CS789: Machine Learning and Neural Network

Support Vector Machine

Jakramate Bootkrajang

Department of Computer Science
Chiang Mai University

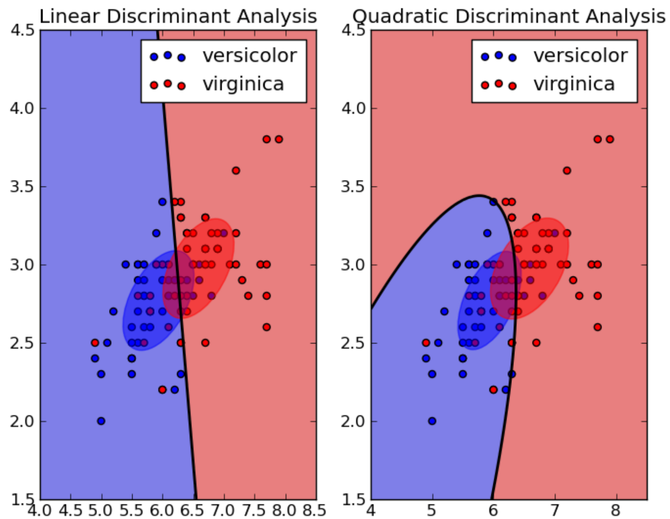
Introduction

- One of the most widely used out-of-the-box discriminative classifier.
- Gives state-of-the-art classification performance.
- Extends naturally to support non-linear classification tasks.

Linear vs Non-linear classifier

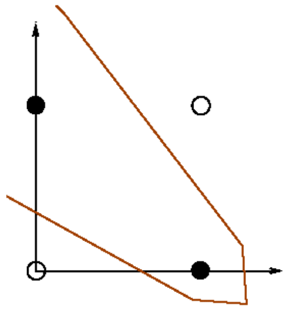
- Linear classifier is in the form
  - $w^T x + b$
  - In words, x is a linear combination of w.
  - b is the bias term.
  - Can be thought of as a line separating classes.
- Non-linear
  - Can be thought of as a curve separating classes.

Linear vs Non-linear classifier



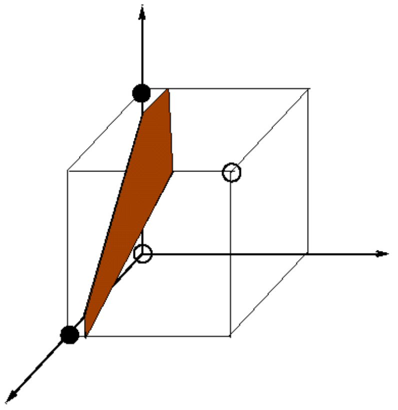
The problem with linear machine

- It cannot solve linearly non-separable classification task such as the XOR problem



But wait!

- See what happen if we embedded the four points of XOR problem in higher dimensional space (i.e. 3D space)

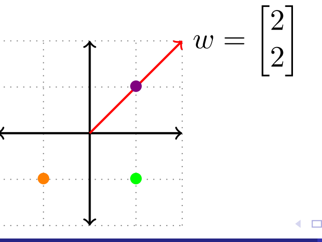


Support Vector Machines (SVM)

- Two key ideas
  - Assuming linearly separable classes, learn separating hyperplane with maximum margin.
  - Expand input into high-dimensional space to deal with linearly non-separable cases (such as the XOR).

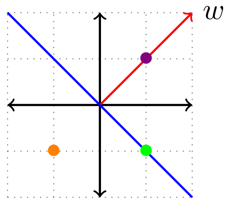
Visualising the decision hyperplane

- Recall a linear classifier $w^T x + b$, or more compactly $\begin{bmatrix} w \\ 1 \end{bmatrix}^T \begin{bmatrix} x \\ b \end{bmatrix}$.
- For classification purpose, we want
  - $w^T x < 0$ for negative class
  - $w^T x > 0$ for positive class.
- For example, $w = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$
  - So, $w^T x_{violet} > 0$ and $w^T x_{orange} < 0$

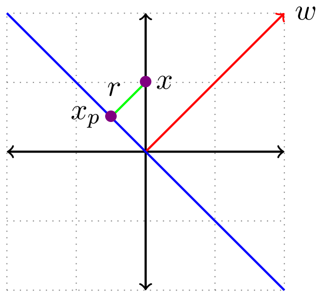


Visualising the decision hyperplane

- Moreover, observe that $w^T x_{green} = 0$.
- A set of points where $w^T x = 0$ defines the decision boundary.
- Geometrically, they are the points(vectors) which are perpendicular to w. (dot product is zero)

