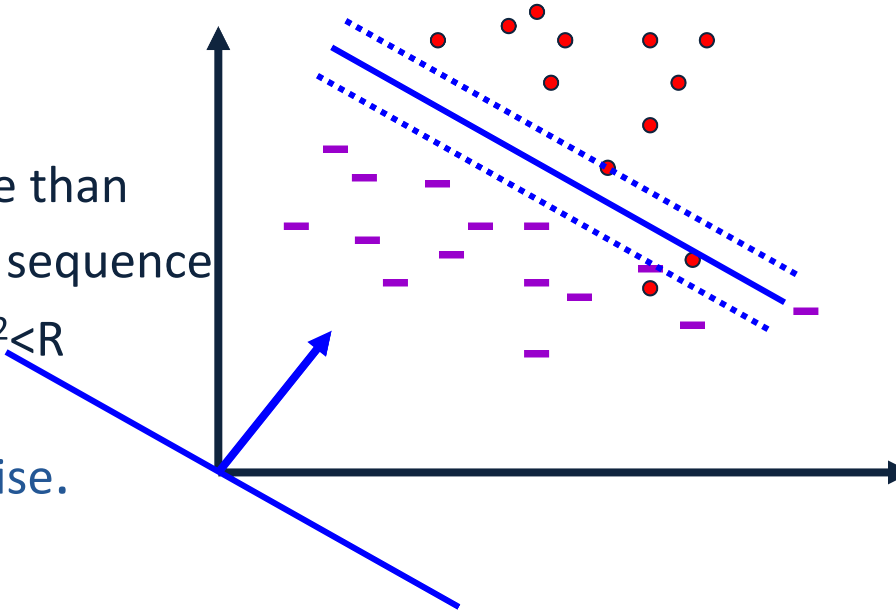
Robustness to Noise

Non separable case

- 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 $-\, y_i\, w \cdot x_i$
- 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.



Perceptron for Boolean Functions

- How many mistakes will the Perceptron algorithms make when learning a k-disjunction?
- Try to figure out the bound
- Find a sequence of examples that will cause Perceptron to make O(n) mistakes on k-disjunction on n attributes.
- (Where is n coming from?)

Winnow Algorithm

```
Initialize: $\theta = n;\ w_i = 1$
Prediction is 1 iff $w \bullet x \ge \theta$
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 $w \bullet x \ge \theta$, $w_i \leftarrow w_i/2$ (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 - Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024;\ w = (1,1,\ldots,1)$

$<(1,1,\ldots,1),+> \quad w \bullet x \geq \theta \quad w = (1,1,\ldots,1)$ ok

$<(0,0,\ldots,0),-> \quad w \bullet x < \theta \quad w = (1,1,\ldots.,1)$ ok

$<(0,0,111,,,,0),-> \quad w \bullet x < \theta \quad w = (1,1,\ldots.,1)$ ok

$<(1,0,0,\ldots,0),+> \quad w \bullet x < \theta \quad w = (2,1,\ldots.,1)$ mistake

$<(1,0,1,1,0..,0),+> \quad w \bullet x < \theta \quad w = (4,1,2,2\ldots,1)$ mistake

$<(1,0,1,0,0..,1),+> \quad w \bullet x < \theta \quad w = (8,1,4,2\ldots,2)$ mistake

.... log(n/2) (for each good variable)

$w = (512,1,256,256,\ldots.,256)$

$<(1,0,1,0\ldots,1),+> \quad w \bullet x \geq \theta \quad w = (512,1,256,256,\ldots.,256)$ ok

$<(0,0,1,0.111..,0),-> \quad w \bullet x \geq \theta \quad w = (512,1,0,..0,\ldots,256)$ mistake **(elimination version)**

....

$w = (1024,1024,0,0,0,1,32,\ldots,1024,1024)$ **(final hypothesis)**

Notice that the same algorithm will learn a conjunction over these variables (w=(256,256,0,…32, …256,256))

Winnow – Mistake Bound

Analysis

- 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)

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.

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)

Analysis

2. $v < 2(u + 1)$

Total weight TW=n initially

Mistake on positive: $TW(t+1) < TW(t) + n$

Mistake on negative: $TW(t+1) < TW(t) - n/2$

$0 < TW < n + u\,n - v\,n/2 \Rightarrow v < 2(u+1)$

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

Summary of Algorithms

Perceptron & Winnow

- Examples: $x \in \{0,1\}^n$; or $x \in R^n$ (indexed by k) ; Hypothesis: $w \in R^n$
- Prediction: $y \in \{-1,+1\}$: Predict: $y = 1$ iff $w \cdot x > \theta$
- Update: Mistake Driven
- Additive weight update algorithm: $w \leftarrow w + r\, y_k\, x_k$
  - (Perceptron, Rosenblatt, 1958. Variations exist)
  - In the case of Boolean features:

$$\text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow w_i + 1 \text{ (if } x_i = 1) \text{ (promotion)}$$
$$\text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i - 1 \text{ (if } x_i = 1) \text{ (demotion)}$$

- Multiplicative weight update algorithm $w_i \leftarrow w_i \exp\{y_k\, x_i\}$
- (Winnow, Littlestone, 1988. Variations exist)
  - Boolean features:

$$\text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow 2w_i \text{ (if } x_i = 1) \text{ (promotion)}$$
$$\text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i/2 \text{ (if } x_i = 1) \text{ (demotion)}$$

Practical Issues and Extensions

Extensions

- There are many extensions that can be made to these basic algorithms.
- Some are necessary for them to perform well
  - Regularization (next; will be motivated in the next section, COLT)
- Some are for ease of use and tuning
  - Converting the output of a Perceptron/Winnow to a conditional probability

$$P(y = +1 \mid x) = [1 + \exp(-Awx)]^{-1}$$

  - Can tune the parameter A
- Multiclass classification (later)
- Key efficiency issue: Infinite attribute domain

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 $\{(x_1, y_1),\ldots(x_m, y_m)\}$
- Number of epochs T
- Output: a list of weighted perceptrons $\{(v_1, c_1),\ldots,(v_k, c_k)\}$
- Initialize: $k=0$; $v_1 = 0$, $c_1 = 0$
- Repeat T times:
  - For $i = 1,\ldots m$:
  - 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 $\{(v_1, c_1),\ldots(v_k, c_k)\}$; a new example x

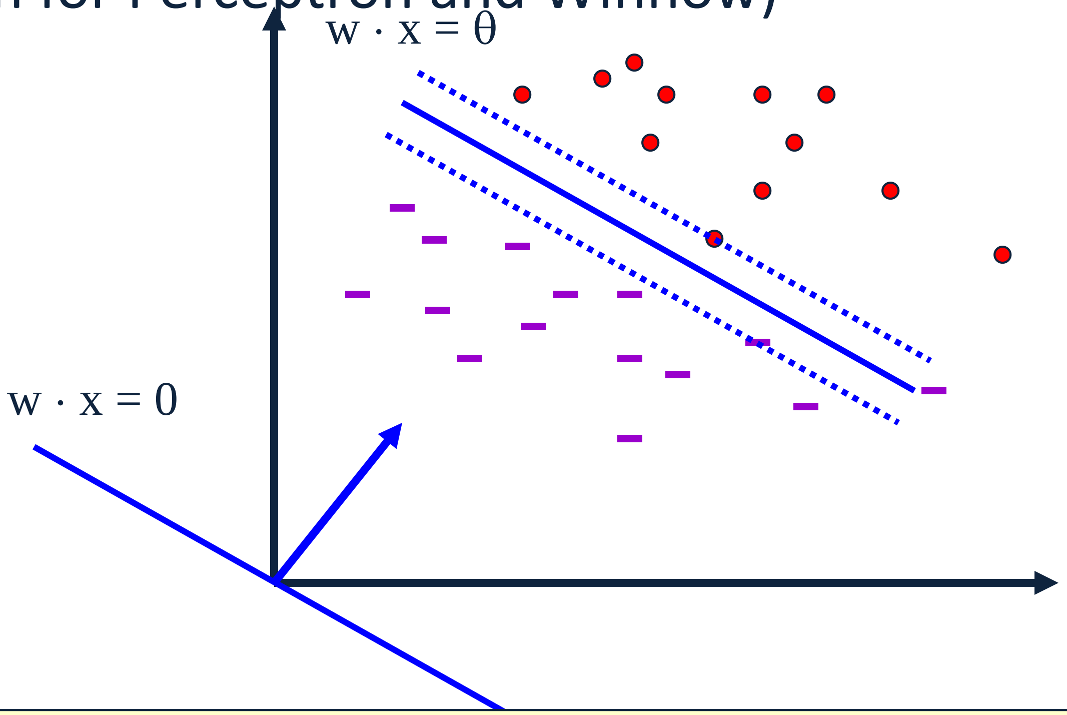
Predict the label(x) as follows:

$$y(x) = \text{sign} [\sum_{1, k} c_i \, \text{sign}(v_i \cdot x)]$$

Averaged Perceptron

II Perceptron with Margin

- Thick Separator (aka as Perceptron with Margin) (Applies both for Perceptron and Winnow)
- Promote if:
  - $w x - \theta < \gamma$
- Demote if:
  - $w x - \theta > \gamma$



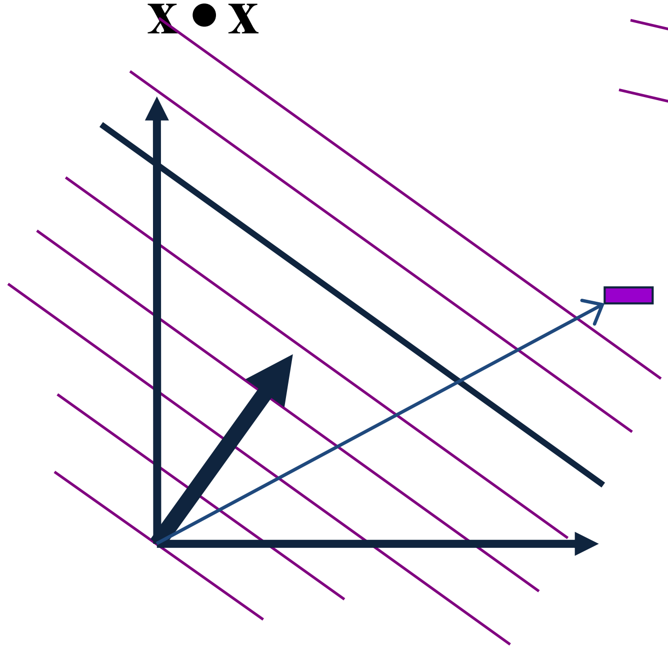
Note: γ is a functional margin. Its effect could disappear as w grows. Nevertheless, this has been shown to be a very effective algorithmic addition. (Grove & Roth 98,01; Karov et. al 97)

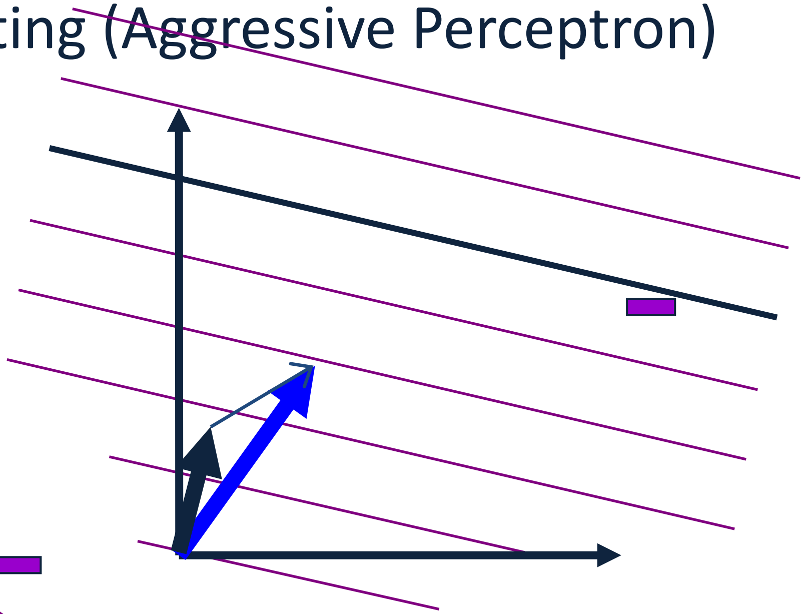
Other Extensions

- Threshold relative updating (Aggressive Perceptron)

$$\mathbf{w} \leftarrow \mathbf{w} + r\,\mathbf{x}$$

