UVA CS 6316 – Fall 2015 Graduate: Machine Learning

Lecture 9: Support Vector Machine (Cont. Revised Advanced Version)

Dr. Yanjun Qi
University of Virginia
Department of Computer Science

ATT: there exist some inconsistency of math notations across slides. Notations in each slide are mostly self-consistent.

Where we are ? ➔ Five major sections of this course

- ~~Regression (supervised)~~
- Classification (supervised)
- Unsupervised models
- Learning theory
- Graphical models

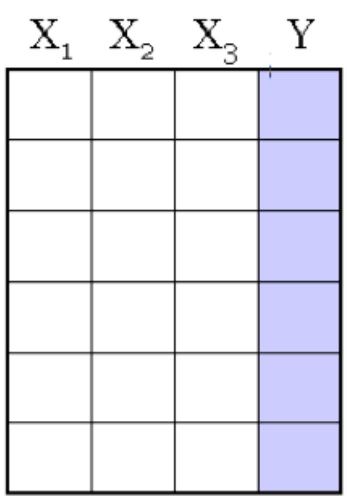
Where we are ? ➔ Three major sections for classification

- We can divide the large variety of classification approaches into roughly three major types

1. Discriminative
   - directly estimate a decision rule/boundary
   - e.g., support vector machine, decision tree

2. Generative:
   - build a generative statistical model
   - e.g., Bayesian networks

3. Instance based classifiers
   - Use observation directly (no models)
   - e.g. K nearest neighbors



A Dataset for binary classification

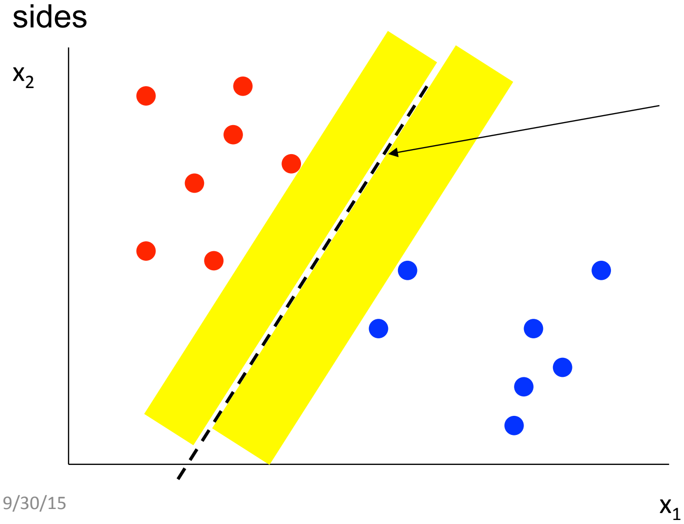
$$f : \boxed{X} \longrightarrow \boxed{Y}$$

Output as Binary Class Label: 1 or -1

- **Data**/*points/instances/examples/samples/records*: [rows]
- **Features**/*attributes/dimensions/independent variables/covariates/predictors/regressors*: [columns, except the last]
- **Target**/*outcome/response/label/dependent variable*: special column to be predicted [last column]

Max margin classifiers

- Instead of fitting all points, focus on boundary points
- Learn a boundary that leads to the largest margin from points on both sides

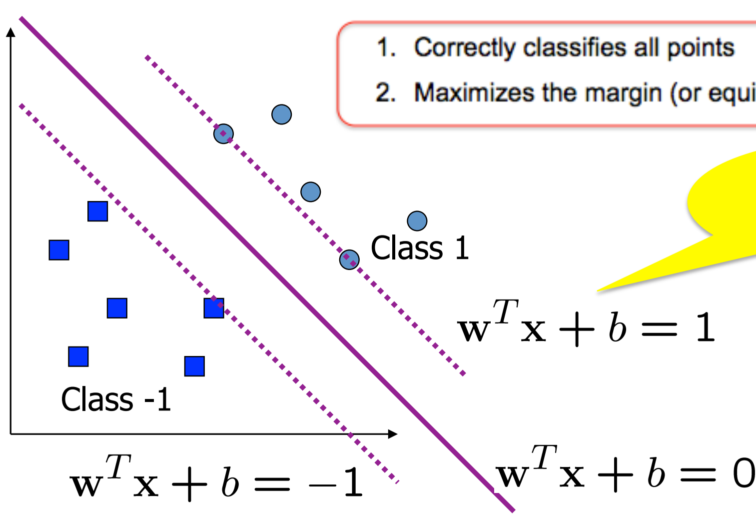


Why?

- Intuitive, 'makes sense'
- Some theoretical support
- Works well in practice

When linearly Separable Case

- The decision boundary should be as far away from the data of both classes as possible



1. Correctly classifies all points
2. Maximizes the margin (or equivalently minimizes $w^T w$)

W is a p-dim vector; b is a scalar

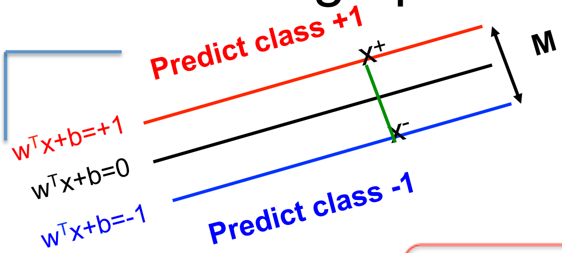
Today

- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Today

- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Optimization Step
i.e. learning optimal parameter for SVM



$$M = \frac{2}{\sqrt{w^T w}}$$

1. Correctly classifies all points
2. Maximizes the margin (or equivalently minimizes w^Tw)

Min $(w^Tw)/2$

subject to the following constraints:

For all x in class + 1

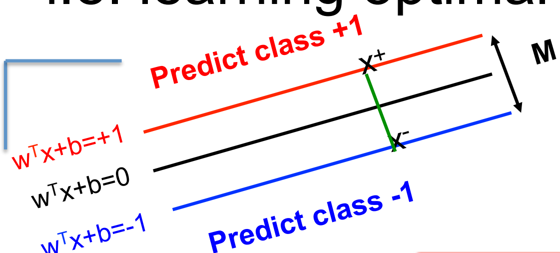
$w^Tx+b >= 1$ $y_i = 1$

For all x in class - 1

$w^Tx+b <= -1$ $y_i = -1$

A total of n constraints if we have n input samples

Optimization Step
i.e. learning optimal parameter for SVM



$$M = \frac{2}{\sqrt{w^T w}}$$

1. Correctly classifies all points
2. Maximizes the margin (or equivalently minimizes w^Tw)

Min $(w^Tw)/2$

subject to the following constraints:

For all x in class + 1

$w^Tx+b >= 1$

For all x in class - 1

$w^Tx+b <= -1$

A total of n constraints if we have n input samples

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{i=1}^{p} w_i^2$$

subject to $\forall \mathbf{x}_i \in Dtrain : y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1$

Optimization Review: Ingredients

- Objective function
- Variables
- Constraints

Find values of the variables that minimize or maximize the objective function while satisfying the constraints

Optimization with Quadratic programming (QP)

Quadratic programming solves optimization problems of the following form:

$$\min_U \frac{u^T R u}{2} + d^T u + c$$

subject to n inequality constraints:

$$a_{11}u_1 + a_{12}u_2 + \ldots \leq b_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$a_{n1}u_1 + a_{n2}u_2 + \ldots \leq b_n$$

and k equivalency constraints:

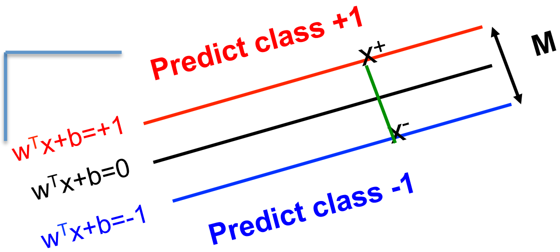
$$a_{n+1,1}u_1 + a_{n+1,2}u_2 + \ldots = b_{n+1}$$
$$\vdots \quad \vdots \quad \vdots$$
$$a_{n+k,1}u_1 + a_{n+k,2}u_2 + \ldots = b_{n+k}$$

Quadratic term

When a problem can be specified as a QP problem we can use solvers that are better than gradient descent or simulated annealing

