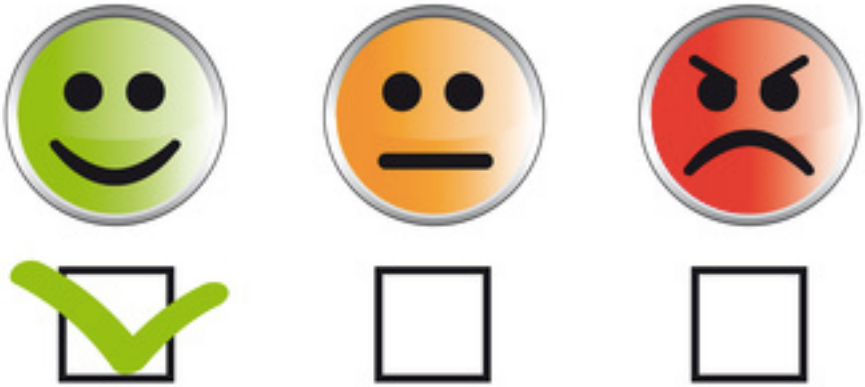
ML4NLP
Learning as Optimization



Dan Goldwasser

Purdue University

dgoldwas@purdue.edu

Reading for next time

Using classification to understand human interactions

speed dating *noun*

Menu

speed dating [uncountable]

an event at which you meet and talk to a lot of different people for only a few minutes at at a time. People do this in order to try to meet someone and have a romantic relationship.

Link to reading material in Piazza, under resources

Slide taken from D. Jurafsky

Classification

- *A fundamental machine learning tool*
  - Widely applicable in NLP
- **Supervised learning**: Learner is given a collection of labeled documents
  - Emails: Spam/not spam; Reviews: Pos/Neg
- Build a **function** mapping documents to labels
  - Key property: ***Generalization***
    - function should work well on new data

Sentiment Analysis

Dude, I just watched this horror flick! Selling points: nightmares scenes, torture scenes, terrible monsters that was so bad a##!

Don't buy the popcorn it was terrible, the monsters selling it must have wanted to torture me, it was so bad it gave me nightmares!

What should your learning algorithm look at?

Deceptive Reviews

Which of these two hotel reviews is deceptive opinion spam?

My husband and I stayed at the James Chicago Hotel for our anniversary. This place is fantastic! We knew as soon as we arrived we made the right choice! The rooms are BEAUTIFUL and the staff very attentive and wonderful!! The area of the hotel is great, since I love to shop I couldn't ask for more!! We will definatly be back to Chicago and we will for sure be back to the James Chicago.

I have stayed at many hotels traveling for both business and pleasure and I can honestly stay that The James is tops. The service at the hotel is first class. The rooms are modern and very comfortable. The location is perfect within walking distance to all of the great sights and restaurants. Highly recommend to both business travellers and couples.

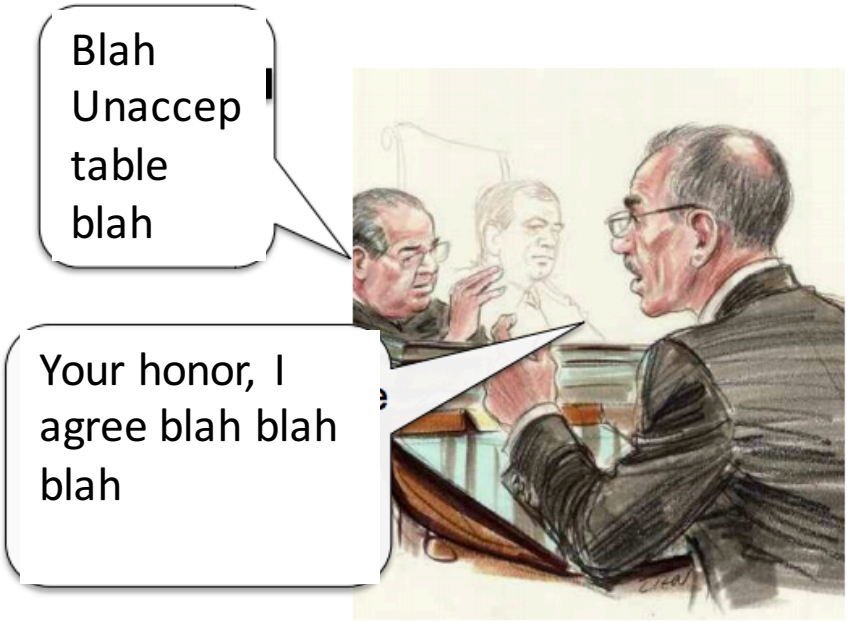
What should your learning algorithm look at?

Finding Deceptive Opinion Spam by Any Stretch of the Imagination. Ott etal. ACL 2011

Power Relations

Can subtle, *domain-independent* linguistic cues reveal (situational) power?

Who's in charge?



What should your learning algorithm look at?

Echoes of Power: Language Effects and Power Differences in Social Interaction. Danescu-Niculescu-Mizil et-al . WWW 2012.

Naive Bayes

$$V_{MAP} = \text{argmax}_v \, P(x_1, x_2, \ldots, x_n \mid v)P(v)$$

$$\begin{aligned}
&P(x_1, x_2, \ldots, x_n \mid v_j) = \\
&= P(x_1 \mid x_2, \ldots, x_n, v_j)P(x_2, \ldots, x_n \mid v_j) \\
&= P(x_1 \mid x_2, \ldots, x_n, v_j)P(x_2 \mid x_3, \ldots, x_n, v_j)P(x_3, \ldots, x_n \mid v_j) \\
&= \ldots\ldots\ldots \\
&= P(x_1 \mid x_2, \ldots, x_n, v_j)P(x_2 \mid x_3, \ldots, x_n, v_j)P(x_3 \mid x_4, \ldots, x_n, v_j)\ldots P(x_n \mid v_j)
\end{aligned}$$

Assumption: feature values are independent given the target value

$$= \prod_{i=1}^{n} P(x_i \mid v_j)$$

Naïve Bayes: *practical stuff*

- **Underflow prevention**
  - Multiplying probabilities will result in floating point underflow (round to zero)
  - Summing log probabilities will solve the problem:

    NB Decision: $\text{argmax}_{c\_j} \log P(c_j) + \sum_i \log P(x_i \mid c_j)$

- **Feature "tweaking"**
  - "Pure BoW" is often too naïve
  - Stemming/normalization (hate, hated, hating, hates)
  - Phrase extraction (e.g., negation, scientific/numeric expr.)
  - Up weighting – increase the counts of "important" words

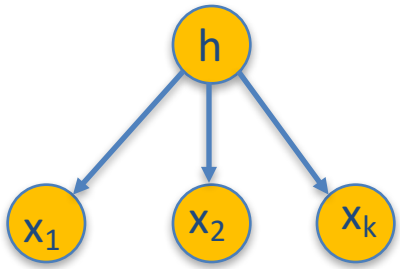
Linear Classifiers

- **Linear threshold functions**
  - Associate a weight (w_i) with each feature (x_i)
  - **Prediction**: $\text{sign}(b + w^T x) = \text{sign}\ (b + \Sigma\ w_i\ x_i)$
    - $b + w^T x \geq 0$ predict y=1
    - Otherwise, predict y=-1
- ***NB is a linear threshold function***
  - Weight vector (w) is assigned by estimating the label probability, given the model independency assumptions
- *Linear threshold functions are a very popular representation!*

- **Generative Vs. Discriminative models**
  - Generative models represent the joint probability P(x,y), while discriminative models represent P(y|x) directly

Generative vs. Discriminative

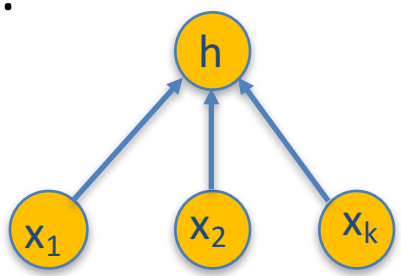
- Language models and NB are examples of ***generative models***
- Generative models capture the ***joint probability*** of the input and outputs **`P(x,y)`**
  - Most of the early work in statistical NLP is generative: *Language models, NB, HMM, Bayesian Networks, PCFG, etc.*
- *We think about generative models as the hidden variables generating the observed data*
- *Super-easy training = counting!*



