Support Vector Machine

Huy V. Vo

MASSP

2021

Reminder

- Learning problem:
  - Input data: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \mathbb{R}^k$.
  - Approximator: $f_w : \mathbb{R}^d \to \mathbb{R}^k$ where w is the parameters of f. Find w such that f predicts exactly the labels of unseen data, i.e., $f_w(x^{(t)}) = y^{(t)}$ for unseen data point $x^{(t)}$.
  - Loss function $\ell : \mathbb{R}^d \times \mathbb{R}^k \to \mathbb{R}$. $\ell(f_w(x^{(t)}), y^{(t)})$ is small if $f_w(x^{(t)})$ is close to $y^{(t)}$.
  - Test data and train data are supposed to be drawn from the same distribution $\to$ find w such that $f_w(x)$ and y are close on the training data.
  - We solve the optimization problem:

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x^{(i)}), y^{(i)}) + \lambda \Omega(w),$$

    where $\Omega(w)$ is a regularization term on w and its weight λ. Functions ℓ, f_w and Ω are usually chosen convex to ease the optimization.

Reminder

- Linear Regression (LinReg):
  - $w = (w_0, w_1, ..., w_d)$ and $f_w(x) = w_1 x_1 + w_2 x_2 + ... + w_0$.
  - $l(y, y') = \|y - y'\|^2$.
  - $\Omega(w) = \|w\|_{L_2}^2, \|w\|_{L_1}, \ldots$
- Logistic Regression (LogReg):
  - $w = (w_0, w_1, ..., w_d)$ and $f_w(x) = \sigma(w_1 x_1 + w_2 x_2 + ... + w_0)$ where $\sigma(x) = \frac{1}{1+\exp(-x)}$.
  - $l(y, y') = y' \log(y) + (1 - y') \log(1 - y')$.
  - .. or in another way: $f_w(x) = w_1 x_1 + w_2 x_2 + ... + w_0$ and $l(y, y') = y' \log(\sigma(y)) + (1 - y') \log(1 - \sigma(y))$.
  - $\Omega(w) = \|w\|_{L_2}^2, \|w\|_{L_1}, \ldots$
- The approximators of LinReg and LogReg are linear functions. They are linear models.

Support Vector Machine

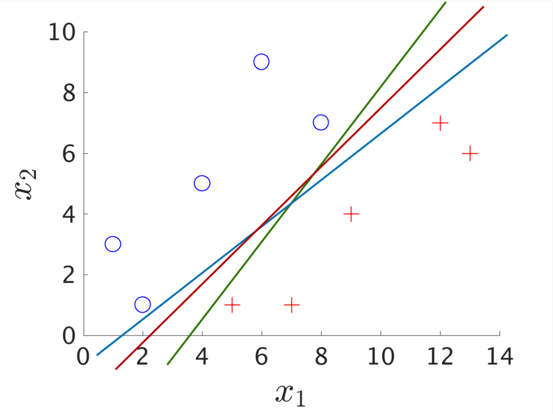
- A linear model: $f_w(x) = \sigma(w_1 x_1 + w_2 x_2 + ... + w_0)$.
- Hinge loss: $\ell(y, y') = \max(0, 1 - yy')$.
- Optimization problem:

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} f_w(x^{(i)})) + \lambda \|w\|^2,$$

- This is a convex optimization problem. It can be solved with appropriate solvers.
- However...
  - What are support vectors?
  - Why bothers with yet another loss function (Hinge loss)?

Toy dataset

- Data: $x^{(i)} \in \mathbb{R}^2, y^{(i)} \in \{\pm 1\}$.



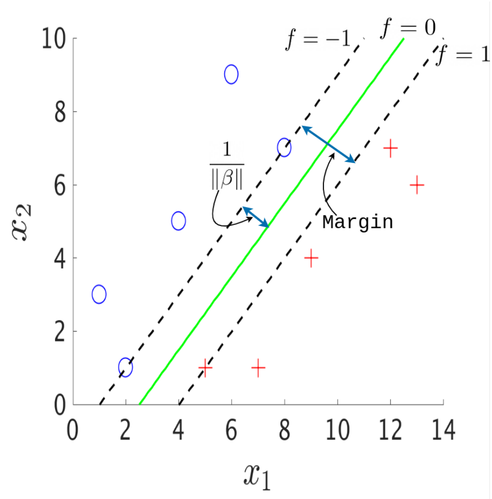
x_1	x_2	y
1	3	-1
2	1	-1
4	5	-1
6	9	-1
8	7	-1
5	1	1
7	1	1
9	4	1
12	7	1
13	6	1

Outline

- Maximum margin problem:
  - Linearly separable data: primal and dual problems.
  - Real data: primal and dual problems with slack variables.
- Non-linear classification with SVM: Kernel trick.
- Annex: Mathematical materials.