SVM as a QP problem

R as I matrix, d as zero vector, c as 0 value



$$M = \frac{2}{\sqrt{w^T w}}$$

$$\min_U \frac{u^T R u}{2} + d^T u + c$$

subject to n inequality constraints:

$$a_{11}u_1 + a_{12}u_2 + \ldots \le b_1$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$a_{n1}u_1 + a_{n2}u_2 + \ldots \le b_n$$

and k equivalency constraints:

$$a_{n+1,1}u_1 + a_{n+1,2}u_2 + \ldots = b_{n+1}$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$a_{n+k,1}u_1 + a_{n+k,2}u_2 + \ldots = b_{n+k}$$

Min $(w^Tw)/2$

subject to the following inequality constraints:

For all x in class + 1

$w^Tx+b >= 1$

For all x in class - 1

$w^Tx+b <= -1$

A total of n constraints if we have n input samples

Today

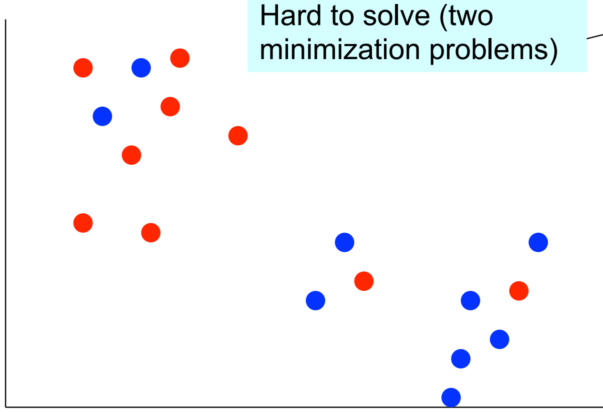
- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Non linearly separable case

- So far we assumed that a linear plane can perfectly separate the points
- But this is not usally the case

 - noise, outliers

Hard to solve (two minimization problems)



How can we convert this to a QP problem?

- Minimize training errors?

 min w^Tw

 min #errors

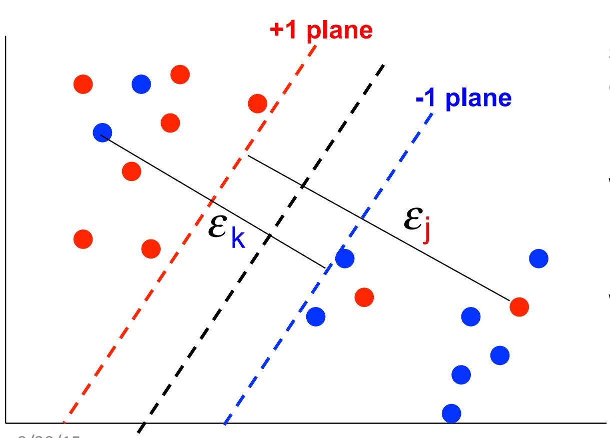
- Penalize training errors:

 min w^Tw+C*(#errors)

Hard to encode in a QP problem

Non linearly separable case

- Instead of minimizing the number of misclassified points we can minimize the ***distance*** between these points and their correct plane



The new optimization problem is:

$$\min_w \frac{w^Tw}{2} + \sum_{i=1}^{n} C\varepsilon_i$$

subject to the following inequality constraints:

For all x_i in class + 1

$w^Tx_i+b >= 1-\varepsilon_i$

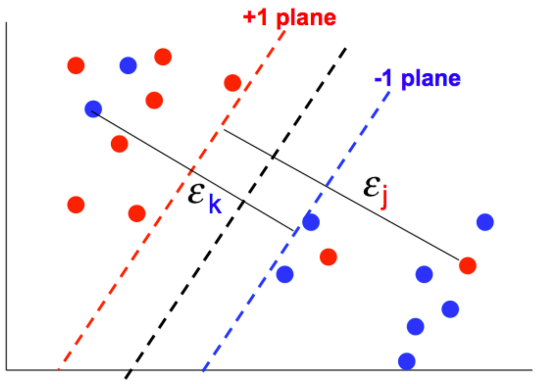
For all x_i in class - 1

$w^Tx_i+b <= -1+\varepsilon_i$

$w: p$
$b: 1$
$\varepsilon_i: n$

Wait. Are we missing something?

Final optimization for non linearly separable case



The new optimization problem is:

$$\min_w \frac{w^T w}{2} + \sum_{i=1}^{n} C\varepsilon_i$$ hyperpara

subject to the following inequality constraints:

For all x_i in class + 1

$w^T x_i + b \geq 1 - \varepsilon_i$

For all x_i in class - 1

$w^T x_i + b \leq -1 + \varepsilon_i$

A total of n constraints

For all i

$$\varepsilon_i \geq 0$$

Another n constraints

Where we are

Two optimization problems: For the separable and non separable cases

$$\min_w \frac{w^T w}{2}$$

For all x in class + 1

$w^T x + b \geq 1$

For all x in class - 1

$w^T x + b \leq -1$

$$\min_w \frac{w^T w}{2} + \sum_{i=1}^{n} C\varepsilon_i$$

For all x_i in class + 1

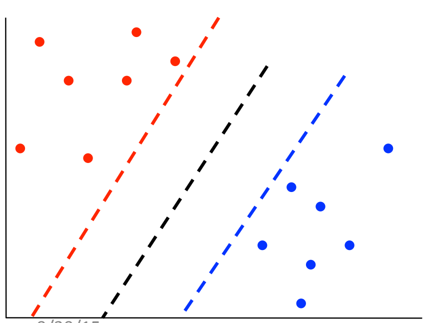
$w^T x_i + b \geq 1 - \varepsilon_i$

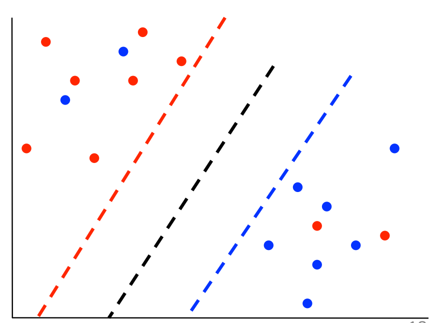
For all x_i in class - 1

$w^T x_i + b \leq -1 + \varepsilon_i$

For all i

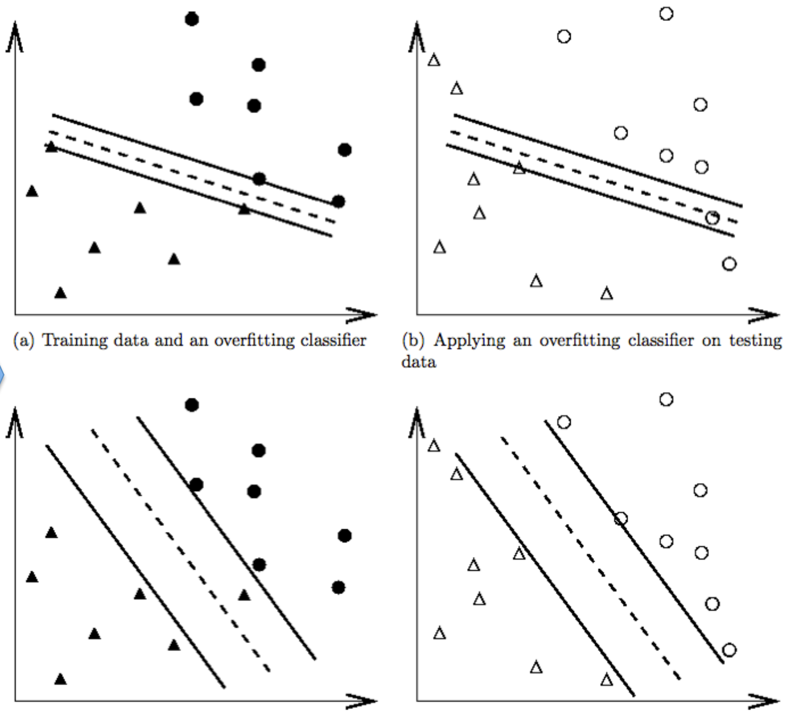
$$\varepsilon_i \geq 0$$





Model Selection, find right C

Select the right penalty parameter C



(a) Training data and an overfitting classifier

(b) Applying an overfitting classifier on testing data

(c) Training data and a better classifier

(d) Applying a better classifier on testing data

Today

- ❑ Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Where we are

Two optimization problems: For the separable and non separable cases

Min $(w^Tw)/2$

For all x in class + 1

$w^Tx+b >= 1$

For all x in class - 1

$w^Tx+b <=-1$

$$\min_w \frac{w^Tw}{2} + \sum_{i=1}^{n} C\varepsilon_i$$

For all x_i in class + 1

$w^Tx_i+b >= 1-\varepsilon_i$

For all x_i in class - 1

$w^Tx_i+b <= -1+\varepsilon_i$

For all i

$$\varepsilon_i \geq 0$$

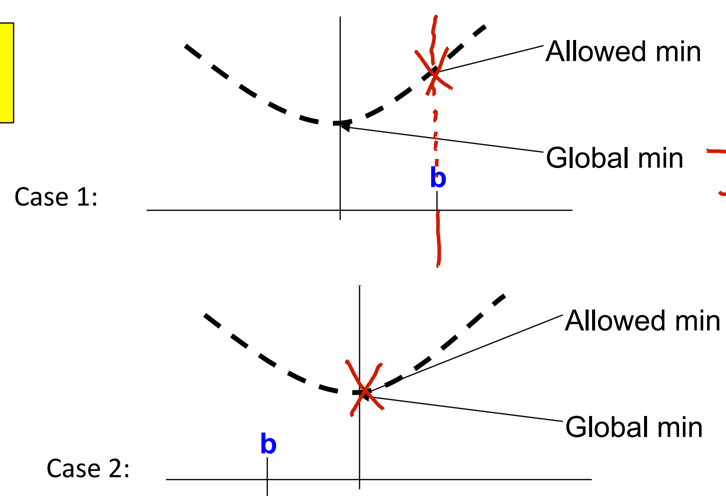
- Instead of solving these QPs directly we will solve a dual formulation of the SVM optimization problem
- The main reason for switching to this type of representation is that it would allow us to use a neat trick that will make our lives easier (and the run time faster)

Optimization Review: Constrained Optimization

$f(u)$

$\min_u u^2$

s.t. $u >= b$



$b>0$

$f(u)=b^2$

$b<0$

$f(u)=0$

Optimization Review: Constrained Optimization with Lagrange

- When equal constraints
- ➔ optimize *f(x)*, subject to $g_i(x)=0$

- Method of Lagrange multipliers: convert to a higher-dimensional problem
- Minimize

$(x_1, x_2, \ldots, x_n)$

n

(w, b)

$$f(x) + \sum \lambda_i g_i(x)$$

α

- w.r.t. $(x_1 \ldots x_n; \lambda_1 \ldots \lambda_k)$

$n+k$

Introducing a Lagrange multiplier for each constraint
Construct the Lagrangian for the original optimization problem

Optimization Review: Lagrangian (more general standard form)

standard form problem (not necessarily convex)

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \le 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

variable $x \in \mathbf{R}^n$, domain $\mathcal{D}$, optimal value $p^\star$

Lagrangian: $L : \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^p \to \mathbf{R}$, with $\mathbf{dom}\, L = \mathcal{D} \times \mathbf{R}^m \times \mathbf{R}^p$,

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

- weighted sum of objective and constraint functions
- λ_i is Lagrange multiplier associated with $f_i(x) \le 0$
- ν_i is Lagrange multiplier associated with $h_i(x) = 0$

From Stanford "Convex Optimization — Boyd & Vandenberghe

$\min_u u^2$

s.t. u >= b

① $\begin{cases} \min\limits_u f_0(u) = u^2 \\ \text{s.t. } \; b - u \le 0 \end{cases}$

② $L(\underset{1\times 1}{u}, \underset{1\times 1}{\alpha}) = u^2 + \underbrace{\alpha}_{\ge 0}\underbrace{(b-u)}_{\le 0}$

③ $\dfrac{\partial L(u,\alpha)}{\partial u} = 2u - \alpha = 0$

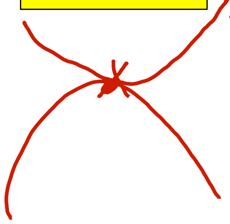
$u = \dfrac{\alpha}{2}$

$\min_u u^2$

s.t. u >= b

$g(\alpha) = L(u, \alpha) = \dfrac{\alpha^2}{4} + \alpha\left(b - \dfrac{\alpha}{2}\right)$

$u = \alpha/2$



$g(\alpha) = -\dfrac{\alpha^2}{4} + b\alpha$

$\dfrac{\partial g(\alpha)}{\partial \alpha} = -\dfrac{\alpha}{2} + b = 0, \quad \alpha \ge 0$

$\Rightarrow$ Dual $\begin{cases} b > 0, & \alpha = 2b, & g(\alpha) = b^2 \\ b < 0, & \alpha = 0, & g(\alpha) = 0 \end{cases}$

$\Rightarrow$ Primal $\begin{cases} b > 0, & f(u) = b^2 \\ b < 0, & f(u) = 0 \end{cases}$

Optimization Review: Lagrangian Duality

- The Primal Problem

Primal:

$$\begin{aligned} \min_w \quad & f_0(w) \\ \text{s.t.} \quad & f_i(w) \le 0, \quad i = 1, \ldots, k \\ & h_i(w) = 0, \quad i = 1, \ldots, l \end{aligned}$$

The generalized Lagrangian:

$$\mathcal{L}(w, \alpha, \beta) = f_0(w) + \sum_{i=1}^{k} \alpha_i f_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

the *a*'s ($a_i \ge 0$) and *b*'s are called the Lagarangian multipliers

Lemma:

$$\max_{\alpha, \beta, \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta) = \begin{cases} f_0(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{o/w} \end{cases}$$

A re-written Primal:

$$\min_w \max_{\alpha, \beta, \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta)$$

Optimization Review: Dual Func

Recall that Lagrange multipliers can be applied to turn the following problem:

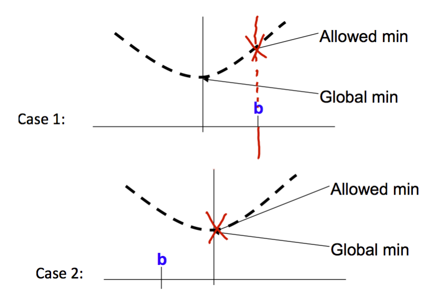
$\min_x x^2$

s.t. $x \ge b$ → $b - x \le 0$

To

$L_{x,\alpha}\ x^2 + \alpha(b-x)$ ← $\min_x \max_\alpha x^2 - \alpha(x-b)$

s.t. $\alpha \ge 0$



Optimization Review: Lagrangian Duality, cont.

- Recall the Primal Problem:

$$\min_w \max_{\alpha,\beta,\alpha_i \geq 0} \mathcal{L}(w,\alpha,\beta)$$

- The Dual Problem:

$$\max_{\alpha,\beta,\alpha_i \geq 0} \min_w \mathcal{L}(w,\alpha,\beta)$$

- **Theorem (weak duality):**

$$d^* = \max_{\alpha,\beta,\alpha_i \geq 0} \min_w \mathcal{L}(w,\alpha,\beta) \;\leq\; \min_w \max_{\alpha,\beta,\alpha_i \geq 0} \mathcal{L}(w,\alpha,\beta) = p^*$$

- **Theorem (strong duality):**

Iff there exist a saddle point of $\mathcal{L}(w,\alpha,\beta)$, we have $d^* = p^*$

Optimization Review: Lagrange dual function

Lagrange dual function: $g : \mathbf{R}^m \times \mathbf{R}^p \to \mathbf{R}$,

$$g(\lambda,\nu) = \inf_{x\in\mathcal{D}} L(x,\lambda,\nu)$$

$$= \underbrace{\inf_{x\in\mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \underbrace{\lambda_i}_{\geq 0} \underbrace{f_i(x)}_{\leq 0} + \sum_{i=1}^{p} \nu_i \underbrace{h_i(x)}_{0} \right)}_{L(x,\lambda,\nu)}$$

g is concave, can be $-\infty$ for some λ, ν

lower bound property: if $\lambda \succeq 0$, then $g(\lambda,\nu) \leq p^\star$

Inf(.): greatest lower bound

proof: if $\tilde{x}$ is feasible and $\lambda \succeq 0$, then

$$f_0(\tilde{x}) \geq L(\tilde{x},\lambda,\nu) \geq \inf_{x\in\mathcal{D}} L(x,\lambda,\nu) = g(\lambda,\nu)$$

minimizing over all feasible $\tilde{x}$ gives $p^\star \geq g(\lambda,\nu)$

An alternative representation of the SVM QP

- We will start with the linearly separable case
- Instead of encoding the correct classification rule and constraint we will use Lagrange multipliers to encode it as part of the our minimization problem

Min $(w^T w)/2$ ⇐ u^2

s.t.