$$r = \frac{\theta - \mathbf{w} \bullet \mathbf{x}}{\mathbf{x} \bullet \mathbf{x}}$$





Equivalent to updating on the same example multiple times

SNoW (also in LBJava)

- Several of these extensions (and a couple more) are implemented in the SNoW learning architecture that supports several linear update rules (Winnow, Perceptron, naïve Bayes)
- Supports
  - Regularization(averaged Winnow/Perceptron; Thick Separator)
  - Conversion to probabilities
  - Automatic parameter tuning
  - True multi-class classification
  - Feature Extraction and Pruning
  - Variable size examples
  - Good support for large scale domains in terms of number of examples and number of features.
  - Very efficient
  - Many other options
- [Download from: http://cogcomp.cs.illinois.edu/page/software]

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 $(w^+ - w^-)\bullet x \le \theta$, $\quad w_i^+ \leftarrow 2w_i^+ \quad w_i^- \leftarrow \frac{1}{2}w_i^-$ where $x_i = 1$ (promotion)

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

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

Winnow – A Robust Variation

- Winnow is robust in the presence of various kinds of noise.
  - (classification noise, attribute noise)
- Moving Target:
  - The target function changes with time.
- Importance:
  - sometimes we learn under some distribution but test under a slightly different one. (e.g., natural language applications)
  - The algorithm we develop provides a good insight into issues of Adaptation

Extensions

Winnow – A Robust Variation

Extensions

- 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)

Good project:

Push weights to 0 (simple hypothesis, L1 regularization (Lasso)) vs. bounding them away from zero

– impact on adaptation

Winnow R – Mistake Bound

Extensions

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

2. $v < 2(u + 1)$

Total weight TW=n initially

Mistake on positive: $TW(t+1) < TW(t) + n$

Mistake on negative: $TW(t+1) < TW(t) - n/4$ ←

$0 < TW < n + u\,n - v\,n/4 \Rightarrow v < 4(u+1)$

Administration

TODAY:

- Administration: HW, Projects
- Flipped Class
- Continuing with On-Line Learning

HW2

- Decision Trees, Expressivity of Models, Features
- Key Reporting Module (RM):
  - Train a model on a given Training Set
  - Report 5-fold cross validation
  - Report results on a supplied Test Set.
- (a) Convert Data to Feature Representation (given; can be augmented)
  - 2000 * 270 dimensions
- (b) Program SGD; run RM
- (c) Use Weka to Learn DT using ID2; run RM.
- (d) Use Weka to learn DT(depth=4) and DT(depth=8); run RM
- (e) Use Weka to generate 100 different DT(d=4)
  - Generate 100 dimensional data, each dimension is the prediction of a DT
  - Run (b) on the new data
- Compare algorithms from b,c,d,e.

Stopping Criterion left ambiguous deliberately. Multiple options; think and make a decision (e.g., based on loss; based on error).

Features; feature types; instances space transformation.

Projects

- Projects proposals are due on March 10 2017
- Within a week we will give you an approval to continue with your project along with comments and/or a request to modify/augment/do a different project. There will 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

- Fake News Challenge :- http://www.fakenewschallenge.org/
- 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

What have you learned (on your own)

- The feasibility of Mistake Bounds
  - Con
  - Halving
  - Perceptron
- Algorithms
  - Perceptron
    - + Analysis
  - Winnow
    - + Analysis (special case)
    - The general case
- Algorithms could behave differently
  - Averaged version of Perceptron/Winnow is as good as any other linear learning algorithm, if not better.

- Why do I include Perceptron in this bullet?
- What's interesting about it?

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

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

- 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 $Q(z, w)$

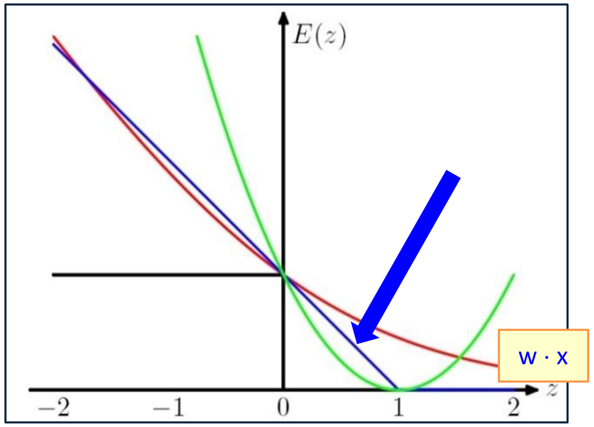
Stochastic Gradient Algorithms

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

- 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): $w_{t+1} = w_t + r\, y_t\, x_t$

Think about the case where **x** is a Boolean vector.

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 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".

Algorithmic Approaches

- Focus: Two families of algorithms (one of the on-line representative)
  - Additive update algorithms: Perceptron
    - SVM is a close relative of Perceptron
  - Multiplicative update algorithms: Winnow
    - Close relatives: Boosting, Max entropy/Logistic Regression

How to Compare?

- Generalization
  - (since the representation is the same): How many examples are needed to get to a given level of accuracy?
- Efficiency
  - How long does it take to learn a hypothesis and evaluate it (per-example)?
- Robustness; Adaptation to a new domain, ….

