

# Optimization Project: Support Vector Machine

K. Kamtue & Cl. Réda

ENS Cachan

January 12th, 2017

## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

## 1 Project description

### ■ Project

- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

# Project

## Support Machine Vector

### Objective

**Classify data**

# Project

## Support Machine Vector

### Objective

Classify data

- Applied to **binary classification** ( $y_i \in \{1, -1\}$ );

# Project

## Support Machine Vector

### Objective

Classify data

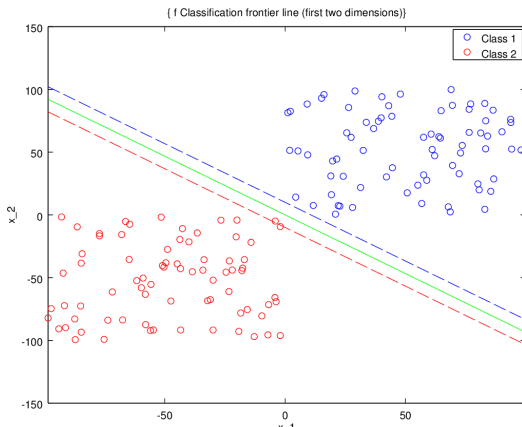
- Applied to **binary classification** ( $y_i \in \{1, -1\}$ );
- Looking for a **hyperplane**  $f : x \rightarrow \omega^T x (+b)$  such as:

$$\forall i, f(x_i) = \begin{cases} < 0 & \text{si } y_i = -1 \\ > 0 & \text{si } y_i = 1 \end{cases} \Leftrightarrow \forall i, y_i \times f(x_i) > 0 \quad (1)$$

# Project

## Support Machine Vector

Figure: Example with two classes (**red** and **blue**)



# Optimization problem

Looking for the optimization problem

## Naive optimization problem

$\gamma$ : distance between the lines  $f(x) = 1$  and  $f(x) = -1$ .



# Optimization problem

Looking for the optimization problem

## Naive optimization problem

$\gamma$ : distance between the lines  $f(x) = 1$  and  $f(x) = -1$ .

$$\begin{aligned} \max_{\omega} \quad & \gamma = \frac{2}{\|\omega\|} \\ \text{subject to} \quad & \forall i, y_i \times f(x_i) > 0 \end{aligned}$$

# Optimization problem

Looking for the optimization problem

## Naive optimization problem

$\gamma$ : distance between the lines  $f(x) = 1$  and  $f(x) = -1$ .

$$\begin{aligned} \max_{\omega} \gamma &= \frac{2}{\|\omega\|} \\ \text{subject to } \forall i, y_i \times f(x_i) &> 0 \end{aligned}$$

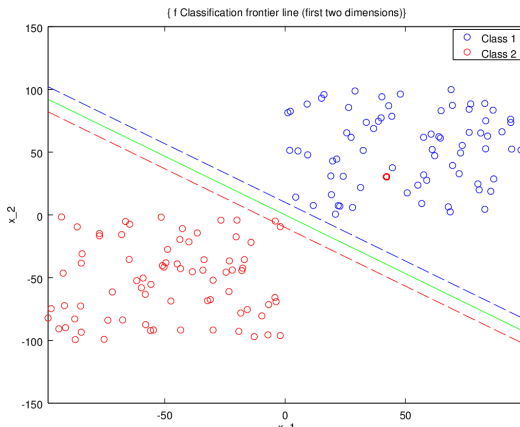
$$\begin{aligned} \Leftrightarrow \min_{\omega} \frac{1}{2} \|\omega\|^2 \\ \text{subject to } \forall i, y_i \times f(x_i) &> 0 \end{aligned}$$

**Beware:** if the data set is not linearly separable!

# Optimization

Looking for the optimization problem

Figure: Example with two classes (**red** and **blue**)



# Optimization problem

Adapting the problem to **non-separable** sets

Let  $z_i$  be  $\max(0, 1 - y_i \times f(x_i))$  (**Hinge loss**).

# Optimization problem

Adapting the problem to **non-separable** sets

Let  $z_i$  be  $\max(0, 1 - y_i \times f(x_i))$  (**Hinge loss**).