$(w^T x_i + b) y_i >= 1$ ⇐ $x \geq b$

n constraints

⇓ $b - x \leq 0$

$\alpha_i (1 - (w^T x_i + b) y_i) \leq 0$

Recall that Lagrange multipliers can be applied to turn the following problem:

$$L_{primal} = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right)$$

$$\min_{w,b} \max_{\alpha} \frac{w^T w}{2} - \sum_i \alpha_i [(w^T x_i + b) y_i - 1]$$

$\alpha_i \geq 0 \qquad \forall i$

train

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w - \sum_i \alpha_i x_i y_i = 0$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum \alpha_i y_i = 0$$

The Dual Problem

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

- We minimize L with respect to w and b first:

$$\nabla_w L(w,b,\alpha) = w - \sum_{i=1}^{train} \alpha_i y_i x_i = 0, \quad (*)$$

$$\nabla_b L(w,b,\alpha) = \sum_{i=1}^{train} \alpha_i y_i = 0, \quad (**)$$

Note that (*) implies: $w = \sum_{i=1}^{train} \alpha_i y_i x_i \Rightarrow L(w,b,\alpha) \quad (***)$

- Plus (***) back to L, and using (**), we have:

$$L(w,b,\alpha) = \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i,j=1} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

Summary: Dual for SVM

Solving for **w** that gives maximum margin:

1. Combine objective function and constraints into new objective function, using Lagrange multipliers \alpha$_i$

$$L_{primal} = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i \left(y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right)$$

2. To minimize this Lagrangian, we take derivatives of **w** and *b* and set them to 0:

Summary: Dual for SVM

3. Substituting and rearranging gives the dual of the Lagrangian:

$$L_{dual} = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

which we try to maximize (not minimize).

4. Once we have the *alpha*$_i$, we can substitute into previous equations to get **w** and *b*.
5. This defines **w** and *b* as linear combinations of the training data.

Optimization Review: Dual Problem

- Solving dual problem if the dual form is easier than primal form
- Need to change primal minimization to dual maximization (OR → Need to change primal maximization to dual minimization)
- Only valid when the original optimization problem is convex/ concave (strong duality)

Primal Problem, e.g.,

$$x^* = \arg\min_x f(x)$$

$$\text{subject to } \mathrm{h}(x) = c$$

Strong duality

Dual Problem, e.g.,

$$\lambda^* = \arg\max_\lambda g(\lambda)$$

$$g(\lambda) = \inf_x (f(x) + \lambda(h(x) - c))$$

Summary: Dual SVM for linearly separable case

Substituting w into our target function and using the additional constraint we get:

Dual formulation $n \ \alpha_i$

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

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \qquad \forall i$$

Min $(w^T w)/2$

subject to the following inequality constraints:

For all x in class + 1

$w^T x + b >= 1$

For all x in class - 1

$w^T x + b <= -1$

A total of n constraints if we have n input samples

Easier than original QP, more efficient algorithms exist to find a_i

Optimization Review:

Complementary slackness

assume strong duality holds, $x^\star$ is primal optimal, $(\lambda^\star, \nu^\star)$ is dual optimal

inf (.): greatest lower bound

$$\begin{aligned} f_0(x^\star) = g(\lambda^\star, \nu^\star) &= \inf_x \left(f_0(x) + \sum_{i=1}^m \lambda_i^\star f_i(x) + \sum_{i=1}^p \nu_i^\star h_i(x) \right) \\ &\leq f_0(x^\star) + \sum_{i=1}^m \lambda_i^\star f_i(x^\star) + \underbrace{\sum_{i=1}^p \nu_i^\star h_i(x^\star)}_{=0} \\ &\leq f_0(x^\star) \end{aligned}$$

obj ⇒ $f(u^*)$ $\quad g(\alpha^*)$

$u^* \begin{cases} b>0 & u^* = b \\ b<0 & u^* = 0 \end{cases}$ $\quad \alpha^* = 2b$, $\alpha^* = 0$

$\lambda_i^\star \geq 0$, $f_i(x^\star) \leq 0$

hence, the two inequalities hold with equality

$\Rightarrow \alpha_i (1 - (w^T x_i + b) y_i)$

① $\alpha_i = 0$

② $\alpha_i > 0$

- $x^\star$ minimizes $L(x, \lambda^\star, \nu^\star)$
- $\lambda_i^\star f_i(x^\star) = 0$ for $i = 1, \ldots, m$ (known as complementary slackness):

$$\lambda_i^\star > 0 \Longrightarrow f_i(x^\star) = 0, \qquad f_i(x^\star) < 0 \Longrightarrow \lambda_i^\star = 0$$

Optimization Review:

Karush-Kuhn-Tucker (KKT) conditions

the following four conditions are called KKT conditions (for a problem with differentiable f_i, h_i):

1. primal constraints: $f_i(x) \leq 0$, $i = 1, \ldots, m$, $h_i(x) = 0$, $i = 1, \ldots, p$
2. dual constraints: $\lambda \succeq 0$
3. complementary slackness: $\lambda_i f_i(x) = 0$, $i = 1, \ldots, m$
4. gradient of Lagrangian with respect to x vanishes:

$$\nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} \nu_i \nabla h_i(x) = 0$$

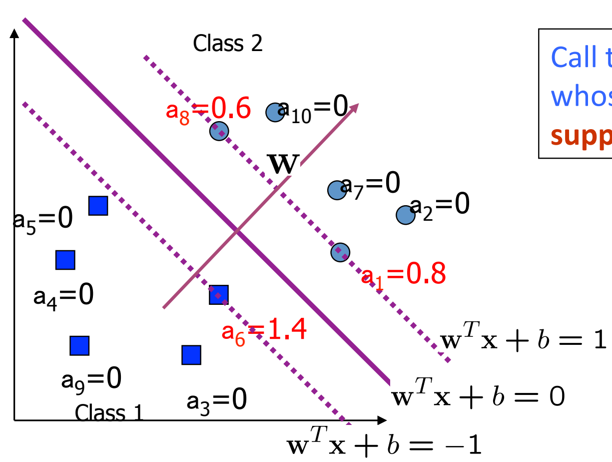
Key for SVM Dual

from page 5–17: if strong duality holds and x, λ, ν are optimal, then they must satisfy the KKT conditions

KKT => Support vectors

- Note the KKT condition --- only a few a_i's can be nonzero!!

$$\alpha_i\left(y_i(\mathbf{w}\cdot\mathbf{x}_i+b)-1\right)=0, \quad i=1,\ldots,n$$

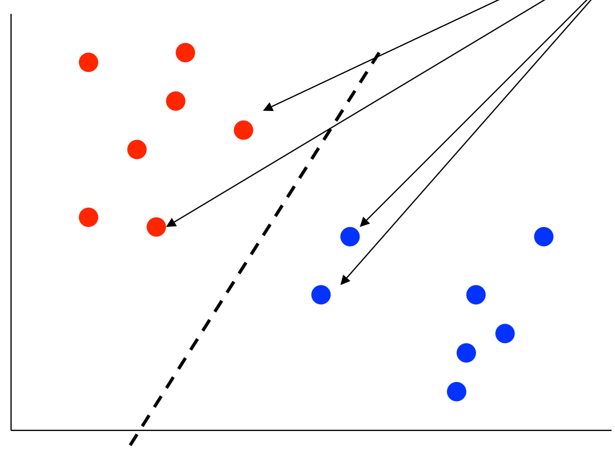


Call the training data points whose a_i's are nonzero the **support vectors** (SV)

Dual SVM - interpretation

training points

$$w = \sum_i \alpha_i x_i y_i$$



$\alpha_i \geq 0$

For α_i that are 0, no influence

Dual SVM for linearly separable case

Our dual target function: $\max_\alpha \sum_{i=1}^{n_{train}} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$

$\sum_{i=1}^{n_{train}} \sum_{j=1}^{n_{train}}$

Training

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \qquad \forall i$$

Dot product for all training samples

Dot product with training samples

To evaluate a new sample x_{ts} we need to compute:

$$w^T x_{ts} + b = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_{ts} + b$$

Testing

$\cancel{\alpha_i \neq 0}$

$\alpha_i > 0$

$$\widehat{y_{ts}} = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i \left(\mathbf{x}_i^T \mathbf{x}_{ts} \right) + b \right)$$

Dual formulation for linearly non separable case

Dual target function:

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i$$

The only difference is that the \alpha are now bounded

Hyperparameter C should be tuned through k-folds CV

To evaluate a new sample x_j we need to compute:

$$w^T x_j + b = \sum_i \alpha_i y_i \mathbf{x_i}^T \mathbf{x_j} + b$$

This is very similar to the optimization problem in the linear separable case, except that there is an upper bound *C* on a_i now

Once again, efficient algorithm exist to find a_i

Fast SVM Implementations

- SMO: Sequential Minimal Optimization
- SVM-Light
- LibSVM
- BSVM
- ……

SMO: Sequential Minimal Optimization

- Key idea
  - Divide the large QP problem of SVM into a series of smallest possible QP problems, which can be solved analytically and thus avoids using a time-consuming numerical QP in the loop (a kind of SQP method).
  - Space complexity: O(n).
  - Since QP is greatly simplified, most time-consuming part of SMO is the evaluation of decision function, therefore it is very fast for linear SVM and sparse data.

