

Recap and Summary of Online Learning

- **Online learning**

- ▶ Iterative game between teacher and learner

- **Design principles of online learning**

- ▶ Trade-off amount of change (conservative) and reduction in loss (corrective)

- **Online learning algorithms**

- ▶ Perceptron (fixed learning rate for all examples)
- ▶ Passive-Aggressive (fixed learning rate for each example)
- ▶ Confidence-weighted classifier (fixed learning for each feature and each example)

Lecture #4: Support Vector Machines

Janardhan Rao (Jana) Doppa

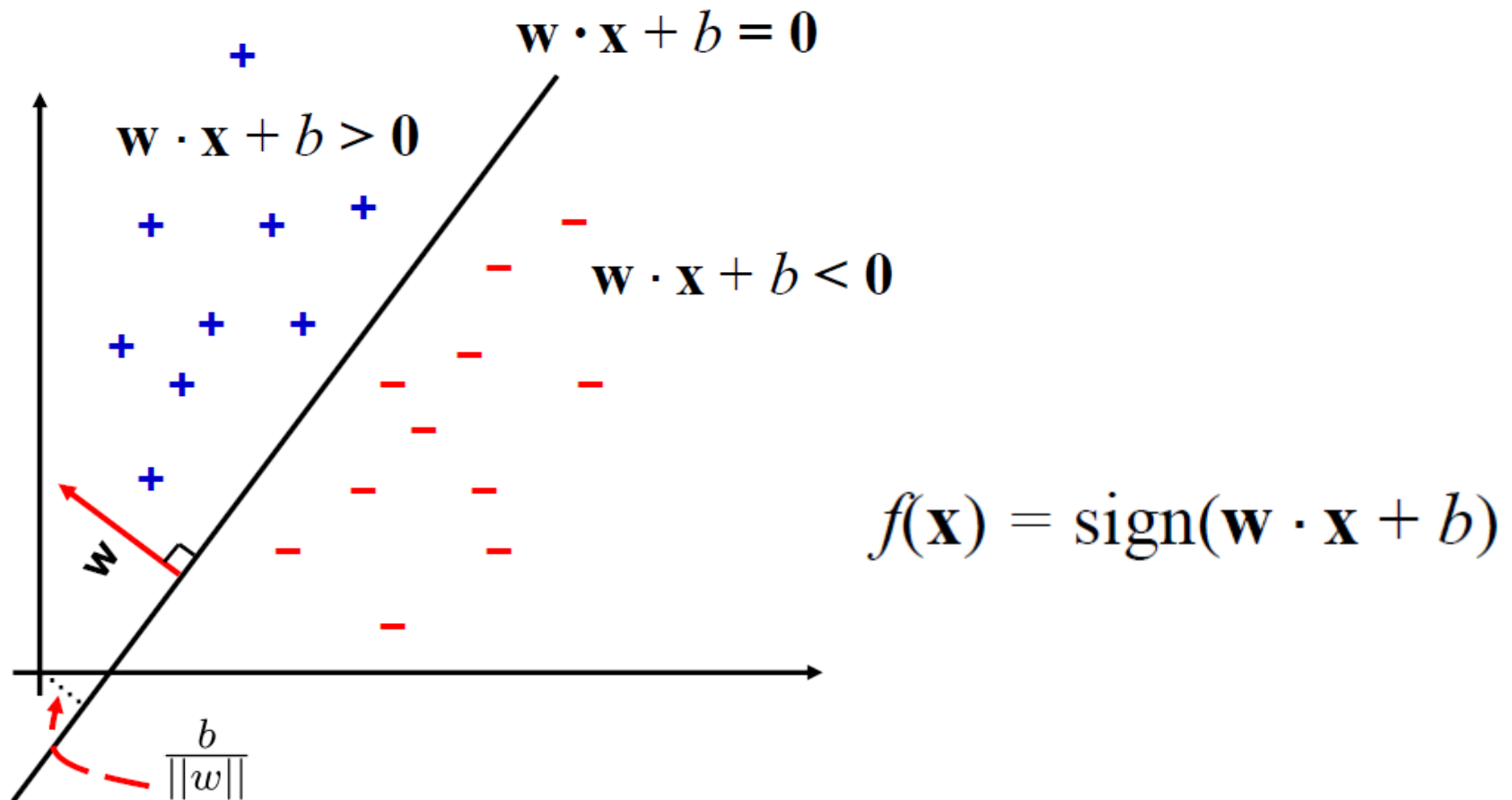
School of EECS, Washington State University

Online vs. Batch

- Local optima? Vs. Global optima
- Less Memory vs. High memory
- Higher time vs. Lower time??
- Error?

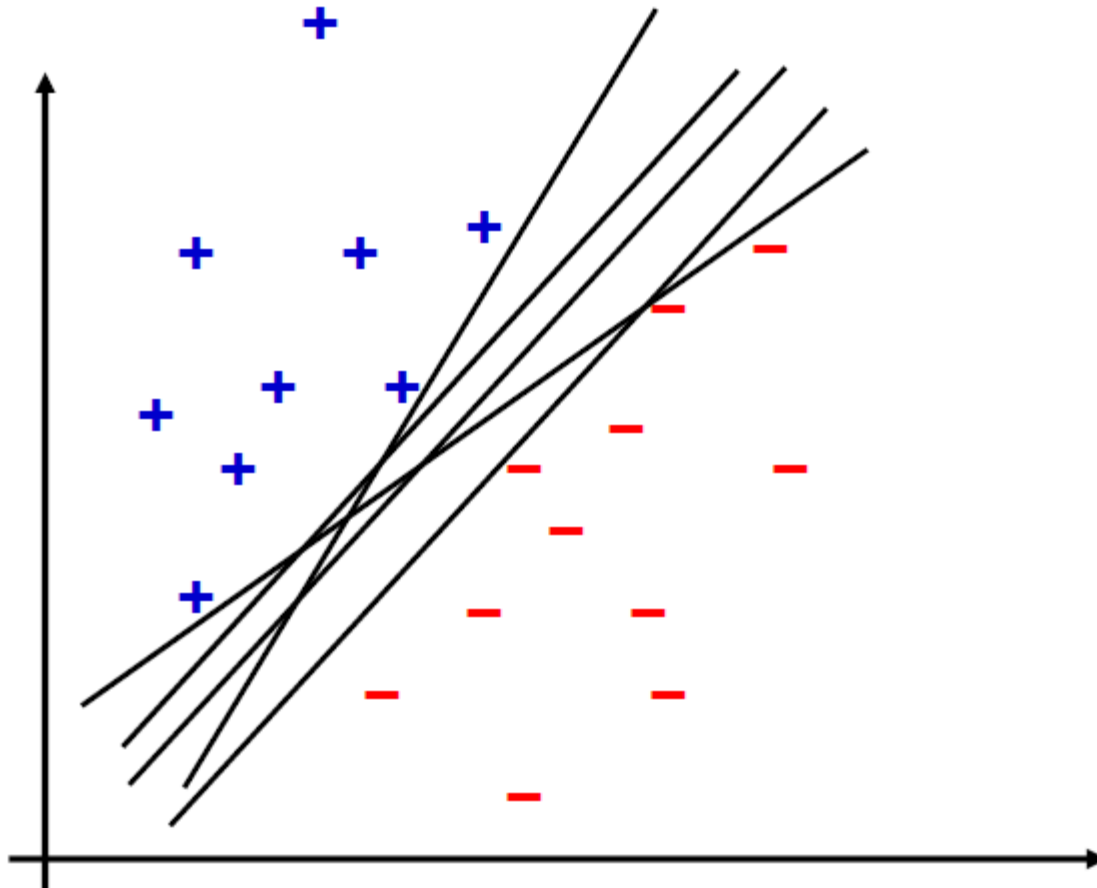
Perceptron Revisited: Linear Separator

- Binary classification can be viewed as the task of separating classes in a given feature space



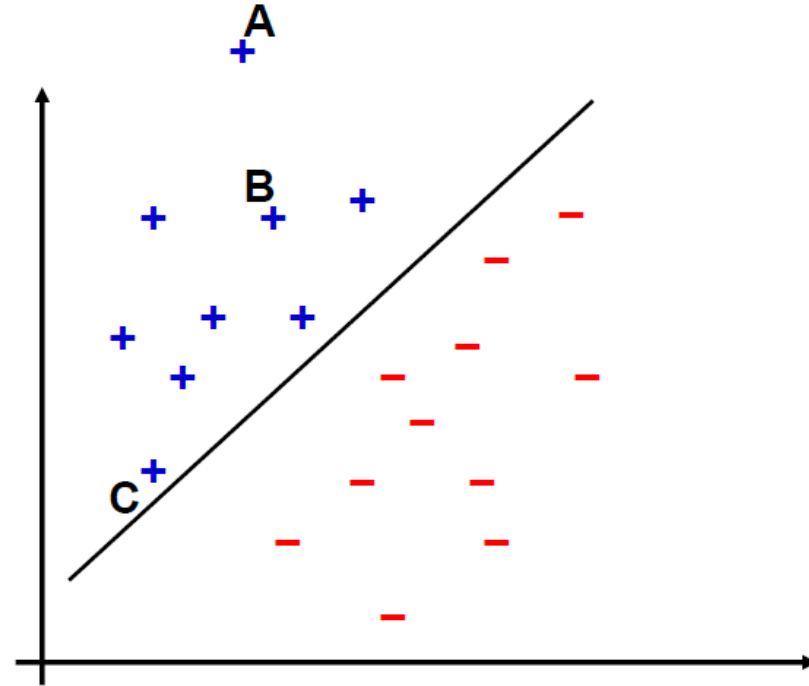
Linear Separators

- Which of the linear separators is optimal?