Maximum margin problem: Linearly separable case

- Furthest boundary from both negative and positive sets.
- Distance to the boundary: $d(x) = \frac{|f(x)|}{\|\beta\|} = \frac{|\langle x,\beta\rangle + \beta_0|}{\|\beta\|}$.
- Distance from the support vectors: $\frac{1}{\|\beta\|}$.
- If $y_i = -1$ then $f(x_i) \leq -1$, if $y_i = 1$ then $f(x_i) \geq 1$ $\rightarrow y_i f(x_i) \geq 1, \forall\, i$.



Maximum margin problem: Linearly separable case

- Primal problem:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$
$$\text{s.t. } y_i(\langle x_i, \beta\rangle + \beta_0) \geq 1 \,\forall\, i.$$

  - Quadratic programming problem, no literal solution, can solve with numerical optimization programs.
  - Example on Python with the toy dataset:

$$\beta = [-\frac{2}{3}, \frac{2}{3}], \beta_0 = \frac{5}{3}$$

```
# !pip install cvxopt
# !pip install qpsolvers
from qpsolvers import solve_qp
import numpy as np
n = 10
X = np.array([[1,3],[2,1],[4,5],[6,9],
              [8,7],[5,1],[7,1],
              [9,4],[12,7],[13,6]]);
X = np.hstack((X, np.ones((10,1))))
y = np.ones((n,1));
y[:5] = -1
G = X*y
h = -np.ones((n,))

P = np.array([[1,0,0],[0,1,0],[0,0,0]],
            dtype=np.float64)
q = np.zeros((3,))

solve_qp(P,q,G,h,solver='cvxopt')
```

```
array([-0.66666668,  0.66666671,  1.66666666])
```

Maximum margin problem: Linearly separable case

- Primal problem:
  - Algorithms for numerical approximation is slow if p is large.
  - The role of support vectors is not clear.
  - Cannot use kernel trick.
- Dual problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

$$\text{s.t. } \alpha_i \geq 0 \, \forall \, i \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0.$$

  - Karush-Kuhn-Tucker (KKT) condition: $\alpha_i [y_i(\langle x, \beta \rangle + \beta_0) - 1] = 0 \, \forall \, i$.
  - Retrieve β: $\beta = \sum_{i=1}^{n} \alpha_i y_i x_i$.
  - Inferencce: $f(x) = \langle x, \beta \rangle + \beta_0 = \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle + \beta_0$.

Maximum margin problem: Linearly separable case

- Dual problem:
  - Example on Python with the toy dataset:

$$\alpha = [0, \frac{1}{3}, 0, 0, \frac{1}{9}, \frac{4}{9}, 0, 0, 0, 0]$$

  - $\beta = \alpha_2 y_2 x_2 + \alpha_5 y_5 x_5 + \alpha_6 y_6 x_6 = [\frac{2}{3}, -\frac{2}{3}]$
  - The points are weighted by α. Only support vectors have a positive α.

```python
# !pip install cvxopt
# !pip install qpsolvers
from qpsolvers import solve_qp
import numpy as np
np.set_printoptions(precision=5)

n = 10
X = np.array([[1,3],[2,1],[4,5],[6,9],
              [8,7],[5,1],[7,1],
              [9,4],[12,7],[13,6]]);
y = np.ones((n,1));
y[:5] = -1
X = X*y
P = np.matmul(X,X.T);
q = -np.ones((n,))

G = -np.eye(n)
h = np.zeros((n,))

A = y.reshape((1,n))
b = np.array([0.0])

alpha = solve_qp(P,q,G,h,A,b,solver='cvxopt')

for el in alpha[:5]:
    print('{:.3f}'.format(el), end=' ')
print()
for el in alpha[5:]:
    print('{:.3f}'.format(el), end=' ')
```

