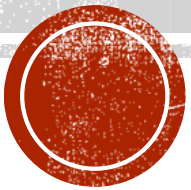
LINEAR MODELS: PART 2



CS 412 Introduction to Machine Learning

Prof. Zheleva

February 20, 2017

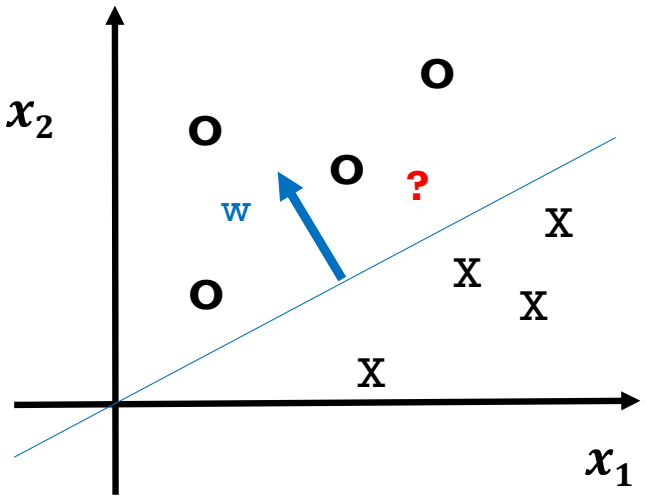
Reading Assignment: CIML: 7

BINARY CLASSIFICATION VIA HYPERPLANES

- A classifier is a hyperplane that separates positive from negative examples (**w**,b)
- At test time, we check on what side of the hyperplane examples fall

$$\hat{y} = sign(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b})$$

- This is a linear classifier model
  - Because the prediction is a linear combination of feature values x
  - Perceptron is one example



LEARNING A LINEAR CLASSIFIER AS AN OPTIMIZATION PROBLEM

- Structural risk minimization framework
  - Tradeoff between low training error and a solution that is simple

$$\min_{\boldsymbol{w},b} L(\boldsymbol{w},b) = \min_{\boldsymbol{w},b}\left(\sum_{n=1}^{N} l(y_n, \boldsymbol{w}\cdot x_n + b) + \lambda R(\boldsymbol{w},b)\right)$$

Objective function

Loss function
Measures how well classifier fits training data

Regularizing function
Helps with avoiding overfitting
Penalizes complex functions
λ is a hyperparameter

Variables over which we are optimizing the objective function

APPROXIMATING THE 0/1 LOSS WITH CONVEX SURROGATE LOSS FUNCTIONS

- Examples (with b=0)

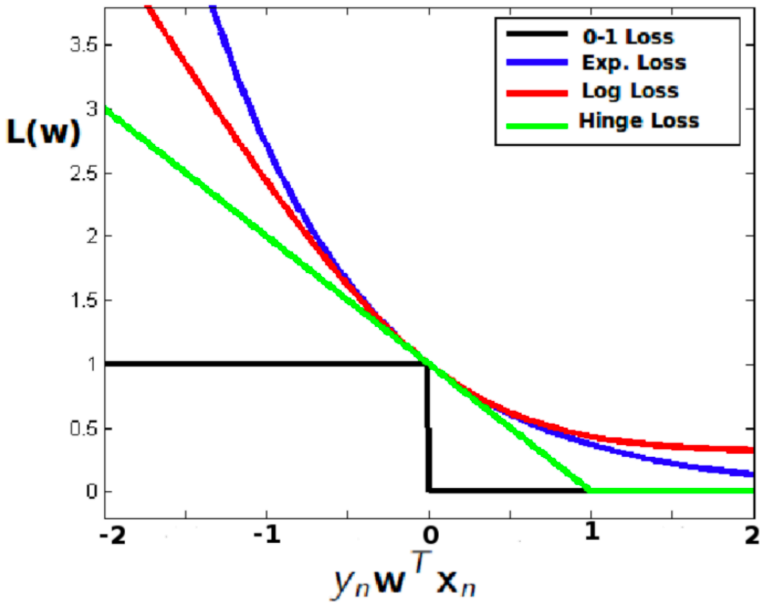
$$\text{Zero/one:} \quad \ell^{(0/1)}(y,\hat{y}) = \mathbf{1}[y\hat{y} \leq 0]$$

$$\text{Hinge:} \quad \ell^{(\text{hin})}(y,\hat{y}) = \max\{0, 1 - y\hat{y}\}$$

$$\text{Logistic:} \quad \ell^{(\log)}(y,\hat{y}) = \frac{1}{\log 2} \log\left(1 + \exp[-y\hat{y}]\right)$$

$$\text{Exponential:} \quad \ell^{(\exp)}(y,\hat{y}) = \exp[-y\hat{y}]$$

$$\text{Squared:} \quad \ell^{(\text{sqr})}(y,\hat{y}) = (y - \hat{y})^2$$



- All are convex upper bounds on 0/1 loss
- Some smooth, some more sensitive to outliers

GRADIENT DESCENT ALGORITHM

$\mathcal{F}$: Objective function to minimize
K: Number of steps
η_1: first step size

Algorithm 21 GRADIENTDESCENT($\mathcal{F}$, K, η_1, ...)