Intuition of Margin

- Consider points A, B, and C
- We are quite confident in our prediction for A because it is far from the decision boundary
- In contrast, we are not so confident in our prediction for C because a slight change in the decision boundary may flip the decision



Given a training set, we would like to make all predictions correct and confident! This leads to the concept of margin.

Functional Margin

- Given a linear classifier parameterized by (\mathbf{w}, b) , we define its functional margin w.r.t training example (x_i, y_i) as:

$$\gamma_i = y_i(\mathbf{w} \cdot x_i + b)$$

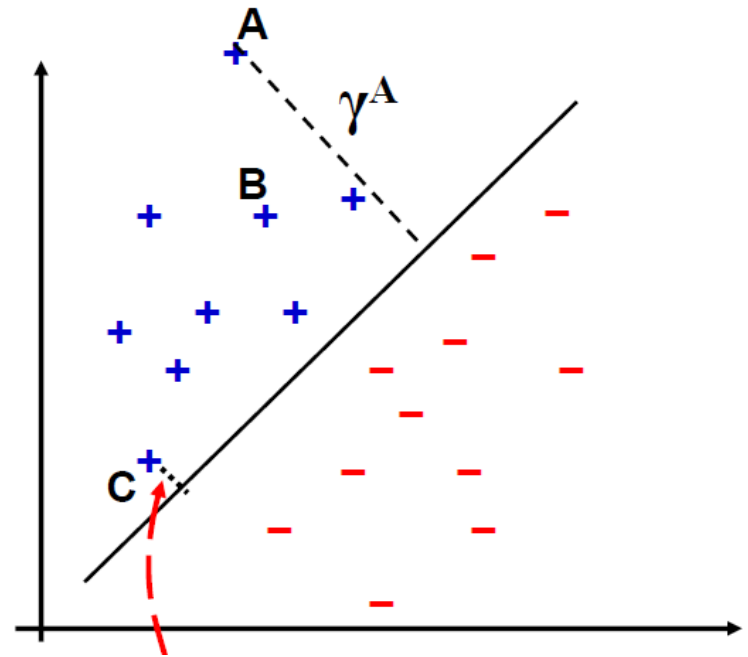
$\gamma_i > 0$ if the example is classified correctly

- If we rescale (\mathbf{w}, b) by a factor α , functional margin gets multiplied by α
 - we can make it arbitrarily large without changing anything meaningful

Geometric Margin

- The geometric margin of (\mathbf{w}, b) w.r.t. example (x_i, y_i) is the distance from x_i to the decision surface, which can be computed as:

$$\gamma_i = \frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$



- Given a training set $S = (x_i, y_i): i = 1, 2, \dots, N$ the geometric margin of the classifier w.r.t. S is

$$\gamma = \min_{i=1,2,\dots,N} \gamma_i$$

Maximum Margin Classifier

- Given a linearly separable training set $(x_i, y_i): i = 1, 2, \dots, N$, we would like to find a linear classifier with maximum margin
- This can be represented as an optimization problem

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

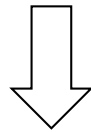
$$\text{subject to: } y^{(i)} \frac{(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|} \geq \gamma, \quad i = 1, \dots, N$$

Maximum Margin Classifier

- Let $\gamma' = \gamma \|w\|$, we can rewrite the optimization problem as follows:

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

$$\text{subject to: } y^{(i)} \frac{(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|} \geq \gamma, \quad i = 1, \dots, N$$



$$\max_{\mathbf{w}, b, \gamma'} \frac{\gamma'}{\|\mathbf{w}\|}$$

$$\text{subject to: } y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq \gamma', \quad i = 1, \dots, N$$

Maximum Margin Classifier

- Note that rescaling w and b by $1/\gamma'$ will not change the classifier -- we can thus further reformulate the optimization problem

$$\begin{aligned} & \max_{\mathbf{w}, b} \frac{\gamma'}{\|\mathbf{w}\|} \\ & \text{subject to : } y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq \gamma', \quad i = 1, \dots, N \end{aligned}$$



$$\begin{aligned} & \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \quad (\text{or equivalently } \min_{\mathbf{w}, b} \|\mathbf{w}\|^2) \\ & \text{subject to : } y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N \end{aligned}$$

Maximum Margin Classifier

- Maximizing the geometric margin is equivalent to minimizing the magnitude of \mathbf{w} subject to maintaining a functional margin of at least 1

$$\max_{\mathbf{w}, b} \frac{\gamma'}{\|\mathbf{w}\|}$$

$$\text{subject to : } y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq \gamma', \quad i = 1, \dots, N$$



$$\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \quad (\text{or equivalently } \min_{\mathbf{w}, b} \|\mathbf{w}\|^2)$$

$$\text{subject to : } y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N$$

Interpretation for this Optimization Problem

- **Regularization** in Machine Learning
- I want to pick simple functions or hypotheses that best explain my data (Occam's Razor)
 - ▲ To achieve this goal, we have two competing objectives: how well are doing on the training data (empirical risk minimization) and how complex my model/function is (regularizer)
 - ▲ Magnitude of w ($\|w\|$) is kind of measuring the complexity
 - ▲ constraints (achieve a functional margin of at least 1) is measuring the training/empirical error

Maximum Margin Classifier: Formulation

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to: } & y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N \end{aligned}$$

- This results in a **quadratic optimization (QP) problem** with linear inequality constraints
- This is a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist
 - ▲ One could solve for **w** using any of these methods

Maximum Margin Classifier: Formulation

$$\begin{array}{ll} \min_{\mathbf{w}, b} & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to:} & y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N \end{array}$$

- We will see that it is useful to first formulate an equivalent dual optimization problem and solve it instead
 - ▲ This requires a bit of machinery!

Aside: Constrained Optimization

- Suppose we want to: minimize $f(x)$ subject to constraints $g_i(x) \leq 0, i = 1, \dots, m$
 - ▶ $\min_x f(x)$ subject to $g_i(x) \leq 0, i = 1, \dots, m$
- Consider the following function known as the lagrangian
- Under certain conditions it can be shown that for a solution x' to the above problem we have

$$\mathcal{L}(x, \alpha) = f(x) + \sum_i \alpha_i g_i(x)$$

$$f(x') = \underbrace{\min_x \max_{\alpha} \mathcal{L}(x, \alpha)}_{\text{Primal form}} = \underbrace{\max_{\alpha} \min_x \mathcal{L}(x, \alpha)}_{\text{Dual form}}$$

Primal form

Dual form

subject to $\alpha_i \geq 0$

Back to the Original Problem

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to } 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b) \leq 0 \quad \text{for } i = 1, \dots, N \end{aligned}$$

- The lagrangian is

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))$$

- We want to solve

$$\max_{\alpha} \min_{w, b} \mathcal{L}(w, b, \alpha) \quad s.t. \quad \alpha_i \geq 0$$

Back to the Original Problem

- We want to solve

$$\max_{\alpha} \min_{w, b} \mathcal{L}(w, b, \alpha) \quad s.t. \quad \alpha_i \geq 0$$

- Setting the gradient w.r.t w and b to zero, we get

$$\mathbf{w} + \sum_{i=1}^n \alpha_i (-y_i) \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

Back to the Original Problem

- Substitute w in lagrangian

$$\begin{aligned}\mathcal{L} &= \frac{1}{2} \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^n \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^n \alpha_i \left(1 - y_i \left(\sum_{j=1}^n \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b \right) \right) \\ &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i - \sum_{i=1}^n \alpha_i y_i \sum_{j=1}^n \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - b \sum_{i=1}^n \alpha_i y_i \\ &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i\end{aligned}$$

- This is a function of α' s only!

Dual Problem

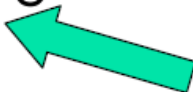

- The new objective function is a function of α' s only
- It is known as the dual problem
 - ▲ If we know all the α' s, we know the weights w
- The original problem is known as the primal problem
- The objective function of the dual problem needs to be maximized!