Naïve Bayes

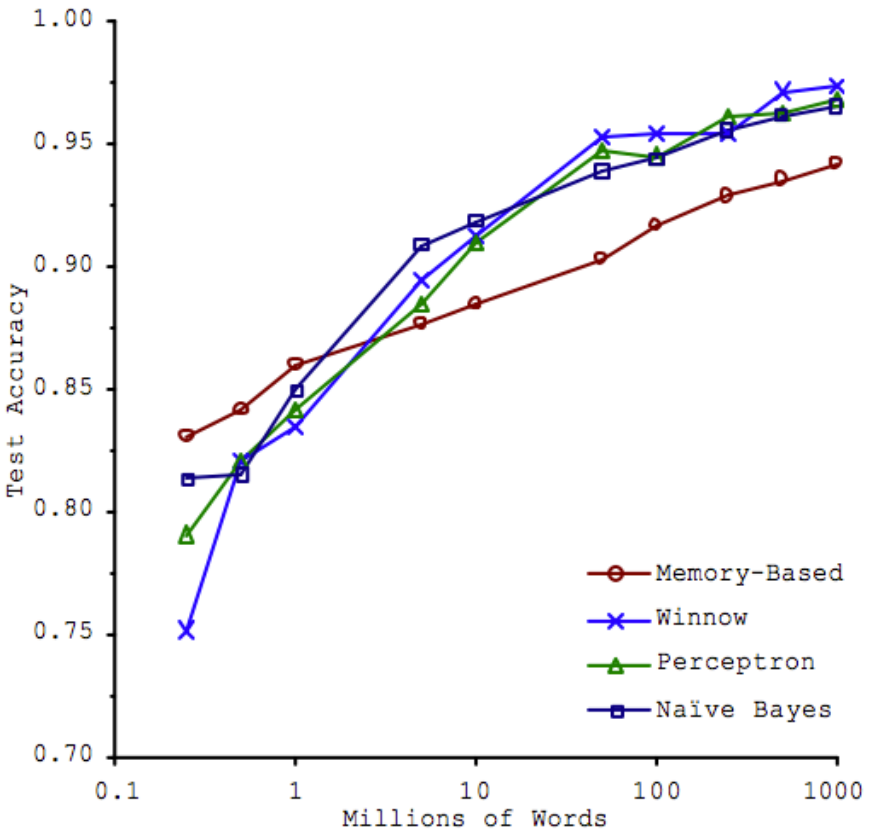
Generative vs. Discriminative

- On the other hand..
- We don't care about the joint probability, we can about the conditional probability – $P(y \mid x)$
- Conditional or **discriminative** models characterize the decision boundary directly (=conditional probability).
  - Work really well in practice, easy to incorporate arbitrary features,..
  - SVM, perceptron, Logistic Regression, CRF, …
- Training is harder (we'll see..)



Logistic regression

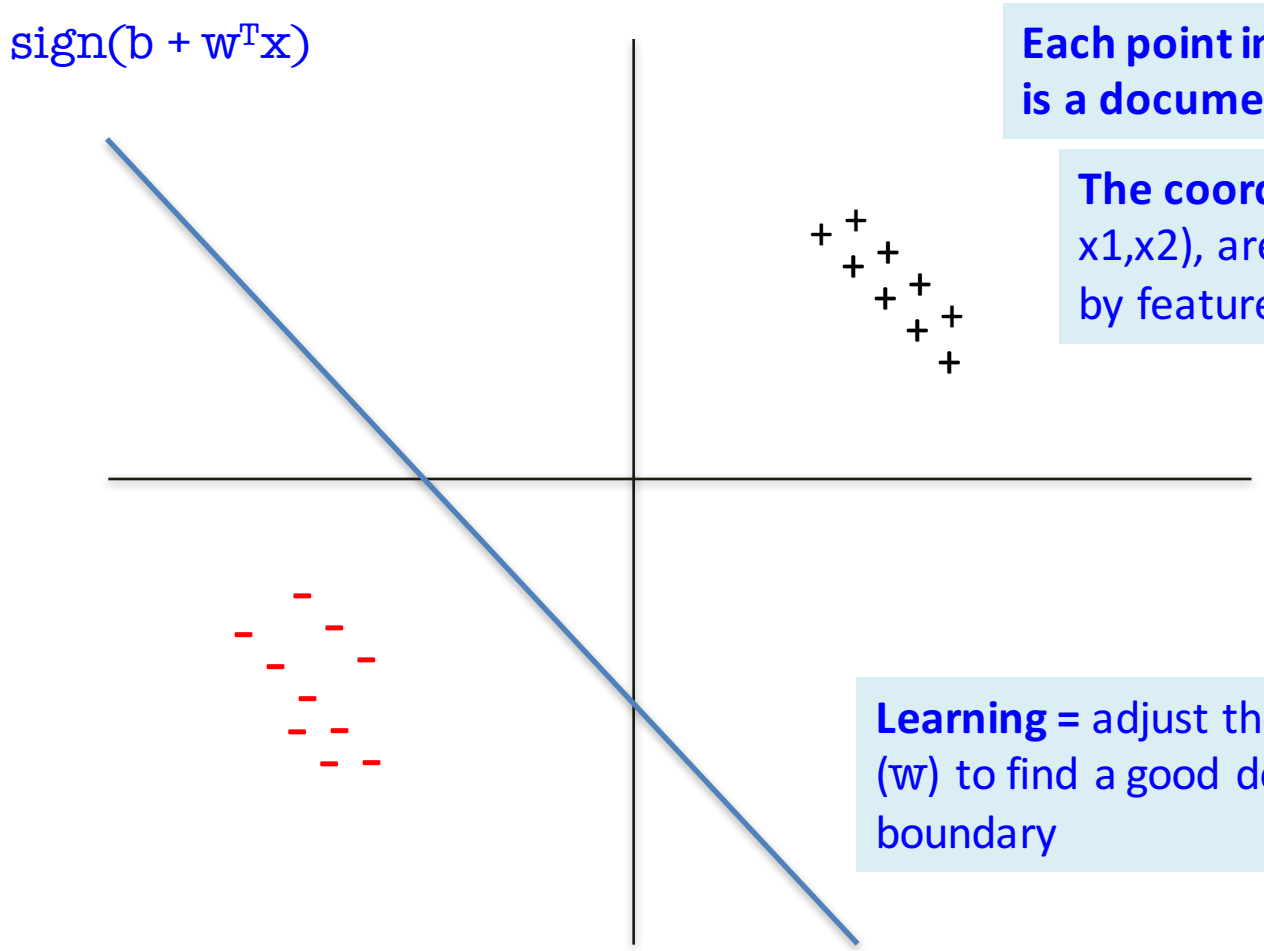
Practical Example



Task: **context sensitive spelling**
{principle, principal},{weather,whether}.

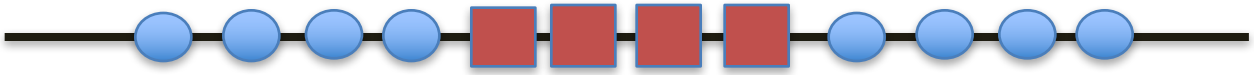
Source: Scaling to very very large corpora for natural language disambiguation
Michele Banko, Eric Brill. Microsoft Research, Redmond, WA. 2001.

