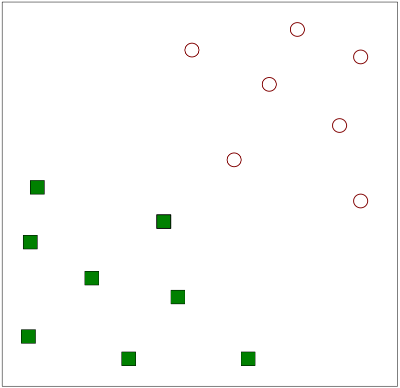
Classification

Linear SVM

Huiping Cao

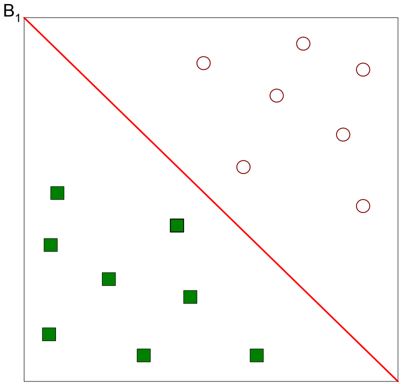
Support Vector Machines

Find a linear hyperplane (decision boundary) that will separate the data.



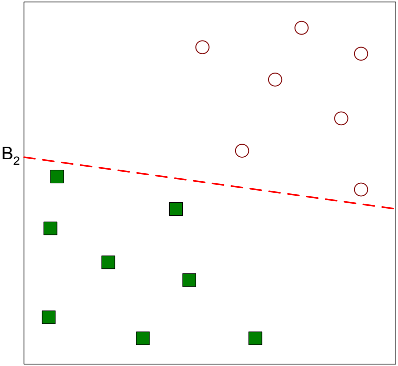
Support Vector Machines

One possible solution.



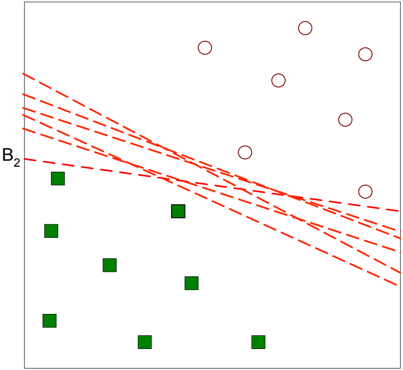
Support Vector Machines

Another possible solution.



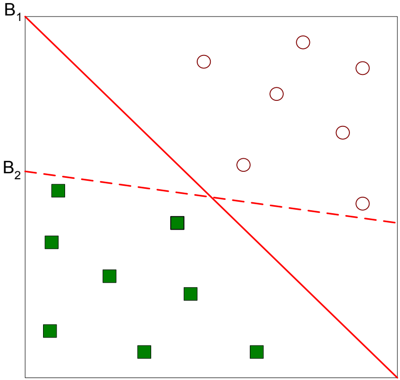
Support Vector Machines

Other possible solutions.



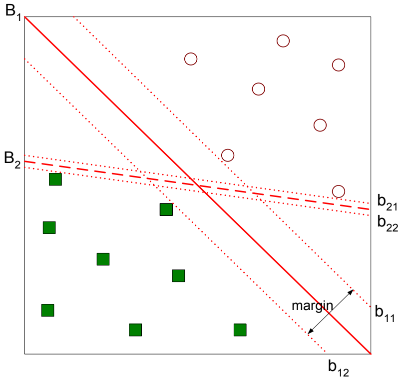
Support Vector Machines

Which one is better? B1 or B2?
How do you define better? The bigger the margin, the better.

