Support Vector Machines

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 XXX**
-

- Multiclass Classification: News categorization (Microsoft, 2018)
  - https://gallery.azure.ai/Experiment/Multiclass-Classification-News-categorization-2

HW Assignment

- **Vast array of approaches to learning SVMs and a plethora of very successful applications**
- **Math is intense**
- **Key concepts**
  - Margin , kernel trick
  - If you get lost come back to the leartning objectives or skip to the app section
- **SVM via gradient descnt**
  - Data Loss and regularization
- **Multinomial perceptron versus Logistic regression**
- **Optional: Implement a multinomial perceptron**

Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual SVM algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

- In machine learning, **support vector machines** (**SVMs**, also **support vector networks**[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.
- In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

SVMs: maximum margin classifiers

- SVMs belong to a family of generalized linear classifiers and can be interpreted as an extension of the perceptron.
- They can also be considered a special case of Tikhonov regularization.
- SVMs minimize the empirical *classification error* and maximize the *geometric margin*;
  - A special property is that they simultaneously minimize the empirical *classification error* and maximize the *geometric margin*; hence they are also known as maximum margin classifiers.

History of SVMs

- **1963: original SVM algorithm**
  - The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963.
- **1992: Nonlinear SVMs and Kernel trick**
  - In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes.[12]
- **1993: Soft margin SVMs**
  - The current standard incarnation (soft margin) was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.[1]

SVMs

- **SVMs**
  - classification, regression
  - More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection.
- Soft-margin support vector machine is an example of an empirical risk minimization (ERM) algorithm for the *hinge loss*.
- **Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.**

SVMs are vast

- Recent algorithms for finding the SVM classifier include sub-gradient descent and coordinate descent. Both techniques have proven to offer significant advantages over the traditional approach when dealing with large, sparse datasets—sub-gradient methods are especially efficient when there are many training examples, and coordinate descent when the dimension of the feature space is high.
- Vast subject area with lots of extensions and applications

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

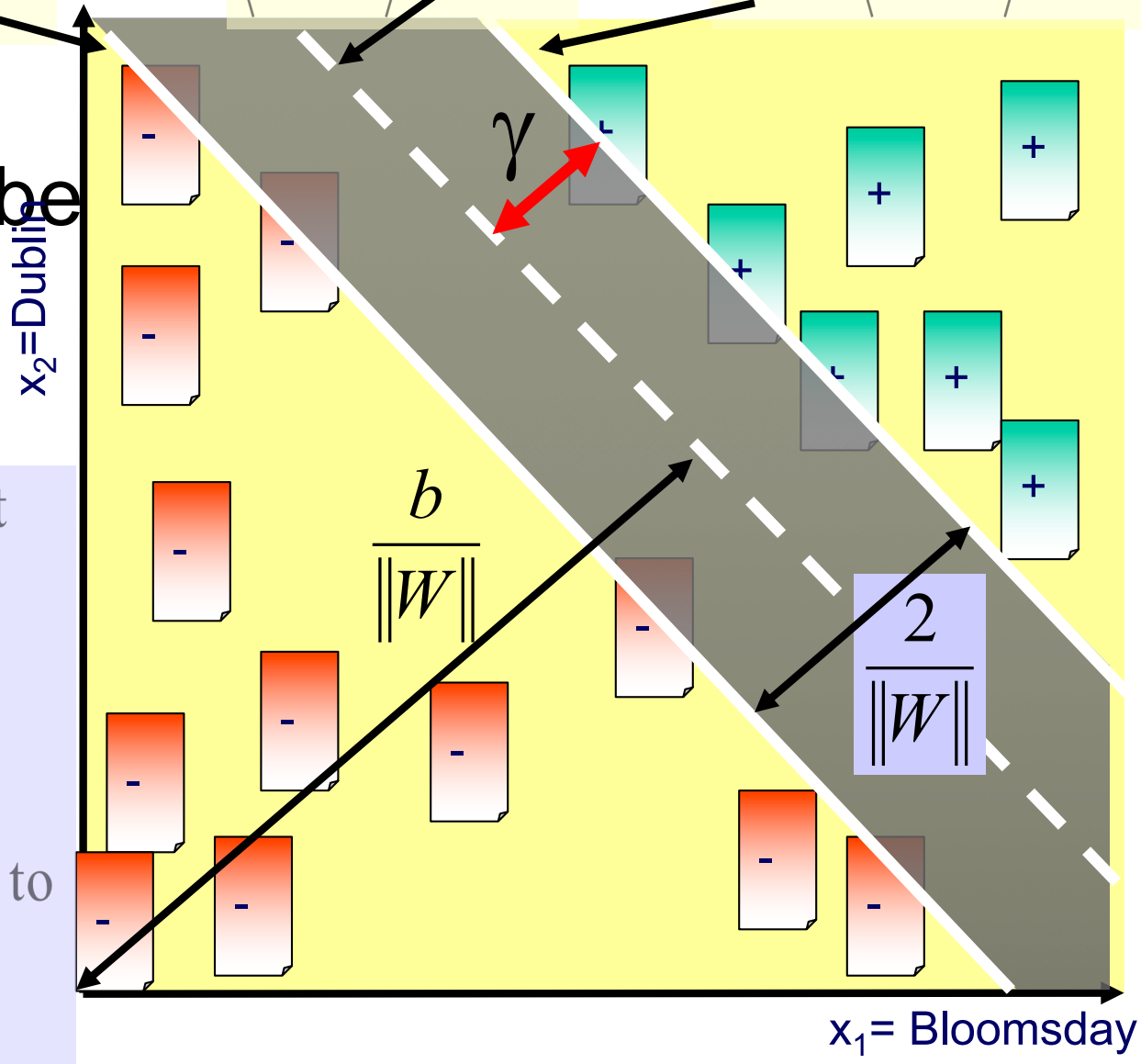
Find H_1 and H_2 such that

$Dist(H_1, H_2)$ is MAX

Or

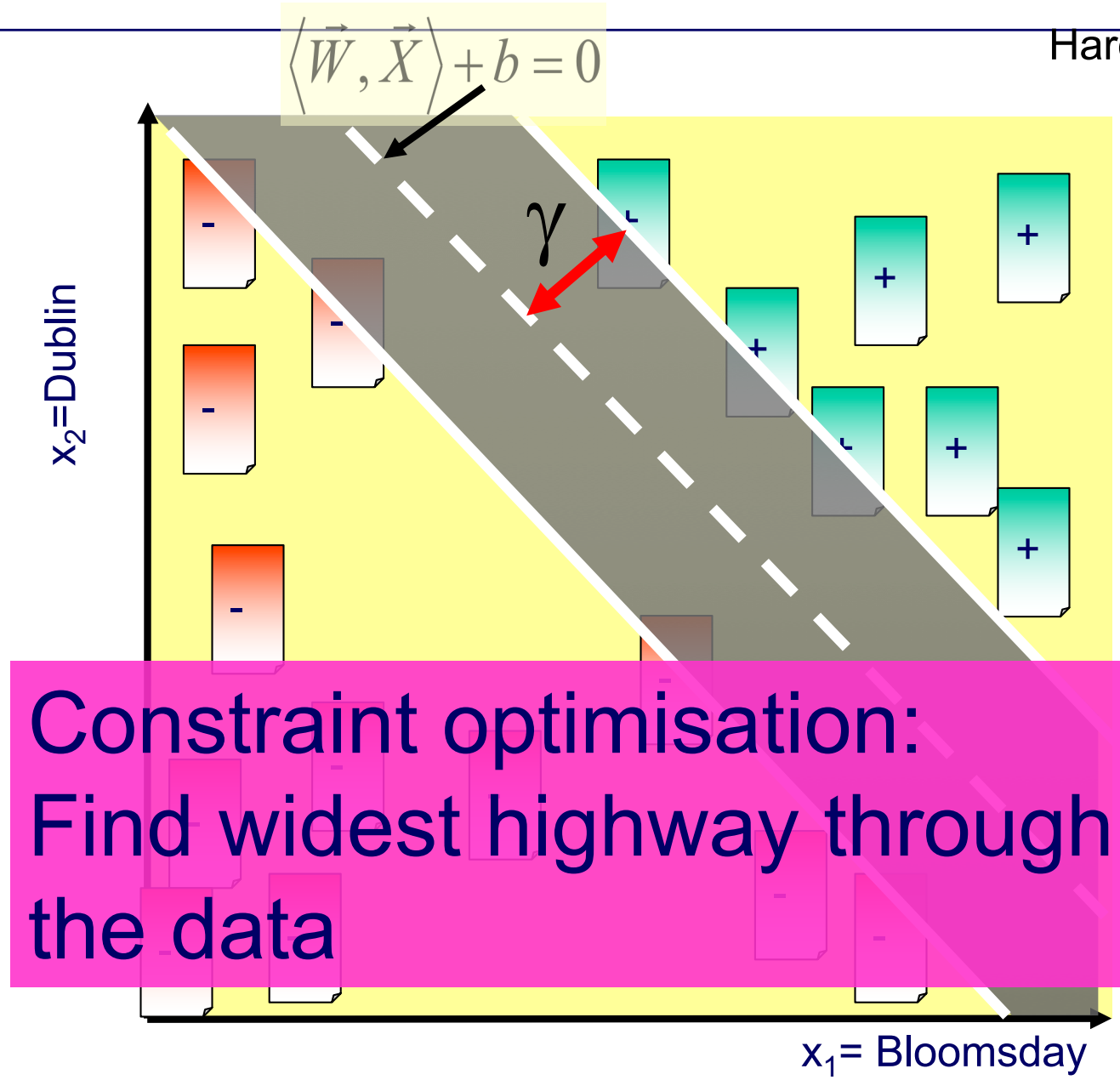
Minimize $\frac{\|\vec{W}\|^2}{2}$ subject to

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



Movie Classifcation

Hard SVMs

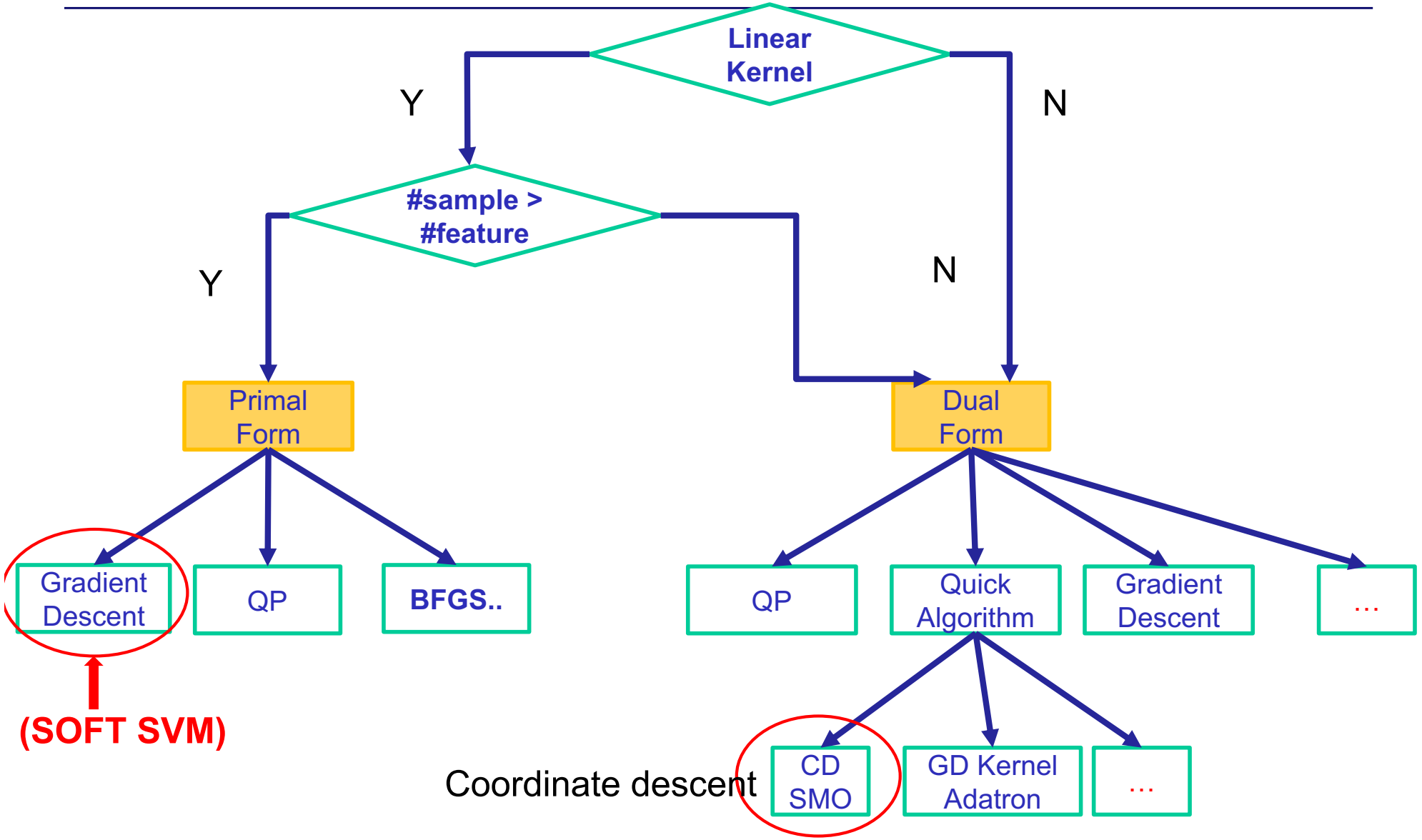


Constraint optimisation:
Find widest highway through the data

Kernels Trick

- In Support Vector Machines, the input is mapped by a nonlinear function to a highdimensional space, and the optimal hyperplane found, the one that has the largest margin.

SVM Learning Algorithm Taxonomy



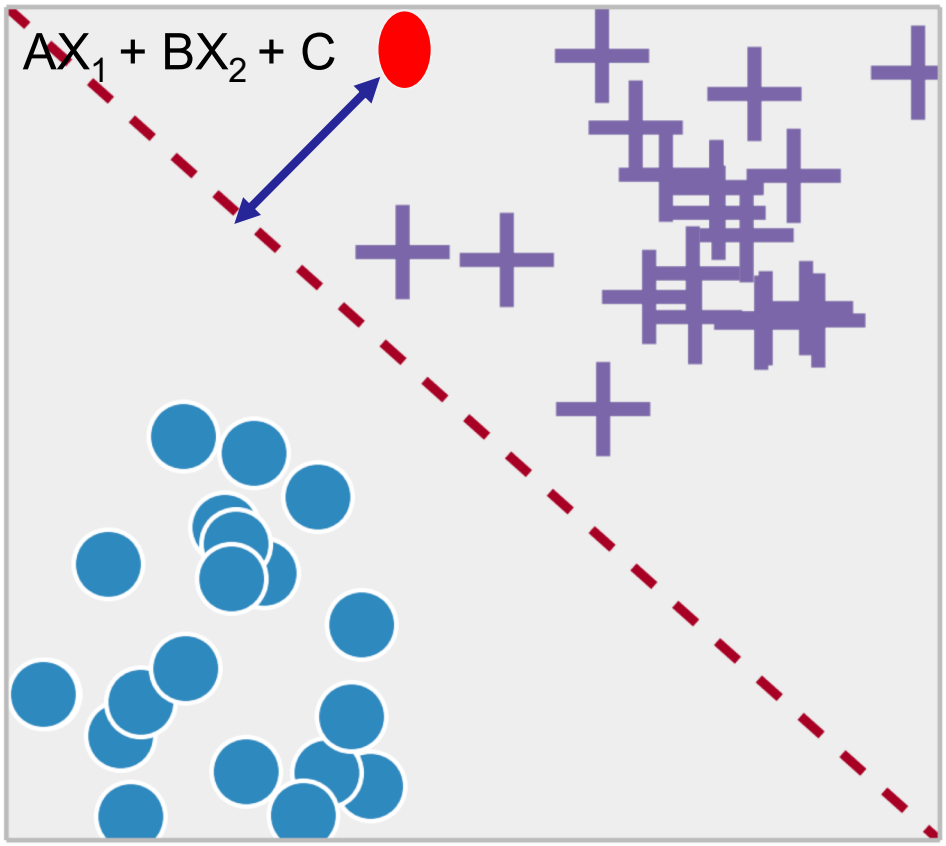
Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual perceptron algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

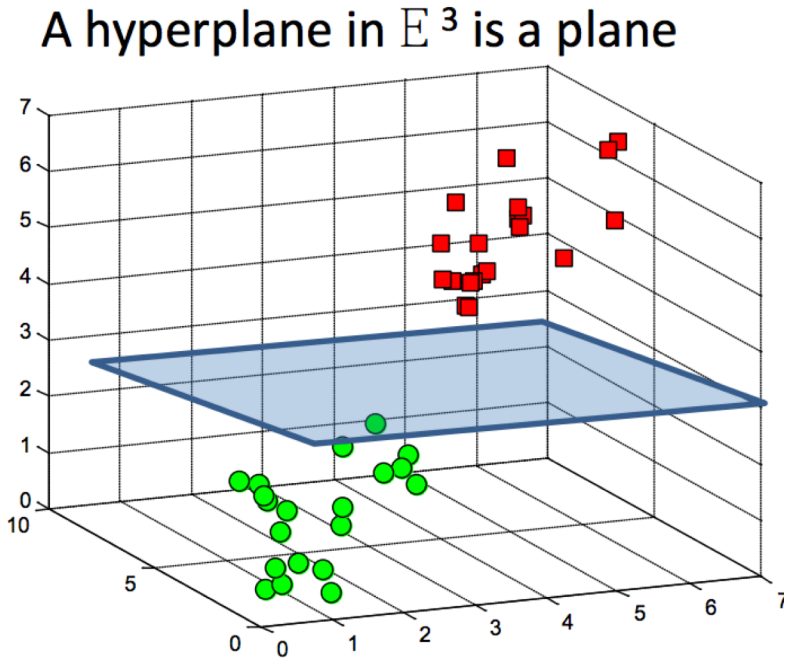
Learn n-1 dim hyperplane

Perpendicular Distance from a point to a hyperplane

Classification



Minimize Residuals



n-1 dimensional subspace

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

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

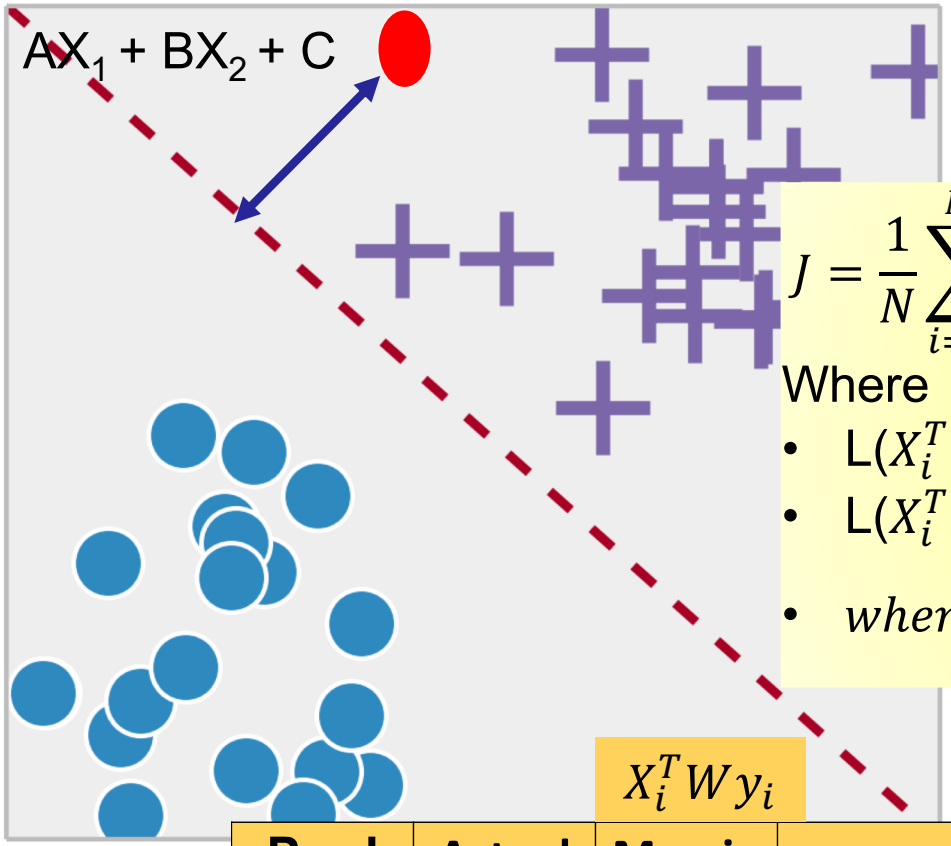
Learn n-1 dim hyperplane

Perpendicular Distance from a point to a hyperplane

$$margin = X_i^T W y_i$$

Minimize Residuals

Classification



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

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

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

n-1 dimensional subspace

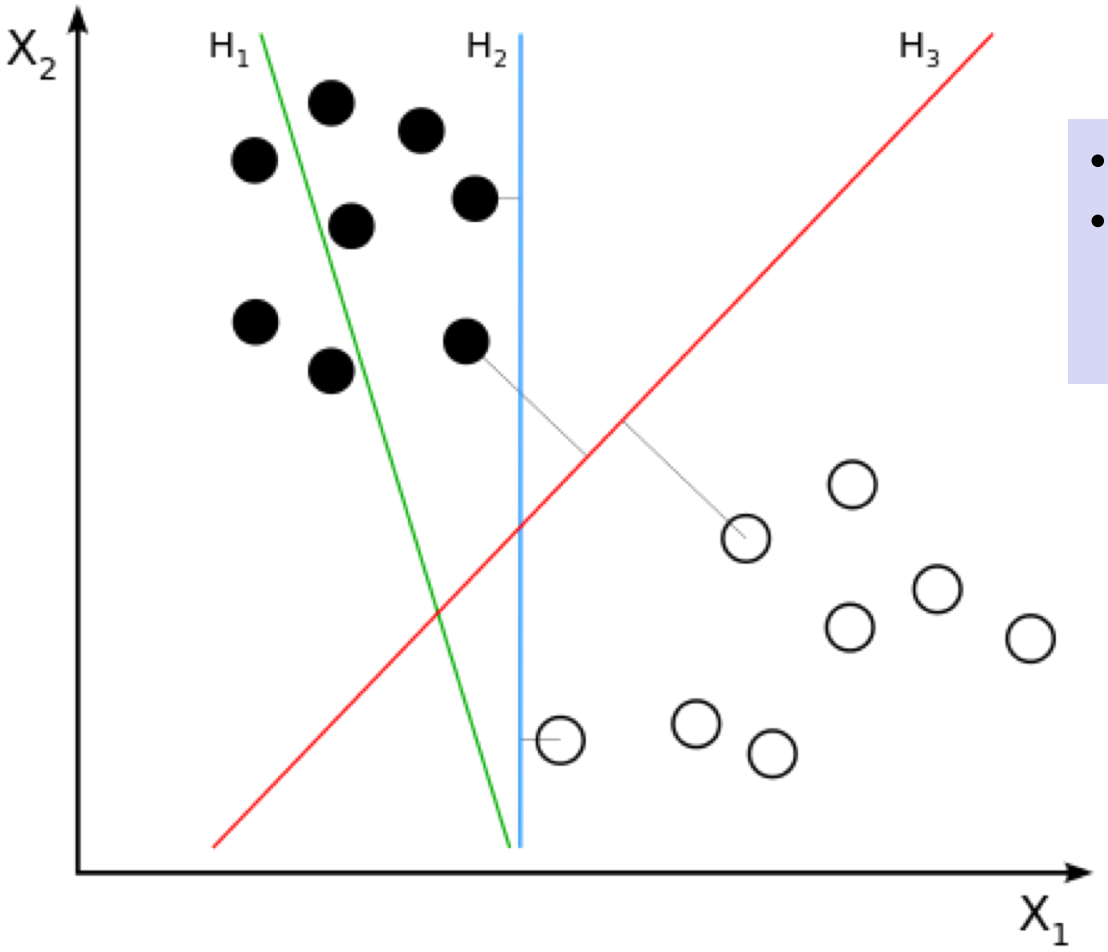
$f(x) = hea$

$where\ \ hea$

$X_i^T W y_i$

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

Binary Linear Classifier: many possibilities



- Data lives in n-space;
- Partition such points using am n–1 dimensional hyperplane

H_1 does not separate the classes.
H_2 does, but only with a small margin.
H_3 separates them with the maximum margin.

linear classifier

- Classifying data is a common task in machine learning.
- Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new Data point will be in.
- In the case of support vector machines, a data point is viewed as a n-dimensional vector (a list of n numbers), and we want to know whether we can separate such points with a n – 1 dimensional hyperplane. This is called a linear classifier.
- There are many hyperplanes that might classify the data.