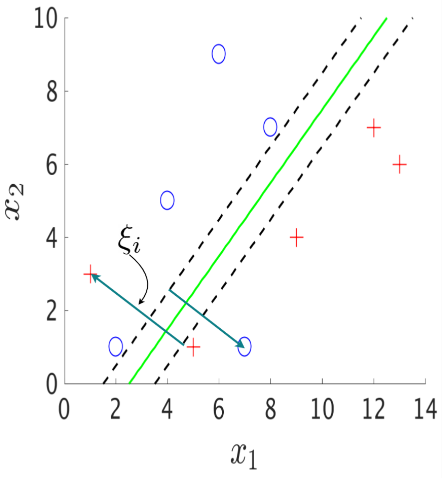
```
0.000 0.333 0.000 0.000 0.111
0.444 0.000 0.000 0.000 0.000
```

Maximum margin problem: Linearly separable case

- Dual problem:
  - The role of support vectors is clear (KKT).
  - The optimization does not depend on data dimension $\rightarrow$ can deal with high-dimension data (text, images, ...).
  - Bothe the optimization and the inference do not depend on the vectors, only their inner products $\rightarrow$ kernel trick.

Maximum margin problem: Real data

- Misclassified error ξ_i: The amount needed to move x_i to the right side.
- If $y_i = 1$, we need $\langle x_i, \beta \rangle + \beta_0 + \xi_i \geq 1$ or $y_i(\langle x_i, \beta \rangle + \beta_0) + \xi_i \geq 1$.
- If $y_i = -1$, we need $\langle x, \beta_i \rangle + \beta_0 - \xi_i \leq -1$ or $y_i(\langle x_i, \beta \rangle + \beta_0) + \xi_i \geq 1$.
- Maximize the margin while also minimizing the total misclassified error.



Maximum margin problem: Real data

- Primal problem:

$$\min_{\beta,\beta_0,\xi_i} \frac{1}{2}\|\beta\|^2 + \sum_{i=1}^{n} \xi_i \quad \text{s.t. } y_i(\langle x, \beta\rangle + \beta_0) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \, \forall\, i.$$

  Note that this formulation is equivalent to the formulation with Hinge loss.

- Dual problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle$$

$$\text{s.t. } \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C \, \forall\, i,$$

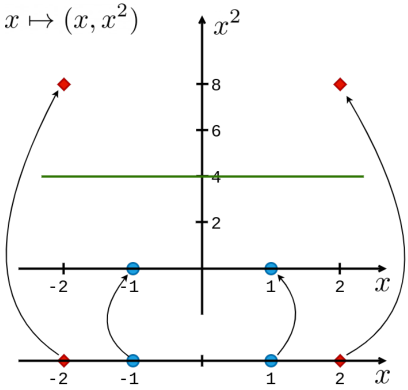
  where C is the tolerance for errors.

Nonlinear classification

- Some data is not linearly separable in its original space but is in another projected space $\rightarrow$ we can use SVM in the latter.
- Finding a suitable projected space is difficult.
- Sometimes, the projected space is of infinite dimension, e.g.,

$$x \mapsto g_x(u) = \begin{cases} u \text{ if } u \leq x \\ x \text{ otherwise} \end{cases}$$

with $x \in [0, 1]$ and $\langle g_x, g_y \rangle = \int_0^1 g_x'(u) g_y'(u) du$.



Kernels

- Given a projection $g(x)$, the inner product $K(x,y) = \langle g(x), g(y) \rangle$ suffices. K is called a kernel.
- Kernel trick: Instead of finding suitable g, we find suitable K.
- Dual problem becomes

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

$$\text{s.t. } \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C \,\forall\, i,$$

Kernels

- Polynomial kernels: $K(x, y) = (x^T y)^n$.
- Exponential kernels: $K(x, y) = \exp x^T y$.
- Composite kernels: $K_1 + K_2,\ K_1 K_2,\ cK_1,\ \exp K_1,\ \ldots$
- RBF: $K(x, y) = \exp -\dfrac{\|x - y\|^2}{2\sigma^2}$.
- Scikit-learn:

sklearn.svm.SVC

class `sklearn.svm.`**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

Annex: Mathematical materials