Sentence Representation

S= I don’t know whether to laugh or cry

- Define a set of features:
  - features are relations that hold in the sentence
- Map a sentence to its feature-based representation
  - The feature-based representation will give some of the information in the sentence
- Use this as an example to your algorithm

Domain Characteristics

Sentence Representation

S= I don't know whether to laugh or cry

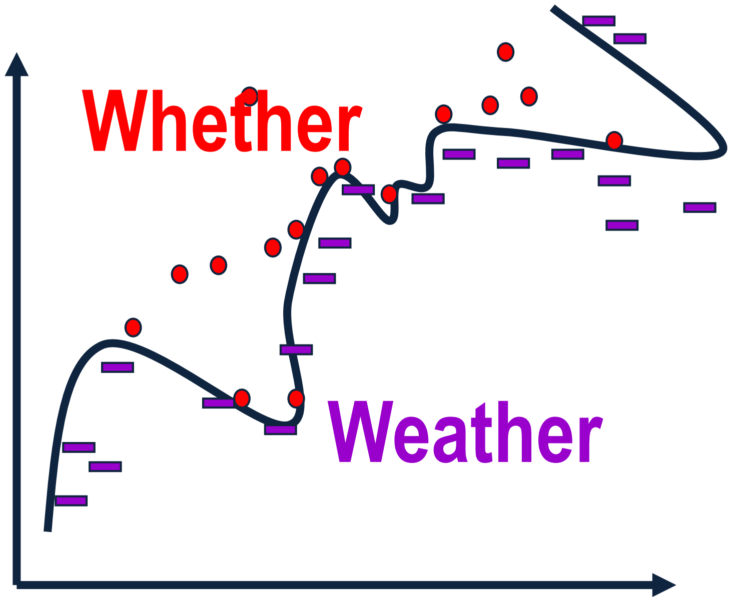
- Define a set of features:
  - features are properties that hold in the sentence

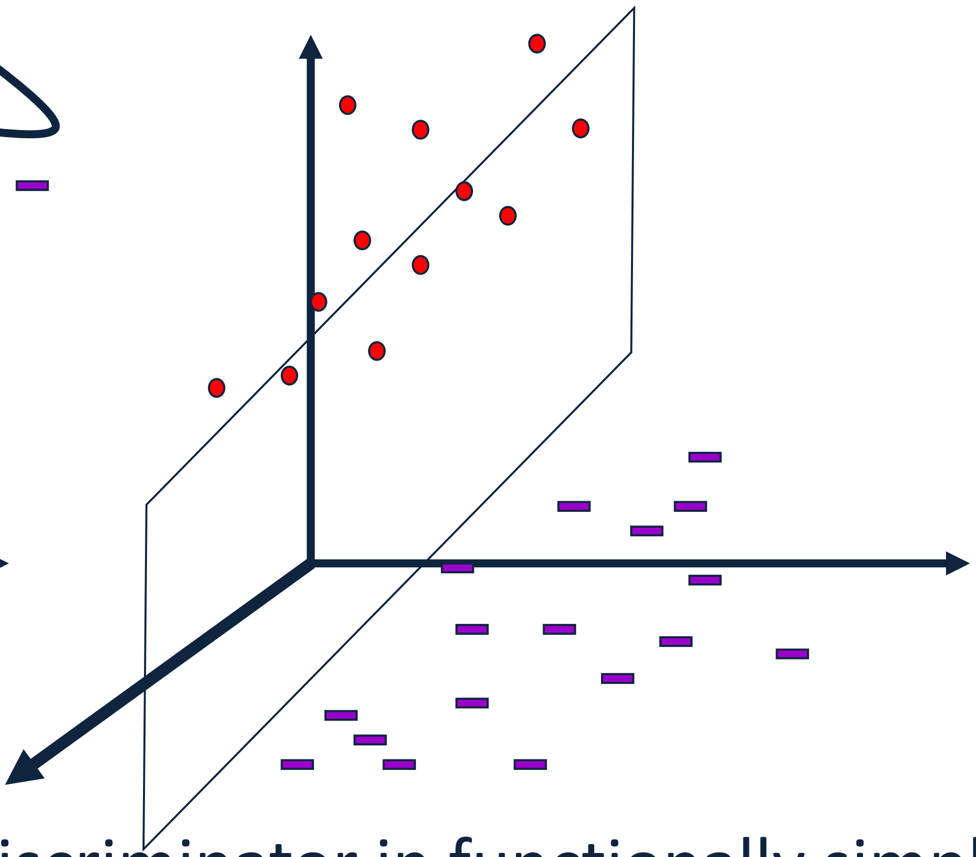
- Conceptually, there are two steps in coming up with a feature-based representation
  - What are the information sources available?
    - Sensors: words, order of words, properties (?) of words
  - What features to construct based on these?

Domain Characteristics

Why is this distinction needed?

Embedding





New discriminator in functionally simpler

$x_1 x_2 \overline{x}_3 \vee \overline{x}_1 x_4 \overline{x}_3 \vee x_3 \overline{x}_2 x_5$ $y_1 \vee y_4 \vee y_5$

Domain Characteristics

Domain
Characteristics

- The number of potential features is very large
- The instance space is sparse
- Decisions depend on a small set of features: the function space is sparse
- Want to learn from a number of examples that is small relative to the dimensionality

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 algoirithms depends heavily on sparseness of features space:
  - Advantage to additive. Generalization depend on input ||x||
    - (Kivinen/Warmuth 95).

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: $\|\|x\|\|_\infty = \max_i \|x_i\|$

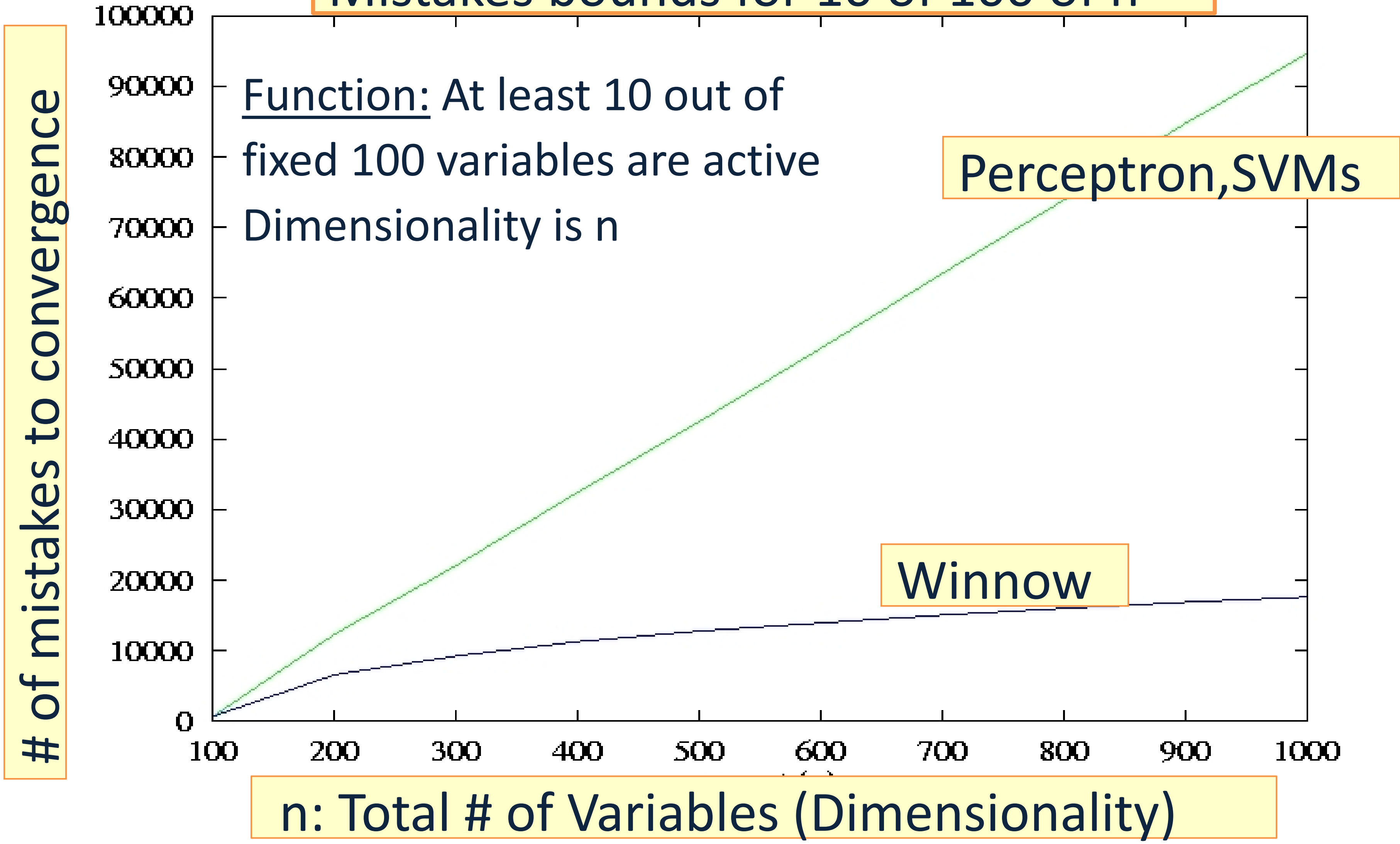
  - Multiplicative algorithms:
    - Bounds depend on $||\mathbf{u}||$, the separating hyperplane; i: example #)
    - $M_w = 2\ln n \; ||u||_1^2 \; \underline{\max_i ||x^{(i)}||_\infty^2} \; / \min_i (u \cdot x^{(i)})^2$
    - Do not care much about data; advantage with sparse target u
  - Additive algorithms:
    - Bounds depend on $||\mathbf{x}||$ (Kivinen / Warmuth, '95)
    - $M_p = ||u||_2^2 \; \underline{\max_i ||x^{(i)}||_2^2} / \min_i (u \cdot x^{(i)})^2$
    - Advantage with few active features per example

$M_w = 2\ln n \, ||u||_1^2 \max_i ||x^{(i)}||_\infty^2 / \min_i (u \cdot x^{(i)})^2$
$M_p = ||u||_2^2 \max_i ||x^{(i)}||_2^2 / \min_i (u \cdot x^{(i)})^2$

Examples

- Extreme Scenario 1: Assume the u has exactly k active features, and the other n-k are 0. That is, only k input features are relevant to the prediction. Then:
  - $||\mathbf{u}||_2, = k^{1/2}$; $||\mathbf{u}||_1, = k$; $\max ||x||_2, = n^{1/2}$; $\max ||x||_\infty, = 1$
  - We get that: $M_p = kn$; $M_w = 2k^2 \ln n$
  - Therefore, if k<<n, Winnow behaves much better.
- Extreme Scenario 2: Now assume that u=(1, 1,….1) and the instances are very sparse, the rows of an nxn unit matrix. Then:
  - $||\mathbf{u}||_2, = n^{1/2}$; $||\mathbf{u}||_1, = n$; $\max ||x||_2, = 1$; $\max ||x||_\infty, = 1$
  - We get that: $M_p = n$; $M_w = 2n^2 \ln n$
  - Therefore, Perceptron has a better bound.