Empirical Risk Minimization(ERM)

- Provides a criterion to decide on h:

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} loss(\boldsymbol{x}_i, \boldsymbol{y}_i; h)$$

- Background preferences over h can be included in **regularized** empirical risk minimization:

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} loss(\boldsymbol{x}_i, \boldsymbol{y}_i; h) + R(h)$$

Perceptron and SVM have similar loss functions

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

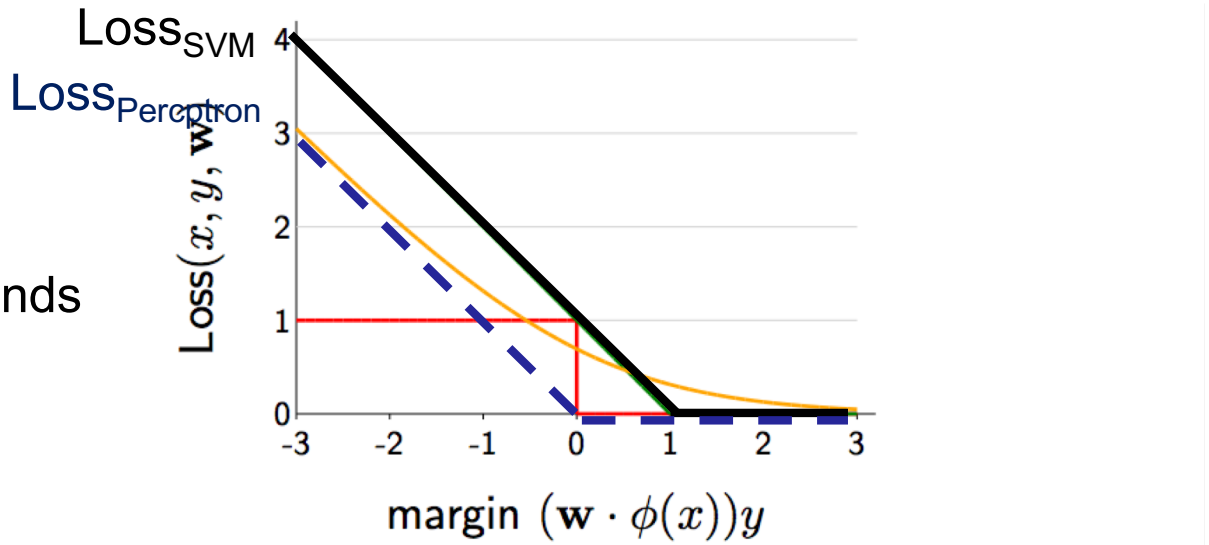
2. $\nabla Loss_{P\_Hinge}(W) = \begin{cases} 0 & if\ X_i^T W y_i > 0 \\ -X_i^T y_i & otherwise \end{cases}$

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

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

Lagrangian

Hinge loss upperbounds
0-1 loss



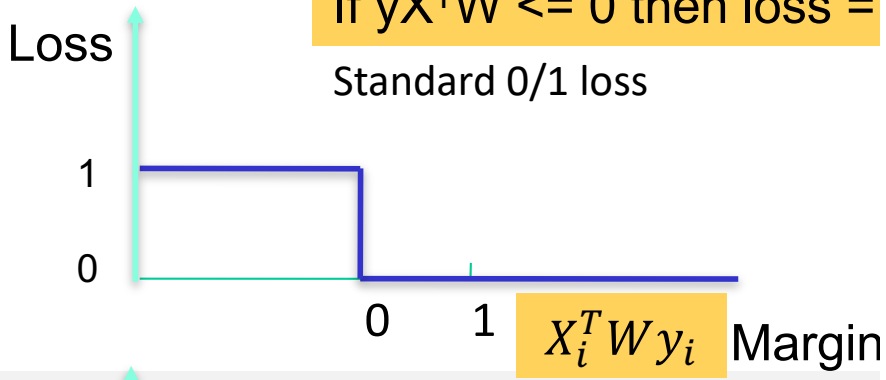
- Intuition: Try to increase margin even when it already exceeds 1

SVM hinge Loss function

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

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

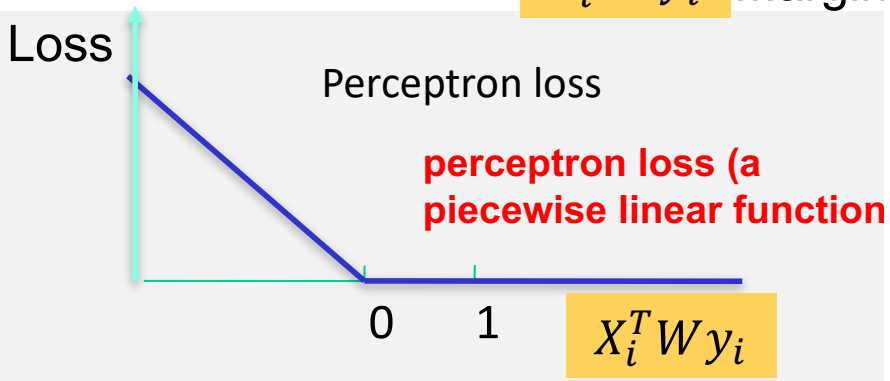
Penalizes all incorrectly classified examples with the same amount



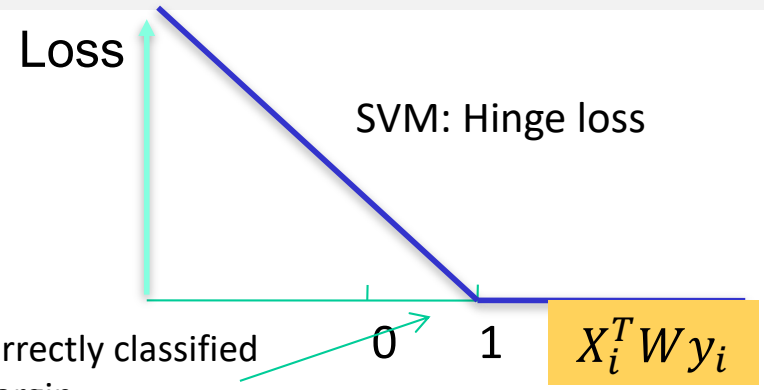
Penalizes incorrectly classified examples X proportionally to the size of $|X^Tw|$

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

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



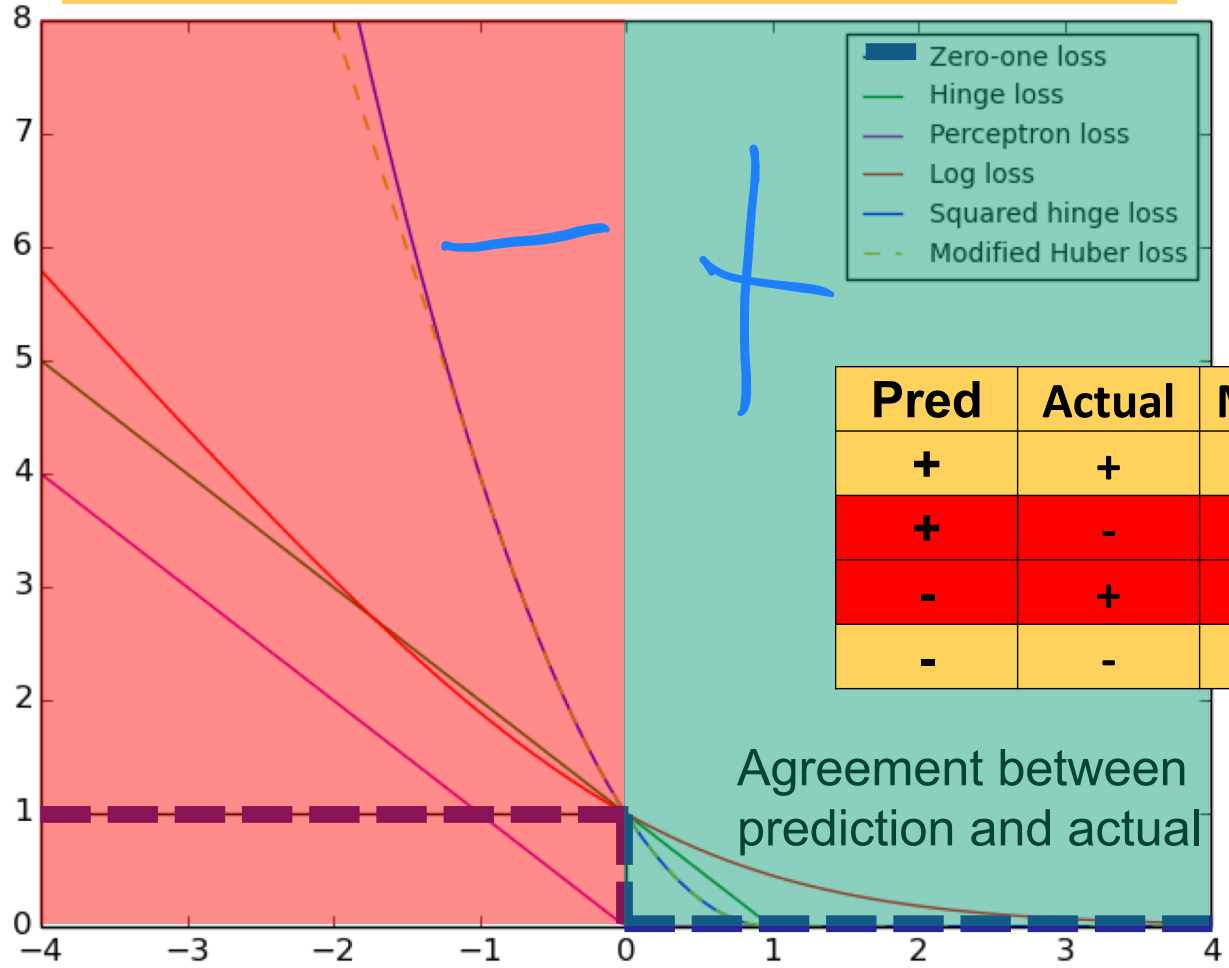
Penalizes incorrectly classified examples and correctly classified examples that lie within the margin



Examples that are correctly classified but fall within the margin

Machine Learning Objective Function: classification

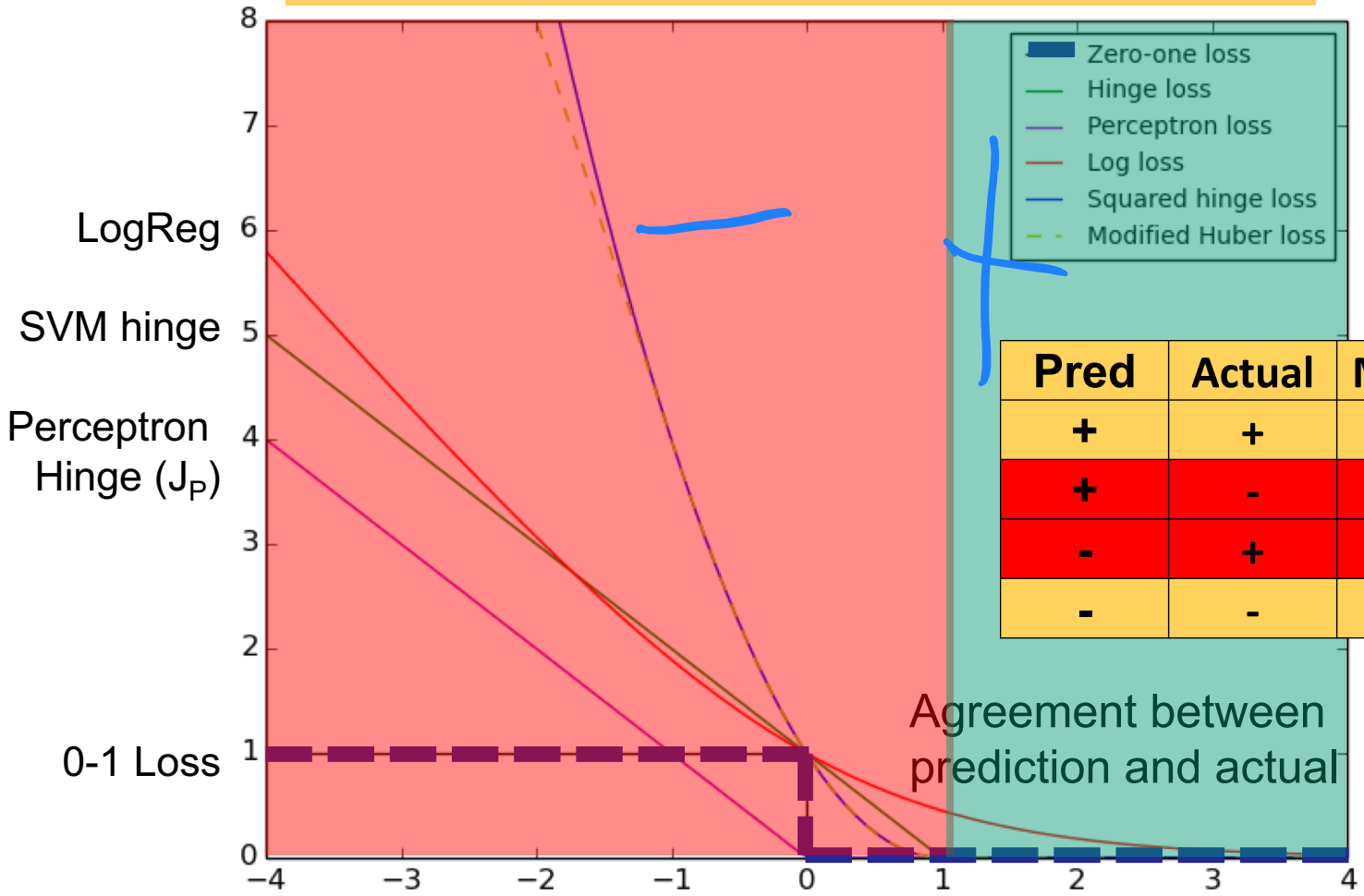
Classification loss functions



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

SVM data Loss

Classification loss functions



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

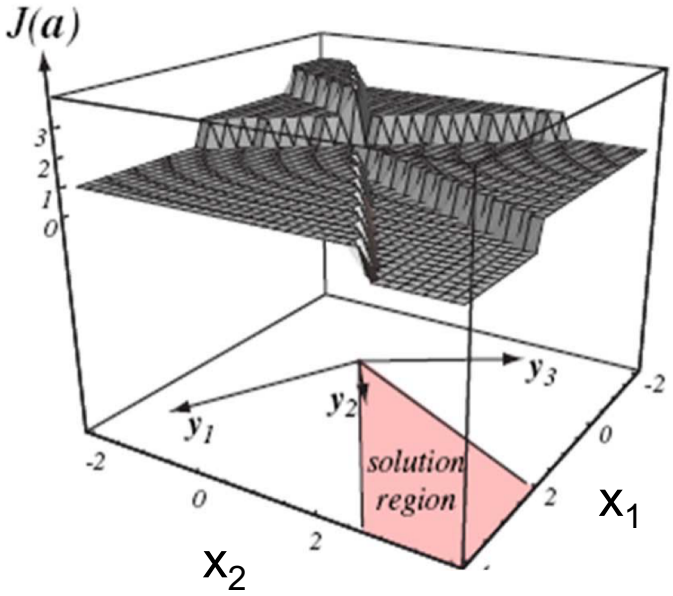
SVM Loss Function

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

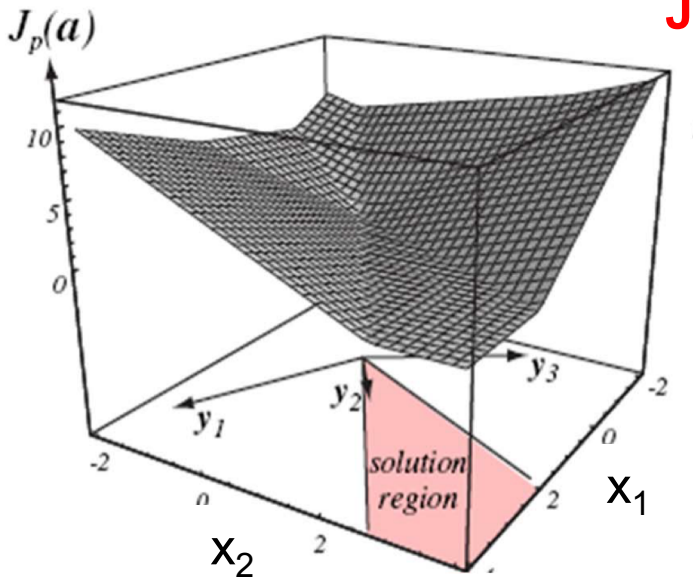
$$J_P = mimimize\left[\frac{1}{N}\sum_{i=1}^{N} Max\left(0, -X_i^T W y_i\right)\right]$$

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

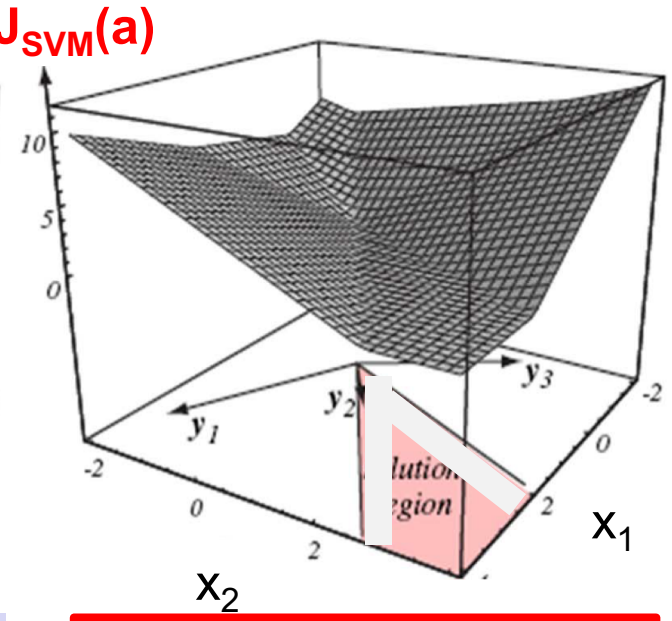
- The term $\boldsymbol{Max\left(0, -X_i^T W y_i\right)}$ s 0 when y_i is predicted correctly otherwise it is equal to the "confidence" in the mis-prediction
- Has a nice gradient leading to the solution region



0/1 loss



Perceptron criterion



SVM criterion

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.

Deriving the stochastic gradient descent algorithm for the SVM

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

Gradient is a vector of partial derivative

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

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

SVM learning rule, learning rate η=1

Online SVM 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 $(X_i^T w y_i \leq 1)$: w = w + $y_i \mathbf{x}_i$

Does not work well

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.

- **Does not work well**

- **Looking for something better and a more general framework that allows us to develop many different types of learning algos**

Prefer NOT to Augment: makes math easier Kernel trick etc.

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

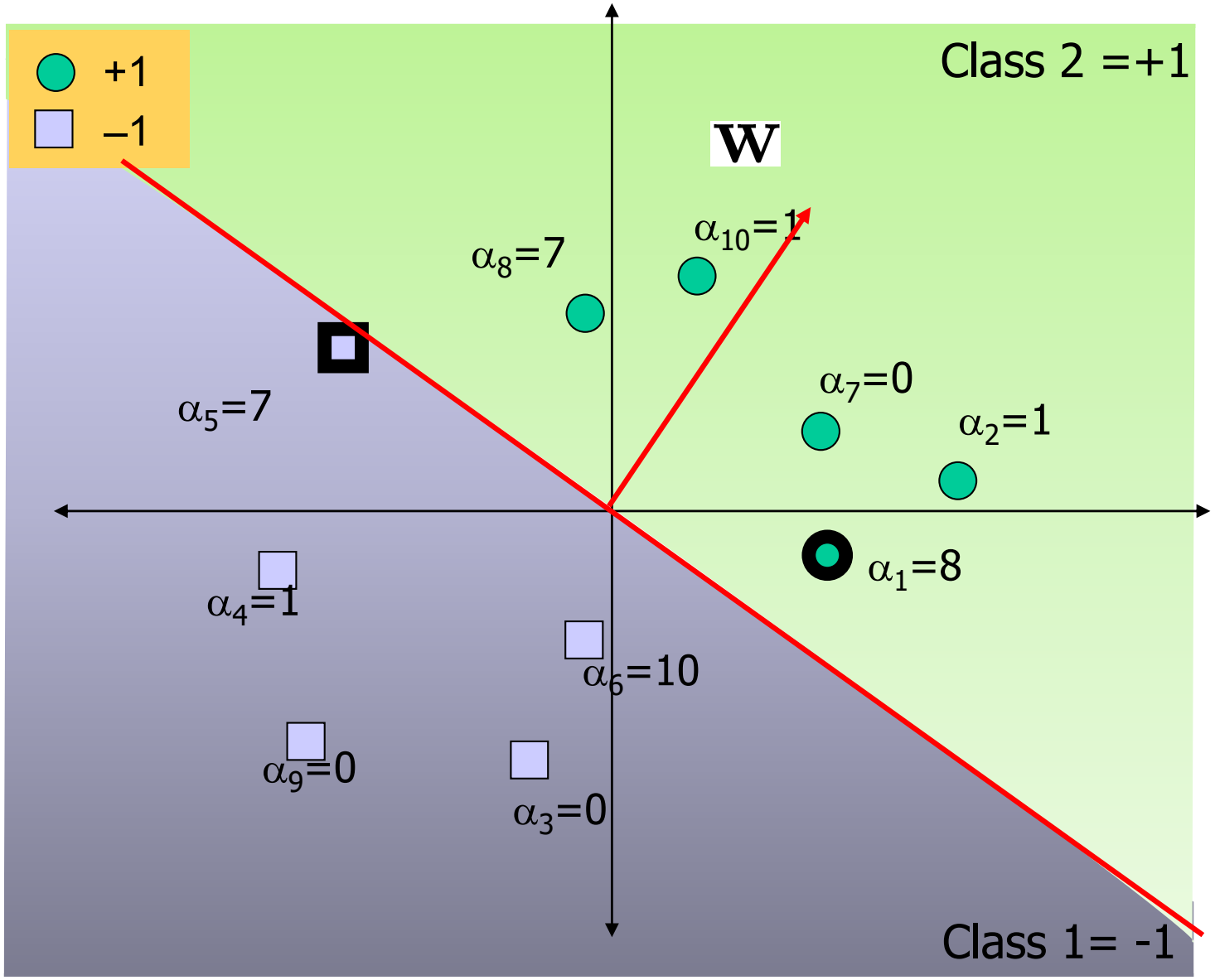
Learn a linear function $g(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} = \langle w_0, w_1, w_2, w_3, w_4 \rangle$

- 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

Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual perceptron algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