Linear Classifiers

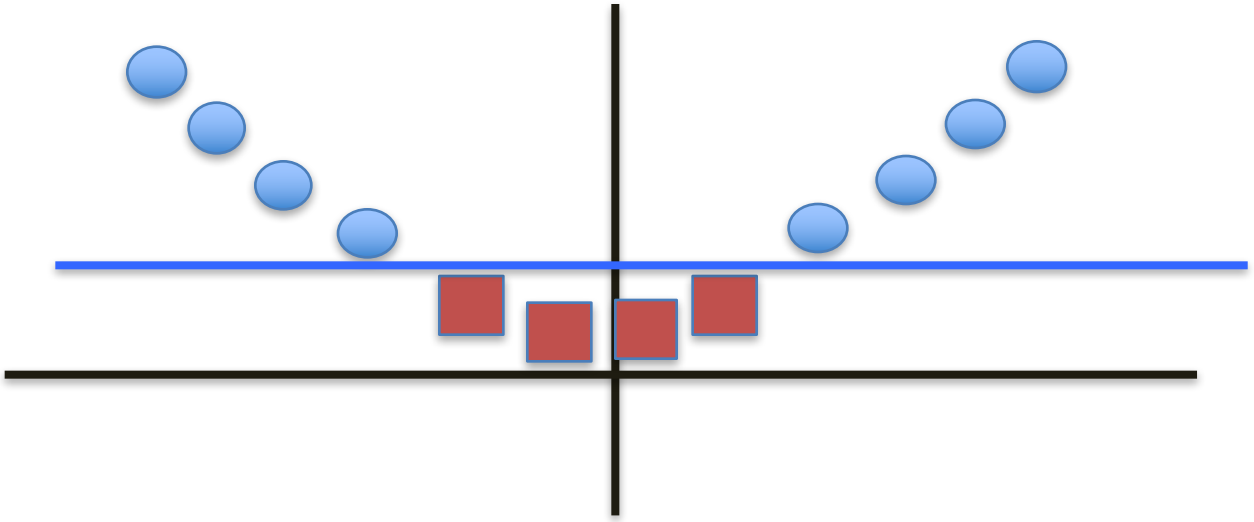


Expressivity

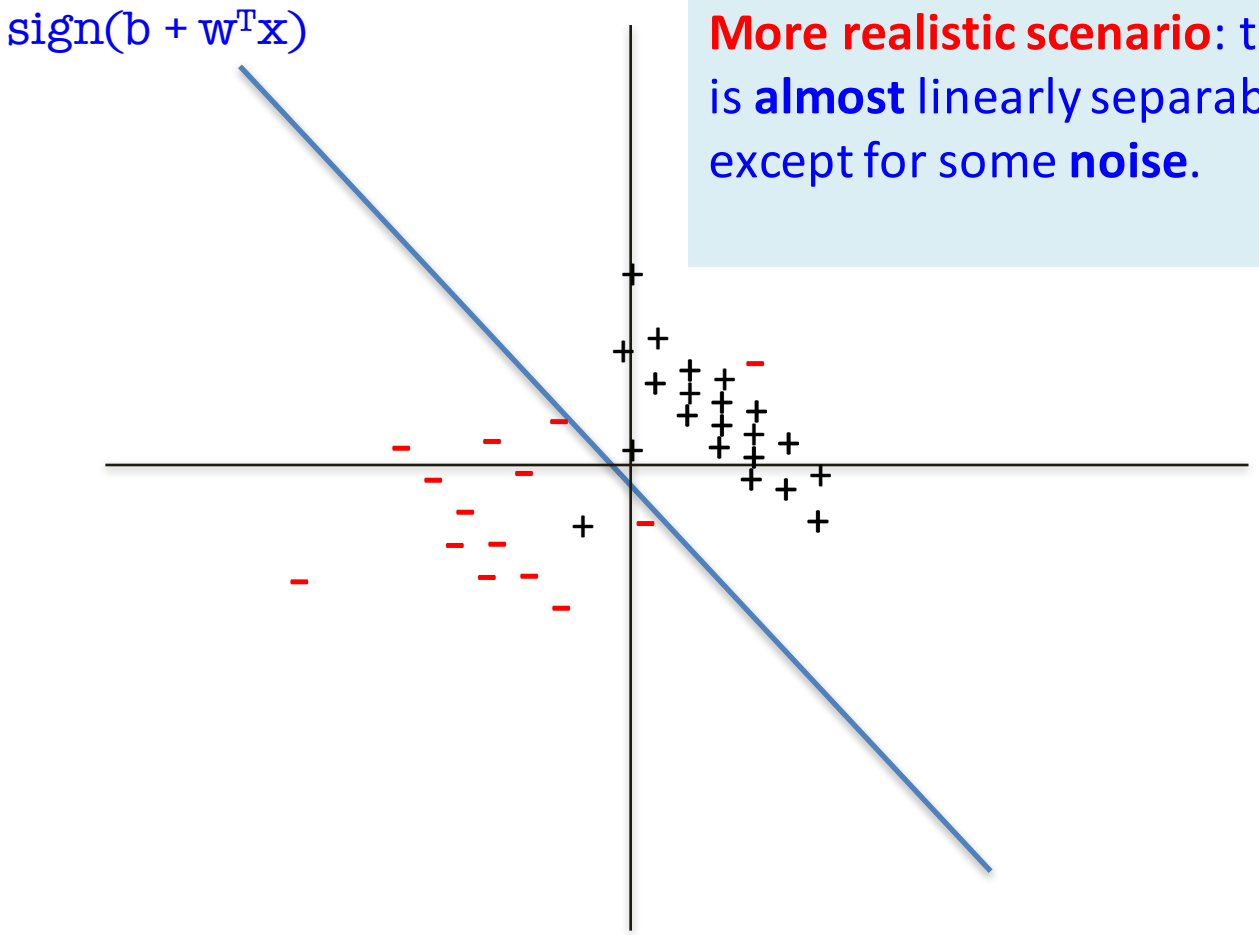
By transforming the feature space these functions can be made linear



Represent each point in 2D as (x,x^2)



Expressivity



More realistic scenario: the data is **almost** linearly separable, except for some **noise**.

Features

- So far we have discussed BoW representation
  - *In fact, you can use a very rich representation*
- **Broader definition**
  - Functions mapping attributes of the input to a Boolean/categorical/numeric value

$$\phi_1(x) = \begin{cases} 1 & x_1 \ is \ capitalized \\ 0 & otherwise \end{cases}$$

$$\phi_k(x) = \begin{cases} 1 & x \ contains \ "good" \ more \ than \ twice \\ 0 & otherwise \end{cases}$$

- **Question**: *assume that you have a lexicon, containing positive and negative sentiment words. How can you use it to improve over BoW?*

Perceptron

- One of the earliest learning algorithms
  - Introduced by Rosenblatt 1958 to model neural learning
- **Goal**: directly search for a separating hyperplane
  - If one exists, perceptron will find it
  - If not, …
- **Online** algorithm
  - Considers one example at a time (NB – looks at entire data)
- **Error driven** algorithm
  - Updates the weights only when a mistake is made

Perceptron

- We learn f:X→ {-1,+1} represented as f =sgn{w•x)
- Where X= $\{0,1\}^n$ or X= R^n and w ^{2} 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' = sgn{w•x)

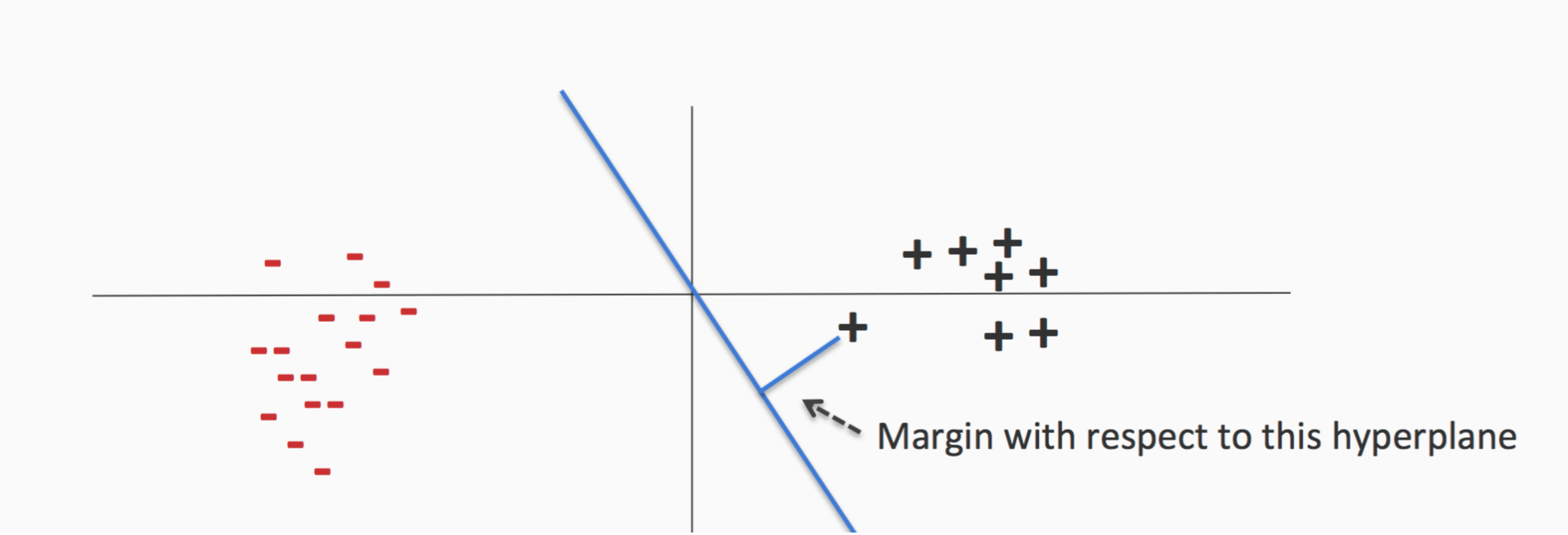
   b. If ***y'≠y***, **update** the weight vector:

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

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

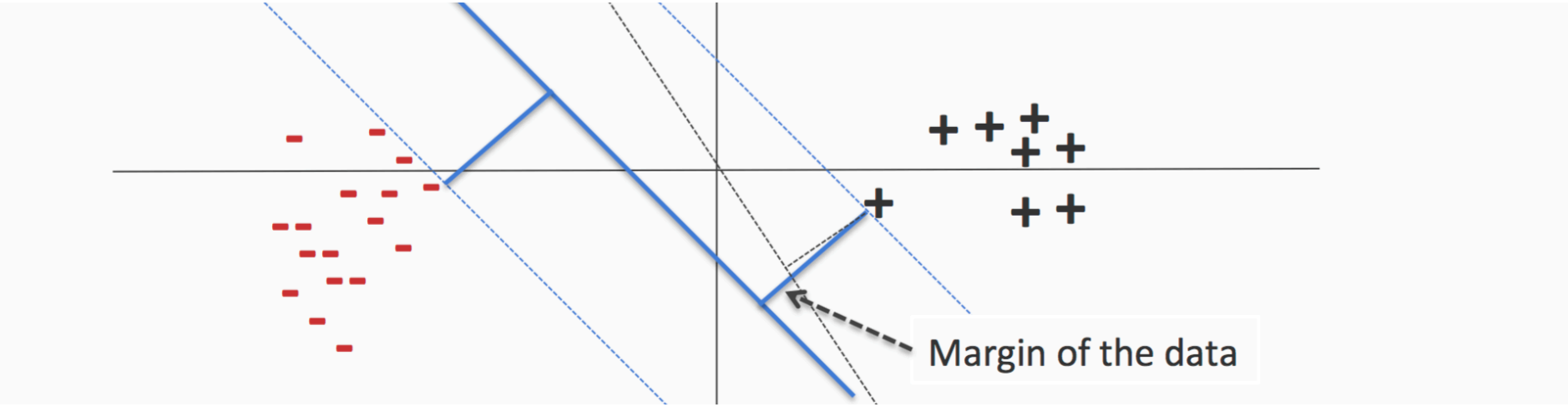
Margin

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



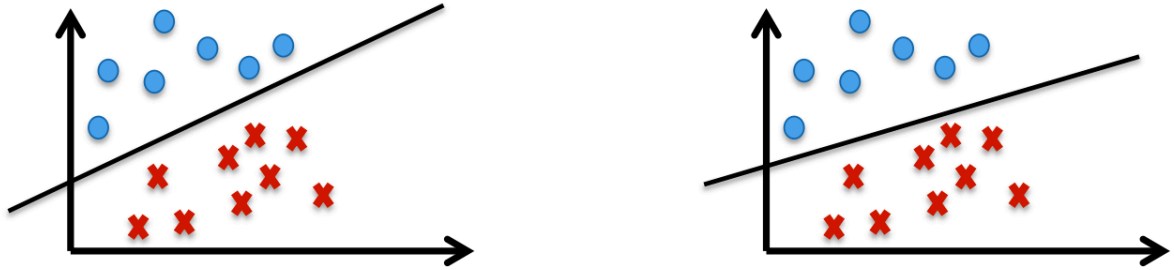
Margin

- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.
- The margin of a data set (γ) is the maximum margin possible for that dataset using any weight vector.