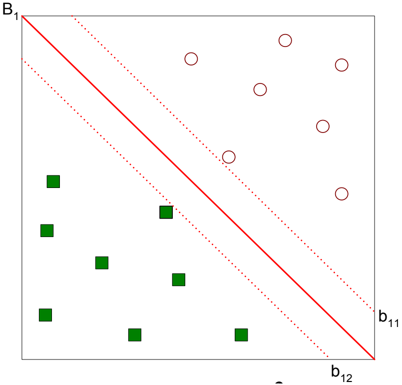


Support Vector Machines

Find hyperplane that maximizes the margin; so B1 is better than B2

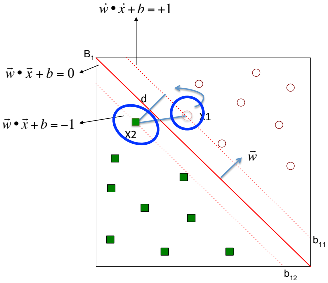


Support Vector Machines



- $B_1 : \mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$
- $b_{11} : \mathbf{w}^\mathsf{T}\mathbf{x} + b = 1$
- $b_{12} : \mathbf{w}^\mathsf{T}\mathbf{x} + b = -1$

Support Vector Machines



- The margin of the decision boundaries:
  - $\mathbf{w}^\mathsf{T}(\mathbf{x}_1 - \mathbf{x}_2) = 2$
  - $\mathbf{w}^\mathsf{T}(\mathbf{x}_1 - \mathbf{x}_2) = \|\mathbf{w}\| \times \|(\mathbf{x}_1 - \mathbf{x}_2)\| \times cos(\theta)$, where $\| \cdot \|$ is the norm of a vector.
  - $\|\mathbf{w}\| \times \|(\mathbf{x}_1 - \mathbf{x}_2)\| \times cos(\theta) = \|\mathbf{w}\| \times d$, where d is the length of vector $(\mathbf{x}_1 - \mathbf{x}_2)$ in the direction of vector $\mathbf{w}$
  - Thus, $\|\mathbf{w}\| \times d = 2$
  - Margin: $d = \frac{2}{\|\mathbf{w}\|}$

Learn a linear SVM Model

- The training phase of SVM is to estimate the parameters $\mathbf{w}$ and b.

- The parameters must follow two conditions:

$$y_i = \begin{cases} 1, & \text{if } \mathbf{w}^\intercal \mathbf{x}_i + b \geq 1 \\ -1, & \text{if } \mathbf{w}^\intercal \mathbf{x}_i + b \leq -1 \end{cases}$$

Formulate the problem – rationale

- We want to maximize margin: $d = \frac{2}{\|\mathbf{w}\|}$
  which is equivalent to minimize $L(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2}$.
- Subjected to the following constraints:

$$y_i = \begin{cases} 1 & \text{if } \mathbf{w}^\mathsf{T}\mathbf{x}_i + b \geq 1 \\ -1 & \text{if } \mathbf{w}^\mathsf{T}\mathbf{x}_i + b \leq -1 \end{cases}$$

  which is equivalent to

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1, i = 1, \cdots, N$$

Formulate the problem

- Formalized to the following constrained optimization problem.
  Objective function:

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

  subject to

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1, i = 1, \cdots, N$$

- **Convex** optimization problem:
  - Objective function is quadratic
  - Constraints are linear
  - Can be solved using the standard **Lagrange multiplier** method.

Lagrange formulation

- Lagrangian for the optimization problem (take into account the constraints by rewriting the objective function).

$$\mathcal{L}_P(\mathbf{w}, b, \lambda) = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{N} \lambda_i (y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - 1)$$

  minimize w.r.t. $\mathbf{w}$ and b and maximize w.r.t. each $\lambda_i \geq 0$
  where λ_i: Lagrange multiplier
- To minimize the Lagrangian, take the derivative of $\mathcal{L}_P(\mathbf{w}, b, \lambda)$ w.r.t. $\mathbf{w}$ and b and set them to 0:

$$\frac{\partial \mathcal{L}_P}{\partial \mathbf{w}} = 0 \Longrightarrow \mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

$$\frac{\partial \mathcal{L}_P}{\partial b} = 0 \Longrightarrow \sum_{i=1}^{N} \lambda_i y_i = 0$$