Dual Problem

- The objective function of the dual problem needs to be maximized!
- Therefore, the dual problem is:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $\alpha_i \geq 0,$ $\sum_{i=1}^n \alpha_i y_i = 0$



Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

Dual Problem

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $\alpha_i \geq 0$, $\sum_{i=1}^n \alpha_i y_i = 0$

- This is also a QP problem
 - ▲ A global maxima of α' s can always be found
- Weights w can be recovered by
$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$
- b can also be recovered (we will skip the details)

Characteristics of Solution

- Many of the α_i 's are zero
 - ▲ Weights \mathbf{w} is a linear combination of small number of data points
- \mathbf{x}_i with non-zero α_i are called support vectors (SVs)
 - ▲ The decision boundary is determined only by the SVs
 - ▲ Let $t_j (j = 1, 2, \dots, s)$ be the indices of the s support vectors. We can write

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

Characteristics of Solution

- For classifying a new input example \mathbf{z} , compute

$$\mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$$

- ▲ Classify \mathbf{z} as positive if the sum is positive, and negative otherwise
- ▲ **Note:** \mathbf{w} need not be formed explicitly, rather we can classify \mathbf{z} by taking inner products with the support vectors (useful when we generalize the notion of inner product later)

Some Observations

- #1. Weights are linear combination of your training examples
- #2. Dual weights (alpha variables) determine which training examples count towards the decision boundary and by how much
- Sign (
 - ▲ $\sum_i \alpha_i * X_i * X$ // all positive training examples
 - ▲ $-\sum_j \alpha_j * X_j * X$ // all negative training examples
 - ▲)

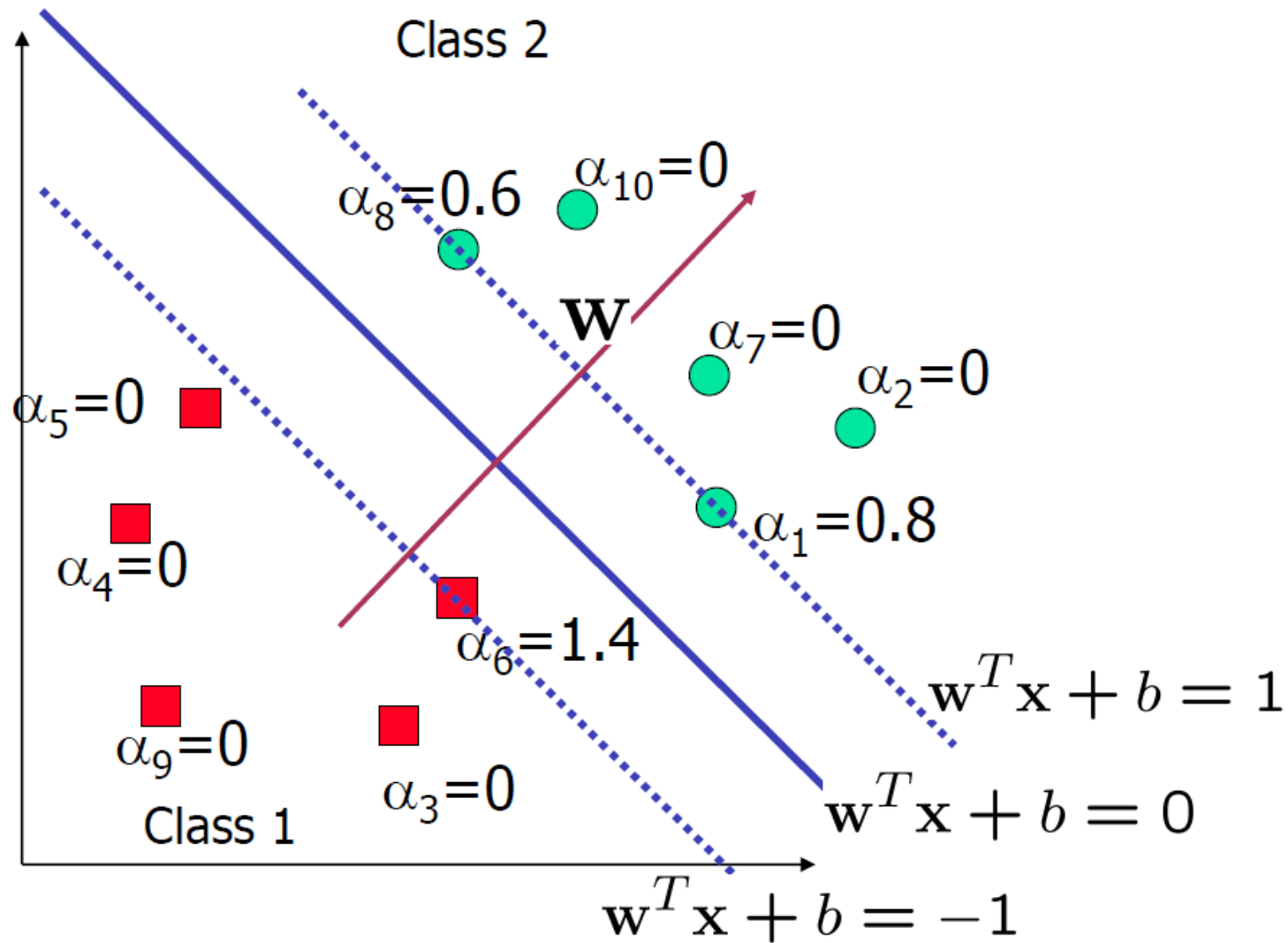
The Quadratic Programming Problem

- Many approaches have been proposed
 - ▲ Loqo, cplex, etc. (see <http://www.numerical.rl.ac.uk/qp/qp.html>)
- Most are “interior-point” methods
 - ▲ Start with an initial solution that can violate the constraints
 - ▲ Improve this solution by optimizing the objective function and/or reducing the amount of constraint violation
- For SVM, sequential minimal optimization (SMO) seems to be the most popular

The Quadratic Programming Problem

- **SMO algorithm**
 - ▶ A QP with two variables is trivial to solve
 - ▶ Each iteration of SMO picks a pair of (α_i, α_j) and solve the QP with these two variables
 - ▶ repeat until convergence
- In practice, we can just regard the QP solver as a “black-box” without bothering how it works

A Geometric Interpretation

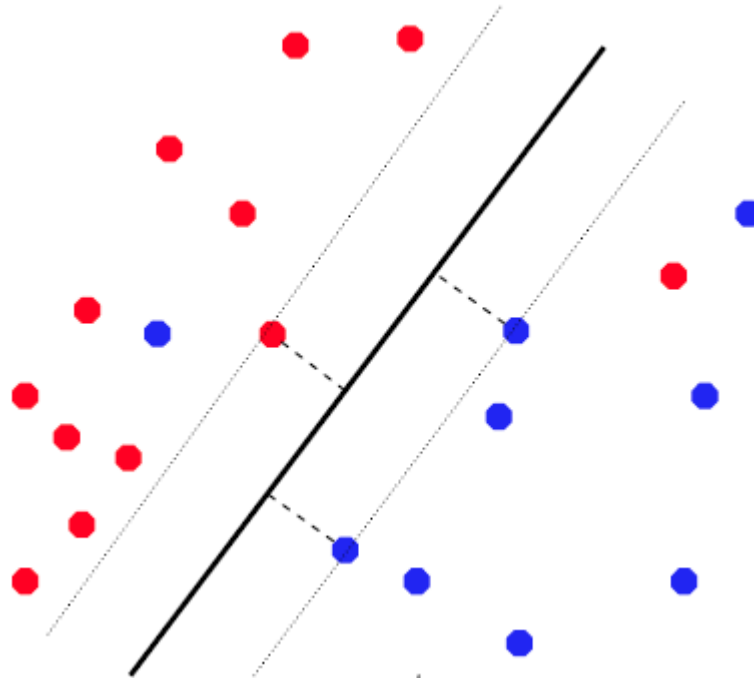


Summary so far

- We demonstrated that we prefer to have linear classifiers with large margin
- We formulated the problem of finding the maximum margin linear classifier as a quadratic optimization problem
- This problem can be solved by solving its dual problem, and efficient QP algorithms are available
- Problem Solved?