Learning using the Perceptron Algorithm

- Perceptron guarantee: find a linear separator (if the data is separable)



- There could be many models consistent with the training data
  - How Did the perceptron algorithm deal with this problem?
- Trading some training error for better generalization
  - This problem is aggravated when we explode the feature space

Perceptron

How would you tweak the Perceptron to make it - *(1) realistic? (2) "better"?*

- We learn f:X→ {-1,+1} represented as f =sgn{w•x)
- Where X= $\{0,1\}^n$ or X= R^n and w ^{2} 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' = sgn{w•x)

   b. If ***y'≠y***, **update** the weight vector:

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

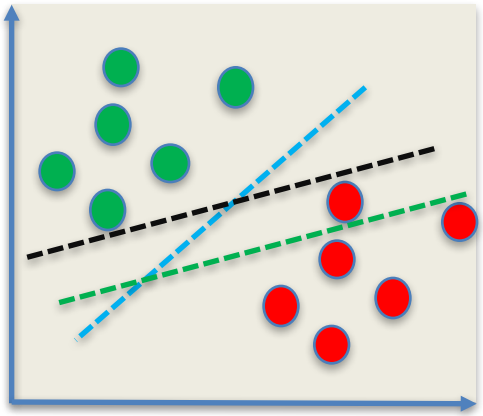
   Otherwise, if **y'=y**, leave weights unchanged.

Learning as Optimization

- **Discriminative Linear classifiers**
  - So far we looked **perceptron**
  - Combines model (*linear representation*) with algorithm (update rule)
  - Let's try to **abstract** – we want to find a linear function that performs 'the best' on the data
    - What are good properties of this classifier?
    - Want to explicitly control for error + "simplicity"
  - **How can we discuss these terms separately from a specific algorithm?**
    - Search space of all possible linear functions
    - **Find a specific function that has certain properties..**

Learning as Optimization

- Instead we can think about learning as optimizing an objective function (e.g., ***minimize mistakes***)



- We can incorporate other considerations by modifying the objective function (*regularization*)
- Sometime referred to as *structural risk minimization*

Loss Function

- To formalize performance let's define a **loss function**: $loss(y, \hat{y})$
  - Where $\hat{y}$ is the gold label
- The loss function measures the error on a single instance
  - Specific definition depends on the learning task

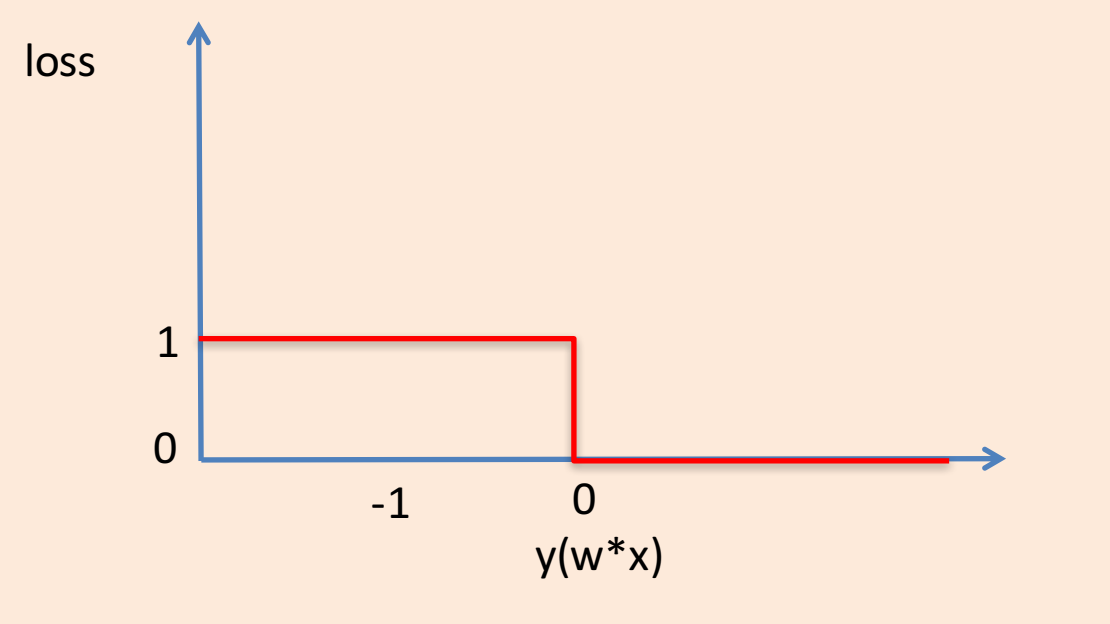
Regression

$$loss(y, \hat{y}) = (y - \hat{y})^2$$

Binary classification

$$loss(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & otherwise \end{cases}$$

0-1 Loss



$$loss(y, f(x)) = 0 \quad \text{iff} \quad y = \hat{y}$$
$$loss(y, f(x)) = 1 \quad \text{iff} \quad y \neq \hat{y}$$

$$loss(y \cdot f(x)) = 0 \quad \text{iff} \quad y \cdot f(x) > 0 \quad (\text{correct})$$
$$loss(y \cdot f(x)) = 1 \quad \text{iff} \quad y \cdot f(x) < 0 \quad (\text{Misclassified})$$

The **Empirical** Risk of f(x)

- The empirical risk of a classifier on a dataset D is its average loss on the items in d

$$R_D(f) = \frac{1}{D} \sum_{D} loss(y_i, f(x_i))$$

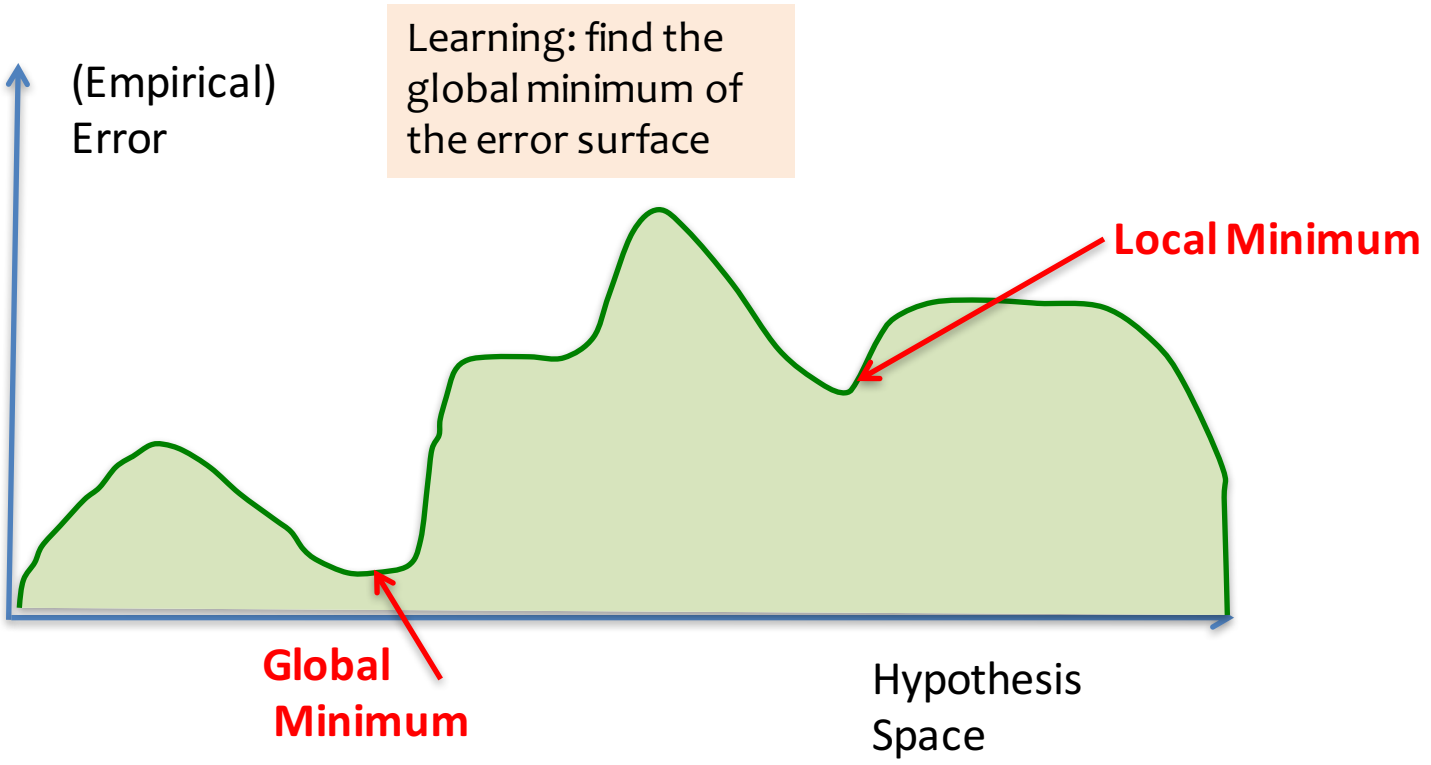
- **Realistic learning objective**: find an *f* that minimizes the empirical risk