Lagrange multiplier

- ~~Quadratic programming (~~QP) solves $\lambda = \lambda_1, \lambda_2, \cdots, \lambda_N$, where most of them are zeros.
- Karush-Kuhn-Tucker (KKT) conditions

$$\lambda_i \geq 0$$

The constraint (zero form with extreme value)

$$\lambda_i(y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - 1) = 0$$

  - Either λ_i is zero
  - Or $y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - 1) = 0$
- Support vector $\mathbf{x}_i$: $y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b - 1) = 0$ and $\lambda_i > 0$
- Training instances that do not reside along these hyperplanes have $\lambda_i = 0$

Get $\mathbf{w}$ and b

- $\mathbf{w}$ and b depend on support vectors $\mathbf{x}_i$ and its class label y_i.
- $\mathbf{w}$ value

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

- b value

$$b = y_i - \mathbf{w}^\intercal \mathbf{x}_i$$

Idea:

- Given a support vector $(\mathbf{x}_i, y_i)$, we have $y_i(\mathbf{w}^\intercal \mathbf{x}_i + b) - 1 = 0$.
- Multiply y_i on both sides, $\rightarrow y_i^2(\mathbf{w}^\intercal \mathbf{x}_i + b) - y_i = 0$.
- $y_i^2 = 1$ because $y_i = 1$ or $y_i = -1$.
- Then, $(\mathbf{w}^\intercal \mathbf{x}_i + b) - y_i = 0$.

Get **w** and b – Example

x_{i1}	x_{i2}	y_i	λ_i
0.4	0.5	1	100
0.5	0.6	-1	100
0.9	0.4	-1	0
0.7	0.9	-1	0
0.17	0.05	1	0
0.4	0.35	1	0
0.9	0.8	-1	0
0.2	0	1	0

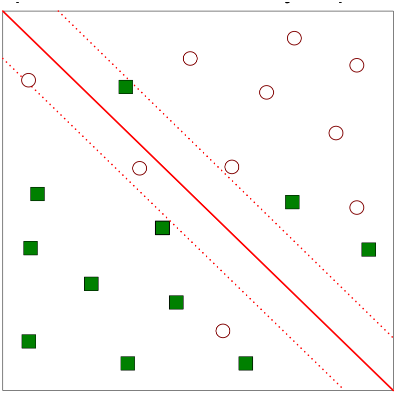
- Solve λ using quadratic programming packages
- $\mathbf{w}^\mathsf{T} = (w_1, w_2)$
    - $w_1 = \sum_{i=1}^{2} \lambda_i y_i x_{i1} = 100 * 1 * 0.4 + 100 * (-1) * 0.5 = -10$
    - $w_2 = \sum_{i=1}^{2} \lambda_i y_i x_{i2} = 100 * 1 * 0.5 + 100 * (-1) * 0.6 = -10$
- $b = 1 - \mathbf{w}^\mathsf{T}\mathbf{x}_1 = 1 - ((-10) * 0.4 + (-10) * (0.5)) = 10$

SVM: given a test data point z

- $y_z = sign(\mathbf{w}^\intercal \mathbf{z} + b) = sign((\sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i^\intercal)\mathbf{z} + b)$
- if $y_z = 1$, the test instance is classified as positive class
- if $y_z = -1$, the test instance is classified as negative class

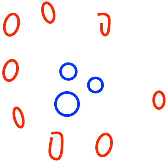
SVM – discussions

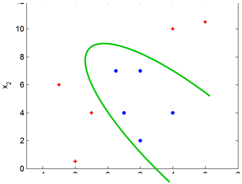
- What is the problem if not linearly separable?



SVM – discussions

- Nonlinear separable





- Do not work in $\mathcal{X}$ space. Transform the data into higher dimensional $\mathcal{Z}$ space such that the data are linearly separable.