SMO

- At each step, SMO chooses 2 Lagrange multipliers to jointly optimize, find the optimal values for these multipliers and updates the SVM to reflect the new optimal values.
- Three components
  - An analytic method to solve for the two Lagrange multipliers
  - A heuristic for choosing which multipliers to optimize
  - A method for computing b at each step, so that the KTT conditions are fulfilled for both the two examples (corresponding to the two multipliers)

Choosing Which Multipliers to Optimize

- First multiplier
  - Iterate over the entire training set, and find an example that violates the KTT condition.
- Second multiplier
  - Maximize the size of step taken during joint optimization.
  - $|E_1 - E_2|$, where E_i is the error on the i-th example.

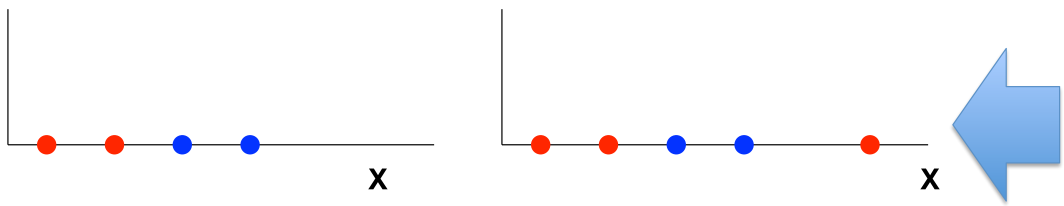
Today

- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Classifying in 1-d

Can an SVM correctly classify this data?

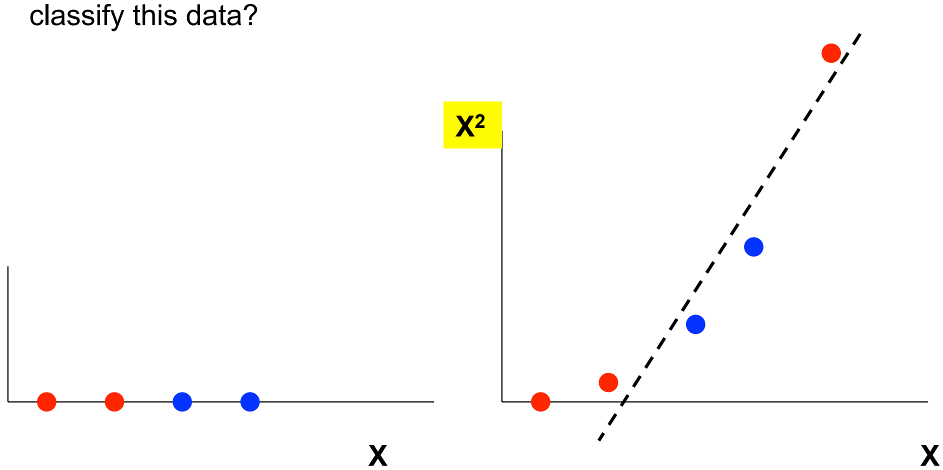
What about this?



Classifying in 1-d

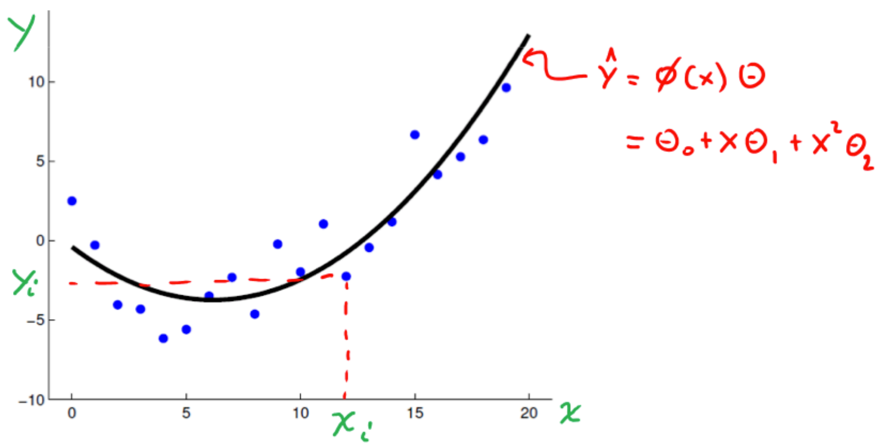
Can an SVM correctly classify this data?

And now? (extend with polynomial basis)



RECAP: **Polynomial regression**

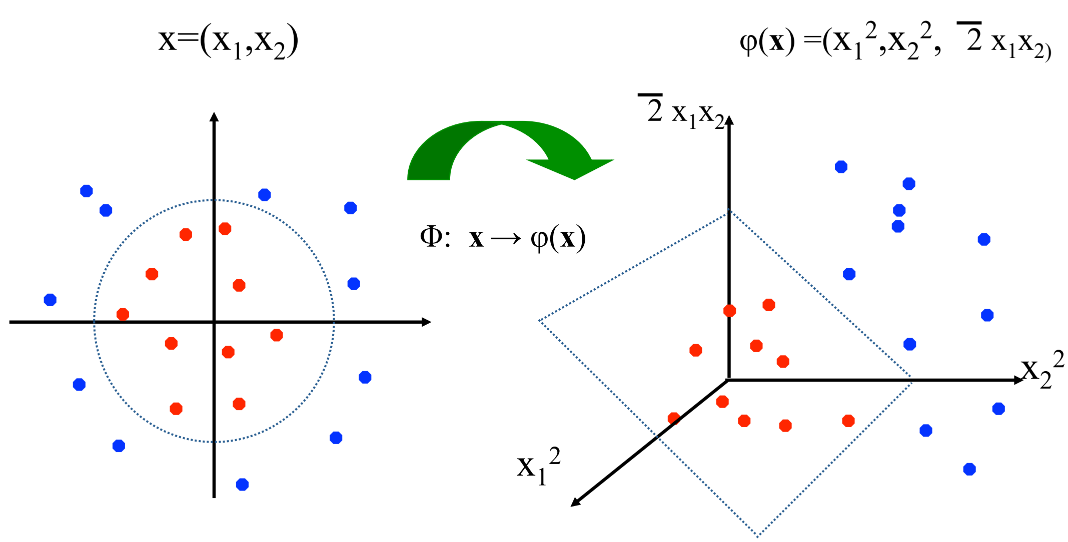
For example, $\phi(x) = [1, x, x^2]$



Dr. Nando de Freitas's tutorial slide

Non-linear SVMs: 2D

- The original input space (x) can be mapped to some higher-dimensional feature space ($\phi(\mathbf{x})$)where the training set is separable:

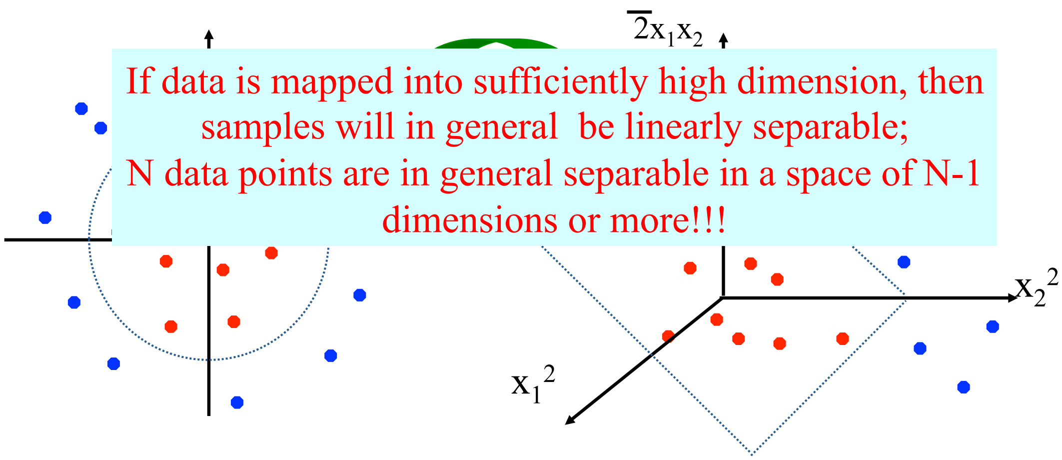


This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm_tutorial.ppt***

Non-linear SVMs: 2D

- The original input space (x) can be mapped to some higher-dimensional feature space ($\phi(\mathbf{x})$)where the training set is separable:

$x=(x_1,x_2)$ $\varphi(\mathbf{x}) =(x_1^2, x_2^2, \overline{2}x_1x_2)$



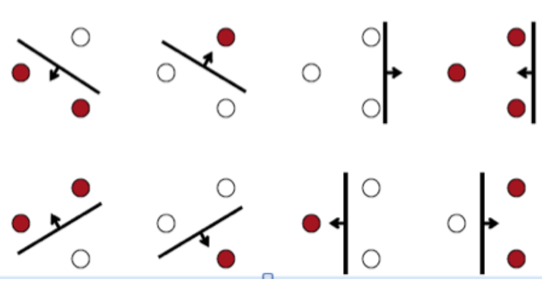
If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;
N data points are in general separable in a space of N-1 dimensions or more!!!

This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm_tutorial.ppt***

A little bit theory: Vapnik-Chervonenkis (VC) dimension

If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;
N data points are in general separable in a space of N-1 dimensions or more!!!

- **VC dimension of the set of oriented lines in R^2 is 3**
  - It can be shown that the VC dimension of the family of oriented separating hyperplanes in R^N is at least N+1

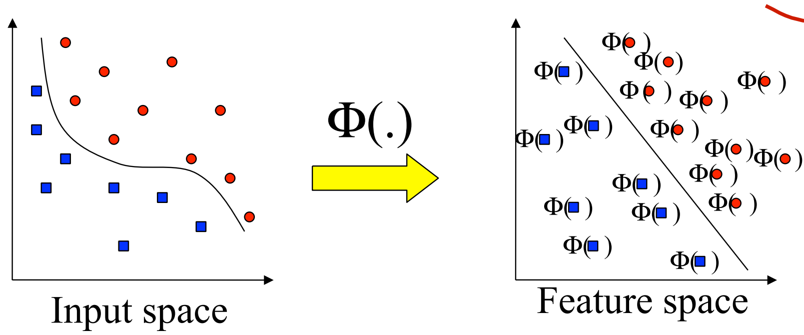


Transformation of Inputs

- Possible problems
  - High computation burden due to high-dimensionality
  - Many more parameters

Is this too much computational work?

- SVM solves these two issues simultaneously
  - "Kernel tricks" for efficient computation
  - Dual formulation only assigns parameters to samples, not features



Quadratic kernels

- While working in higher dimensions is beneficial, it also increases our running time because of the dot product computation
- However, there is a neat trick we can use
- consider all quadratic terms for $x_1, x_2 \ldots x_m$

$$\max_\alpha \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \qquad \forall i$$

m

m is the number of features in each vector

weights on quadratic terms will become clear in the next slide

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

m+1 linear terms

m quadratic terms

m(m-1)/2 pairwise terms

$X \to \Phi(X)$

$$K(\mathbf{x}, z) := \Phi(\mathbf{x})^T \Phi(z)$$

Dot product for quadratic kernels

How many operations do we need for the dot product?

$O(m^2)$ $\quad$ $O(m^2)$

$O(m^2)$

$$\Phi(x)^T\Phi(z) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{bmatrix} \bullet \begin{bmatrix} 1 \\ \sqrt{2}z_1 \\ \vdots \\ \sqrt{2}z_m \\ z_1^2 \\ \vdots \\ z_m^2 \\ \sqrt{2}z_1z_2 \\ \vdots \\ \sqrt{2}z_{m-1}z_m \end{bmatrix} = \sum_i 2x_iz_i + \sum_i x_i^2z_i^2 + \sum_i \sum_{j=i+1} 2x_ix_jz_iz_j + 1$$

m $\quad$ m $\quad$ m(m-1)/2 $\quad$ **=~ m²**

$$K(\mathbf{x}, z) := \Phi(\mathbf{x})^T\Phi(z)$$

The kernel trick

How many operations do we need for the dot product?

$$\Phi(x)^T\Phi(z) = \sum_i 2x_iz_i + \sum_i x_i^2z_i^2 + \sum_i \sum_{j=i+1} 2x_ix_jz_iz_j + 1$$

$O(m^2)$

m $\quad$ m $\quad$ m(m-1)/2 $\quad$ **=~ m²**

However, we can obtain dramatic savings by noting that

$$\Phi(x)^T\Phi(z) = (x^Tz+1)^2 = (x.z+1)^2 = (x.z)^2 + 2(x.z) + 1$$
$$= (\sum_i x_iz_i)^2 + \sum_i 2x_iz_i + 1$$
$$= \sum_i 2x_iz_i + \sum_i x_i^2z_i^2 + \sum_i \sum_{j=i+1} 2x_ix_jz_iz_j + 1$$

$K(x,z)$

$O(m)$

We only need m operations!

So, if we define the **kernel function** as follows, there is no need to carry out basis function \phi(.) explicitly

$$K(\mathbf{x}, z) = (x^Tz+1)^2$$

Where we are

Our dual target function:

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x_i})^T \Phi(\mathbf{x_j})$$

$K(x_i, x_j)$

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \qquad \forall i$$

mn^2 operations at each iteration

To evaluate a new sample x_k we need to compute:

$$w^T \Phi(\mathbf{x}_k) + b = \sum_i \alpha_i y_i \Phi(\mathbf{x_i})^T \Phi(\mathbf{x}_k) + b$$

$K(x_i, x_k)$

mr operations where r are the number of support vectors (whose \alpha$_i$>0)

So, if we define the **kernel function** as follows, there is no need to carry out phi(.) representation explicitly

$$K(\mathbf{x}, z) = (x^T z + 1)^2$$

use of kernel function to avoid carrying out \pha(.) explicitly is known as the kernel trick

Summary: Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training,

Original Linear

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

With kernel function - nonlinear

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j})$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

Summary: Modification Due to Kernel Function

- For testing, the new data **x_ts**

Original Linear

$$\widehat{y_{ts}} = \mathrm{sign}\left(\sum_{i \in \mathrm{train}} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_{ts} + b \right)$$

With kernel function - nonlinear

$$\widehat{y_{ts}} = \mathrm{sign}\left(\sum_{i \in \mathrm{train}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_{ts}) + b \right)$$

$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is called the kernel function.

More examples of kernel functions

- Linear kernel (we've seen it) $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Polynomial kernel (we just saw an example)

$$K(\mathbf{x}, \mathbf{x}') = \left(1 + \mathbf{x}^T \mathbf{x}'\right)^d$$ d

$O(m^d)$

$O(m)$

where p = 2, 3, … To get the feature vectors we concatenate all pth order polynomial terms of the components of x (weighted appropriately)

- Radial basis kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-r \left\| \mathbf{x} - \mathbf{x}' \right\|^2\right)$$

In this case., r is hyperpara. The feature space of the RBF kernel has an infinite number of dimensions

Never represent features explicitly
- ☐ Compute dot products in closed form

Very interesting theory – Reproducing Kernel Hilbert Spaces
- ☐ Not covered in detail here

Kernel Function : Implicit Basis Representation

- For some kernels (e.g. RBF) the implicit transform basis form \phi(**x**) is infinite-dimensional!
  - But calculations with kernel are done in original space, so computational burden and curse of dimensionality aren't a problem.

An example: Support vector machines with polynomial kernel

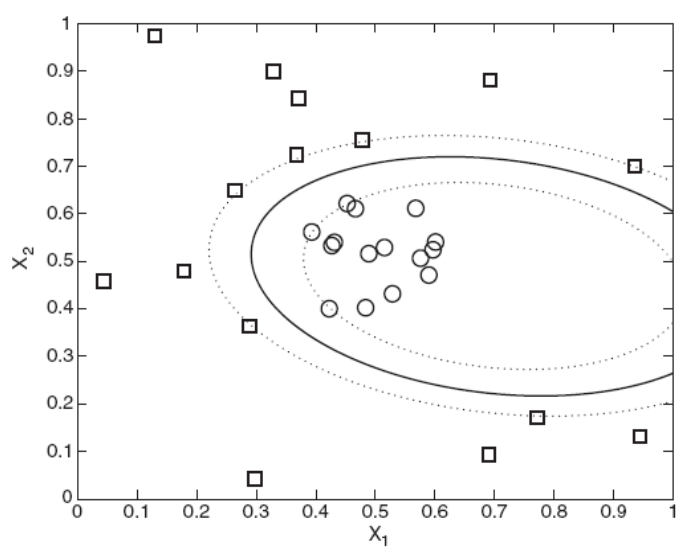


Figure 5.29. Decision boundary produced by a nonlinear SVM with polynomial kernel.