Mistakes bounds for 10 of 100 of n



Efficiency

- Dominated by the size of the feature space
- Most features are functions (e.g. conjunctions) of raw attributes

$$X(x_1, x_2, x_3, \ldots x_k) \rightarrow \mathrm{X}(\chi_1(\mathbf{x}), \chi_2(\mathbf{x}), \chi_3(\mathbf{x}) \ldots \chi_n(\mathbf{x})) \qquad \mathbf{n} >> \mathbf{k}$$

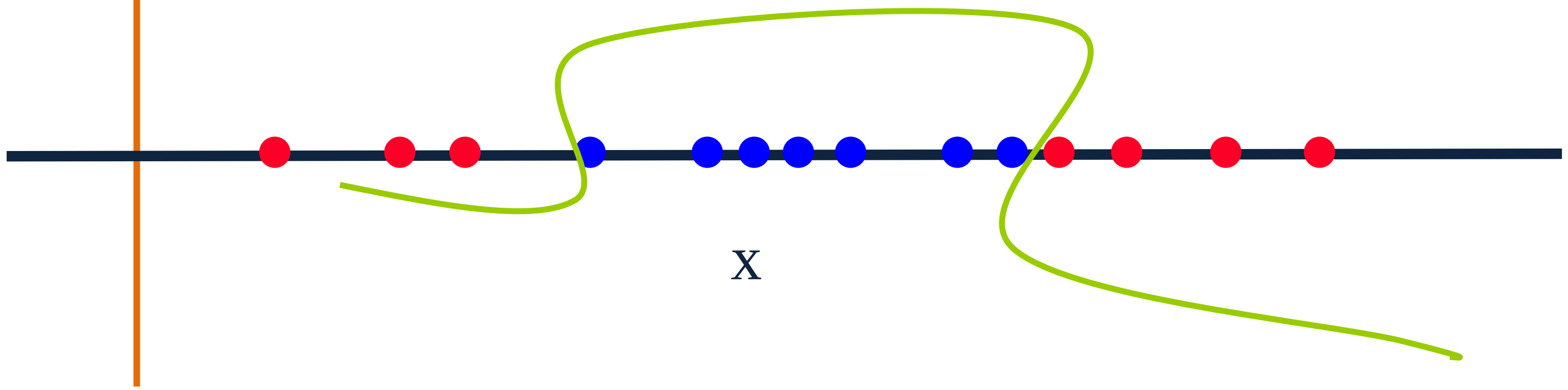
- Additive algorithms allow the use of Kernels
  - No need to explicitly generate complex features

$$f(x) = \sum_i c_i K(x, x_i)$$

- Could be more efficient since work is done in the original feature space, but expressivity is a function of the kernel expressivity.

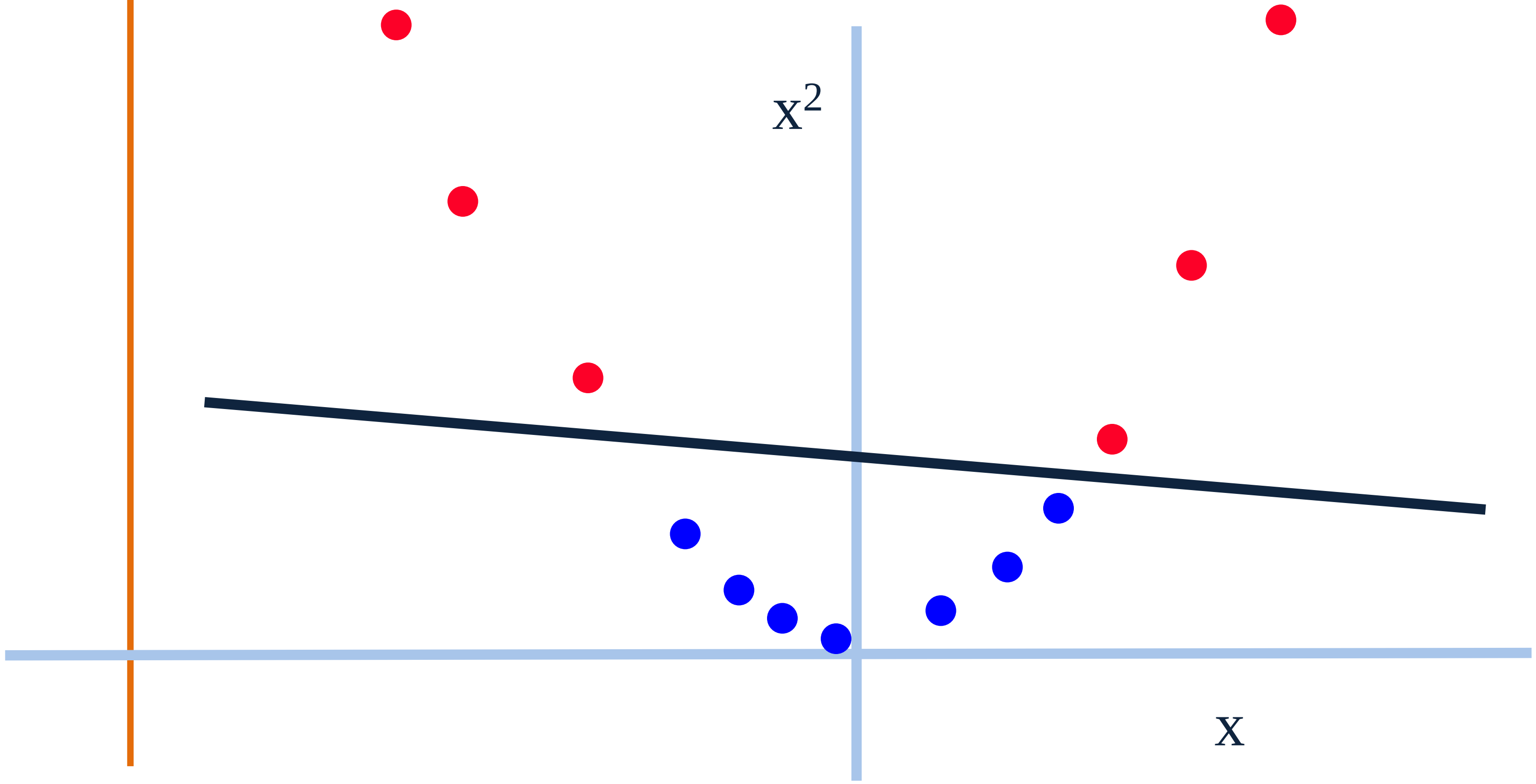
Functions Can be Made Linear

- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions

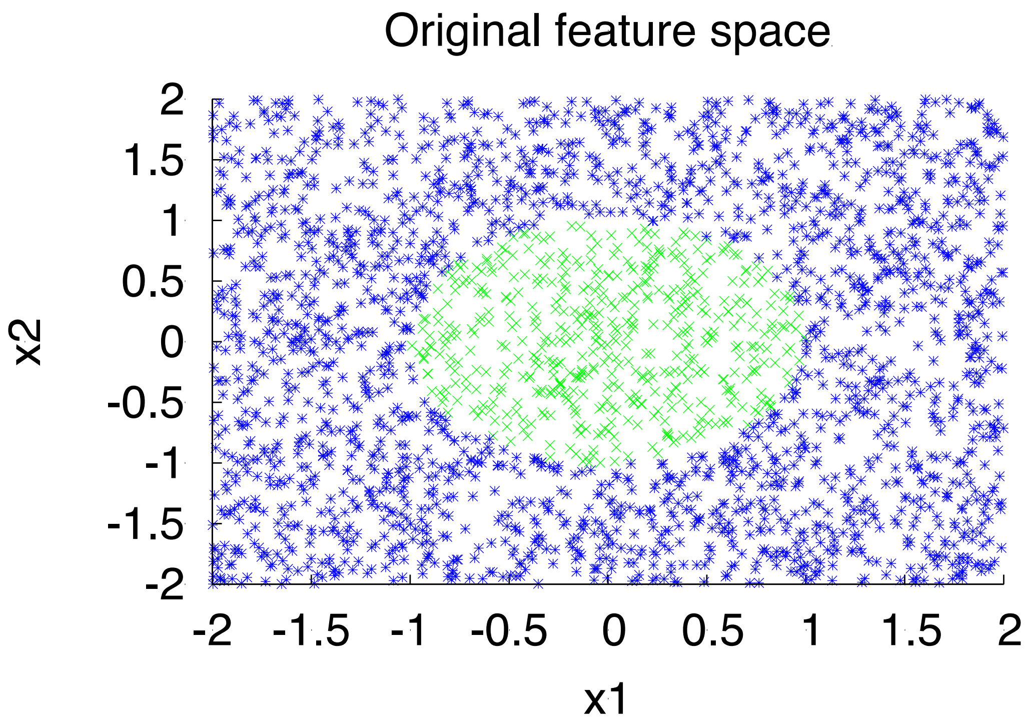


Blown Up Feature Space

- Data are separable in <x, x^2> space



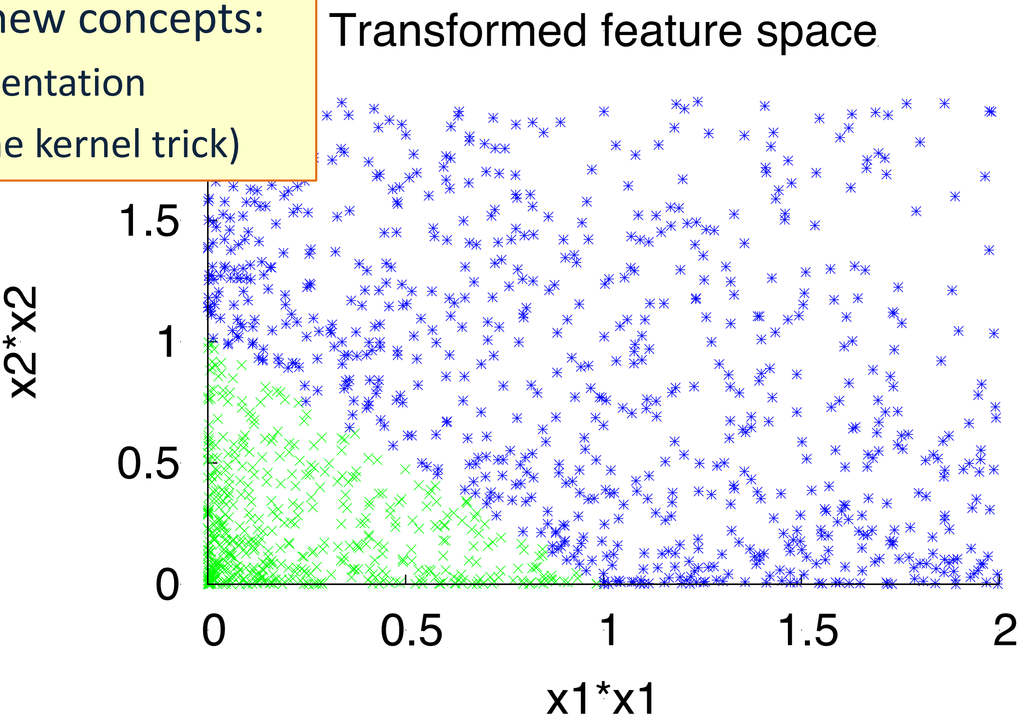
Making data linearly separable



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

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 $x'_1 + x'_2 \leq 1$

Dual Representation

$$\textbf{Examples}: \mathbf{x} \in \{0,1\}^n; \qquad \textbf{Hypothesis}: \mathbf{w} \in R^n$$

$$f(x) = Th_\theta\left(\sum_{i=1}^n w_i x_i(x)\right)$$

$$\text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow w_i + 1 \text{ (if } x_i = 1) \text{ (promotion)}$$

$$\text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i - 1 \text{ (if } x_i = 1) \text{ (demotion)}$$

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

$$w = w + x^1 + x^2 - x^4$$

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

$$w = \sum_{1,m} r\, \alpha_i\, y_i\, x^i$$

- Where α_i is the number of mistakes made on x^i.