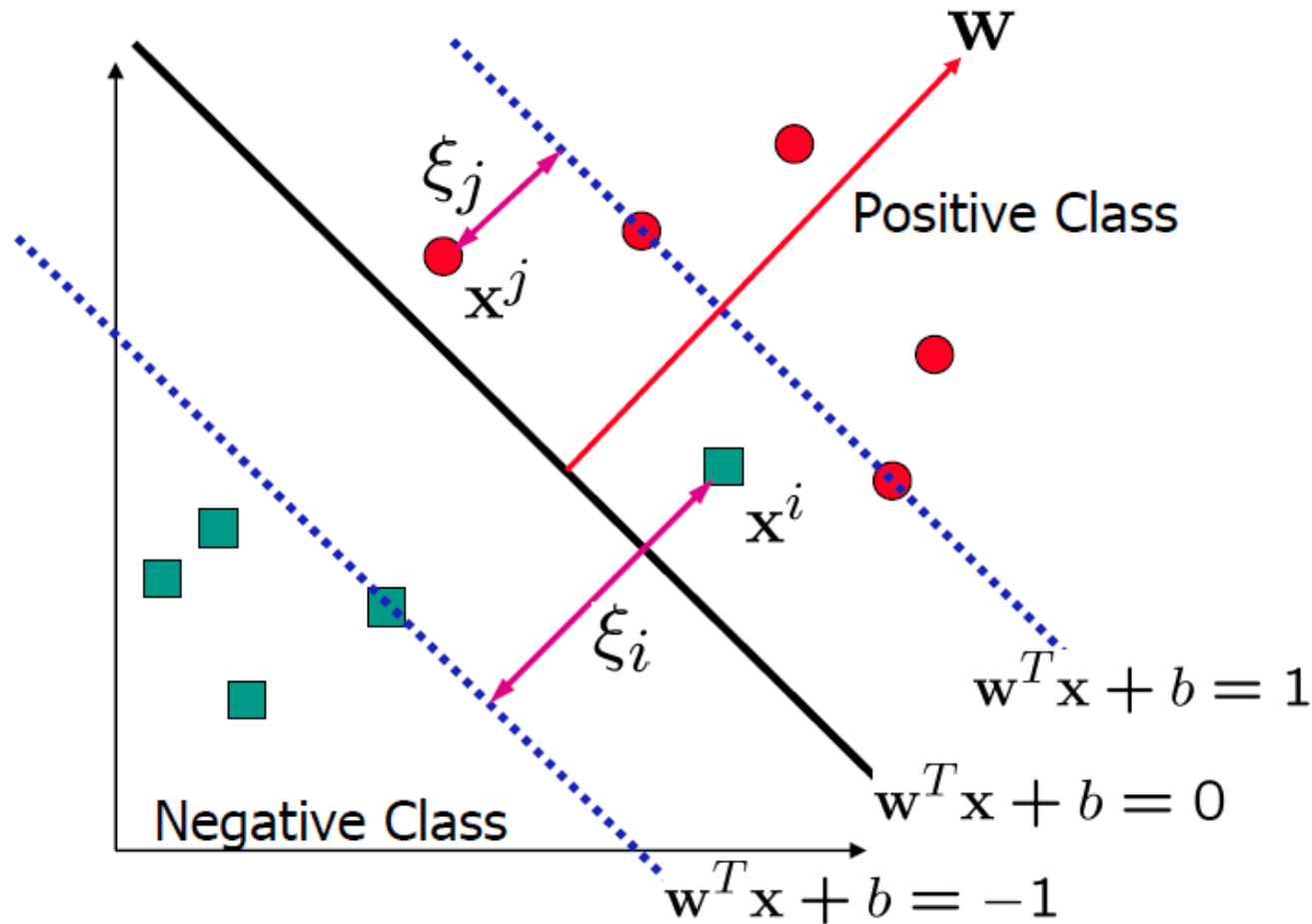
Non-Separable data and Noise

- What if the data is not linearly separable?
- We may have noise in data, and maximum margin classifier is not robust to noise!



Hard Margin \rightarrow Soft Margin

- Allow functional margins to be less than 1
 - ▲ But we will charge a penalty



Soft Margin Maximization

Hard margin

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to: $y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N$



Soft margin

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{i=1}^N \xi_i$$

subject to: $y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N$
 $\xi_i \geq 0, \quad i = 1, \dots, N$

- Introduce **slack variables** ξ_i to allow functional margins to be smaller than 1

C Parameter

- Trades-off two objectives
 - ▲ Maximize margin (generalization behavior of the classifier on unseen data)
 - ▲ Minimize the total amount of slack penalty (empirical error or fit to the data)
- How do I pick the best C value in practice?
 - ▲ Select models or hyper-parameters is called “cross-validation”
 - ▲ You divide your training data into two parts: sub-train (70%) and validation (30%)
 - ▲ Infinite choices for C? => AutoML (BO)

Soft Margin Maximization

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{i=1}^N \xi_i$$

subject to : $y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N$

$$\xi_i \geq 0, \quad i = 1, \dots, N$$

- Parameter C controls the tradeoff between maximizing the margin (generalization error) and fitting the training examples (training error)

Dual Formulation of Soft Margin

$$\max \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y^i y^j \langle \mathbf{x}^i \cdot \mathbf{x}^j \rangle$$

Subject to: $\sum_{i=1}^N \alpha_i y^i = 0$

$$0 \leq \alpha_i \leq C \quad i=1, \dots, N$$

- The dual problem is almost identical to the separable case, except for that α_i 's are now bounded by C (tradeoff parameter)
- C puts a box constraint on α_i 's, the weights of the support vectors
- Limits the influence of outliers

Dual Formulation of Soft Margin

support vectors ($\alpha_i > 0$)

$$c > \alpha_i > 0: \quad y^i(w \cdot x^i + b) = 1, \text{ i.e., } \xi_i = 0$$

$$\alpha_i = c: \quad y^i(w \cdot x^i + b) \leq 1, \text{ i.e., } \xi_i \geq 0$$

- We now also have support vectors for data that have functional margin less than one (in addition to those that equal 1), but their α_i 's will only equal C
- Optimal weights can be computed as:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

Linear SVMs: Overview

- So far our classifier is a *separating hyperplane*
- Most “important” training points are support vectors; they define the hyperplane
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrange multipliers α_i

Summary of Last Lecture

- Hard-Margin SVMs for linearly separable data
 - ▲ Characteristics of the dual solution
 - ▲ Weight vector is determined by a small no. of training examples called Support Vectors
- Soft-Margin SVMs
 - ▲ To deal with non-separable and noisy data
 - ▲ Relax the margin requirement by introducing slack variables
 - ▲ C parameter trades-off the training error (sum of slacks) and the generalization error (margin)

Linear SVMs: Overview

- For both training and classification, we see training data appear only inside inner products

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y^i y^j \langle \mathbf{x}^i \cdot \mathbf{x}^j \rangle$ is maximized
and

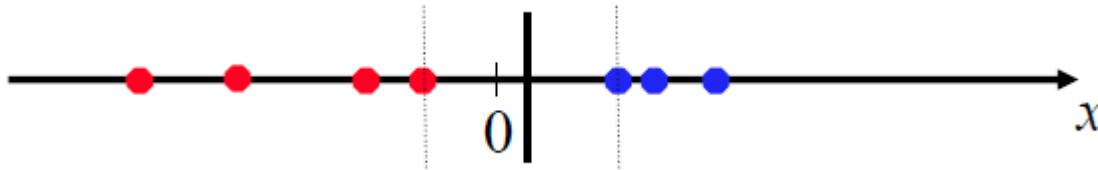
(1) $\sum \alpha_i y^i = 0$

(2) $0 \leq \alpha_i \leq c$ for all α_i

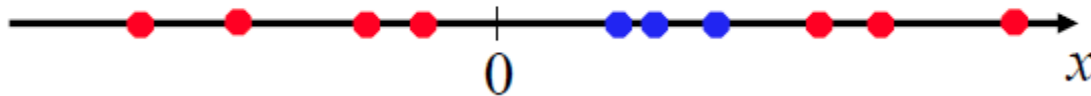
$$f(\mathbf{x}) = \sum \alpha_i y^i \langle \mathbf{x}^i \cdot \mathbf{x} \rangle + b$$

Non-Linear SVMs

- Datasets that are linearly separable with some noise work out great

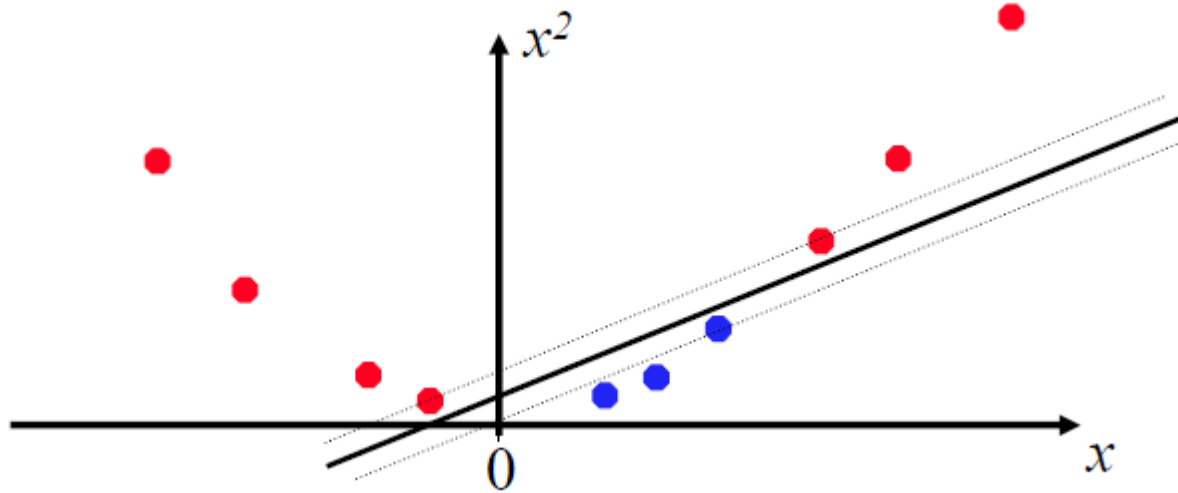


- But what are we going to do if the dataset is just too hard?