Having the problem **convex** and always **feasible**

Penalty for **classification errors** with  $(z_i)_i$  and  $C$  :

$$\begin{aligned} \min_{\omega, z} \quad & \frac{1}{2} \|\omega\|^2 + C \sum_{i \leq m} z_i \\ \text{subject to} \quad & \\ & \forall i, z_i \geq 0 \\ & \forall i, y_i \times (\omega^T x_i) \geq 1 - z_i \end{aligned}$$

## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

# Implementation

Solving the optimization problem

- Use **Newton's method**:

Reminder: Update of  $x$  with **Newton's method**

$$x_{n+1} \leftarrow x_n + s \times \nabla^2 \text{obj}(x_n)^{-1} \nabla \text{obj}(x_n)$$

(finding **step size** value  $s$  by **backtracking line search**)

# Implementation

Solving the optimization problem

- Use **Newton's method**:

Reminder: Update of  $x$  with **Newton's method**

$$x_{n+1} \leftarrow x_n + s \times \nabla^2 \text{obj}(x_n)^{-1} \nabla \text{obj}(x_n)$$

(finding **step size** value  $s$  by **backtracking line search**)

- Make the problem independent from **dimension**;



# Implementation

Solving the optimization problem

- Use **Newton's method**:

Reminder: Update of  $x$  with **Newton's method**

$$x_{n+1} \leftarrow x_n + s \times \nabla^2 \text{obj}(x_n)^{-1} \nabla \text{obj}(x_n)$$

(finding **step size** value  $s$  by **backtracking line search**)

- Make the problem independent from **dimension**;
- Use **logarithmic barrier method**.

# Implementation

Independence from dimension: **dual problem**

After Lagrangian calculation and minimization in  $\omega$ :

# Implementation

Independence from dimension: **dual problem**

After Lagrangian calculation and minimization in  $\omega$ :

## Dual problem

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^{+m}} & -\frac{1}{2} \left\| \sum_i \lambda_i y_i x_i \right\|_2^2 + \mathbf{1}^T \lambda \\ \text{subject to } & \forall i, 0 \leq \lambda_i \leq C \\ & \text{(KKT conditions)} \end{aligned}$$

# Implementation

Independence from dimension: **dual problem**

After Lagrangian calculation and minimization in  $\omega$ :

## Dual problem

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^{+m}} & -\frac{1}{2} \left\| \sum_i \lambda_i y_i x_i \right\|_2^2 + \mathbf{1}^T \lambda \\ \text{subject to } & \forall i, 0 \leq \lambda_i \leq C \\ & \text{(KKT conditions)} \end{aligned}$$

## Get primal solution from dual solution

$$\omega^* = \sum_i \lambda_i^* y_i x_i$$

# Implementation

Make the problem independant from dimension

Use the **kernel trick** :

## Dual problem

Let  $K$  be  $X^T X$  (linear kernel):

$$\begin{aligned} \max \quad & -\frac{1}{2} \lambda^T \text{diag}(y) K \text{diag}(y) \lambda + \mathbf{1}^T \lambda \\ \text{subject to} \quad & \forall i, 0 \leq \lambda_i \leq C \end{aligned}$$

# Implementation

Delete inequality constraints

Use the **logarithmic barrier method** :

# Implementation

Delete inequality constraints

Use the **logarithmic barrier method** :

Barrier function

$$\begin{aligned}\Phi(\lambda) &= \sum_i (-\log(C - \lambda_i) - \log(\lambda_i)) \\ &= -\sum_i \log((C - \lambda_i)\lambda_i)\end{aligned}$$

# Implementation

Delete inequality constraints

Use the **logarithmic barrier method** :

Barrier function

$$\begin{aligned}\Phi(\lambda) &= \sum_i (-\log(C - \lambda_i) - \log(\lambda_i)) \\ &= -\sum_i \log((C - \lambda_i)\lambda_i)\end{aligned}$$

Final optimization problem

$$\max -\frac{1}{2}\lambda^T \text{diag}(y)K\text{diag}(y)\lambda + \mathbf{1}^T \lambda + \Phi(\lambda)$$



## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

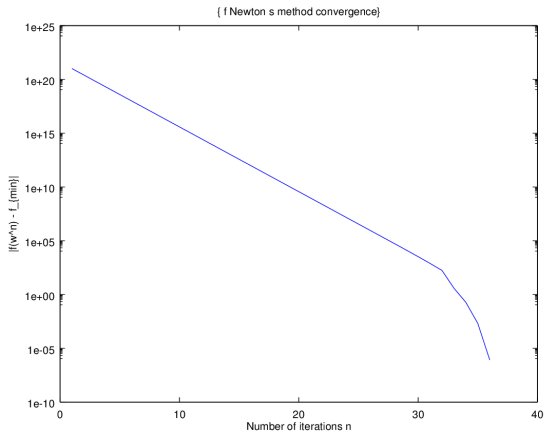
## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

# Testing the implementation

## Newton's method convergence



# Testing the implementation

Dependence on the sample size

Table: Time complexity dependence

Set	C	d	n	Iteration number	Time (s)
1	5	40000	10	11	0.315
1	5	40	100	12	0.715
1	5	40	1000	large	> 1,000

# Testing the implementation

Performance in function of  $C$

Performed on the same sample set:

**Table:** Computation time & Performance in function of  $C$

$C$	Time (s)	Training Error	Val Error	Test Error
1	132.15	6	2	3
10	0.74	6	2	3
100	0.89	1	12	3

## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

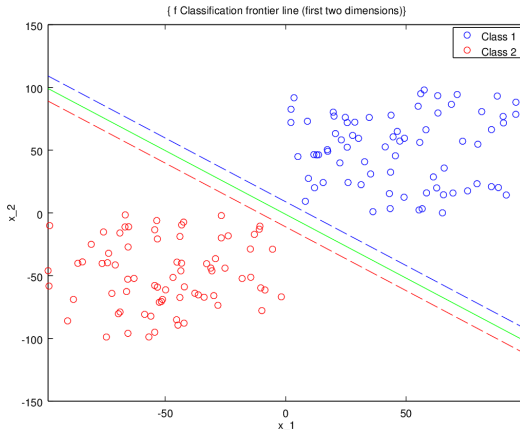
- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

# Plotting the **classification frontier**

For  $C = 5$ ,  $n = 150$ ,  $d = 200$

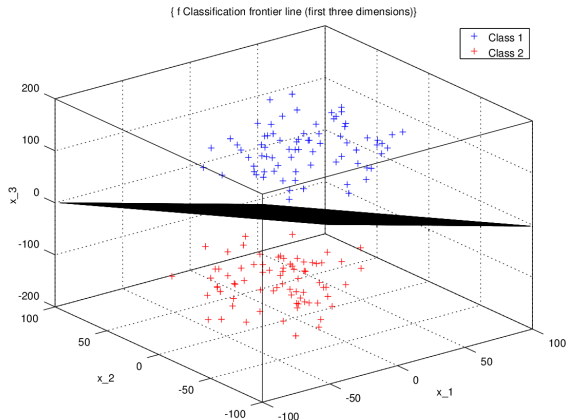
Normalized points with Gaussian distribution (2D):



# Plotting the **classification frontier**

For  $C = 5$ ,  $n = 150$ ,  $d = 200$

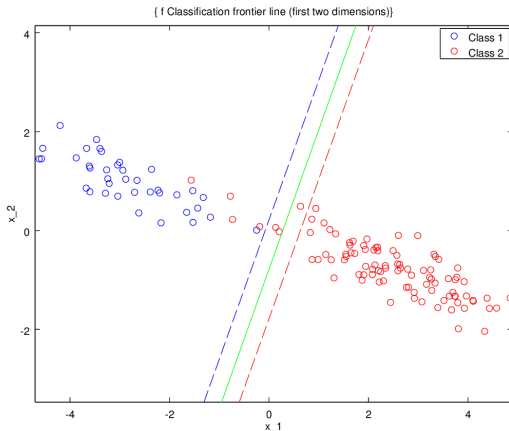
Normalized points with Gaussian distribution (2D):



# Plotting the **classification frontier**

Pour  $C = 5, n = 100, d = 2$

Generation with Gaussian distribution (2D) (set A) :

