Perceptron learning
Classification via gradient descent

James G. Shanahan [1,2,3]

[1]Church and Duncan Group,

[2]Information School, UC Berkeley

[3]School of Informatics, Computing and Engineering, Indiana University

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Reading Material

- **Python machine learning, Raschka and Mirijalii, 2017, Pages 17-34**
- https://en.wikipedia.org/wiki/Perceptron
- https://en.wikipedia.org/wiki/Multiclass_classification
- Multiclass Classification: News categorization (Microsoft, 2018)
  - https://gallery.azure.ai/Experiment/Multiclass-Classification-News-categorization-2

HW Assignment

- **Multinomial perceptron versus Logistic regression**
- **Optional: Implement a multinomial perceptron**

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

Linear discriminant functions

- **Linear separators and Loss functions**
  - Review of linear separators
  - Loss Functions
- **Perceptron**
  - Perceptron learning algorithm
  - Perceptron learning graphically speaking
  - Perceptron Extensions
  - Multinomial/Multiclass Perceptron

Perceptron Machine: dawn of modern ML

- **Perceptron Classifier**
  - In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not).
  - It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.
  - The algorithm allows for online learning, in that it processes elements in the training set one at a time.
- **The perceptron algorithm dates back to the late 1950s; its first implementation, in custom hardware, was one of the first artificial neural networks to be produced.**

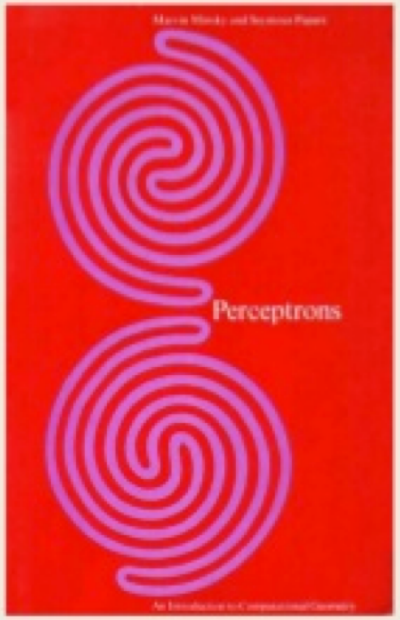
Perceptron Roller-coaster

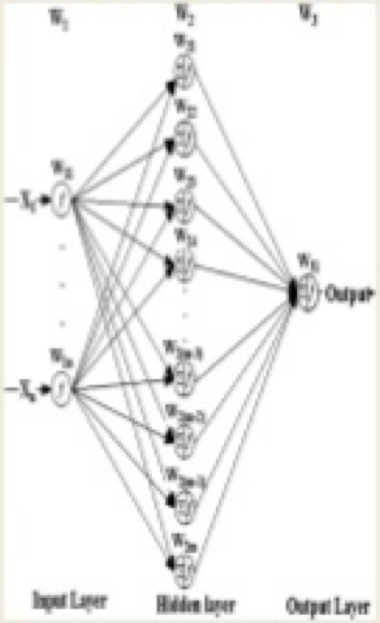
"Scruffies" Rise/Fall/Rise/Fall/Rise

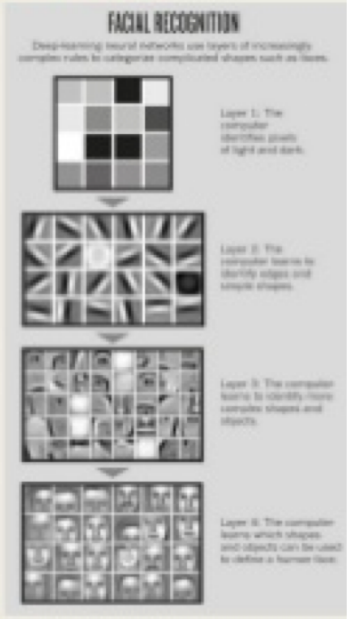
http://www.nature.com/polopoly_fs/7.14689.1389093731!/image/deep-learning-graphic.jpg_gen/derivatives/landscape_400/deep-learning-graphic.jpg

http://www.rutherfordjournal.org/article040101.html

http://opticalengineering.spiedigitallibrary.org/article.aspx?articleid=1714547







1957 Rosenblatt Perceptron

"The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

https://en.wikipedia.org/wiki/Perceptron

1969 Minsky & Papert
Can't do XOR!

https://constructingkids.files.wordpress.com/2013/05/minsky-papert-71-csolomon-x640.jpg

http://www.i-programmer.info/images/stories/BabBag/AI/book.jpg

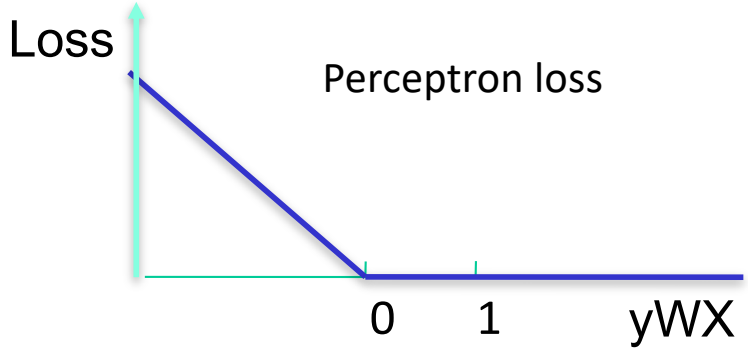
Backpropagation
1986 Rumelhart
(1963 Bryson
1974 Werbos)

Deep Learning
2007 Hinton
(1989 LeCun
1992 Schmidhuber)

Artificial Neurons

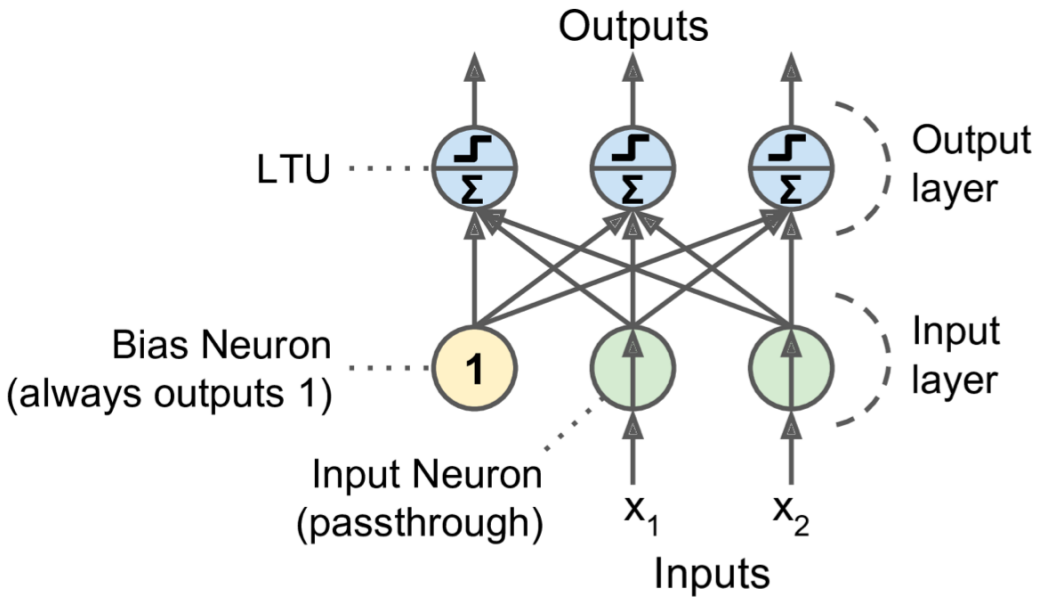
- **1943: The McCulloch & Pitts (perceptron) neuron**
  - Weighted sum of inputs
  - Activation Function
    - "Link" function in GLM/regression (e.g., sigmoid or hyperbolic tangent link for logistic regression)
- **1957: Perceptron learning algo (Rosenblatt 1957)**
  - Learning rate and learning (update) rule
  - Classification problems
  - Neural network as a network of logistic regressions

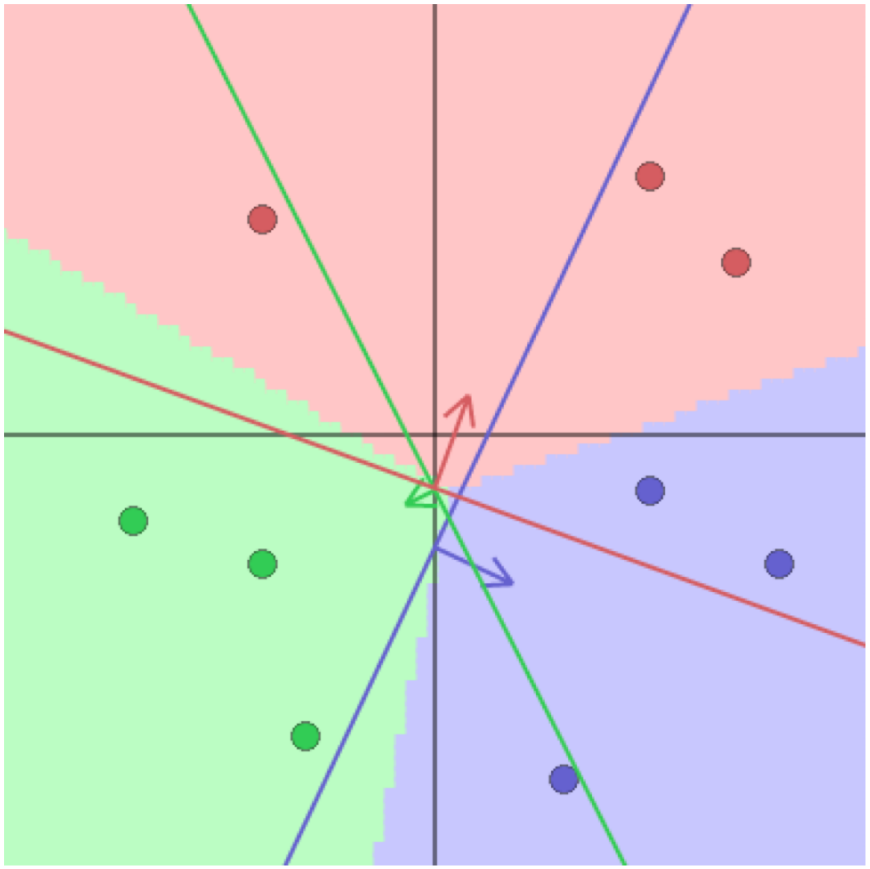
Penalizes incorrectly classified examples x proportionally to the size of |w'x|



A 2-3 Multinomial Perceptron

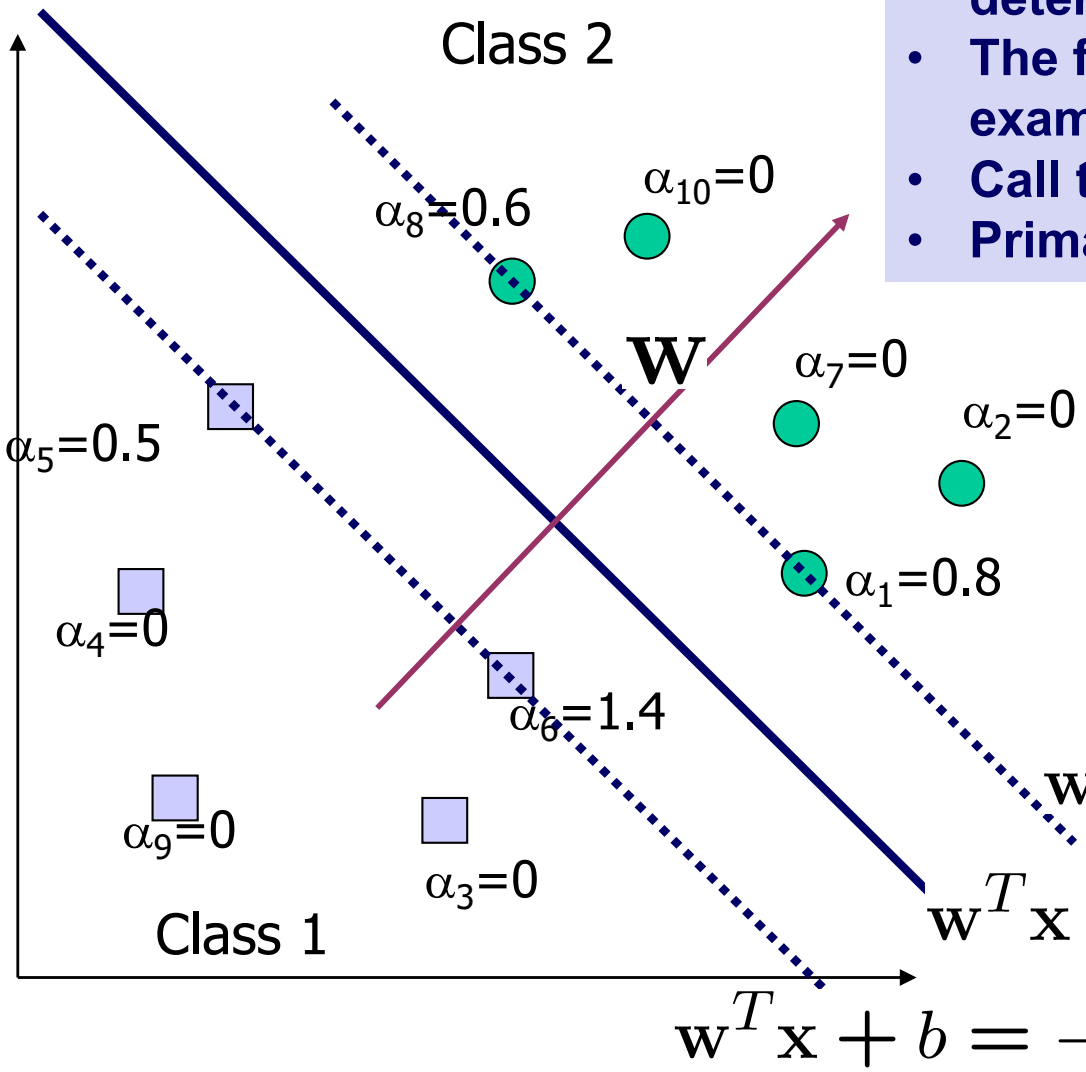
- **Classification rule is similar to logistic regression**
  - For each class calculate the perpendicular distance
  - $y' = \text{argmax}_y (X^T w_y)$. #class with largest perdDist

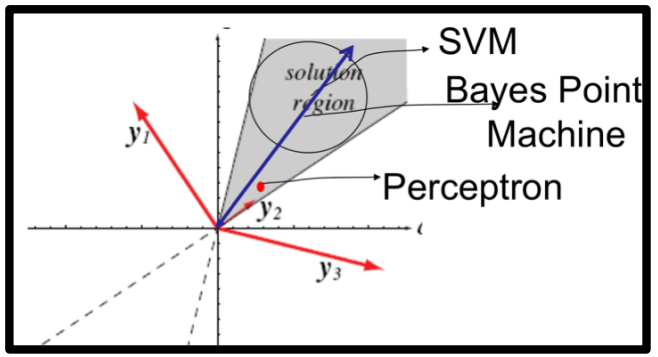




Margin-based Perceptron → SVMs [1992]

- Use cross fold validation to determine the best margin
- The frequently misclassified examples have high α's.
- Call them Support Vectors
- Primal ←→ Dual form





Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

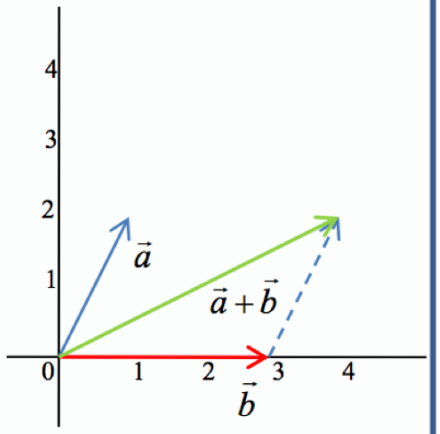
Vector Algebra

a + b

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} + \vec{b} = (4,2)$

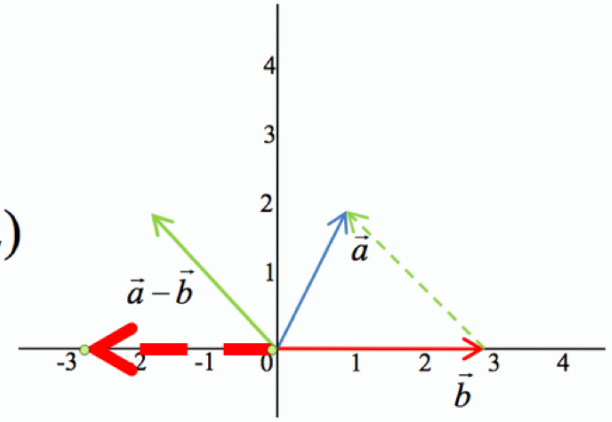


a – b

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



4. Euclidian length or L2-norm

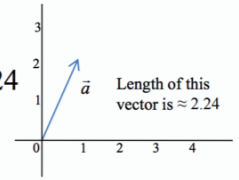
Consider a vector $\vec{a} = (a_1, a_2, \ldots, a_n)$

Define the L2-norm: $\|\vec{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$

We often denote the L2-norm without subscript, i.e. $\|\vec{a}\|$

$\vec{a} = (1,2)$

$\|\vec{a}\|_2 = \sqrt{5} \approx 2.24$



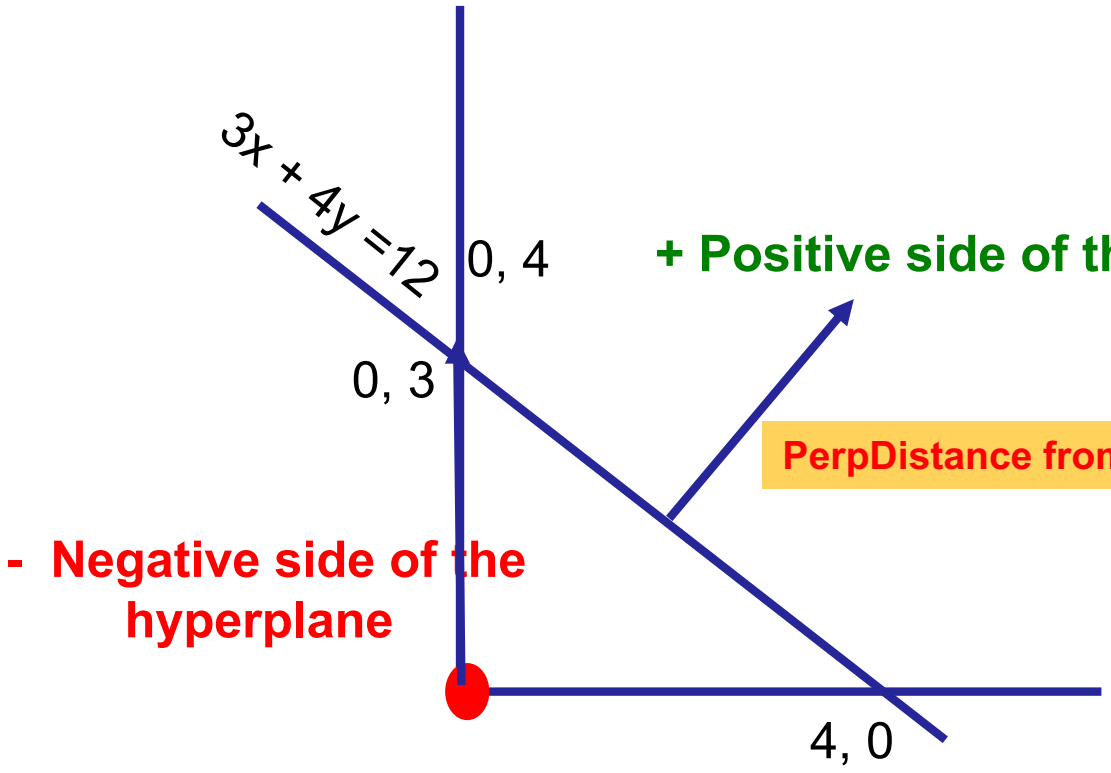
L2-norm is a typical way to measure length of a vector; other methods to measure length also exist.

Separating hyperplanes

- Hyperplanes are not about prediction of values (like in linear regression) but separating classes in the case of classification
- This section focuses on Separating Planes, Separating hyperplanes and vector geometry

HyperPlane 3x + 4y =12

QUESTION: What far is (0,4) from the hyperplane?



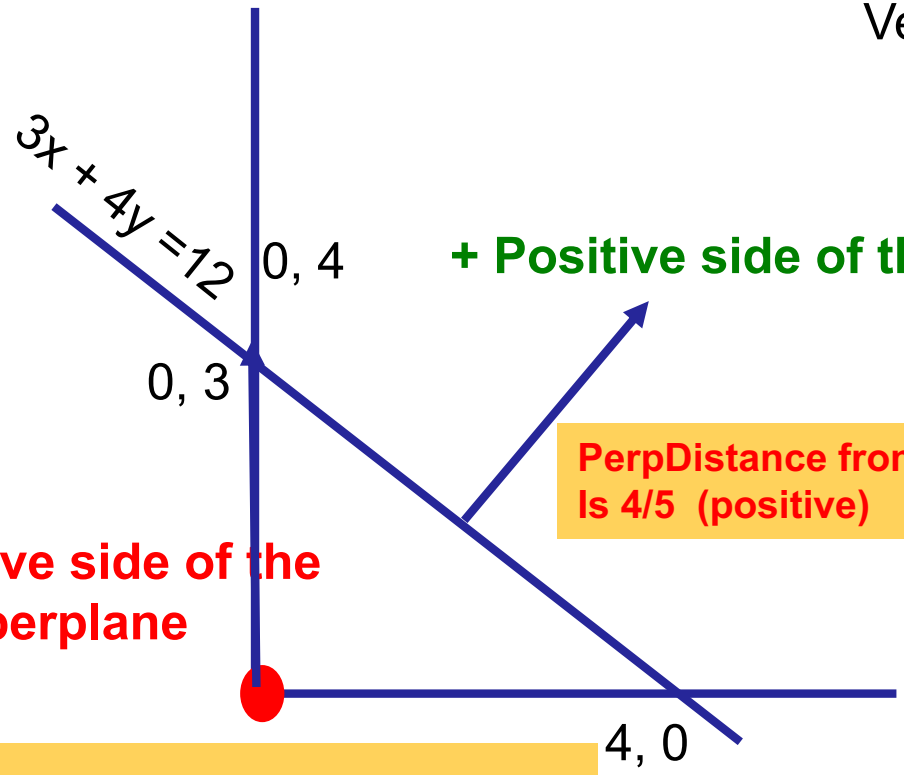
PerpDistance from point (0,4) to the hyperplane 3x + 4y =12

HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?

Shortcut: 3(0) + 4(0) -10 / SQTR((3 * 3) (4*4)) = -10/5=-2

Vector that normal to Hyperpane (3, 4)



+ Positive side of the hyperplane

PerpDistance from point (0,4) to the hyperplane 3x + 4y =12 **Is 4/5 (positive)**

- Negative side of the hyperplane

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$

Shortcut:
(3(0) + 4(0) -12)
---------- = -12/5 = -2.4
SQTR((3 * 3) + (4*4))

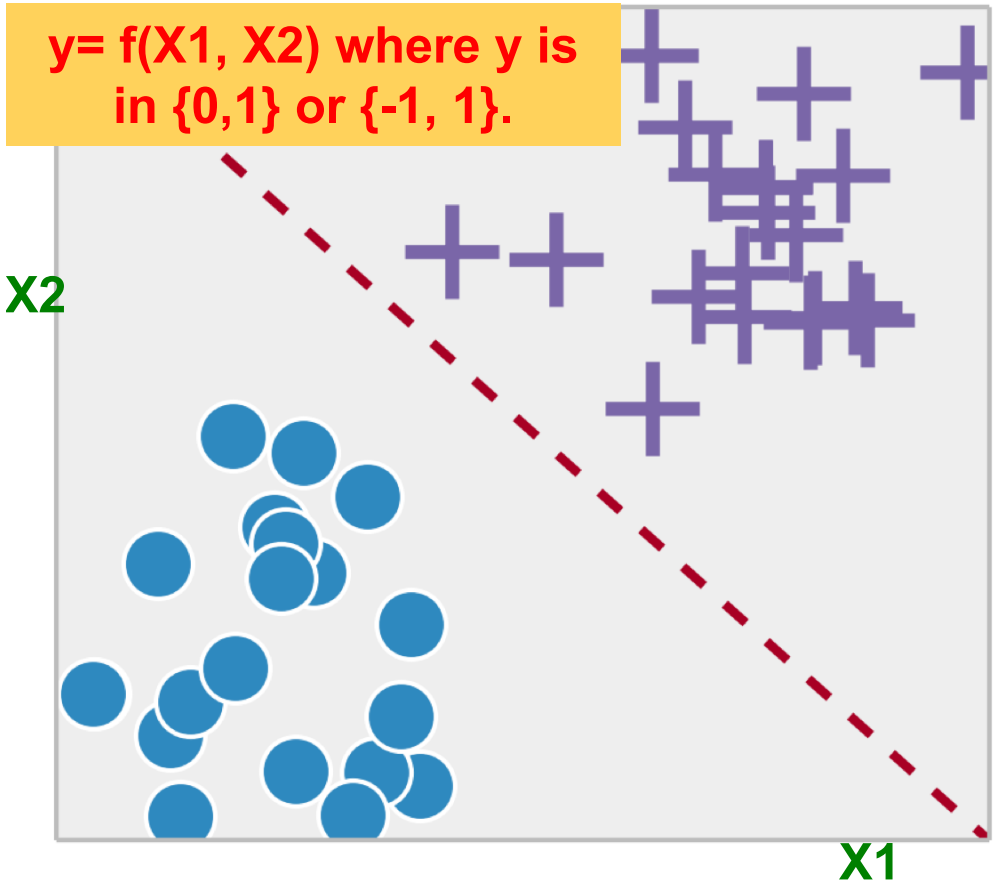
Hyperplanes partition the input space(a line in a 2 input variable problem) and do NOT predict real values

Partitions versus predicts

3Dim world

Classification

y= f(X1, X2) where y is in {0,1} or {-1, 1}.

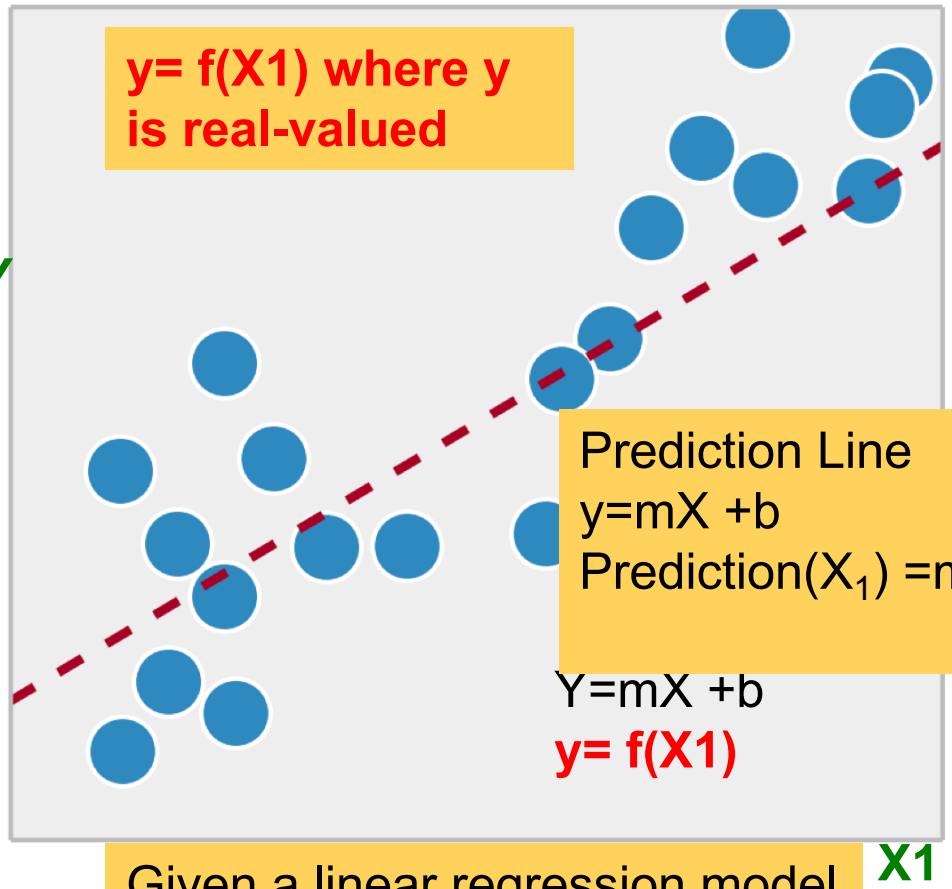


Separating hyperplane partitions
$AX_1 + BX_2 + C$
$Class(X_1, X_2) = sign(AX_1 + BX_2 + C)$

Minimize Residuals

Regression

y= f(X1) where y is real-valued

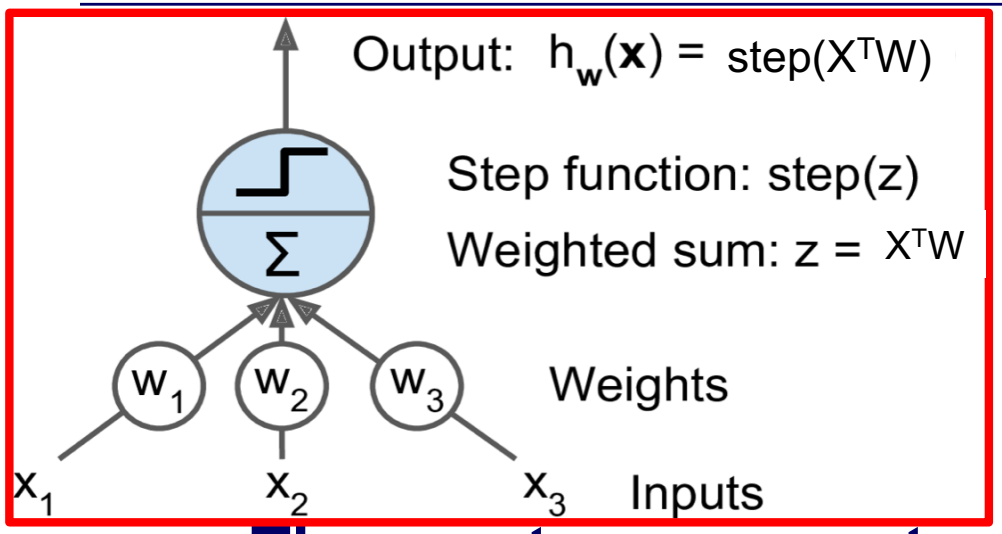


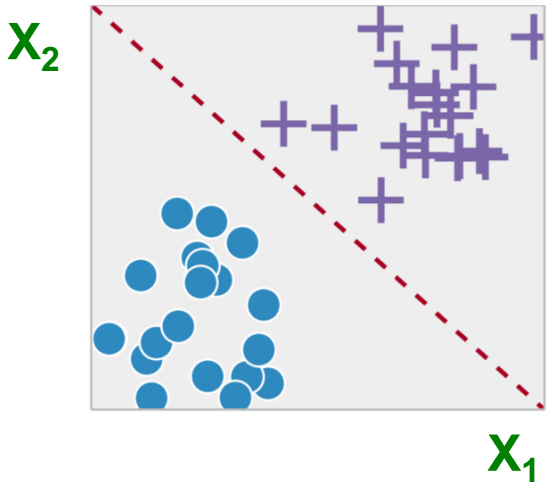
Prediction Line
y=mX +b
Prediction(X_1) =mX

Y=mX +b
y= f(X1)

Given a linear regression model W, Please type in the loss function for linear regression

Perceptron: Linear Threshold Unit (LTU)





- **The most common step function used in Perceptrons is the Heaviside step function**

$$\mathrm{f(x)} = \mathrm{heaviside}\left(X_i^T W\right)$$

$$where\ \ heaviside\left(X_i^T W\right) = \begin{cases} 0 & X_i^T W < 0 \\ 1 & X_i^T W \geq 0 \end{cases}$$

- **Sometimes the sign function is used instead**

$$\mathrm{f(x)} = \mathrm{sgn}\left(X_i^T W\right)$$

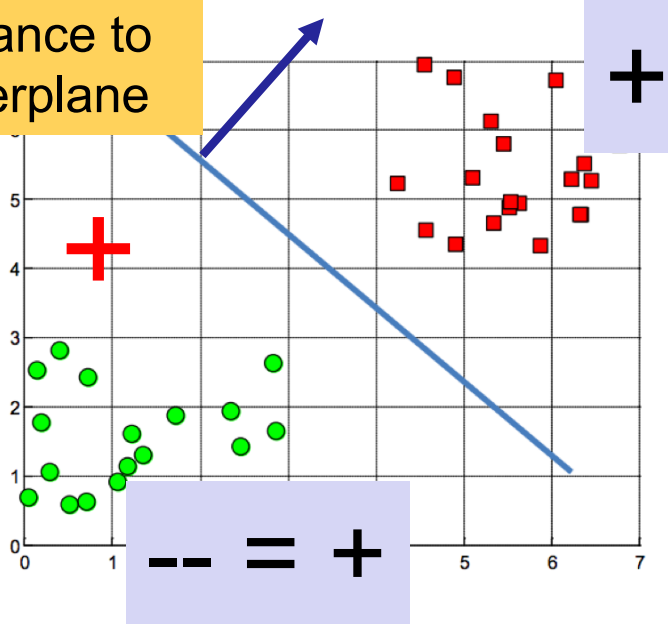
$$where\ sgn\left(X_i^T W\right) = \begin{cases} -1 & X_i^T W < 0 \\ 0 & X_i^T W = 0 \\ 1 & X_i^T W > 0 \end{cases}$$

Hyperplanes as decision surfaces

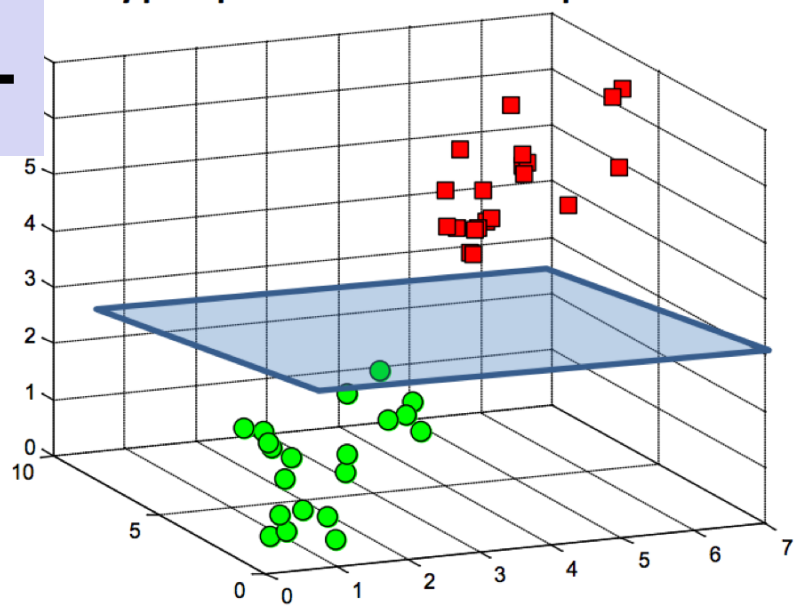
- A hyperplane is a linear decision surface that splits the space into two parts;
- It is obvious that a hyperplane is a binary classifier.