Non-Linear SVMs

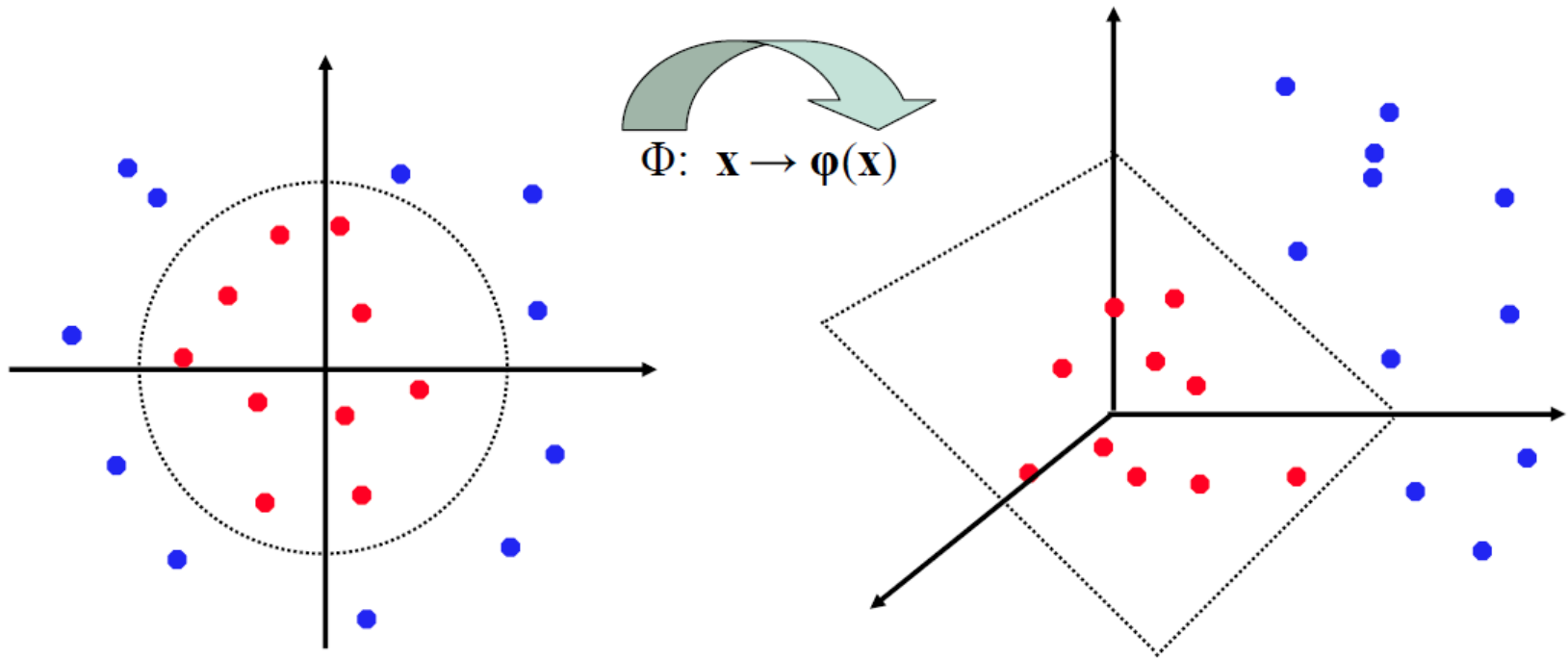
- How about mapping the data to a higher dimensional space?



Non-Linear SVMs: Feature Spaces

- **General idea**

- ▲ For any data set, the original feature space can always be mapped to some higher-dimensional feature space such that the data is linearly separable



Example: Quadratic Space

- Assume m input dimensions
 - ▲ $x = (x_1, x_2, \dots, x_m)$
- The number of dimensions increase rapidly
 - ▲ Expensive to compute!
- You may be wondering about the $\sqrt{2}'s$
 - ▲ You will find out soon!

The diagram illustrates the quadratic feature map $\Phi(\mathbf{x})$ as a column vector. The terms are grouped into four categories with colored brackets on the right:

- Constant Term** (red bracket): The first term, 1.
- Linear Terms** (green bracket): The next m terms, $\sqrt{2}x_1, \sqrt{2}x_2, \dots, \sqrt{2}x_m$.
- Pure Quadratic Terms** (purple bracket): The next m terms, $x_1^2, x_2^2, \dots, x_m^2$.
- Quadratic Cross-Terms** (blue bracket): The final terms, $\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}x_1x_m, \sqrt{2}x_2x_3, \dots, \sqrt{2}x_{m-1}x_m$. An arrow points from the text "the $\sqrt{2}'s$ " in the previous block to the $\sqrt{2}$ coefficient in the first cross-term.

The full expression is:

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{pmatrix}$$

Non-Linear Mappings

- How will you do training and testing of the classifier?
 - ▲ Transform the entire training data into PHI space
 - ▲ Train the classifier $F(\text{PHI}(x))$
 - ▲ Testing: first get $\text{PHI}(x)$ and then pass it to classifier
- 1. Computation
 - ▲ $\text{PHI}(x)$ for every training example
- 2. Memory
 - ▲ X vs. $\text{PHI}(x)$ the size is much larger
 - ▲ $M*N$ vs. $M^2 * N$ (M is the number of features and N is the number of training examples)

Non-Linear Mappings

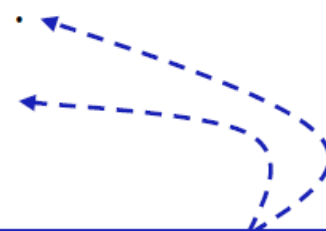
- How will you do training and testing of the classifier?
 - ▲ Transform the entire training data into PHI space
 - ▲ Train the classifier $F(\text{PHI}(x))$
 - ▲ Testing: first get $\text{PHI}(x)$ and then pass it to classifier
- Testing
 - ▲ You get X , then you compute $F(X) = O(M)$
 - ▲ $\text{PHI}(X) \Rightarrow O(M^2)$ --- Feature computation
 - ▲ $W.\text{PHI}(x) \Rightarrow O(M^2)$ – classifier decision computation
 - ▲ $O(M)$ vs. $O(2 * M^2)$

Kernel Functions

- The linear classifier relies on inner product between vectors: $K(x_i, x_j) = \langle x_i \cdot x_j \rangle$
- If every data point is mapped into high-dimensional space via some transformation $\Phi: x \rightarrow \phi(x)$, the inner product becomes $K(x_i, x_j) = \langle \phi(x_i) \cdot \phi(x_j) \rangle$
- A **kernel function** is a function that is equivalent to an inner product in some feature space
 - ▲ Example: $K(x_i, x_j) = (x_i \cdot x_j + 1)^2$
 - ▲ This is equivalent to mapping to the quadratic space!

Quadratic Kernel

- Consider a 2-d input space (generalizes to n-d)

$$\begin{aligned} K(\mathbf{x}^i, \mathbf{x}^j) &= (\mathbf{x}^i \cdot \mathbf{x}^j + 1)^2 \\ &= (x_1^i x_1^j + x_2^i x_2^j + 1)^2 \\ &= x_1^{i2} x_1^{j2} + 2x_1^i x_2^i x_1^j x_2^j + x_2^{i2} x_2^{j2} + 2x_1^i x_1^j + 2x_2^i x_2^j + 1 \\ &= (x_1^{i2}, \sqrt{2} x_1^i x_2^i, x_2^{i2}, \sqrt{2} x_1^i, \sqrt{2} x_2^i, 1) \cdot \\ &\quad (x_1^{j2}, \sqrt{2} x_1^j x_2^j, x_2^{j2}, \sqrt{2} x_1^j, \sqrt{2} x_2^j, 1) \\ &= \Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^j) \end{aligned}$$


nonlinear mapping of \mathbf{x}^i
and \mathbf{x}^j to quadratic space

- A kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\phi(x)$ explicitly)

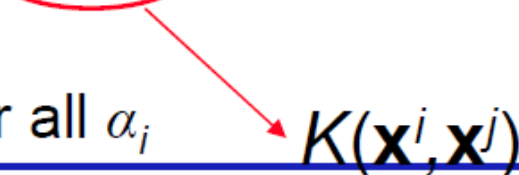
Quadratic Kernel

- A kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\phi(x)$ explicitly)
- Computing inner product of quadratic features is $O(m^2)$ time vs. $O(m)$ time for kernel

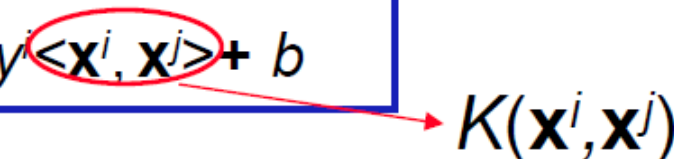
Non-Linear SVMs

- Dual problem formulation

Find $\alpha_1 \dots \alpha_N$ such that
 $\sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y^i y^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle$ is maximized and
(1) $\sum \alpha_i y^i = 0$
(2) $0 \leq \alpha_i \leq c$ for all α_i



- To classify a new point, we compute

$$f(\mathbf{x}) = \sum \alpha_i y^i \langle \mathbf{x}^i, \mathbf{x} \rangle + b$$


- Optimization techniques to find α_i 's remain the same
- This shows the utility of the dual formulation

Kernel Functions

- In practice, the user specifies the kernel function K , without explicitly stating the transformation $\phi(\cdot)$
- Given a kernel function, finding its corresponding transformation can be very cumbersome
 - ▲ This is why people only specify the kernel function without worrying about the exact transformation
- **Another view:** a kernel function computes some kind of measure of similarity between objects
- If you have a reasonable measure of similarity for your application, can we use it as the kernel in an SVM?

