Introduction to Support Vector Machines

CS 536: Machine Learning

Littman (Wu, TA)

Administration

Slides borrowed from Martin Law (from the web).

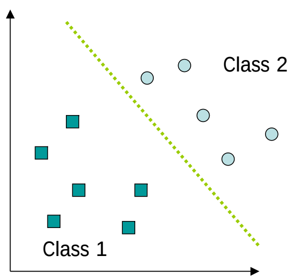
Outline

- History of support vector machines (SVM)
- Two classes, linearly separable
  - What is a good decision boundary?
- Two classes, not linearly separable
- How to make SVM non-linear: kernel trick
- Demo of SVM
- Epsilon support vector regression (ε-SVR)
- Conclusion

History of SVM

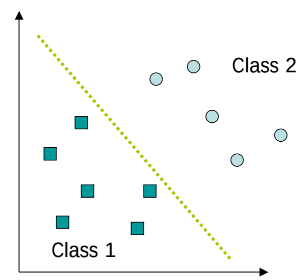
- SVM is a classifier derived from statistical learning theory by Vapnik and Chervonenkis
- SVM was first introduced in COLT-92
- SVM became famous when, using pixel maps as input, it gave accuracy comparable to NNs with hand-designed features in a handwriting recognition task
- Currently, SVM is closely related to:
  - Kernel methods, large margin classifiers, reproducing kernel Hilbert space, Gaussian process, Boosting

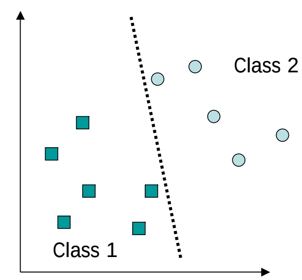
Linear Separable Case



- Many decision boundaries can separate these two classes
- Which one should we choose?

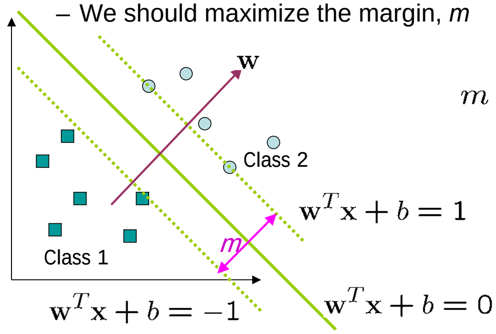
Bad Decision Boundaries





Margin Should Be Large

- The decision boundary should be as far away from the data as possible
  - We should maximize the margin, m



$$m = \frac{2}{||\mathbf{w}||}$$

The Optimization Problem

- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly $\Rightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$
- A constrained optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

The Optimization Problem

- We can transform the problem to its dual

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

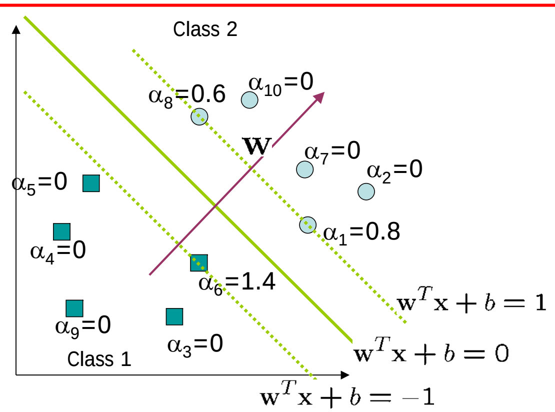
$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
  - Global maximum of α_i can always be found
- **w** can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

Characteristics of the Solution

- Many of the α_i are zero
  - **w** is a linear combination of a small number of examples
  - Sparse representation
- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j=1, ..., s) be the indices of the s support vectors. We can write $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- For testing with a new data **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 **z** as class 1 if the sum is positive, and class 2 otherwise

A Geometric Interpretation

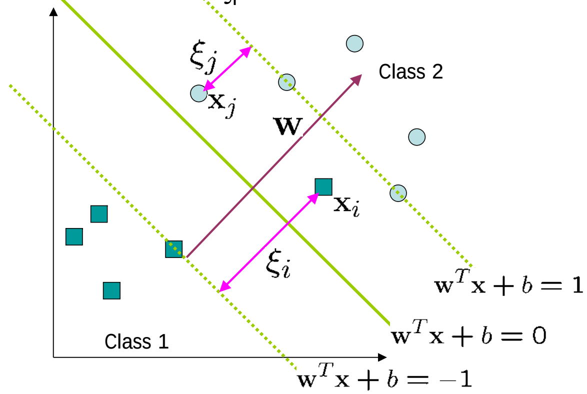


Some Notes

- There are PAC-type bounds on the error on unseen data for SVM as a function of the margin
  - The larger the margin, the tighter the bound
  - The smaller the number of SV, the tighter the bound
- Note that in both training and testing, the data are referenced only as inner products, $\mathbf{x}^T\mathbf{y}$
  - This is important for generalizing to the non-linear case