A hyperplane in $\mathbb{E}^2$ is a line

Distance to hyperplane



A hyperplane in $\mathbb{E}^3$ is a plane



A hyperplane in $\mathbb{E}^n$ is an n-1 dimensional subspace

Hyperplanes as decision surfaces

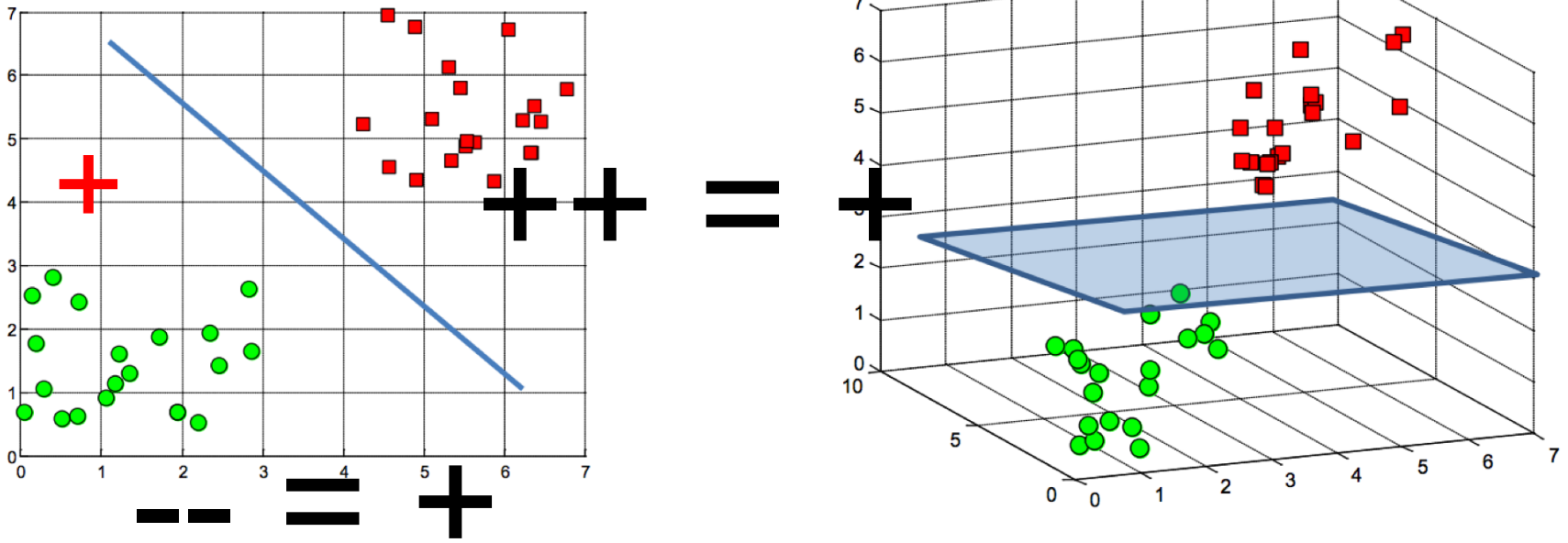
- A hyperplane is a linear decision surface that splits the space into two parts;
- It is obvious that a hyperplane is a binary classifier.

$w_1 x_1 + w_2 x_2 + w0 == 0$ (points on the hyperplane)
$w_1 x_1 + w_2 x_2 + w0 > 0$ (points on +side of hyperplane)
$w_1 x_1 + w_2 x_2 + w0 < 0$ (points on – side of hyperplane)

$w_1 x_1 + w_2 x_2 + w_3 x_3 + w0$

A hyperplane in $\mathbb{E}^2$ is a line

A hyperplane in $\mathbb{E}^3$ is a plane

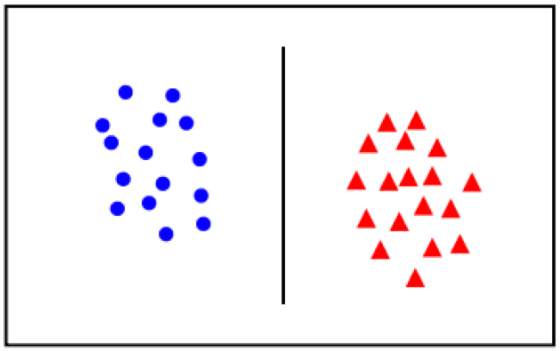


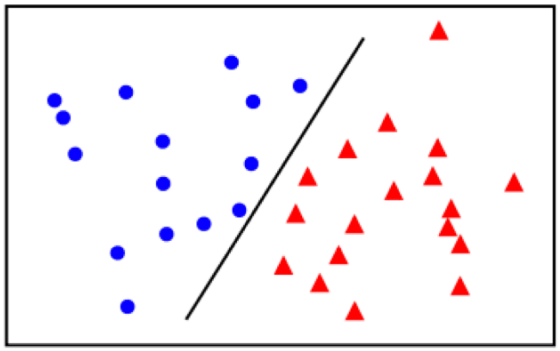
A hyperplane in $\mathbb{E}^n$ is an n-1 dimensional subspace

Linear Classifiers

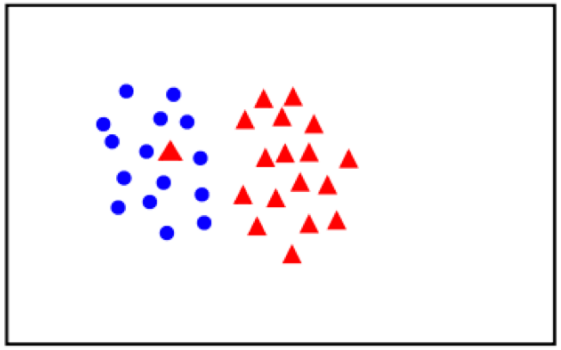
- **Many common text classifiers are linear classifiers**
  - Naïve Bayes
  - Perceptron
  - Rocchio
  - Logistic regression
  - Support vector machines (with linear kernel)
  - Linear regression with threshold
- **Despite this similarity, noticeable performance differences**
  - For separable problems, there is an infinite number of separating hyperplanes. Which one do you choose?
  - What to do for non-separable problems?
  - Different training methods pick different hyperplanes
- **Classifiers more powerful than linear often don't perform better on text problems. Why?**

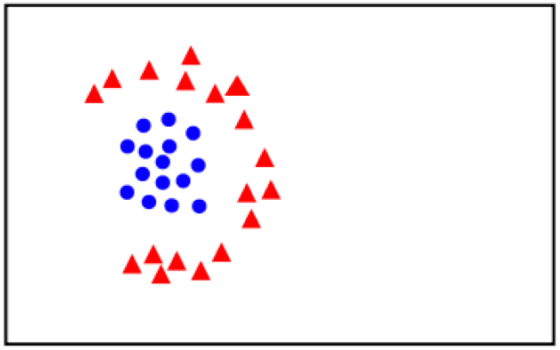
linearly separable





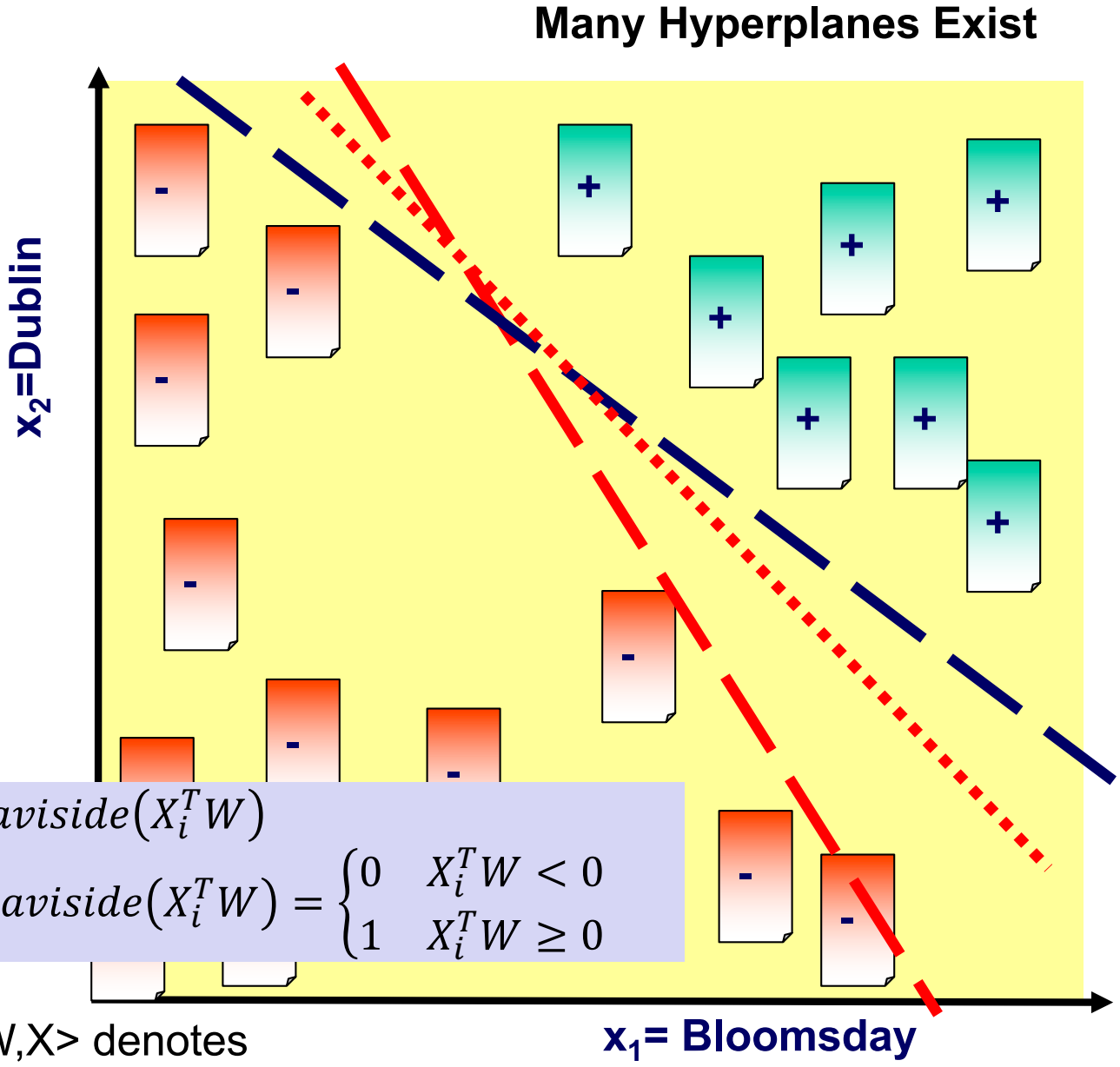
not linearly separable





Linear Discriminant Model

A linear discriminant function is linear in the components of X



Training Data

E.g.	x_1	y
1	3	-1
2		+1
…	…	…
L	0	-1

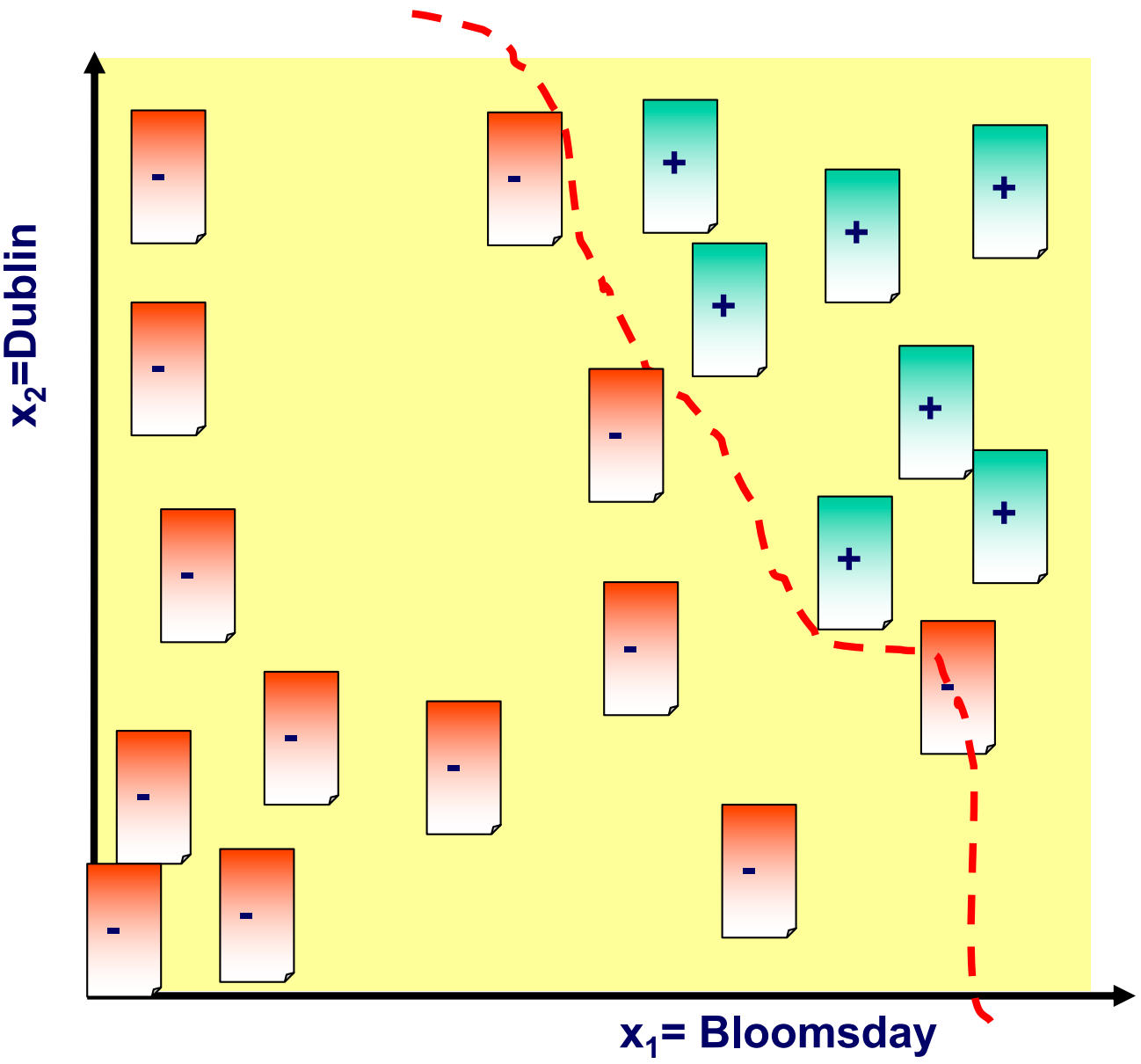
$$f(x) = heaviside(X_i^T W)$$

$$where\ \ heaviside(X_i^T W) = \begin{cases} 0 & X_i^T W < 0 \\ 1 & X_i^T W \geq 0 \end{cases}$$

Nonlinear Discriminant Model

A nonlinear discriminant function is nonlinear in the components of X

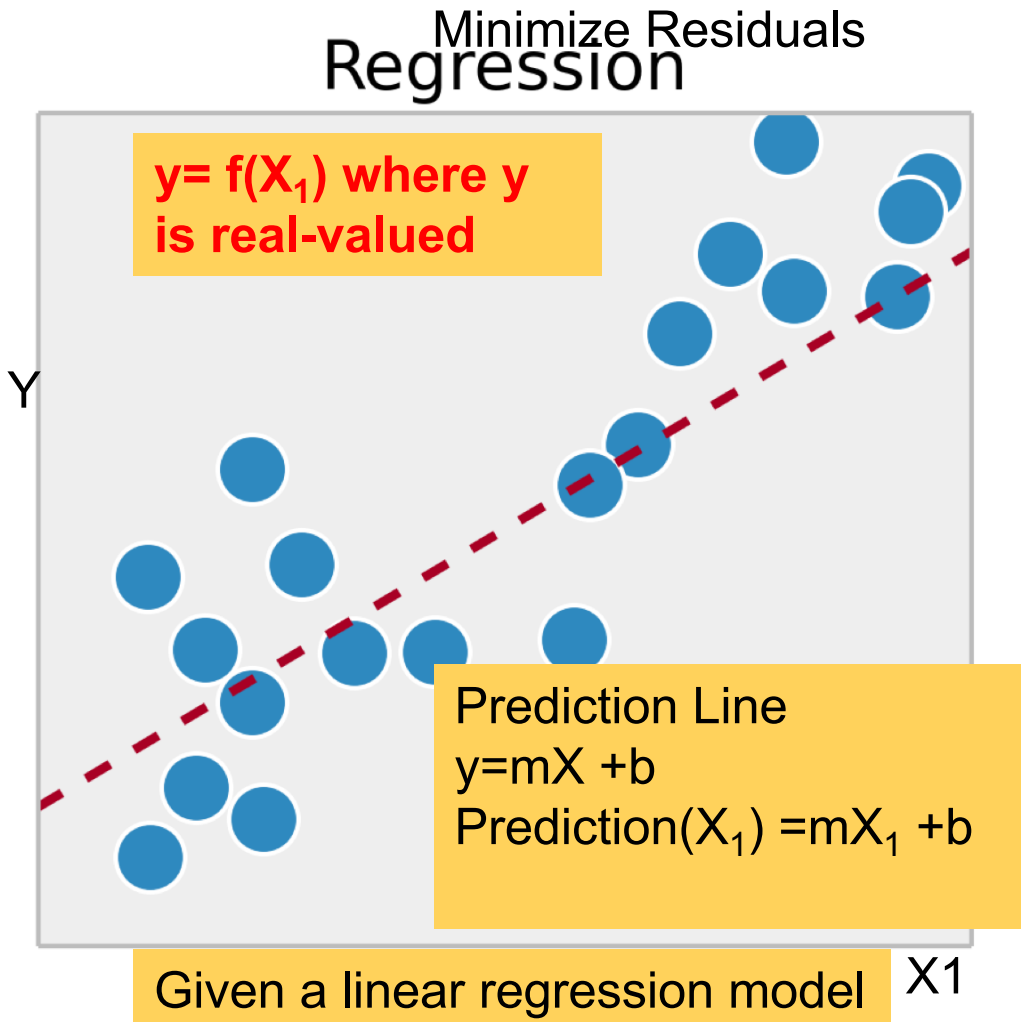
E.g., $y=ax_1^2+bx_2^3+c$



Training Data

E.g.	x_1	x_2	y
1	3	0	-1
2			+1
…	…	…	…
L	0	4	-1

Linear Regression Loss Function

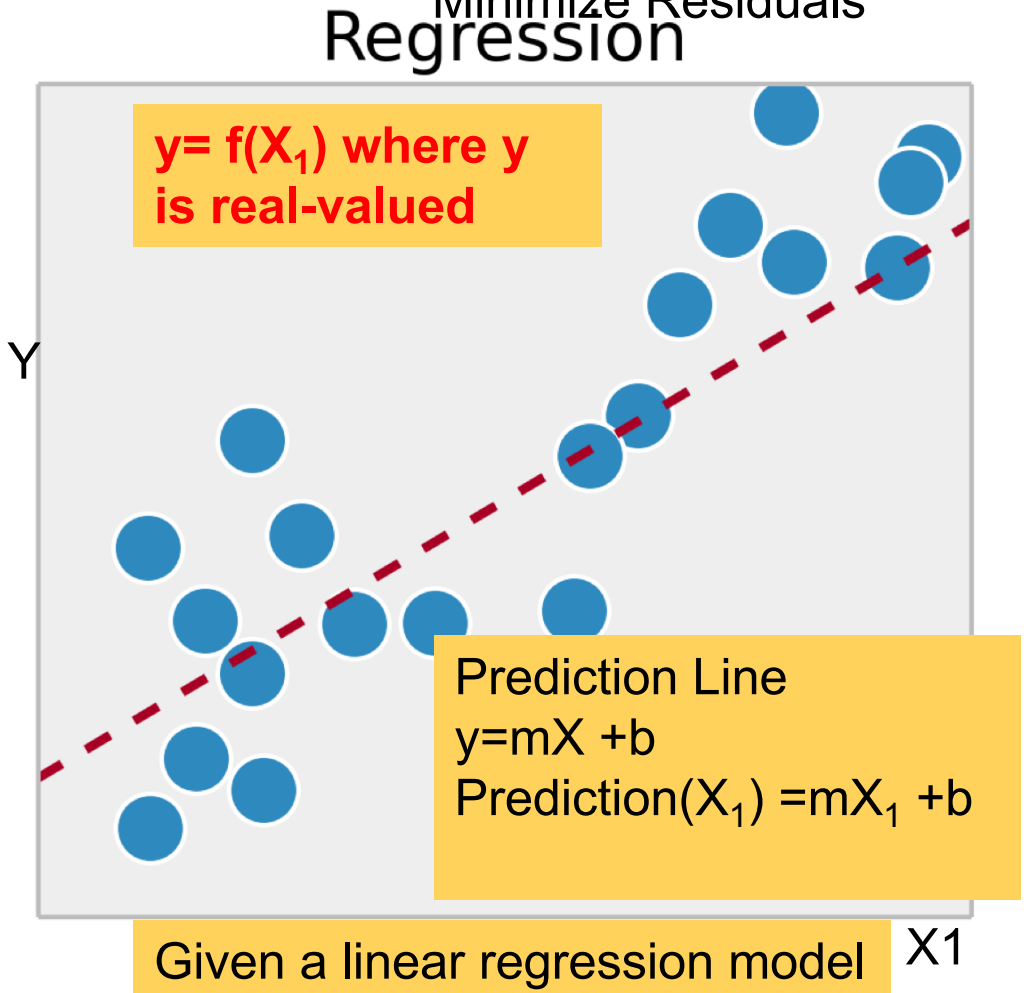


Given a linear regression model W, Please type in the loss function for linear regression

Linear Regression Loss Function

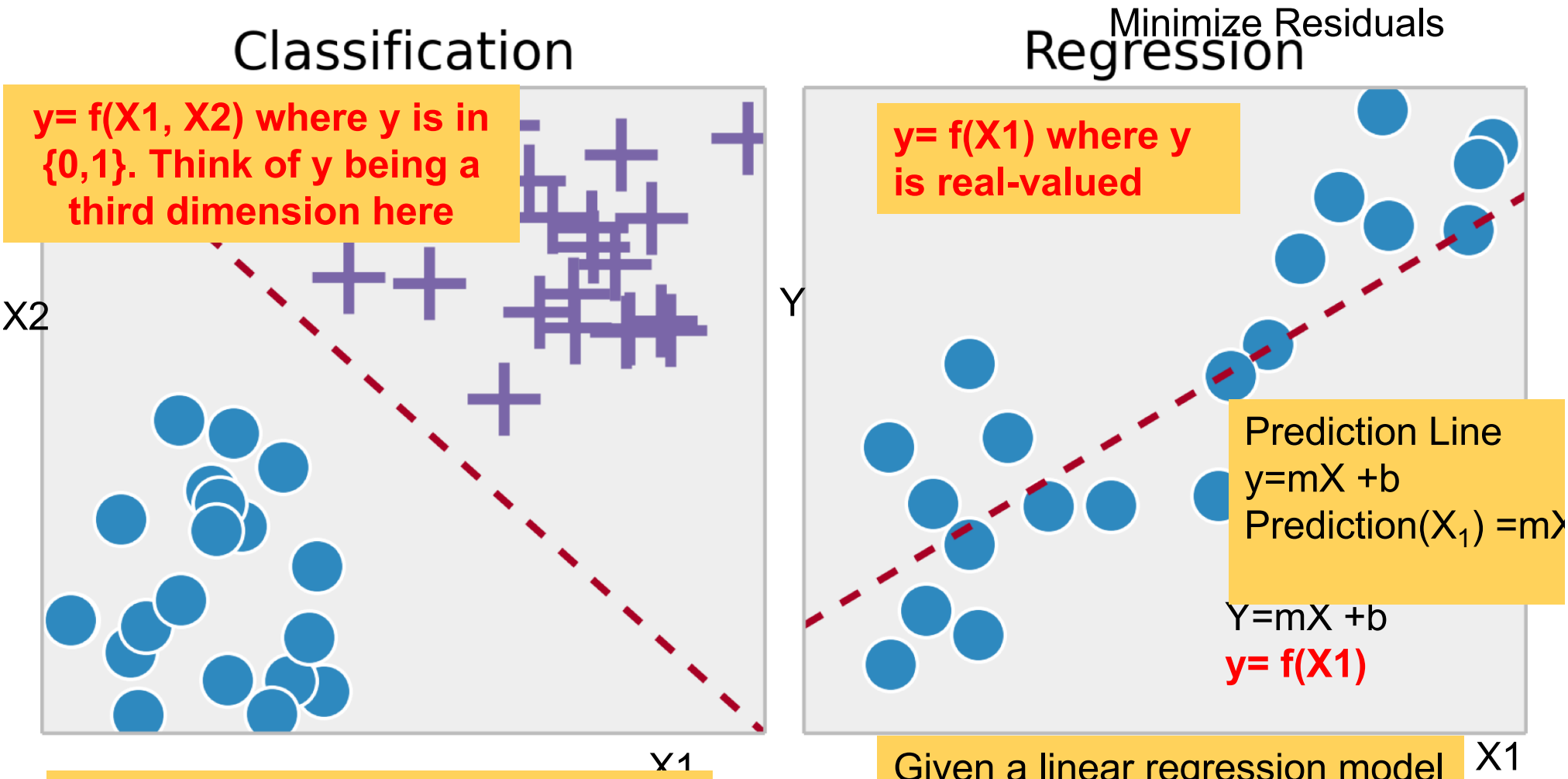
$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals



Given a linear regression model W, Please type in the loss function for linear regression

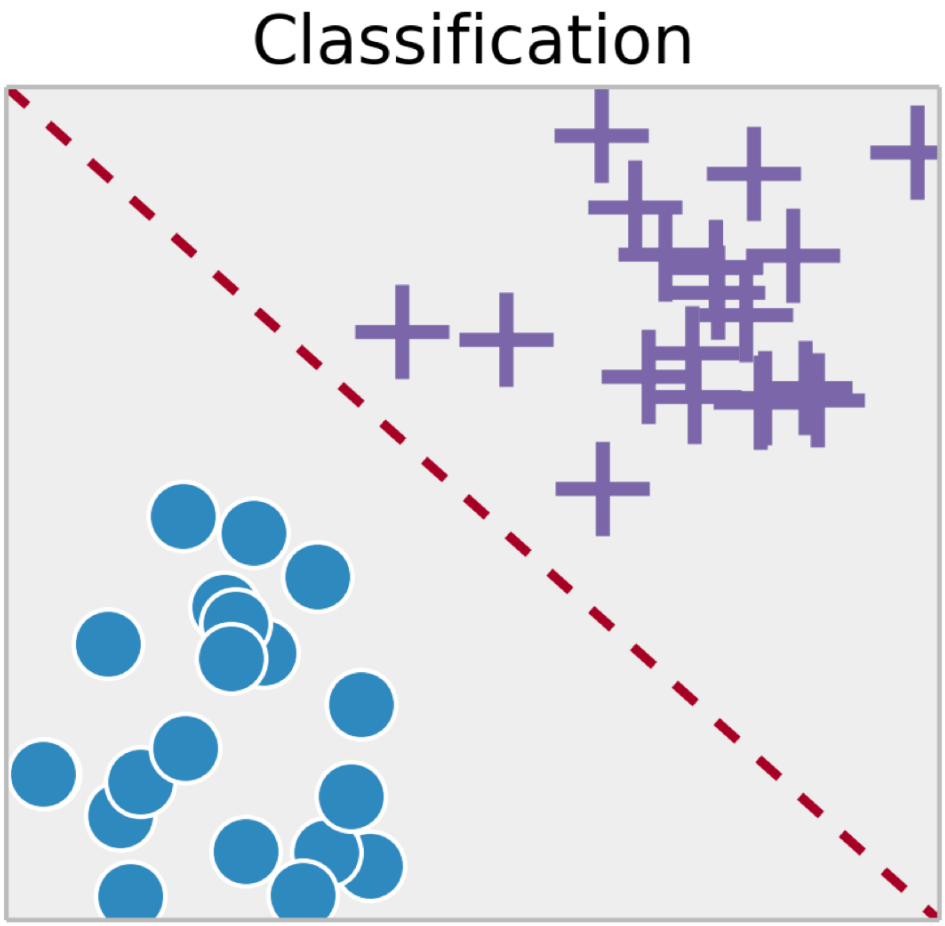
Convex optimization in ML: Loss Functions

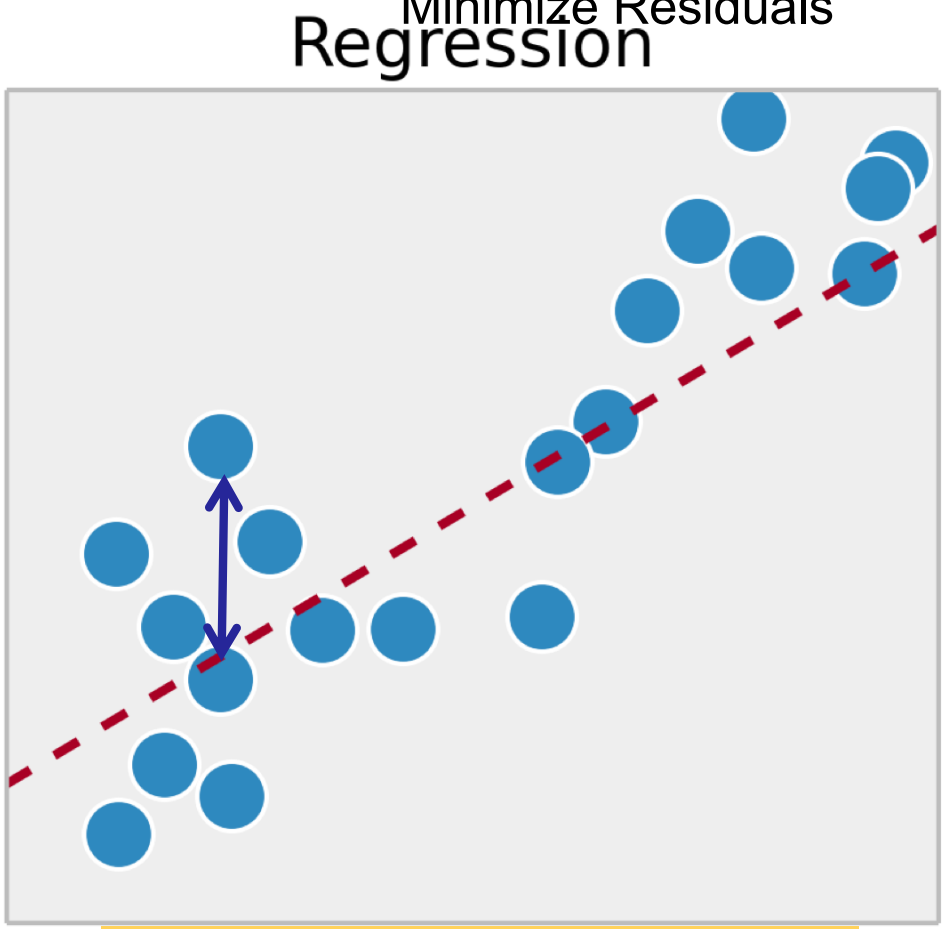


Convex optimization in ML

$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals

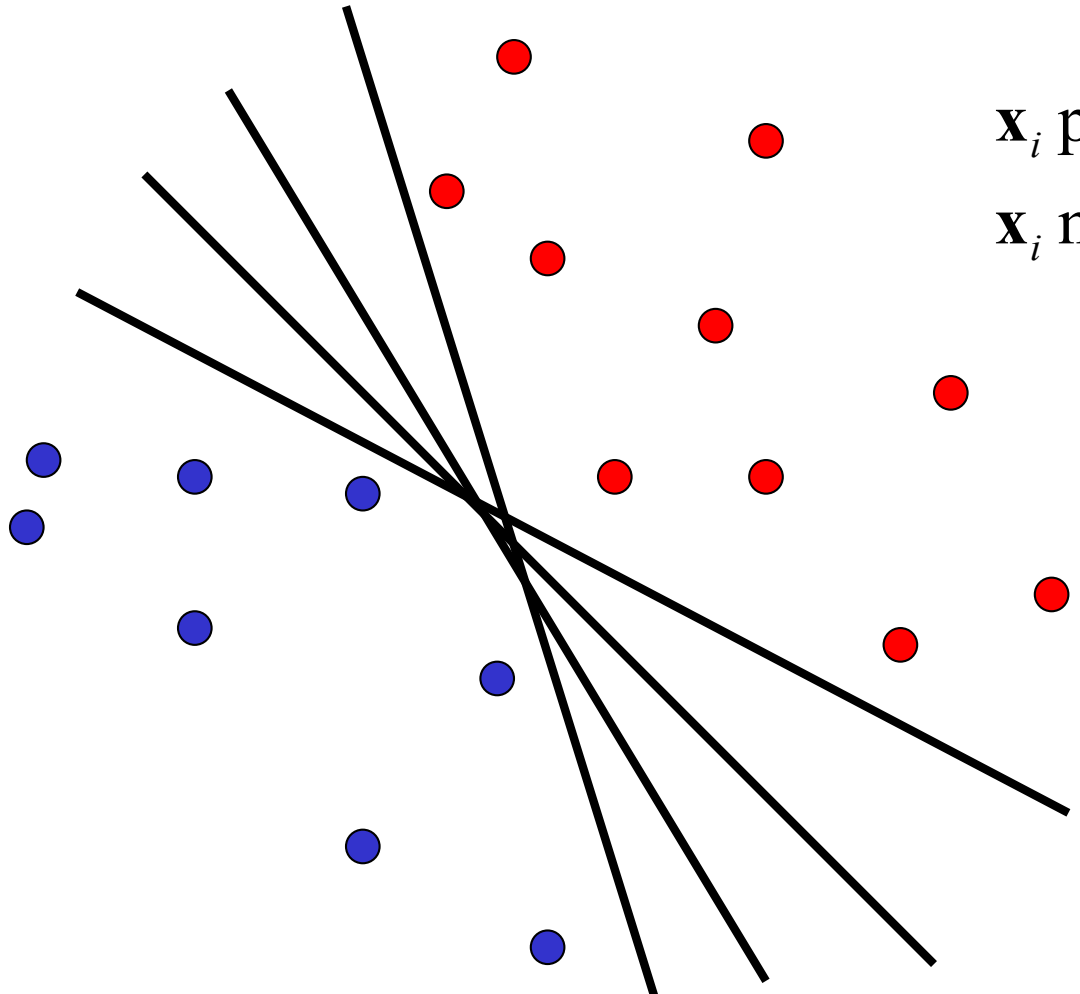




Please type in the loss function for linear regression (W is the model)