Empirical Risk Minimization

- Learning: Given a training dataset D, return the classifier f(x) that minimizes the empirical risk
- *Given this definition we can view learning as a* ***minimization problem***
  - The objective function to minimize (empirical risk) is defined with respect to a specific loss function
  - Our minimization procedure (aka learning) will be influenced by the choice of loss function
    - **Some are easier to minimize than other!**

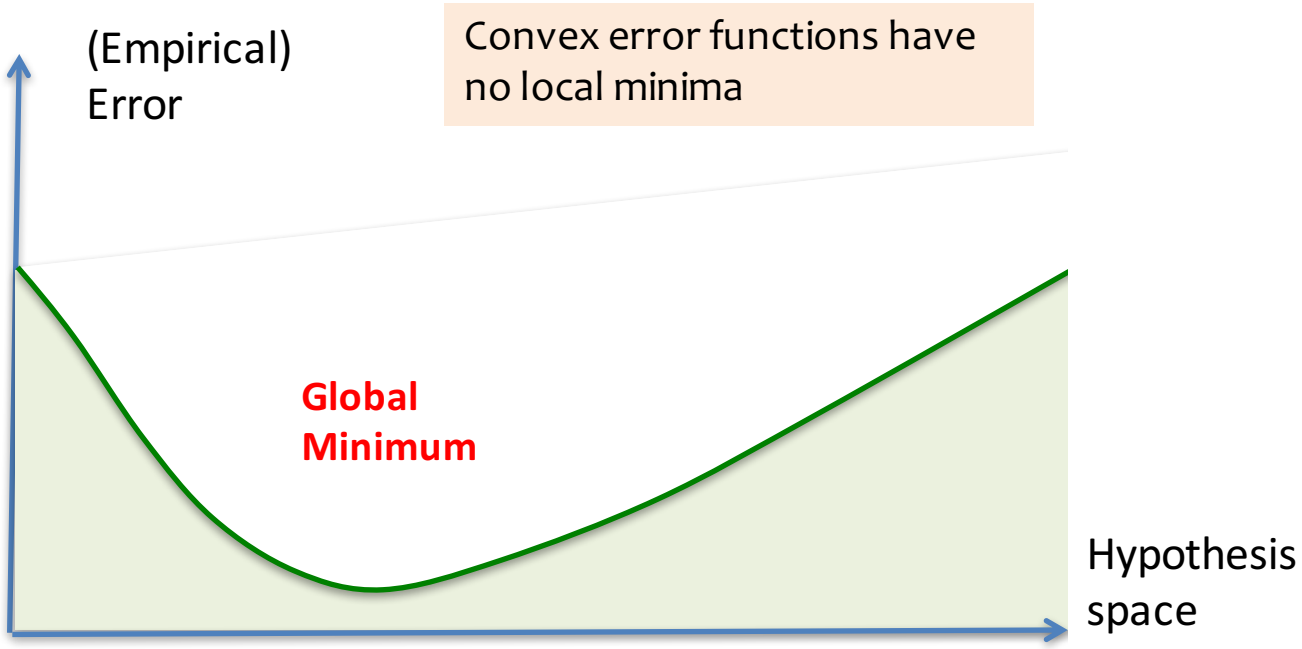
Error Surface

- **Linear classifiers**: hypothesis space parameterized by w
- *Error/Loss/Risk* are all functions of w



Convex Error Surfaces

- **Convex functions** have a single minimum point
  - Local minimum = global minimum
  - *Easier to optimize*

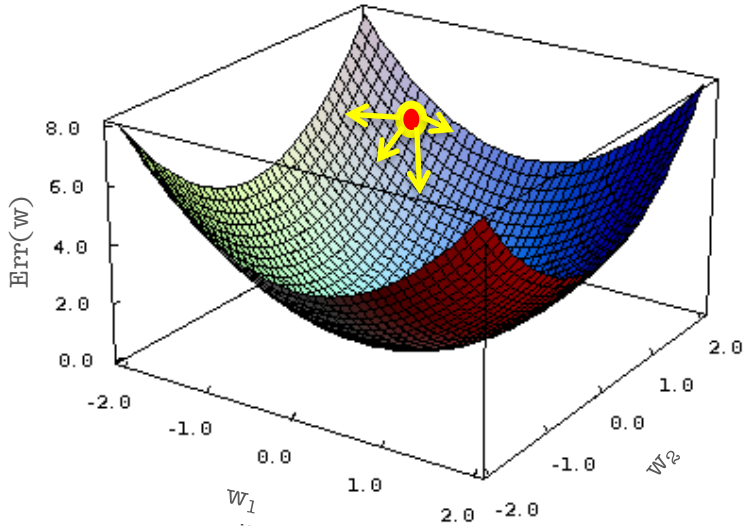


Error Surface for Squared Loss

We add the ½ for convenience

$$Err(\mathrm{w}) = \frac{1}{2}\sum_{d \in D}(\hat{y}_d - y_d)^2$$

$$y = \mathrm{w}^T x$$



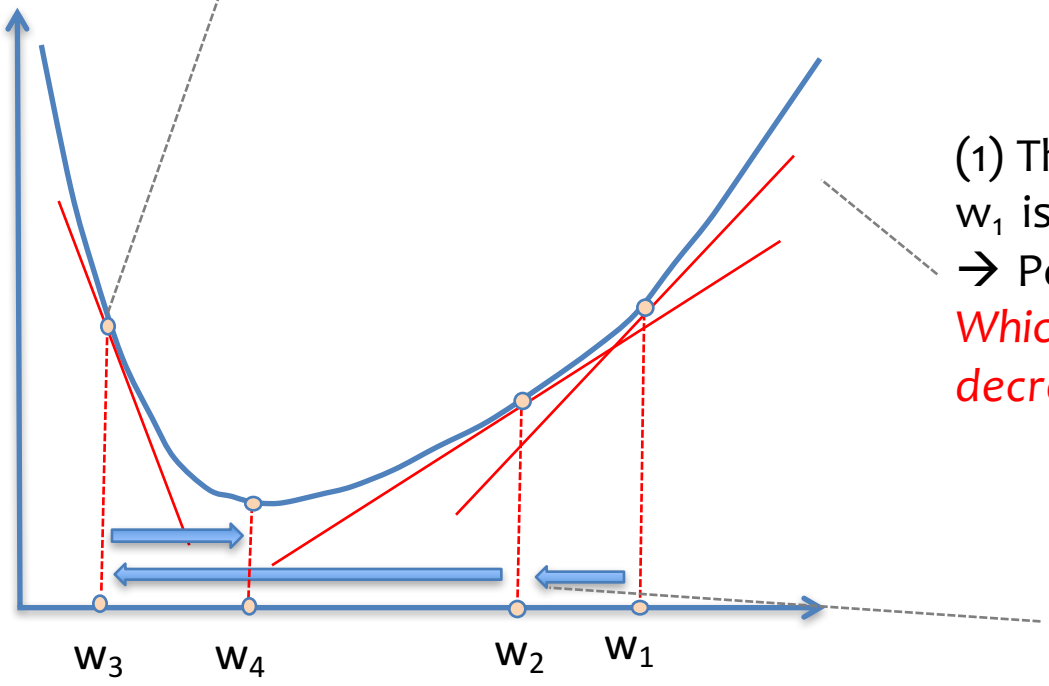
Since $\hat{y}$ is a constant (for a given dataset), the Error function is a *quadratic function* of W (paraboloid)

➔ Squared Loss function is convex!

How can we find the global minimum?

Gradient Descent Intuition

(3) We also need to determine the step size (aka learning rate).
What happens if we overshoot?



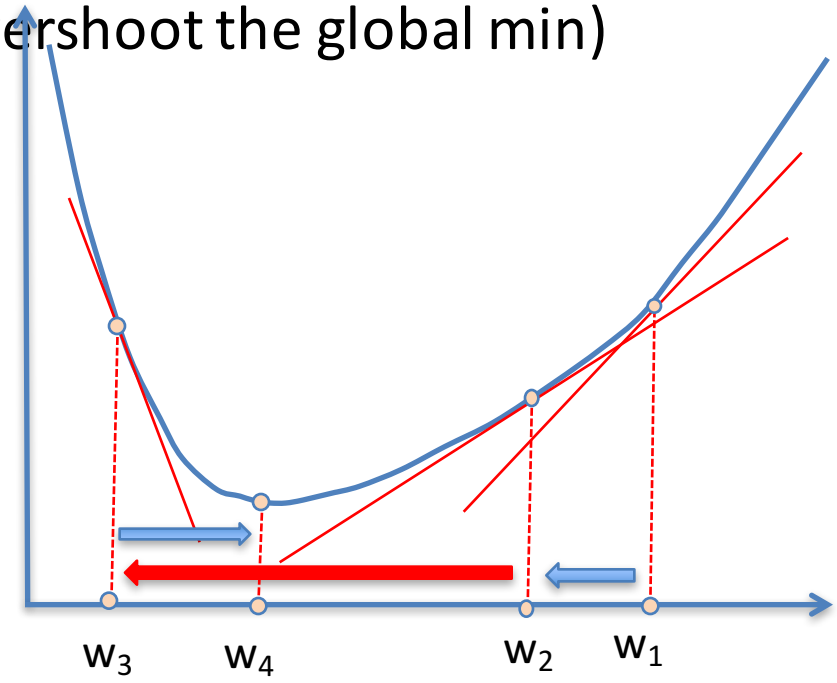
(1) The derivative of the function at w_1 is the slope of the tangent line
→ Positive slope (increasing)
Which direction should we move to decrease the value of Err(w)?