1: $z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing
2: **for** $k = 1 \ldots K$ **do**
3: $g^{(k)} \leftarrow \nabla_z \mathcal{F}\big|_{z^{(k-1)}}$ // compute gradient at current location
4: $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient
5: **end for**
6: **return** $z^{(K)}$

NOW LET'S CALCULATE GRADIENTS FOR MULTIVARIATE OBJECTIVES

- Consider the following learning objective
  - Loss function: Logistic loss
  - Regularizer: Squared l_2 norm

$$\min_{\boldsymbol{w},b} L(\boldsymbol{w},b) = \min_{\boldsymbol{w},b} \left(\sum_{n=1}^{N} exp(-y_n(\boldsymbol{w} \cdot x_n + b)) + \frac{\lambda}{2} \|w\|_2^2 \right)$$

- What do we need to do to run gradient descent?
  - Find gradient for w and b

TODAY: LINEAR MODELS CONTD.

- Subgradients
- Closed form solutions to optimization problems
- (Linear) Support Vector Machines

SUBGRADIENTS

- Problem: some objective functions are not differentiable everywhere
  - Hinge loss (at $y\hat{y}=1$)
  - l_1 norm (when $w_d=0$)
- Solution: subgradient optimization
  - Let's ignore the problem, and just try to apply gradient descent anyway!!
  - We will just differentiate by parts...
    - Where function is differentiable, take the derivative
    - Where function is not differentiable, the subderivative is the set of all possible derivative-like lines

SUBGRADIENT OF HINGE LOSS

$$\partial_w \max\{0, 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\} \tag{7.24}$$

$$= \partial_w \begin{cases} 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \tag{7.25}$$

$$= \begin{cases} \partial_w 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ \partial_w 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \tag{7.26}$$

$$= \begin{cases} \mathbf{0} & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ -y_n \boldsymbol{x}_n & \text{otherwise} \end{cases} \tag{7.27}$$

SUBGRADIENT DESCENT OF HINGE LOSS

Algorithm 22 HINGEREGULARIZEDGD($\mathbf{D}$, λ, $MaxIter$)

1: $\boldsymbol{w} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $b \leftarrow 0$ // initialize weights and bias
2: **for** $iter = 1 \ldots MaxIter$ **do**
3: $\boldsymbol{g} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $g \leftarrow 0$ // initialize gradient of weights and bias
4: **for all** $(\boldsymbol{x}, y) \in \mathbf{D}$ **do**
5: **if** $y(\boldsymbol{w} \cdot \boldsymbol{x} + b) \leq 1$ **then**
6: $\boldsymbol{g} \leftarrow \boldsymbol{g} + y\,\boldsymbol{x}$ // update weight gradient
7: $g \leftarrow g + y$ // update bias derivative
8: **end if**
9: **end for**
10: $\boldsymbol{g} \leftarrow \boldsymbol{g} - \lambda \boldsymbol{w}$ // add in regularization term
11: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{g}$ // update weights
12: $b \leftarrow b + \eta g$ // update bias
13: **end for**
14: **return** $\boldsymbol{w}, b$

WHAT IS THE PERCEPTRON OPTIMIZING?

Algorithm 5 PERCEPTRONTRAIN(**D**, *MaxIter*)

```
1:  w_d ← 0, for all d = 1 ... D                          // initialize weights
2:  b ← 0                                                 // initialize bias
3:  for iter = 1 ... MaxIter do
4:      for all (x,y) ∈ D do
5:          a ← Σ_{d=1}^{D} w_d x_d + b                   // compute activation for this example
6:          if ya ≤ 0 then
7:              w_d ← w_d + y x_d, for all d = 1 ... D    // update weights
8:              b ← b + y                                 // update bias
9:          end if
10:     end for
11: end for
12: return w_0, w_1, ..., w_D, b
```

Loss function is a variant of the hinge loss
$\max(0, y_n(\boldsymbol{w}^T \boldsymbol{x_n} + \boldsymbol{b}))$

CLOSED FORM OPTIMIZATION

- Consider the following learning objective
  - Loss function: Squared error loss
  - Regularizer: Squared l_2 norm

$$\min_{\boldsymbol{w},b} L(\boldsymbol{w}, b) = \min_{\boldsymbol{w},b} \left(\frac{1}{2} \sum_{n=1}^{N} \left(y_n - (\boldsymbol{w} \cdot x_n + b) \right)^2 + \frac{\lambda}{2} \|w\|_2^2 \right)$$

- Gradient descent would work
  - Can we do better?