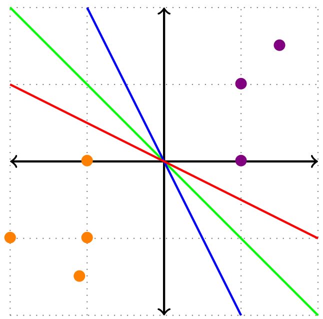


Distance of a point x from the decision hyperplane



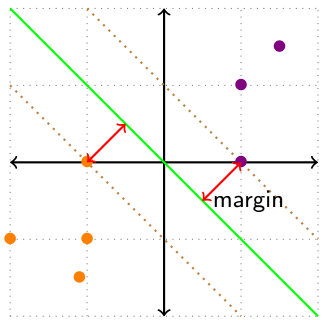
- Represent $x = x_p + r\frac{w}{||w||}$ (r $\times$ unit vector)
- Since $w^T x_p = 0$, we then have $w^T x = w^T x_p + w^T r \frac{w}{||w||}$
- In other words, $r = \frac{w^T x}{||w||}$, (note that r is invariant to scaling of w.)

Many choices of separating hyperplane



SVM's goal = maximum margin

According to a theorem from learning theory, from all possible linear decision functions the one that maximises the margin of the training set will minimise the generalisation error.

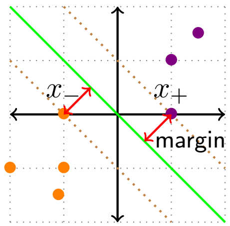


Two types of margins

- Functional margin: $w^T x$
  - Can be increased without bound by multiplying a constant to w.
- Geometric margin: $r = \frac{w^T x}{||w||}$
  - The one that we want to maximise.
  - Subject to the constraint that training examples are classified correctly.

Maximum margin (1/2)

- Since we can scale the functional margin, we can demand the functional margin for the nearest points to be $+1$ and -1 on the two side of the decision boundary.
- Denoting a nearest positive example by x_+ and a nearest negative example by x_-, we have
  - $w^T x_+ = +1$
  - $w^T x_- = -1$



Maximum (geometric) margin

- We then compute the geometric margin from functional margin constraints

$$
\begin{aligned}
\text{margin} &= \frac{1}{2}\left(\frac{w^T x_+}{||w||} - \frac{w^T x_-}{||w||}\right) \\
&= \frac{1}{2||w||}(w^T x_+ - w^T x_-) \\
&= \frac{1}{||w||}
\end{aligned}
$$

Maximum margin: summing up

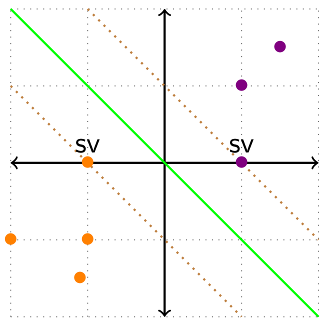
- Given a *linearly separable* training set $S = \{x_i, y_i\}_{i=1}^m$.
- We need to find w which maximise $\frac{1}{||w||}$.
- Maximising $\frac{1}{||w||}$ is equivalent to minimising $||w||^2$
- The objective of SVM is then the following quadratic programming

$$
\begin{aligned}
&\text{minimise:} \quad \frac{1}{2}||w||^2 \\
&\text{subject to:} \\
&\qquad w^T x_i \geq +1 \quad \text{for} \quad y_i = +1 \\
&\qquad w^T x_i \leq -1 \quad \text{for} \quad y_i = -1
\end{aligned}
$$

Or equivalently: $y_i w^T x_i \geq 1$ for all i

Support vectors

- The training points that are nearest to the decision boundary are called support vectors.
- Quiz: what is the output of our decision function for these points?



Solving the quadratic programming with inequality constraints

- Construct & minimise the Lagrangian

$$L(w,\alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i(yw^Tx_i - 1) \qquad (1)$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad \text{for all} \quad i$$

- The optimal w is found by taking derivatives of L w.r.t w

$$\frac{\partial L(w,\alpha)}{\partial w} = w - \sum_{i=1}^{m} \alpha_i y x_i = 0 \qquad (2)$$

- Note that w are expressed as a linear combination of training points.

Solving the quadratic programming

- Karush-Kuhn-Tucker (KKT) condition for optimality requires that

$$\alpha_i(yw^Tx_i - 1) = 0$$

- The Lagrange multipliers α_i are called 'dual variables'
- Each training point has an associated dual variable.
- The condition implies that only support vectors will have non-zero α_i.
  - as its functional output is required to be exactly $+1$ or -1.