(2) The gradient determines the direction of steepest increase of Err(w) *(go in the opposite direction)*

What is the gradient of Error(w) at this point?

Note about GD step size

- **Setting the step size to a very small value**
  - Slow convergence rate
- **Setting the step size to a large value**
  - May oscillate (consistently overshoot the global min)
- **Tune experimentally**
  - More sophisticated algorithm set the value automatically



The Gradient of Error(w)

The gradient is a generalization of the derivative

$$\nabla Err(\mathrm{w}) = \left(\frac{\partial Err(w)}{\partial w_0}, \frac{\partial Err(w)}{\partial w_1}, \ldots, \frac{\partial Err(w)}{\partial w_n} \right)$$

The gradient is a vector of partial derivatives.

It Indicates the *direction of steepest increase* in Err(w), for each one of w's coordinates

Gradient Descent Updates

- Compute the gradient of the training error at each iteration
  - Batch mode: compute the gradient over all training examples

$$\nabla Err(\mathrm{w}) = \left(\frac{\partial Err(w)}{\partial w_0}, \frac{\partial Err(w)}{\partial w_1}, \ldots, \frac{\partial Err(w)}{\partial w_n} \right)$$

- Update w:

$$\mathrm{w}^{i+1} = \mathrm{w}^{i} - \alpha \nabla Err(\mathrm{w}^{i})$$

Learning rate (>0)

Computing $\nabla Err(w^i)$ for Squared Loss

$$Err(w) = \frac{1}{2}\sum_{d\in D}(y_d - f(x_d))^2$$

$$\frac{\partial Err(\mathbf{w})}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(y_d - f(\mathbf{x}_d))^2$$

$$= \frac{1}{2}\frac{\partial}{\partial w_i}\sum_{d\in D}(y_d - f(\mathbf{x}_d))^2$$

$$= \frac{1}{2}\sum_{d\in D} 2(y_d - f(\mathbf{x}_d))\frac{\partial}{\partial w_i}(y_d - \mathbf{w}\cdot\mathbf{x}_d)$$

$$= -\sum_{d\in D}(y_d - f(\mathbf{x}_d))x_{di}$$

Batch Update Rule for Each w_i

- **Implementing gradient descent**: *as you go through the training data, accumulate the change in each w_i of W*

$$\Delta w_i = \alpha \sum_{d=1}^{D} (y_d - \mathbf{w}^i \cdot \mathbf{x}_d) x_{di}$$

Gradient Descent for Squared Loss

Initialize $\mathbf{w}^0$ randomly
for i = 0…T:
    $\Delta\mathbf{w}$ = (0, …., 0)
    for every training item d = 1…D:
        $f(\mathbf{x_d}) = \mathbf{w}^i \cdot \mathbf{x_d}$
        for every component of $\mathbf{w}$ j = 0…N:
            $\Delta w_j \mathrel{+}= \alpha(y_d - f(\mathbf{x_d})) \cdot x_{dj}$
    $\mathbf{w}^{i+1} = \mathbf{w}^i + \Delta\mathbf{w}$
    return $\mathbf{w}^{i+1}$ when it has converged

Batch vs. Online Learning

- The Gradient Descent algorithm makes updates after going over the entire data set
  - Data set can be huge
  - Streaming mode (we cannot assume we saw all the data)
  - Online learning allows "adapting" to changes in the target function
- Stochastic Gradient Descent
  - Similar to GD, updates after each example
  - Can we make the same convergence assumptions as in GD?
- Variations: update after a subset of examples

Stochastic Gradient Descent

Initialize $\mathbf{w^0}$ randomly

for m = 0…M:

$$f(\mathbf{x_m}) = \mathbf{w^i} \cdot \mathbf{x_m}$$

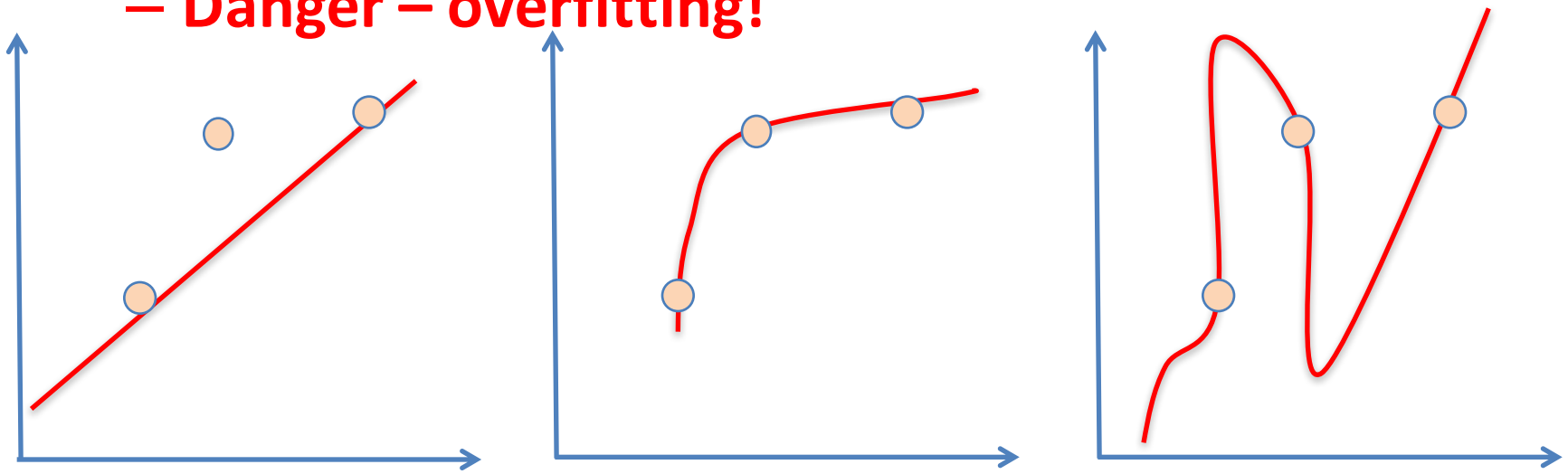
$$\Delta w_j = \alpha(y_d - f(\mathbf{x_m})) \cdot x_{mj}$$

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

return $\mathbf{w^{i+1}}$ when it has converged

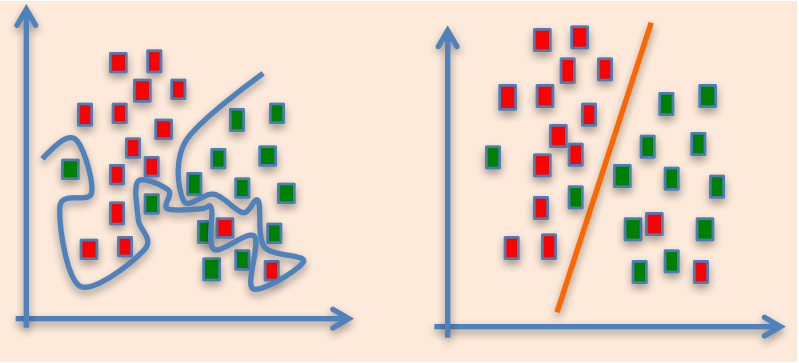
Polynomial Regression

- GD: general optimization algorithm
  - Works for classification (different loss function)
  - Incorporate polynomial features to fit a complex model
  - **Danger – overfitting!**



Regularization

- We want a way to express preferences when searching for a classification function (i.e., ***learning***)
- ***Many functions can agree with the data, which one should we choose?***
  - ***Intuition:*** *prefer simpler models*
  - *Sometimes we are even willing to trade a higher error rate for a simpler model (why?)*
- *Add a regularization term:*
  - *This is a form of* ***inductive bias***



$$\min_{\mathrm{w}} \sum_{n} loss(y_n, \mathrm{w}_n) + \lambda R(\mathrm{w})$$

How different values impact learning?