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j$$

$\rightarrow$

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{z}_i^{\mathsf{T}} \mathbf{z}_j$$

- Apply linear SVM, support vectors live in $\mathcal{Z}$ space

Learning non-Linear SVM

Optimization problem:

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{z}_i^{\mathsf{T}} \mathbf{z}_j$$

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i)^{\mathsf{T}} \Phi(\mathbf{x}_j)$$

Issues:

- What type of mapping function Φ should be used?
- How to do the computation in high dimensional space?
- Most computations involve dot product $\Phi(\mathbf{x}_i)^{\mathsf{T}}\Phi(\mathbf{x}_j)$

Learning non-Linear SVM

- Define a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$: expressed in terms of the coordinates in the original space
- Kernel trick: $K(\mathbf{x}_i, \mathbf{x}_j) = \underline{\Phi(\mathbf{x}_i)}^{\intercal}\underline{\Phi(\mathbf{x}_j)}$.
- Examples
  - $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\intercal}\mathbf{x}_j + 1)^p$ (Polynomial)
  - $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}}$ (Radial Basis Function (RBF) kernel, also called Gaussian kernel)
  - $K(\mathbf{x}_i, \mathbf{x}_j) = tanh(k\mathbf{x}_i^{\intercal}\mathbf{x}_j - \delta)$ (Sigmoid)

Python – Scikit-learn implementation

- **Parameter kernel='rbf'** to represent the radial basis function kernel Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, rbf will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape(n_samples,n_samples).
- Example