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

$$= \left(\sum_{1,m} r\alpha_i\, y_i\, x^i\right) \cdot x$$
$$= \sum_{1,m} r\alpha_i\, y_i\, (x^i \cdot x)$$

Kernel Based Methods

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

- 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

Kernel Base Methods

$$\mathbf{Examples: x \in \{0,1\}^n;} \qquad \mathbf{Hypothesis: w \in R^n}$$

$$f(x) = Th_\theta(\sum_{i=1}^{n} w_i x_i(x))$$

$$\text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow w_i + 1 \text{ (if } x_i = 1) \text{ (promotion)}$$

$$\text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i - 1 \text{ (if } x_i = 1) \text{ (demotion)}$$

- Let I be the set t_1, t_2, t_3 …of monomials (conjunctions) over the feature space $x_1, x_2 \ldots x_n$.
- Then we can write a linear function over this new feature space.

$$f(x) = Th_\theta(\sum_{i \in I} w_i t_i(x))$$

$$\mathbf{Example:}\ x_1x_2x_4(11010) = 1 \quad x_3x_4(11010) = 0 \quad x_1x_2x_4(11011) = 1$$

Kernel Based Methods

$$\text{Examples}: x \in \{0,1\}^n; \qquad \text{Hypothesis}: w \in R^n$$

$$f(x) = Th_\theta(\sum_{i \in I} w_i t_i(x))$$

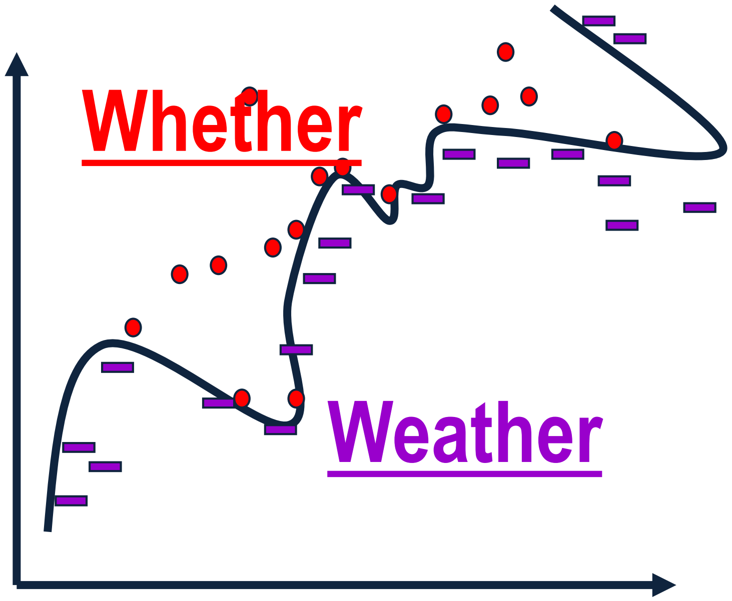
$$\text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow w_i + 1 \text{ (if } x_i = 1) \text{ (promotion)}$$

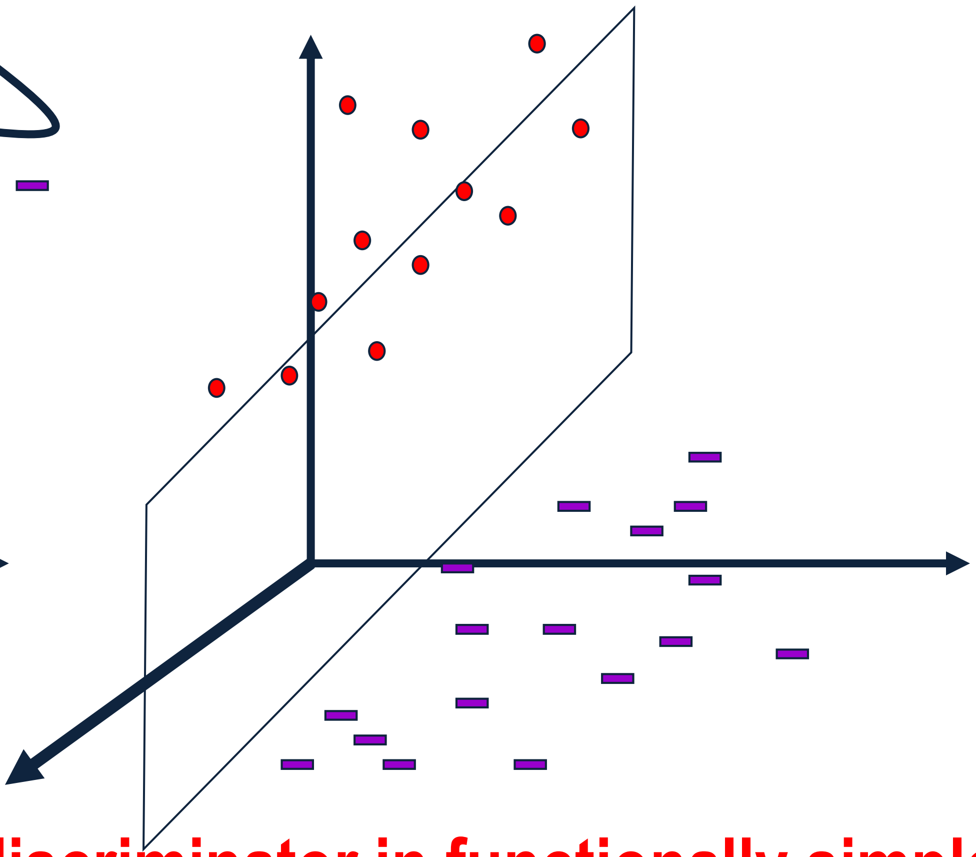
$$\text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i - 1 \text{ (if } x_i = 1) \text{ (demotion)}$$

- Great Increase in expressivity
- Can run Perceptron (and Winnow) but the convergence bound may suffer exponential growth.

- Exponential number of monomials are true in each example.
- Also, will have to keep many weights.

Embedding





New discriminator in functionally simpler

$x_1 x_2 \overline{x}_3 \vee \overline{x}_1 x_4 \overline{x}_3 \vee x_3 \overline{x}_2 x_5$ $\qquad y_1 \vee y_4 \vee y_5$

The Kernel Trick(1)

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

$$f(x) = Th_\theta(\sum_{i \in I} w_i t_i(x))$$

$$\text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow w_i + 1 \text{ (if } x_i = 1) \text{ (promotion)}$$

$$\text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i - 1 \text{ (if } x_i = 1) \text{ (demotion)}$$

Kernel Trick

- Consider the value of w used in the prediction.
- Each previous mistake, on example z, makes an additive contribution of +/-1 to w, iff t(z) = 1.
- The value of w is determined by the number of mistakes on which t() was satisfied.

The Kernel Trick(2)

$\textbf{Examples}: x \in \{0,1\}^n;$ $\textbf{Hypothesis}: w \in R^n$

$$f(x) = Th_\theta(\sum_{i \in I} w_i t_i(x))$$

If Class = 1 but $w \bullet x \le \theta$, $w_i \leftarrow w_i + 1$ (if $x_i = 1$) (promotion)

If Class = 0 but $w \bullet x \ge \theta$, $w_i \leftarrow w_i - 1$ (if $x_i = 1$) (demotion)

- P – set of examples on which we Promoted
- D – set of examples on which we Demoted
- M = P ∪ D

$$f(x) = Th_\theta(\sum_{i \in I} \left[\sum_{z \in P, t_i(z)=1} 1 - \sum_{z \in D, t_i(z)=1} 1 \right] t_i(x)) =$$

$$= Th_\theta(\sum_{i \in I} \left[\sum_{z \in M} S(z) t_i(z) t_i(x) \right]$$

The Kernel Trick(3)

$$f(x) = Th_\theta(\sum_{i\in I} w_i t_i(x))$$

- P – set of examples on which we Promoted
- D – set of examples on which we Demoted
- M = P ∪ D

$$f(x) = Th_\theta(\sum_{i\in I}\left[\sum_{z\in P, t_i(z)=1} 1 - \sum_{z\in D, t_i(z)=1} 1\right] t_i(x)) =$$

$$= Th_\theta(\sum_{i\in I}\left[\sum_{z\in M} S(z) t_i(z) t_i(x)\right]$$

- Where S(z)=1 if z ∈P and S(z) = -1 if z ∈D. Reordering:

$$f(x) = Th_\theta(\sum_{z\in M} S(z) \sum_{i\in I} t_i(z) t_i(x))$$

The Kernel Trick(4)

$$f(x) = Th_\theta(\sum_{i \in I} w_i t_i(x))$$

- S(y)=1 if y $\in$P and S(y) = -1 if y $\in$D.

$$f(x) = Th_\theta(\sum_{z \in M} S(z) \sum_{i \in I} t_i(z) t_i(x))$$

- A mistake on z contributes the value +/-1 to all monomials satisfied by z. The total contribution of z to the sum is equal to the number of monomials that satisfy both x and z.