Regularization

- A very popular choice of regularization term is to minimize the norm of the weight vector
  - For example $||\mathrm{w}|| = \sqrt{\sum_d \mathrm{w}_d^2}$
  - For convenience: ½ squared norm $\min_{\mathrm{w}} = \sum_n loss(y_n, \mathrm{w}_n) + \frac{\lambda}{2}||\mathrm{w}||^2$
    - Q: What is the update gradient of the loss function?
    - *At each iteration we subtract the weights by λ*w*
- In general, we can pick other norms (p-norm)
  - *Referenced as L-p norm*
  - Different p will have a different effect!

$$||\mathrm{w}||_p = \left(\sum_d |\mathrm{w}_d|^p\right)^{\frac{1}{p}}$$

Classification

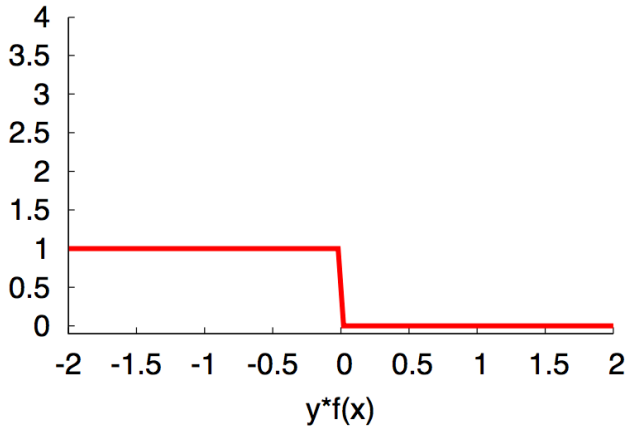
- **So far**:
  - General optimization framework for learning
  - Minimize regularized loss function

$$\min_{\mathrm{w}} = \sum_{n} loss(y_n, \mathrm{w}_n) + \frac{\lambda}{2}||\mathrm{w}||^2$$

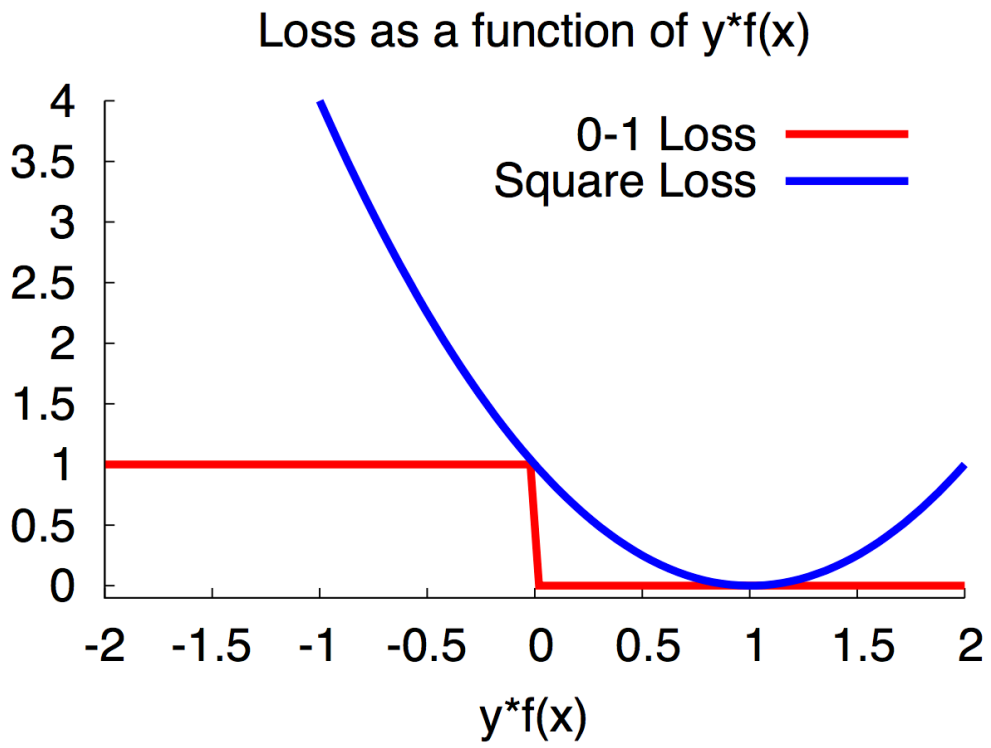
  - Gradient descent is an all purpose tool
    - Computed the gradient of the *square loss function*
- *Moving to classification should be very easy*
  - Simply replace the loss function
- **…**

Classification

- Can we minimize the empirical error directly?
  - Use the 0-1 loss function
- ***Problem:*** *Cannot be optimized directly*
  - *Non convex and non differentiable*
- **Solution**: define a smooth loss function, an upper bound to the 0-1 loss function



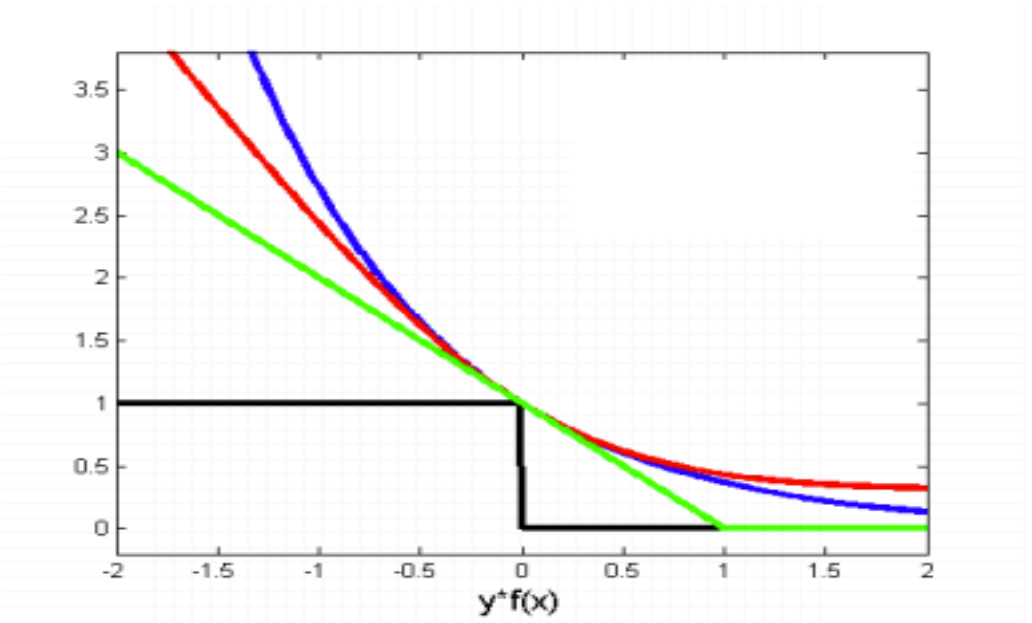
Square Loss is an **Upper bound** to 0/1 Loss



Is the square loss a good candidate to be a surrogate loss function for 0-1 loss?

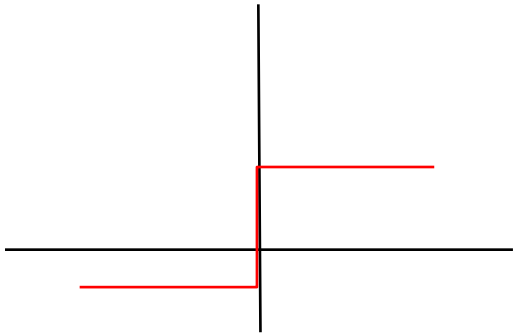
Surrogate Loss functions

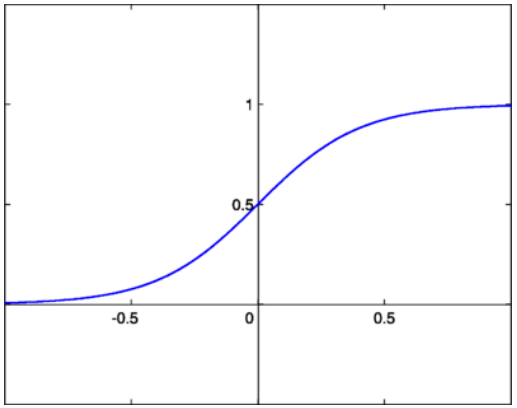
- Surrogate loss function: smooth approximation to the 0-1 loss
  - Upper bound to 0-1 loss



Logistic Function

- Smooth version of the threshold function





$$h_{\mathrm{w}}(x) = g(\mathrm{w}^T x)$$

$$z = \mathrm{w}^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid (logistic) function

- Known as a sigmoid/logistic function
  - Smooth transition between 0-1
- Can be interpreted as the conditional probability
- Decision Boundary
  - y=1: h(x) >0.5➔ $w^t x >= 0$
  - Y=0: h(x) <0.5 ➔$w^t x < 0$

Logistic Regression

- **Learning**: optimize the likelihood of the data
  - Likelihood: probability of our data under current parameters
  - For easier optimization, we look into the log likelihood (*negative*)
- **Cost Function**
  - **If y = 1:** - log P(y=1| x, w) = - log (g(w,x))
  - *If the model gives very high probability to y=1, what is the cost?*
  - **If y = 0:** - log P(y=0| x, w) = - log 1- P(y=1| x, w) = - log (1- g(w,x))
- Or more succinctly (with l_2 regularization)

$$Err(\mathrm{w}) = -\sum_i y_i \log h(x_i) + (1 - y_i)\log (1 - h(x_i)) + \frac{1}{2}\lambda||w||^2$$

- This function is convex and differentiable
  - Compute the gradient, minimize using gradient descent