What functions are Kernels?

- Consider some finite set of m points, let matrix K be defined as follows

$K =$

$K(\mathbf{x}^1, \mathbf{x}^1)$	$K(\mathbf{x}^1, \mathbf{x}^2)$	$K(\mathbf{x}^1, \mathbf{x}^3)$...	$K(\mathbf{x}^1, \mathbf{x}^m)$
$K(\mathbf{x}^2, \mathbf{x}^1)$	$K(\mathbf{x}^2, \mathbf{x}^2)$	$K(\mathbf{x}^2, \mathbf{x}^3)$		$K(\mathbf{x}^2, \mathbf{x}^m)$
...
$K(\mathbf{x}^m, \mathbf{x}^1)$	$K(\mathbf{x}^m, \mathbf{x}^2)$	$K(\mathbf{x}^m, \mathbf{x}^3)$...	$K(\mathbf{x}^m, \mathbf{x}^m)$

- This is called **Kernel Matrix**
- Mercer's Theorem:
 - ▲ A function K is a kernel function if and only if its corresponding kernel matrix is symmetric and positive semi-definite

Examples of Kernel Functions

- Linear: $K(x_i, x_j) = \langle x_i, x_j \rangle$
 - ▲ Mapping: $\Phi: x \rightarrow \phi(x)$, such that $\phi(x) = x$
- Polynomial of power p : $K(x_i, x_j) = (1 + x_i \cdot x_j)^p$
 - ▲ Mapping: $\Phi: x \rightarrow \phi(x)$, where $\phi(x)$ corresponds to polynomial (p) mapping
- Gaussian (Radial Basis Function): $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$
 - ▲ Mapping: $\Phi: x \rightarrow \phi(x)$, where $\phi(x)$ has infinite dimensions; every point is mapped to a function (Gaussian)

Examples of Kernel Functions

- Gaussian (Radial Basis Function): $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$
 - ▲ Mapping: $\Phi: x \rightarrow \phi(x)$, where $\phi(x)$ has infinite dimensions; every point is mapped to a function (Gaussian)
 - ▲ RBF kernel values decreases with distance and ranges between zero (in the limit) and one ($x_i = x_j$)
- String Kernels
 - ▲ No. of common substrings of length k
- Graph Kernels
- Time-Series Kernels

Properties of Kernels

- Not all functions $K(x_i, x_j)$ are kernels!
- Conditions for a function to be a kernel
 - ▲ **Symmetry**: $\forall x_i, x_j; K(x_i, x_j) = K(x_j, x_i)$
 - ▲ **Positivity**: for a set of m points x_1, x_2, \dots, x_m ; define the kernel matrix as $K_{ij} = K(x_i, x_j)$. For all m data points and all $m \times 1$ vectors t , $t^T K t \geq 0$
- These are necessary and sufficient conditions

Properties of Kernels

- How to show that a function $K(x_i, x_j)$ is a kernel?
 - ▲ Either find the feature map $\Phi: x \rightarrow \phi(x)$ such that $K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$ OR
 - ▲ Show that symmetry and positivity conditions hold
- How to show that a function $K(x_i, x_j)$ is not a kernel?
 - ▲ Show a counter-example for symmetry or positivity conditions

Summary and Recap of Last Lecture

- **Hard-Margin SVMs** for linearly separable data
 - ▲ Characteristics of the dual solution
 - ▲ Weight vector is determined by a small no. of training examples called Support Vectors
- **Soft-Margin SVMs**
 - ▲ To deal with non-separable and noisy data
 - ▲ Relax the margin requirement by introducing slack variables
 - ▲ **C** parameter trades-off the training error (sum of slacks) and the generalization error (margin)

Summary and Recap of Last Lecture

- **Non-linear SVMs via Kernel Trick**
 - ▲ Map data from low-dimensional space to high-dimensional space $\phi(x)$ to make the learning problem easier via linear SVMs
 - ▲ Kernel functions *implicitly* maps data to a high-dimensional space (without the need to compute each $\phi(x)$ explicitly)
- Kernels and Mercer's theorem
- Multi-class classification
 - ▲ One vs. one reduction
 - ▲ One vs. all reduction

Multi-Class SVMs

- **One-vs-One Reduction**

- ▶ Learn $k(k-1)/2$ binary classifiers
- ▶ Classify a new example via majority vote

- **One-vs-All Reduction**

- ▶ K weight vectors (Rep-I)
- ▶ Single weight vector (Rep-II)

Reducing Multi-Class to Binary

- **One vs. One**
- Three classes: red, blue, green
- 1. Red vs. Blue
- 2. Blue vs. Green
- 3. Red vs. Green
- Given a new example x , we get decisions from all three:
 - ▲ Red, Green, Red \Rightarrow Red is our prediction

Drawback of One vs. One

- Training time-complexity
 - ▲ K choose 2 number of classifiers, where K is the number of classes
 - ▲ $O(K^2)$ number of binary classifiers
- Testing time-complexity
 - ▲ $O(K^2)$

One vs. All

- Four classes: red, blue, green, yellow
- **One vs. One:** 4 choose 2 \Rightarrow 6
- **One vs. All:**
 - ▲ Red vs. Rest
 - ▲ Blue vs. Rest
 - ▲ Green vs. Rest
 - ▲ Yellow vs. Rest
 - ▲ 4 Classifiers
 - ▲ Given a new example: pick the class that gets the highest score

One vs. All

- One weight vector for each class
 - ▲ W_1 (“d” weights of class 1)
 - ▲ W_2 (“d” weights of class 2)
 - ▲ ...
- Single weight vector for all classes
 - ▲ W (“k*d” number of weights)
 - ▲ $W = [W_1, W_2, \dots, W_K]$
- Score of class “r”
 - ▲ $W_r.X \Rightarrow$ in the first representation
 - ▲ $W.F(X, r) \Rightarrow$ in the second representation

Multi-class Classification

- **Single prototype**

$$W = \begin{matrix} & w_1 & & w_2 & & w_3 & & w_K \\ \begin{bmatrix} & & & & & & & \end{bmatrix} & \begin{bmatrix} & & & & & & & \end{bmatrix} & \begin{bmatrix} & & & & & & & \end{bmatrix} & \dots & \begin{bmatrix} & & & & & & & \end{bmatrix} \end{matrix}$$

- ▶ $K * D$ parameters

- $\varphi(x, y) = (\llbracket y = 1 \rrbracket x, \llbracket y = 2 \rrbracket x, \dots, \llbracket y = k \rrbracket x)$

- ▶ $\varphi(x, 2)$

$$\begin{bmatrix} 0 & & & & & & & \end{bmatrix} \begin{bmatrix} & x & & & & & & \end{bmatrix} \begin{bmatrix} 0 & & & & & & & \end{bmatrix} \dots \begin{bmatrix} & & & & 0 & & & \end{bmatrix}$$

Multi-Class SVM

- Multi-class SVM [Crammer and Singer 2001]

$$\min_w \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \delta_i$$

$$s.t: \quad w \cdot \varphi(x_i, y_i) - w \cdot \varphi(x_i, y) \geq 1 - \delta_i$$

$$\forall y \in Y \setminus \{y_i\}, \forall i = 1, \dots, n$$

Critical Steps for using SVM

- Select the kernel function to use (important but often trickiest part of SVM).
- In practice, try the following in the same order
 - ▲ linear kernel
 - ▲ low degree polynomial kernel
 - ▲ RBF kernel with a reasonable width σ
 - ▲ Supported by off-the-shelf software (e.g., LibSVM or SVM-Light)

-

Critical Steps for using SVM

- Select the value of tradeoff parameter C and the parameter of the kernel function
 - ▲ Try the values suggested by the SVM software
 - ▲ $C = \{0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000\}$
 - ▲ You can set apart a **validation set** to determine the values of the parameter
- SVM Software
 - ▲ LibSVM -- <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
 - ▲ SVM-Light -- http://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html

AutoML

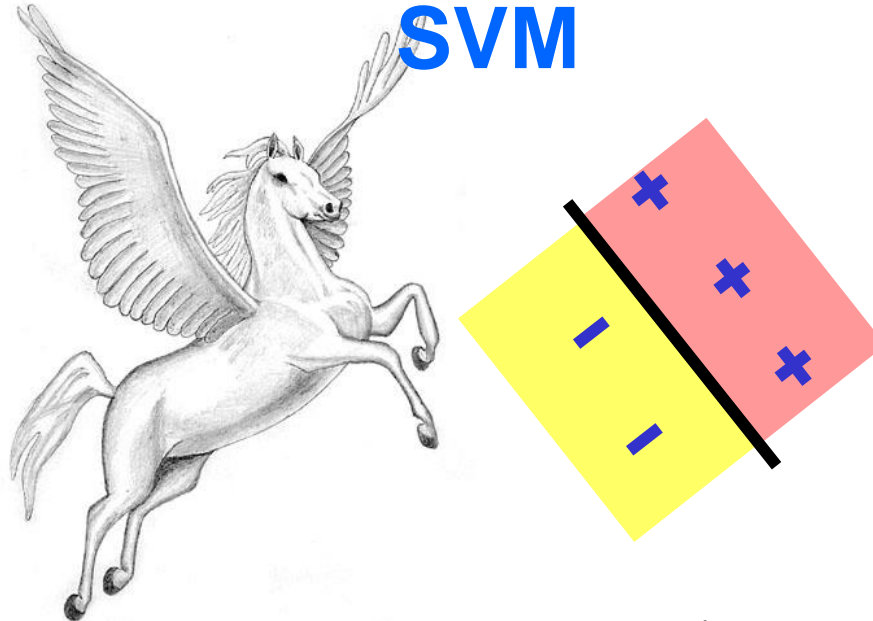
- Sub-area of ML that studies methods for automated machine learning
- Automate hyper-parameter tuning
- Bayesian Optimization (This class)
- Bandits