Linear classifiers

- **Find linear function (*hyperplane*) to separate positive and negative examples**



$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

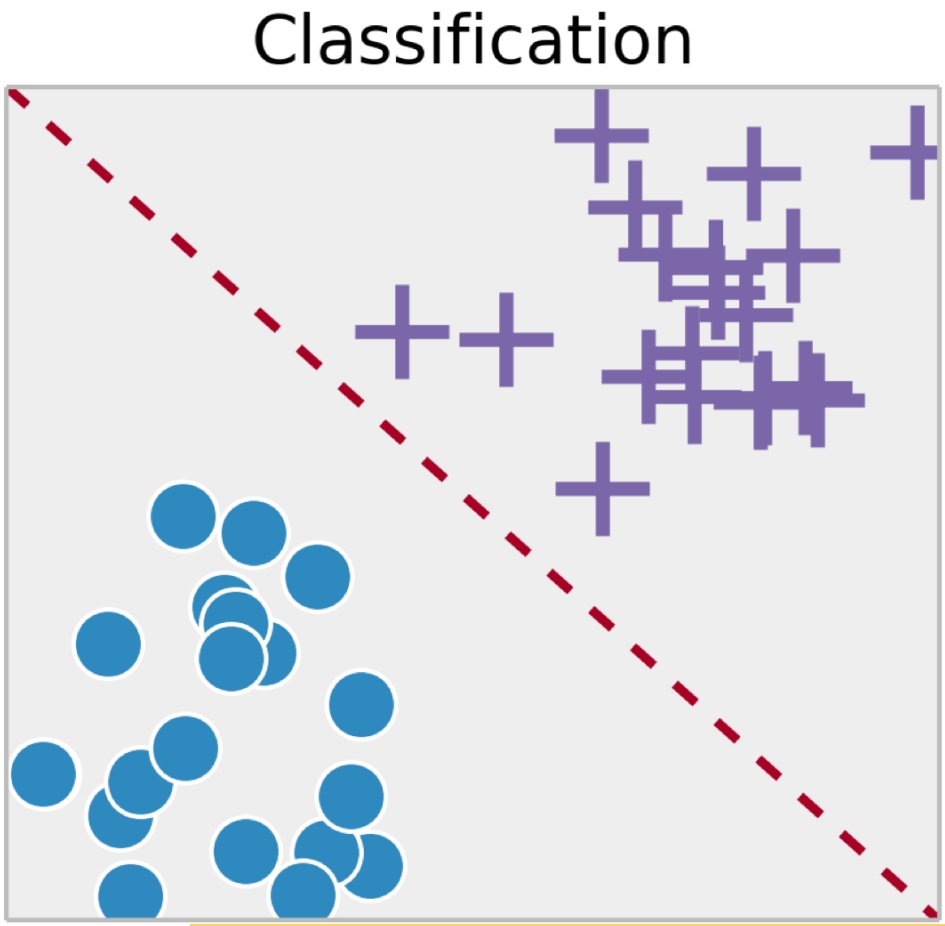
$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

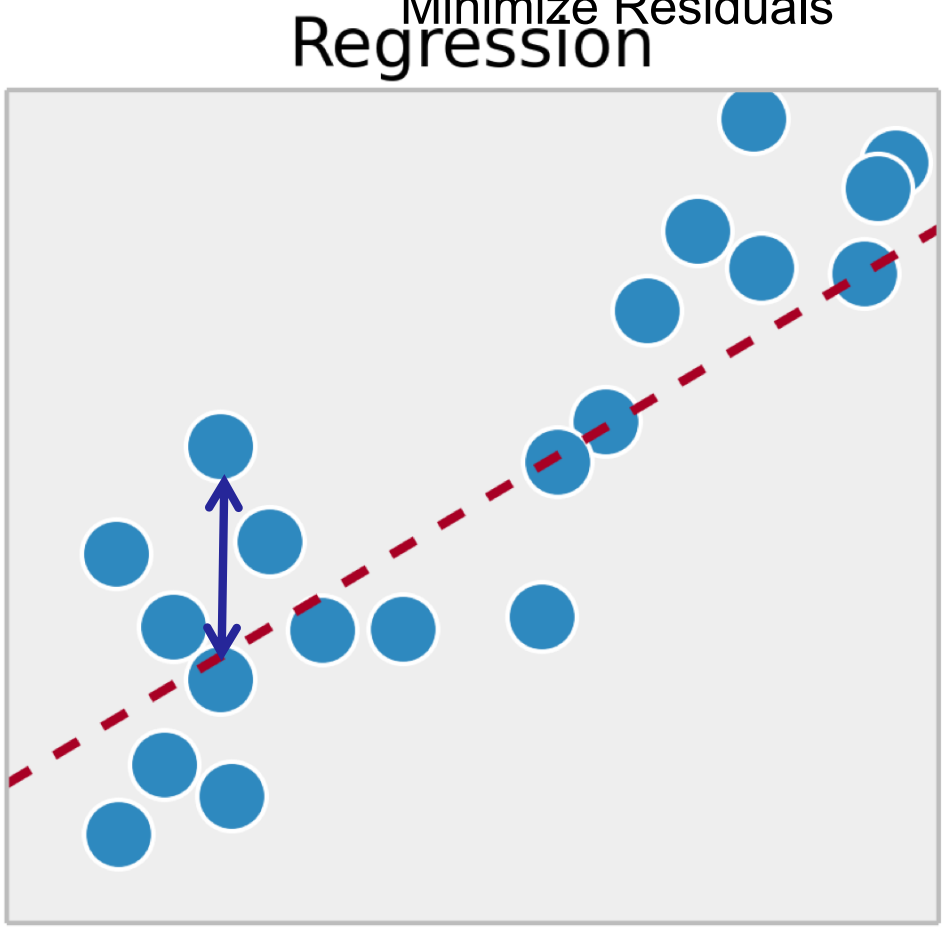
Which hyperplane is best?

Convex optimization in ML

$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals





QUIZ: Think of a loss function for Classification

Augment; Encode the class; Learn

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	-1
2					+1
…	…	…	…	…	…
L (aka m)	0	4	…	8	-1

Instance\Attr	x_0	x_1	x_2	…	x_n	y
1	1	3	0	..	7	-1
2	1					+1
…	1	…	…	…	…	…
L (aka m)	1	0	4	…	8	-1

Class Encodings

- {0, 1} [Logistic regression] versus {-1, 1} [Perceptron, SVM, NN]
- For multiple classes we will do a OHE (one-hot-encoding of the target variable

Learn a linear function $g(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} = \langle w_0, w_1, w_2, w_3, w_4 \rangle$

- Each **w** corresponds to one hypothesis
  $h(\mathbf{x}) = \text{sign}(g(\mathbf{x},\mathbf{w}))$
- A prediction is correct if $y\,\mathbf{w}^T\mathbf{x}>0$
- Goal of learning is to find a good **w**
  - e.g., a **w** such that $h(\mathbf{x})$ makes few mis-predictions

An Optimization Problem

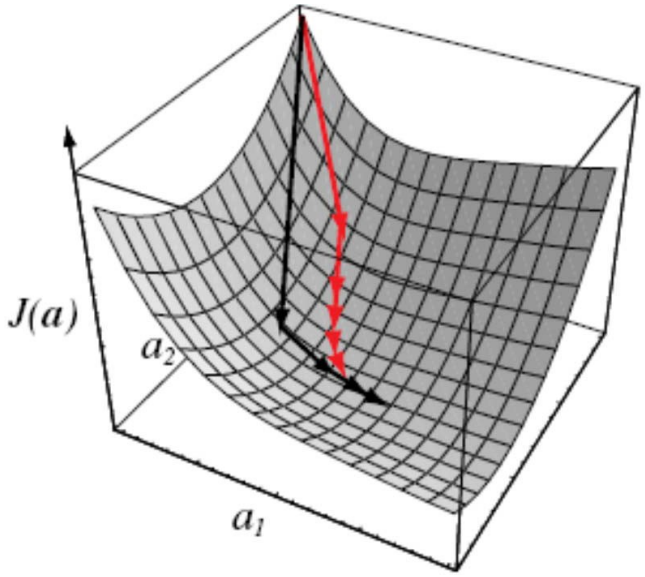
- Formulate learning problem as an optimization problems
  - Given:
    - A set of *N* training examples

      $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$
    - A loss function *L*
  - Find the weight vector **w** that minimizes the objective function - the expected/average loss on training data

$$J = \frac{1}{N}\sum_{i=1}^{N} L(X_i^T W, y_i)$$

- Many machine learning algorithms apply some optimization algorithm to find a good hypothesis.

Gradient Descent Search



- Start with initial **w** = (w_0,..., w_n)
- Compute gradient $\nabla J(\mathbf{w}_0) = (\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_n})_{\mathbf{w}_0}$

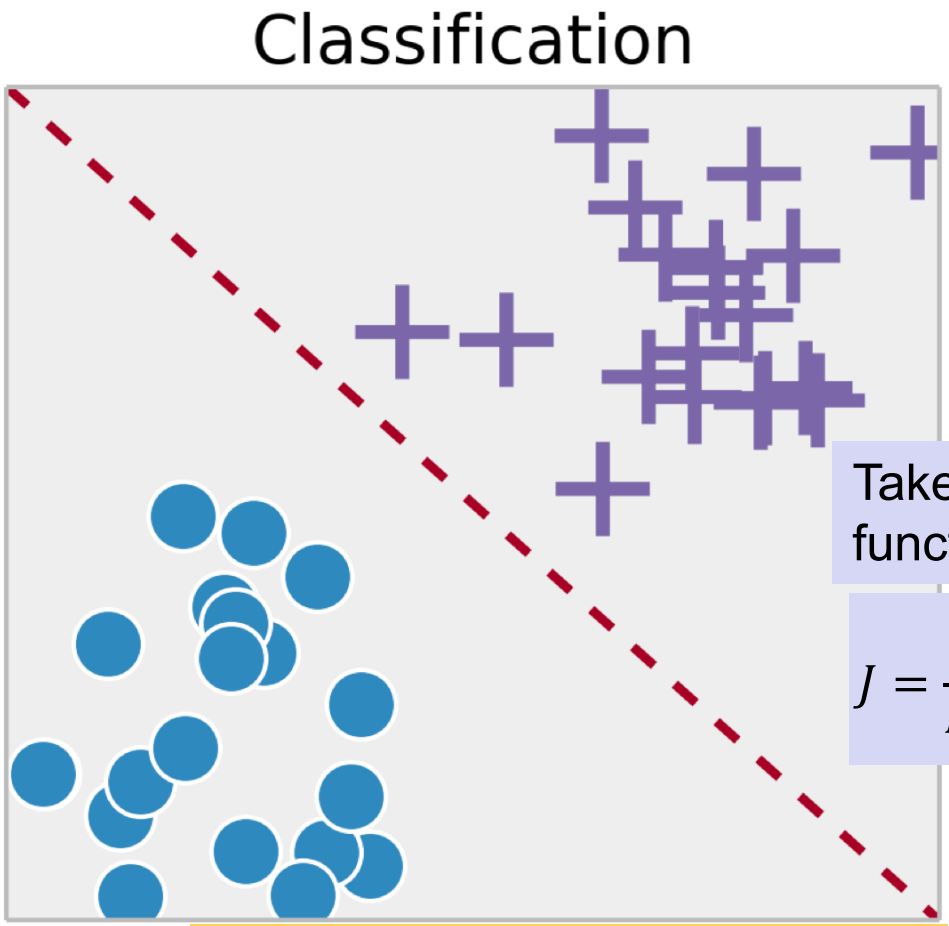
- $W_{t+1} = W_t - \eta \nabla J(W_t)$ Where η is the "step size" parameter
- Repeat until convergence

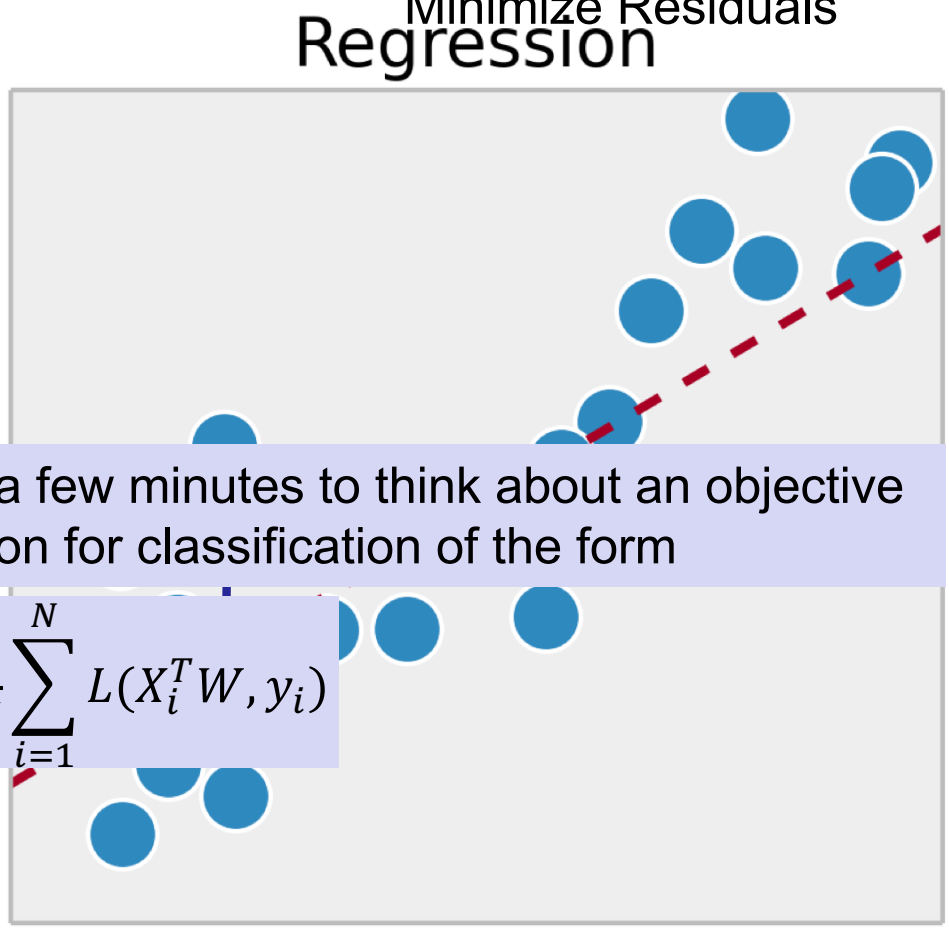
Remaining question: what objective to use?

Convex optimization in ML

$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals





Take a few minutes to think about an objective function for classification of the form

$$J = \frac{1}{N}\sum_{i=1}^{N} L(X_i^T W, y_i)$$

QUIZ: Think of a loss function for Classification

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

Convex optimization in ML: Loss Functions

- **Many methods in machine learning are based on finding parameters that minimize some objective function.**
- **Very often, the objective function is a weighted sum of two terms:**
  - a cost function and regularization term.
  - In statistics terms the (log-)likelihood and (log-)prior.
  - If both of these components are convex, then their sum is also convex.
  - Loss functions are summed over examples so the sum of a convex functions is a convex function

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term (R(w) expressed as a function of the model complexity)

Data Loss term regularization term (aka penalty)

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

f_w

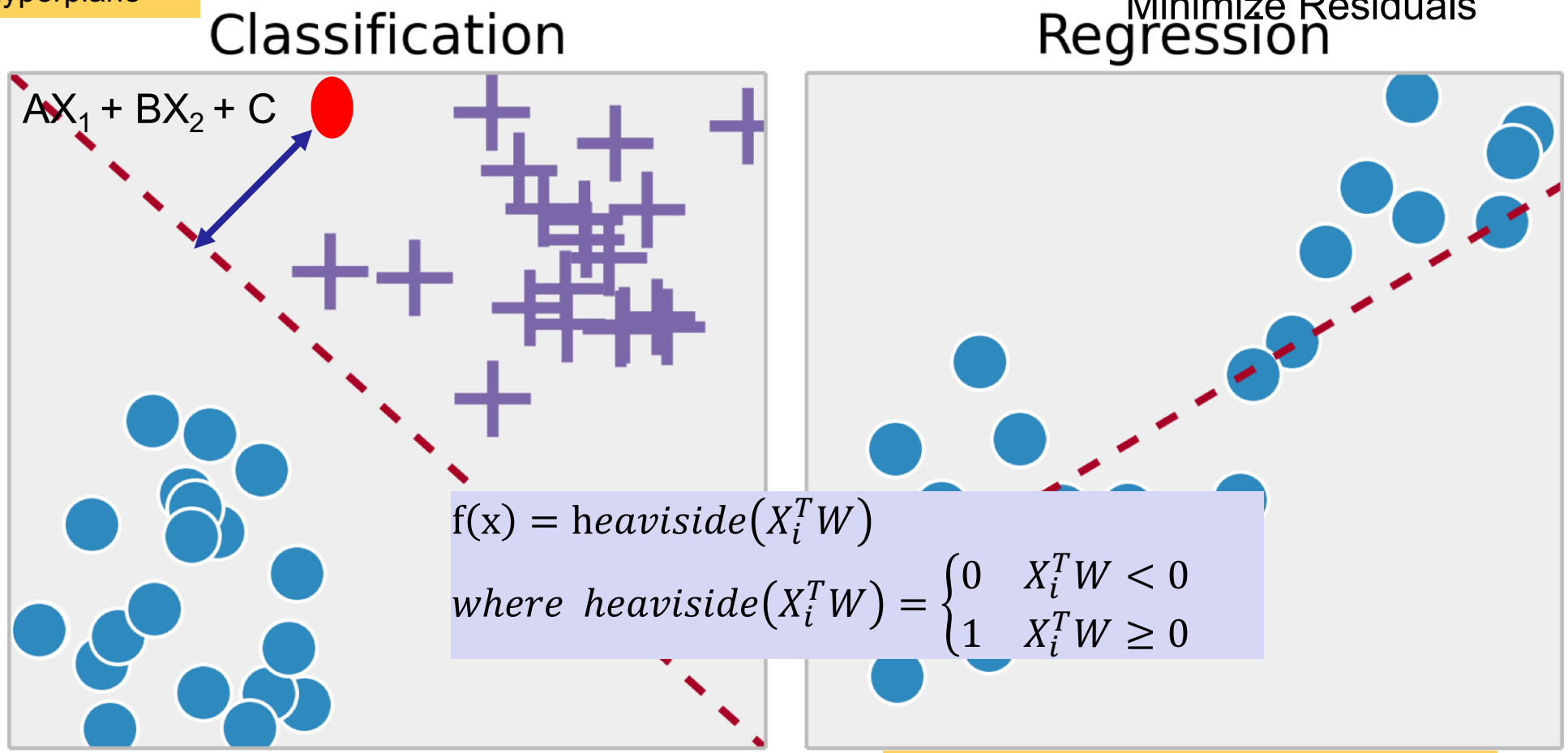
Focus only on the data loss term
E.g., Linear regression, logistic regression Soft SVMs

Convex optimization in ML

Perpendicular Distance from a point to a hyperplane

$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals



$$f(x) = heaviside(X_i^T W)$$

$$where\ \ heaviside(X_i^T W) = \begin{cases} 0 & X_i^T W < 0 \\ 1 & X_i^T W \geq 0 \end{cases}$$

Please type in the loss function for linear regression (W is the model)

Hyperplanes as decision surfaces

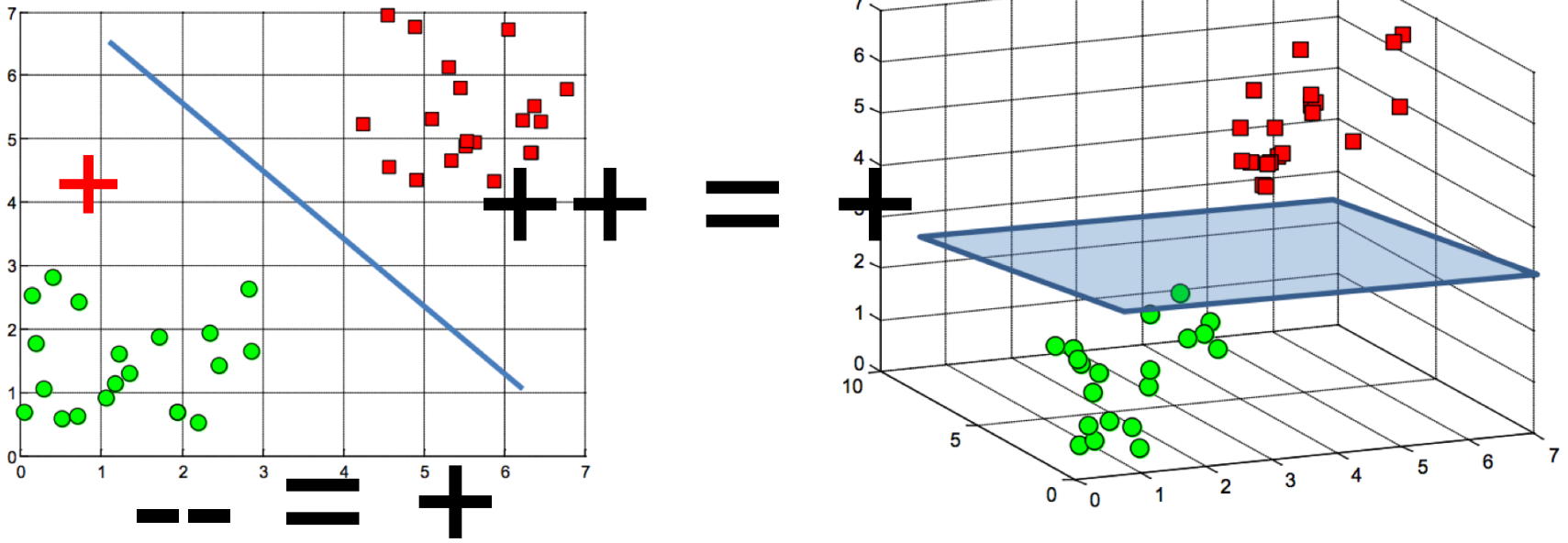
- A hyperplane is a linear decision surface that splits the space into two parts;
- It is obvious that a hyperplane is a binary classifier.

$w_1 x_1 + w_2 x_2 + w0 == 0$ (points on the hyperplane)
$w_1 x_1 + w_2 x_2 + w0 > 0$ (points on +side of hyperplane)
$w_1 x_1 + w_2 x_2 + w0 < 0$ (points on – side of hyperplane)

$w_1 x_1 + w_2 x_2 + w_3 x_3 + w0$

A hyperplane in $\mathbb{E}^2$ is a line

A hyperplane in $\mathbb{E}^3$ is a plane



A hyperplane in $\mathbb{E}^n$ is an n-1 dimensional subspace

Margin: label normalized perpendicular distance

$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals

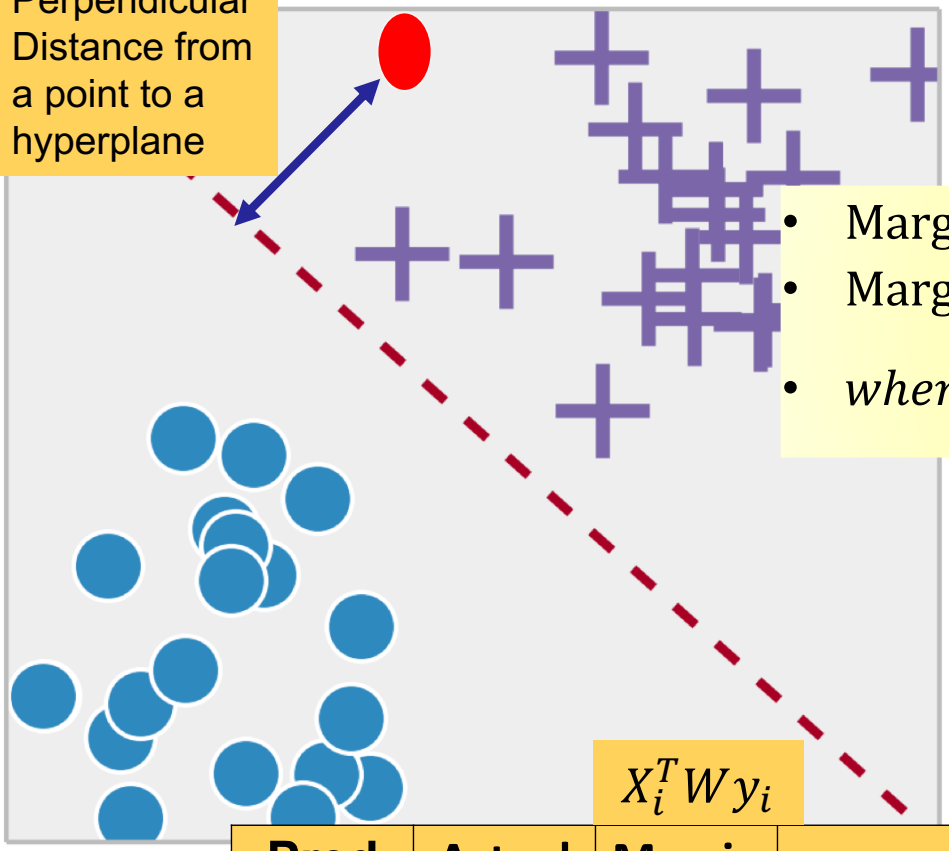
$X_i^T W y_i$

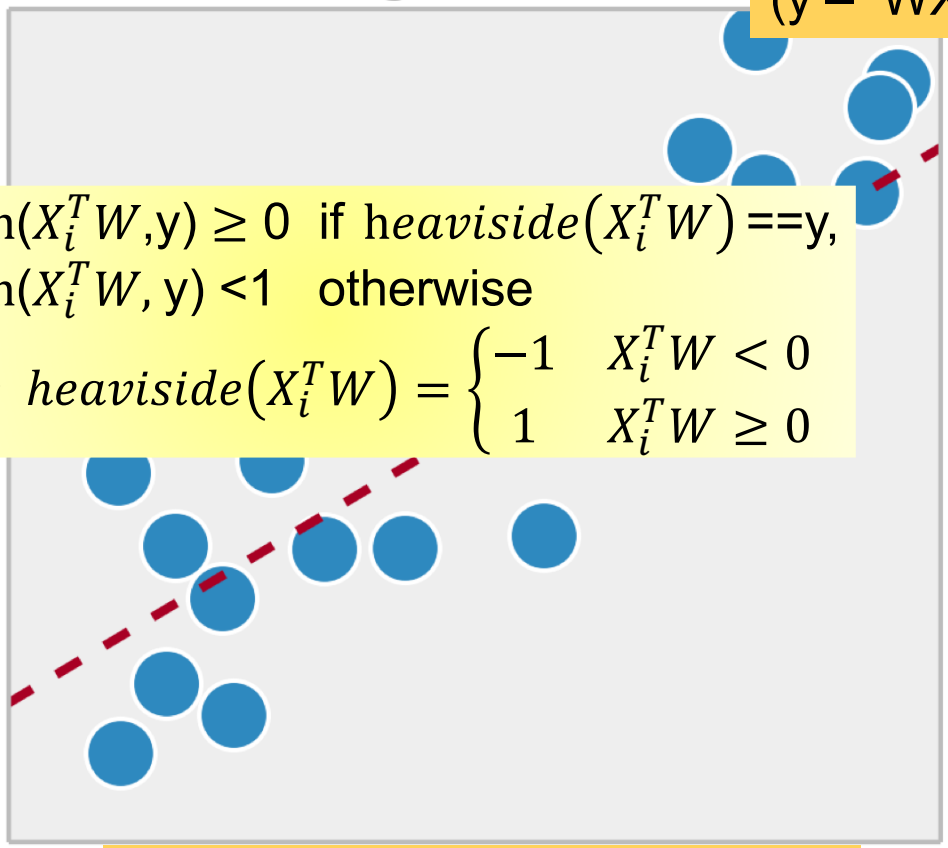
Classification

Regression

$(y - WX)^2$

Perpendicular Distance from a point to a hyperplane





- $\text{Margin}(X_i^T W, y) \geq 0$ if $heaviside(X_i^T W) == y$,
- $\text{Margin}(X_i^T W, y) < 1$ otherwise
- $where\ heaviside(X_i^T W) = \begin{cases} -1 & X_i^T W < 0 \\ 1 & X_i^T W \geq 0 \end{cases}$

$X_i^T W y_i$

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Please type in the loss function for linear regression (W is the model)

Convex optimization in ML

$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

$X_i^T W y_i$

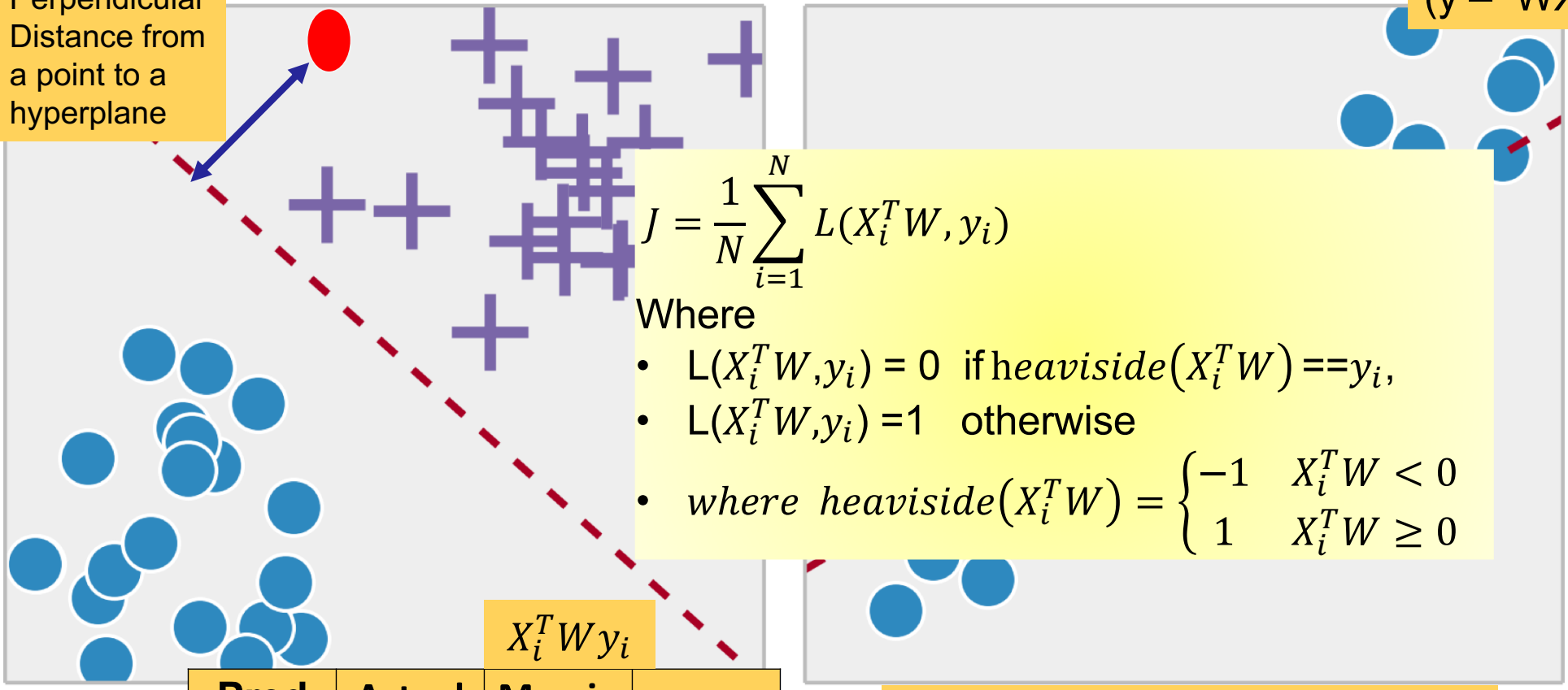
Minimize Residuals

Classification

Regression

$(y - WX)^2$

Perpendicular Distance from a point to a hyperplane



$$J = \frac{1}{N}\sum_{i=1}^{N} L(X_i^T W, y_i)$$

Where

- $L(X_i^T W, y_i) = 0$ if $heaviside(X_i^T W) == y_i$,
- $L(X_i^T W, y_i) = 1$ otherwise
- $where\ \ heaviside(X_i^T W) = \begin{cases} -1 & X_i^T W < 0 \\ 1 & X_i^T W \geq 0 \end{cases}$