Perceptron: no more mistakes

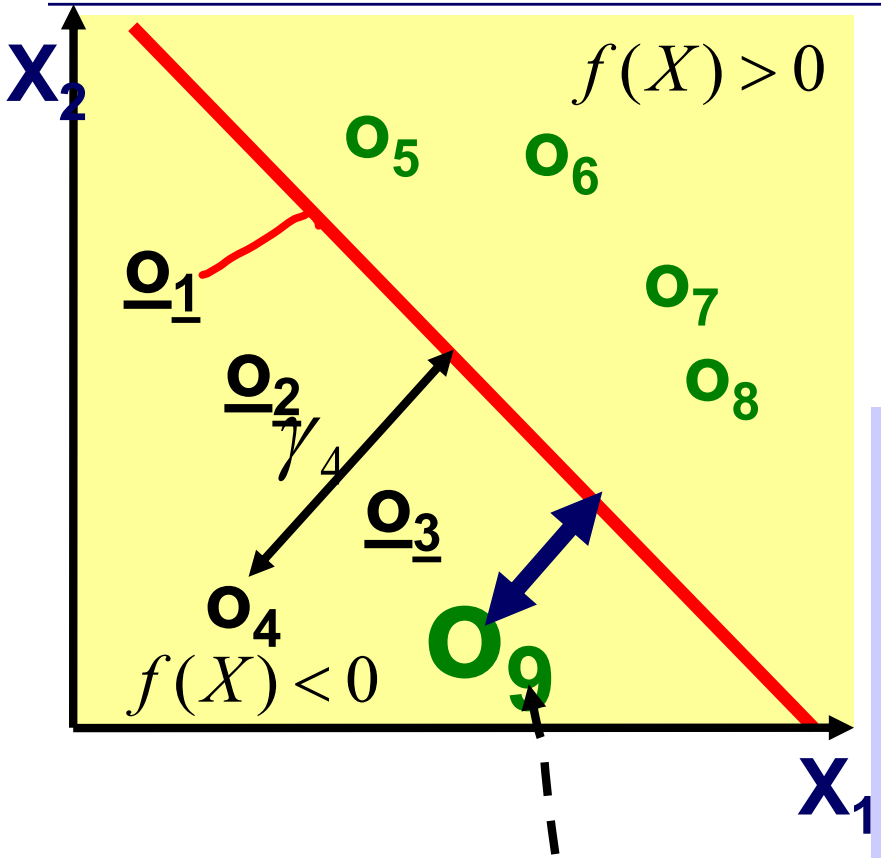


$X_i^T W > 0$

$X_i^T W = 0$

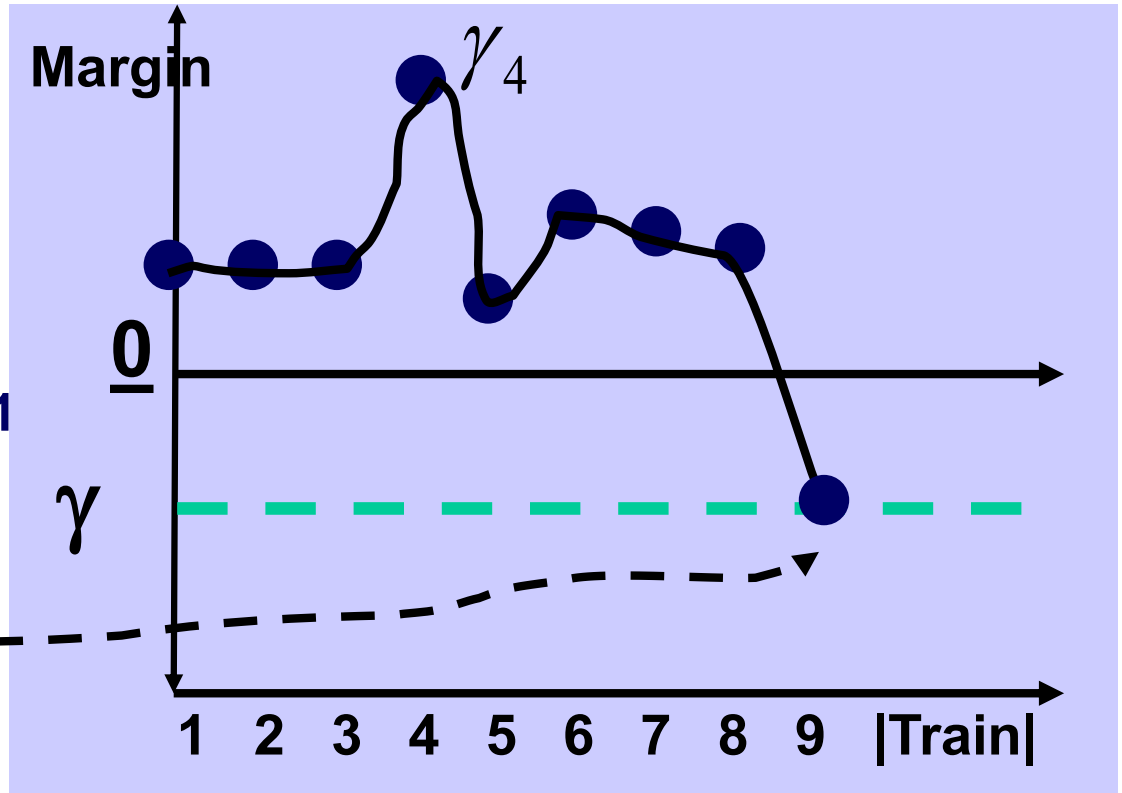
$X_i^T W < 0$ Violet

Margin and Margin Distribution



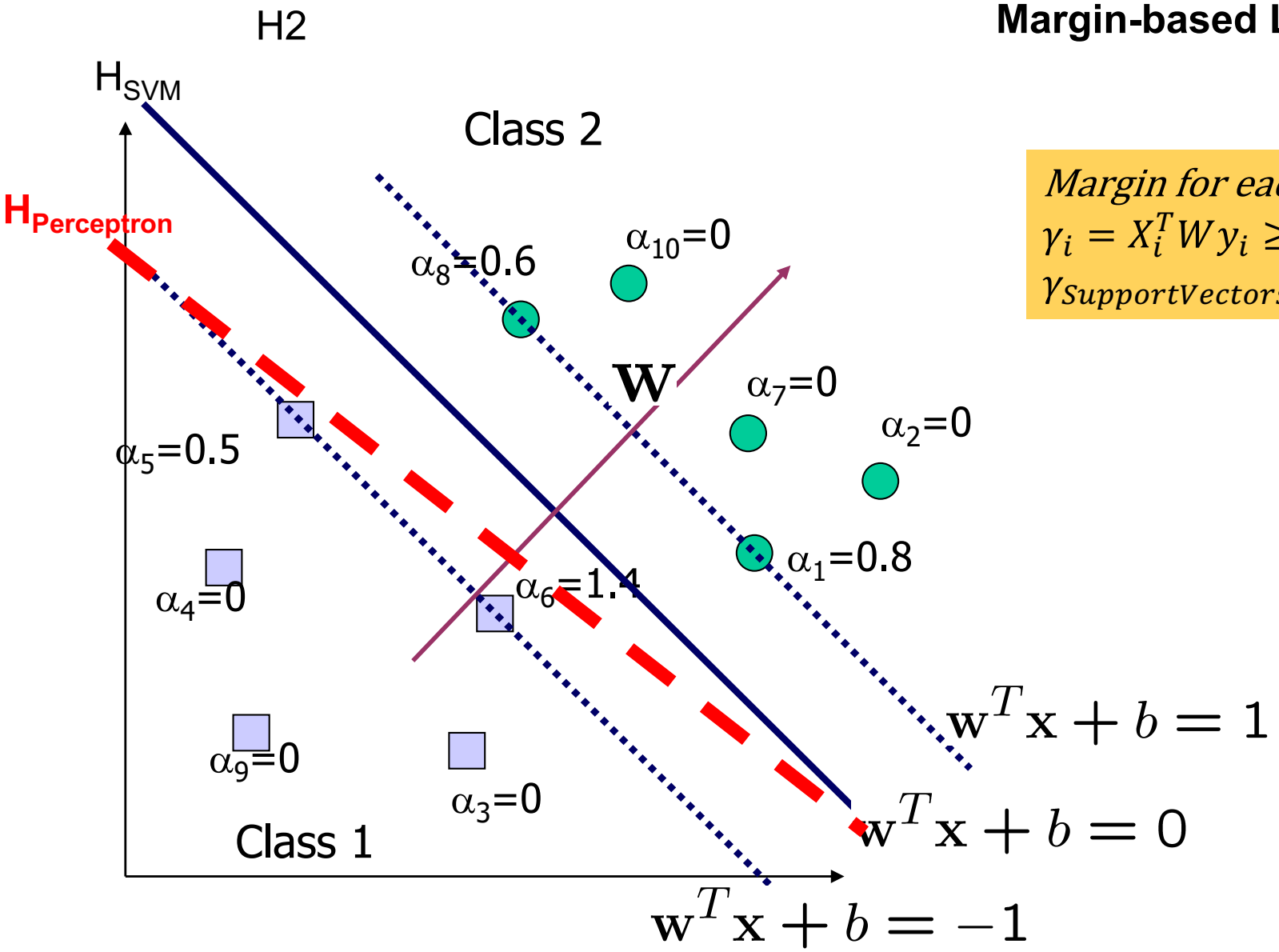
Take any linear separator (e.g., a logistic regression or perceptron model)

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



Negative Margin for example o9

Perceptron vs SVM Learning
The search for a separating hyperplane

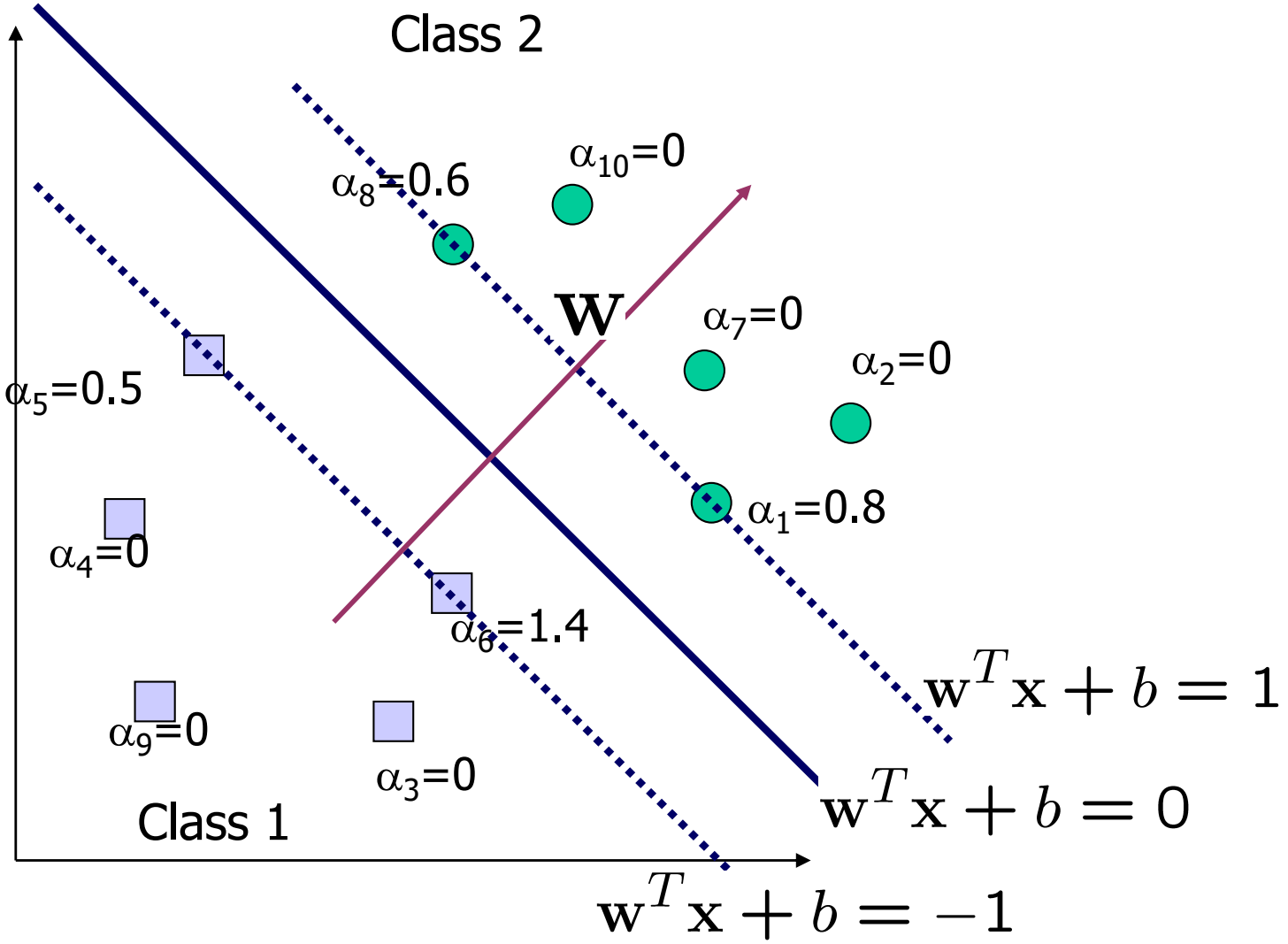


Margin for each example

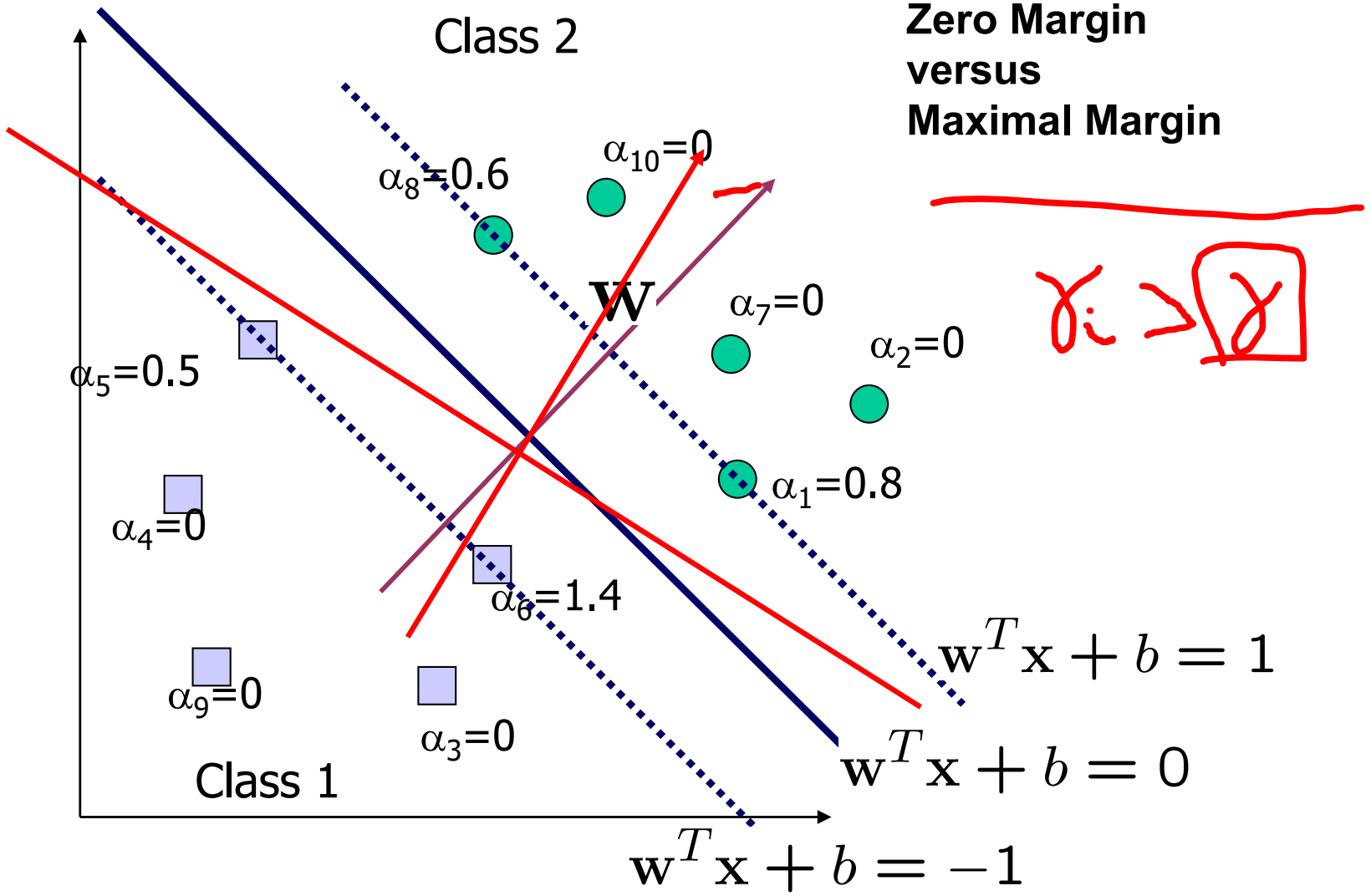
$\gamma_i = X_i^T W y_i \geq 1$

$\gamma_{SupportVectors}$=1

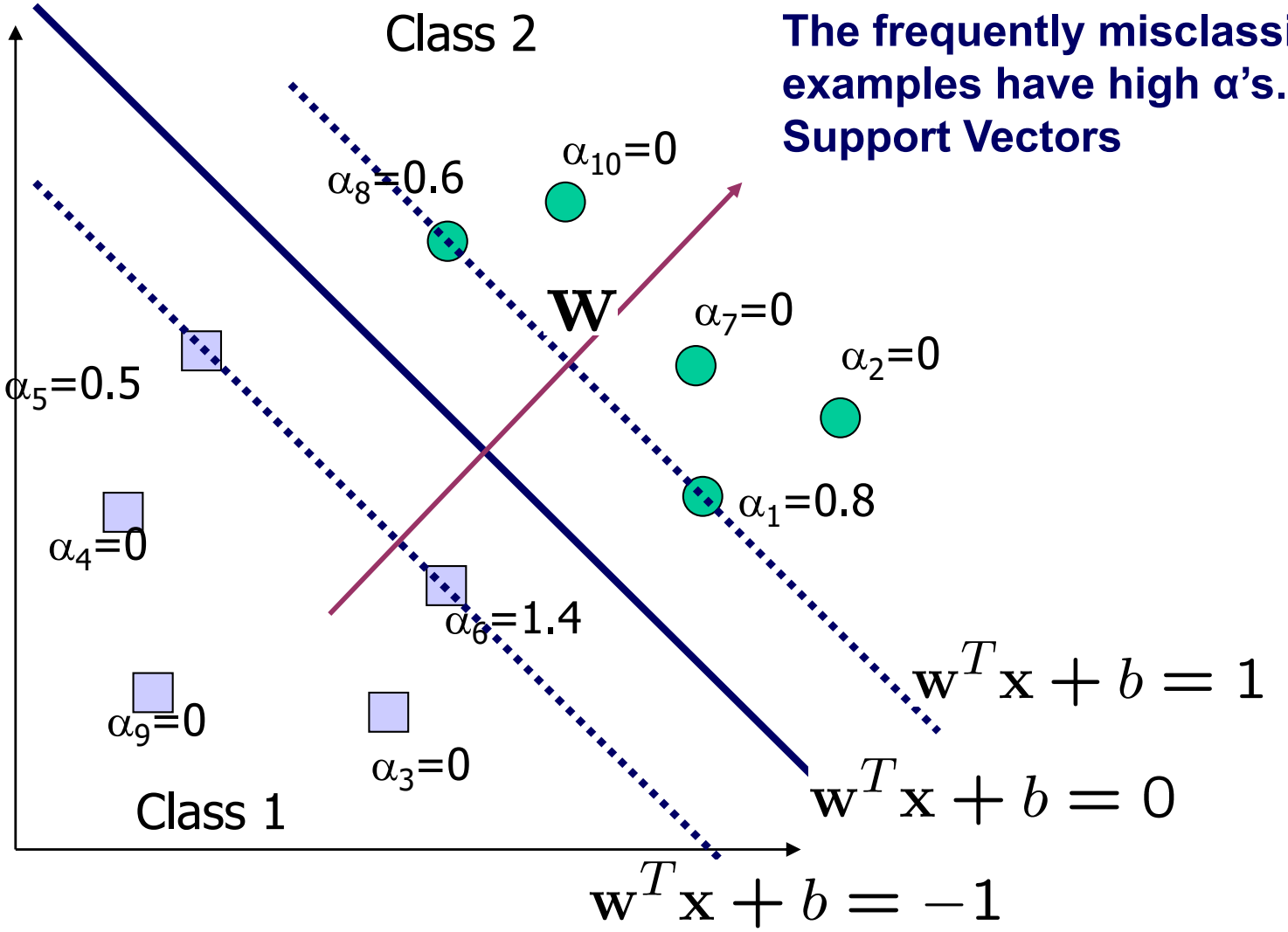
Margin-based Perceptron Learning



Margin-based Perceptron Learning



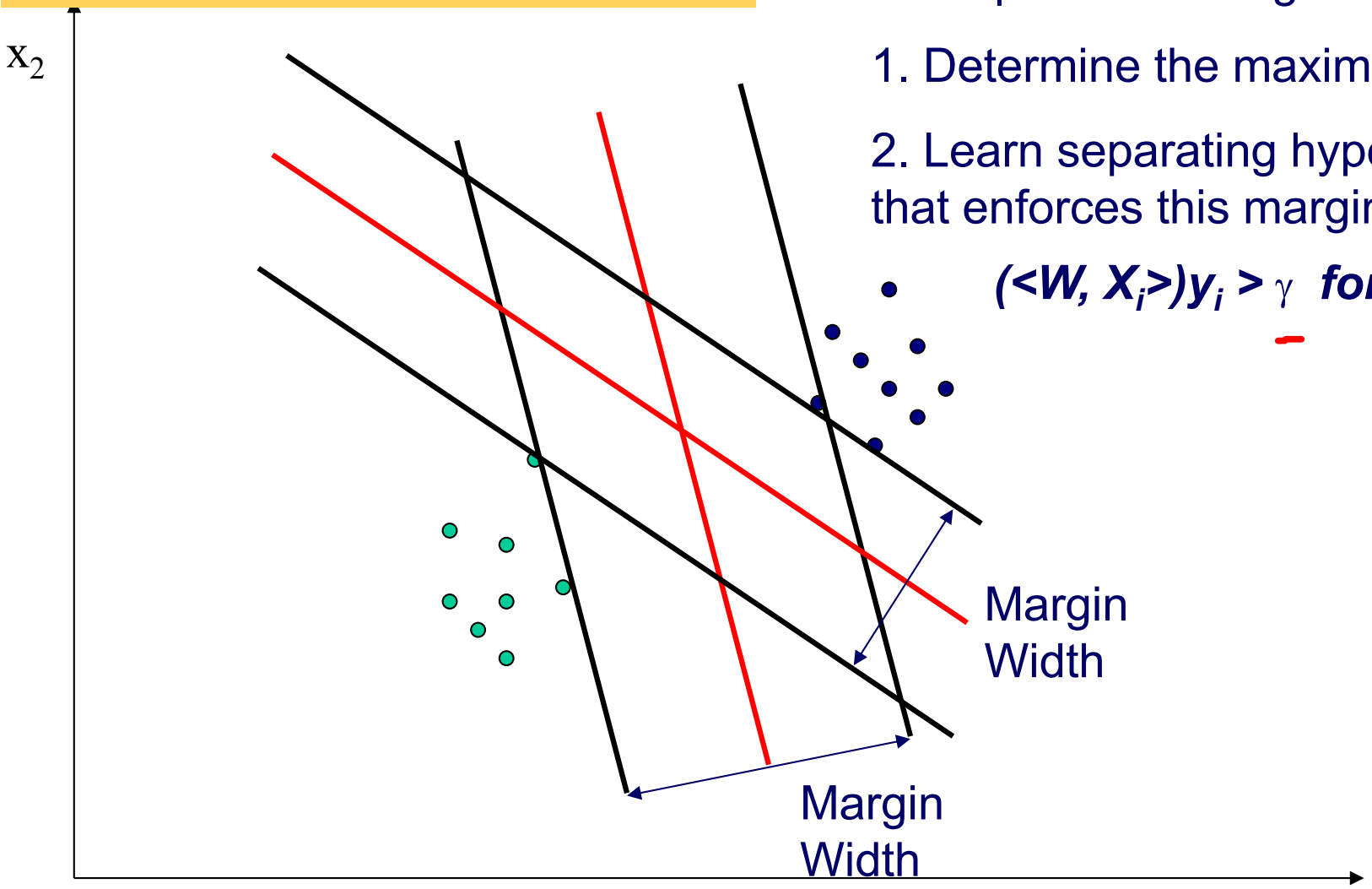
Margin-based Perceptron Learning



The frequently misclassified examples have high α's. Call them Support Vectors

Maximizing the Margin

If we could Determine margin
we could then enforce it using the PA



Chicken-Egg Conundrum for Perceptron Learning:

1. Determine the maximum margin γ
2. Learn separating hyperplane, W, that enforces this margin

$$(\langle W, X_i \rangle) y_i > \gamma \quad \text{for } I = 1, \ldots, L$$

Functional Margin for a Training Set

- **Define the margin of an example *i* with respect to a hyperplane *(W, b)* to be the quantity $\gamma_{fun,i}$:**

$$\mathbf{x}_i\mathbf{w} + b \geq +\gamma_{Fun} \quad \forall\ y_i = +1,$$

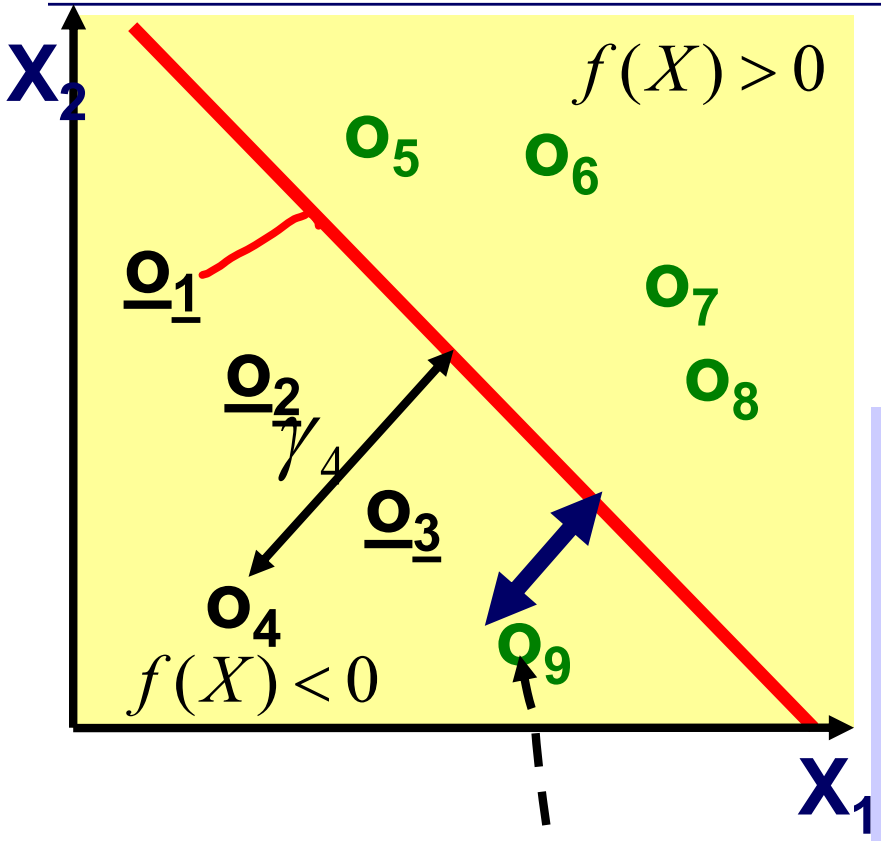
$$\mathbf{x}_i\mathbf{w} + b \leq -\gamma_{Fun} \quad \forall\ y_i = -1,$$

$$\text{or equivalently, } y_i(\mathbf{x}_i\mathbf{w} + b) \geq \gamma_{Fun} \quad \forall\ i = 1,\ldots,L.$$

- **If $\gamma_{fun,i} > 0$ then correct classification of example (X_i, y_i).**
- **The margin distribution is the distribution of margins of the examples in a training dataset S.**
- **The minimum of the margin distribution corresponds to the margin of the hyperplane (W, b) w.r.t S.**

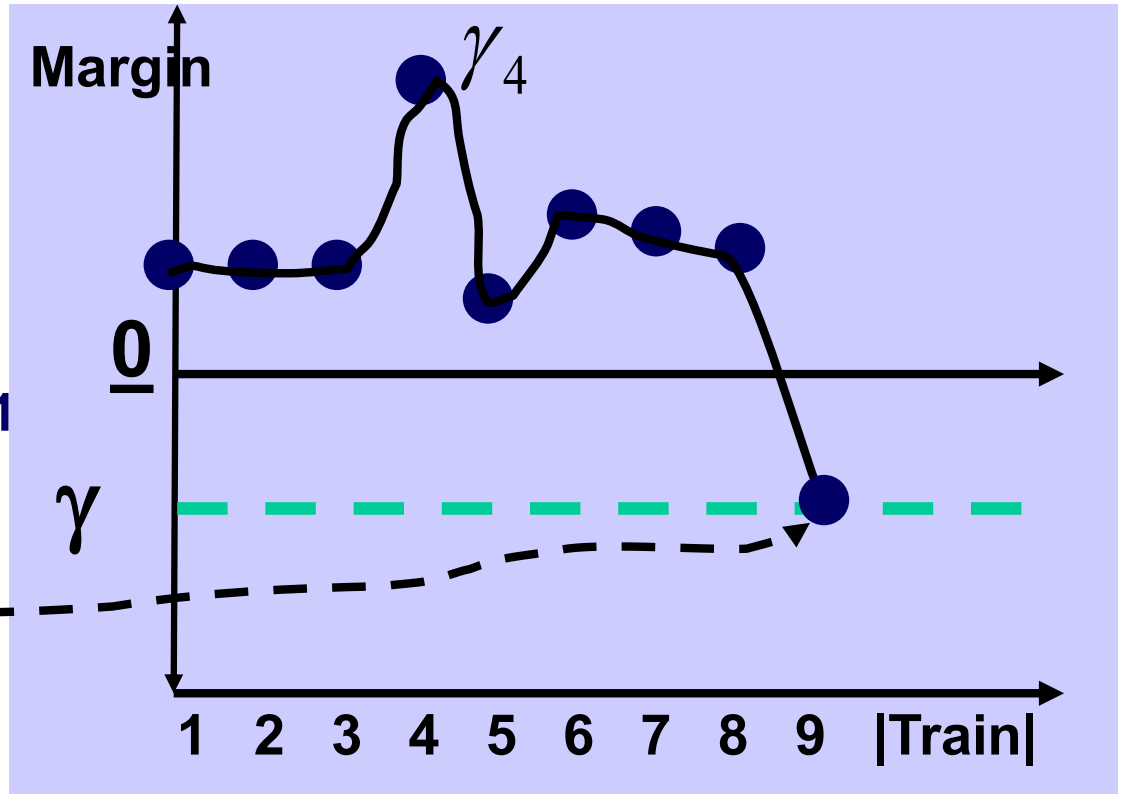
Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Margin and Margin Distribution