SVMs Summary

- Advantages of SVMs
 - ▲ Polynomial time exact optimization
 - ▲ Kernels allow very flexible hypotheses (decision boundaries)
 - ▲ Can be applied to very complex data types, e.g., sequences, graphs
- Disadvantages of SVMs
 - ▲ Must choose a good kernel and kernel parameters
 - ▲ Scalability issues with very-large data sets
 - Recent work has made this much less an issue
 - Stream SVM: <http://www.ibis.t.u-tokyo.ac.jp/masin/streamsvm.html>
 - SVM-Perf: http://www.cs.cornell.edu/people/tj/svm_light/svm_perf.html

PEGASOS

Primal Efficient sub-GrAdient SOLver for
SVM



Shai Shalev-Shwartz

The Hebrew University
Jerusalem, Israel



Yoram Singer



Nati Srebro



Support Vector Machines

QP form:

$$\begin{aligned} \operatorname{argmin}_{\mathbf{w}, \xi_i \geq 0} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & \forall i, y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 - \xi_i \end{aligned}$$

More “natural” form:

$\operatorname{argmin}_{\mathbf{w}} f(\mathbf{w})$ where:

$$f(\mathbf{w}) \stackrel{\text{def}}{=} \underbrace{\frac{\lambda}{2} \|\mathbf{w}\|^2}_{\text{Regularization term}} + \underbrace{\frac{1}{m} \sum_{i=1}^m \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\}}_{\text{Empirical loss}}$$

Previous Work

- Dual-based methods
 - ▲ Interior Point methods
 - Memory: m^2 , time: $m^3 \log(\log(1/\epsilon))$
 - ▲ Decomposition methods
 - Memory: m , Time: super-linear in m
- Online learning & Stochastic Gradient
 - ▲ Memory: $O(1)$, Time: $1/\epsilon^2$ (linear kernel)
 - ▲ Memory: $1/\epsilon^2$, Time: $1/\epsilon^4$ (non-linear kernel)

Typically, online learning algorithms do not converge to the optimal solution of SVM

PEGASOS

$$A_t = S$$

Subgradient method

$$|A_t| = 1$$

Stochastic gradient

Number of iterations T

INITIALIZE. Choose \mathbf{w}_1 s.t. $\|\mathbf{w}_1\| \leq 1/\sqrt{\lambda}$

FOR $t = 1, 2, \dots, T$

Choose $A_t \subseteq S$

$$A_t^+ = \{(\mathbf{x}, y) \in A_t : y \langle \mathbf{w}_t, \mathbf{x} \rangle < 1\}$$

$$\nabla_t = \lambda \mathbf{w}_t - \frac{\eta_t}{|A_t|} \sum_{(\mathbf{x}, y) \in A_t^+} y \mathbf{x}$$

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

$$\mathbf{w}'_t = \mathbf{w}_t - \eta_t \nabla_t$$

$$\mathbf{w}_{t+1} = \min \left\{ 1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}'_t\|} \right\} \mathbf{w}'_t$$

OUTPUT: \mathbf{w}_{T+1}

Subgradient

Projection

Machine Learning Theory

- Probably Approximately Correct (PAC)
- Infinite bag of training examples (joint distr.)
- Experiment
 - ▶ You draw a sample of size (m) as training data
 - ▶ You learn a model/classifier from it
 - ▶ You measure its accuracy on the infinite bag
- To achieve an accuracy of “epsilon” with a confidence “delta”, how big should be my sample size “ m ”? Run in polynomial time.

Run-Time of Pegasos

- Choosing $|A_t|=1$ and a linear kernel over \mathbb{R}^n
→ Run-time required for Pegasos to find ε accurate solution w.p. $1-\delta$

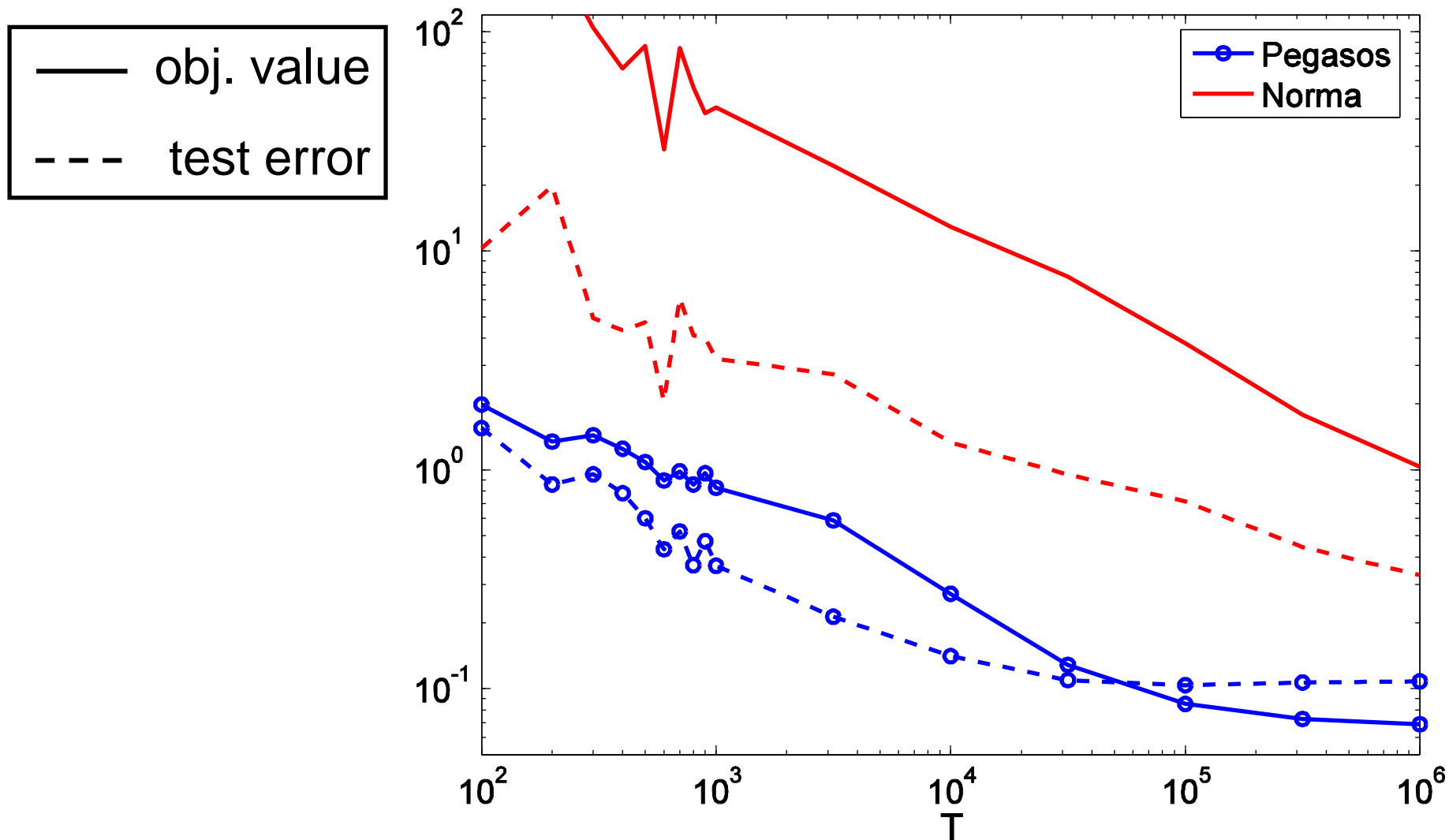
$\tilde{O}\left(\frac{d}{\lambda \varepsilon \delta}\right)$, where d is the sparsity bound

- Run-time does not depend on #examples
- Depends on “difficulty” of problem (λ and ε)