$X_i^T W y_i$

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

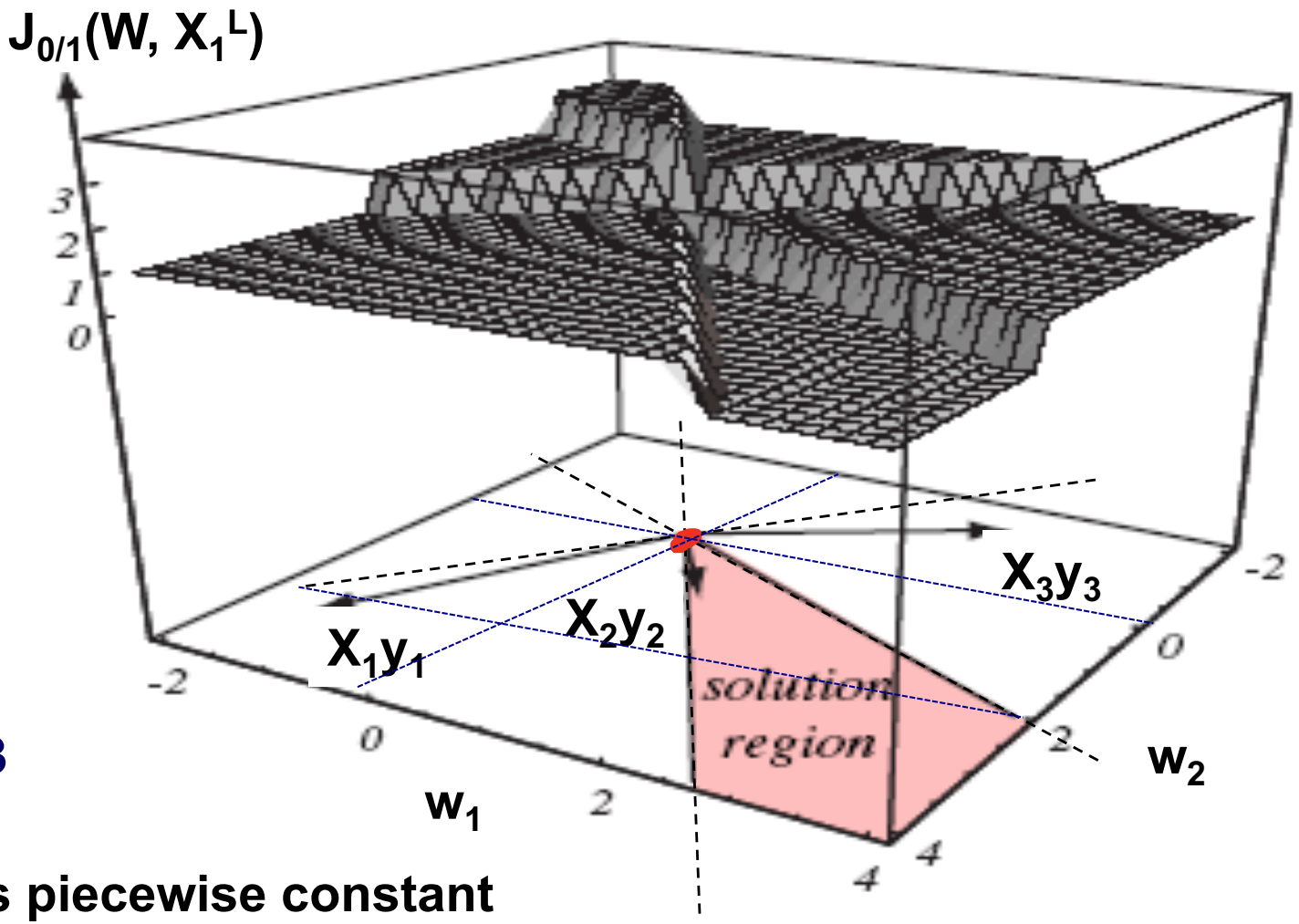
Please type in the loss function for linear regression (W is the model)

0-1 Loss Function: Number of Errors

E.g.	x_1	x_2	y
1	-2	3	+1
2	-1	-1	-1
3	2	-3	1

$J_{0/1}(W, X_1^L)$ = Number of Errors

0-1 Loss Function



3 examples (label normalized, i.e., Xy)
⇒Legal region corresponds to the intersection of positive half-spaces
=> Max $J_{0/1}(W, X_1^L)$ =3

However, $J(W, X_1^L)$ is piecewise constant
=> Very poor candidate for gradient search

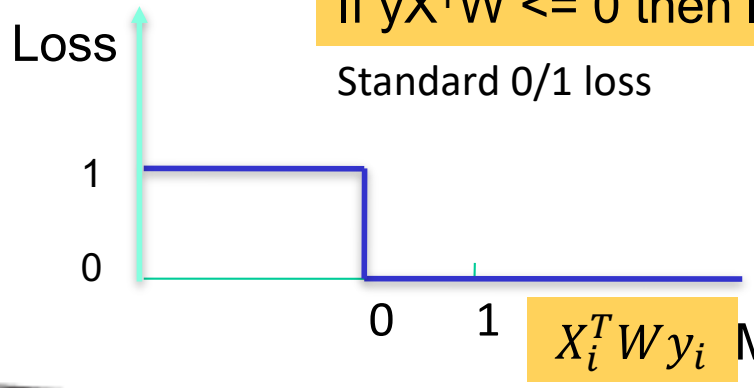
0-1 Loss function

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Penalizes all incorrectly classified examples with the same amount

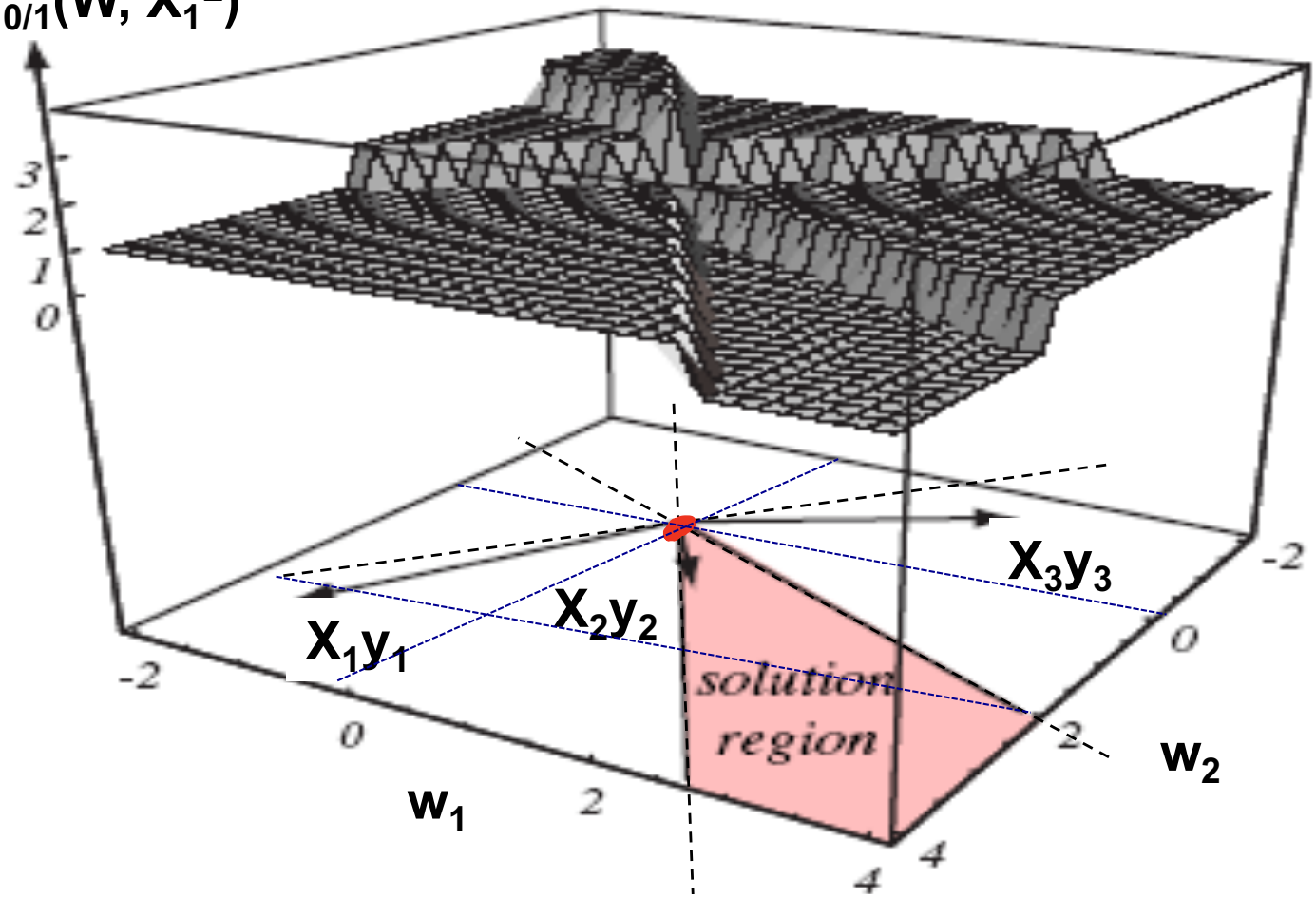
If $yX^TW > 0$ then no loss
If $yX^TW <= 0$ then loss =

Standard 0/1 loss



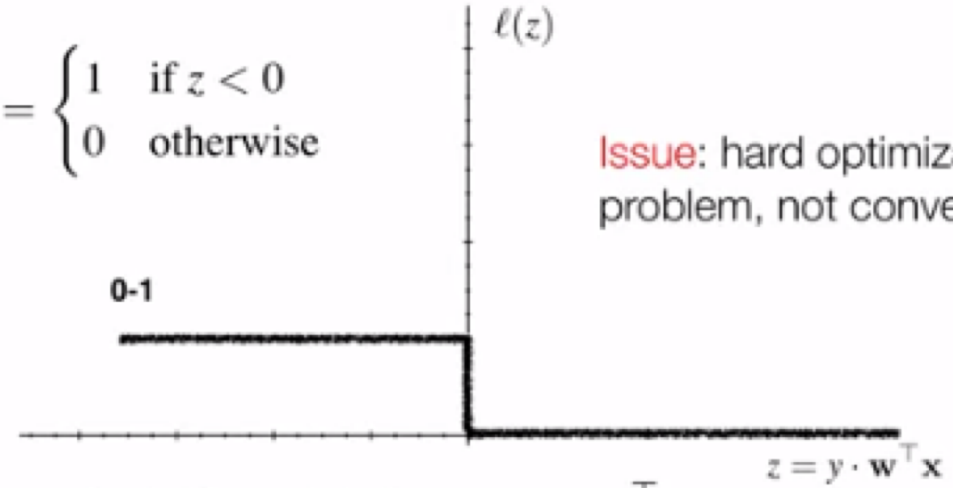
0-1 Loss Function

$J_{0/1}(W, X_1^L)$



0/1 Loss Minimization

$$\ell_{0/1}(z) = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$



Issue: hard optimization problem, not convex!

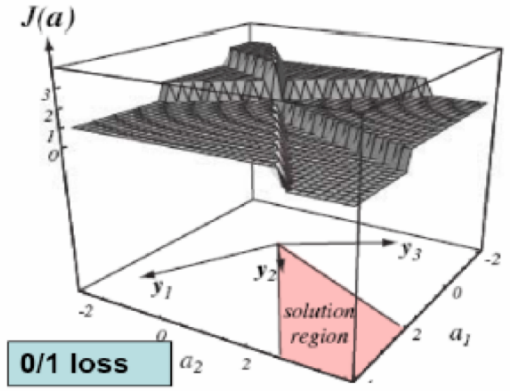
Let $y \in \{-1, 1\}$ and define $z = y \cdot \mathbf{w}^\top \mathbf{x}$

z is positive if y and $\mathbf{w}^\top \mathbf{x}$ have same sign, negative otherwise

0/1 loss is Not Convex
Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables);

- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N}\sum_{i=1}^{N} L(\text{sgn}(w \cdot x_i), y_i)$

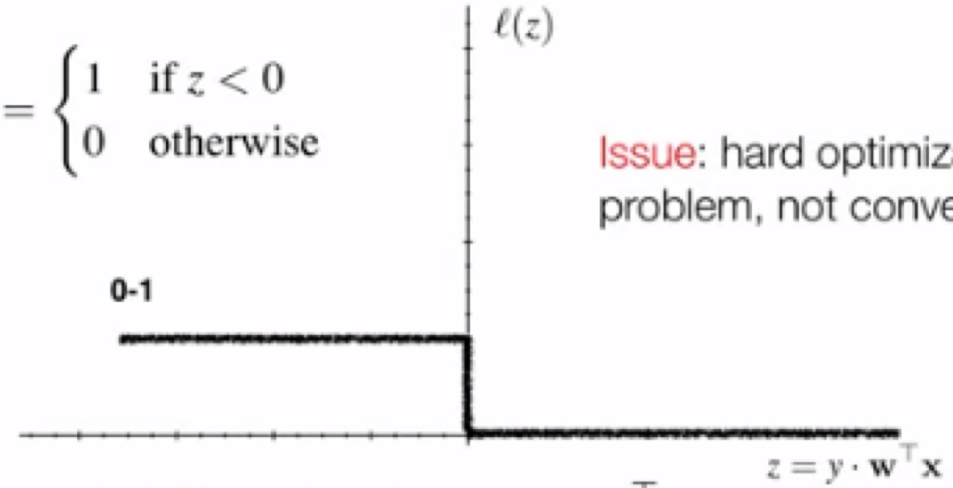
  L(y',y) = 0 when y'=y, otherwise L(y',y)=1
- Does not produce useful gradient since the surface of J is flat



What can we do?
- Approximate 0/1 loss with a convex loss surrogate
- See next slide

0/1 Loss Minimization

$$\ell_{0/1}(z) = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$



Issue: hard optimization problem, not convex!

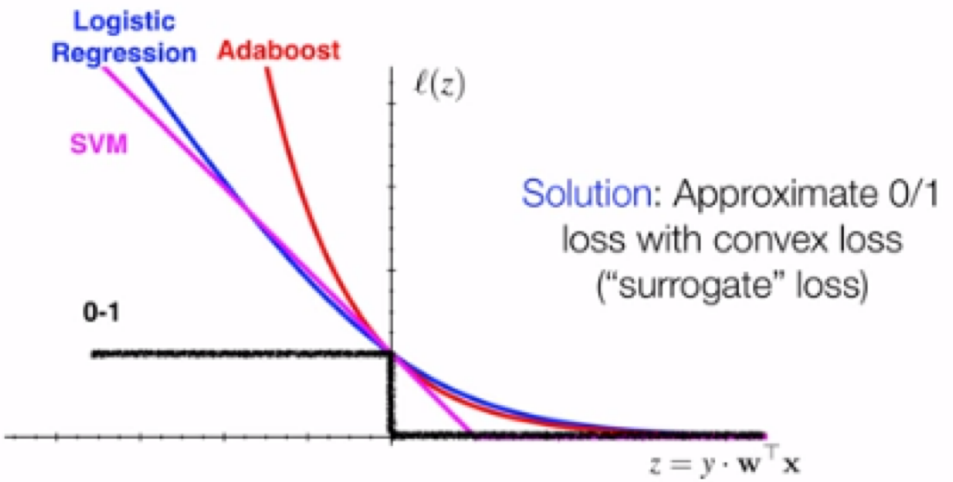
0/1 loss is Not Convex
Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables);

Let $y \in \{-1, 1\}$ and define $z = y \cdot \mathbf{w}^\top \mathbf{x}$

z is positive if y and $\mathbf{w}^\top \mathbf{x}$ have same sign, negative otherwise

Approximate 0/1 Loss

Approximate 0/1 loss with a convex loss surrogate



Solution: Approximate 0/1 loss with convex loss ("surrogate" loss)

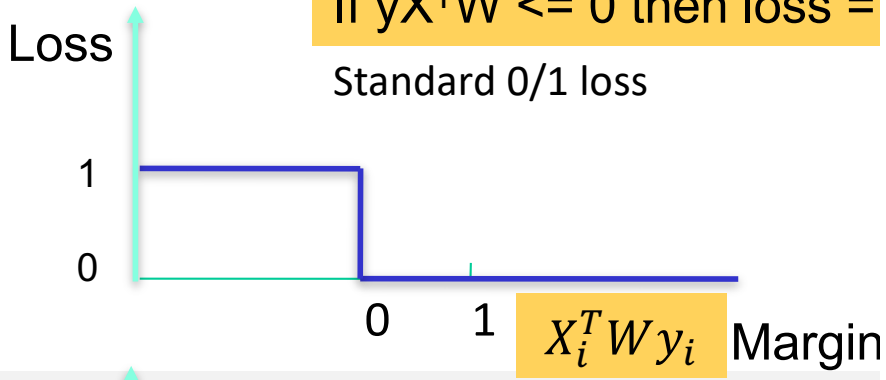
SVM (hinge), Logistic regression (logistic), Adaboost (exponential)

Loss functions

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

If $yX^TW > 0$ then no loss
If $yX^TW <= 0$ then loss =

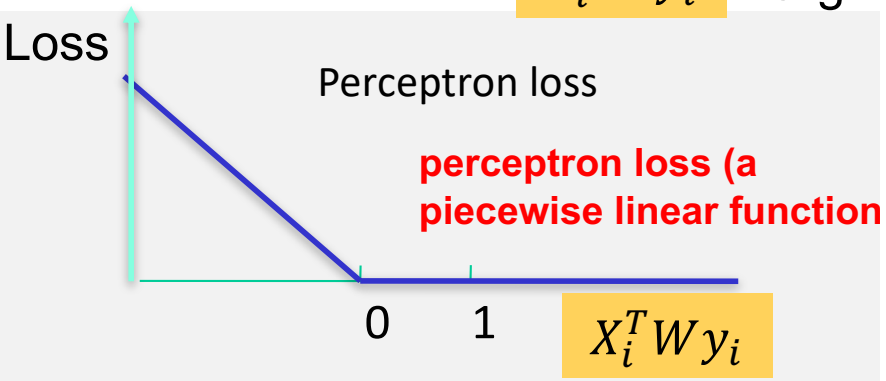
Penalizes all incorrectly classified examples with the same amount



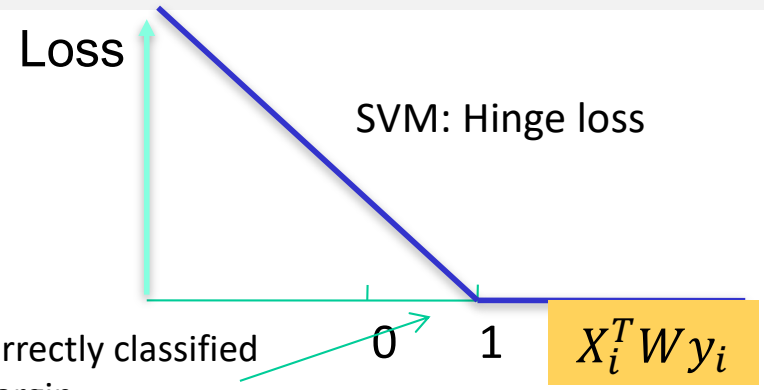
Penalizes incorrectly classified examples X proportionally to the size of $|X^Tw|$

$$J_P = \sum_{i=1}^{N} Ifelse\left(X_i^T W y_i < 0, 1, 0\right) - X_i^T W y_i$$

$$J_P = \frac{1}{N} \sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)$$



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



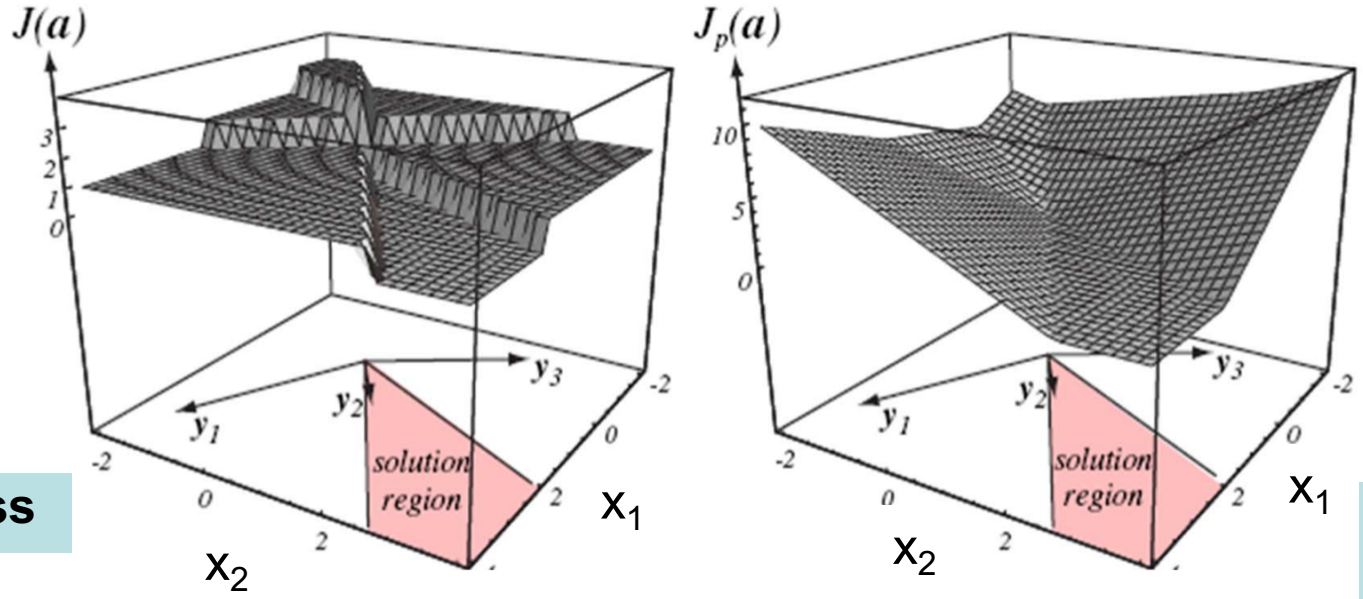
Examples that are correctly classified but fall within the margin

A smoother Loss Functions

- Instead we will consider the "**perceptron criterion**" (a slightly modified version of hinge loss):

$$J_P = \frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)$$

- The term $\boldsymbol{Max\left(0, -X_i^T W y_i\right)}$ s 0 when y_i is predicted correctly otherwise it is equal to the "confidence" in the mis-prediction
- Has a nice gradient leading to the solution region



0/1 loss

Perceptron criterion

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term (R(w) expressed as a function of the model complexity)

Data Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

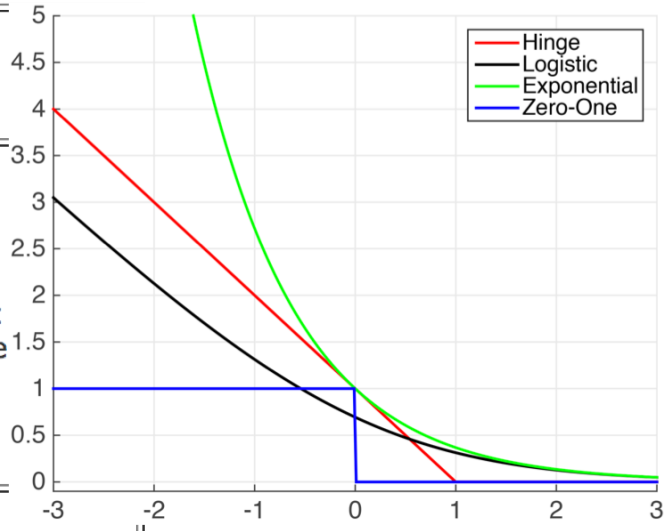
$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

f_w

Focus only on the data loss term
E.g., Linear regression, logistic regression Soft SVMs

Commonly Used Binary Classification Loss Functions

Loss $\ell(h_{\mathbf{w}}(\mathbf{x}_i, y_i))$	Usage	Comments
Hinge-Loss $\max\left[1 - h_{\mathbf{w}}(\mathbf{x}_i)y_i, 0\right]^p$	• Standard SVM($p = 1$) • (Differentiable) Squared Hingeless SVM ($p = 2$)	When used for Standard SVM, the loss function denotes the size of the margin between linear separator and its closest points in either class. Only differentiable everywhere with $p = 2$.
Log-Loss $\log(1 + e^{-h_{\mathbf{w}}(\mathbf{x}_i)y_i})$	Logistic Regression	One of the most popular loss functions in Machine Learning, since its outputs are well-calibrated probabilities.
Exponential Loss $e^{-h_{\mathbf{w}}(\mathbf{x}_i)y_i}$	AdaBoost	This function is very aggressive. The loss of a mis-prediction increases *exponentially* with the value of $-h_{\mathbf{w}}(\mathbf{x}_i)y_i$. This can lead to nice convergence results, for example in the case of Adaboost, but it can also cause problems with noisy data.
Zero-One Loss $\delta(\text{sign}(h_{\mathbf{w}}(\mathbf{x}_i)) \neq y_i)$	Actual Classification Loss	Non-continuous and thus impractical to optimize.



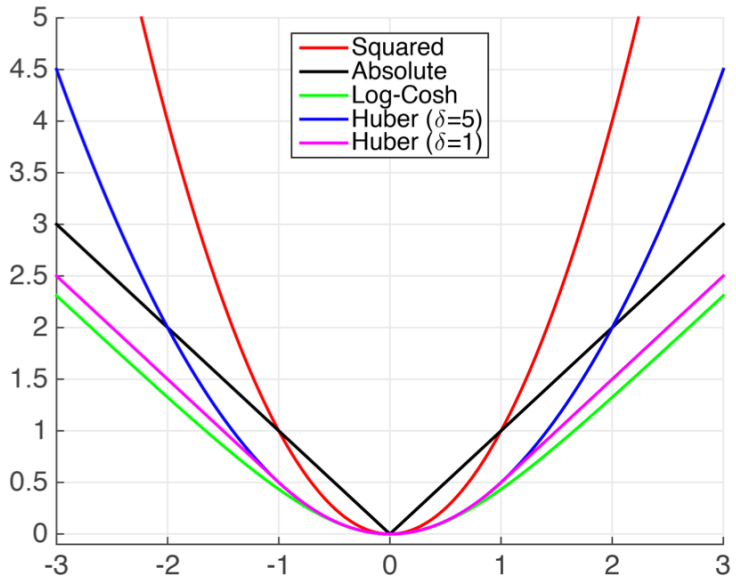
http://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote10.html

Table 4.1: Loss Functions With Classification $y \in \{-1, +1\}$

Commonly Used Regression Loss Functions

Loss $\ell(h_{\mathbf{w}}(\mathbf{x}_i, y_i))$	Comments
Squared Loss $(h(\mathbf{x}_i) - y_i)^2$	◦ Most popular regression loss function ◦ Estimates Mean Label ◦ ADVANTAGE: Differentiable everywhere ◦ DISADVANTAGE: Somewhat sensitive to outliers/noise ◦ Also known as Ordinary Least Squares (OLS)
Absolute Loss $\lvert h(\mathbf{x}_i) - y_i \rvert$	◦ Also a very popular loss function ◦ Estimates Median Label ◦ ADVANTAGE: Less sensitive to noise ◦ DISADVANTAGE: Not differentiable at 0
Huber Loss ◦ $\frac{1}{2}(h(\mathbf{x}_i) - y_i)^2$ if $\lvert h(\mathbf{x}_i) - y_i \rvert < \delta$, ◦ otherwise $\delta(\lvert h(\mathbf{x}_i) - y_i \rvert - \frac{\delta}{2})$	◦ Also known as Smooth Absolute Loss ◦ ADVANTAGE: "Best of Both Worlds" of Squared and Absolute Loss ◦ Once-differentiable ◦ Takes on behavior of Squared-Loss when loss is small, and Absolute Loss when loss is large.
Log-Cosh Loss $log(cosh(h(\mathbf{x}_i) - y_i))$, $cosh(x) = \frac{e^x + e^{-x}}{2}$	ADVANTAGE: Similar to Huber Loss, but twice differentiable everywhere

Table 4.2: Loss Functions With Regression, i.e. $y \in \mathbb{R}$

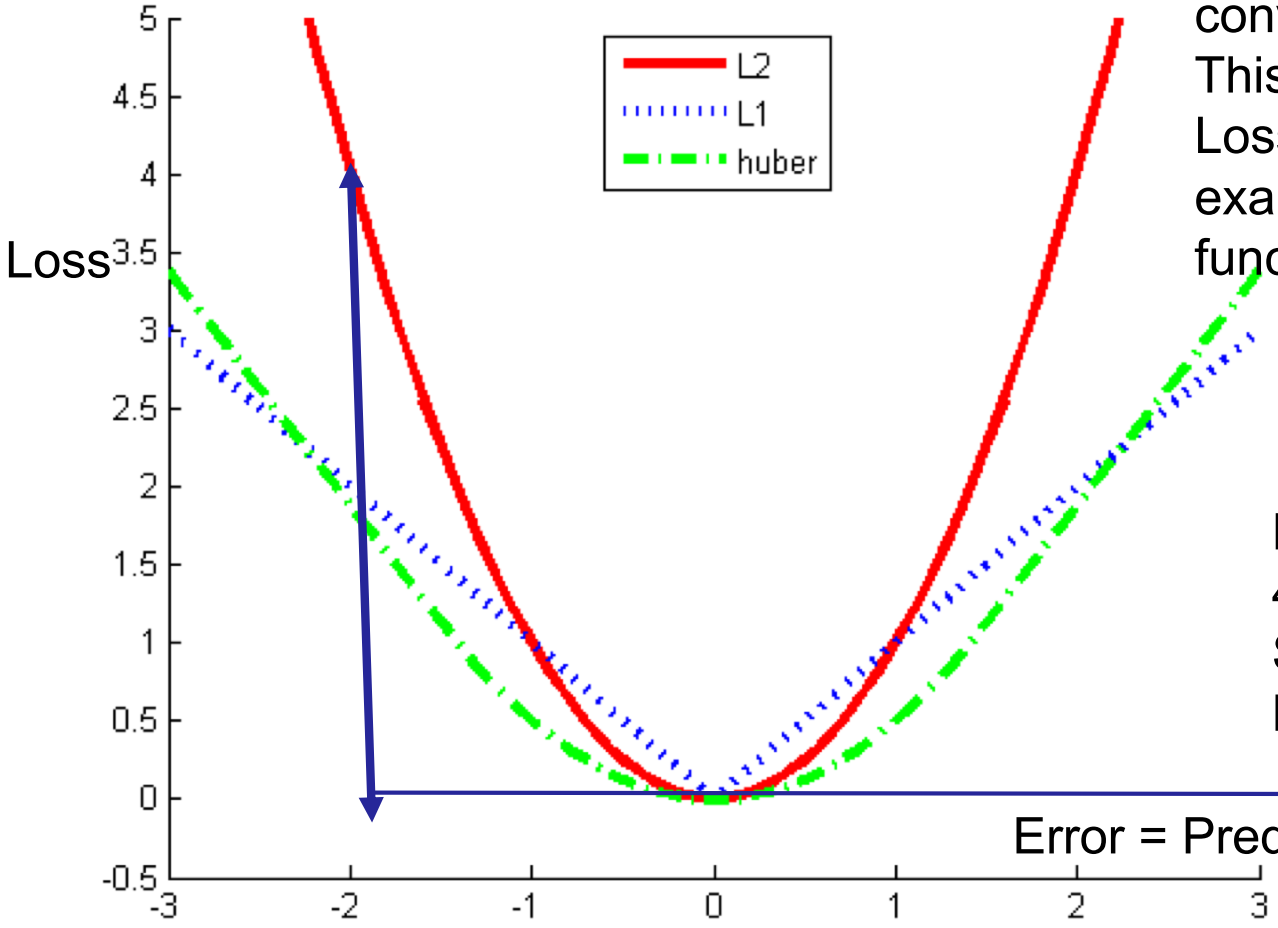


http://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote10.html

Machine Learning Objective Function: regression loss function

Model: $y = \theta^T x$

Goal: $\theta^* = \min_\theta \sum_{i=1}^{m} (\theta^T x_i - y_i)^2$



Regression loss functions: all are convex

This is for one example in 1.7Dim;

Loss: combine over all training examples; sum of such convex functions is also convex

Predicted – actual

4 - 2 = difference

Squared Error = 4

L1 = |-2| = 2

Error = Predicted - actual

Flexibility of "loss + penalty" framework

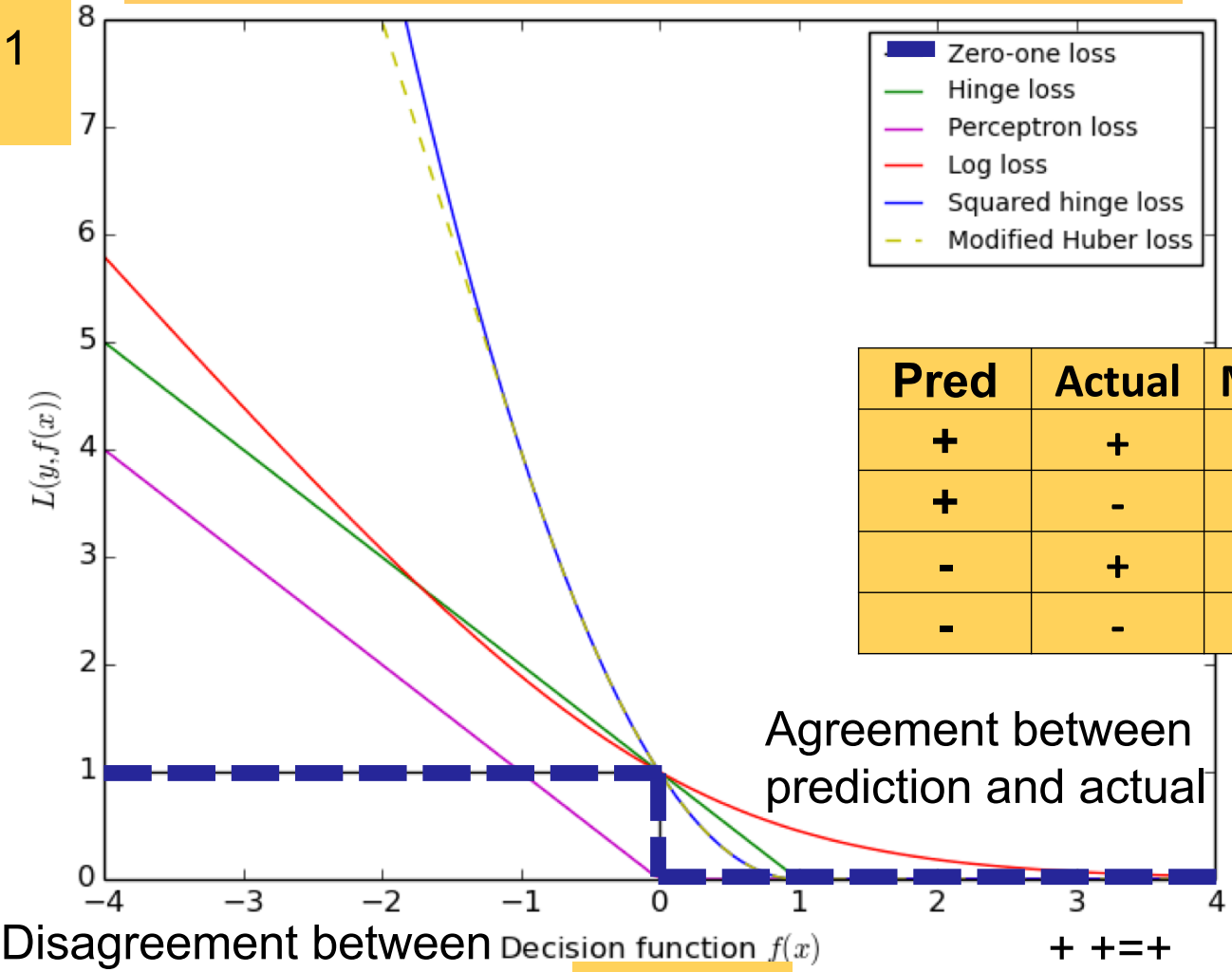
Minimize (*Loss* + λ *Penalty*)