Solving the quadratic programming

- It is possible to find the dual of the objective function (eq.1).
  - Dual problem: the new objective having dual variables as its parameters
- Plugging eq.2 into eq.1 to obtain,

$$D(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{m} \alpha_i$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad \text{for all} \quad i$$

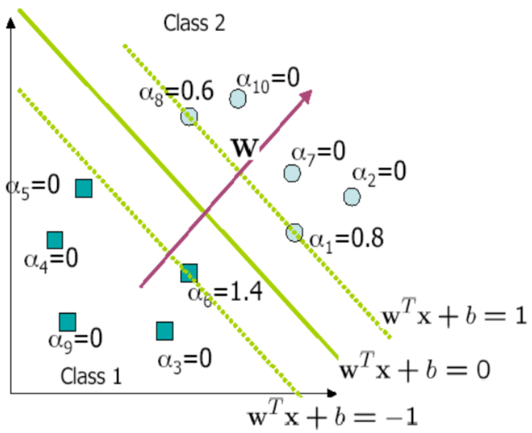
- Note that data enteres only in the form of dot products.

Solving the quadratic programming

- We can use optimation package to solve the above dual problem for α.

Solver	Julia Package	`solver=`	License	LP	SOCP	MILP
Bonmin	AmplNLWriter.jl	`BonminNLSolver()` *	EPL	X		X
	CoinOptServices.jl	`OsilBonminSolver()`				
Cbc	Cbc.jl	`CbcSolver()`	EPL			X
Clp	Clp.jl	`ClpSolver()`	EPL	X		
Couenne	AmplNLWriter.jl	`CouenneNLSolver()` *	EPL	X		X
	CoinOptServices.jl	`OsilCouenneSolver()`				
CPLEX	CPLEX.jl	`CplexSolver()`	Comm.	X	X	X
ECOS	ECOS.jl	`ECOSSolver()`	GPL	X	X	
GLPK	GLPKMath...	`GLPKSolver[LP|MIP]()`	GPL	X		X
Gurobi	Gurobi.jl	`GurobiSolver()`	Comm.	X	X	X
Ipopt	Ipopt.jl	`IpoptSolver()`	EPL	X		
KNITRO	KNITRO.jl	`KnitroSolver()`	Comm.			
MOSEK	Mosek.jl	`MosekSolver()`	Comm.	X	X	X
NLopt	NLopt.jl	`NLoptSolver()`	LGPL			
SCS	SCS.jl	`SCSSolver()`	MIT	X	X	

The learned SVM



Classifying new data points

- Once the parameter α^* (or w^* if primal is used) is found by solving the quadratic optimisation on the training set of points, we can use it to classify new unseen point.
- Given new *test* point x_q, its class membership is

$$
\begin{aligned}
\text{sign}(w^T x_q) &= \sum_{i=1}^{m} \alpha_i^* y_i x_i^T x_q \\
&= \sum_{i \in SV} \alpha_i^* y_i x_i^T x_q
\end{aligned}
$$

Important properties of SVMs

- Sparse
  - Only support vectors are important.
- Data enters in the form of dot products.
  - Ready for kernel trick.
- Dual objective is convex.
- However, SVM is quite sensitive to noisy data (mislabelled data)
  - One such noisy data can dramatically change the decision boundary

Non linearly separable data ?

- We can not hope for every data being linearly separable
  - Indeed, many of the real-world datasets are linearly inseparable
- This includes naturally overlapping classes (no way to be completely separated)
- And also datasets which are quite noisy
  - Originally linearly separable but due to some noise the observed data is not.
- What to do?

Regularised SVM (1/3)

- We will relax constraint, $y_i w^T x_i \geq 1$ by allowing it to be less than 1
- This is accomplished by using slack variables ξ_i for each x_i and write
  - $y_i w^T x_i \geq 1 - \xi_i$
- Our new objective is then

minimise: $$\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i$$

subject to:

$$y_i w^T x_i \geq 1 - \xi_i \quad \text{for all} \quad i$$
$$\xi_i \geq 0 \quad \text{for all} \quad i$$

Regularised SVM (2/3)

- This is an L_1-regularisation
- Parameter C controls the trade-off between fitting the data well and allowing some slackness
- Predictive performance of SVM is known to depend on C paramater
  - Picking C usually done via cross-validation

minimise: $$\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i$$

subject to:

$$y_i w^T x_i \geq 1 - \xi_i \quad \text{for all} \quad i$$
$$\xi_i \geq 0 \quad \text{for all} \quad i$$

Regularised SVM (3/3)

- Of course, we can also derive its dual form

$$D(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{m}\alpha_i$$

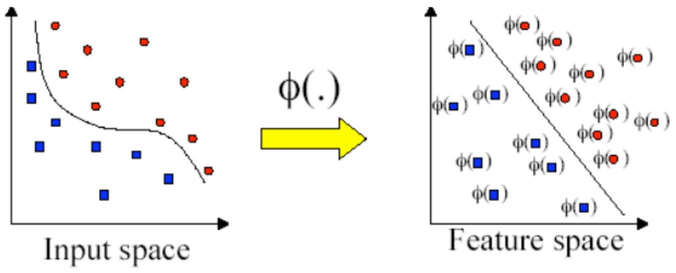
s.t. $\quad 0 \leq \alpha_i \leq C, \quad \text{for all} \quad i$

Non-linear classifiers

- What to do when data is not linearly separable?
- First approach
  - Use non-linear model, e.g., neuron-network, NDA with full covariance
  - (problems: many parameters, local minima, experiences needed to train)
- Second approach
  - Transform data into a richer feature space (including high dimensioal/non-linear features), then use a linear classifier.