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

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



Negative Margin for example o9

Maximum Margin Classifier: Take 1

- **The classifier that produces the maximum margin (over the training data)**
- **I.e., the hyperplane that is furthest from the data**

$$\underset{\gamma_{Fun},W,b}{\text{Maximize}}\ \gamma_{Fun}$$

$$\text{subject to} \quad \mathbf{x_i w} + b \geq +\gamma_{Fun} \quad \forall\ y_i = +1,$$

$$\mathbf{x_i w} + b \leq -\gamma_{Fun} \quad \forall\ y_i = -1,$$

$$\text{or equivalently,}\ y_i(\mathbf{x_i w} + b) \geq \gamma_{Fun} \quad \forall\ i = 1,\ldots,L.$$

Hyperplane (W,b) can take any value and makes the margin γ_{fun} unbounded as W is unbounded E.g.,

$$y_i(\mathbf{x_i w} + b) - 1 = \gamma_{Fun,i}$$

$$y_i(10 * \mathbf{x_i w} + 10 * b) - 1 = \gamma_{Fun,i}$$

Geometric Margin

- **Functional margin has a scaling problem**

$$y_i\left(\mathbf{X_i w} + b\right) = \gamma_{Fun,i}$$

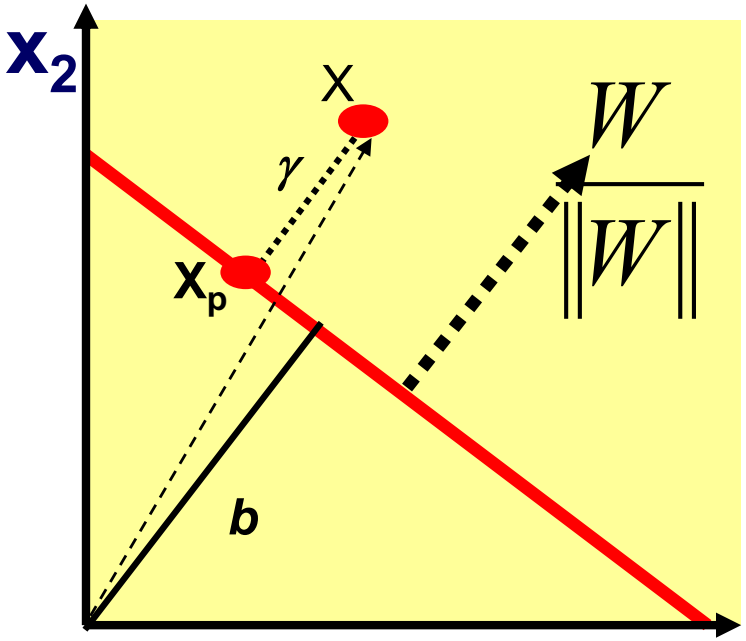
Objective function is infinite

$$y_i\left(10 * \mathbf{X_i w} + 10 * b\right) = \gamma_{Fun,i}$$

- **Avoid this by normalizing the hyperplane**

$$y_i\left(\frac{\mathbf{X_i w}}{\|W\|} + \frac{b}{\|W\|}\right) = \gamma_{Geo,i}$$

Derive Geometric Margin



$$X_P = X - \gamma \frac{W}{\|W\|}$$

$W^T X_P + b = 0$ (X_p is the projection X on H)

$$W^T \left(X - \gamma \frac{W}{\|W\|} \right) + b = 0$$

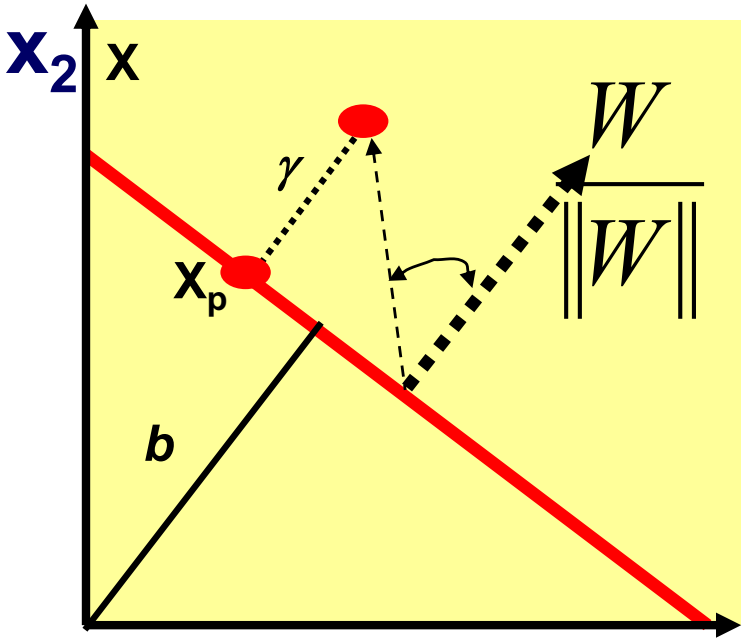
solve for γ

$$\gamma \frac{W^T W}{\|W\|} = W^T X + b$$

$$\gamma \|W\| = W^T X + b$$

$$\gamma = \frac{W^T X}{\|W\|} + \frac{b}{\|W\|}$$

Functional Margin vs Geometric



$$if\ (\|W\| = 1)\ \text{then}\ \gamma_{Fun} = \gamma_{Geo}$$

$$else\ \gamma_{geo} = \frac{\gamma_{fun}}{\|W\|}$$

Maximum Margin Classifier: take 2

- **The classifier that produces the maximum margin (over the training data)**
- **I.e., the hyperplane that is furthest from the data**
- **Previously:**

$$\underset{\gamma,W,b}{\text{Maximize}}\ \gamma_{Fun}$$

$$\text{subject to}\quad y_i(\mathbf{x_i w}_i + b) \geq \gamma_{Fun}\ \ \forall\ i = 1,\ldots,L.$$

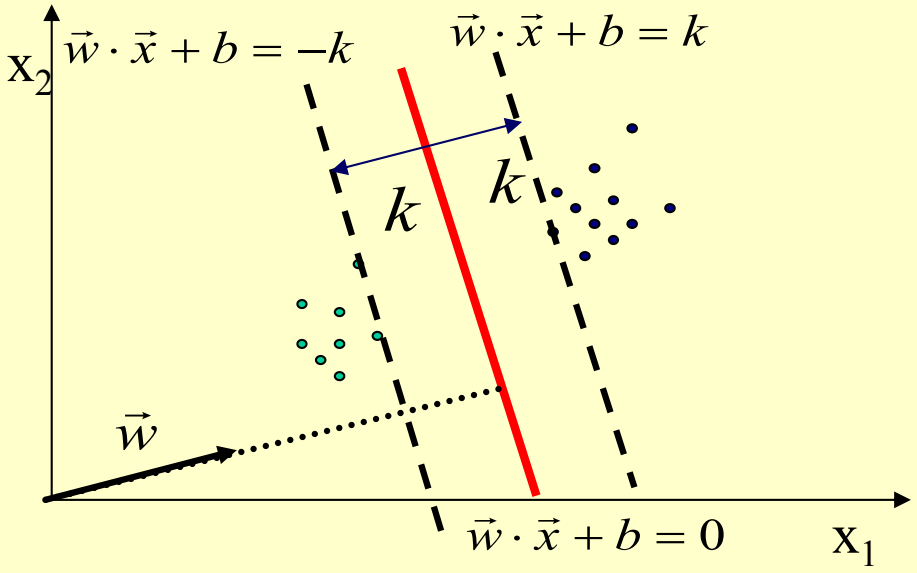
Versus

Better; constrain γ
But the objective
is non-concave

$$\underset{\gamma,W,b}{\text{Maximize}}\ \frac{\gamma_{fun}}{\|W\|}$$

$$\text{subject to}\quad y_i(\mathbf{x_i w}_i + b) \geq \gamma_{fun}\ \forall\ i = 1,\ldots,L$$

Setting Up the Optimization Problem



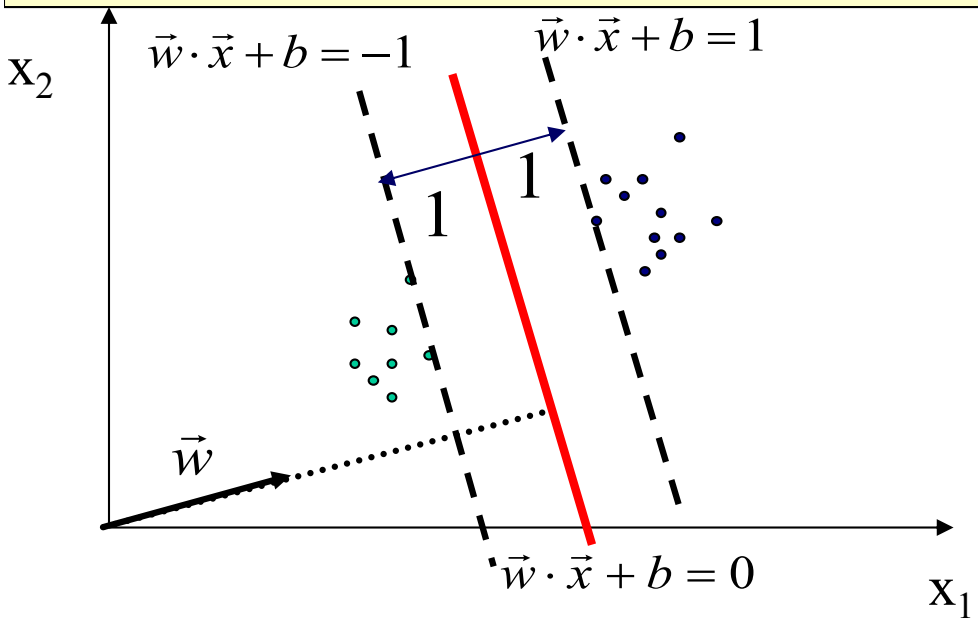
The width of the margin is: $\gamma = \frac{2k}{\|\mathbf{w}\|}$

$$\text{Max } \frac{2k}{\|\mathbf{w}\|^2} \text{ subject to constraints}$$

$$\mathbf{x}_i\mathbf{w} + b \geq +K \quad \forall\ y_i = +1,$$

$$\mathbf{x}_i\mathbf{w} + b \leq -K \quad \forall\ y_i = -1,$$

$$\text{or equivalently, } y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0 \ \forall\ i = 1,\ldots,L.$$



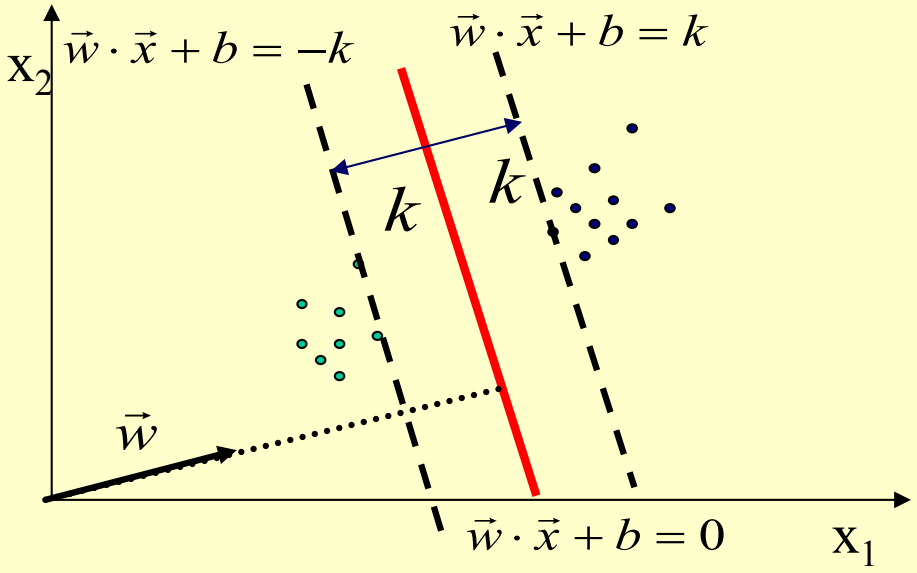
There is a scale and unit for data so that *k=1*. Then problem becomes:

$$\text{Max } \frac{2}{\|\mathbf{w}\|^2}$$

subject to constraints

$$y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0 \ \forall\ i = 1,\ldots,L.$$

Setting Up the Optimization Problem



The width of the margin is: $\gamma = \frac{2k}{\|\mathbf{w}\|}$

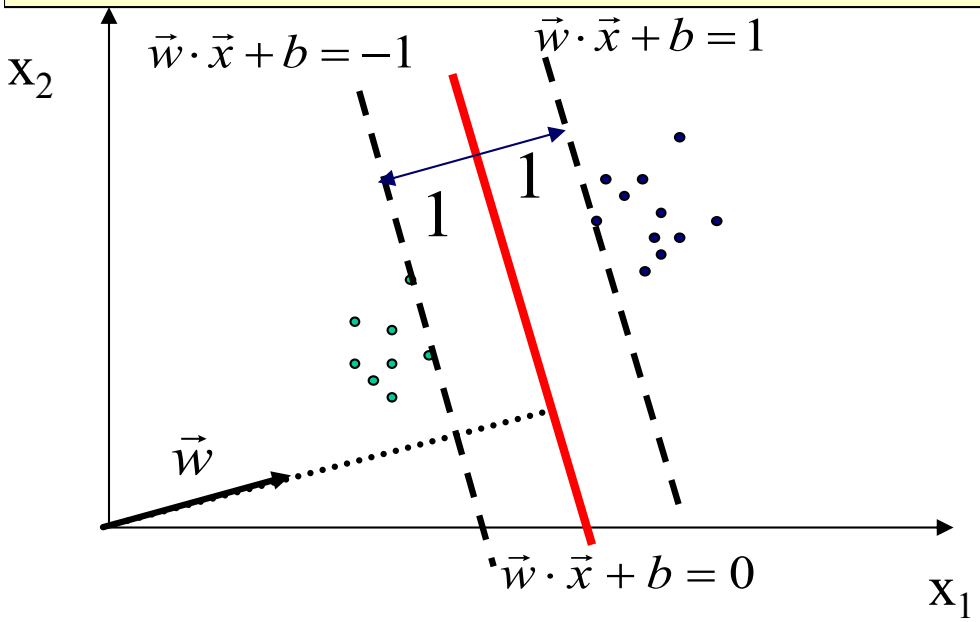
$$\text{Max } \frac{2k}{\|\mathbf{w}\|^2} \text{ subject to constraints}$$

k is unbounded

$$\mathbf{x}_i\mathbf{w} + b \geq +K \quad \forall\ y_i = +1,$$

$$\mathbf{x}_i\mathbf{w} + b \leq -K \quad \forall\ y_i = -1,$$

or equivalently, $y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0 \ \forall\ i = 1,\ldots,L.$



There is a scale and unit for data so that *k=1*. Then problem becomes:

$$\text{Maximize } \frac{2}{\|\mathbf{w}\|^2} \Leftrightarrow \text{Minimize } \frac{\|\mathbf{W}\|^2}{2}$$

subject to constraints

$$y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0 \ \forall\ i = 1,\ldots,L.$$

Maximum Margin Classifier: take 3

- **The classifier that produces the maximum margin (over the training data)**
- **I.e., the hyperplane that is furthest from the data**

Better!
But the objective is non-convex

$$\underset{\gamma,W,b}{\text{Maximize}} \frac{\gamma_{fun}}{\|W\|}$$

$$\text{subject to} \quad y_i(\mathbf{x_i w}_i + b) \geq \gamma_{fun} \;\forall\; i = 1,\ldots,L$$

Versus

Best!
Convex objective

$$\underset{\gamma,W,b}{\text{Maximize}} \frac{1}{\|W\|^2}$$

$$\text{subject to} \quad y_i(\mathbf{x_i w}_i + b) \geq 1 \;\forall\; i = 1,\ldots,L$$

$\gamma_{Fun} = 1$

Scaling constraint: γ_{Fun} of (W, b) WRT S is set to 1

Maximum Margin Classifier

- **Optimal maximum margin (over the training data)**
- **Can be solved using quadratic programming or gradient descent?**

$$\underset{\gamma,W,b}{\text{Maximize}} \frac{1}{\|W\|}$$

$$\text{subject to} \quad y_i(\mathbf{x_i w} + b) \geq 1 \ \forall \ i = 1,\ldots,L$$

$$\underset{\gamma,W,b}{\text{Minimize}} \frac{1}{2}\|W\|^2$$

$$\text{subject to} \quad y_i(\mathbf{x_i w} + b) \geq 1 \ \forall \ i = 1,\ldots,L$$

Maximizing the Margin

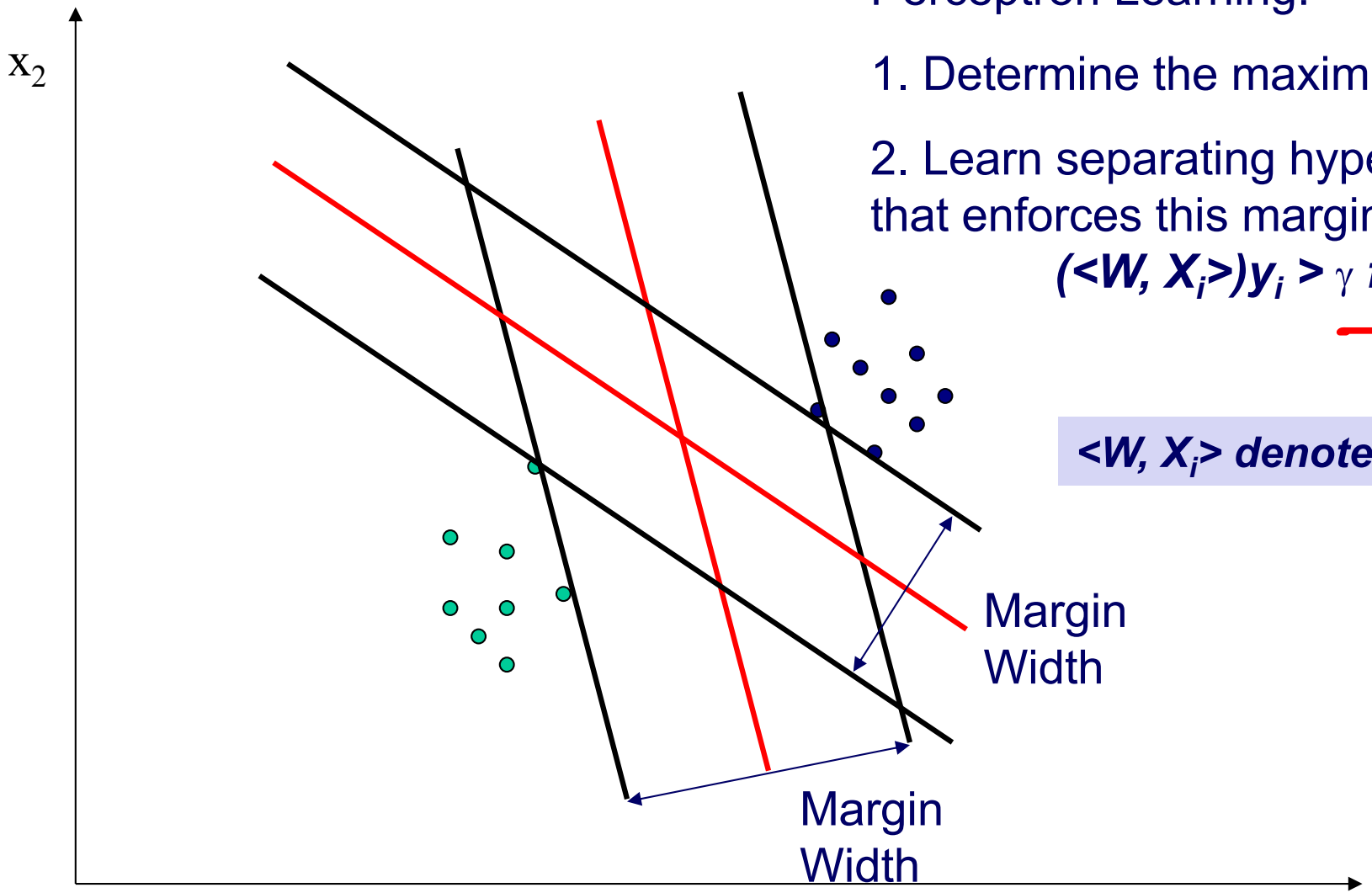
Chicken-Egg Conundrum for Perceptron Learning:

1. Determine the maximum margin γ

2. Learn separating hyperplane, W, that enforces this margin

$$(\langle W, X_i \rangle) y_i > \gamma \text{ for } I = 1, \ldots, L$$

$\langle W, X_i \rangle$ denotes dot product



Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual SVM algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

Primal – dual SVMs via Lagrangian

- **From constrained to unconstrained**

Equality to inequality contstraints

- **Equality constraints**

minimize $f(x,y) = x^2 + 2y$,
subject to: $3x + 2y + 1 = 0$

Constrained

$$g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$$

Unconstrained Lagrangian

- **Generalized Lagrangian for Inequality constraints**
  - $3x + 2y + 1 \geq 0$
  - where the $\alpha^{(i)}$ variables are called the Karush–Kuhn–Tucker (KKT) multipliers, and they must be greater or equal to zero.

Hard SVM

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)}\left(t^{(i)}\left(\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b\right) - 1\right)$$

$$\text{with} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

From constrained to unconstrained

minimize $f(x,y) = x^2 + 2y$,
subject to: $3x + 2y + 1 = 0$

Constrained

$$g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$$

Unconstrained Lagrangian

- **Suppose you want to find the values of x and y that minimize the function $f(x,y) = x^2 + 2y$, subject to an equality constraint: $3x + 2y + 1 = 0$.**
- **Using the Lagrange multipliers method, we start by defining a new function called the Lagrangian (or Lagrange function): $g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$.**
- **Each constraint (in this case just one) is subtracted from the original objective, multiplied by a new variable called a Lagrange multiplier.**

From constrained to unconstrained

Joseph-Louis Lagrange showed that if $(\hat{x}, \hat{y})$ is a solution to the constrained optimization problem, then there must exist an $\hat{\alpha}$ such that $(\hat{x}, \hat{y}, \hat{\alpha})$ is a *stationary point* of the Lagrangian (a stationary point is a point where all partial derivatives are equal to zero). In other words, we can compute the partial derivatives of $g(x, y, \alpha)$ with regards to x, y, and α; we can find the points where these derivatives are all equal to zero; and the solutions to the constrained optimization problem (if they exist) must be among these stationary points.

Find the stationary point of the Langrangian i.e., $\nabla = 0$

minimize $f(x,y) = x^2 + 2y$,
subject to: $3x + 2y + 1 = 0$

Constrained

$$g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$$

Unconstrained Lagrangian

In this example the partial derivatives are:

$$\begin{cases} \frac{\partial}{\partial x} g(x, y, \alpha) = 2x - 3\alpha \\ \frac{\partial}{\partial y} g(x, y, \alpha) = 2 - 2\alpha \\ \frac{\partial}{\partial \alpha} g(x, y, \alpha) = -3x - 2y - 1 \end{cases}$$

When all these partial derivatives are equal to 0, we find that $2\hat{x} - 3\hat{\alpha} = 2 - 2\hat{\alpha} = -3\hat{x} - 2\hat{y} - 1 = 0$, from which we can easily find that $\hat{x} = \frac{3}{2}$, $\hat{y} = -\frac{11}{4}$, and $\hat{\alpha} = 1$. This is the only stationary point, and as it respects the constraint, it must be the solution to the constrained optimization problem.

Equality to inequality contstraints

- **Equality constraints**

minimize $f(x,y) = x^2 + 2y$,
subject to: $3x + 2y + 1 = 0$

Constrained

$$g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$$

Unconstrained Lagrangian

- **Generalized Lagrangian for Inequality constraints**
  - $3x + 2y + 1 \geq 0$
  - where the $\alpha^{(i)}$ variables are called the Karush–Kuhn–Tucker (KKT) multipliers, and they must be greater or equal to zero.

Hard SVM

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)}\left(t^{(i)}\left(\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b\right) - 1\right)$$

$$\text{with} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

Inequality constraints: solution to Lagrangian must satisfy the KKT conditions

- **Just like with the Lagrange multipliers method, you can compute the partial derivatives and locate the stationary points.**
- **If there is a solution, it will necessarily be. among the stationary points $\hat{w}, \hat{b}, \hat{\alpha}$ that respect the KKT conditions:**
  - Respect the problem's constraints: $t^{(i)}\left((\hat{\mathbf{w}})^T \cdot \mathbf{x}^{(i)} + \hat{b}\right) \geq 1 \quad \text{for } i = 1, 2, \cdots, m,$
  - Verify $\hat{\alpha}^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m,$
  - Either $\hat{\alpha}^{(i)} = 0$ or the i$^{\text{th}}$ constraint must be an *active constraint*, meaning it must hold by equality: $t^{(i)}\left((\hat{\mathbf{w}})^T \cdot \mathbf{x}^{(i)} + \hat{b}\right) = 1$. This condition is called the *complementary slackness* condition. It implies that either $\hat{\alpha}^{(i)} = 0$ or the i$^{\text{th}}$ instance lies on the boundary (it is a support vector).