- Define a dot product in the t-space:

$$K(x,z) = \sum_{i \in I} t_i(z) t_i(x)$$

- We get the standard notation:

$$f(x) = Th_\theta(\sum_{z \in M} S(z) K(x,z))$$

Kernel Based Methods

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

- What does this representation give us?

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

- We can view this Kernel as the distance between x,z in the t-space.

Kernel Trick

- But, K(x,z) can be measured in the original space, without explicitly writing the t-representation of x, z

$x_1x_3(001) = x_1x_3(011) = 1$

$x_1(001) = x_1(011) = 1$; $x_3(001) = x_3(011) = 1$

$\phi(001) = \phi(011) = 1$

If any other variables appears in the monomial, it's evaluation on x, z will be different.

Kernel Trick

$$f(x) = Th_\theta\left(\sum_{z\in M} S(z)K(x,z)\right) \quad K(x,z) = \sum_{i\in I} t_i(z)t_i(x)$$

- Consider the space of all 3^n monomials (allowing both positive and negative literals). Then,

$$K(x,z) = \sum_{i\in I} t_i(z)t_i(x) = 2^{same(x,z)}$$

- When same(x,z) is the number of features that have the same value for both x and z.
- We get:

$$f(x) = Th_\theta\left(\sum_{z\in M} S(z)(2^{same(x,z)})\right)$$

- Example: Take n=3; x=(001), z=(011), monomials of size 0,1,2,3
- **Proof:** let k=same(x,z); construct a "surviving" monomials by: (1) choosing to include one of these k literals with the right polarity in the monomial, or (2) choosing to not include it at all. Monomials with literals outside this set disappear.

Kernel Trick

Example

$$f(x) = \mathbf{Th}_\theta\left(\sum_{z \in M} \mathbf{S(z)K(x,z)}\right) \quad \mathbf{K(x,z)} = \sum_{i \in I} \mathbf{t_i(z) t_i(x)}$$

Kernel trick

- Take $X=\{x_1, x_2, x_3, x_4\}$
- I = The space of all 3^n monomials; $|I| = 81$
- Consider x=(1100), z=(1101)
- Write down I(x), I(z), the representation of x, z in the I space.
- Compute $I(x) \cdot I(z)$.
- Show that
- $K(x,z) = I(x) \cdot I(z) = \sum_I t_i(z)\, t_i(x) = 2^{same(x,z)} = 8$
- Try to develop another kernel, e.g., where I is the space of all conjunctions of size 3 (exactly).

Implementation: Dual Perceptron

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

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

- 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(z).
- Rather than remembering the weight vector w, remember the set M (P and D) – all those examples on which we made mistakes.

- Dual Representation

Administration

Questions

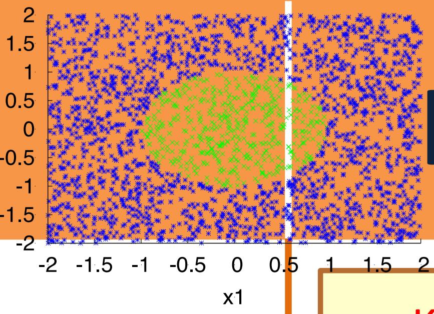
- Hw3 is out.
- Projects:
  - Some of you are thinking about the Fake News Challenge.
  - Hard, but interesting.
- Quizzes:
  - Most of you are doing it.
  - Scores are ~95%
  - Questions indicate that you are thinking about it…

Example: Polynomial Kernel

- Prediction with respect to a separating hyper planes (produced by Perceptron, SVM) can be computed as a function of dot products of feature based representation of examples.
- We want to define a dot product in a high dimensional space
- Given two examples $x = (x_1, x_2, \ldots x_n)$ and $y = (y_1, y_2, \ldots y_n)$ we want to map them to a high dimensional space [example- quadratic]:
  - $\Phi(x_1,x_2,\ldots,x_n) = (1, x_1,\ldots,x_n, x_1^2,\ldots,x_n^2, x_1x_2,\ldots,x_{n-1}x_n)$
  - $\Phi(y_1,y_2,\ldots,y_n) = (1, y_1,\ldots,y_n ,y_1^2,\ldots,y_n^2, y_1y_2,\ldots,y_{n-1}y_n)$

  Sq(2)

  and compute the dot product $A = \Phi(x)^T\Phi(y)$ [takes time]
- Instead, in the original space, compute
  - $B = k(x , y)= [1+ (x_1,x_2, \ldots x_n)^T (y_1,y_2, \ldots y_n)]^2$
- Theorem: A = B (Coefficients do not really matter)



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(.,.)? $f(x) = \mathrm{Th}_\theta(\sum_{z\in M} S(z)K(x,z))$
  - A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space. $K(x,z) = \sum_{i\in I} t_i(z)t_i(x)$
- Take the quadratic kernel: $k(x,z) = (x^Tz)^2$

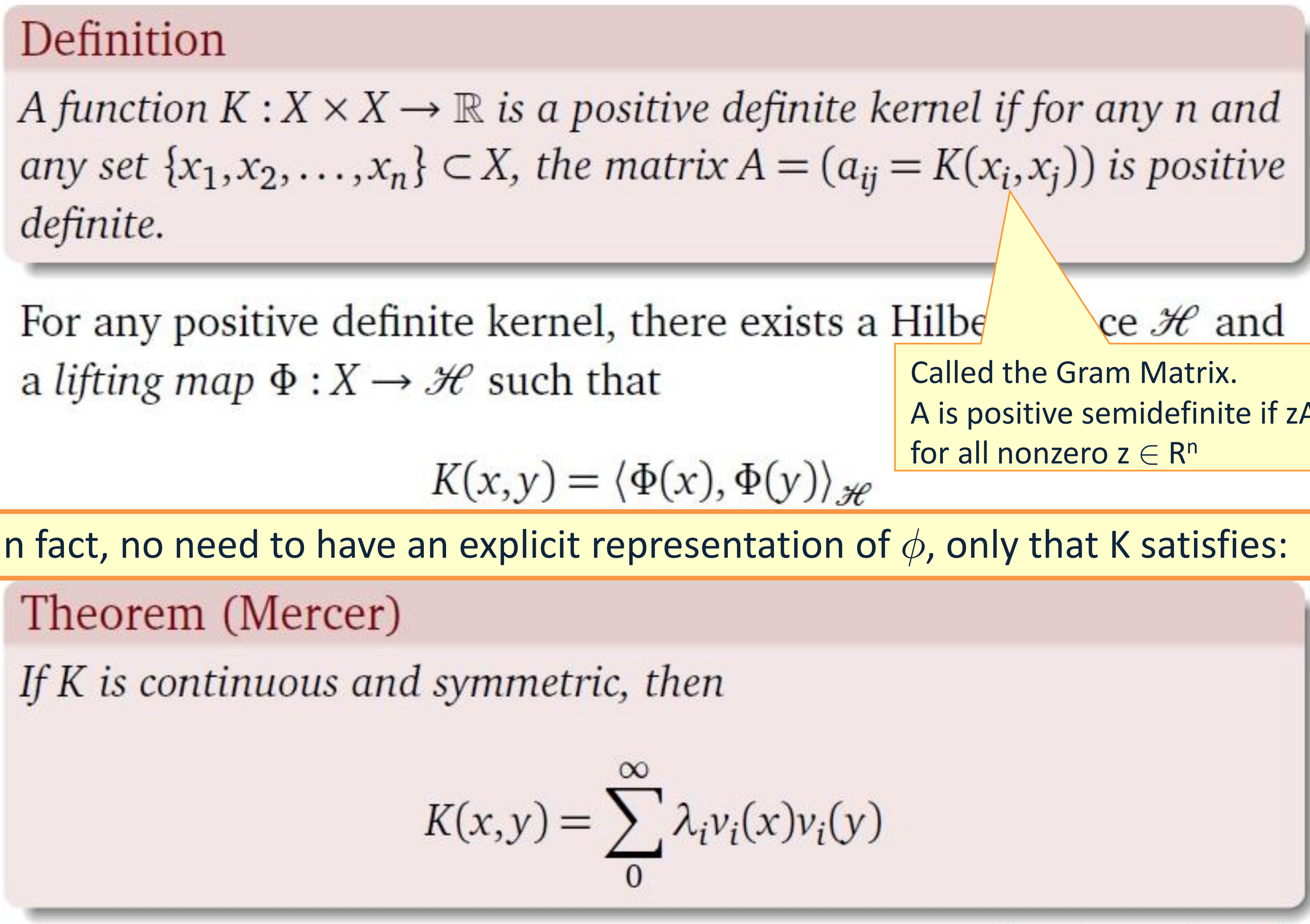
We proved that K is a valid kernel by explicitly showing that it corresponds to a dot product.

- Example: Direct construction (2 dimensional, for simplicity):
- $\mathbf{K(x,z) = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2}$
- $\mathbf{= (x_1^2, \mathrm{sqrt}\{2\}\, x_1x_2, x_2^2)\ (z_1^2, \mathrm{sqrt}\{2\}\, z_1z_2, z_2^2)}$
- $\mathbf{= \Phi(x)^T\, \Phi(z) \rightarrow}$ **A dot product in an expanded space.**
- It is not necessary to explicitly show the feature function ϕ.
- General condition: construct the kernel matrix $\{k(x_i, z_j)\}$; check that it's positive semi definite.

The Kernel Matrix

- The Gram matrix of a set of *n* vectors S = {$\mathbf{x}_1 \ldots \mathbf{x}_n$} is the $n \times n$ matrix **G** with $\mathbf{G}_{ij} = \mathbf{x}_i \mathbf{x}_j$
  - The kernel matrix is the Gram matrix of {$\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$}
  - (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 $\phi(\mathbf{x}_i)$

Kernels – General Conditions



Definition

A function $K : X \times X \to \mathbb{R}$ is a positive definite kernel if for any n and any set $\{x_1, x_2, \ldots, x_n\} \subset X$, the matrix $A = (a_{ij} = K(x_i, x_j))$ is positive definite.