Learning in the feature space

Map data into a feature space where they are linearly separable



Non-linear SVMs

- Recall that the linear SVMs depends only on $x^T x$.

$$D(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{m} \alpha_i$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad \text{for all} \quad i$$

- After the mapping, the non-linear algorithm will depend only on $\phi(x_i)^T \phi(x_j)$

$$D(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) + \sum_{i=1}^{m} \alpha_i$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad \text{for all} \quad i$$

- The dot product $\phi(x_i)^T \phi(x_j)$ is known as kernel function.

Kernels

- A function that gives the dot product between the vectors in feature space induced by the mapping ϕ.

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

- In a matrix form, over all data, the matrix is also called Gram matrix.

$$K = \begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_j) & \dots & K(x_1, x_m) \\ \vdots & \ddots & \vdots & & \vdots \\ K(x_i, x_1) & \dots & K(x_i, x_j) & \dots & K(x_1, x_m) \\ \vdots & & \vdots & \ddots & \vdots \\ K(x_m, x_1) & \dots & K(x_m, x_j) & \dots & K(x_m, x_m) \end{bmatrix}$$

- Gram matrix, K, is positive semi-definite, i.e. $\alpha K \alpha \geq 0$ for all $\alpha \in \mathbb{R}^m$.

Kernels: Polynomial kernel

- Example: mapping $\mathbf{x}, \mathbf{y}$ points in 2D input to 3D feature space.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

- The corresponding mapping ϕ is

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \quad \phi(\mathbf{y}) = \begin{bmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{bmatrix}$$

- So we get a kernel $K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$
  - A polynomial kernel of degree 2.
- Using kernel trick, we may not even need to know the mapping ϕ.

Kernels: Gaussian kernel

- Defined as

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

- This comes from writing

$$e^a = 1 + a + \cdots + \frac{1}{k!}a^k \qquad \text{Taylor's expansion}$$

- Let $a = \mathbf{x}^T\mathbf{y}$, one can see that $e^{\mathbf{x}^T\mathbf{y}}$ is a kernel with infinite dimension.
- Normalising $e^{\mathbf{x}^T\mathbf{y}}$ with σ and dividing the term by $e^{||\mathbf{x}||}$ and $e^{||\mathbf{y}||}$ to get the Gaussian kernel.

Making kernels

- New kernels can be made from valid kernels as long as the resulting Gram matrix is positive definite.
- The following operations are allowed
  - $K(x, y) = K_1(x, y) + K_2(x, y)$ (addition)
  - $K(x, y) = \lambda K_1(x, y)$ (scaling)
  - $K(x, y) = K_1(x, y) \times K_2(x, y)$ (multiplication)
- There is a theorem called Mercer's theorem that characterises valid kernels.

Many other kernels

- Linear kernel $K(x, y) = x^Ty + c$
- Exponential kernel $K(x, y) = \exp(-\frac{||x-y||}{2\sigma^2})$
- Sigmoid kernel $K(x, y) = \tanh(\alpha x^Ty + c)$
- Histogram intersection kernel $K(x, y) = \sum_{i=1}^n \min(x_i, y_i)$
- Cauchy kernel $K(x, y) = \frac{1}{1+\frac{||x+y||^2}{\sigma^2}}$
- Discrete structure kernel: string kernel, tree kernel, graph kernel.
- And many more ...

Bad kernel

- Gram matrix encodes similarities between input data points.
- A bad kernel would be a kernel function which gives near diagonal Gram matrix

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ & & 1 & & \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

- No clusters, no structure.

SVMs and Kernels

- Prepare the data matrix.
- Select the kernel function to use, compute Gram matrix.
- Execute the training algorithm using a QP solver to obtain the α_i values
- Unseen data can be classified using the α_i values and the support vectors

$$f(x) = \text{sign}(\sum_{i=1}^{SV} \alpha_i y_i K(x_i, x))$$

Applications

- Handwritten digits recognition
  - Dataset: US Postal service
  - 4% error was obtained
  - about only 4% of the training data were SVs.
- Text categorisation
- Face detection
- DNA analysis
- And many more ...

Summary

- SVMs learn linear decision boundaries. (discriminative approach)
  - They pick the hyperplane that maximises the (geometric) margin.
  - The optimal hyperplane turns out to be a linear combination of support vectors.
- Transform nonlinear problems to higher dimensional space using kernel functions.
- In hope for linearly-separable classes in the transformed space.