KKT SOC Necessary and sufficient

Note that the KKT conditions are necessary conditions for a stationary point to be a solution of the constrained optimization problem. Under some conditions, they are also sufficient conditions. Luckily

Learning the Maximal Margin Classier

- **To construct optimal hyperplane**
- **Minimize**
  - Subject to

$$\Phi(W) = \min\left(\frac{\|W\|^2}{2}\right)$$

$$y_i\,((x_i\ W_i) + b) \geq 1, i = 1,\ldots,l$$

- **Constrained Optimization problem → Lagrangian**
  - Lagrange multipliers method: transform a constrained optimization objective into an unconstrained one, by moving the constraints into the objective function

$$L(w,b,\alpha) = \tfrac{1}{2}\|W\|^2 - \sum_{i=1}^{l}\alpha_i(y_i\cdot((x_i\ W\ + b) - 1)$$

Add eqn for each constraint

$$L(w,b,\alpha) = \tfrac{1}{2}\|W\|^2 - \sum_{i=1}^{l}\alpha_i(y_i\cdot(x_i\ W\ ) + b)) + \sum_{i=1}^{l}\alpha_i$$

Refactor α

Lagrange Cookbook

- **To construct optimal hyperplane**
- **Minimize** $\tau(w) = \frac{1}{2}\|w\|^2$
  - Subject to $y_i\,(\,X_i\;\;W\;\;+b) \geq 1, i = 1,\ldots,l$
- **Corresponding unConstrained Optimization problem, the Lagrangian**

$$L(w,b,\alpha) = \tfrac{1}{2}\|W\|^2 - \sum_{i=1}^{l}\alpha_i(y_i\cdot(\,X_i\;\;W\;\;+b)-1)$$

$$\frac{\partial}{\partial b}L(w,b,\alpha) = 0 \quad \frac{\partial}{\partial w}L(w,b,\alpha) = 0$$

- **Primal variables vanish**

$$\frac{\partial L(\mathbf{w},b,\alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}\alpha_i y_i \mathbf{x}_i = 0$$

$$\sum_{i=1}^{l} a_i y_i = 0 \qquad w = \sum_{i=1}^{l}\alpha_i y_i x_i$$

$$\frac{\partial L(\mathbf{w},b,\alpha)}{\partial b} = -\sum_{i=1}^{l}\alpha_i y_i = 0$$

Lagrange Cookbook

- Primal variables vanish

$$\sum_{i=1}^{l} a_i y_i = 0 \qquad w = \sum_{i=1}^{l} \alpha_i y_i x_i$$ Eq. (1)

- All examples are classified correctly
  - Lagrange equivalent optimization

$$\alpha_i \cdot [y_i((X_i \ W) + b) - 1] = 0, \quad i = 1,\ldots,l$$

$$\Phi(w) = Min\left(\tfrac{1}{2} W^T W\right)$$

Primal $$L(w,b,\alpha) = Min\left(\tfrac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i [y_i(X_i \ W + b) - 1]\right)$$ Using Eq. (1)

Dual $$Q(\alpha) = Max\left(\sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j X_i^T X_j\right)$$

Hard SVM Learning

QP

objective function

$$\max_{\alpha} L(\alpha) = \max_{\alpha} \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{j=1}^{L} y_i y_j \langle X_i, X_j \rangle \alpha_i \alpha_j,$$

Shorthand

$$\max_{\alpha} L(\alpha) = \max_{\alpha} I^T \alpha - \frac{1}{2} \alpha^T Q \alpha$$

constraints

$$0 \le \alpha_i, \quad \forall i \in [1,\ldots,L]$$

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0. \quad \text{Due to } b \left(\frac{\partial \mathrm{L}}{\partial \mathrm{b}} = \sum_{i=1}^{\ell} y_i \alpha_i \right)$$

KKT

$$\alpha_i = 0 \Rightarrow y_i f(\vec{x}_i) > 1,$$

Correctly classified training example but non-SV

$$\alpha_i > 0 \Rightarrow y_i f(\vec{x}_i) = 1$$

Correct and Support Vector

The result of this optimization process is a vector of co-efficients, $\alpha^T = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ for which $W(\alpha)$ is maximum and satisfies the KKT conditions.

Hard SVM Dual as a QP

QP

objective function

$$\max_{\alpha} L(\alpha) = \max_{\alpha} \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{j=1}^{L} y_i y_j \left\langle X_i, X_j \right\rangle \alpha_i \alpha_j,$$

constraints

$$0 \le \alpha_i, \quad \forall i \in [1,\ldots,L]$$

Positive +equality

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0. \quad \text{Due to } b \left(\frac{\partial \mathrm{L}}{\partial \mathrm{b}} = \sum_{i=1}^{\ell} y_i \alpha_i \right)$$

Shorthand

$$\max_{\alpha} L(\alpha) = \max_{\alpha} I^T \alpha - \frac{1}{2} \alpha^T Q \, \alpha$$

KKT

$$\alpha_i = 0 \Rightarrow y_i f(\vec{x}_i) > 1,$$

Correctly classified training example but non-SV

$$\alpha_i > 0 \Rightarrow y_i f(\vec{x}_i) = 1$$

Correct and Support Vector

The result of this optimization process is a vector of co-efficients, $\alpha^T = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ for which $W(\alpha)$ is maximum and satisfies the KKT conditions.

Hard SVM Dual Vs Primal

$$L_p(\mathbf{w}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle - \sum \alpha_i [y_i (\langle \mathbf{w}, x_i \rangle + b) - 1] \qquad (4)$$

Q

$$\alpha \geq 0$$

constraints

Shorthand

$$\max_{\alpha} L(\alpha) = \max_{\alpha} I^T \alpha - \frac{1}{2} \alpha^T Q \, \alpha$$

$$0 \leq \alpha_i, \quad \forall i \in [1, \ldots, L]$$

Positive +equality

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0. \quad \text{Due to } b \left(\frac{\partial \mathrm{L}}{\partial \mathrm{b}} = \sum_{i=1}^{\ell} y_i \alpha_i \right)$$

K

$$\partial' L / \partial W = 0$$

$$\partial' L / \partial b = 0$$

$$y_i \cdot (\langle X_i, W \rangle + b) - 1 \geq 0$$

$$\alpha_i \geq 0$$

$$\alpha_i (y_i \cdot (\langle X_i, W \rangle + b) - 1) = 0 \qquad \forall \, \mathrm{i} \in [1, \ldots, \mathrm{n}] \text{ KKT Complementarity cond}^{\mathrm{n.}}$$

From Primal to Dual Representation

$$Class(X) = f(X) = \text{sgn}(< W, X > + b)$$

$$\text{Since } W = \sum_{i=1}^{|Train|} \alpha_i y_i X_i$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i \langle X_i, X \rangle + b \right)$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i \langle \Phi(X_i), \Phi(X) \rangle + b \right)$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i K(X_i, X) + b \right)$$

- **Dual versions tend to be easier to solve**

Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual perceptron algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

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

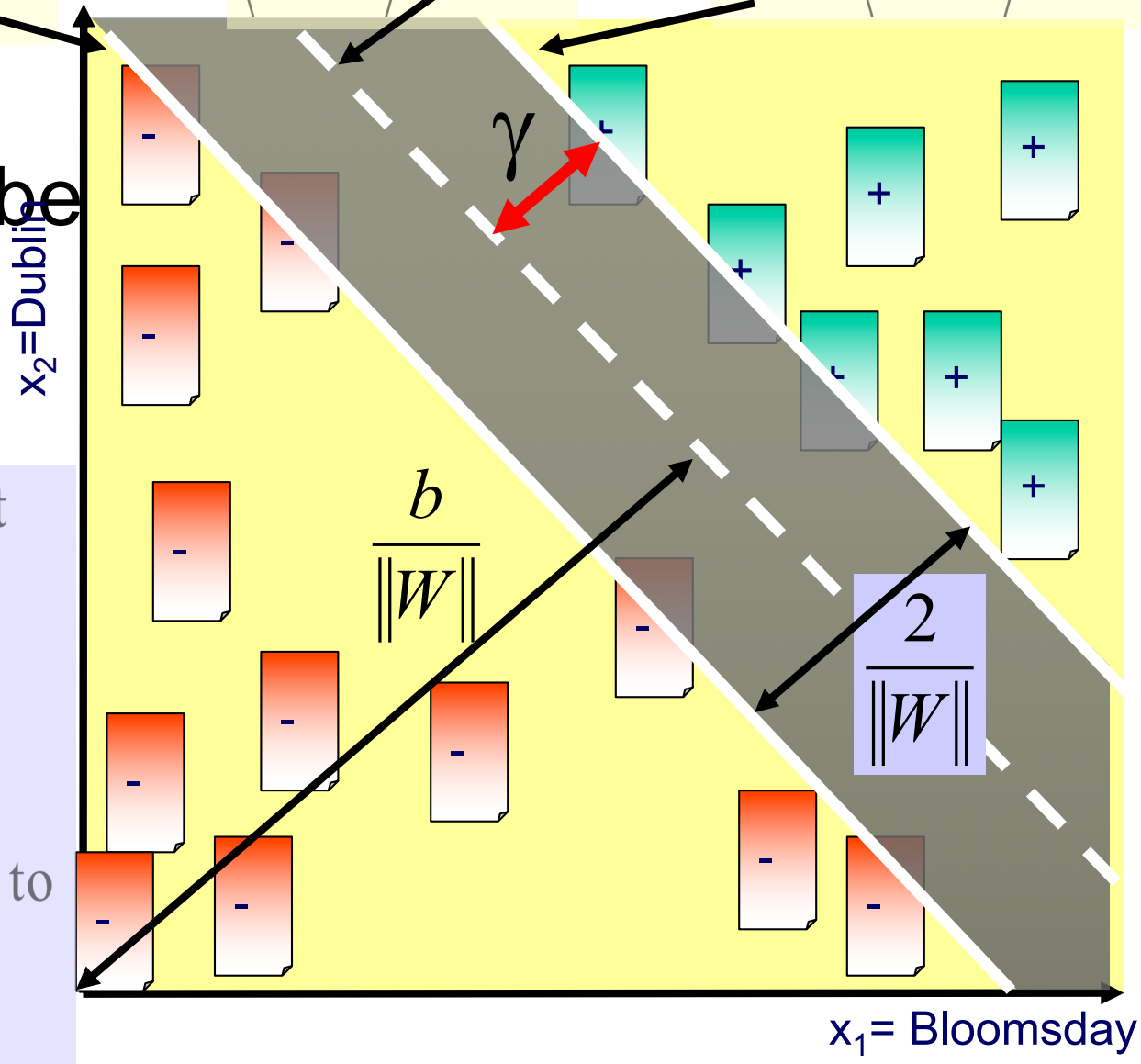
Find H_1 and H_2 such that

$Dist(H_1, H_2)$ is MAX

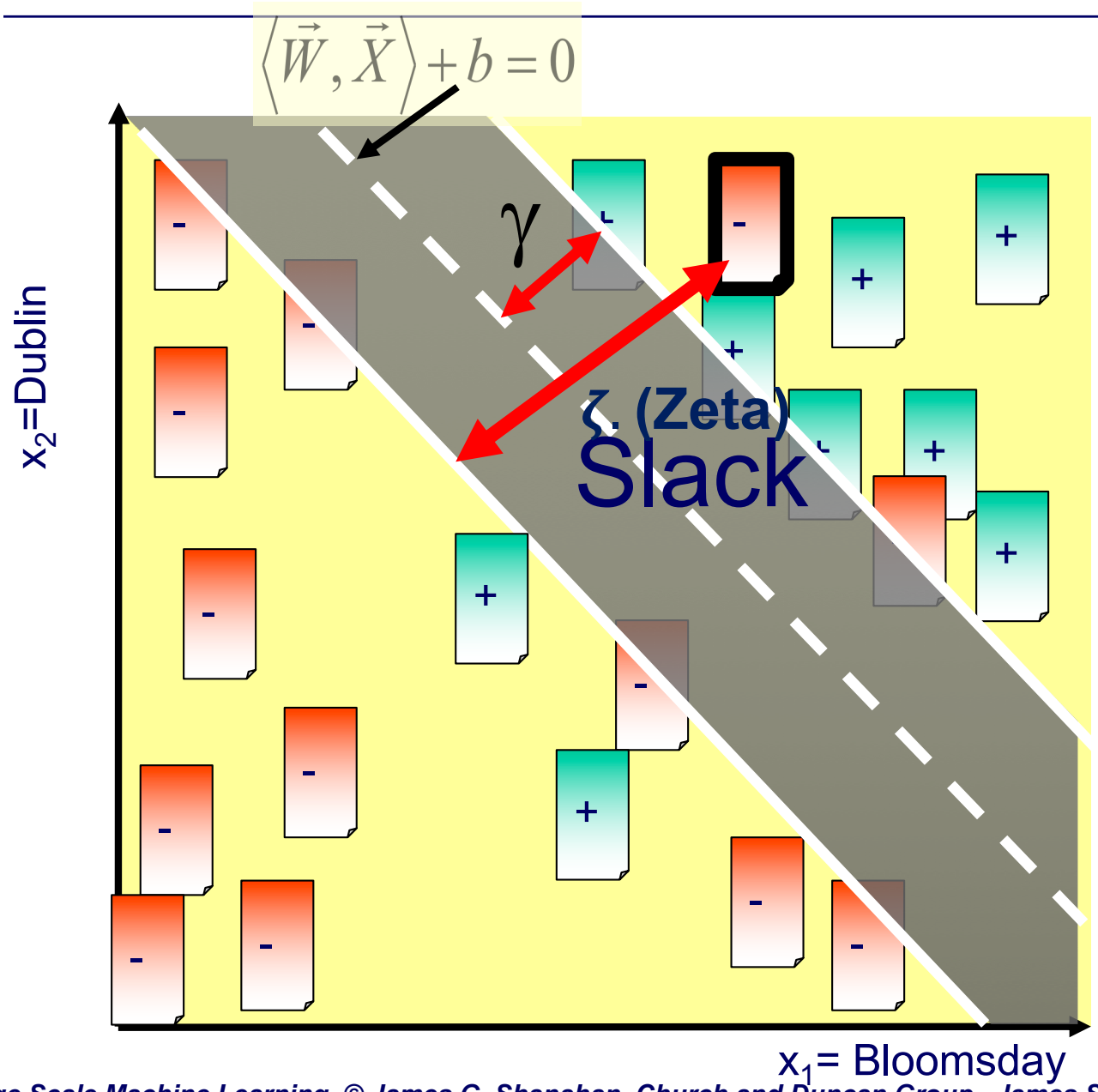
Or

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)

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 $\mathbf{b}$.

Dual Soft SVM Learning Problem

objective function

$$\max_{\alpha} W(\alpha) = \max_{\alpha} \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{j=1}^{\ell} y_i y_j k(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j,$$

constraints

QP

$$0 \le \alpha_i \le C, \quad \forall i,$$

$$\sum_{i=1}^{\ell} y_i a_i = 0.$$

KKT

$$\alpha_i = 0 \Rightarrow y_i f(\vec{x}_i) \ge 1,$$

Correctly classified training example but non-SV

$$0 < \alpha_i < C \Rightarrow y_i f(\vec{x}_i) = 1,$$

Correct and Support Vector

$$\alpha_i = C \Rightarrow y_i f(\vec{x}_i) \le 1.$$

Incorrectly classified training example

Where KKT = Karush-Kuhn-Tucker

Dual Soft SVM

Soft Margin SVM (Primal)

$$\min(W,b) = 0.5 \times \|w\|^2 + C\sum_{i=1}^{L}\xi_i$$

$$\text{subject to: } y_i\left(\langle W, X_i\rangle + b\right) + \xi_i \geq 1 \quad \forall\, i = 1,\ldots,L$$

$$\text{and } \xi_i \geq 0 \quad \forall\, i = 1,\ldots,L$$

Soft Margin SVM (Dual)

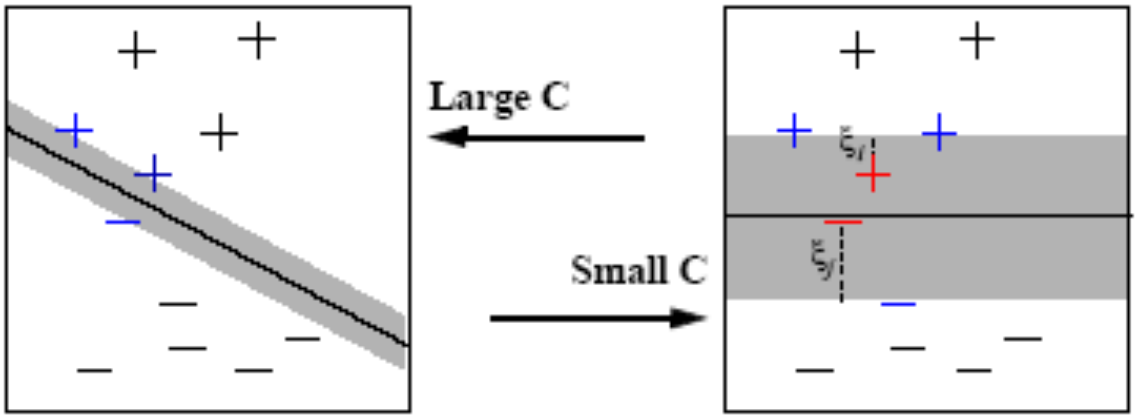
$$\max_{\alpha} W(\alpha)\sum_{i=1}^{L}\alpha_i - \frac{1}{2}\sum_{i=1}^{L}\sum_{j=1}^{L} y_i y_j \langle X_i, X_j\rangle \alpha_i \alpha_j,$$

$$0 \leq \alpha_i \leq C, \quad \forall i = 1,\ldots,L$$

$$\sum_{i=1}^{L} y_i a_i = 0.$$

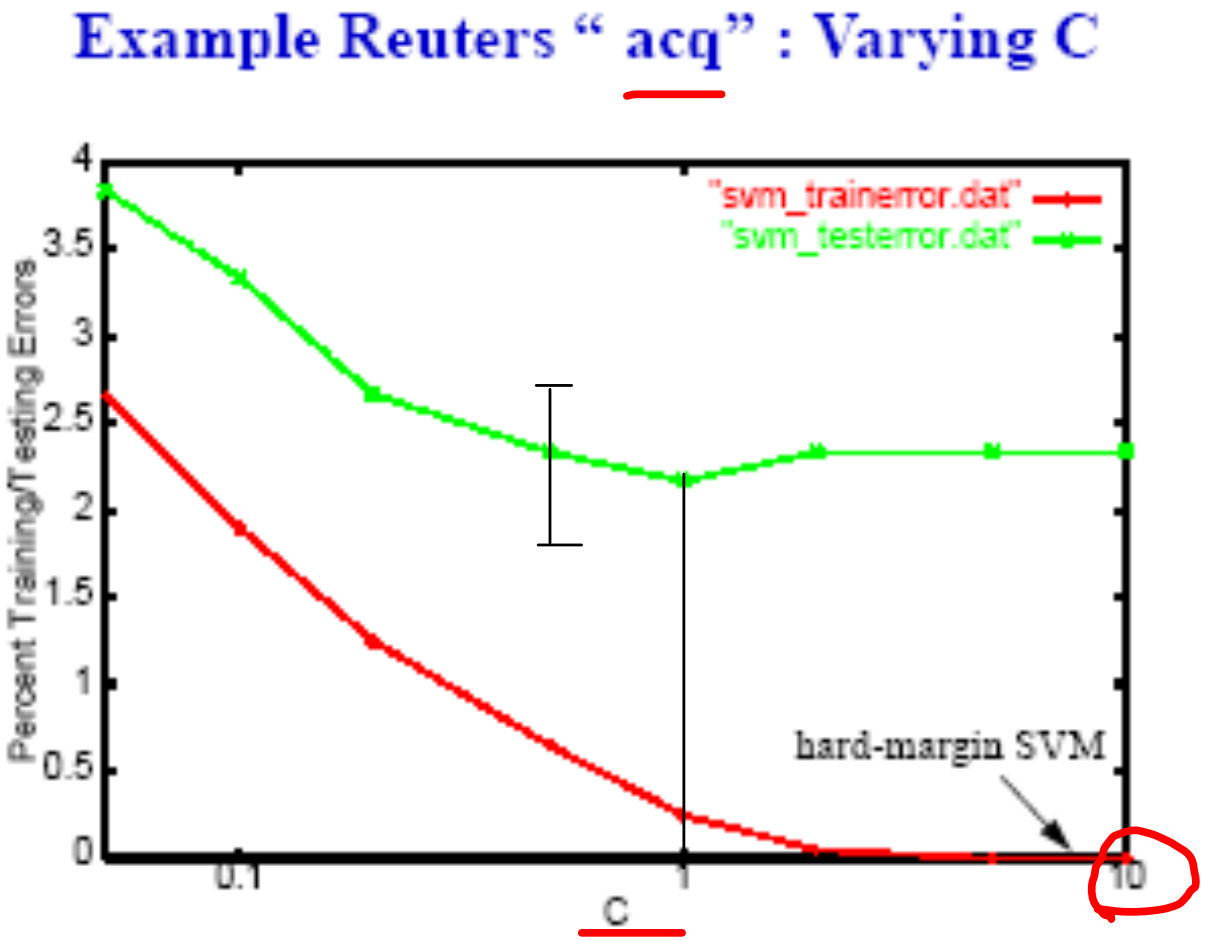
Upper bounds for α

- **Large C => Hard Margin (allow very few errors)**
- **Small C => allow a lot of slack and therefore large margin**



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

Controlling Soft Margin Separation



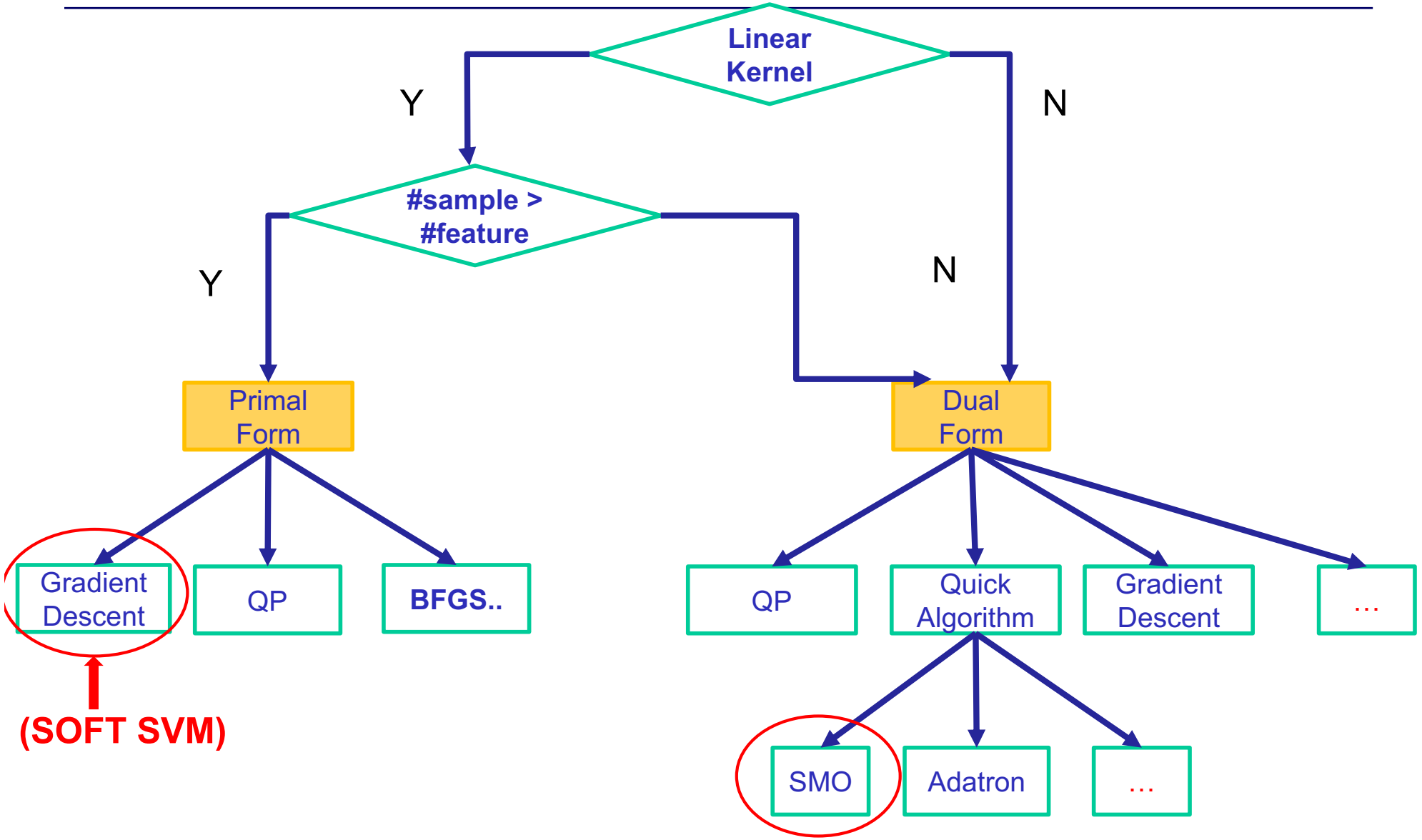
Observation: Typically no local optima, but not necessarily...