Loss function	Penalty function	Resulting algorithm
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_2^2$	SVMs
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_2^2$	Ridge regression
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_1$	Lasso
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda_1\|\vec{w}\|_1 + \lambda_2\|\vec{w}\|_2^2$	Elastic net
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_1$	1-norm SVM

Machine Learning Objective Function: classification

Encode classes as
+1 versus -1
1 versus 0

Classification loss functions



Loss

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Agreement between prediction and actual

Disagreement between prediction and actual

+ +=+

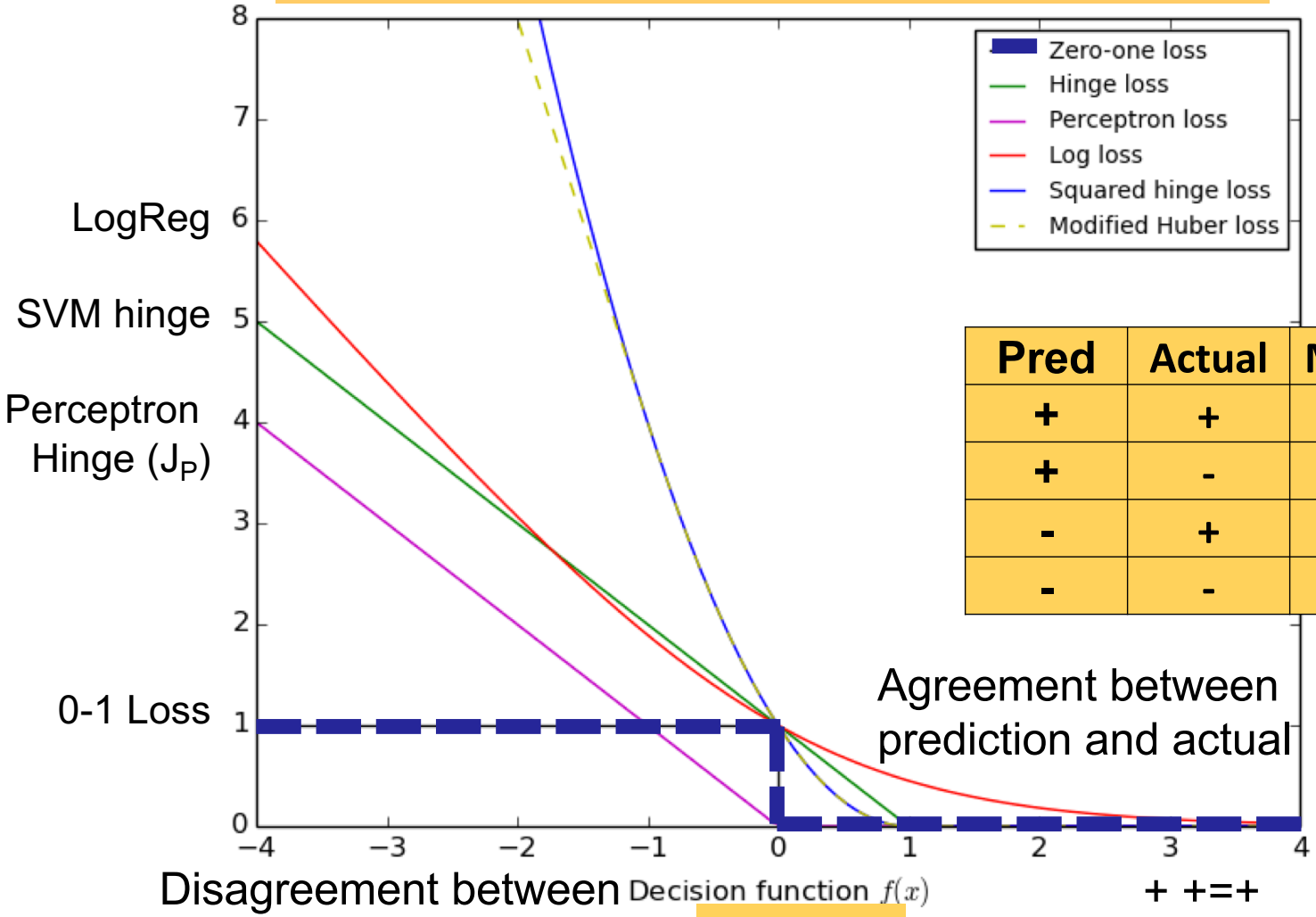
y X f(X)
y X $X^T W$

Margin

Machine Learning Objective Function: classification

$$J_P = mimimize\left[\frac{1}{N}\sum_{i=1}^{N} Max(0, -X_i^T W y_i)\right]$$

Classification loss functions



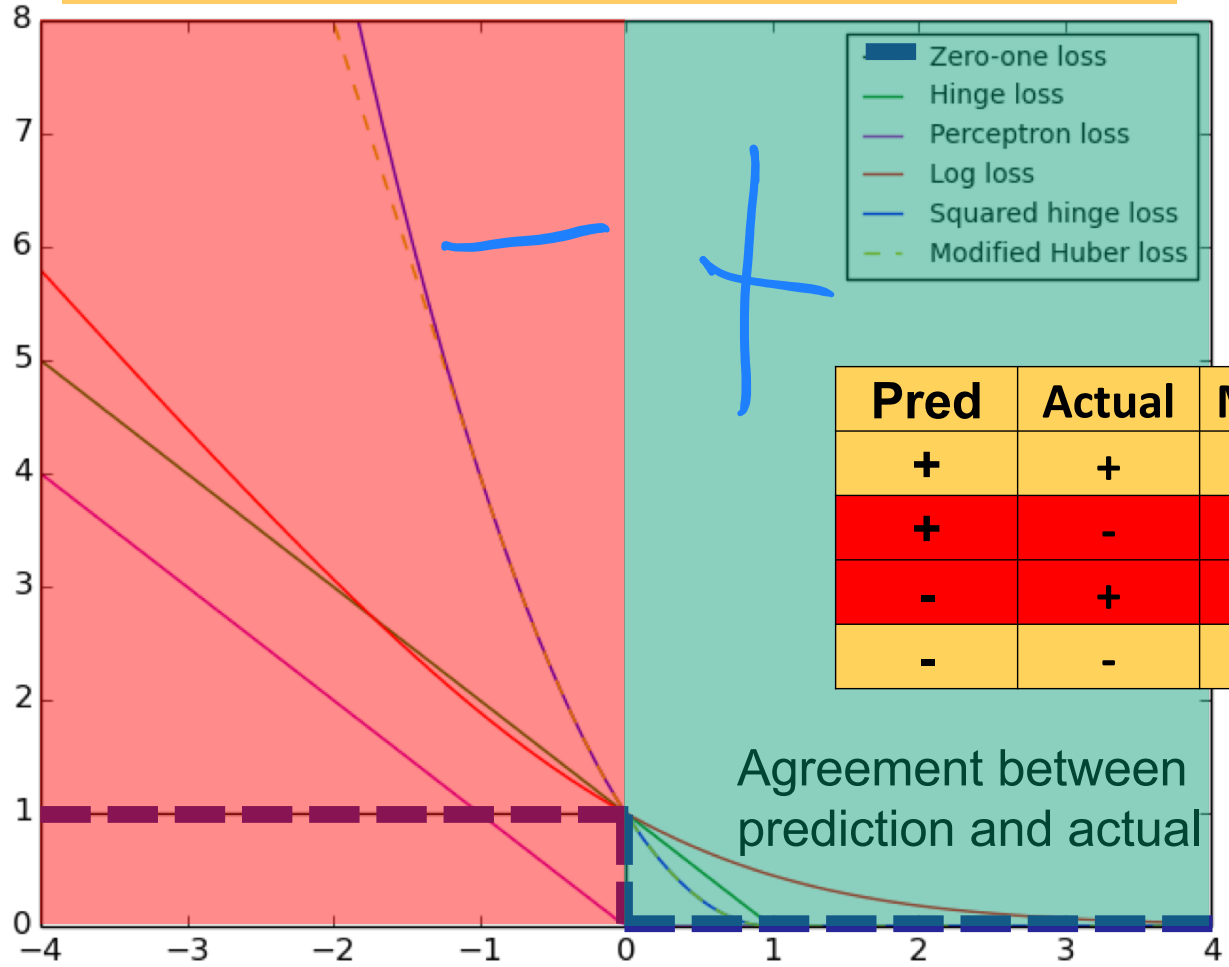
Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

y ✕ f(X)
y X $X^T W$

Margin

Machine Learning Objective Function: classification

Classification loss functions



Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term (R(w) expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

$$y^{(i)} \in \{-1, 1\} \quad (14.3)$$

$$f_w(x^{(i)}) = w^T x^{(i)} \quad (14.4)$$

The entire loss is Twice differentiable BUT sometimes either one of these maybe not be twice differentiable everywhere

- **Let's build on this and reinvent the perceptron and its corresponding learning algorithm**

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

Augment; Encode the class; Learn

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	-1
2					+1
…	…	…	…	…	…
L (aka m)	0	4	...	8	-1

Instance\Attr	x_0	x_1	x_2	…	x_n	y
1	1	3	0	..	7	-1
2	1					+1
…	1	…	…	…	…	…
L (aka m)	1	0	4	...	8	-1

Class Encodings

- {0, 1} [Logistic regression] versus {-1, 1} [Perceptron, SVM, NN]
- For multiple classes we will do a OHE (one-hot-encoding of the target variable

Learn a linear function $g(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} = <w_0, w_1, w_2, w_3, w_4>$

- Each **w** corresponds to one hypothesis
  $h(\mathbf{x}) = \text{sign}(g(\mathbf{x},\mathbf{w}))$
- A prediction is correct if $y\,\mathbf{w}^T\mathbf{x}>0$
- Goal of learning is to find a good **w**
  - e.g., a **w** such that $h(\mathbf{x})$ makes few mis-predictions

Learning a weight vector, w:

An Optimization Problem

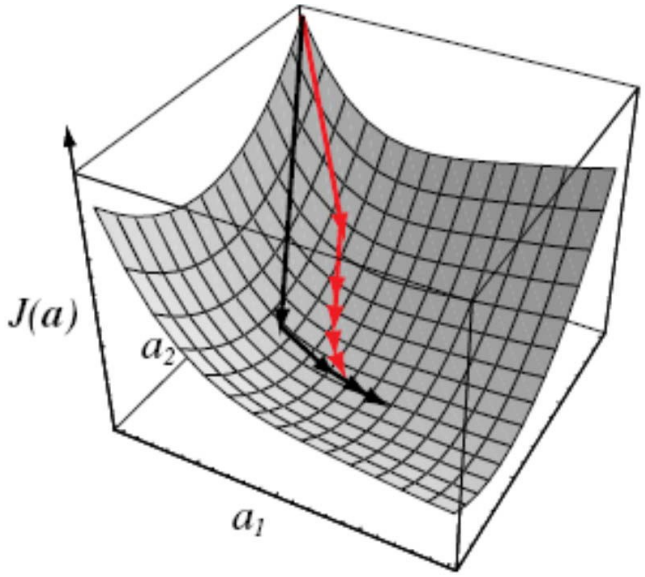
- Formulate learning problem as an optimization problems
  - Given:
    - A set of *N* training examples

      $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$
    - A loss function *L*
  - Find the weight vector **w** that minimizes the objective function - the expected/average loss on training data

$$J = \frac{1}{N}\sum_{i=1}^{N} L(X_i^T W, y_i)$$

- Many machine learning algorithms apply some optimization algorithm to find a good hypothesis.

Gradient Descent Search



- Start with initial **w** = (w_0,..., w_n)
- Compute gradient $\nabla J(\mathbf{w}_0) = (\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_n})_{\mathbf{w}_0}$

- $W_{t+1} = W_t - \eta \nabla J(W_t)$ Where η is the "step size" parameter
- Repeat until convergence

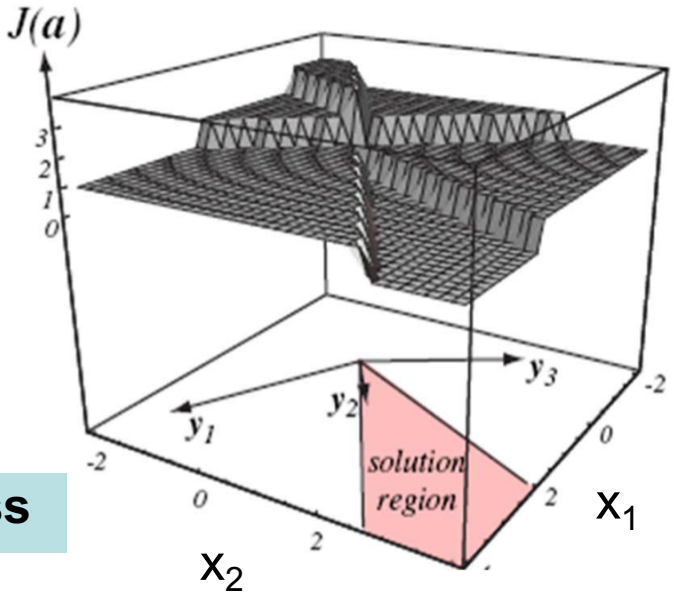
Remaining question: what objective to use?

Loss Functions

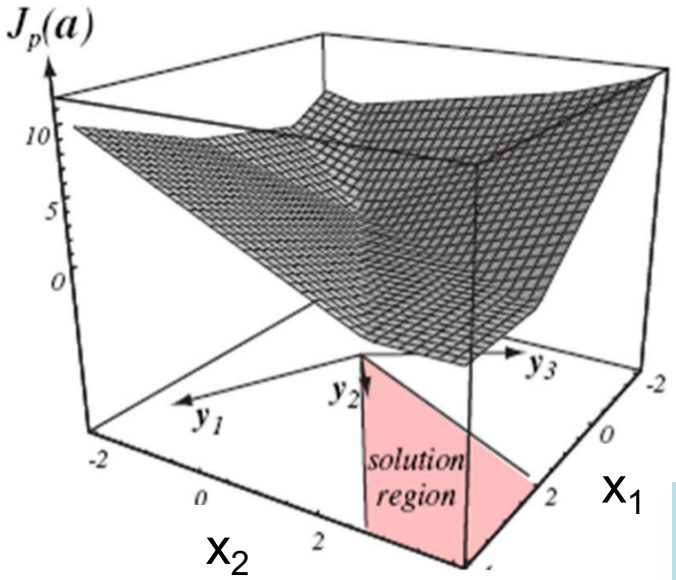
- Instead we will consider the "**perceptron criterion**" (a slightly modified version of hinge loss):

$$J_P = mimimize\left[\frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)\right]$$

- The term $\boldsymbol{Max\left(0, -X_i^T W y_i\right)}$ s 0 when y_i is predicted correctly otherwise it is equal to the "confidence" in the mis-prediction
- Has a nice gradient leading to the solution region



0/1 loss



Perceptron criterion

Deriving the stochastic gradient descent algorithm for the Perceptron

- The objective function consists of a sum over data points--- we can update the parameter after observing each example
- This is referred to as Stochastic gradient descent approach

1. $J_P(W) = Minimize\left[\frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)\right]$
2. **Focus on one example**: $X_i \ , y_i$
3. $J_P(W) = Max\left(0, -X_i^T W y_i\right)$
4. $$\frac{\partial J_P(W)}{\partial w_j} = \begin{cases} 0 & if\ X_i^T W y_i > 0 \\ -X_{ij}^T y_i & otherwise \end{cases}$$
5. $$\nabla J_P(W) = \begin{cases} 0 & if\ X_i^T W y_i > 0 \\ -X_i^T y_i & otherwise \end{cases}$$

Gradient is a vector of partial derivative

After observing $\mathbf{x}_i, y_i$, if we make a mistake $\mathbf{W} \leftarrow \mathbf{W} - \mathbf{X}_i y_i$

$$W = W - \eta \times \nabla J_P(W)$$
$$W = W - \eta \times (-X_i^T y_i)$$
$$W = W + \eta \times X_i^T y_i$$

In perceptron learning rule, learning rate η=1

Online Perceptron Algorithm

$$W = W - \eta \times \nabla J_P(W)$$
$$W = W - \eta \times (-X_i^T y_i)$$
$$W = W + \eta \times X_i^T y_i$$

Let w = (0,0,0,...,0). #init weight vector
Repeat until no mistakes

- Permute trainingSet
- for (X_i , y_i) in trainingSet:

  if $(X_i^T w y_i \leq 0)$: w = w + $y_i \mathbf{x}_i$

Online learning refers to the learning mode in which the model update is performed each time a single observation is received.

Batch learning in contrast performs model update after observing the whole training set.

Gradient functions have similar structure

- **Linear regression**
  - Objective function : $\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$
  - Gradient for Linear Regression:

$$\frac{\partial}{\partial \theta_j}\mathrm{MSE}(\theta) = \frac{2}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

Weighted sum of the training data

- **Classification**
  - Gradient for Binomial Logistic Regression
    - $(\Pr(y_i | X_i; W) - y_i)\, X_i$

$$\frac{\partial}{\partial \theta_j}\mathrm{J}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T \cdot \mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Weighted sum of the training data

  - Gradient for Perceptron (for errors):
    - 1. $J_P(W) = \frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)$
    - 2. $\nabla J_P(W) = \begin{cases} 0 & if\ X_i^T W y_i > 0 \\ -X_i^T y_i & otherwise \end{cases}$

Unweighted sum of the mistakes

- **Next up, let's look at the mechanics of the perceptron learning algorithm more closely geometrically speaking**

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

Online Perceptron Algorithm

$$W = W - \eta \times \nabla J_P(W)$$
$$W = W - \eta \times (-X_i^T y_i)$$
$$W = W + \eta \times X_i^T y_i$$

Let w = (0,0,0,...,0). #init weight vector
Repeat until no mistakes

- Permute trainingSet
- for (X_i , y_i) in trainingSet:

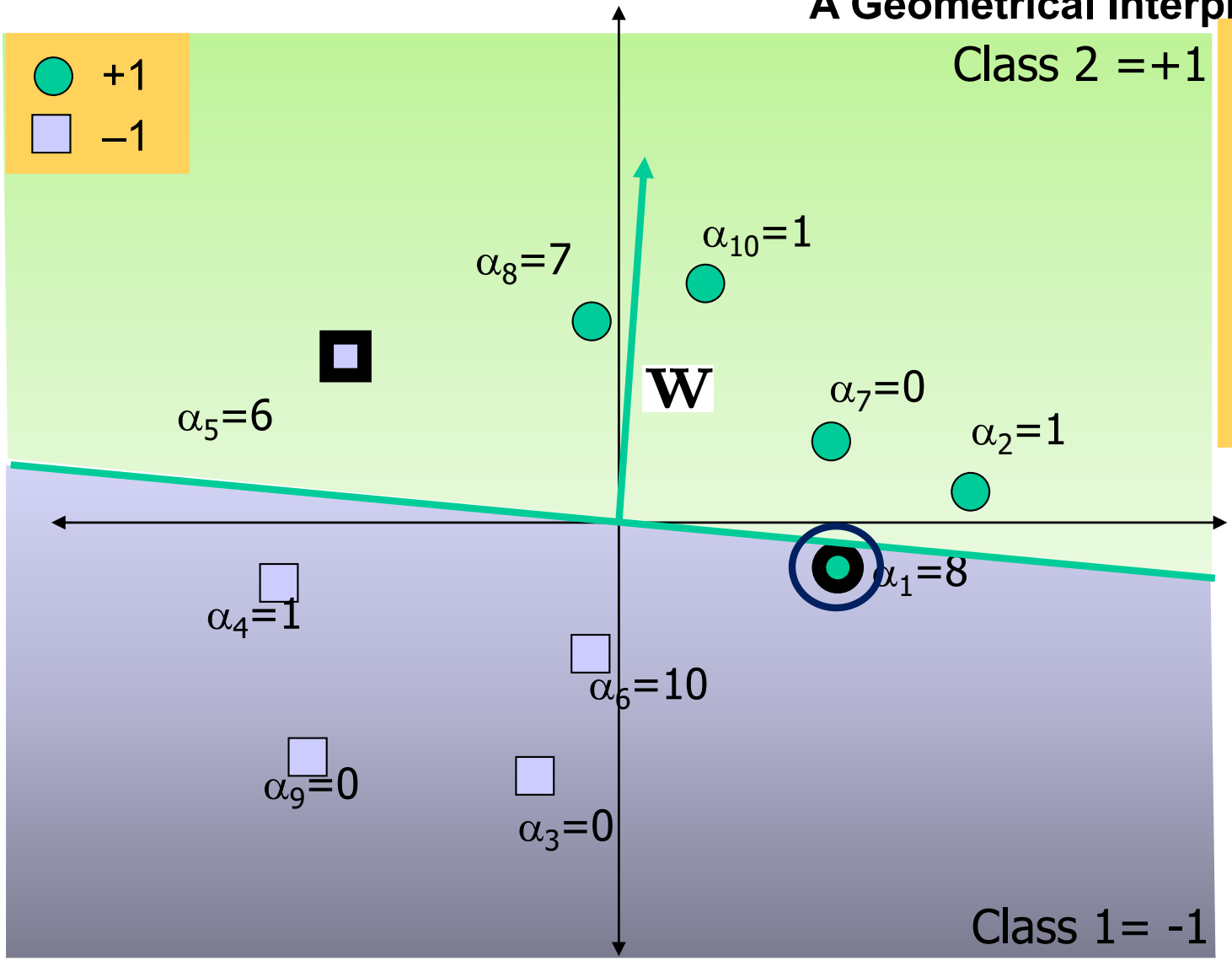
  if $(X_i^T w y_i \leq 0)$: w = w + $y_i \mathbf{x}_i$

Online learning refers to the learning mode in which the model update is performed each time a single observation is received.

Batch learning in contrast performs model update after observing the whole training set.

Perceptron Learning

A Geometrical Interpretation



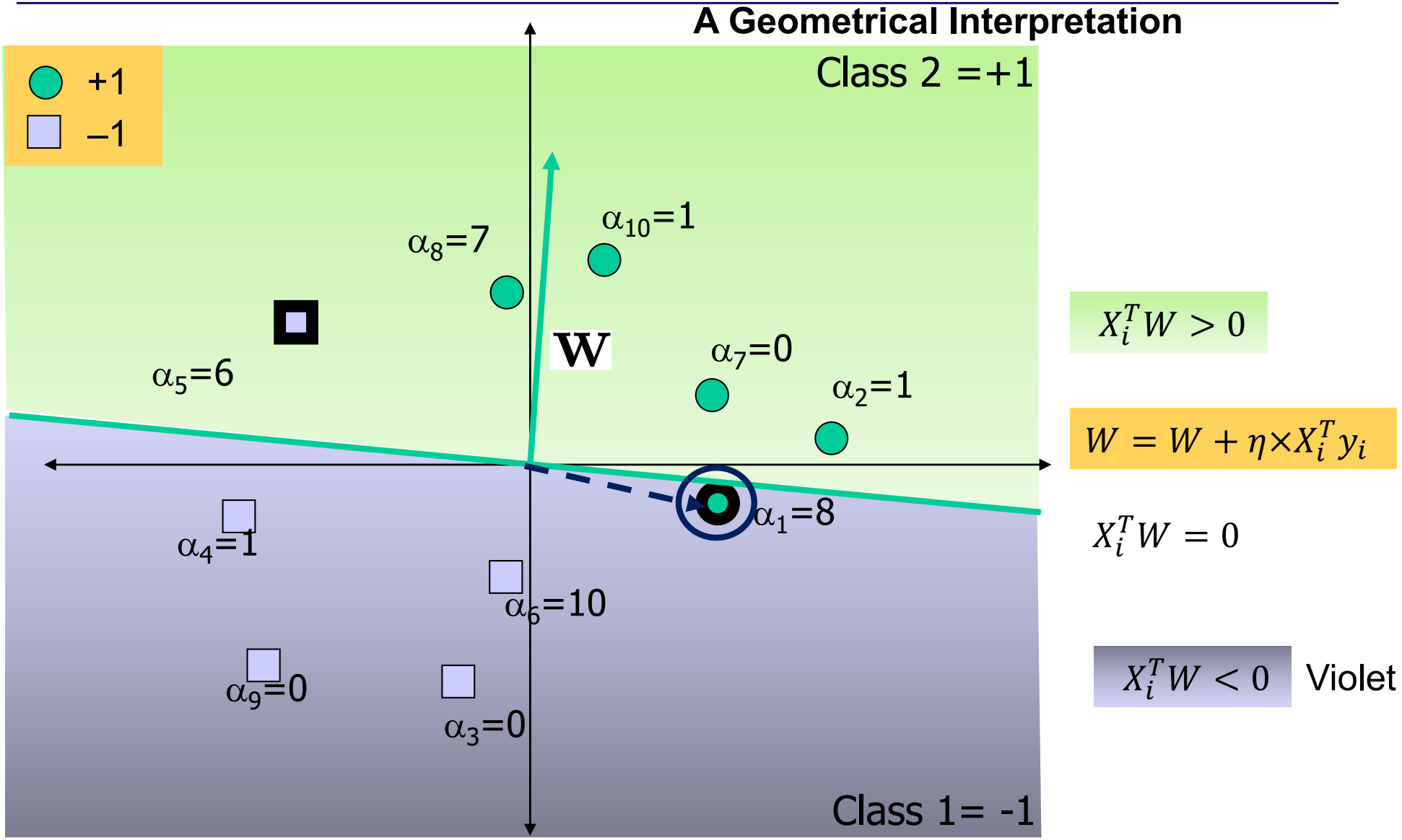
NOTE: α_i act as counters for count the number of times this example was used to update the W
ASSUME: Augmented dataset so hyperplane will go thru origin

$X_i^T W > 0$

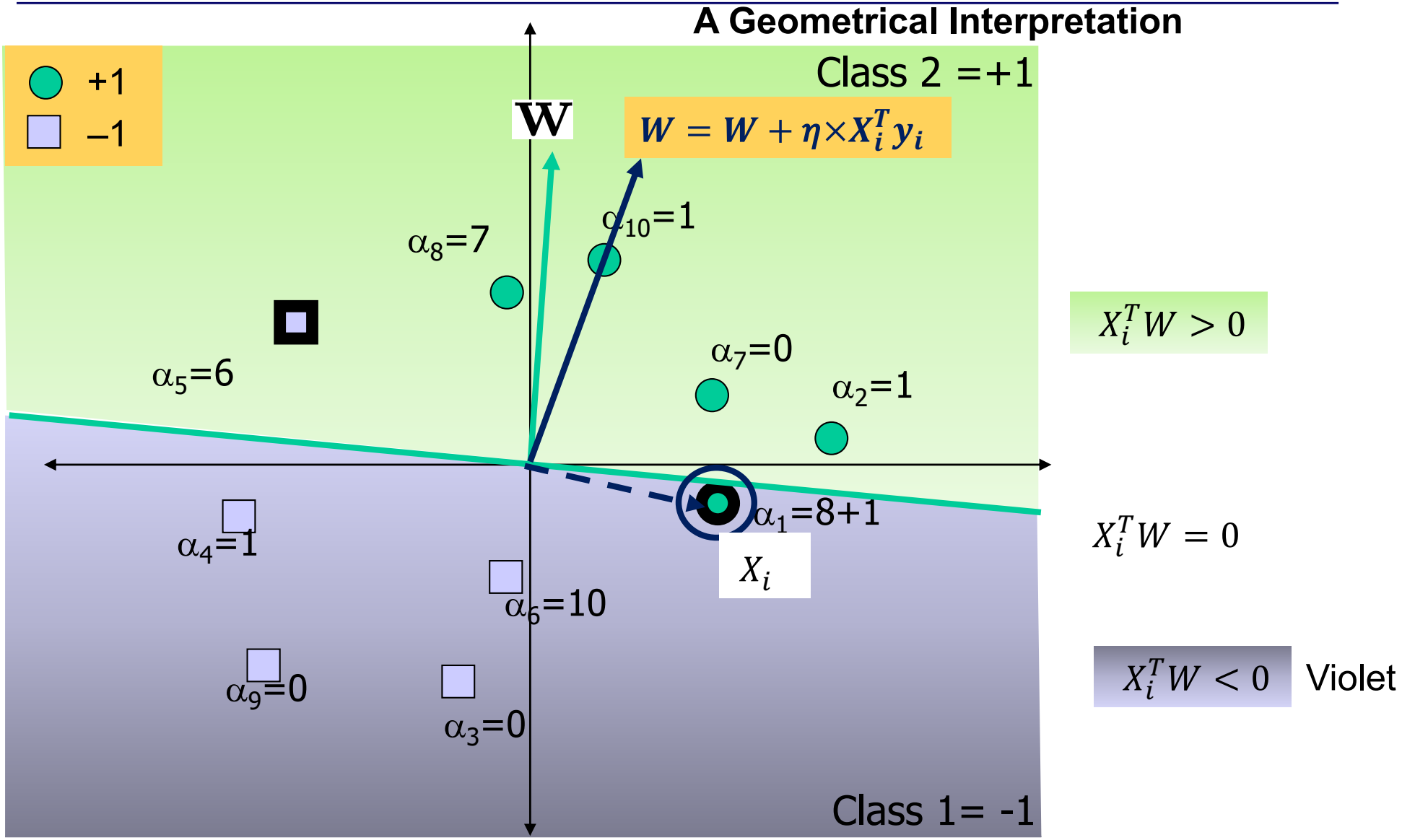
$X_i^T W = 0$

$X_i^T W < 0$ Violet

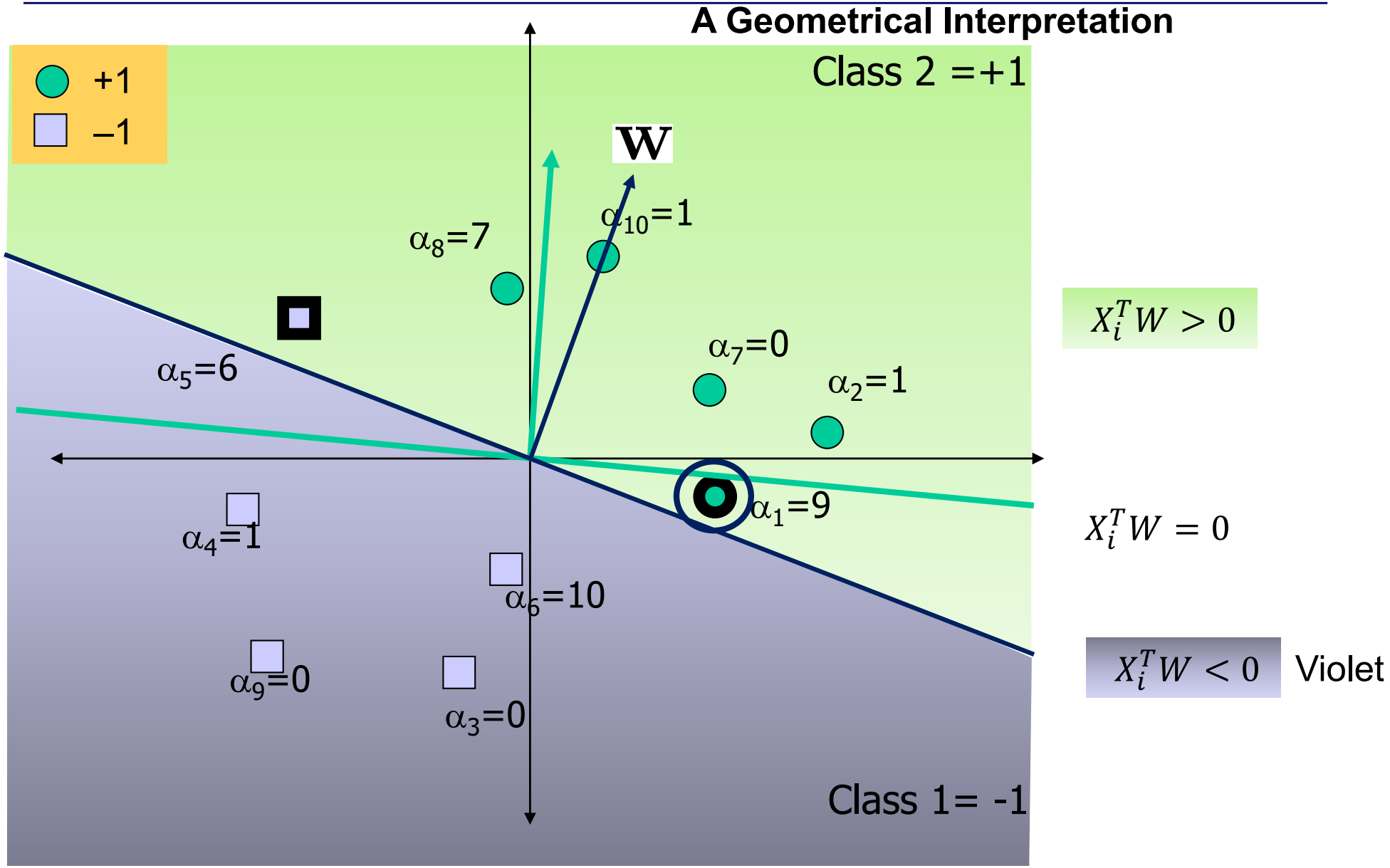
Perceptron Learning: mistake for +1 example