`

Logistic Regression

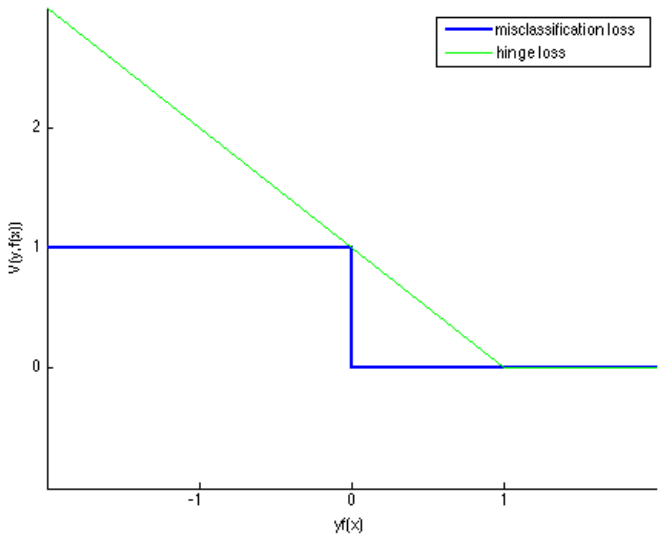
- Let's consider the stochastic gradient descent rule for LR:

$$\mathrm{w}^j = \mathrm{w}^j - \alpha(y_i - h(x_i))\ x^j$$

- What is the difference compared to:
  - **Linear Regression?**
  - **Perceptron?**

Next up: **Hinge Loss**

- Another popular choice for loss function is the hinge loss $L(y, f(x)) = \max(0, 1 - y\, f(x))$
  - We will discuss in the context of support vector machines (SVM)



It's easy to observe that:

(1) The hinge loss is an upper bound to the 0-1 loss

(2) The hinge loss is a good approximation for the 0-1 loss

(3) BUT …

It is not differentiable at $y(w^T x)=1$

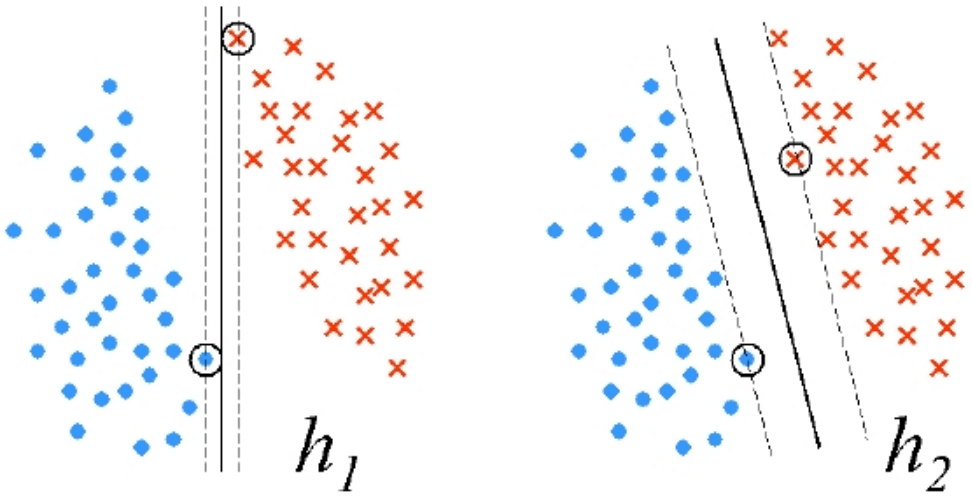
Solution: Sub-gradient descent

Maximal Margin Classification

Motivation for the notion of ***maximal margin***

Defined w.r.t. a dataset S :

$$\gamma(S) = \max \quad \min_{(x,y) \in S} y\, w^T x / ||w||$$



Some Definitions

- **Margin**: distance of the closest point from a hyperplane

This is known as the *geometric margin*, the **numerator** is known as the *functional margin*

$$\gamma = \min_{x_i, y_i} \frac{y_i(\mathbf{w}^T x_i + b)}{||\mathbf{w}||}$$

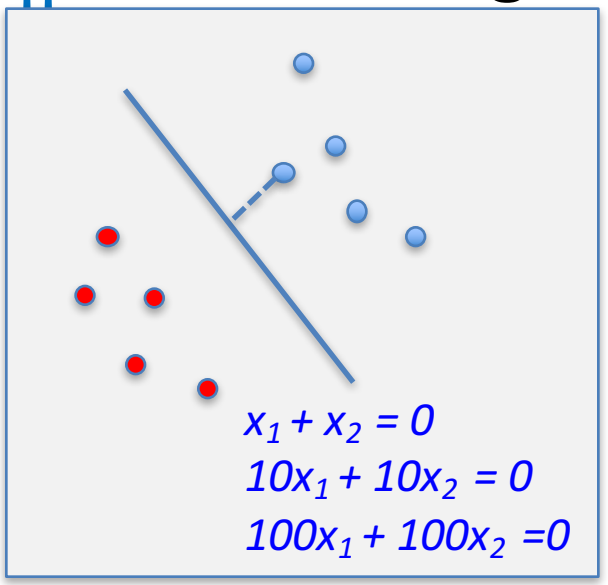
- Our new objective function
  - Learning is essentially solving:

$$\max_{\mathbf{w}} \gamma$$

Maximal Margin

- We want to find $\max_{w} \gamma$
- **Observation**: *we don't care about the magnitude of w!*
- *Set the functional margin to 1, and minimize* ***W***
- $\max_w \gamma$ is equivalent to $\min_w ||w||$ in this setting
- **To make life easy**:

  let's minimize $\min_w ||w||^2$



Hard SVM Optimization

- This leads to a well defined constrained optimization problem, known as Hard SVM:

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

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

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

Hard vs. Soft SVM

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

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

Minimize: $\frac{1}{2} ||w||^2 + c \sum_i \xi_i$

Subject to: $\forall (x,y) \in S$: $y_i\, w^T x_i \geq 1-\xi_i \quad ; \xi_i > 0, i=1...m$

Objective Function for Soft SVM

- The relaxation of the constraint: $y_i w_i^T x_i \geq 1$ can be done by introducing a slack variable ξ (per example) and requiring: $y_i w_i^T x_i \geq 1 - \xi_i; \; (\xi_i \geq)$
- Now, we want to solve:

$$\text{Min } \tfrac{1}{2} ||w||^2 + c \Sigma \boldsymbol{\xi_i} \quad (\text{subject to } \xi_i \geq 0)$$

- Which can be written as:

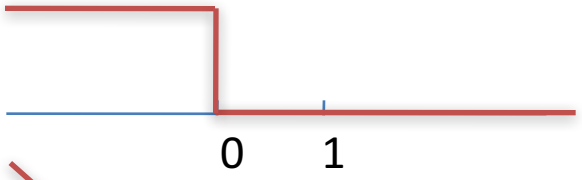
$$\text{Min } \tfrac{1}{2} ||w||^2 + c \boldsymbol{\Sigma \max(0, 1 - y_i w^T x_i).}$$

- *What is the interpretation of this?*
- This is the **Hinge loss** function

Sub-Gradient

Standard 0/1 loss

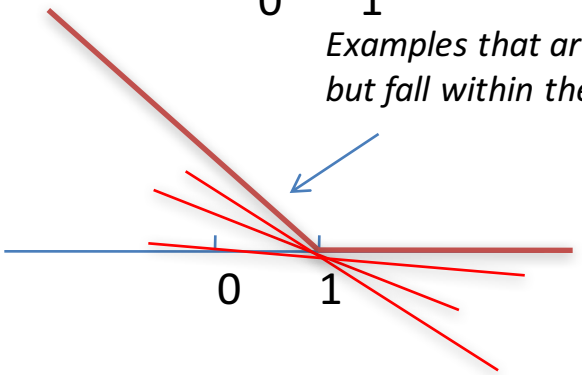
Penalizes all incorrectly classified examples with the same amount



Non Convex

Hinge loss

Penalizes incorrectly classified examples and correctly classified examples that lie within the margin



Examples that are correctly classified but fall within the margin

Convex, but not differentiable at x=1

Solution: *subgradient*

*The **sub-gradient** of a function c at x_0 is any vector v such that:* $\forall x : c(x) - c(x_0) \geq v \cdot (x - x_0)$.

*At **differentiable** points this set only contains the gradient at x_0*

***Intuition**: the set of all tangent lines (lines under c, touching c at x_0)*

$$\partial_w \max\{0, 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\}$$

$$= \partial_w \begin{cases} 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases}$$

$$= \begin{cases} \partial_w 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ \partial_w y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases}$$

$$= \begin{cases} \mathbf{0} & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ y_n \boldsymbol{x}_n & \text{otherwise} \end{cases}$$

Image from CIML58

Summary

- Support Vector Machine
  - Find max margin separator
  - Hard SVM and Soft SVM
  - Can also be solved in the dual
    - Allows adding kernels
- Many ways to optimize!
  - Current: stochastic methods in the primal, dual coordinate descent
- **Key ideas to remember:**
  - **Learning**: Regularization + empirical loss minimization
  - ***Surprise***: Similarities to Perceptron (with small changes)

Summary

- Introduced an optimization framework for learning:
  - Minimization problem
  - Objective: data loss term and regularization cost term
  - Separate learning objective from learning algorithm
    - Many algorithms for minimizing a function
- Can be used for regression and classification
  - Different loss function
  - GD and SGD algorithms
- Classification: use surrogate loss function
  - Smooth approximation to 0-1 loss

Questions?