For any positive definite kernel, there exists a Hilbe ce $\mathscr{H}$ and a *lifting map* $\Phi : X \to \mathscr{H}$ such that

$$K(x,y) = \langle \Phi(x), \Phi(y) \rangle_{\mathscr{H}}$$

Called the Gram Matrix.
A is positive semidefinite if $zAz^T > 0$ for all nonzero $z \in R^n$

In fact, no need to have an explicit representation of ϕ, only that K satisfies:

Theorem (Mercer)

If K is continuous and symmetric, then

$$K(x,y) = \sum_{0}^{\infty} \lambda_i v_i(x) v_i(y)$$

Polynomial kernels

- Linear kernel: $k(\mathbf{x}, \mathbf{z}) = \mathbf{xz}$
- Polynomial kernel of degree *d*: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz})^d$
  (only *d*th-order interactions)
- Polynomial kernel up to degree *d*: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz} + c)^d$ (c>0)
  (all interactions of order *d* or lower)

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 $k(\mathbf{x}, \mathbf{x'})$:
    $k'(\mathbf{x}, \mathbf{x'}) = \exp(k(\mathbf{x}, \mathbf{x'}))$

Constructing New Kernels (2)

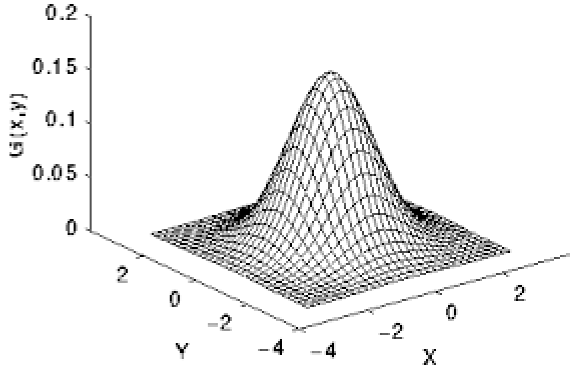
- You can construct $k'(\mathbf{x}, \mathbf{x'})$ from $k_1(\mathbf{x}, \mathbf{x'})$, $k_2(\mathbf{x}, \mathbf{x'})$ by:
  - Adding $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k'(\mathbf{x}, \mathbf{x'}) = k_1(\mathbf{x}, \mathbf{x'}) + k_2(\mathbf{x}, \mathbf{x'})$
  - Multiplying $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k'(\mathbf{x}, \mathbf{x'}) = k_1(\mathbf{x}, \mathbf{x'})k_2(\mathbf{x}, \mathbf{x'})$
- Also:
  - If $\phi(\mathbf{x}) \in R^m$ and $k_m(\mathbf{z}, \mathbf{z'})$ a valid kernel in R^m,
    $k(\mathbf{x}, \mathbf{x'}) = k_m(\phi(\mathbf{x}), \phi(\mathbf{x'}))$ is also a valid kernel
  - If $\mathbf{A}$ is a symmetric positive semi-definite matrix,
    $k(\mathbf{x}, \mathbf{x'}) = \mathbf{xAx'}$ is also a valid kernel
- In all cases, it is easy to prove these directly by construction.

Gaussian Kernel

(aka radial basis function kernel)

- $k(x, z) = \exp(-(x - z)^{2/c})$
  - $(x - z)^2$: squared Euclidean distance between **x** and **z**
  - $c = \sigma^2$: a free parameter
  - very small c: K ≈ identity matrix (every item is different)
  - very large c: K ≈ unit matrix (all items are the same)

  - k(**x**, **z**) ≈ 1 when **x**, **z** close
  - k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar



Gaussian Kernel

- $k(x, z) = \exp(-(x - z)^{2/c})$
- Is this a kernel?
- $k(\mathbf{x}, \mathbf{z}) = \exp(-(\mathbf{x} - \mathbf{z})^2/2\sigma^2)$

  $= \exp(-(\mathbf{xx} + \mathbf{zz} - 2\mathbf{xz})/2\sigma^2)$

  $= \exp(-\mathbf{xx}/2\sigma^2) \exp(\mathbf{xz}/\sigma^2) \exp(-\mathbf{zz}/2\sigma^2)$

  $= f(\mathbf{x}) \exp(\mathbf{xz}/\sigma^2) f(\mathbf{z})$
- $\exp(\mathbf{xz}/\sigma^2)$ is a valid kernel:
  - $\mathbf{xz}$ is the linear kernel;
  - we can multiply kernels by constants ($1/\sigma^2$)
  - we can exponentiate kernels

Unlike the discrete kernels discussed earlier, here you cannot easily explicitly blow up the feature space to get an identical representation.

Summary – Kernel Based Methods

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

- 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. [Khardon,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, $t_1 << t_2$, 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)

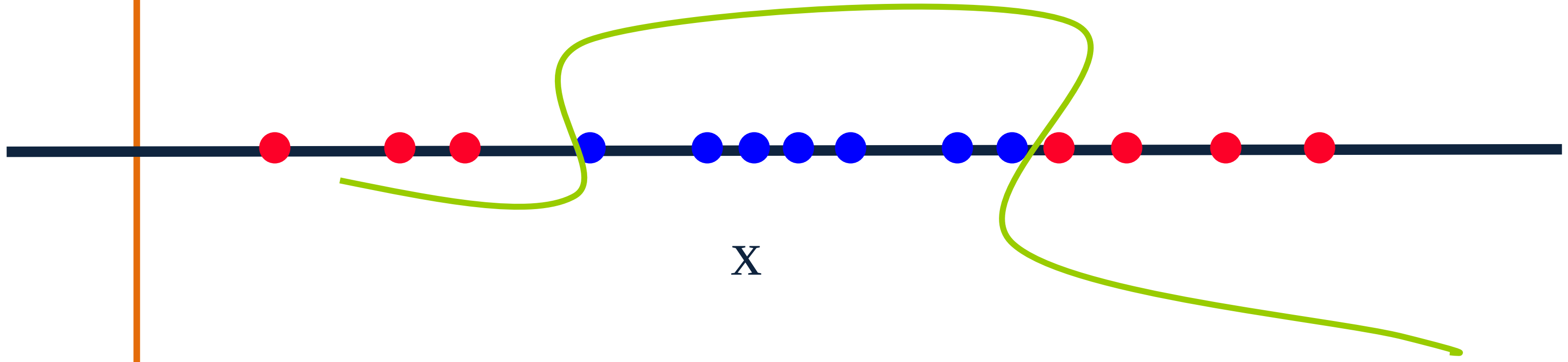
- Given: A linearly separable set of points $S=\{x_1,\ldots x_n\} \in R^n$ with separator $w \in R^n$
- Embed S into a higher dimensional space $n'>n$, by adding zero-mean random noise e to the additional dimensions.
- Then $w' \cdot x' = (w,0) \cdot (x,e) = w \cdot x$
- So $w' \in R^{n'}$ still separates 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 smaller, which implies generalization suffers.
- Intuition: adding a lot of noisy/irrelevant features cannot help

Conclusion- Kernels

- The use of Kernels to learn in the dual space is an important idea
  - Different kernels may expand/restrict the hypothesis space in useful ways.
  - Need to know the benefits and hazards
- To justify these methods we must embed in a space much larger than the training set size.
  - Can affect generalization
- Expressive structures in the input data could give rise to specific kernels, designed to exploit these structures.
  - E.g., people have developed kernels over parse trees: corresponds to features that are sub-trees.
  - It is always possible to trade these with explicitly generated features, but it might help one's thinking about appropriate features.