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

Learning as Optimization

- **The perceptron algorithm is basically a gradient descent procedure for solving simultaneous linear inequalities (where we specify the required margin)**
- **Represent hyperplane learning as a Lagrange optimization task (learn the maximum margin)**
  - Exploit linear programming/quadratic techniques
    - procedures for maximizing or minimizing linear/quadratic functions subject linear equality or inequality constraints
  - Standard and Slack formulations
  - Lagrange optimization algorithms
    - Simplex Algorithm, ellipsoid algorithm, interior point algorithms
    - Non-LP/QP-based Closed form (SMO, SMOK1, SMOK2)

SVM Learning Algorithm Taxonomy



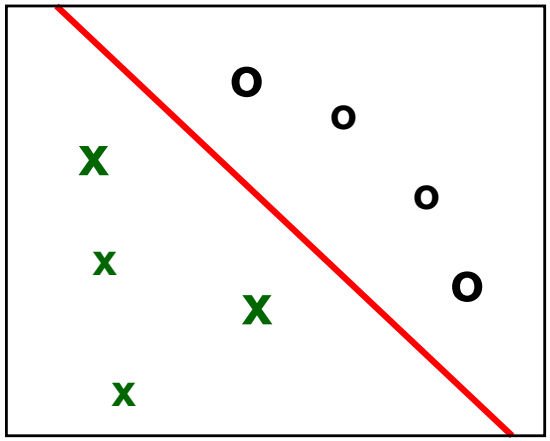
- **Solve Nonlinear problems via Explore kernels next**

Outline

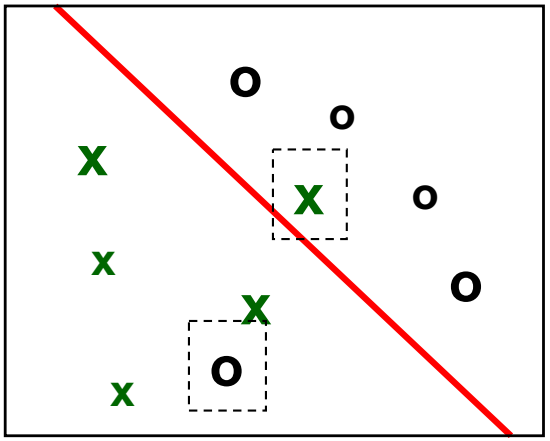
1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual perceptron algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

Non-Linearly Separable Data

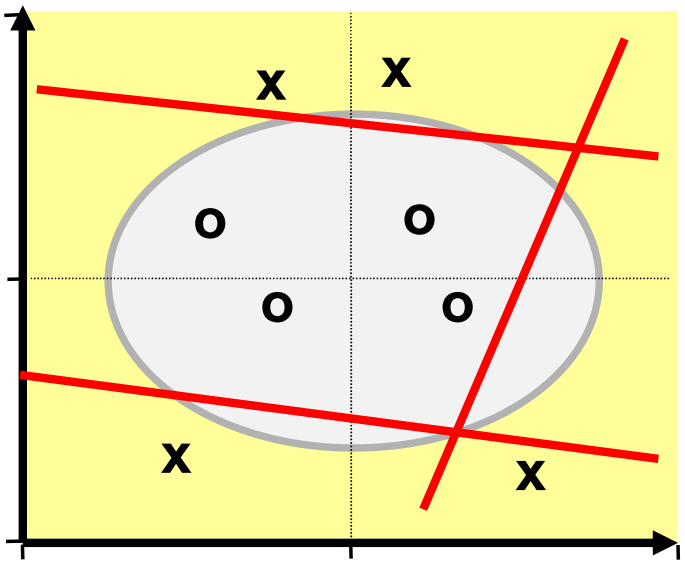
Linearly Separable



Noisy Linearly Separable



NOT Linearly Separable



linearly separable data:

Use Hard/Soft SVMs;

Noisy linearly separable data:

Use Soft SVMs;

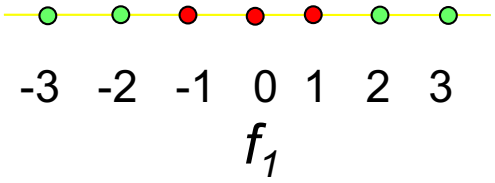
Text is generally linearly separable but is sometimes noisy

Non-Linearly Separable Data: Use kernels

Quadratic Machine

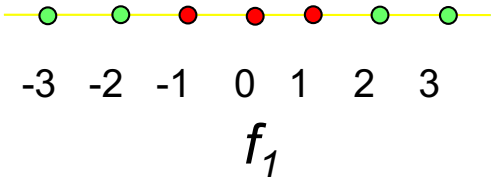
- **All quadratic surfaces (2nd order)**
  - ellipsoid
  - parabola
  - etc.
- **That significantly increases the number of problems that can be solved, but still many problem which are not quadrically separable**
- **Could go to 3rd and higher order features, but number of possible features grows exponentially**
- **Multi-layer neural networks will allow us to discover high-order features automatically from the input space**

Simple Quadric Example



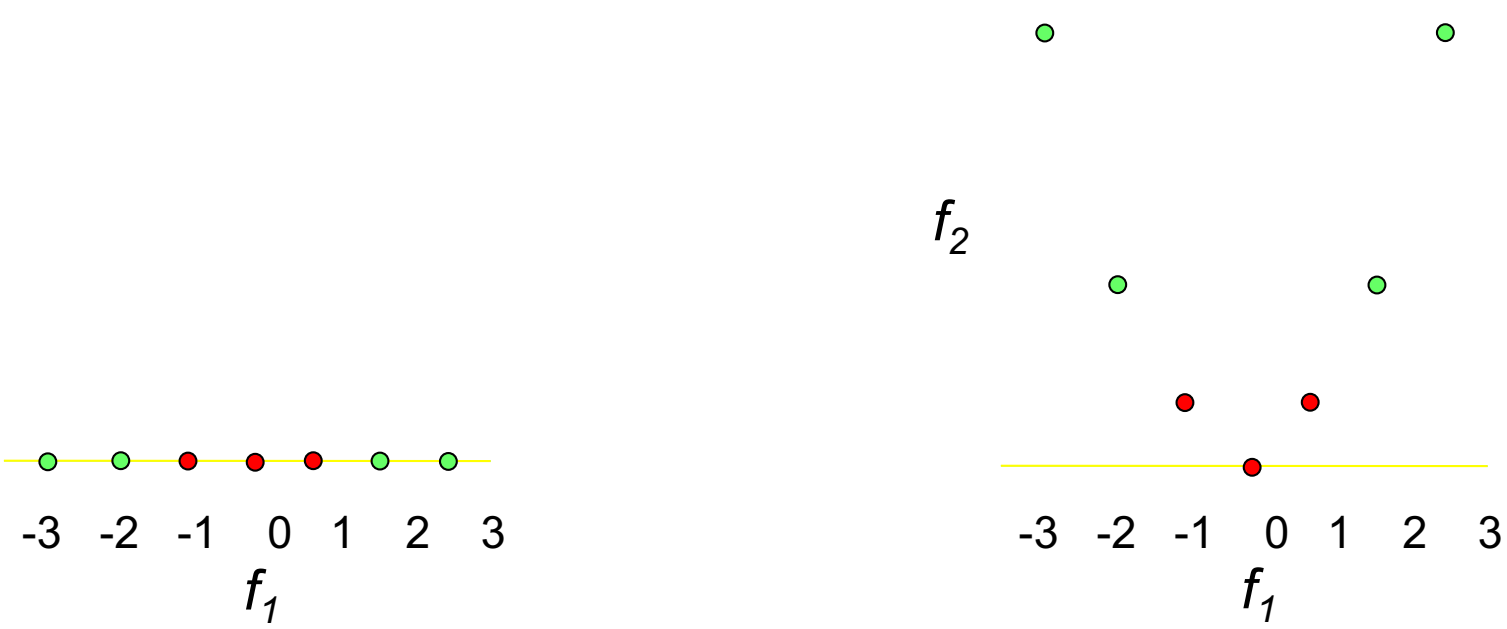
- **Perceptron with just feature f_1 cannot separate the data**
- **Could we add a transformed feature to our perceptron?**

Simple Quadric Example



- Perceptron with just feature f_1 cannot separate the data
- Could we add a transformed feature to our perceptron?
- $f_2 = f_1^2$

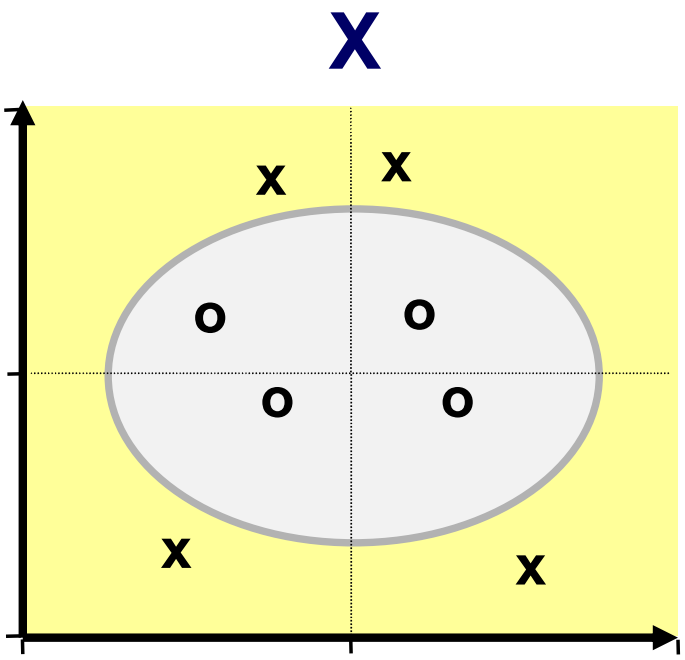
Simple Quadric Example

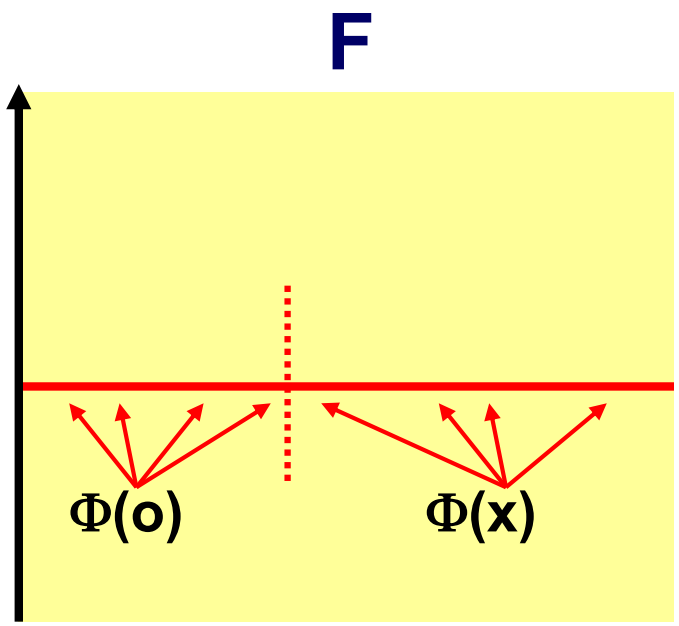


- Perceptron with just feature f_1 cannot separate the data
- Could we add another feature to our perceptron $f_2 = f_1^2$
- Note could also think of this as just using feature f_1 but now allowing a quadric surface to separate the data

Feature Transformations

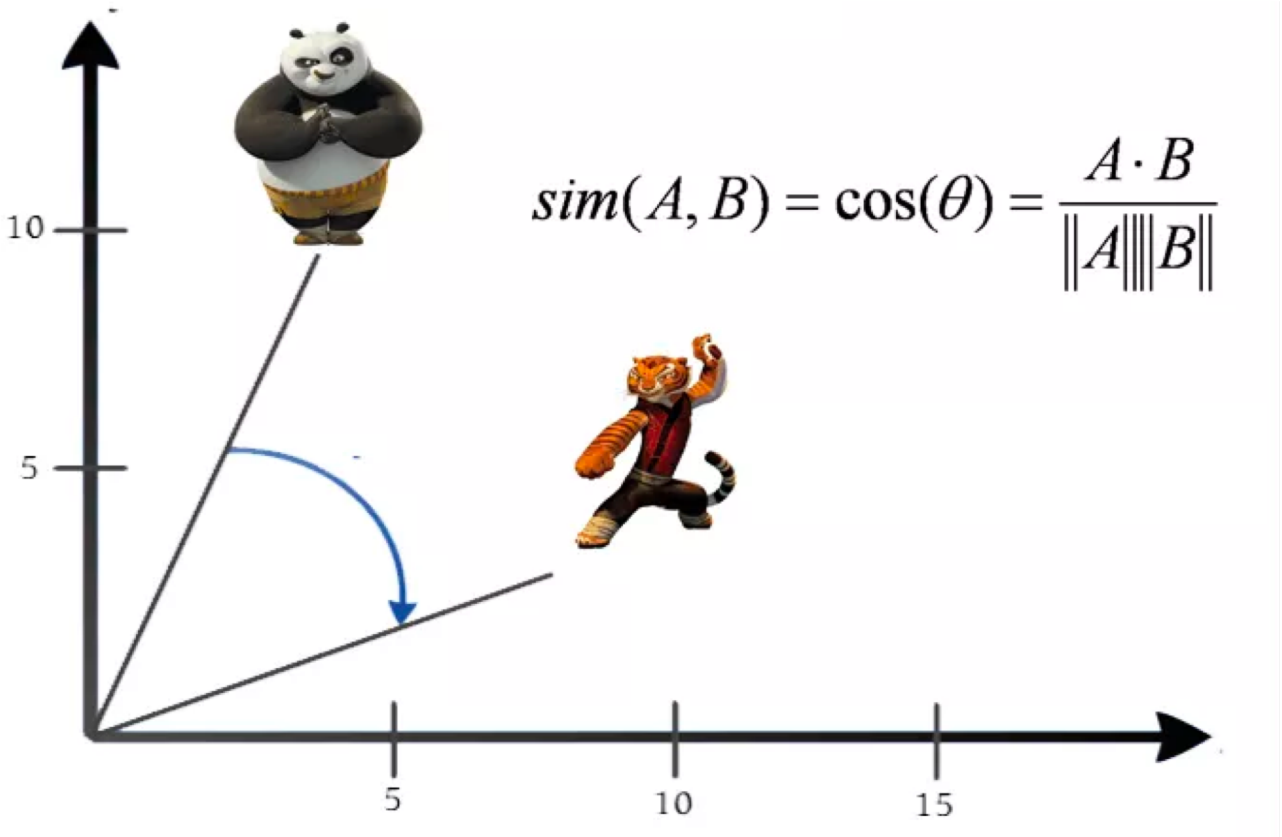
$$\Phi(x_1, x_2) = x_1^2 + x_2^2 - 1$$





- Binary Classification Problem
- Classify points inside ellipse as legal and outside as illegal

Dot product as a similarity metric



$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

$$\langle X, W \rangle = X^T \cdot W = \|\mathrm{X}\| \times \|\mathrm{W}\| \times \cos(\theta)$$

Kernel and Kernel Trick

- **A kernel is a similarity measure between two examples**

$$\langle X, W \rangle = X^T \cdot W = \|X\| \times \|W\| \times \cos(\theta)$$

- **Linear Kernel: $\langle X_i, X_j \rangle = K(X_i, X_j)$**

- **Find a mapping ϕ such that, in the new space, problem solving is easier (e.g. linear separation)**
- **If every datapoint is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the inner product becomes: $\langle x_i, x_j \rangle = K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$**
- **A *kernel function* is some function that corresponds to an inner product into some feature space.**
- **But the mapping is left implicit (Kernel Trick)**

From Primal to Dual Representation

$$Class(X) = f(X) = \text{sgn}(< W, X > + b)$$

$$\text{Since } W = \sum_{i=1}^{|Train|} \alpha_i y_i X_i$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i \langle X_i, X \rangle + b \right)$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i \langle \Phi(X_i), \Phi(X) \rangle + b \right)$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i K(X_i, X) + b \right)$$

Non-linear SVMs Mathematically

- **Dual problem formulation:**

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum\sum \alpha_i \alpha_j y_i y_j K(\mathbf{X_i}, \mathbf{X_j})$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

- **The learnt model is:**

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x_i}, \mathbf{x_j}) + b$$

- **Optimization techniques for finding α_i's remain the same!**

Kernels: Intuitive Idea

- Find a mapping ϕ such that, in the new space, problem solving is easier (e.g. linear)
- The *kernel* represents the similarity between two objects (documents, terms, …), defined as the dot-product in this new vector space
- But the mapping is left implicit
- By using an appropriate nonlinear kernel function, many problems which are linearly inseparable but nonlinearly separable in the original pattern space become linearly separable in the higher dimensional feature space.
- Easy generalization of a lot of dot-product (or distance) based pattern recognition algorithms

Polynomial Kernels – Kernel Trick

- **A polynomial kernel returns a similarity value (dot product) between the images of two object vectors**

$$\text{Given} \quad X = (x_1, x_2), Z = (z_1, z_2)$$

$$K(X,Z) = \left(\langle X, Z \rangle\right)^2 = \left(x_1 z_1 + x_2 z_2\right)^2$$

$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2$$ **Refactor individual vectors**

$$= \left\langle \left(x_1^2, x_2^2, \sqrt{2} x_1 x_2\right), \left(z_1^2, z_2^2, \sqrt{2} z_1 z_2\right) \right\rangle$$ **Monomials of deg=2**

$$= \langle \Phi(X), \Phi(Z) \rangle \quad \text{where } \Phi(\mathrm{X}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

Example

Given $X = (2,3), Z = (2,3)$

$$K(X,Z) = \langle X,Z \rangle^2 = \left(\langle [2,3],[2,3] \rangle\right)^2 = 13^2 = 169$$

Alternatively if tranform into our new feature space F using

Is $\langle \Phi(X), \Phi(X) \rangle == \langle X,Z \rangle^2$????

Example

$$\text{Given} \quad X = (2,3), Z = (2,3)$$

$$K(X,Z) = \langle X, Z \rangle^2 = \left(\langle [2,3],[2,3] \rangle \right)^2 = 13^2 = 169$$

$$\text{Alternatively if tranform into our new feature space F using}$$

$$\Phi(\mathrm{X}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \ \text{THEN}$$

$$\left\langle \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right), \left(z_1^2, z_2^2, \sqrt{2}z_1z_2\right) \right\rangle$$

$$\left\langle [4,9,\sqrt{2}\times 6],[4,9,\sqrt{2}\times 6] \right\rangle = 16 + 81 + 72 = 169$$

Kernel similarity function implicitly uses all monomials of degree 2

The "Kernel Trick"

- **The linear classifier relies on inner product between vectors $K(x_i,x_j)=x_i^T x_j$ or $< x_i^T x_j >$**
- **If every datapoint is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the inner product becomes: $K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j)$**
- **A *kernel function* is some function that corresponds to an inner product into some feature space.**

- **Example:**
  2-dimensional vectors $x=[x_1 \ x_2]$; let $K(x_i,x_j)=(1 + x_i^T x_j)^2$,
  Need to show that $K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j)$:
  $K(x_i,x_j)=(1 + x_i^T x_j)^2$

$(1 + x_i^T x_j)^2 = 1+ x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$

$K(\varphi(x_i), \varphi(x_j)) = [1 \ x_{i1}^2 \ \sqrt{2}\, x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T \ [1 \ x_{j1}^2 \ \sqrt{2}\, x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}]$

where $\varphi(x) = [1 \ x_1^2 \ \sqrt{2}\, x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2]$

The Kernel Gram Matrix

- **With KM-based learning, the sole information used from the training data set is the Kernel Gram Matrix**

$$K_{training} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_m) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_m) \\ \dots & \dots & \dots & \dots \\ k(\mathbf{x}_m, \mathbf{x}_1) & k(\mathbf{x}_m, \mathbf{x}_2) & \dots & k(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

- **If the kernel is valid, K is Semi positive definite and symmetric.**

Kernels

- The function $K(a, b) = (a^T . b)^2$ is called a 2nd-degree polynomial kernel.
- In Machine Learning, a kernel is a function capable of computing the dot product $\phi(a)^T . \phi(b)$ based only on the original vectors a and b, without having to compute (or even to know about) the transformation ϕ.

$$\text{Linear:} \quad K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \cdot \mathbf{b}$$

$$\text{Polynomial:} \quad K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^T \cdot \mathbf{b} + r\right)^d$$

$$\text{Gaussian RBF:} \quad K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right)$$

$$\text{Sigmoid:} \quad K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T \cdot \mathbf{b} + r\right)$$

Good kernel functions observe Mercer's Theorem

Mercer's Theorem

According to *Mercer's theorem*, if a function $K(\mathbf{a}, \mathbf{b})$ respects a few mathematical conditions called *Mercer's conditions* (K must be continuous, symmetric in its arguments so $K(\mathbf{a}, \mathbf{b}) = K(\mathbf{b}, \mathbf{a})$, etc.), then there exists a function ϕ that maps $\mathbf{a}$ and $\mathbf{b}$ into another space (possibly with much higher dimensions) such that $K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \cdot \phi(\mathbf{b})$. So you can use K as a kernel since you know ϕ exists, even if you don't know what ϕ is. In the case of the Gaussian RBF kernel, it can be shown that ϕ actually maps each training instance to an infinite-dimensional space, so it's a good thing you don't need to actually perform the mapping!

Note that some frequently used kernels (such as the Sigmoid kernel) don't respect all of Mercer's conditions, yet they generally work well in practice.

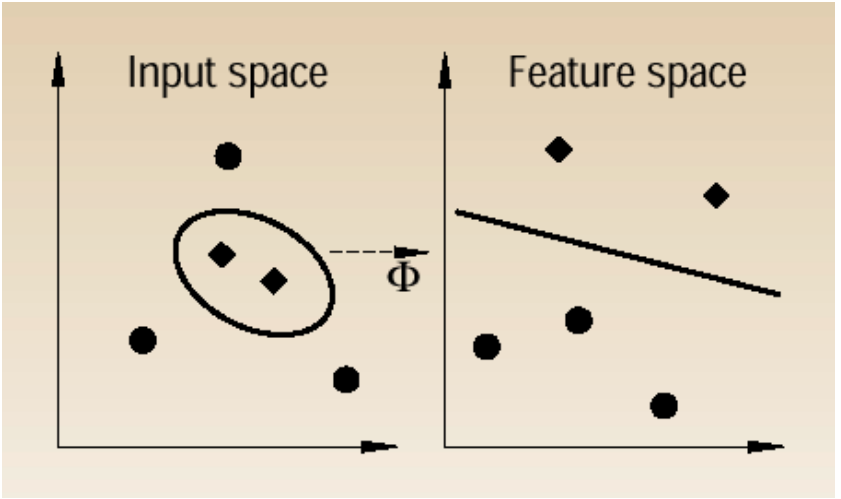
The right kernel for the right task

- **Assume a categorization task: the ideal Kernel matrix is**
  - $k(\mathbf{x}_i,\mathbf{x}_j)$=+1 if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same class
  - $k(\mathbf{x}_i,\mathbf{x}_j)$=-1 if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to different classes
  - → concept of target alignment (adapt the kernel to the labels), where alignment is the similarity between the current gram matrix and the ideal one [“two clusters” kernel]
- **A bad kernel is the diagonal kernel**
  - $k(\mathbf{x}_i,\mathbf{x}_j)$=+1 if $\mathbf{x}_i = \mathbf{x}_j$
  - $k(\mathbf{x}_i,\mathbf{x}_j)$=0 elsewhere
  - All points are orthogonal : no clusters, no more structure

Feature Spaces and Kernels

- Feature Space
  - Input space mapped to some other dot product space F (feature space) via a nonlinear mapping

$$\phi = R^N \rightarrow F$$



- Kernel Trick
  - Evaluation of decision function require dot product but never the mapped pattern in explicit form $\phi(x)$
  - Complex implicit feature expansion can accomplished cheaply

$$k(x, y) = (\phi(x) \cdot \phi(y))$$

Kernel trick for a 2nd-degree polynomial mapping

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}\, x_1 x_2 \\ {x_2}^2 \end{pmatrix}$$

Polynomial features: 2nd-degree polynomial
From 2 to 3 features

dot product of the transformed vectors

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^T \cdot \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b}\right)^2$$

dot product

The dot product of the transformed vectors is equal to the square of the dot product of the original vectors: $\phi(\mathbf{a})^T\, \phi(\mathbf{b}) = (\mathbf{a}^T \cdot \mathbf{b})^2$

From Primal to Dual Representation

$$Class(X) = f(X) = \text{sgn}(< W, X > + b)$$

$$\text{Since } W = \sum_{i=1}^{|Train|} \alpha_i y_i X_i$$

SVM Model:
Must store SVs

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i \langle X_i, X \rangle + b \right)$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i \langle \Phi(X_i), \Phi(X) \rangle + b \right)$$

$$Class(X) = \text{sgn}\left(\sum_{i=1}^{|Train|} \alpha_i y_i K(X_i, X) + b \right)$$

Predictions

$$h_{\widehat{\mathbf{W}},\hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) = \widehat{\mathbf{W}}^T \cdot \phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi\left(\mathbf{x}^{(i)}\right)\right)^T \cdot \phi\left(\mathbf{x}^{(n)}\right) + \hat{b}$$