Kernel Functions

- In practical use of SVM, only the kernel function (and not \phi(.)) is specified
- Kernel function can be thought of as a similarity measure between the input objects
- Not all similarity measure can be used as kernel function, however Mercer's condition states that any positive semi-definite kernel *K*(*x*, *y*), i.e. $$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

  can be expressed as a dot product in a high dimensional space.

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarize all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, tree kernel, graph kernel, ...)
  - Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors
- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try for most applications.

Why do SVMs work?

- If we are using huge features spaces (e.g., with kernels), how come we are not overfitting the data?
  - ✓ Number of parameters remains the same (and most are set to 0)
  - ✓ While we have a lot of input values, at the end we only care about the support vectors and these are usually a small group of samples
  - ✓ The minimization (or the maximizing of the margin) function acts as a sort of regularization term leading to reduced overfitting

Why SVM Works?

- Vapnik argues that the fundamental problem is not the number of parameters to be estimated. Rather, the problem is about the flexibility of a classifier
- Vapnik argues that the flexibility of a classifier should not be characterized by the number of parameters, but by the capacity of a classifier
  - This is formalized by the "VC-dimension" of a classifier
- The SVM objective can also be justified by structural risk minimization: the empirical risk (training error), plus a term related to the generalization ability of the classifier, is minimized
- Another view: the SVM loss function is analogous to ridge regression. The term $\frac{1}{2}||w||^2$ "shrinks" the parameters towards zero to avoid overfitting

Today

- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide

Software

- A list of SVM implementation can be found at
  - http://www.kernel-machines.org/software.html
- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Summary: Steps for Using SVM in HW

- Prepare the feature-data matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

Practical Guide to SVM

- From authors of as LIBSVM:
  - A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
  - http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

LIBSVM

- http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - ✓Developed by Chih-Jen Lin etc.
  - ✓Tools for Support Vector classification
  - ✓Also support multi-class classification
  - ✓C++/Java/Python/Matlab/Perl wrappers
  - ✓Linux/UNIX/Windows
  - ✓SMO implementation, fast!!!

A Practical Guide to Support Vector Classification

(a) Data file formats for LIBSVM

- Training.dat

+1 1:0.708333 2:1 3:1 4:-0.320755

-1 1:0.583333 2:-1 4:-0.603774 5:1

+1 1:0.166667 2:1 3:-0.333333 4:-0.433962

-1 1:0.458333 2:1 3:1 4:-0.358491 5:0.374429

…

- Testing.dat

(b) Feature Preprocessing

- (1) Categorical Feature
  - Recommend using m numbers to represent an m-category attribute.
  - Only one of the m numbers is one, and others are zero.
  - For example, a three-category attribute such as {red, green, blue} can be represented as (0,0,1), (0,1,0), and (1,0,0)

A Practical Guide to Support Vector Classification

Feature Preprocessing

- (2) Scaling before applying SVM is very important
  - to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges.
  - to avoid numerical difficulties during the calculation
  - Recommend linearly scaling each attribute to the range [1, +1] or [0, 1].

A Practical Guide to Support Vector Classification

Of course we have to use the same method to scale both training and testing data. For example, suppose that we scaled the first attribute of training data from $[-10, +10]$ to $[-1, +1]$. If the first attribute of testing data lies in the range $[-11, +8]$, we must scale the testing data to $[-1.1, +0.8]$. See Appendix B for some real examples.

If training and testing sets are separately scaled to $[0, 1]$, the resulting accuracy is lower than 70%.

```
$ ../svm-scale -l 0 svmguide4 > svmguide4.scale
$ ../svm-scale -l 0 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 69.2308% (216/312) (classification)
```

Using the same scaling factors for training and testing sets, we obtain much better accuracy.

```
$ ../svm-scale -l 0 -s range4 svmguide4 > svmguide4.scale
$ ../svm-scale -r range4 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 89.4231% (279/312) (classification)
```

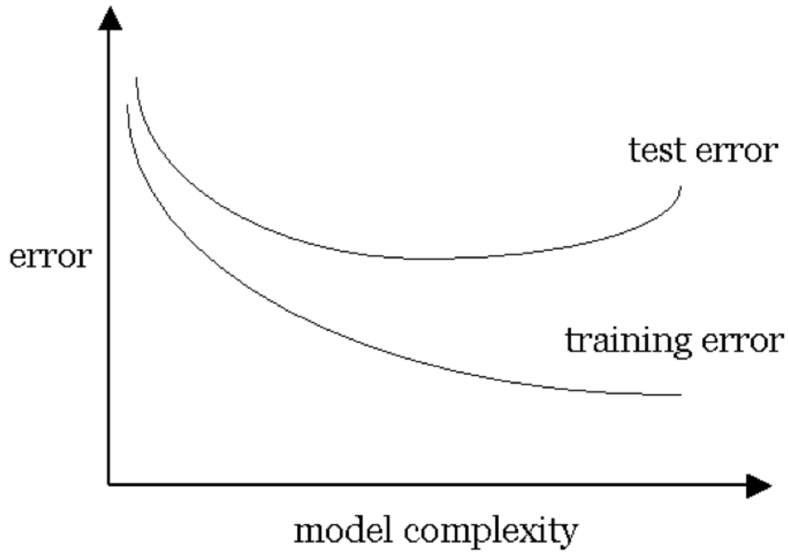

Feature Preprocessing

- (3) missing value
  - Very very tricky !
  - Easy way: to substitute the missing values by the mean value of the variable
  - A little bit harder way: imputation using nearest neighbors
  - Even more complex: e.g. EM based (beyond the scope)

A Practical Guide to Support Vector Classification

(c) Model Selection

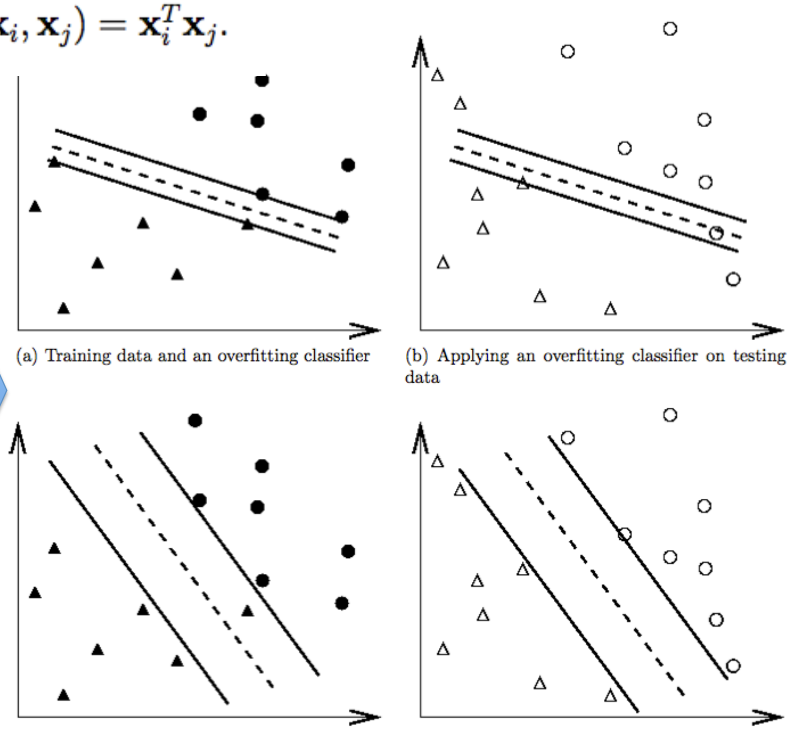
Our goal: find the model M which minimizes the test error:



(c) Model Selection (e.g. for linear kernel)

- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.

Select the right penalty parameter C



(a) Training data and an overfitting classifier

(b) Applying an overfitting classifier on testing data

(c) Training data and a better classifier

(d) Applying a better classifier on testing data

(c) Model Selection

- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.

  two parameters for an RBF kernel: C and γ

- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r)^d$, $\gamma > 0$.