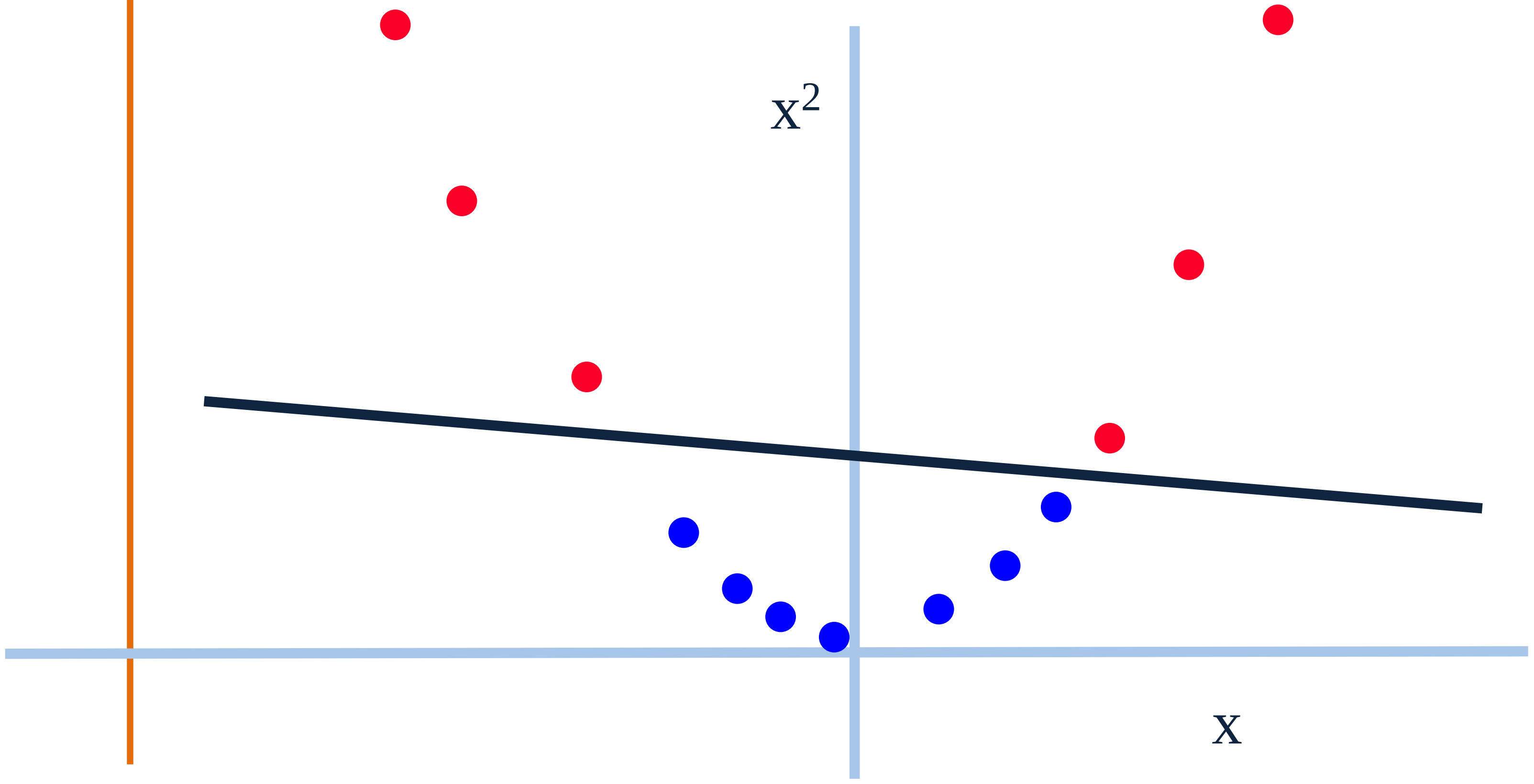
Functions Can be Made Linear

- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions



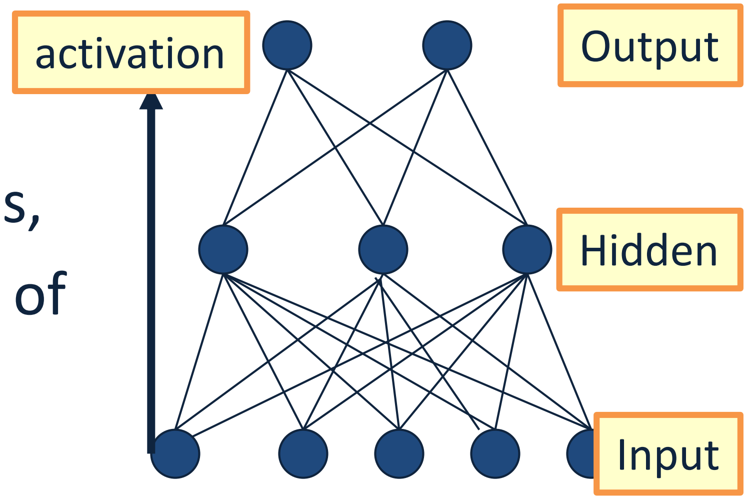
Blown Up Feature Space

- Data are separable in <x, x^2> space



Multi-Layer Neural Network

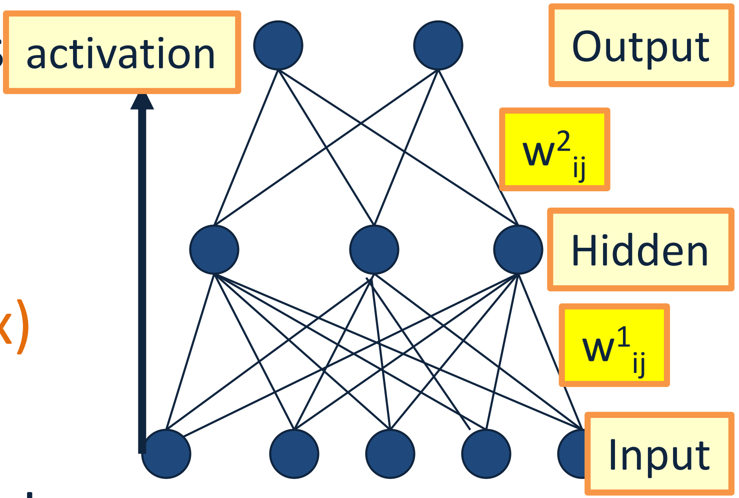
- Multi-layer network were designed to overcome the computational (expressivity) limitation of a single threshold element.
- The idea is to stack several layers of threshold elements, each layer using the output of the previous layer as input.



- Multi-layer networks can represent arbitrary functions, but building effective learning methods for such network was [thought to be] difficult.

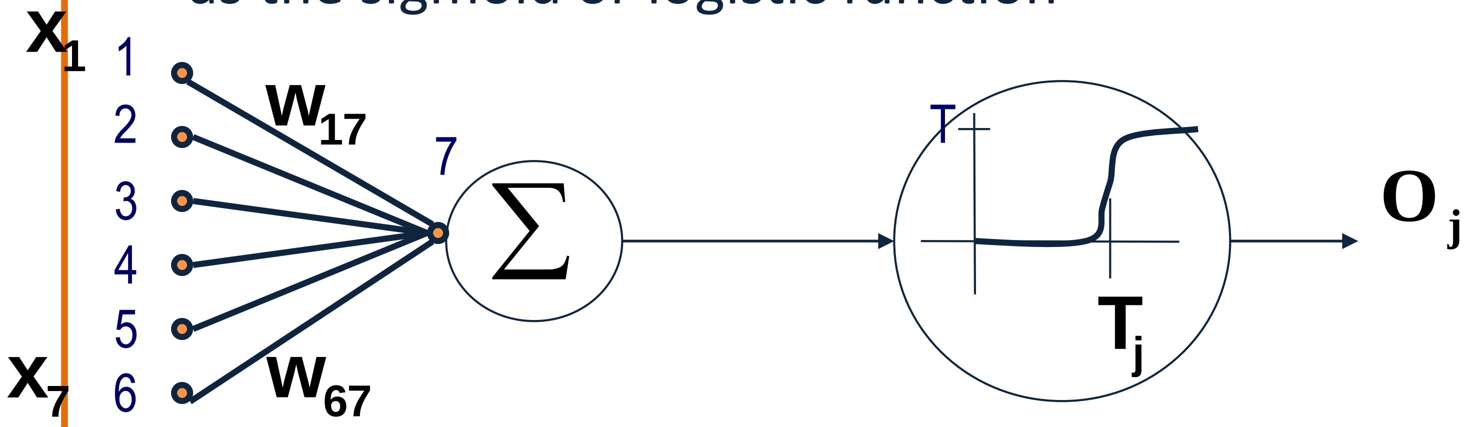
Basic Units

- Linear Unit: Multiple layers of linear functions $o_j = w \cdot x$ produce linear functions. We want to represent nonlinear functions
- Need to do it in a way that facilitates learning
- Threshold units: $o_j = \mathrm{sgn}(w \cdot x)$ are not differentiable, hence unsuitable for gradient descent.
- The key idea was to notice that the discontinuity of the threshold element can be represents by a smooth non-linear approximation: $o_j = [1+ \exp\{-w \cdot x\}]^{-1}$
- (Rumelhart, Hinton, Williiam, 1986), (Linnainmaa, 1970) , see: http://people.idsia.ch/~juergen/who-invented-backpropagation.html)



Model Neuron (Logistic)

- Us a non-linear, differentiable output function such as the sigmoid or logistic function

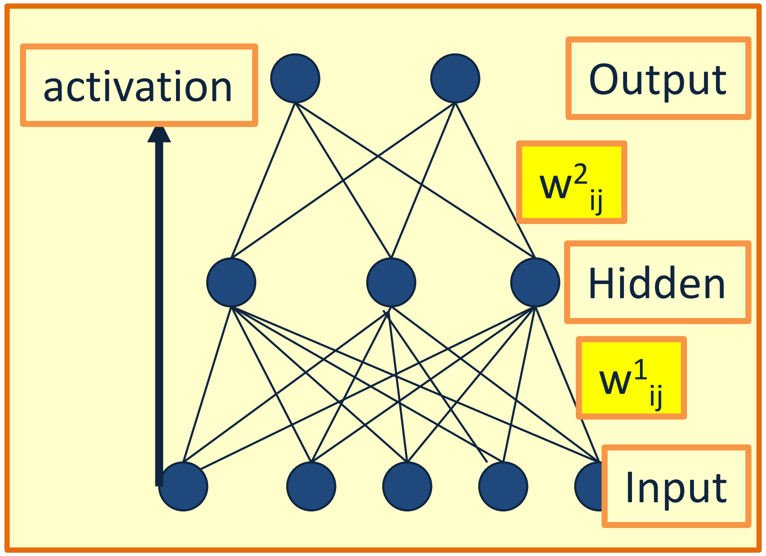


- Net input to a unit is defined as: $\mathbf{net}_j = \sum \mathbf{w}_{ij} \bullet \mathbf{x}_i$
- Output of a unit is defined as:

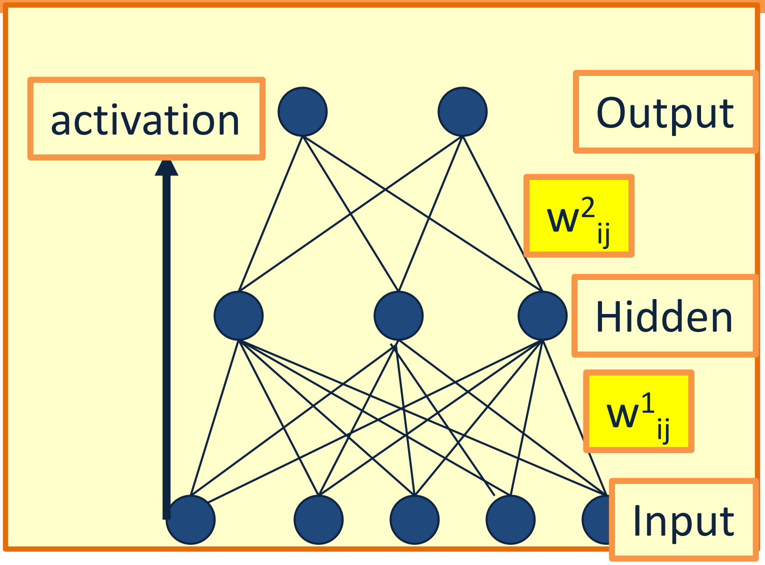
$$\mathbf{O}_j = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-(\mathbf{net}_j - \mathbf{T}_j)}}$$

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 (multiple options) 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 .
- This results in an algorithm called back-propagation.