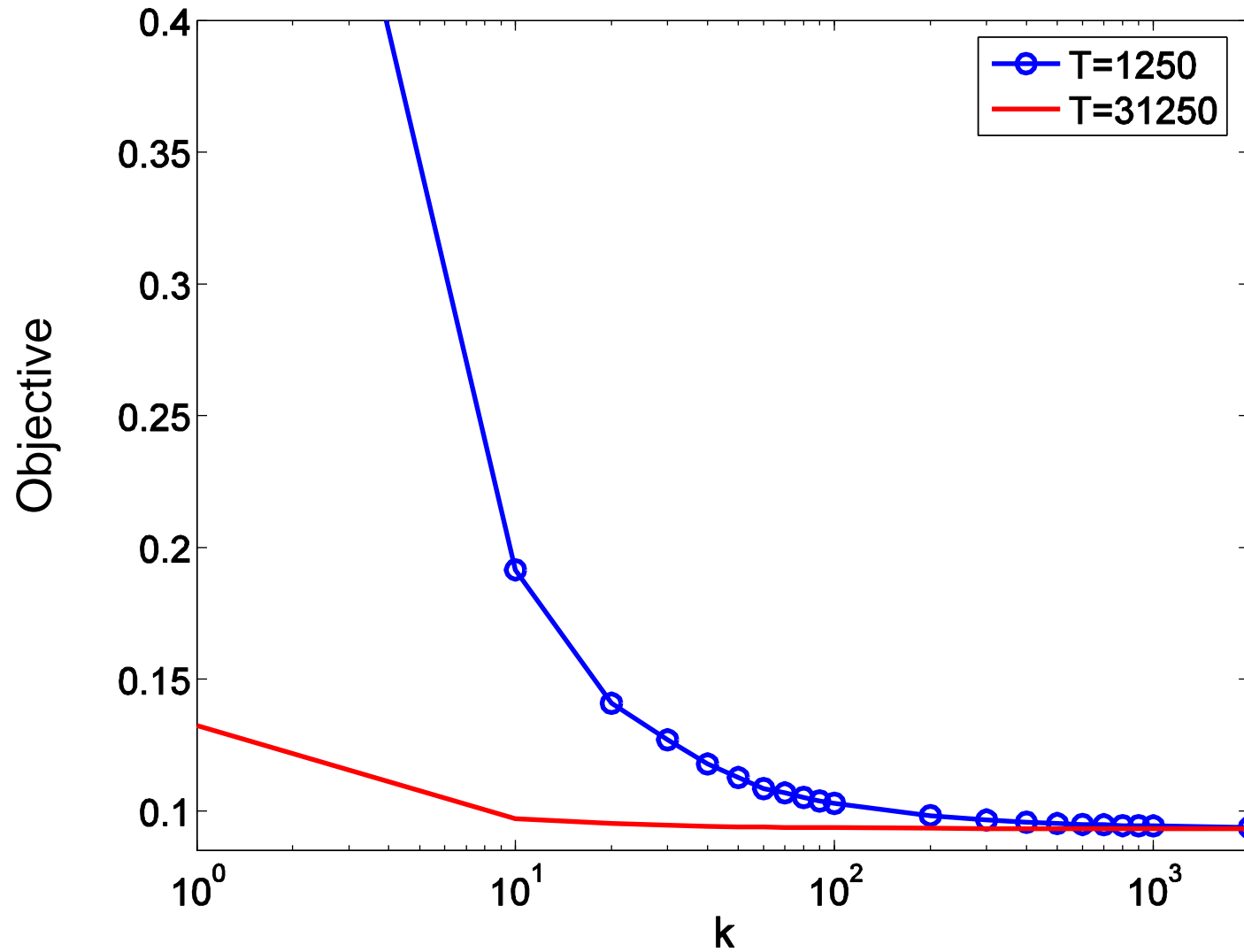
Training Time (in seconds)

	Pegasos	SVM-Perf	SVM-Light
Reuters	2	77	20,075
Covertypes	6	85	25,514
Astro-Physics	2	5	80

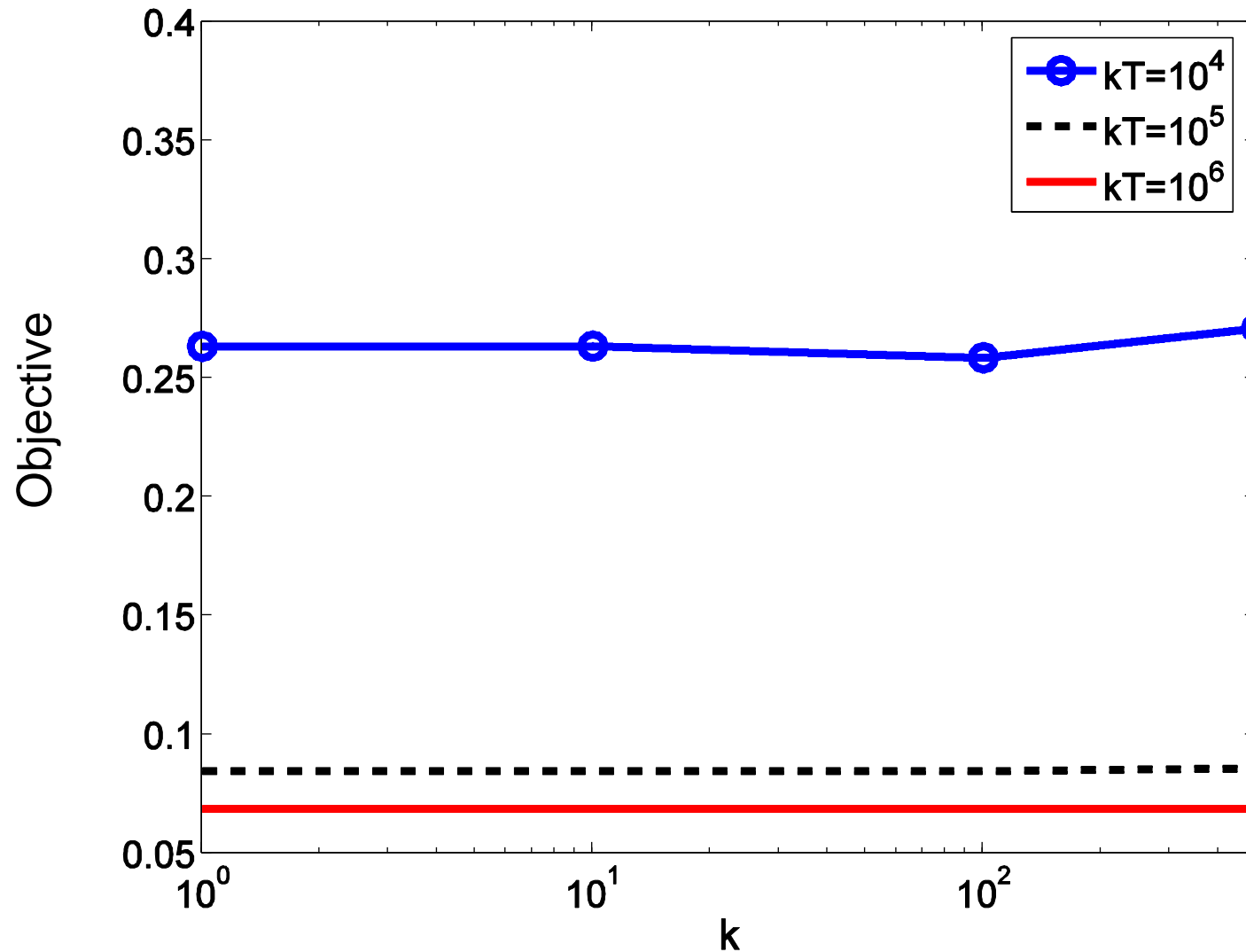
Compare to Norma (on Physics)



Effect of $k=|A_t|$ when T is fixed



Effect of $k=|A_t|$ when kT is fixed



Discussion

- **Pegasos**: Simple & Efficient solver for SVM
- **Sample vs. computational complexity**
 - ▲ Sample complexity: How many examples do we need as a function of VC-dim (λ), accuracy (ε), and confidence (δ)
 - ▲ In Pegasos, we aim at analyzing computational complexity based on λ , ε , and δ (also in Bottou & Bousquet)

Kernelizing Online Learning Algorithms

initialize $f = 0$ **Functional Form**
repeat
 Pick (x_i, y_i) from data
 if $y_i f(x_i) \leq 0$ **then**
 $f(\cdot) \leftarrow f(\cdot) + y_i k(x_i, \cdot)$
until $y_i f(x_i) > 0$ for all i

- Nothing happens if classified correctly
- Weight vector is a linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
- Classifier is a linear combination of inner products

$$f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x)$$

Kernelized Perceptron

Primal Form

update **weights**

$$w \leftarrow w + y_i \phi(x_i)$$

classify

$$f(k) = w \cdot \phi(x)$$

Dual Form

update **linear coefficients**

$$\alpha_i \leftarrow \alpha_i + y_i$$

implicitly equivalent to:

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$



- Nothing happens if classified correctly
- Weight vector is a linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
- Classifier is a linear combination of inner products

$$f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x)$$

Kernelized Perceptron: Example

- $(x_1, +1); (x_2, -1); (x_3, -1); (x_4, +1)$
- First iteration
 - ▲ Make prediction: $F(x_1) = \sum_{i=1}^4 \alpha_i y_i K(x_i, x_1) = -1.5$
 - ▲ -1
 - ▲ Update weights: $\text{Alpha1} = \text{Alpha1} + 1 = 1$
- Second iteration
 - ▲ Make prediction: $F(x_2) = \sum_{i=1}^4 \alpha_i y_i K(x_i, x_2) = 2.0$
 - ▲ +1
 - ▲ Update weights: $\text{Alpha2} = \text{Alpha2} - 1 = -1$

Questions?