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
svm.fit(X, y)
```

Learning non-Linear SVM

- Advantages of using kernel:
  - Don't have to know the mapping function Φ
  - Computing dot product $\Phi(\mathbf{x}_i)^\intercal\Phi(\mathbf{x}_j)$ in the original space is more efficient.
- Not all functions can be kernels
  - Must make sure there is a corresponding Φ in some high-dimensional space

Characteristics of SVM

- Since the learning problem is formulated as a convex optimization problem, efficient algorithms are available to find the global minima of the objective function (many of the other methods use greedy approaches and find locally optimal solutions)
- Overfitting is addressed by maximizing the margin of the decision boundary, but the user still needs to provide the type of kernel function and cost function
- Difficult to handle missing values
- Robust to noise
- High computational complexity for building the model

References

- Chapter 4: Introduction to Data Mining (2nd Edition) by Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar
- Support Vector Machine (SVM): `https://scikit-learn.org/stable/modules/svm.html`

Substituting, $\mathcal{L}_P(\mathbf{w}, b, \lambda)$ to $\mathcal{L}_D(\lambda)$

- Solving $\mathcal{L}_P(\mathbf{w}, b, \lambda)$ is still difficult because it solves a large number of parameters $\mathbf{w}$, b, and λ_i.
- Idea: Transform Lagrangian into a function of the Lagrange multipliers only by substituting $\mathbf{w}$ and b in $\mathcal{L}_P(\mathbf{w}, b, \lambda)$, we get (the dual problem)

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

  Details: see the next slide.
- It is very nice to have $\mathcal{L}_D(\lambda)$ because it is a simple quadratic form in the vector λ.

Substituting details, get the dual problem $\mathcal{L}_D(\lambda)$

$$\frac{\|\mathbf{w}\|^2}{2} = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} = \frac{1}{2}(\sum_{i=1}^{N}\lambda_i y_i \mathbf{x}_i^\mathsf{T}) \cdot (\sum_{j=1}^{N}\lambda_j y_j \mathbf{x}_j) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j \quad (1)$$

$$-\sum_{i}^{N}\lambda_i(y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - 1) = -\sum_{i=1}^{N}\lambda_i y_i \mathbf{w}^\mathsf{T}\mathbf{x}_i - \sum_{i=1}^{N}\lambda_i y_i b + \sum_{i=1}^{N}\lambda_i \quad (2)$$

$$= -\sum_{i=1}^{N}\lambda_i y_i (\sum_{j=1}^{N}\lambda_j y_j \mathbf{x}_j^\mathsf{T})\mathbf{x}_i + \sum_{i=1}^{N}\lambda_i \quad (3)$$

$$= \sum_{i=1}^{N}\lambda_i - \sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j \quad (4)$$

Add both sides, get

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j$$

Finalized $\mathcal{L}_D(\lambda)$

$$\mathcal{L}_D(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

Maximize w.r.t. each λ
subject to

$\lambda_i \geq 0$ for $i = 1, 2, \cdots, N$
and $\sum_{i=1}^{N} \lambda_i y_i = 0$

Solve $\mathcal{L}_D(\lambda)$ using quadratic programming (QP). We get all the λ_i (Out of the scope).

Solve $\mathcal{L}_D(\lambda)$ – QP

- We are maximizing $\mathcal{L}_D(\lambda)$

$$max_{\lambda}(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j)$$

  subject to constraints

  - (1) $\lambda_i \geq 0$ for $i = 1, 2, \cdots, N$ and
  - (2) $\sum_{i=1}^{N}\lambda_i y_i = 0$.

- Translate the objective to minimization because QP packages generally come with minimization.

$$min_{\lambda}(\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j - \sum_{i=1}^{N}\lambda_i)$$

Solve $\mathcal{L}_D(\lambda)$ – QP

$$min_\lambda(\frac{1}{2}\lambda^\intercal \underbrace{\begin{bmatrix} y_1y_1\mathbf{x}_1^\intercal\mathbf{x}_1 & y_1y_2\mathbf{x}_1^\intercal\mathbf{x}_2 & \cdots & y_1y_N\mathbf{x}_1^\intercal\mathbf{x}_N \\ y_2y_1\mathbf{x}_2^\intercal\mathbf{x}_1 & y_2y_2\mathbf{x}_2^\intercal\mathbf{x}_2 & \cdots & y_2y_N\mathbf{x}_2^\intercal\mathbf{x}_N \\ \cdots & \cdots & \cdots & \cdots \\ y_Ny_1\mathbf{x}_N^\intercal\mathbf{x}_1 & y_Ny_2\mathbf{x}_N^\intercal\mathbf{x}_2 & \cdots & y_Ny_N\mathbf{x}_N^\intercal\mathbf{x}_N \end{bmatrix}}_{\text{quadratic coefficients}} \lambda + \underbrace{(-\mathbf{1})^\intercal}_{\text{linear}}\lambda$$

subject to

$$\underbrace{\mathbf{y}^\intercal\lambda = 0}_{\text{linear constraint}}$$

$$\underbrace{0}_{\text{lower bounds}} \leq \lambda \leq \underbrace{\infty}_{\text{upper bounds}}$$

Let Q represent the matrix with the quadratic coefficients

$min_\lambda(\frac{1}{2}\lambda^\intercal Q\lambda + (-\mathbf{1})^\intercal\lambda)$ subject to $\mathbf{y}^\intercal\lambda = 0$; $\lambda \geq 0$

Quadratic programming packages – Octave

https://www.gnu.org/software/octave/doc/interpreter/Quadratic-Programming.html

Solve the quadratic program

$$min_{\mathbf{x}}(0.5\mathbf{x}^{\mathsf{T}} * H * \mathbf{x} + \mathbf{x}^{\mathsf{T}} * q)$$

subject to

$$\begin{cases} A * \mathbf{x} = & b & \\ lb <= & \mathbf{x} & <= ub \\ A_{lb} <= & A_{in} * \mathbf{x} & <= A_{ub} \end{cases}$$

Quadratic programming packages (MATLAB)

Optimization toolbox in MATLAB:

$$min_{\mathbf{x}}(\frac{1}{2}\mathbf{x}^{\mathsf{T}}H\mathbf{x}+\mathbf{f}^{\mathsf{T}}\mathbf{x})$$

such that

$$\begin{cases} A\cdot\mathbf{x} & \leq b, \\ Aeq\cdot\mathbf{x} & = beq, \\ lb & \leq \mathbf{x} \leq ub. \end{cases}$$