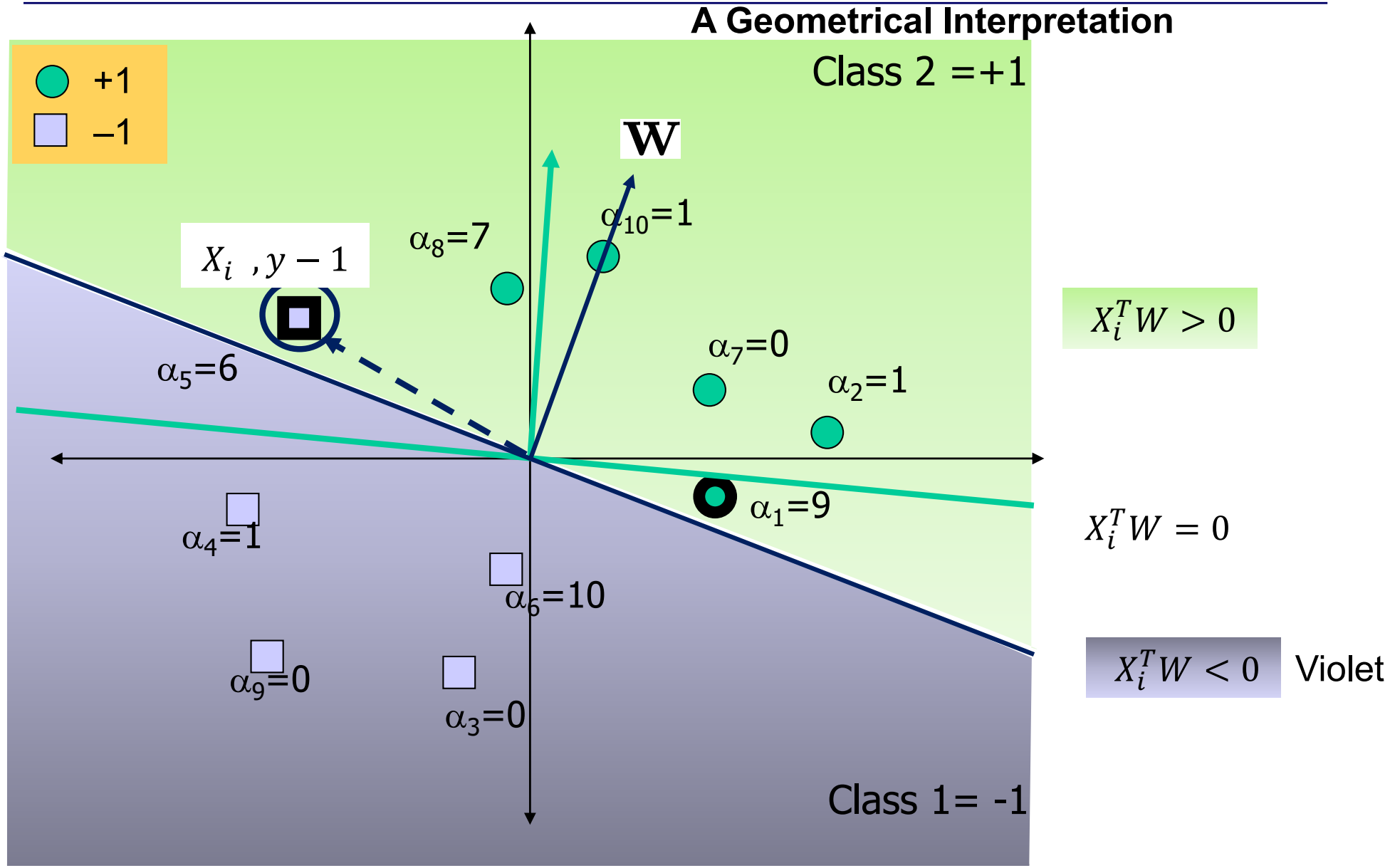
Perceptron Learning: mistake for +1 example



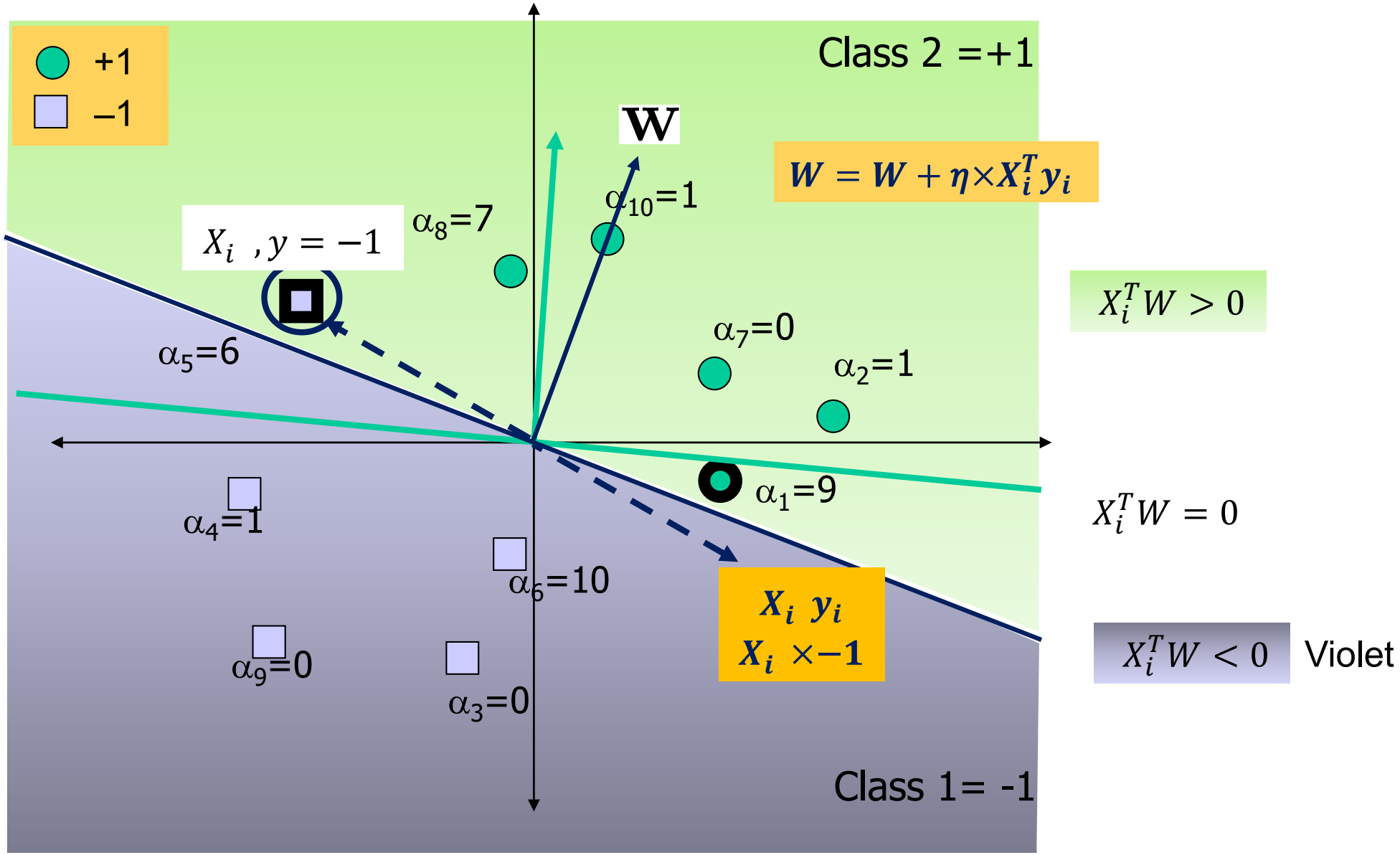
Perceptron Learning: mistake for +1 example



Perceptron Learning: mistake for –1 example

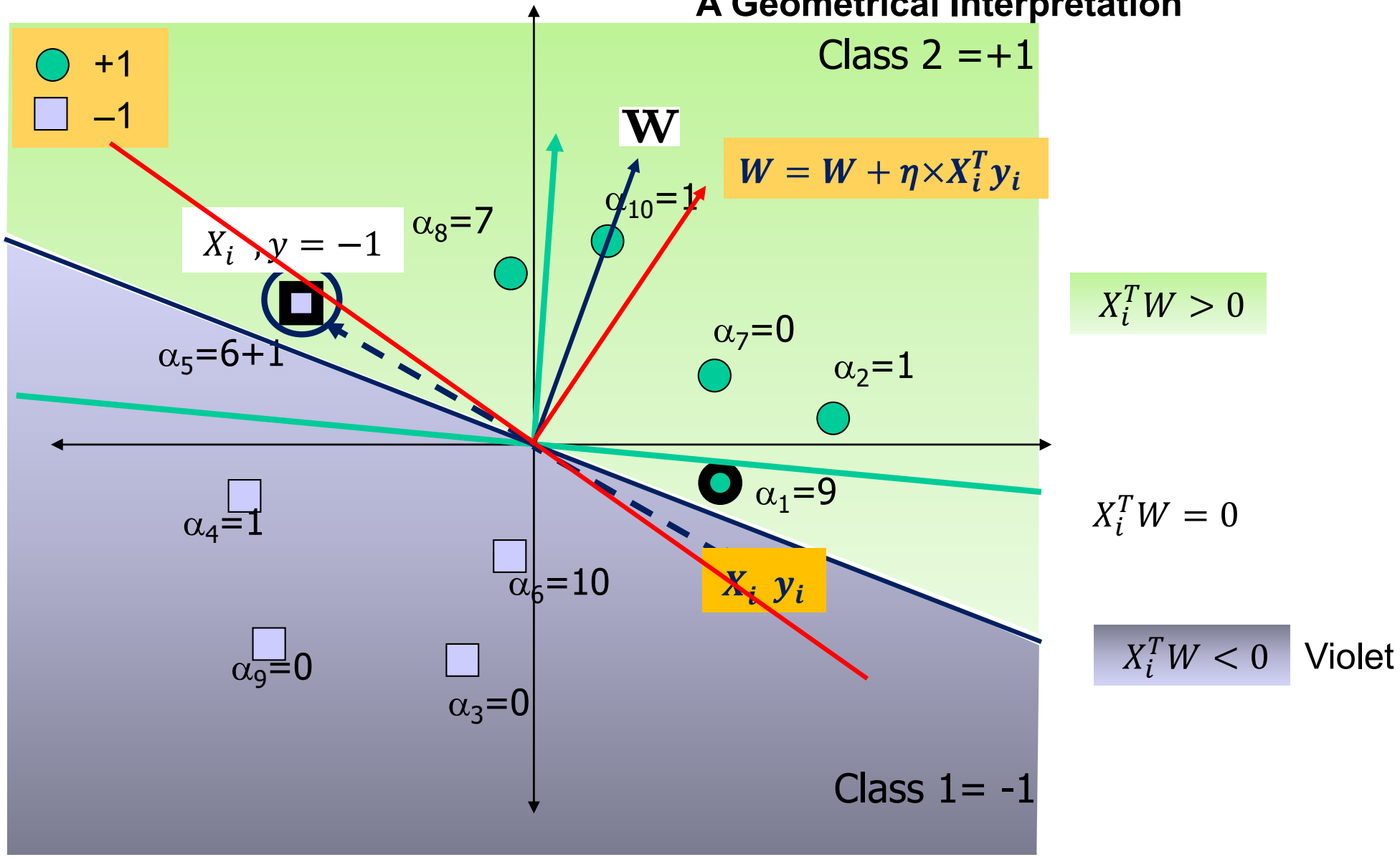


Perceptron Learning: mistake on a negative example

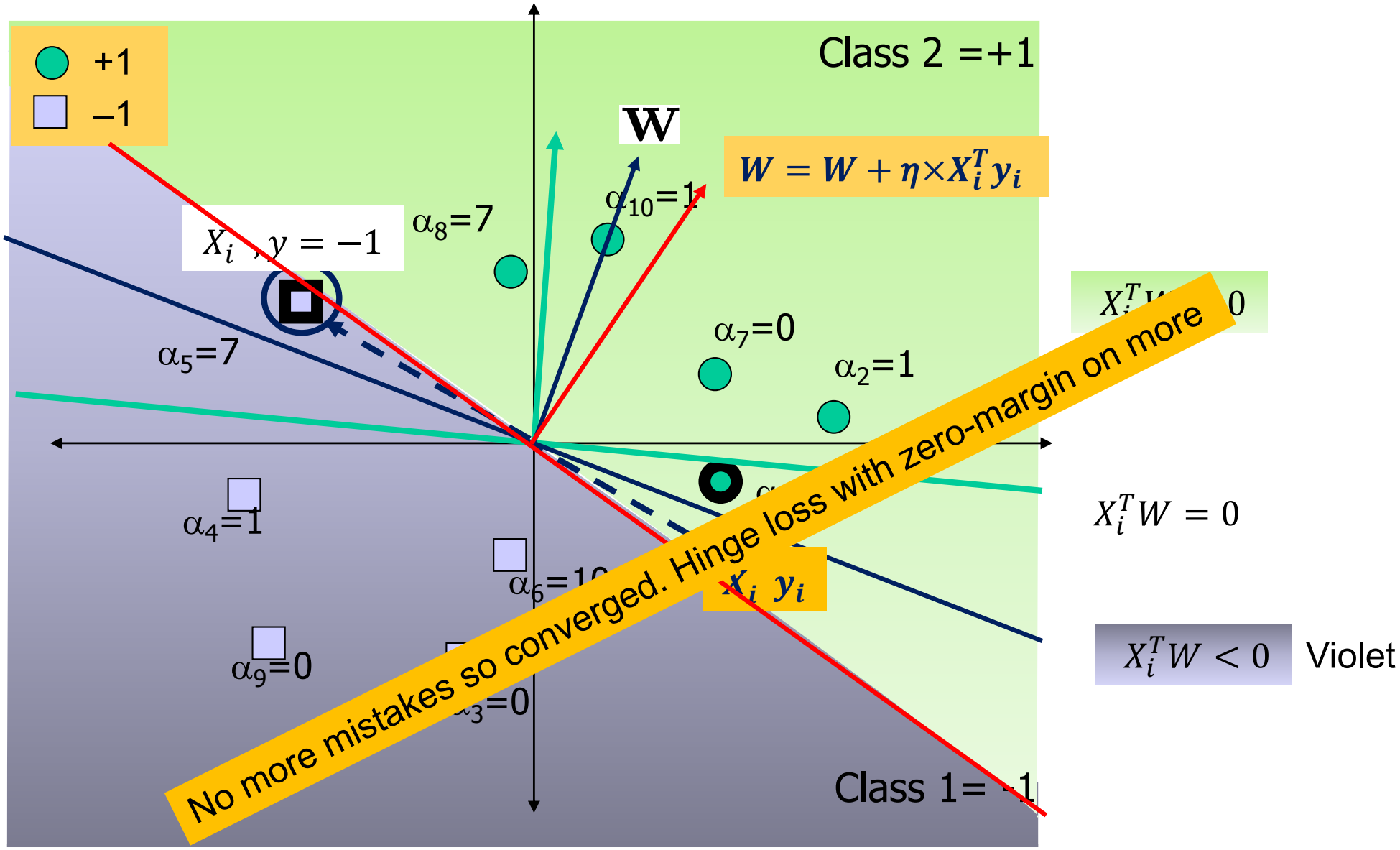


Perceptron Learning: mistake on a negative example

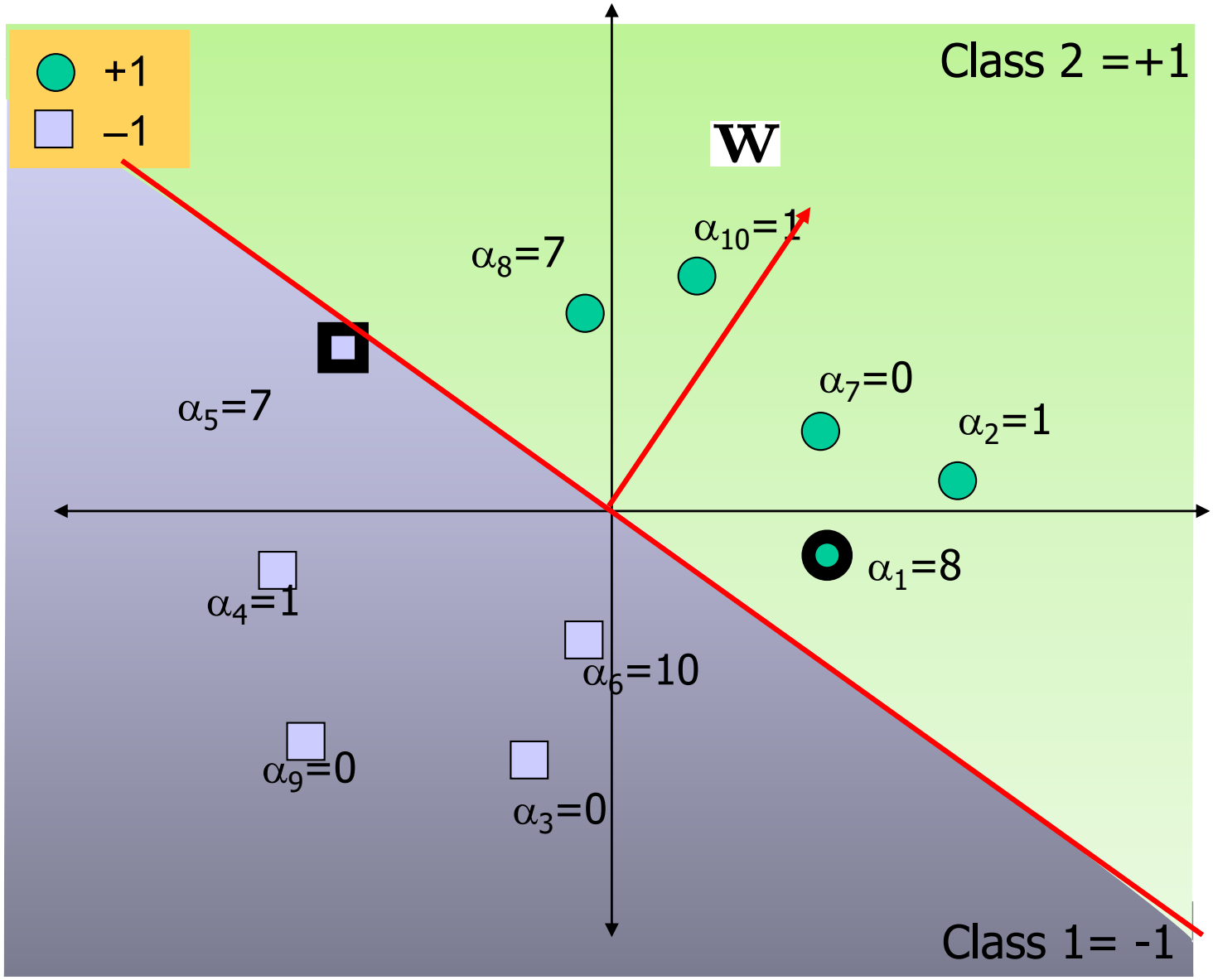
A Geometrical Interpretation



Perceptron Learning: mistake on a negative example



Perceptron Learning: mistake on a negative example

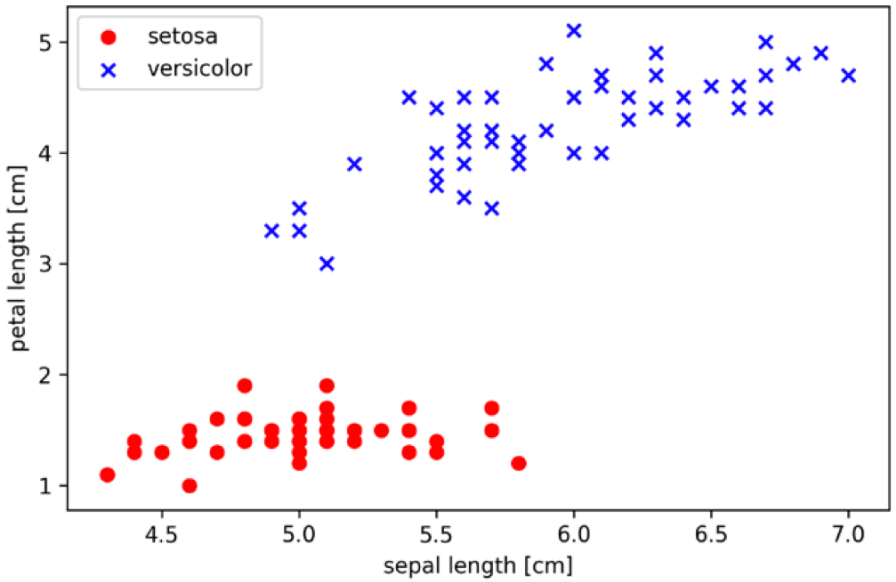


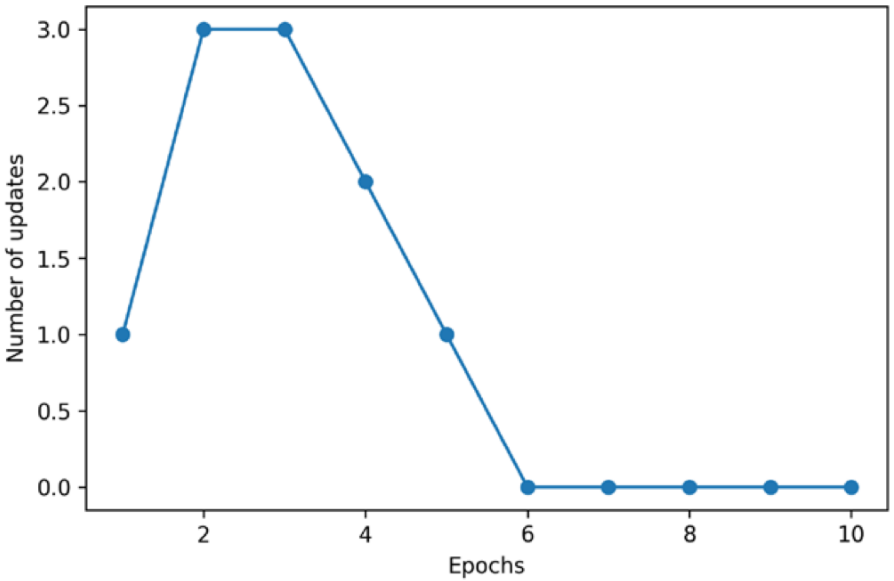
$X_i^T W > 0$

$X_i^T W = 0$

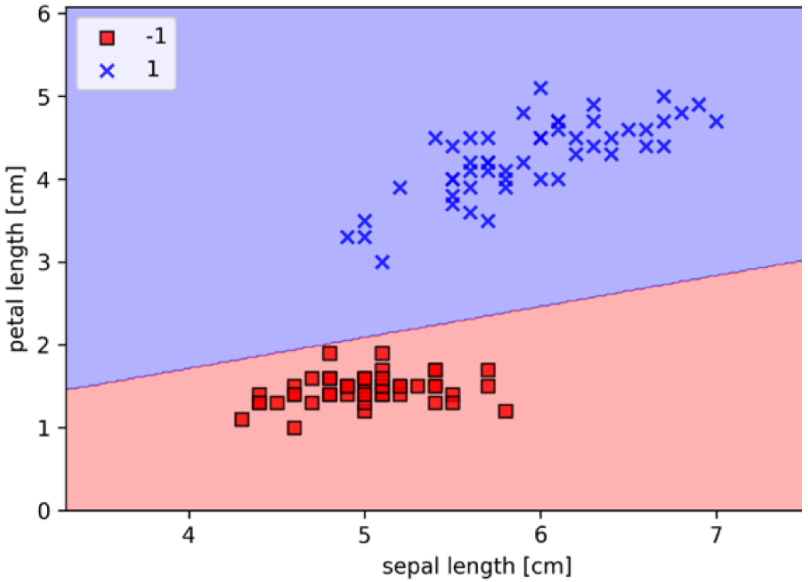
$X_i^T W < 0$ Violet

Misclassification errors vs the epochs






```
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1),
         ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
```



Gradient functions have similar structure

- **Linear regression**
  - Objective function : $\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$
  - Gradient for Linear Regression:

$$\frac{\partial}{\partial \theta_j}\mathrm{MSE}(\theta) = \frac{2}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

Weighted sum of the training data

- **Classification**
  - Gradient … ogistic Regression
    - … $y_i) X_i$

$$\frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T \cdot \mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Weighted sum of the training data

Perceptron has limiations
Next: try to extend

… Gradient for Perceptron (for errors):

- 1. $J_P(W) = \frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)$

Unweighted sum of the mistakes

2. $\nabla J_P(W) = \begin{cases} 0 & if\ X_i^T W y_i > 0 \\ -X_i^T y_i & otherwise \end{cases}$

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

Batch Perceptron Algorithm

1. Given a training set, (X_i , y_i), of size N
2. Let w = (0,0,0,...,0). #init weight vector
3. Repeat until no mistakes
    1. Permute trainingSet
    2. delta = (0,0,0,...,0)
    3. for (X_i , y_i) in trainingSet:
        - if $(X_i^T w y_i \leq 0)$: delta = delta + $y_i \mathbf{x}_i$
    4. delta = delta/N
    5. w = w - η delta

Online learning refers to the learning mode in which the model update is performed each time a single observation is received.

Batch learning in contrast performs model update after observing the whole training set.
Simplest case: $\eta = 1$ and don't normalize – 'Fixed increment perceptron'

η – the step size

- Also referred as the learning rate
- In practice, recommend to decrease η as learning continues
- Some optimization approaches set step-size automatically, e.g., by line search, and converge faster
- If linearly separable, there is only one basin for the hinge loss, thus local minimum is the global minimum

Perceptron: Online versus Batch

- Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{I}\left[y_t(\mathbf{w}^{(t)} \cdot \mathbf{x}_t) \leq 0\right] y_t \mathbf{x}_t$$

- Batch hinge minimization update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \frac{1}{N} \sum_{i=1}^{N} \left\{\mathbb{I}\left[y_i(\mathbf{w}^{(t)} \cdot \mathbf{x}_i) \leq 0\right] y_i \mathbf{x}_i\right\}$$

Perceptron in terms of subgradients

subgradient

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \, \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Batch hinge minimization update

Sum over mistakes

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \frac{1}{N} \sum_{i=1}^{N} \left\{ \mathbb{1}\left[y^{(i)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(i)}) \leq 0\right] y^{(i)}\mathbf{x}^{(i)} \right\}$$

- Difference?

Perceptron update is a stochastic subgradient descent algorithm for hinge loss minimization with a step size = 1

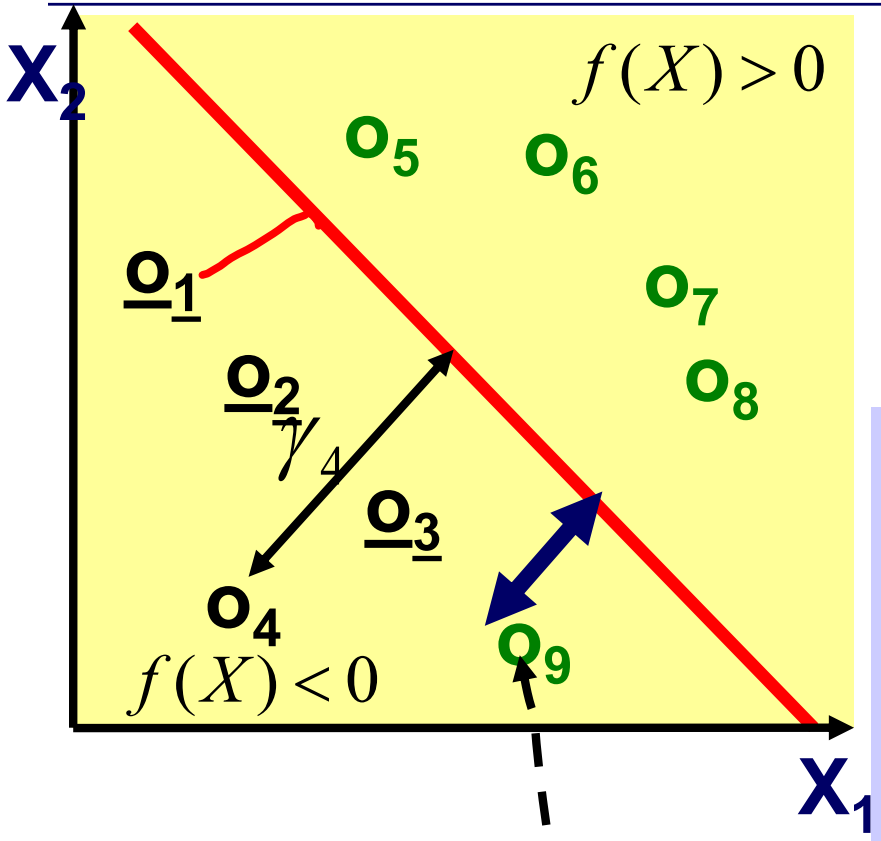
Online VS. Batch Perceptron

- **Batch learning learns with a batch of examples collectively**
- **Online learning learns with one example at a time**
- **Both learning mechanisms are useful in practice**
- **Online Perceptron is sensitive to the order that training examples are received**
- **In batch training, the correction incurred by each mistake is accumulated and applied at once at the end of the iteration**
- **In online training, each correction is applied immediately once a mistake is encountered, which will change the decision boundary, thus different mistakes maybe encountered for online and batch training**
- **Online training performs stochastic gradient descent, an approximation to real gradient descent, which is used by the batch training**

Assess quality of model by examining the margin: γ

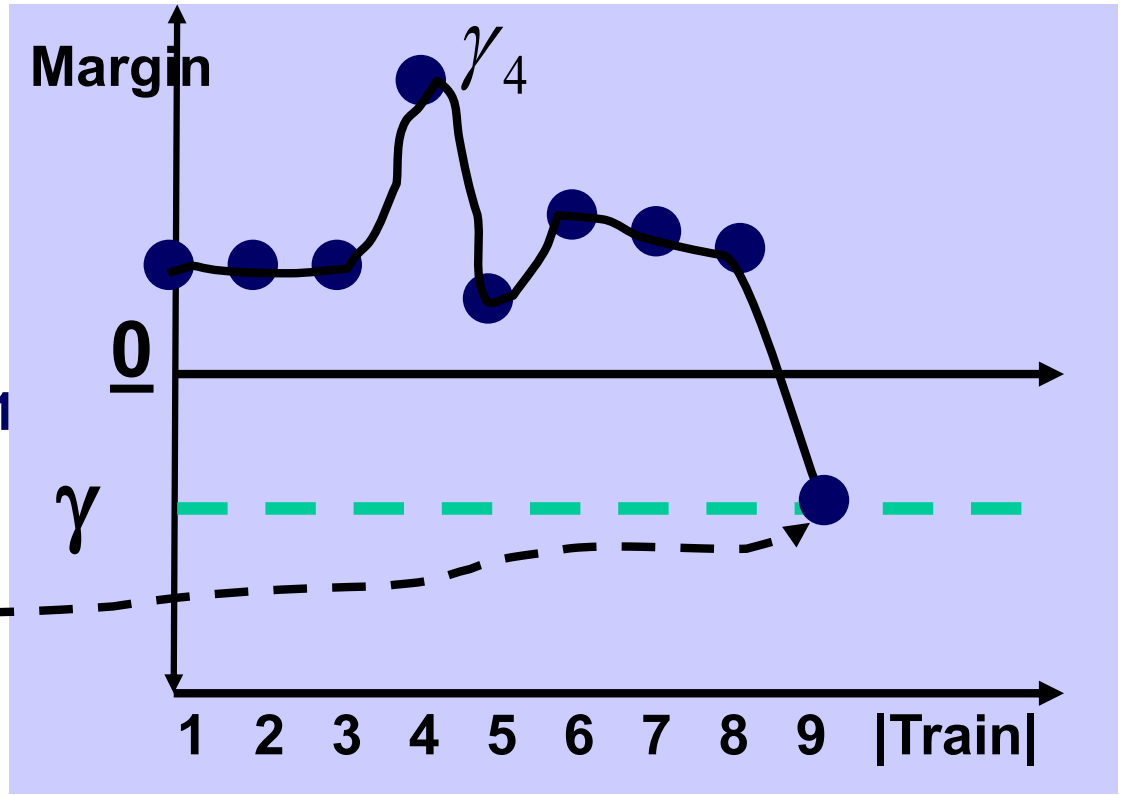
- **γ (gamma) is referred to as the margin**
  - The minimum distance from data points to the decision boundary
  - The bigger the margin, the easier the classification problem is
  - The bigger the margin, the more confident we are about our prediction
- **We will see later in the course this concept leads to one of the recent most exciting developments in the ML field – support vector machines**

Margin and Margin Distribution



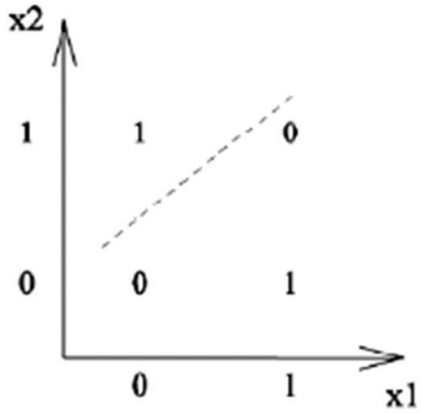
Take any linear separator (e.g., a logistic regression or perceptron model)

$$\gamma_P = \min_{1 \le i \le N} X_i^T W y_i$$



Negative Margin for example o9

Perceptron and not linearly separable data



- In such cases the algorithm will never stop! How to fix?
- Look for decision boundary that make as few mistakes as possible – NP-hard!