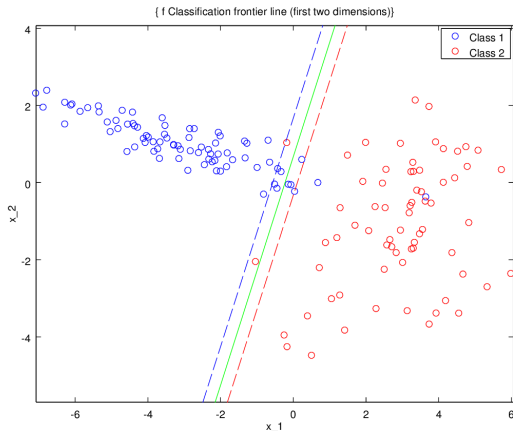




# Plotting the **classification frontier**

Pour  $C = 5, n = 100, d = 2$

Generation with Gaussian distribution (2D) (set B) :



## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- **Cross Validation**
- Coordinate Descent
- ACCPM

## 4 Demo

# Extensions

## Cross Validation

- **Cross validation** (choice of the best value for  $C$ );

# Extensions

## Cross Validation

- **Cross validation** (choice of the best value for  $C$ );

### Leave-one-out technique

Having a sample size of size  $n$ , for each value of  $C$  to test:

- 1 for  $i \in [1, n]$
- 2 **Leave out sample  $i$**
- 3 Train the SVM on other samples
- 4 Test the SVM on sample  $i$
- 5 Get the **Mean-Squared Error** for the  $n$  loops
- 6 If it is the minimum MSE computed so far
- 7 Then update the best value of  $C$

## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

# Extensions

## Coordinate Descent

- Implementation of **Coordinate Descent**;

# Extensions

## Coordinate Descent

### ■ Implementation of **Coordinate Descent**;

#### Reminder: Coordinate Descent

for  $i, j \in [1, d]$ , and iteration  $k$

$$\begin{aligned} a_i^{k+1} &= \operatorname{argmin}_{a_i} f(a_1, a_2, \dots, a_i, \dots, a_d) \\ a_j^{k+1} &= a_j^k \text{ for } j \neq i \end{aligned}$$

# Extensions

## Coordinate Descent results

Performed on the same sample set (as in the testing of the original SVM):

**Table:** Computation time & Performance in function of C

C	Time (s)	Training Error	Val Error	Test Error
1	0.37	11	6	5.33
10	0.34	5	4	3.39
100	0.29	1	8	3.56
10,000	0.29	6	4	2.76



## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

# Extensions

## ACCPM

- Implementation of **Analytic Center Cutting-Plane Method**;

# Extensions

## ACCPM

- Implementation of **Analytic Center Cutting-Plane Method**;

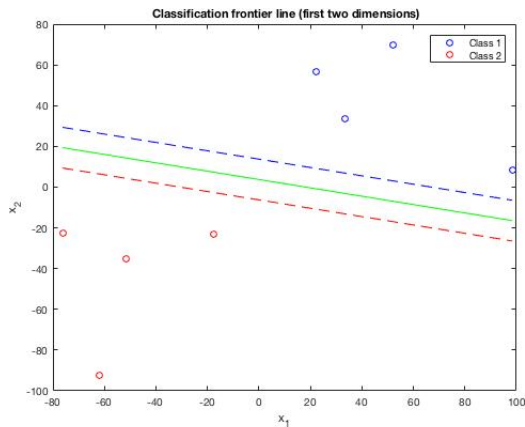
### Reminder: ACCPM

- 1 Compute the **analytic center of constraint polyhedron**
- 2 Compute the objective value and the gradient
- 3 While objective value is evolving greatly enough
- 4 Add an inequality to constraint polyhedron
- 5 Optional: **Constraint Dropping**

# Extensions

## ACCPM results

Figure: For data of size 8, and dimension 2



## 1 Project description

- Project
- Optimization problem
- Implementation

## 2 Results

- Testing the implementation
- Plotting the classification frontier

## 3 Extensions

- Cross Validation
- Coordinate Descent
- ACCPM

## 4 Demo

## Demo of the SVM