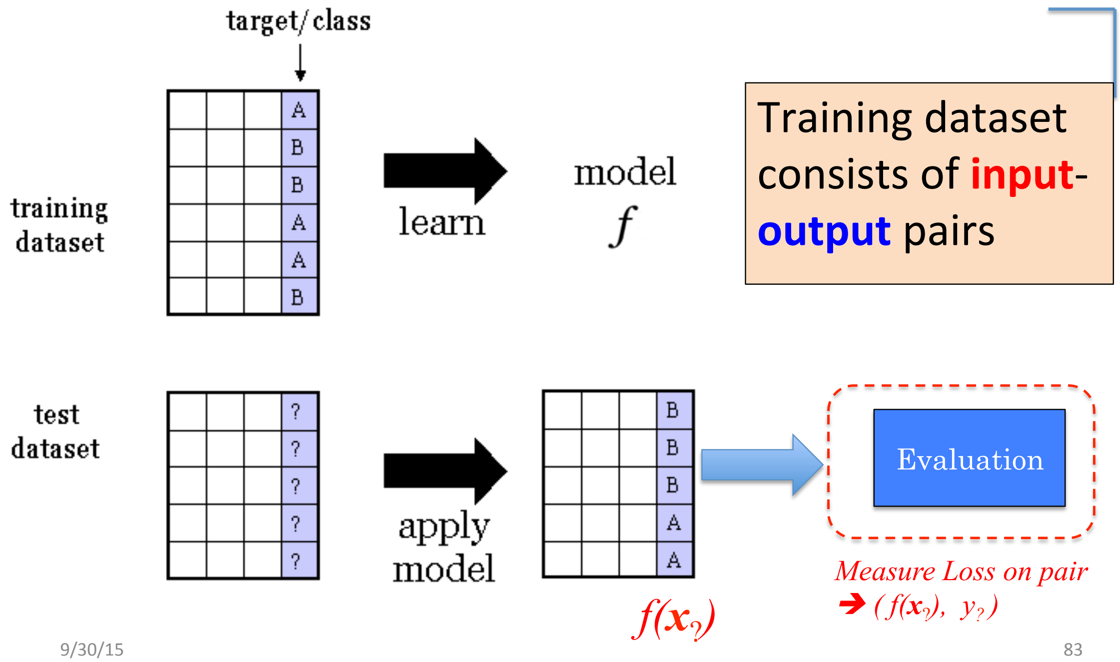
  Three parameters for a polynomial kernel

A Practical Guide to Support Vector Classification

(d) Pipeline Procedures

- (1) train / test
- (2) k-folds cross validation
- (3) k-CV on train to choose hyperparameter / then test

Evaluation Choice-I: Train and Test

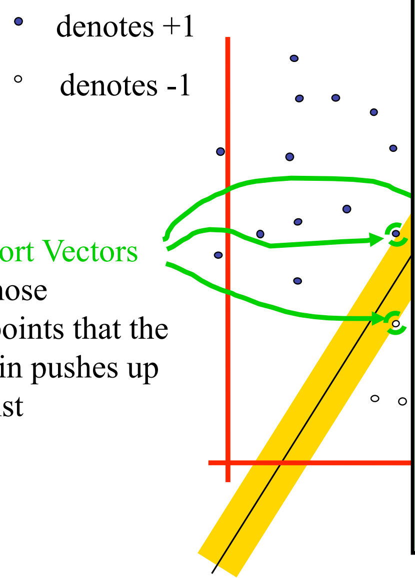


Evaluation Choice-II: Cross Validation

- Problem: don't have enough data to set aside a test set
- Solution: Each data point is used both as train and test
- Common types:
  - -K-fold cross-validation (e.g. K=5, K=10)
  - -2-fold cross-validation
  - -Leave-one-out cross-validation (LOOCV)

A good practice is : to random shuffle all training sample before splitting

Why Maximum Margin for SVM ?



Support Vectors are those datapoints that the margin pushes up against

1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. **LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.**
4. There's some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
5. Empirically it works very very well.

Evaluation Choice-III:

Many beginners use the following procedure now:

- Transform data to the format of an SVM package
- Randomly try a few kernels and parameters
- Test

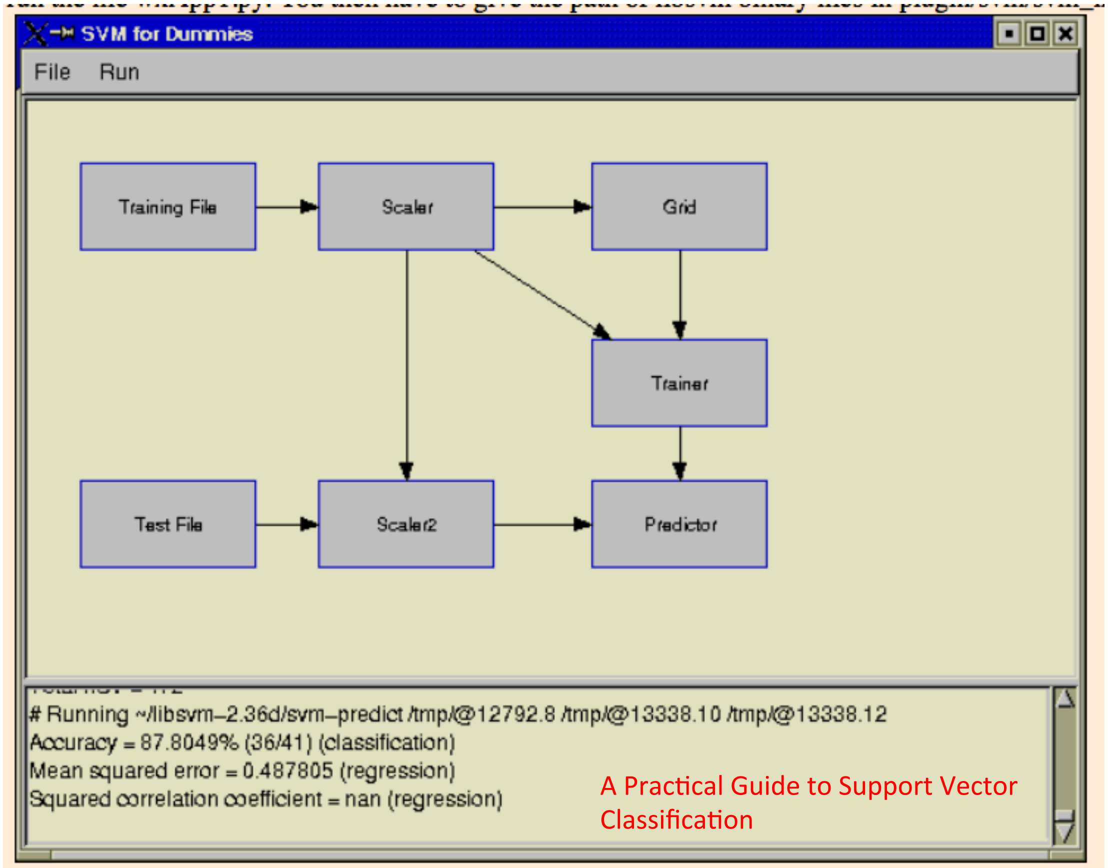
Basic solution For HW2-Q2

We propose that beginners try the following procedure first:

- Transform data to the format of an SVM package
- Conduct simple scaling on the data
- Consider the RBF kernel $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to train the whole training set[5]
- Test

more advanced solution For HW2-Q2

A Practical Guide to Support Vector Classification



Today: Review & Practical Guide

- Support Vector Machine (SVM)
  - ✓ History of SVM
  - ✓ Large Margin Linear Classifier
  - ✓ Define Margin (M) in terms of model parameter
  - ✓ Optimization to learn model parameters (w, b)
  - ✓ Non linearly separable case
  - ✓ Optimization with dual form
  - ✓ Nonlinear decision boundary
  - ✓ Practical Guide
    - ✓ File format / LIBSVM
    - ✓ Feature preprocsssing
    - ✓ Model selection
    - ✓ Pipeline procedure

References

- Big thanks to Prof. Ziv Bar-Joseph and Prof. Eric Xing @ CMU for allowing me to reuse some of his slides
- Elements of Statistical Learning, by Hastie, Tibshirani and Friedman
- Prof. Andrew Moore @ CMU's slides
- Tutorial slides from Dr. Tie-Yan Liu, MSR Asia
- A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
- Tutorial slides from Stanford "Convex Optimization I — Boyd & Vandenberghe