Fixing the Perceptron: run for a fixed number of iterations (P)

- Idea one: only go through the data once, or a fixed number of times

Let w = (0,0,0,...,0). #init weight vector
Repeat N×P times
- Permute trainingSet
- for (X_i , y_i) in trainingSet:
  if $(X_i^T w y_i \leq 0)$: w = w + $y_i \mathbf{x}_i$

- At least this stops
- Problem: the final ***w*** might not be good e.g. right before it stops the algorithm might perform an update on a total outlier

Voted Perceptron

- Keep intermediate hypotheses and have them vote [Freund and Schapire 1998]

Let $w_0 = (0,0,0, \ldots,0)$
$c_0 = 0$
repeat
  Take example $i : (x_i, y_i)$
  $u_i \leftarrow \mathbf{w}_n \cdot \mathbf{x}_i$
  if $y_i \cdot u_i <= 0$
    $\mathbf{w}_{n+1} \leftarrow \mathbf{w}_n + y_i \mathbf{x}_i$
    $c_{n+1} = 0$
    $n = n + 1$
  else
    $c_n = c_n + 1$

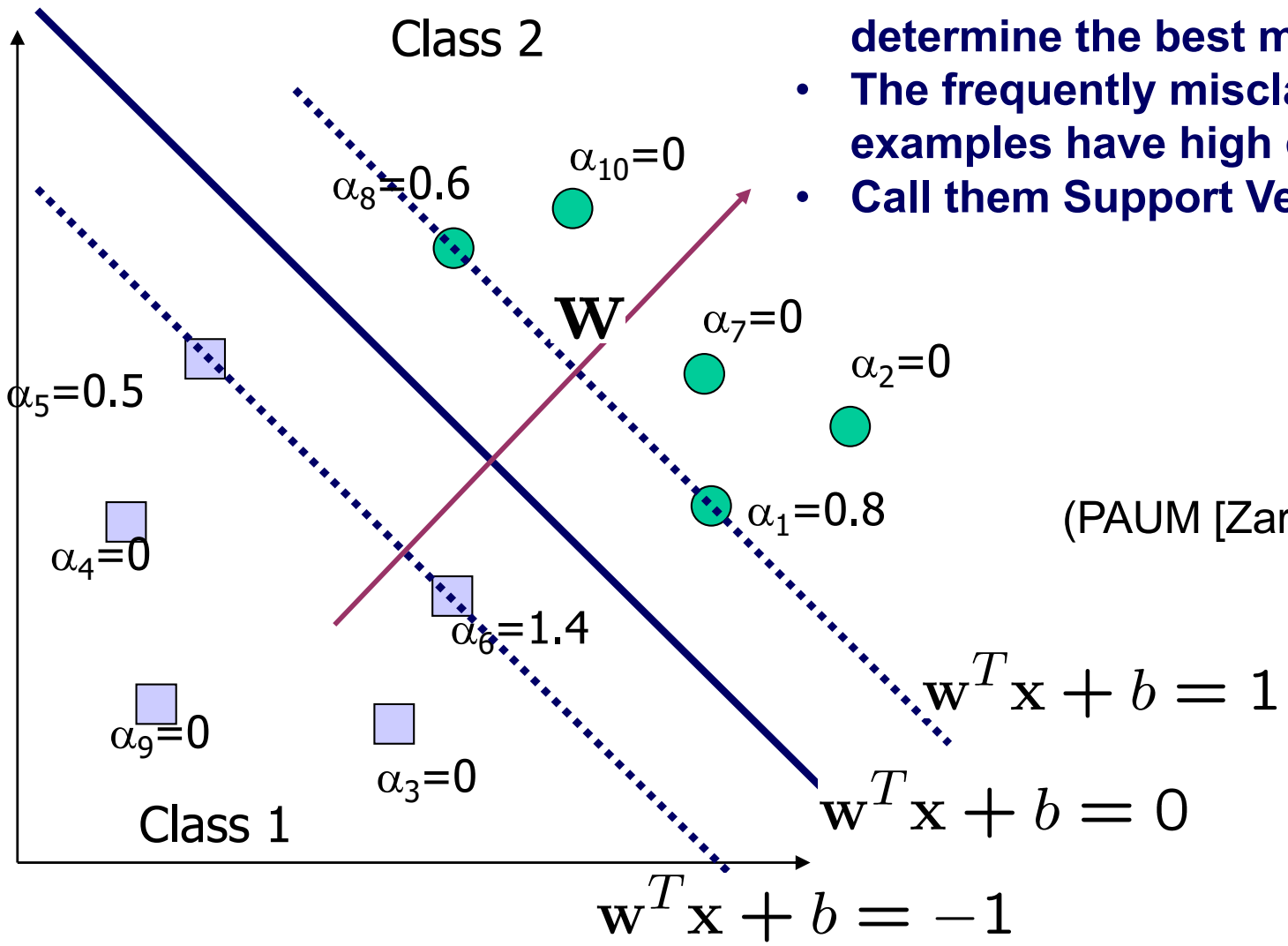
Store a collection of linear separators $\mathbf{w}_0\, \mathbf{w}_1 \ldots$, along with their survival time $c_0, c_1 \ldots$

The c's can be good measures of the reliability of the **w**'s

For classification, take a weighted vote among all separators:

$$\operatorname{sgn}\{\sum_{n=0}^{N} c_n \operatorname{sgn}(\mathbf{w}_n^T \mathbf{x})\}$$

PAUM: Margin-based Perceptron Learning



- **Use cross fold validation to determine the best margin**
- **The frequently misclassified examples have high α's.**
- **Call them Support Vectors**

(PAUM [Zaragosa et al.])

Summary of Perceptron

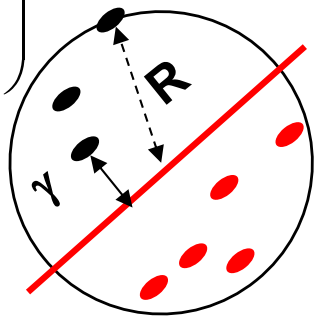
- **Learns a Classifier y'=$\hat{y}$ = *f*(x) directly**
- **Applies gradient descent search to optimize the hinge loss function**
  - Online version performs stochastic gradient descent
- **Guaranteed to converge in finite steps if linearly separable**
  - There exists an upper bound on the number of corrections needed
  - Inversely proportional to the margin of the optimal decision boundary [Block, 1962, Novikoff, 1962]

R, the radius of the ball containing the data

$$NumberOfCorrections \leq \left(\frac{2R}{\gamma}\right)^2$$

Where $R = \max_{i=1}^{|Train|} \sqrt[2]{\langle X_i, X_i \rangle}$

and γ denoted the margin of S



- **If not linearly separable,**
  - Use voted perceptrons
  - Or other perceptron extensions (PAUM [Zaragosa et al.])
  - Or SVMs!

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
6. **Summary**

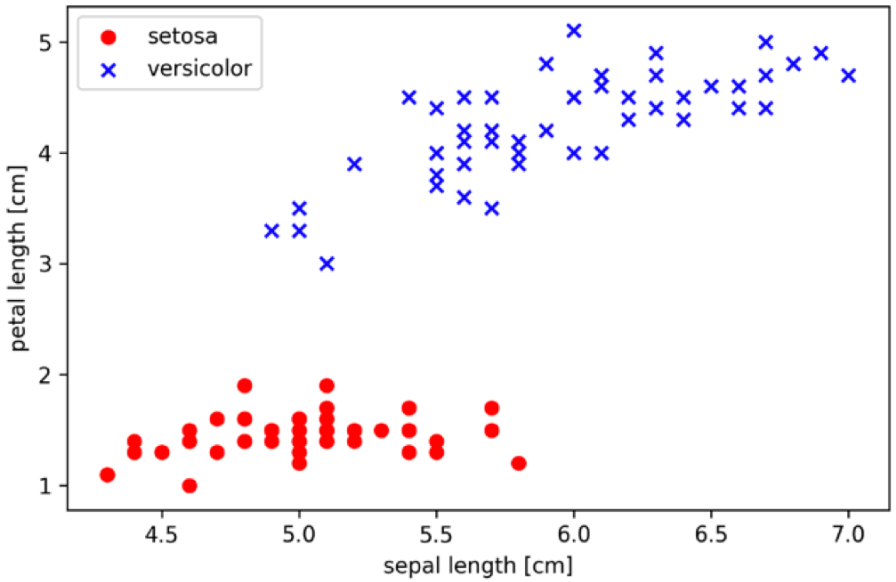
Multiclass classification

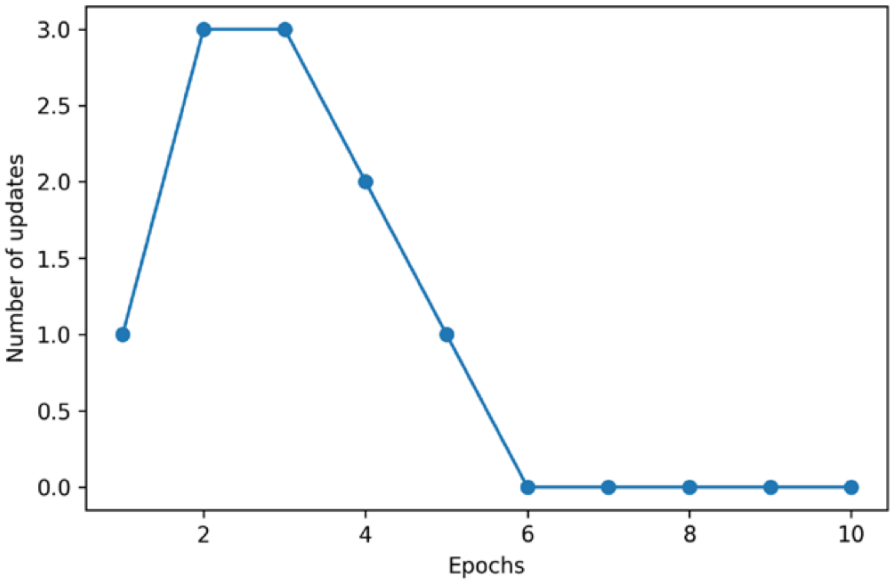
- In machine learning, **multiclass** or **multinomial classification** is the problem of classifying instances into one of three or more classes. (Classifying instances into one of the two classes is called binary classification.)
- While some classification algorithms naturally permit the use of more than two classes, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.
- Multiclass classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance.

Multiclass classification techniques

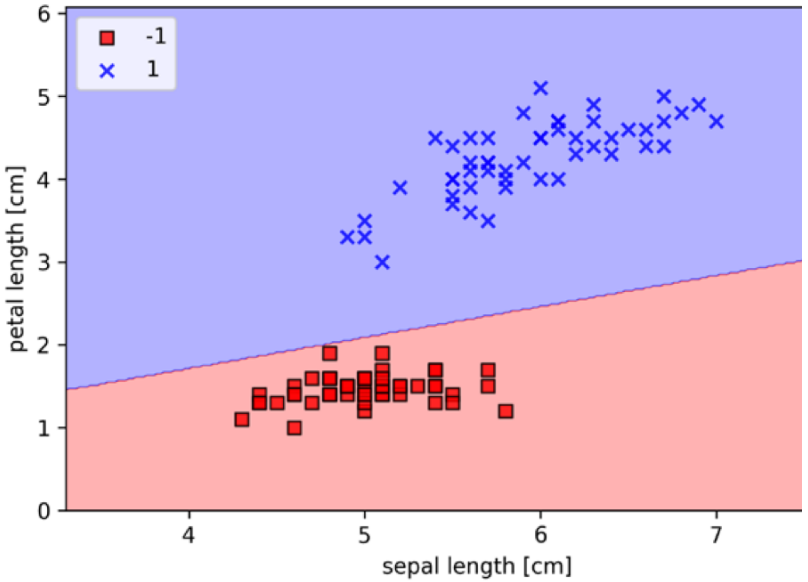
- **(i) Transformation to binary**
  - reducing the multi-class problem into multiple binary problems can also be called problem transformation techniques:
  - *one-vs.-rest; learn K* classifiers $\hat{y} = \underset{k \in \{1 \ldots K\}}{\operatorname{argmax}} f_k(x)$
  - *one-vs.one:*
    - *K* (*K* − 1) / 2 classifiers are applied to an unseen sample and the class that got the highest number of "+1" predictions gets predicted by the combined classifier
- **(ii) Extension from binary:**
  - Multinonial logistic regression, Multinonial perceptrons, deep learning, naïve Bayes, decision trees
- **(iii) Hierarchical classification.**[1]
  - Divide the output space i.e. into a tree. Each parent node is divided into multiple child nodes and the process is continued until each child node represents only one class. Several methods have been proposed based on hierarchical classification.

$K\,(K-1)\,/\,2$ one-vs-one classifiers



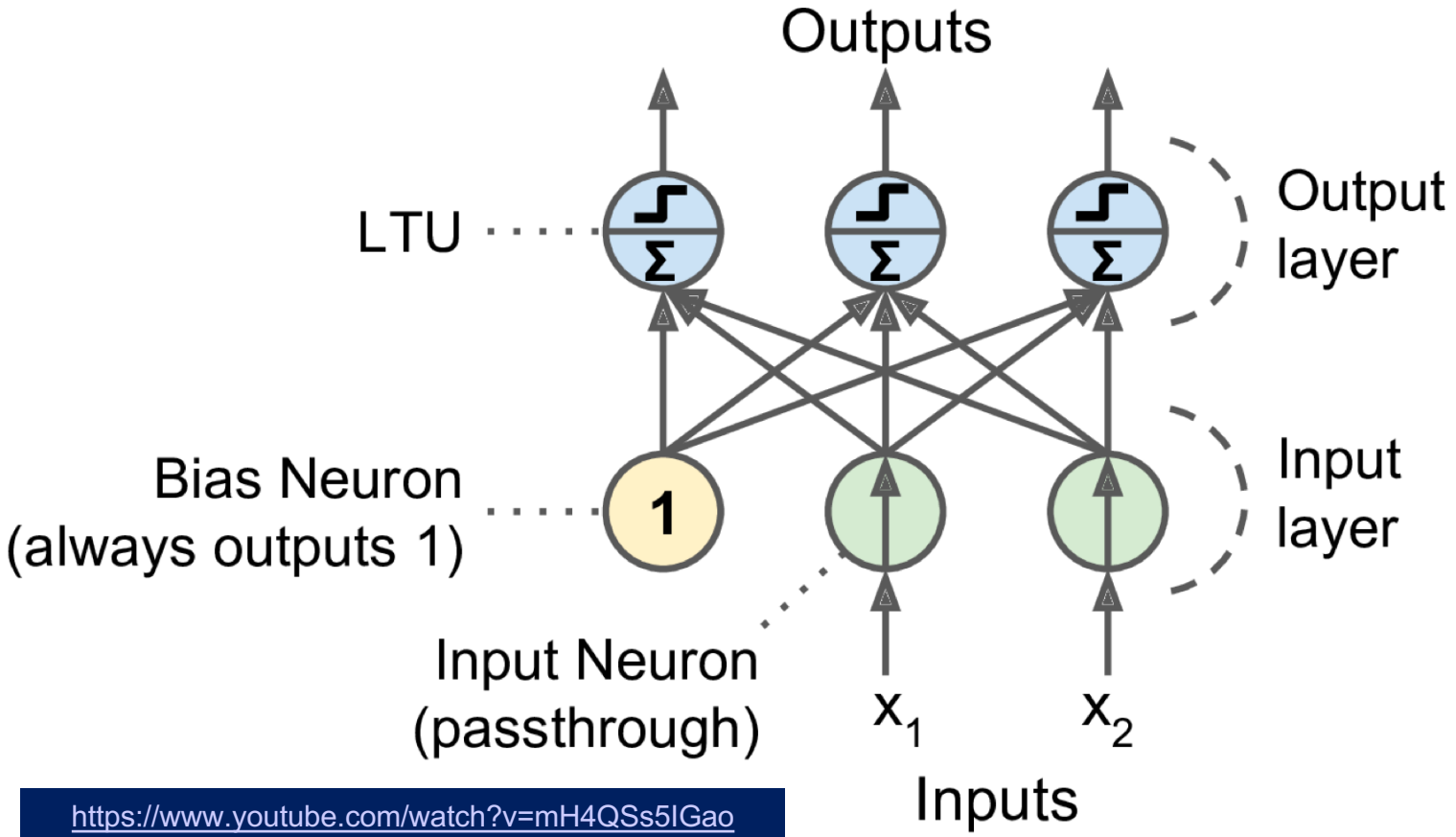



```
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1),
         ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
```



Multinomial Perceptron: A 2-3 perceptron

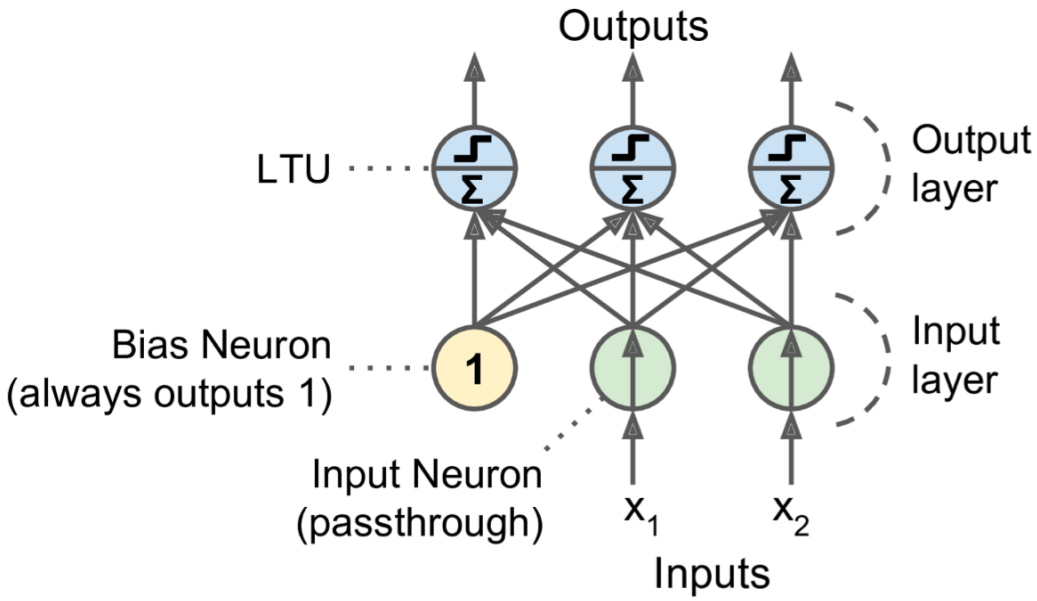
- **A Perceptron with two (2) inputs and three (3) outputs is represented in. This Perceptron can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier.**

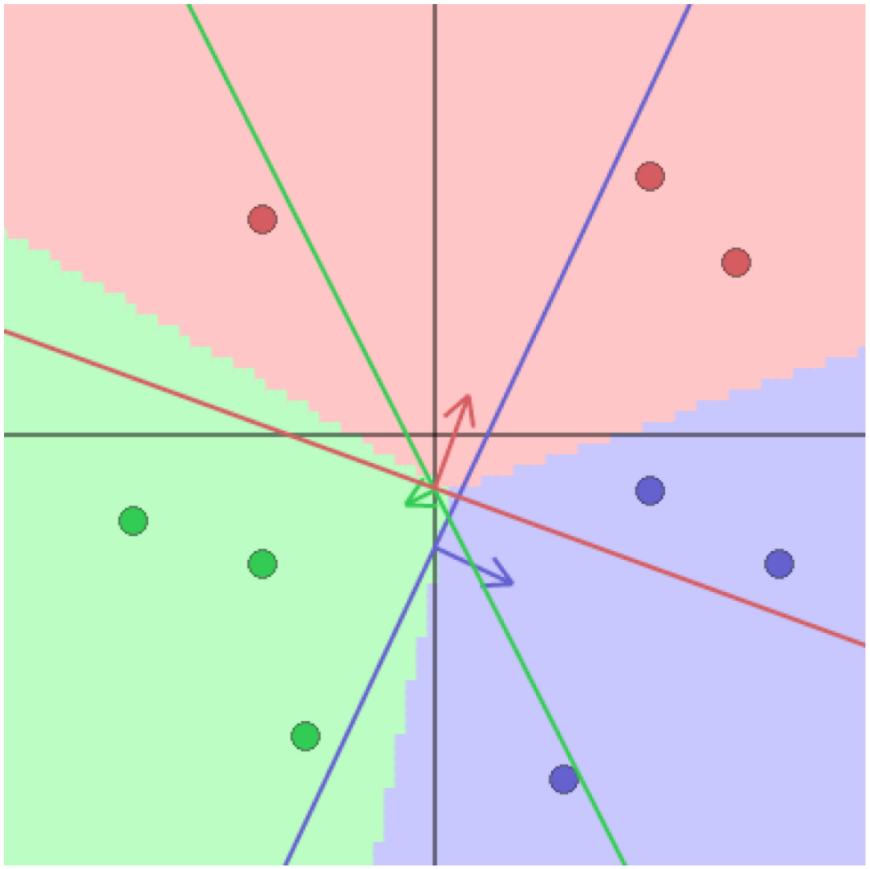


https://www.youtube.com/watch?v=mH4QSs5IGao

A 2-3 Multinomial Perceptron

- **Classification rule is similar to logistic regression**
  - For each class calculate the perpendicular distance
  - $y' = \text{argmax}_y (X^T w_y)$. #class with largest perdDist

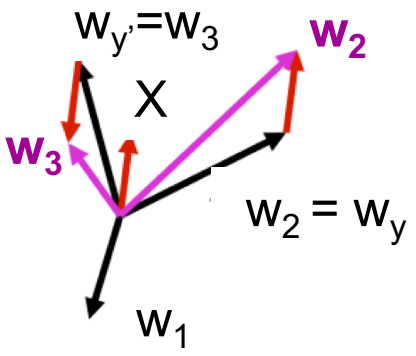


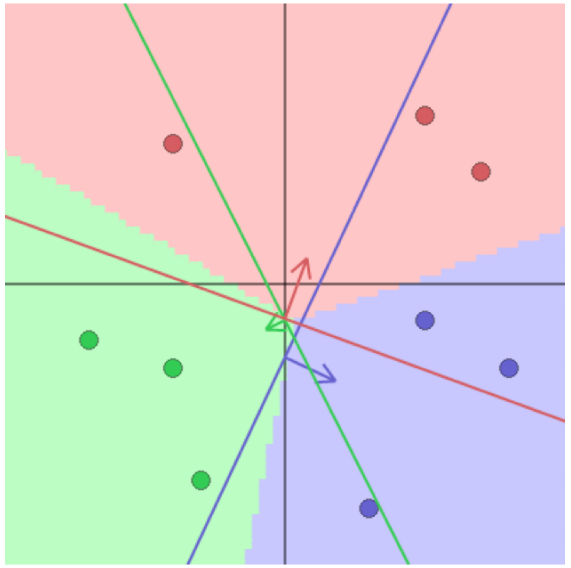


Learning a Multinomial Perceptron

1. **Start with zero weights**
2. **Online learning: pick a training example, <X, 0,1,0>, at a time**
3. **Classify with current weights**
   1. $y' = \text{argmax}_y (X^T w_y)$. #class with largest perdDist
4. **If correct, no change!**
5. **If wrong:**
   1. lower score of wrong answer y', $w_{y'} = w_{y'} - X$
   2. Raise score of actual class y, $w_y = w_y - X$

OHE

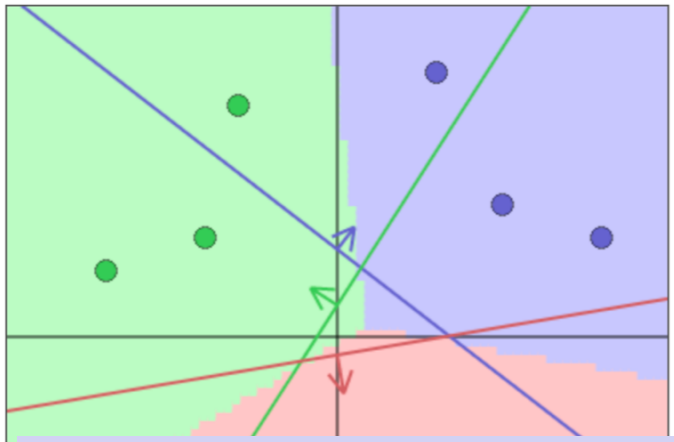




NOTE: Similar but different to multinomial logistic regression. But different. What are the differences?

MultiClass Classification Demo: Interactive Web Demo time

Datapoints are shown as circles colored by their class (red/gree/blue). The background regions are colored by whichever class is most likely at any point according to the current weights. Each classifier is visualized by a line that indicates its zero score level set. For example, the blue classifier computes scores as $W_{0,0}x_0 + W_{0,1}x_1 + b_0$ and the blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



Parameters W, b are shown below. The value is in bold and its gradient (computed with backprop) is in *red, italic* below. Click the triangles to control the parameters.

W[0,0]	W[0,1]	b[0]
0.52 *-0.07*	**0.68** *-0.04*	**-0.18** *0.01*
W[1,0]	**W[1,1]**	**b[1]**
-0.82 *0.08*	**0.52** *-0.02*	**-0.05** *0.01*
W[2,0]	**W[2,1]**	**b[2]**
0.20 *-0.01*	**-1.17** *0.06*	**-0.06** *-0.02*

Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i. The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]). The last column is the data loss for a single example, L_i.

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	0.35	-0.25	-0.43	0.69
0.80	0.30	0	0.44	-0.55	-0.26	0.63
0.30	0.80	0	0.52	0.12	-0.94	0.64
-0.40	0.30	1	-0.19	0.43	-0.50	0.66
-0.30	0.70	1	0.14	0.56	-0.94	0.63
-0.70	0.20	1	-0.41	0.63	-0.44	0.53
0.70	-0.40	2	-0.08	-0.83	0.54	0.58
0.50	-0.60	2	-0.32	-0.77	0.74	0.45
-0.40	-0.50	2	-0.73	0.01	0.44	0.67
						mean:
						0.61

Total data loss: 0.61
Regularization loss: 0.31

LOSS

Maximize $Pr(Y=y_i|X; W)$ # Maximize (conditional) likelihood; should be 1
Mimimize $-Log(Pr(Y=y_i|X_i; W)$ #LOSS; should be 0.

Adapted from http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

Corrected DEMO here on Dropbox
Please download everything from dropbox to your computer and run locally
https://www.dropbox.com/sh/91ivcvd7rckmfq0/AAABLQz9iIxUR8iM2d8bEdcBa?dl=0

Quiz: Update rule for a Multinomial perceptron

Given a Perceptron Model for 3 classes
- w_1=(1,2,-2) w_2=(3,-2, -1) w_3=(-1,2, 4)

And a training example: x = (1, -0.5, 3) and y =2
PERFORM one iteration of online learning
After gradient descent w_1 =(1,2,-2), w_2=(4,-2.5,2), w_3=(-2,2.5,1)

- **Update mistakes**
  - Class is 2, BUT predict 3.

Perpendicular distance

$w_1 \cdot (x) = 1 \cdot 1 + 2 \cdot (-0.5) + (-2) \cdot 3 = -6$

$y' = \text{argmax}_y (X^T w_y)$

Actual class (2) $w_2 \cdot (x) = 3 \cdot 1 - 2 \cdot (-0.5) - 1 \cdot 3 = 1$

Predicted class(3) $w_3 \cdot (x) = -1 \cdot 1 + 2 \cdot (-0.5) + 4 \cdot 3 = 10$

→ prediction: y=3

Update W

$w_2 \leftarrow w_2 + (x)$

$(3,-2,-1) + (1,-0.5,3)$

w_2=(4,-2.5,2)

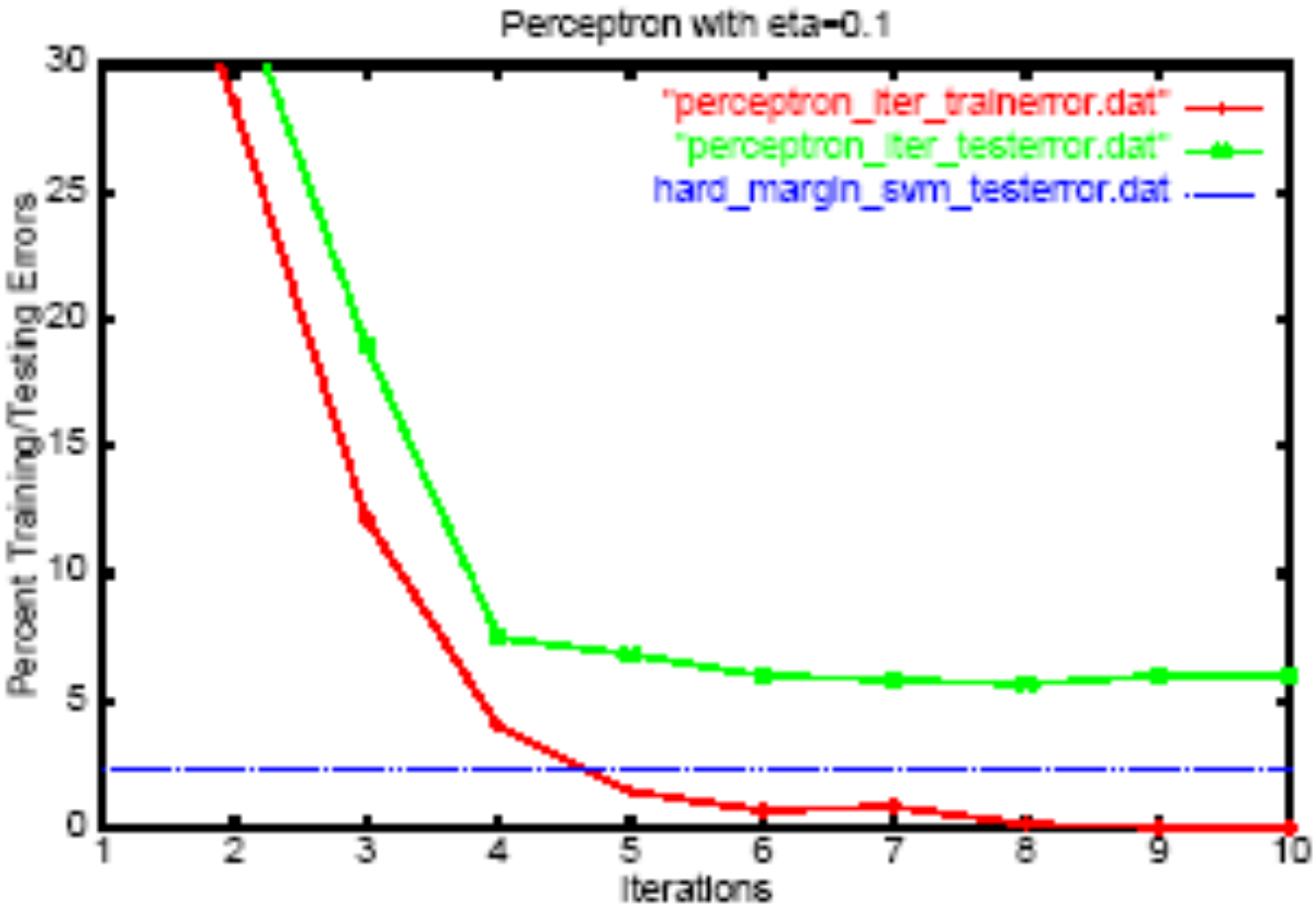
$w_3 \leftarrow w_3 - (x) = (-1,2,4) - (1,-0.5,3)$

w_3=(-2,2.5,1)

If wrong:
lower score of wrong answer y', $w_{y'} = w_{y'} - X$
Raise score of actual class y, $w_y = w_y - X$

Perceptron Learning: Text Example

Experiment: Perceptron for Text Classification



Train on 1000 pos / 1000 neg examples for "acq" (Reuters-21578).