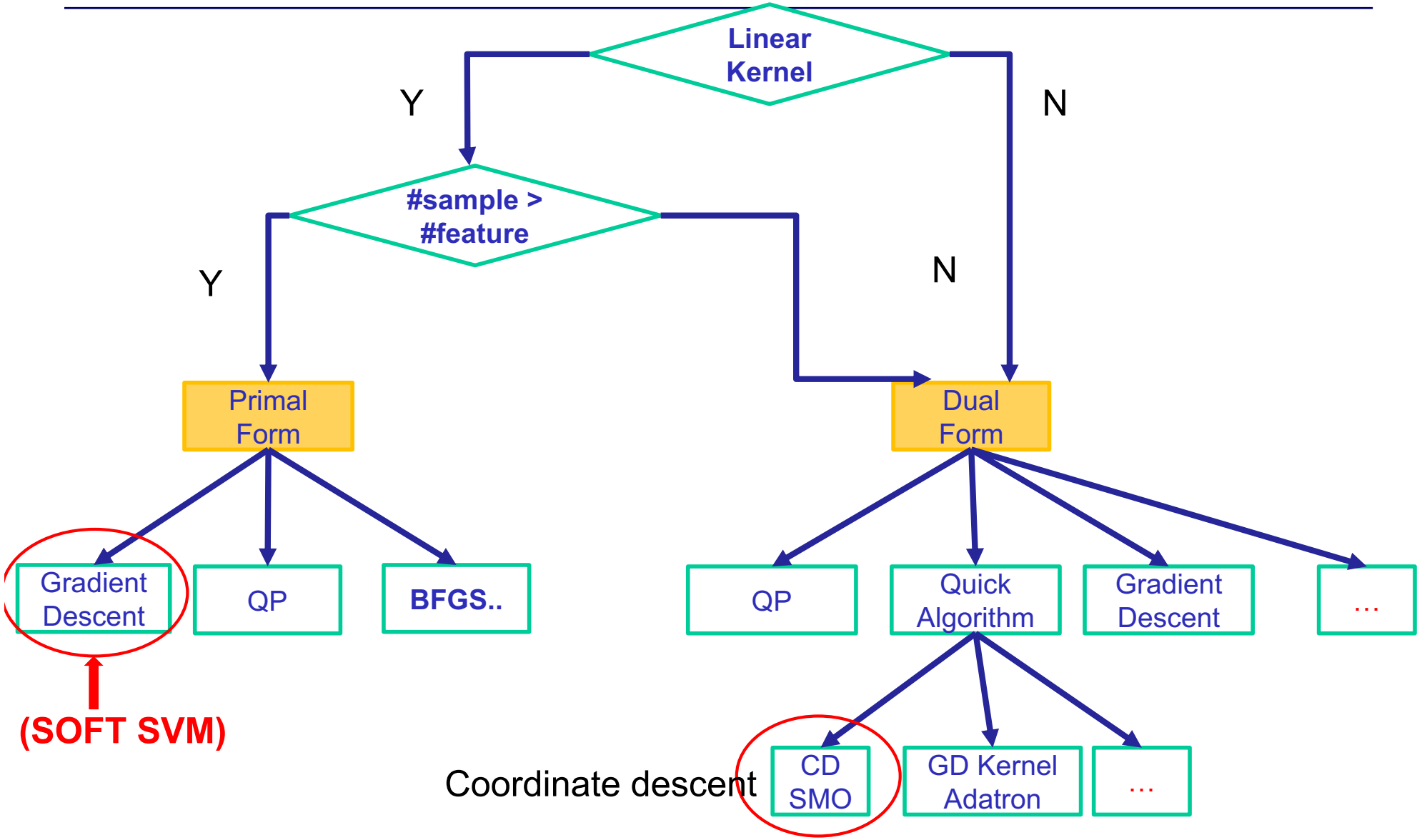
$$= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi\left(\mathbf{x}^{(i)}\right)^T \cdot \phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b}$$

$$= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}\right) + \hat{b}$$

Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual SVM algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

SVM Learning Algorithm Taxonomy



Soft SVM via unconstrained optimization

Objective function

$$\text{Min } \frac{\|\mathbf{w}\|^2}{2}$$

subject to constraints

$$y_i\left(\mathbf{w}\cdot\mathbf{x}_i+b\right)-1\geq 0 \;\; \forall \;\; i=1,\ldots,L.$$

Soft Margin SVM (Primal)

$$\min(W,b)=0.5\times\|w\|^2+C\sum_{i=1}^{L}\xi_i$$

$$\text{subject to: } y_i\left(\langle W,X_i\rangle+b\right)+\xi_i\geq 1 \quad \forall\, i=1,\ldots,L$$

$$\text{and } \xi_i\geq 0 \quad \forall\, i=1,\ldots,L$$

Lagrangean function

$$L_p(\mathbf{w})=\frac{1}{2}\langle\mathbf{w},\mathbf{w}\rangle-\sum\alpha_i[y_i(\langle\mathbf{w},x_i\rangle+b)-1] \qquad (4)$$

$$\alpha\geq 0$$

SVM learning as Optimization problem

Soft Margin SVM (Primal)

$$\min(W,b) = 0.5 \times \|w\|^2 + C\sum_{i=1}^{L} \xi_i$$

$$\text{subject to: } y_i\left(\langle W, X_i \rangle + b\right) + \xi_i \geq 1 \quad \forall i = 1,\ldots,L$$

$$\text{and } \xi_i \geq 0 \quad \forall i = 1,\ldots,L$$

- **Regularized hinge loss:**

Expected hinge loss on the training set

$$min_w \ \lambda/2 \ \ w'w \ + \ 1/m \ \Sigma_i(1 \ - \ y_i(w'x_i - b))_+$$

Trade off between margin and loss

Size of the margin

Positive for correctly classified examples, else negative

$(1 - z)_+ := \max\{0, 1\text{-}z\}$ (hinge loss)

First summand is a quadratic function, the sum is a piecewise linear function. The whole objective: piecewise quadratic.

SVM Optimization: Regularized hinge loss: Linear SVM Gradient has 2 cases

- **Regularized hinge loss:**

Expected hinge loss on the training set

$$min_w \ \lambda/2 \ \ w'w \ + \ 1/m \ \Sigma_i (1 \ - \ \ y_i(w'x_i - b))_+$$

- Gradient of Regularized hinge loss:

Hinge loss and regularization loss

$$\lambda w \qquad \text{if} \quad y_i(w'x_i - b)) > 1$$
$$\lambda w + y_i x_i \qquad \text{Otherwise}$$

$(1 - z)_+ := \max\{0, 1\text{-}z\}$ (hinge loss)

$W_{t+1} = w_t$ + average(gradient)

$W^{t+1} = w_t$ + average(regularization + hinge loss)

First summand is a quadratic function, the sum is a piecewise linear function. The whole objective: piecewise quadratic.

Single Core: SVM via Gradient Descent

```
1  #gradient descent (and with NO stochasticity!)
2  # Objective Function
3  # minw   λ/2   w'w   +   1/m Σi(1   −    yi(w'xi − b))+
4  # gradient
5  #   λw                if      yi(w'xi − b))  > 1    #correctly classified
6  #   λw  + yi xi       Otherwise                     #incorrectly classified
7  def SVM_GD(data,y,w=None,eta=0.01,iter_num=1000,regPara=0.01,stopCriteria=0.0001):
8      data = np.append(data,np.ones((data.shape[0],1)),axis=1)
9      if w==None:
10         w = np.random.normal(size=data.shape[1])
11     for i in range(iter_num):
12         wxy = np.dot(data,w)*y  #labeled margin
13         xy = -data*y[:,None]
14         zipv = zip(xy,wxy)
15         # Gradient of hinge loss: if wxy<0 then hinge loss is xy
16         g = sum([u for u,v in zipv if v<1])/data.shape[0]
17         wreg = w*1  #weight vector
18         wreg[-1] = 0
19         #wreg = np.array([w[0],w[1],0])
20         wdelta = eta*(g+regPara*wreg)
21         if sum(abs(wdelta))<=stopCriteria*sum(abs(w)):
22             break
23         w = w - wdelta
24     return w
```

$$\lambda w \quad \text{if} \quad y_i(w'x_i - b)) > 1$$
$$\lambda w + y_i x_i \quad \text{Otherwise}$$

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/dm2l73iznde7y4f/SVM-Notebook-Linear-Kernel-2015-06-19.ipynb

Convert constrained optimization problem to an unconstrained one using Lagrange multipliers

Learning an SVM has been formulated as a constrained optimization problem over $\mathbf{w}$ and ξ

$$\min_{\mathbf{w}\in\mathbb{R}^d,\xi_i\in\mathbb{R}+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i \text{ subject to } y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1-\xi_i \text{ for } i = 1\ldots N$$

The constraint $y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1-\xi_i$, can be written more concisely as

$$y_i f(\mathbf{x}_i) \geq 1-\xi_i$$

which, together with $\xi_i \geq 0$, is equivalent to

$$\xi_i = \max(0, 1-y_i f(\mathbf{x}_i))$$

Hence the learning problem is equivalent to the unconstrained optimization problem over $\mathbf{w}$

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

Gradient (or steepest) descent algorithm for SVM

Convert constrained optimization problem to an unconstrained one using Lagrange multipliers

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an average

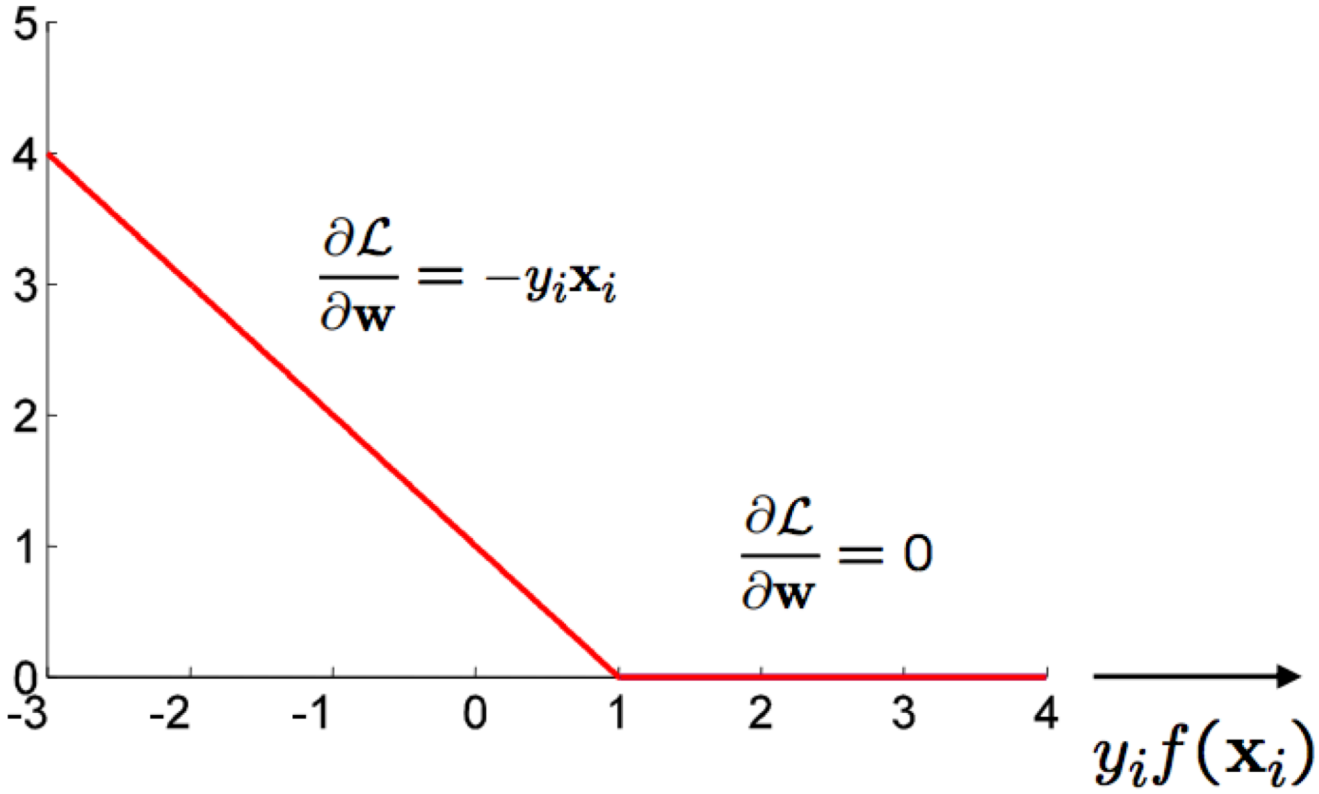
$$\begin{aligned} \min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\ &= \frac{1}{N}\sum_i^N \left(\frac{\lambda}{2}||\mathbf{w}||^2 + \max(0, 1 - y_i f(\mathbf{x}_i))\right) \end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$

Because the hinge loss is not differentiable, a sub-gradient is computed

Sub-gradient for hinge loss

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \qquad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$



Sub-gradient descent algorithm for SVM

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

The iterative update is

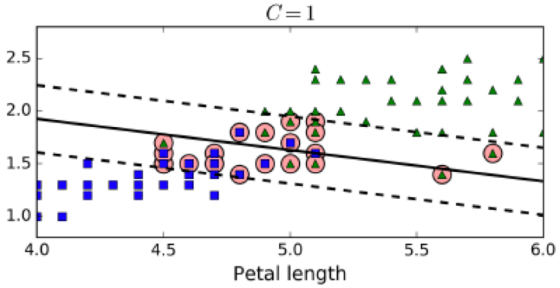
$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta\nabla_{\mathbf{w}_t}\mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta\frac{1}{N}\sum_i^N(\lambda\mathbf{w}_t + \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t))\end{aligned}$$

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 $\eta_t = \frac{1}{\lambda t}$



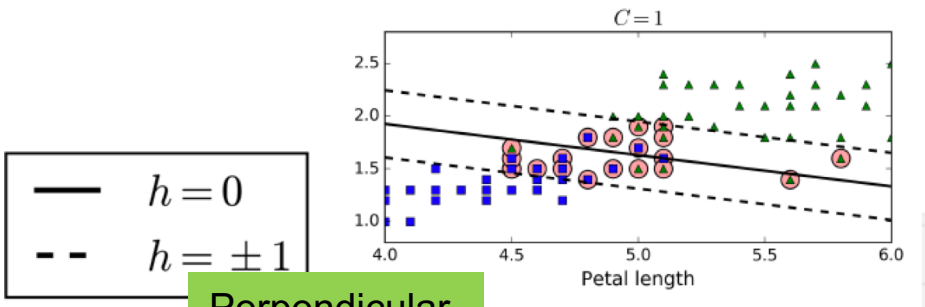
Train a binary SVM: Iris

```
  port numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.float64)  # Iris-Virginica

svm_clf = Pipeline((
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ))

svm_clf.fit(X, y)
```

Then, as usual, you can use the model to make predictions:

```
>>> svm_clf.predict([[5.5, 1.7]])
array([ 1.])
```

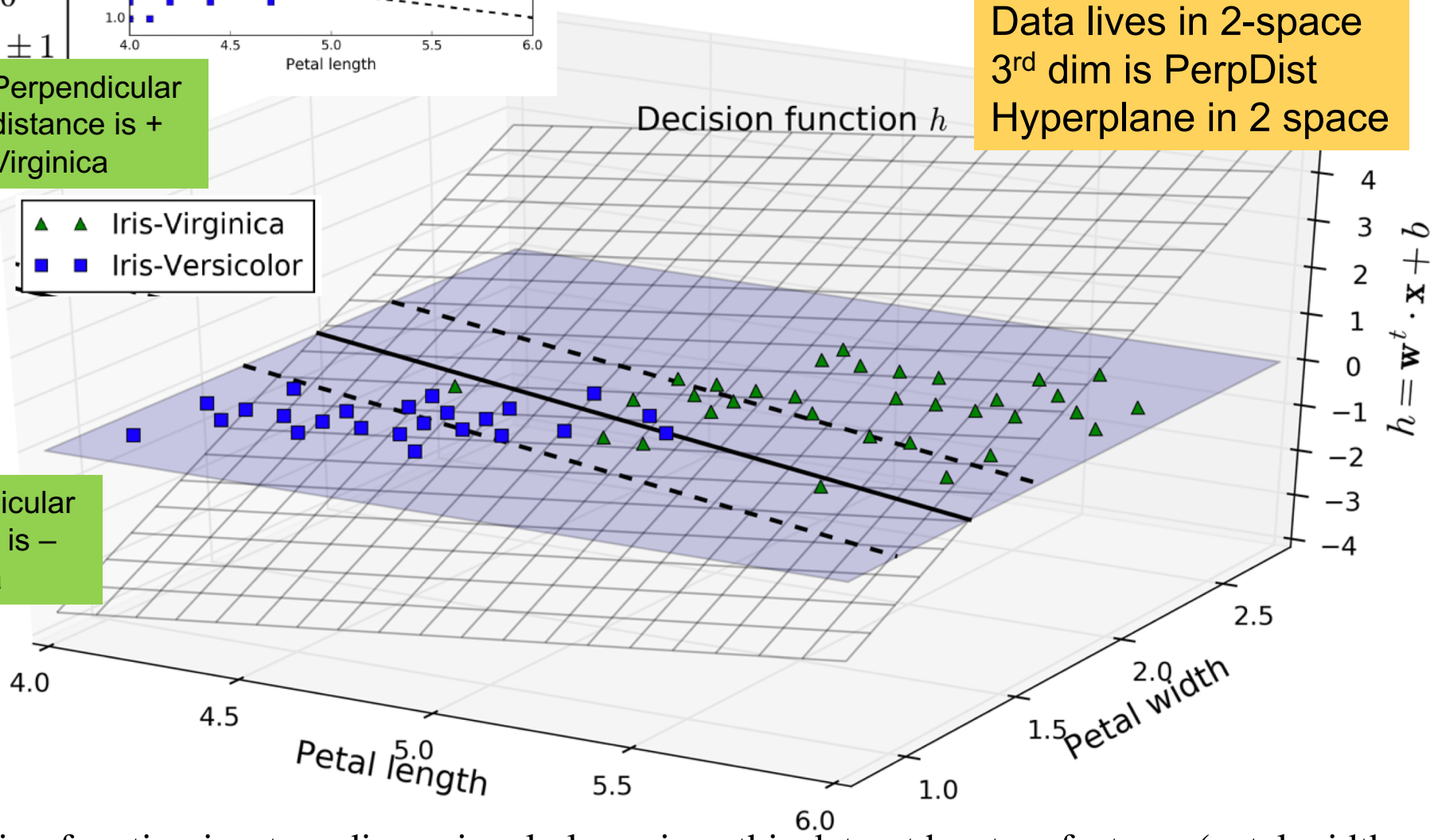
SVM Decision function



NOTE: 3 Dim
Data lives in 2-space
3rd dim is PerpDist
Hyperplane in 2 space

Perpendicular distance is + Virginica

Perpendicular distance is – Virginica



The decision function is a two-dimensional plane since this dataset has two features (petal width and petal length). The decision boundary is the set of points where the decision function is equal to 0: it is the intersection of two planes, which is a straight line (represented by the thick solid line)

Include the bias term in the regularization

- **You can include it if you standardize the data!**

- **Some SKLearn ML algorithms (such as, the LinearSVC class) regularize the bias term, so you should center the training set first by subtracting its mean.**
- **This is automatic if you scale the data using the StandardScaler**

Comparison of Scikit-Learn classes for SVM classification

- **The LinearSVC class is based on the liblinear library, which implements an optimized algorithm for linear SVMs.1 It does not support the kernel trick, but it scales almost linearly with the number of training instances and the number of features: its training time complexity is roughly *O(m X n)*.**
- **SVC is based on libsvm (and supports the kernel trick; not gradient descent based; so is slow)**

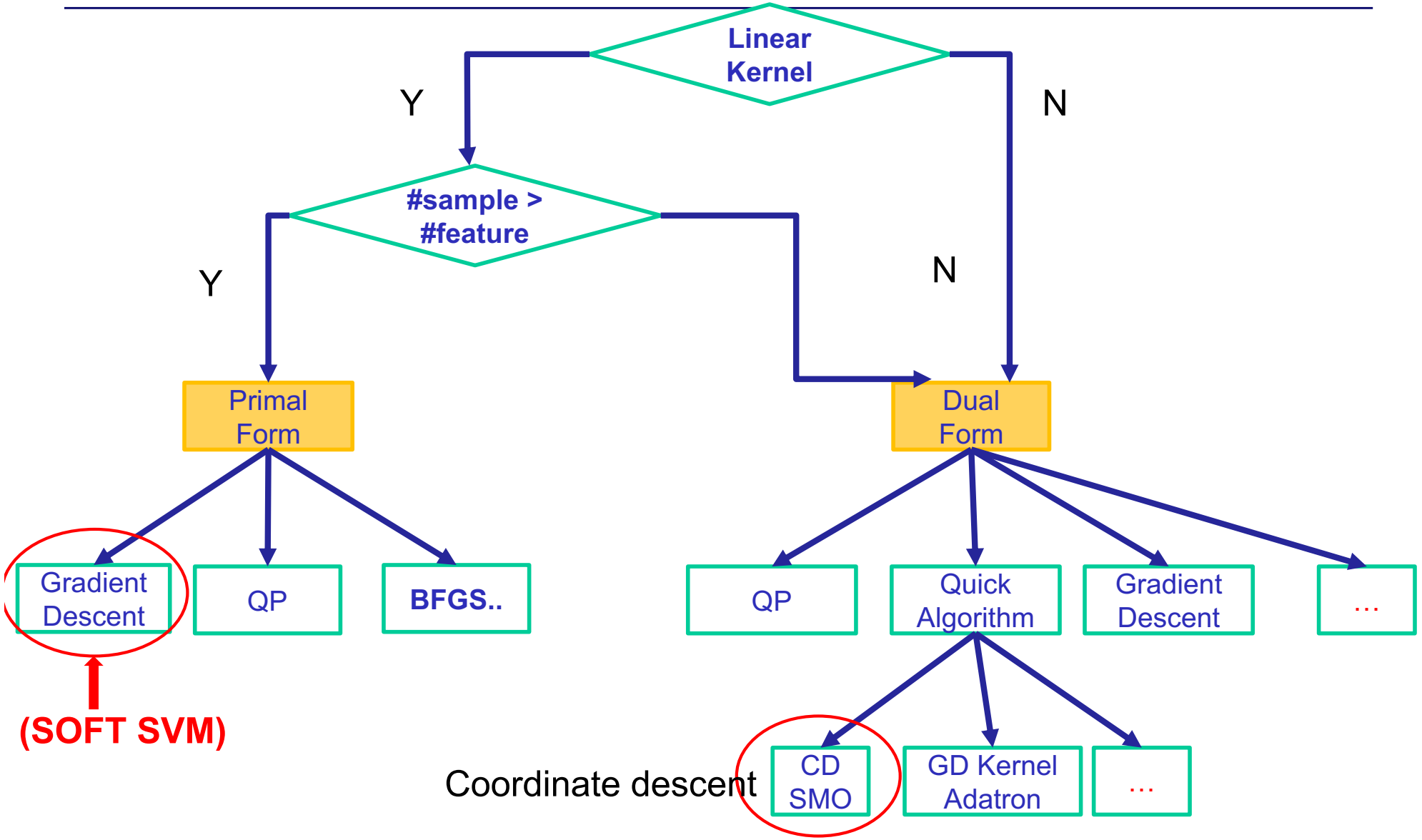
Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes

SVC versus LinearSVC

Alternatively, you could use the `SVC` class, using `SVC(kernel="linear", C=1)`, but it is much slower, especially with large training sets, so it is not recommended. Another option is to use the `SGDClassifier` class, with `SGDClassifier(loss="hinge", alpha=1/(m*C))`. This applies regular Stochastic Gradient Descent (see Chapter 4) to train a linear SVM classifier. It does not converge as fast as the `LinearSVC` class, but it can be useful to handle huge datasets that do not fit in memory (out-of-core training), or to handle online classification tasks.

The `LinearSVC` class regularizes the bias term, so you should center the training set first by subtracting its mean. This is automatic if you scale the data using the `StandardScaler`. Moreover, make sure you set the `loss` hyperparameter to `"hinge"`, as it is not the default value. Finally, for better performance you should set the `dual` hyperparameter to `False`, unless there are more features than training instances (we will discuss duality later in the chapter).

SVM Learning Algorithm Taxonomy



Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual SVM algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

Multinomial Linear Classifiers via GD

Recap

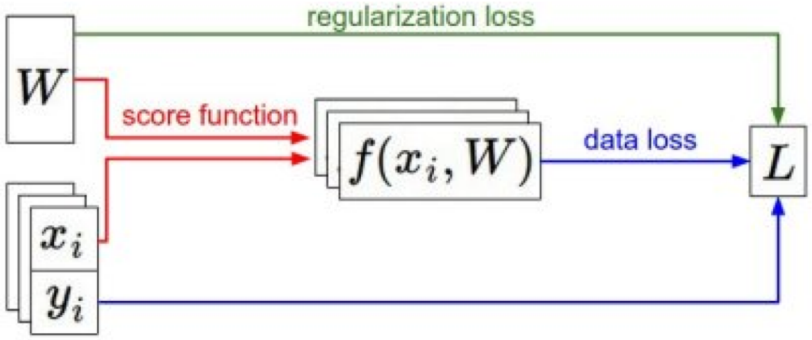
- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full (regularized) loss

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Usually at initialization W are small numbers, so all scores (aka perpendicular distances) will be ~= 0.

What is the loss?