Non-Linearly Separable

- We allow "error" ξ_i in classification



Soft Margin Hyperplane

- Define ξ_i=0 if there is no error for x_i
  - ξ_i are just "slack variables" in optimization theory
  - $\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$
- We want to minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$
  - *C* : tradeoff parameter between error and margin
- The optimization problem becomes

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The Optimization Problem

- The dual of the problem is

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$
$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

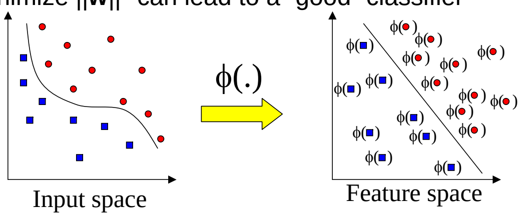
- **w** is also recovered as $\mathbf{w} = \sum_{j=1}^{s}\alpha_{t_j} y_{t_j}\mathbf{x}_{t_j}$
- The only difference with the linearly separable case is that there is an upper bound *C* on α_i
- Once again, a QP solver can be used to find α_i

Extension to Non-linear

- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - Input space: the space containing $\mathbf{x}_i$
  - Feature space: the space of $\phi(\mathbf{x}_i)$ after transformation
- Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - The classification task can be "easier" with a proper transformation. Example: XOR

Extension to Non-linear

- Possible problem of the transformation
  - High computation burden and hard to get a good estimate
- SVM solves these issues simultaneously
  - Kernel tricks for efficient computation
  - Minimize $||\mathbf{w}||^2$ can lead to a "good" classifier



Example Transformation

- Define the kernel function K ($\mathbf{x}$,$\mathbf{y}$) as

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1 y_1 + x_2 y_2)^2$$

- Consider the following transformation

$$\phi(\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

$$\phi(\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]) = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1 y_2)$$

$$\begin{aligned} \langle \phi(\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]), \phi(\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]) \rangle &= (1 + x_1 y_1 + x_2 y_2)^2 \\ &= K(\mathbf{x}, \mathbf{y}) \end{aligned}$$

- The inner product can be computed by K without going through the map $\phi(.)$

Kernel Trick

- The relationship between the kernel function K and the mapping $\phi(.)$ is

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$

  – This is known as the *kernel trick*
- In practice, we specify K, thereby specifying $\phi(.)$ indirectly, instead of choosing $\phi(.)$
- Intuitively, K($\mathbf{x}$,$\mathbf{y}$) represents our desired notion of similarity between data **x** and **y** and this is from our prior knowledge
- K($\mathbf{x}$,$\mathbf{y}$) needs to satisfy a technical condition (Mercer condition) in order for $\phi(.)$ to exist

Example Kernel Functions

- Polynomial kernel with degree d

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

- Radial basis function kernel with width σ

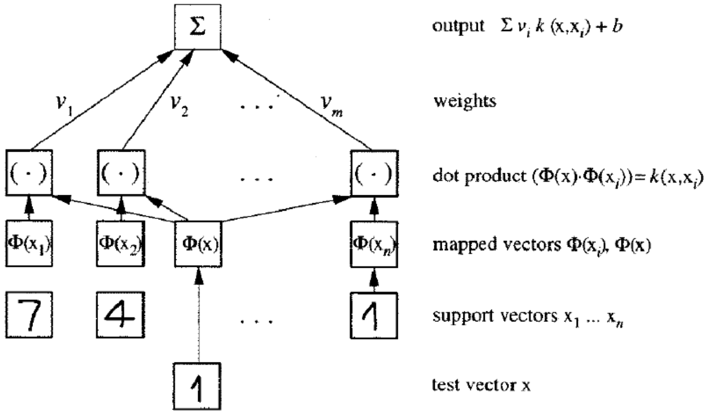
$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  – Closely related to radial basis function neural networks
- Sigmoid with parameter κ and θ

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

  – It does not satisfy the Mercer condition on all κ and θ
- Research on different kernel functions in different applications is *very* active