[Source: http://www.cs.cornell.edu/Courses/CS678/2003sp/slides/perceptron_4up.pdf]

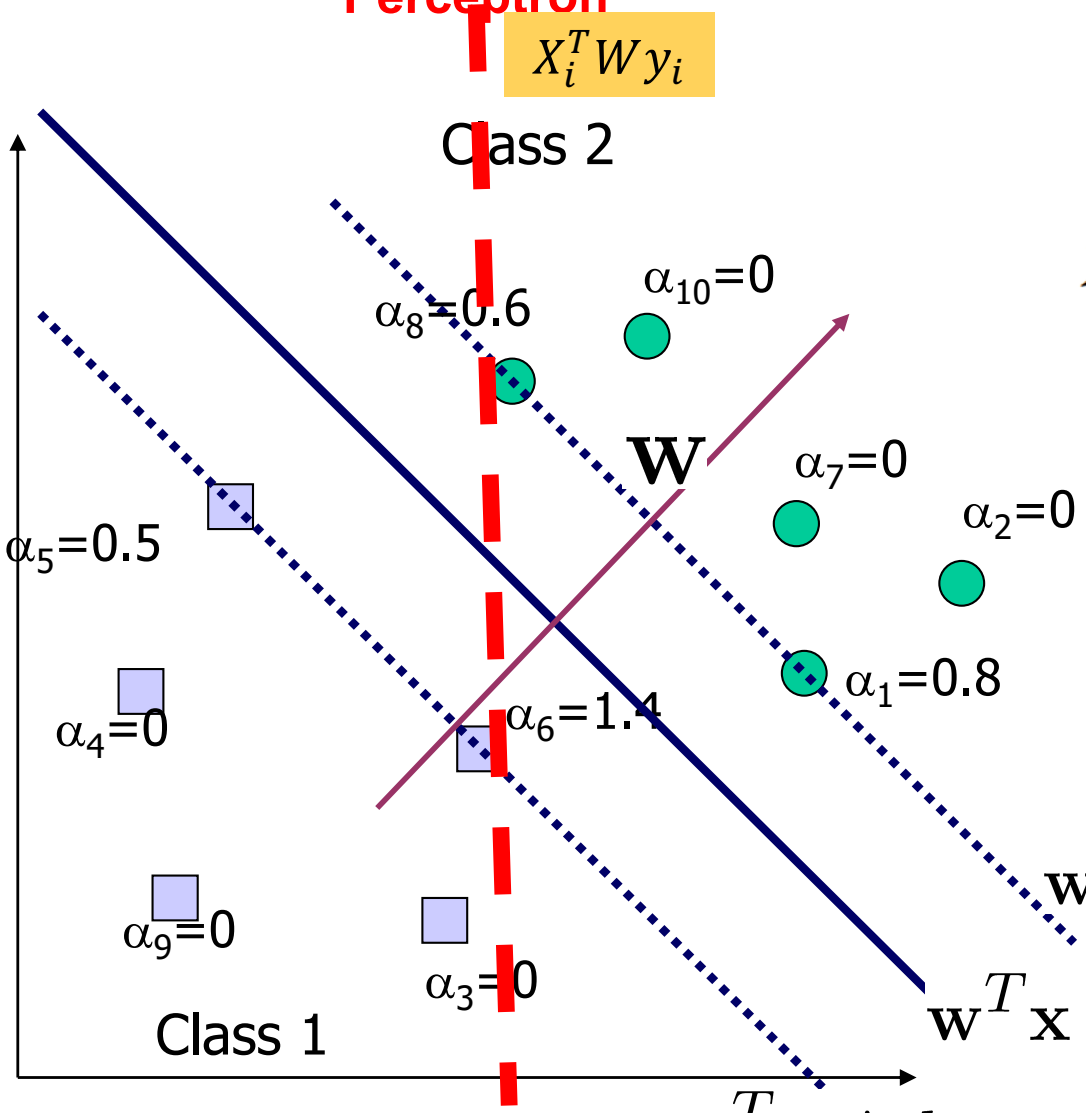
Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
    1. From perceptron to SVM (briefly)
6. **Summary**

Perceptron vs SVM Learning
The search for a separating hyperplane

Perceptron

Margin-based Learning



$$\hat{y} = \begin{cases} 0 \text{ if } \mathbf{w}^T \cdot \mathbf{x} + b < 0, \\ 1 \text{ if } \mathbf{w}^T \cdot \mathbf{x} + b \geq 0 \end{cases}$$

Margin for each example

$\gamma_i = X_i^T W y_i \geq 1$

$\gamma_{SupportVectors}$=1

Margin for each example

$\gamma_i = X_i^T W y_i \geq 1$

Movie Filter

$H_1: \langle \vec{W}, \vec{X} \rangle + b = 1$ $\langle \vec{W}, \vec{X} \rangle + b = 0$ $H_2: \langle \vec{W}, \vec{X} \rangle + b = -1$

Learning can be viewed as optimisation

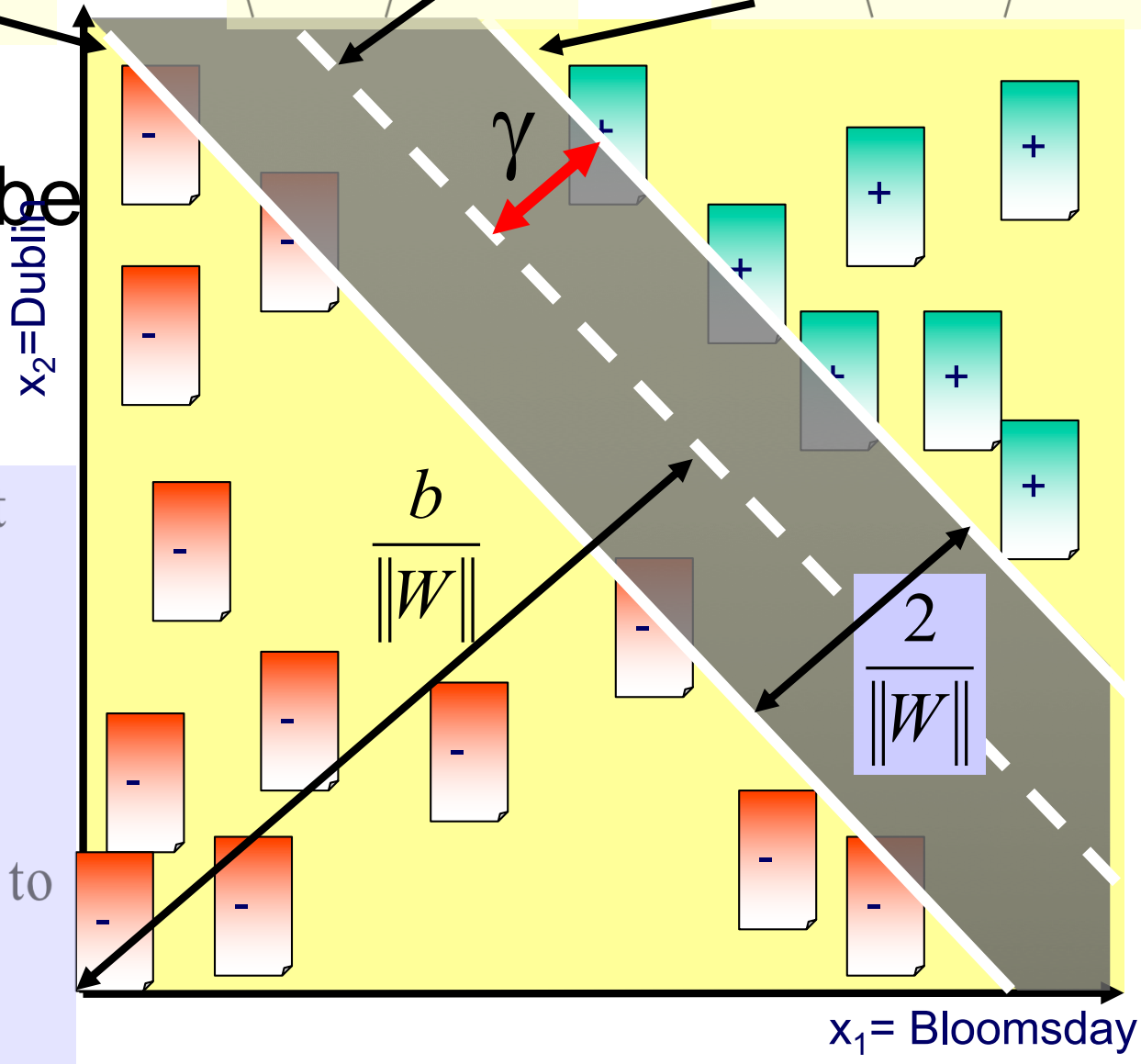
Find H_1 and H_2 such that

$Dist(H_1, H_2)$ is MAX

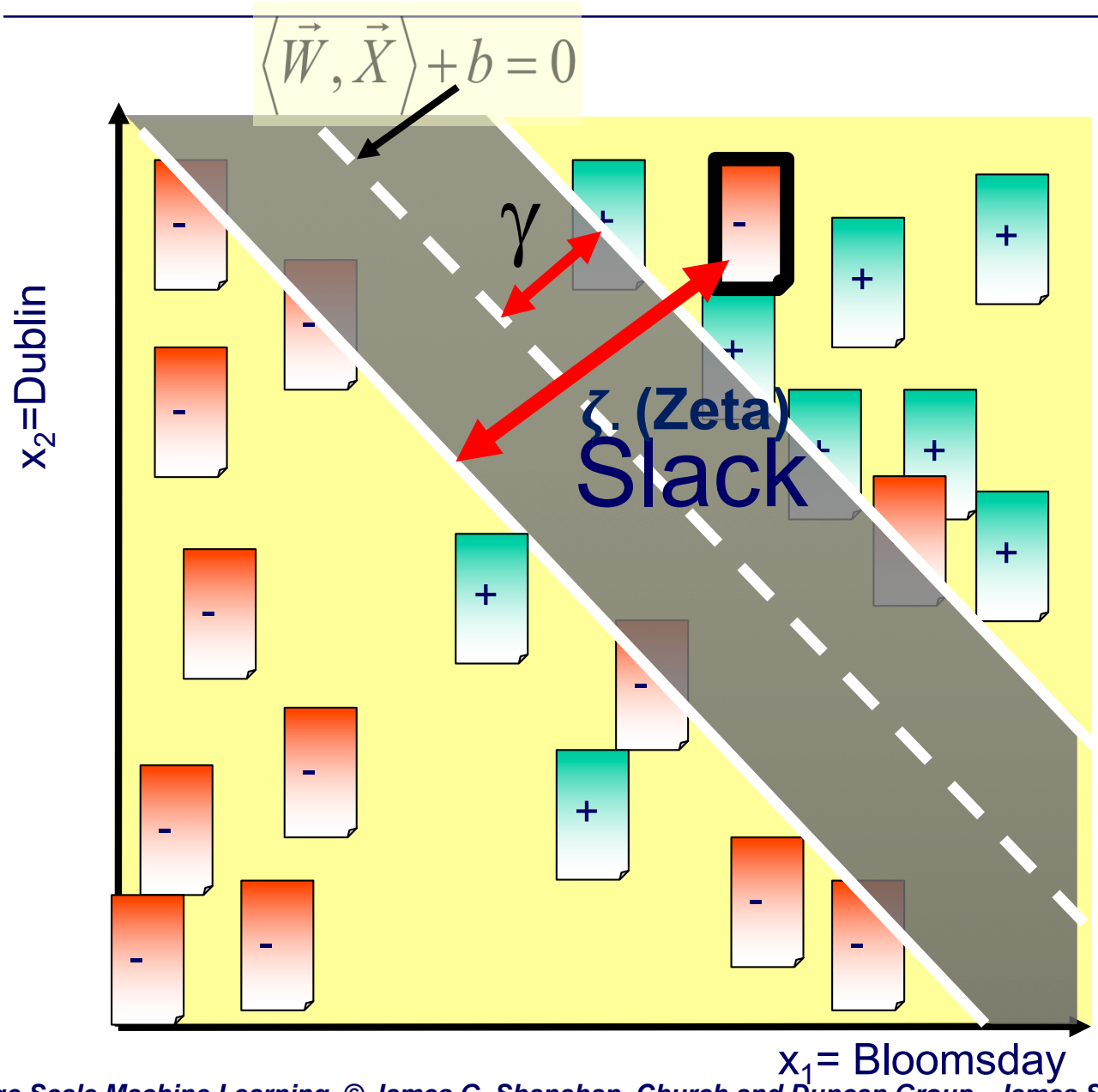
Or

Minimize $\frac{\|\vec{W}\|^2}{2}$ subject to

$y_i(<\vec{W}, \vec{X}_i> + b) \geq 1$



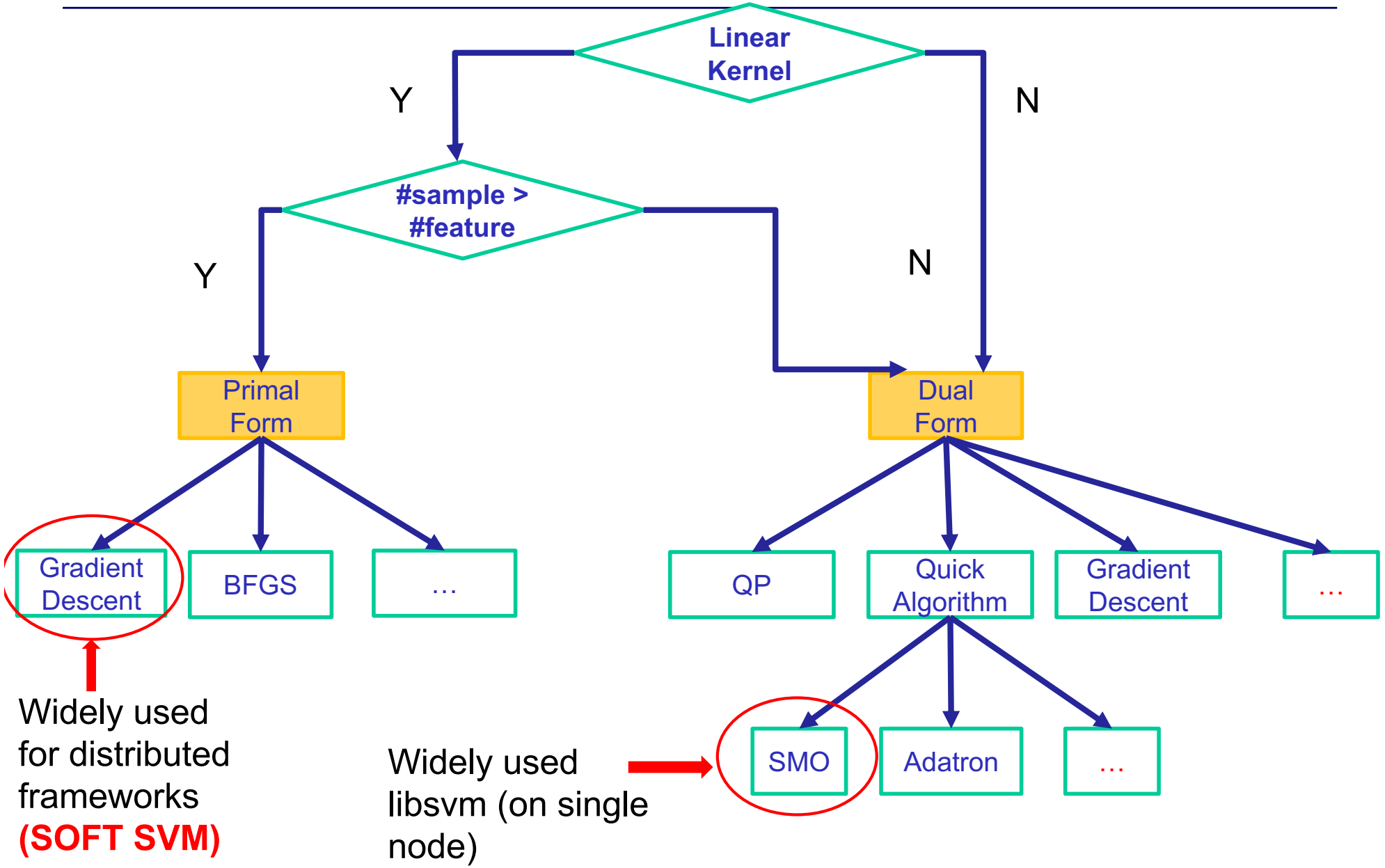
Soft SVMs (limited nonlinear models)



$Slack\ to\ each\ example$
ζ. (Zeta)

- Soft SVMs are limited nonlinear models (tolerate noise);
- Use kernel SVMs to build truly nonlinear models (polynomial; RBF)

SVM Learning Algorithm Taxonomy



Hard/Soft SVMs as quadratic programs

margin slack

$$\underset{\mathbf{w}, b, \zeta}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} + C \sum_{i=1}^{m} \zeta^{(i)}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

QP

$$\underset{\mathbf{p}}{\text{Minimize}} \quad \frac{1}{2}\mathbf{p}^T \cdot \mathbf{H} \cdot \mathbf{p} \quad + \quad \mathbf{f}^T \cdot \mathbf{p}$$

$$\text{subject to} \quad \mathbf{A} \cdot \mathbf{p} \leq \mathbf{b}$$

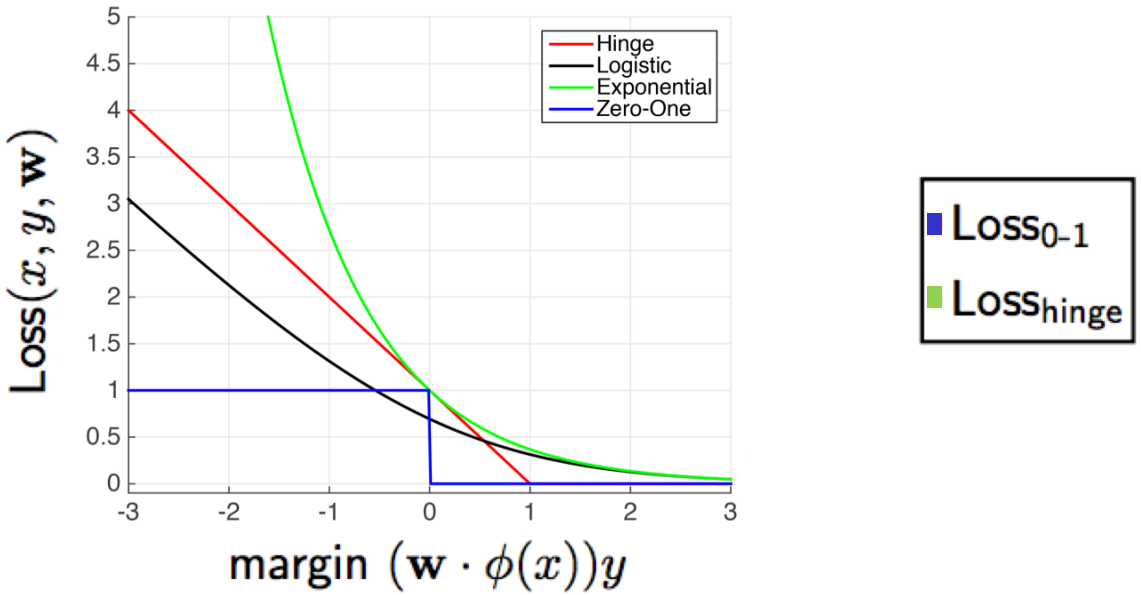
where

- $\mathbf{p}$ is an n_p-dimensional vector (n_p = number of parameters),
- $\mathbf{H}$ is an $n_p \times n_p$ matrix,
- $\mathbf{f}$ is an n_p-dimensional vector,
- $\mathbf{A}$ is an $n_c \times n_p$ matrix (n_c = number of constraints),
- $\mathbf{b}$ is an n_c-dimensional vector.

Note that the expression $\mathbf{A} \cdot \mathbf{p} \leq \mathbf{b}$ actually defines n_c constraints: $\mathbf{p}^T \cdot \mathbf{a}^{(i)} \leq b^{(i)}$ for $i = 1, 2, \cdots, n_c$, where $\mathbf{a}^{(i)}$ is the vector containing the elements of the i^{th} row of $\mathbf{A}$ and $b^{(i)}$ is the i^{th} element of $\mathbf{b}$.

SVM Loss is a Hinge Loss

$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$



- Intuition: hinge loss upper bounds 0-1 loss, has non-trivial gradient
- Try to increase margin if less than 1

Perceptron and SVM have similar loss functions

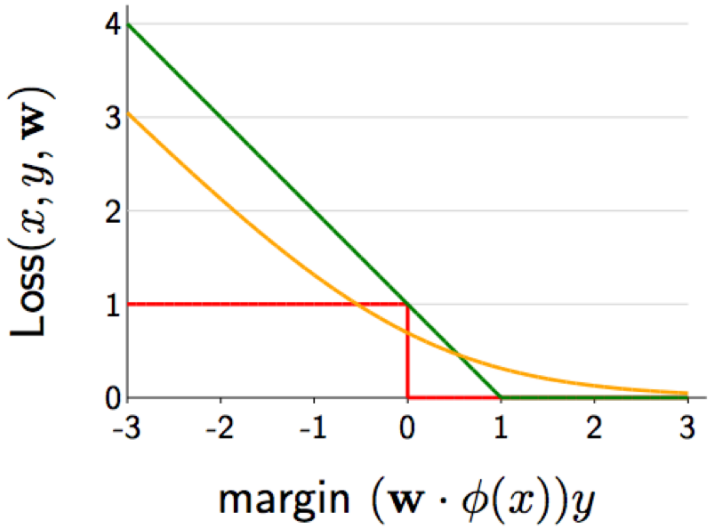
1. $Loss_{P\_Hinge}(W) = \frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)$

2. $\nabla Loss_{P\_Hinge}(W) = \begin{cases} 0 & if\ X_i^T W y_i > 0 \\ -X_i^T y_i & otherwise \end{cases}$

3. $Loss_{SVM\_Hinge}(W) = \frac{1}{N}\sum_{i=1}^{N} Max\left(0, 1 - X_i^T W y_i\right)$

4. $Loss_{Logistic}(W) = \frac{1}{N}\sum_{i=1}^{N} \log(1 + e^{(-X_i^T W y_i)})$

Lagrangian



- Intuition: Try to increase margin even when it already exceeds 1

Sub-gradient descent algorithm for SVM

$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{||\mathbf{w}||^2}_{\text{regularization}} + C\sum_i^N \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

Lagrangian

The iterative update is

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta\nabla_{\mathbf{w}_t}\mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta\frac{1}{N}\sum_i^N(\lambda\mathbf{w}_t + \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t))\end{aligned}$$

where η is the learning rate.

Then each iteration t involves cycling through the training data with the updates:

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda\mathbf{w}_t - y_i\mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \eta\lambda\mathbf{w}_t && \text{otherwise}\end{aligned}$$

In the Pegasos algorithm the learning rate is set at $\eta_t = \frac{1}{\lambda t}$

Outline

1. **Introduction**
2. **Linear separators and Loss functions**
    1. Review of linear separators
    2. Loss Functions
3. **Perceptron**
    1. Perceptron learning algorithm
    2. Perceptron learning graphically speaking
    3. Perceptron Extensions
4. **Multinomial/Multiclass linear classifiers**
5. **Support vector machines (SVMs)**
    1. From perceptron to SVM (briefly)
6. **Summary**

End of lecture

Learning a perceptron

- **Step 1: Augment dataset**
- **Step 2: “label” normalize the data**
- **Step 3: Do gradient descent**

STEP 1: Augmented Representations and Classification Rule

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	-1
2					+1
…	…	…	…	…	…
L (aka m)	0	4	…	8	-1

Augmented Dataset

Instance\Attr	x_0	x_1	x_2	…	x_n	y
1	1	-3	0	..	-7	-1
2	1					+1
…	1	…	…	…	…	…
L (aka m)	1	0	4	…	8	-1

Hyperplane as an Augmented weight vector

$$W = \begin{bmatrix} w_0 \\ w_1 \\ .. \\ w_n \end{bmatrix} = \begin{bmatrix} b \\ w_1 \\ .. \\ w_n \end{bmatrix}$$

Augmented Data vector

$$X = \begin{bmatrix} x_1 \\ .. \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ .. \\ x_n \end{bmatrix}$$

$$Class\ (X) = sign(\langle W, X \rangle + b)$$

Classification rule simplifies

$$Class\ (X) = sign(\langle W, X \rangle)$$

Augmented Representations and Classification Rule

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	-1
2					+1
…	…	…	…	…	…
L (aka m)	0	4	…	8	-1

Hyperplane as an Augmented weight vector

$$W = \begin{bmatrix} w_0 \\ w_1 \\ .. \end{bmatrix} = \begin{bmatrix} b \\ w_1 \\ .. \end{bmatrix}$$

Augmented Data vector

$$X = \begin{bmatrix} x_1 \\ .. \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ .. \end{bmatrix}$$

Augmented Dataset

Step 2:
Multiply each negative example by -1 X y_i

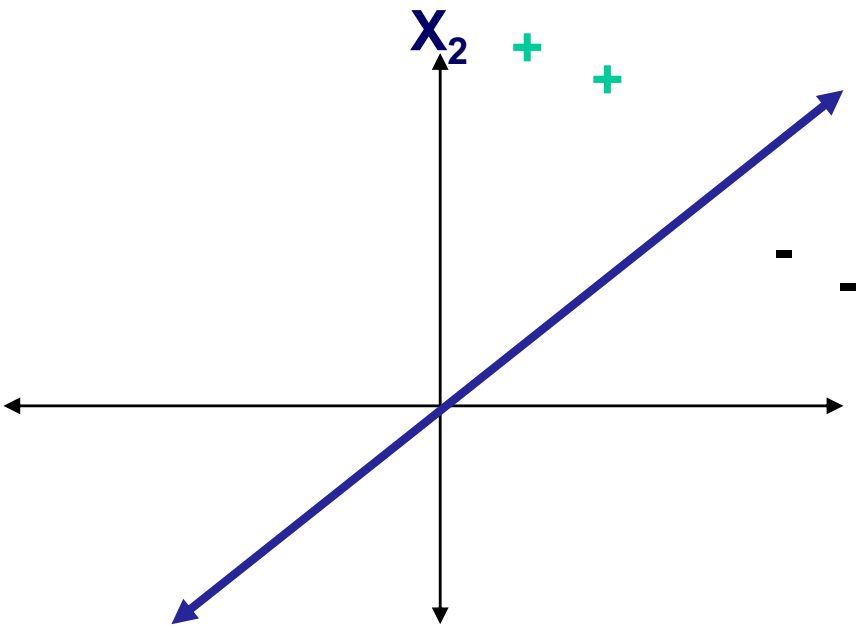
Instance\Attr	x_0	x_1				
1	-1	-3	0	..	-7	-1
2	1					+1
…	1	…	…	…	…	…
L (aka m)	-1	0	-4	…	-8	-1

Classification rule simplifies

$$Class\ (X) = sign(\langle W, X \rangle)$$

Label Normalization 1/3

Positive Side/Halfspace



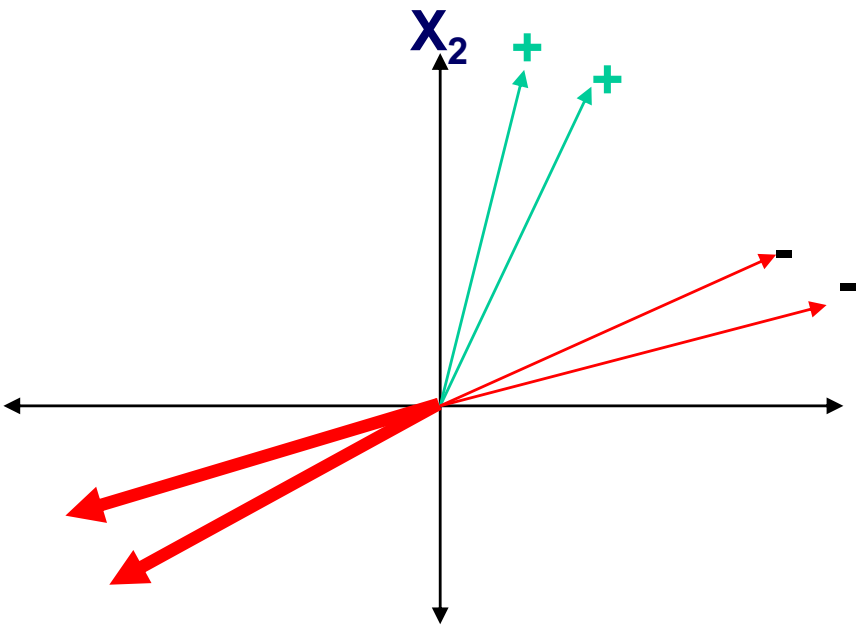
Hyperplane has to go thru the origin

Xy

X x -1

Label Normalization 2/3

Positive Side/Halfspace

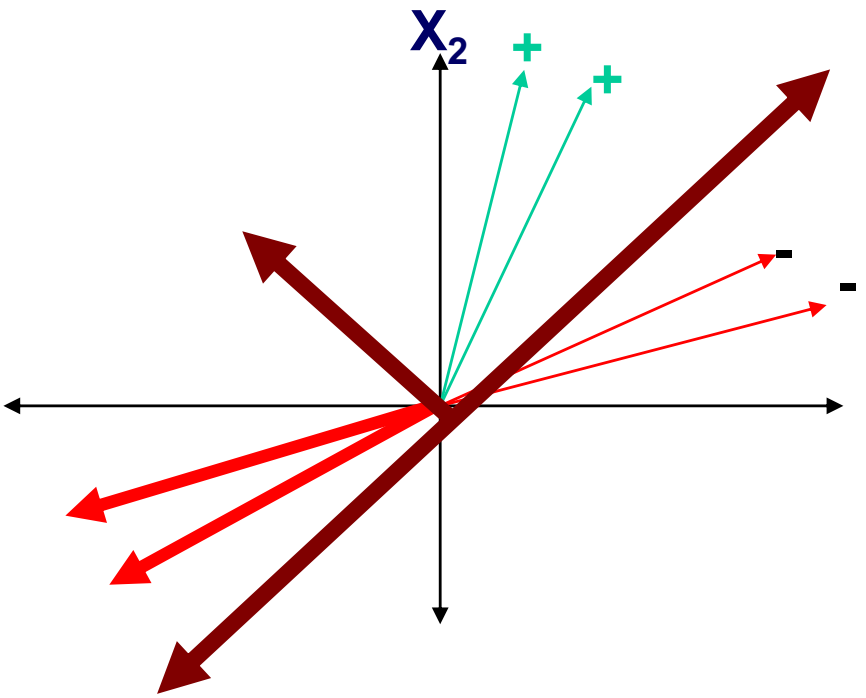


Xy
X x -1

Label Normalization 3/3

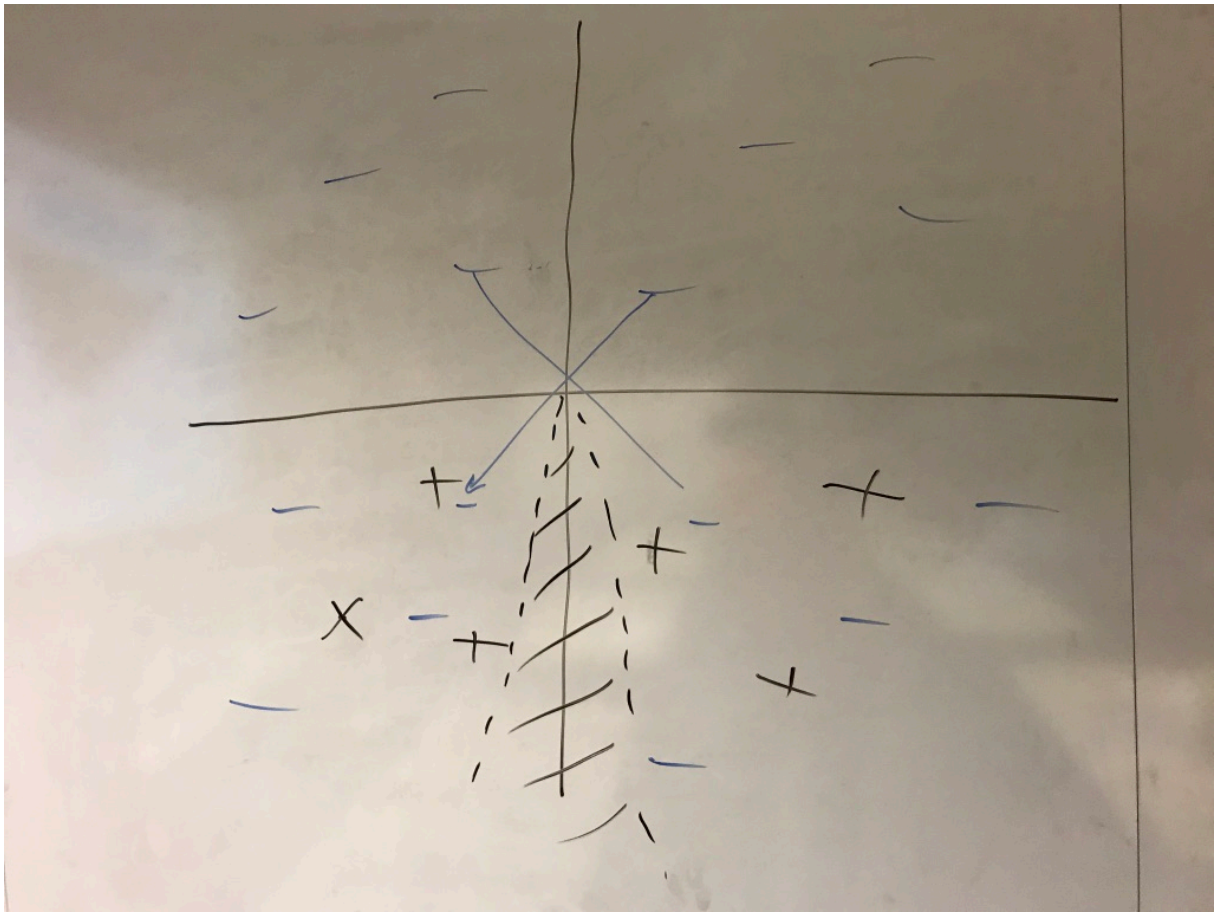
All the label normalized all fall on the positive side of the hyperplane

Positive Side/Halfspace



Xy
X x -1

Extreme case for linearly sep data

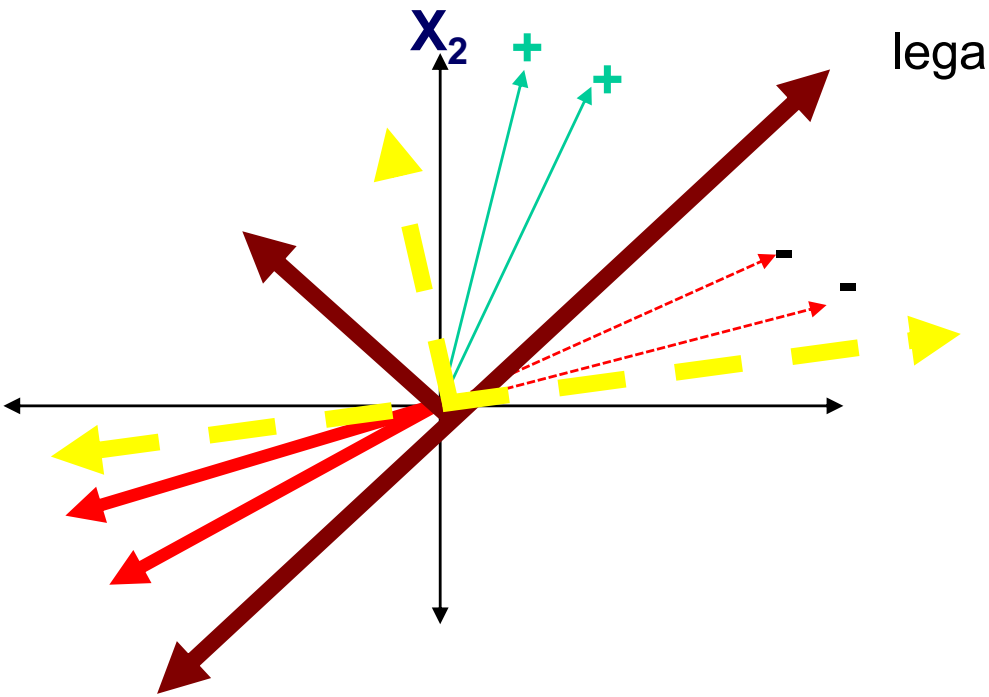


[Thanks Samantha, Minneapolis]

Label Normalization 3/3

All the label normalized fall on the positive side of the “legal” hyperplane (for linearly separable data)

Positive Side/Halfspace



legal

Xy
X x -1

This dashed YELLOW hyperplane does not classify the data properly. Note the red vectors (negative class examples) are on the negative side of the hyperplane. Viewed another way the dotted red vectors (un-label-normalized negative examples) are on the positive side of the dashed hyperplane.