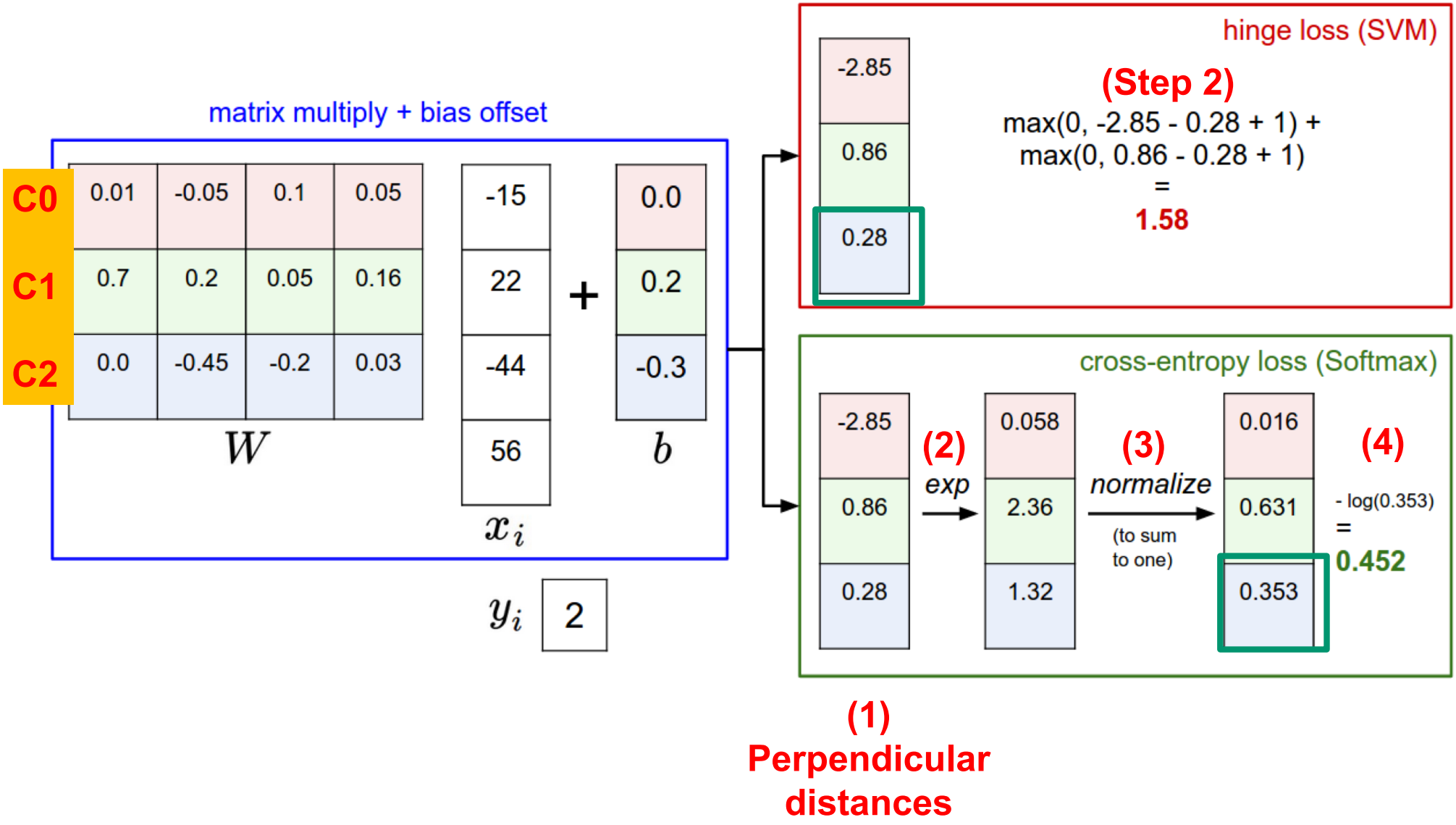
– Log(1/k)

Base 10

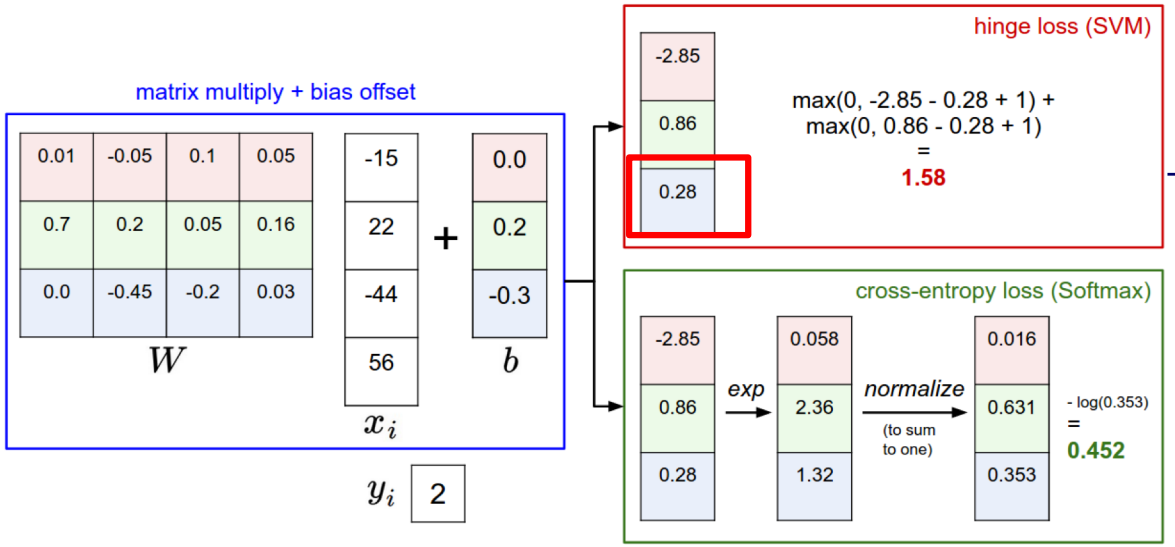
	(1)Perpendicular distances	(2) exp	(3)normalize	probabilities
cat	**3.2**	**24.5**	**0.13**	→ L_i = -log(0.13, 10) = **0.89**
car	5.1	164.0	0.87	
frog	-1.7	0.18	0.00	

- For CIFAR-10 loss after init could be
- -log(1/10)
- 2.302585

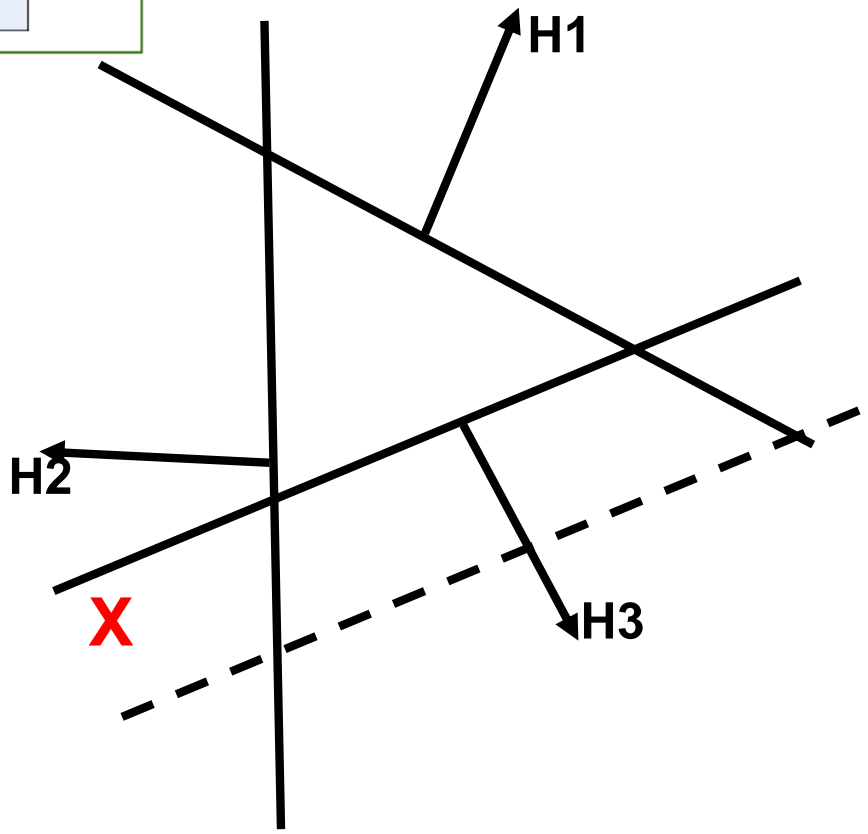
Hinge Loss vs. Cross-entropy Loss



SVM Classifier



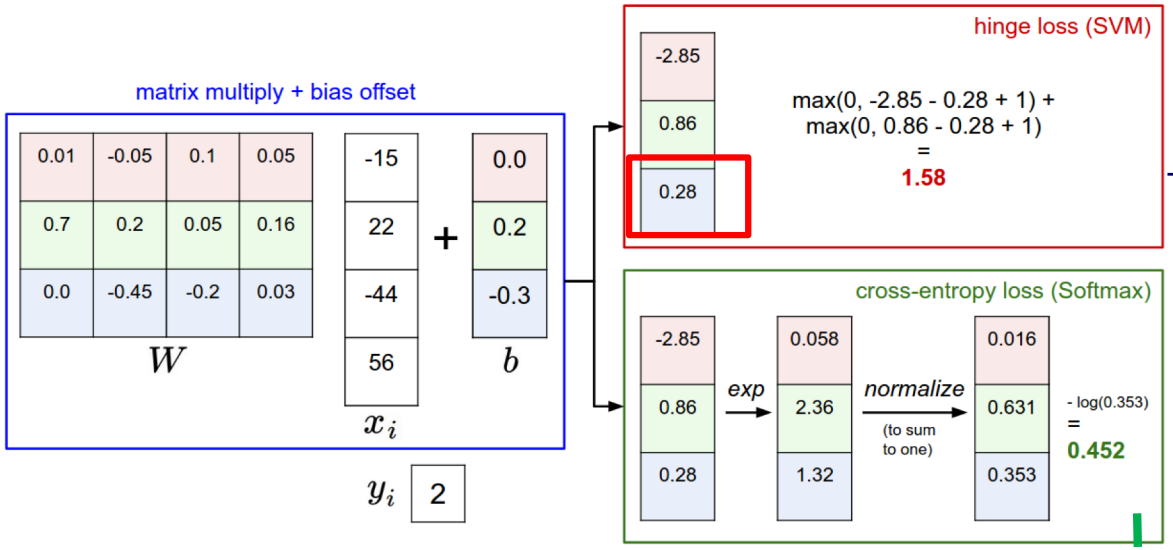
SVM loss demands the test case needs to be actual on the positive side of the true class supporting hyperplane (+1 hyperplane) and in addition the test case can not be on the positive side any other class hyperplane. It there are, the positive perpendicular distance needs to < (the perpendicular distance of the actual class +1)



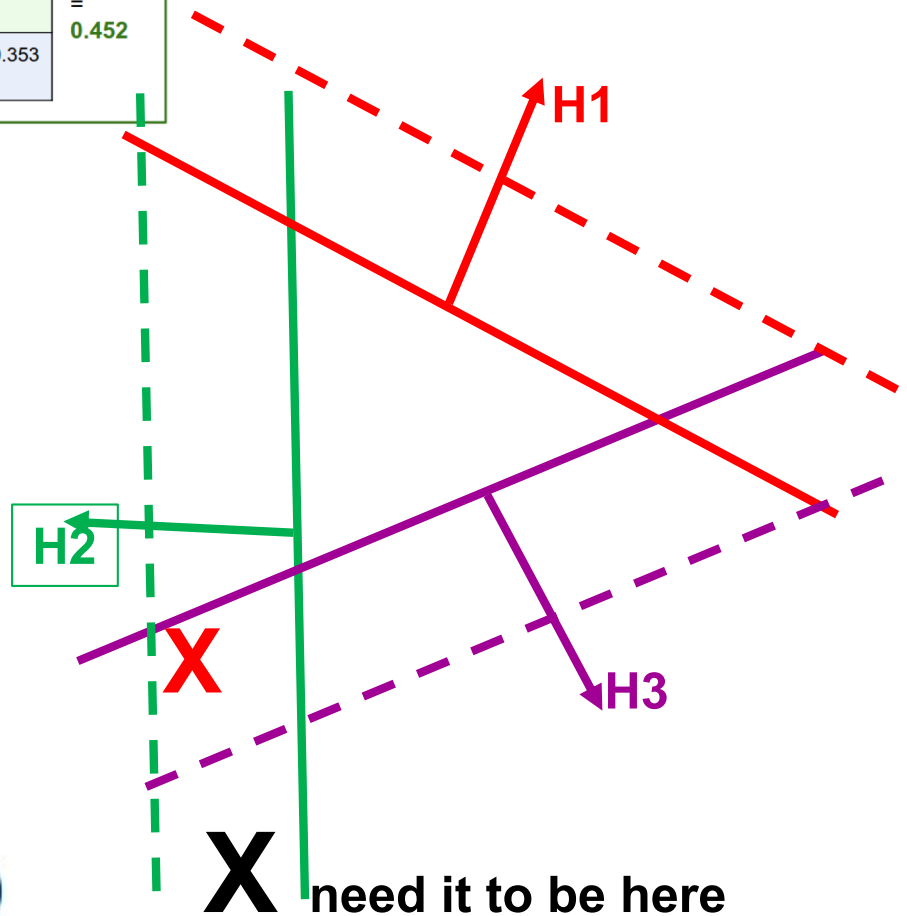
SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SVM Classifier



SVM loss demands the test case needs to be actual on the positive side of the true class supporting hyperplane (+1 hyperplane) and in addition the test case can not be on the positive side any other class hyperplane. It there are, the positive perpendicular distance needs to < (the perpendicular distance of the actual class +1)



SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multinomial Linear Classifiers

Recap

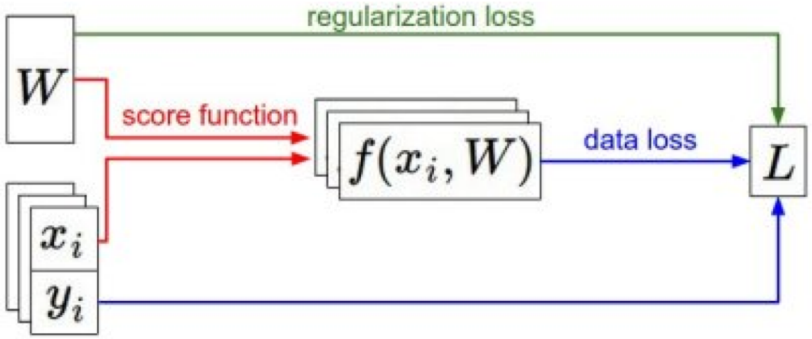
- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

Full (regularized) loss

SVM objective function

- **Regularized objective function for MultiClass SVM**

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)\right] + \alpha R(W)$$

- **Single example loss (k-1 max component)**

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + 1)\right]$$

- **Consider a simple dataset 3 by 1 → 3 classes**
  - that contains three 1-dimensional points and three classes.
  - The full SVM loss (without regularization) becomes
    - 3 examples → 3 data losses with 2 loss components

$L_0 = \underbrace{\max(0, w_1^T x_0 - w_0^T x_0 + 1)}_{\text{Class}_1 - \text{class}_0} + \underbrace{\max(0, w_2^T x_0 - w_0^T x_0 + 1)}_{\text{Class}_2 - \text{class}_0}$	Target y is class 0
$L_1 = \max(0, w_0^T x_1 - w_1^T x_1 + 1) + \max(0, w_2^T x_1 - w_1^T x_1 + 1)$	Target y is class 1
$L_2 = \max(0, w_0^T x_2 - w_2^T x_2 + 1) + \max(0, w_1^T x_2 - w_2^T x_2 + 1)$	Target y is class 2
$L = (L_0 + L_1 + L_2)/3$	Average SVM hinge data loss

For more background see http://cs231n.github.io/optimization-1/

Multiclass SVM Loss

```
def svm_loss_vectorized(W, X, y, reg):
  """
  Structured SVM loss function, vectorized implementation.

  Inputs and outputs are the same as svm_loss_naive.
  """
  loss = 0.0
  dW = np.zeros(W.shape) # initialize the gradient as zero
  num_train = X.shape[0]

  #############################################################################
  # Implement a vectorized version of the structured SVM loss, storing
  # result in loss.
  #############################################################################
  scores = X.dot(W)
  yi_scores = scores[np.arange(scores.shape[0]),y] # http://stackoverf
  margins = np.maximum(0, scores - np.matrix(yi_scores).T + 1)
  margins[np.arange(num_train),y] = 0
  loss = np.mean(np.sum(margins, axis=1))
  loss += 0.5 * reg * np.sum(W * W)

  #############################################################################
  # Implement a vectorized version of the gradient for the structured
  # loss, storing the result in dW.
  #
  # Hint: Instead of computing the gradient from scratch, it may be ea
  # to reuse some of the intermediate values that you used to compute
  # loss.
  #############################################################################
  binary = margins
  binary[margins > 0] = 1
  row_sum = np.sum(binary, axis=1)
  binary[np.arange(num_train), y] = -row_sum.T
  dW = np.dot(X.T, binary)

  # Average
  dW /= num_train

  # Regularize
  dW += reg*W

  return loss, dW
```

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)\right] + \alpha R(W)$$

https://mlxai.github.io/2017/01/06/vectorized-implementation-of-svm-loss-and-gradient-update.html

3 examples, 3 classes



	Image #1	Image #2	Image #3
Dog	-0.39	-4.61	1.03
Cat	1.49	3.28	-2.37
Horse	4.21	1.46	-2.27

Figure 2: An example of applying hinge loss to a 3-class image classification problem.

Let's again compute the loss for the dog class:

```
Multi-class SVM Loss                                                    Python
1 >>> max(0, 1.49 - (-0.39) + 1) + max(0, 4.21 - (-0.39) + 1)
2 8.48
3 >>>
```

Notice how that our summation has expanded to include two terms — the difference between the predicted dog score and ***both*** the cat and horse score.

Similarly, we can compute the loss for the cat class:

```
Multi-class SVM Loss                                                    Python
1 >>> max(0, -4.61 - 3.28 + 1) + max(0, 1.46 - 3.28 + 1)
2 0
3 >>>
```

And finally the loss for the horse class:

```
Multi-class SVM Loss                                                    Python
1 >>> max(0, 1.03 - (-2.27) + 1) + max(0, -2.37 - (-2.27) + 1)
2 5.199999999999999
3 >>>
```

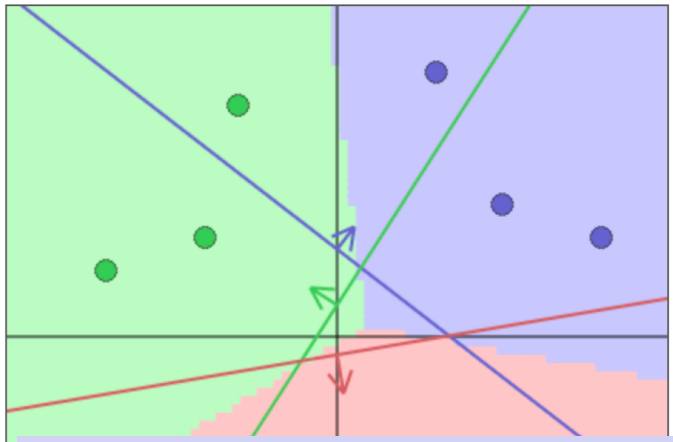
The total loss is therefore:

```
Multi-class SVM Loss                                                    Python
1 >>> (8.48 + 0.0 + 5.2) / 3
2 4.56
3 >>>
```

https://www.pyimagesearch.com/2016/09/05/multi-class-svm-loss/

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

$w_2 \leftarrow w_2 + (x)$

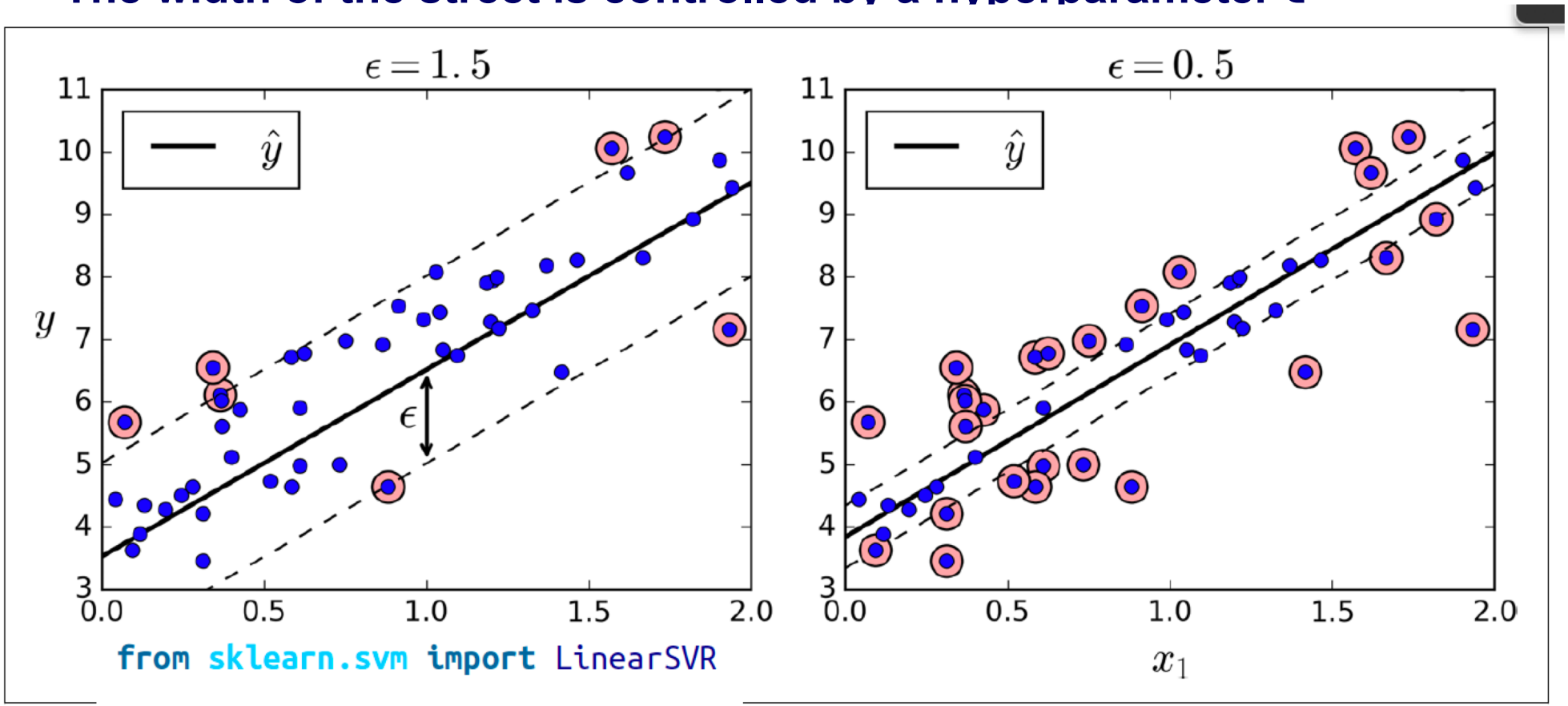
$(3,-2,-1) + (1,-0.5,3)$ → w_2=(4,-2.5,2)

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

SVM Regression

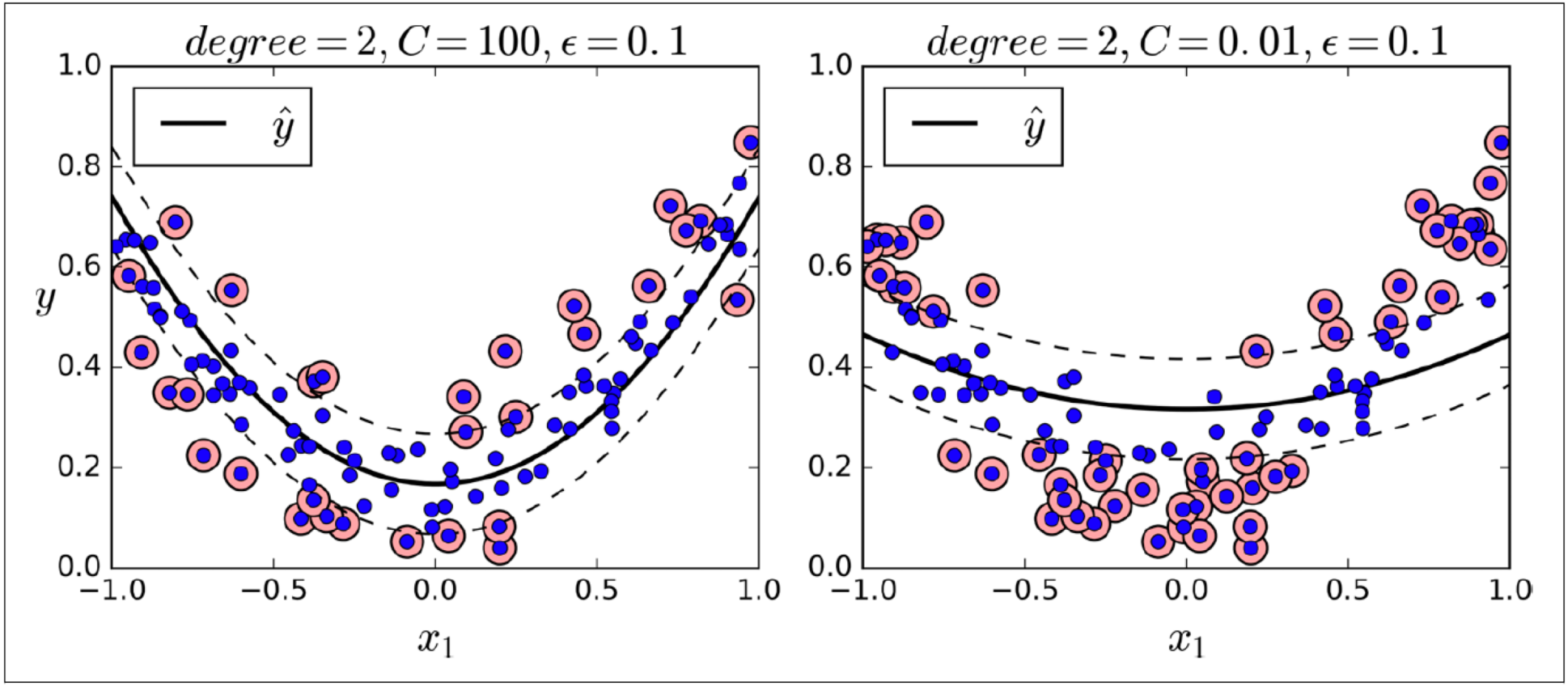
- **The trick is to reverse the objective:**
  - instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations (i.e., instances off the street).
- **The width of the street is controlled by a hyperparameter ϵ**




```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

Polynomial Kernels for SVR

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```



Outline

1. Introduction
2. Support vector machines (SVMs) loss criteria
3. Maximal margin classifier
4. Primal and dual SVM algorithm
5. Soft SVMs (vs Hard SVMs)
6. Kernel SVMs
7. Small app
8. SVM generalizations
    1. Multiclass SVMs
    2. SVMs for regression
9. Summary

End of lecture