Handwriting Recognition



Using Kernel Functions

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

Original

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

With kernel function

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

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

Using Kernel Functions

- For testing, the new data **z** is classified as Class 1 if $f \geq 0$, and as Class 2 if $f < 0$

Original

With kernel function

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum^{s} \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

Example

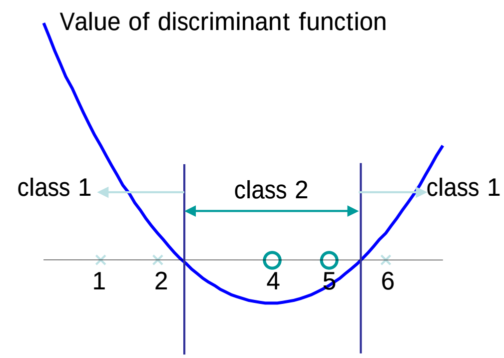
- By using a QP solver, we get
  - α_1=0, α_2=2.5, α_3=0, α_4=7.333, α_5=4.833
  - Note that the constraints are indeed satisfied
  - The support vectors are {x_2=2, x_4=5, x_5=6}
- The discriminant function is

$$f(y) = 2.5(1)(2y+1)^2 + 7.333(-1)(5y+1)^2 + 4.833(1)(6y+1)^2 + b$$
$$= 0.6667x^2 - 5.333x + b$$

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x_2, x_4, x_5 lie on $y_i(\mathbf{w}^T\phi(z)+b)=1$

$$\Longrightarrow f(y) = 0.6667x^2 - 5.333x + 9$$

Example



Multi-class Classification

- SVM is basically a two-class classifier
- One can change the QP formulation to allow multi-class classification
- More commonly, the data set is divided into two parts "intelligently" in different ways and a separate SVM is trained for each way of division
- Multi-class classification is done by combining the output of all the SVM classifiers
  - Majority rule
  - Error correcting code
  - Directed acyclic graph

Software

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

Steps for Classification

- Prepare the pattern 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 values
- Unseen data can be classified using the α_i values and the support vectors

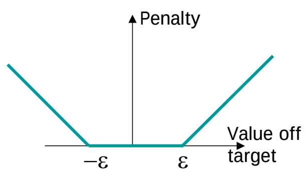
SVM Strengths & Weaknesses

- Strengths
  - Training is relatively easy
    - No local optimal, unlike in neural networks
  - It scales relatively well to high dimensional data
  - Tradeoff between classifier complexity and error can be controlled explicitly
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Weaknesses
  - Need a "good" kernel function

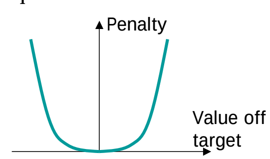
ε Support Vector Regression

- Linear regression in feature space
- Unlike in least square regression, the error function is ε-insensitive loss function
  - Intuitively, mistake less than ε is ignored
  - This leads to sparsity similar to SVM

ε-insensitive loss function



Square loss function



ε Support Vector Regression

- Given: a data set $\{x_1, ..., x_n\}$ with target values $\{u_1, ..., u_n\}$, we want to do ε-SVR
- The optimization problem is

$$\text{Min } \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$\text{subject to } \begin{cases} u_i - \mathbf{w}^T\mathbf{x}_i - b \le \epsilon + \xi_i \\ \mathbf{w}^T\mathbf{x}_i + b - u_i \le \epsilon + \xi_i^* \\ \xi_i \ge 0, \xi_i^* \ge 0 \end{cases}$$

- Similar to SVM, this can be solved as a quadratic programming problem

ε Support Vector Regression

- *C* is a parameter to control the amount of influence of the error
- The $||w||^2$ term serves as controlling the complexity of the regression function
  - This is similar to ridge regression
- After training (solving the QP), we get values of α_i and α_i^*, which are both zero if $\mathbf{x}_i$ does not contribute to the error function
- For a new instance **z**,

$$f(\mathbf{z}) = \sum_{j=1}^{s} (\alpha_{t_j} - \alpha_{t_j}^*) K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

Other Kernel Methods

- A lesson learned in SVM: a linear algorithm in the feature space is equivalent to a non-linear algorithm in the input space
- Classic linear algorithms can be generalized to its non-linear version by going to the feature space
  - Kernel principal component analysis, kernel independent component analysis, kernel canonical correlation analysis, kernel k-means, 1-class SVM are some examples

Conclusion

- SVM is a useful method for classification
- Two key concepts of SVM: maximize the margin and the kernel trick
- Much active research is taking place on areas related to SVM
- Many SVM implementations are available on the web for you to try on your data set!

Resources

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html