OPTIMIZATION AS MATRIX OPERATIONS

- Training data is matrix **X** of size *NxD*
  - $X_{n,d}$: value of *d*th feature on *n*th example
- Labels represented as a column vector **Y** of dimension *N*
- Weights represented as a column vector **w** of size *D*
- Matrix-vector product **a=Xw** has dimension *N (bias omitted)*
- Predictions

$$a_n = [\boldsymbol{X}\boldsymbol{w}]_n = \sum_d \boldsymbol{X}_{n,d} w_d$$

EXPANDED MATRIX MULTIPLICATION

$$\widehat{\boldsymbol{Y}} = [\boldsymbol{X}\boldsymbol{w}]_n = \sum_d \boldsymbol{X}_{n,d} w_d$$

$$\underbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,D} \\ x_{2,1} & x_{2,2} & \dots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,D} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} \sum_d x_{1,d} w_d \\ \sum_d x_{2,d} w_d \\ \vdots \\ \sum_d x_{N,d} w_d \end{bmatrix}}_{\hat{Y}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\hat{Y}}$$

USE CALCULUS TO MINIMIZE

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \min_{\boldsymbol{w}} \left(\frac{1}{2} \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{Y} \|^2 + \frac{\lambda}{2} \|w\|_2^2\right)$$

- Take gradient

$$\begin{aligned}\nabla_w \mathcal{L}(w) &= \mathbf{X}^\top (\mathbf{X}w - \mathbf{Y}) + \lambda w \\ &= \mathbf{X}^\top \mathbf{X}w - \mathbf{X}^\top \mathbf{Y} + \lambda w \\ &= \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right) w - \mathbf{X}^\top \mathbf{Y}\end{aligned}$$

- Set derivative to 0

$$\begin{aligned}&\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right) w - \mathbf{X}^\top \mathbf{Y} = 0 \\ \iff &\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right) w = \mathbf{X}^\top \mathbf{Y} \\ \iff &w = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right)^{-1} \mathbf{X}^\top \mathbf{Y}\end{aligned}$$

- Optimal weights computed by a few matrix multiplications and inverses!

MATH REVIEW | MATRIX MULTIPLICATION AND INVERSION

If $\mathbf{A}$ and $\mathbf{B}$ are matrices, and $\mathbf{A}$ is $N \times K$ and $\mathbf{B}$ is $K \times M$ (the inner dimensions must match), then the matrix product $\mathbf{AB}$ is a matrix $\mathbf{C}$ that is $N \times M$, with $\mathbf{C}_{n,m} = \sum_k \mathbf{A}_{n,k}\mathbf{B}_{k,m}$. If $\boldsymbol{v}$ is a vector in $\mathbb{R}^D$, we will treat is as a *column vector*, or a matrix of size $D \times 1$. Thus, $\mathbf{A}\boldsymbol{v}$ is well defined if $\mathbf{A}$ is $D \times M$, and the resulting product is a vector $\boldsymbol{u}$ with $u_m = \sum_d \mathbf{A}_{d,m} v_d$.

Aside from matrix product, a fundamental matrix operation is inversion. We will often encounter a form like $\mathbf{A}\boldsymbol{x} = \boldsymbol{y}$, where $\mathbf{A}$ and $\boldsymbol{y}$ are known and we want to solve for $\mathbf{A}$. If $\mathbf{A}$ is square of size $N \times N$, then the inverse of $\mathbf{A}$, denoted $\mathbf{A}^{-1}$, is also a square matrix of size $N \times N$, such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}_N = \mathbf{A}^{-1}\mathbf{A}$. I.e., multiplying a matrix by its inverse (on either side) gives back the identity matrix. Using this, we can solve $\mathbf{A}\boldsymbol{x} = \boldsymbol{y}$ by multiplying both sides by $\mathbf{A}^{-1}$ on the left (recall that order matters in matrix multiplication), yielding $\mathbf{A}^{-1}\mathbf{A}\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{y}$ from which we can conclude $\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{y}$. Note that not all square matrices are invertible. For instance, the all zeros matrix does not have an inverse (in the same way that $1/0$ is not defined for scalars). However, there are other matrices that do not have inverses; such matrices are called **singular**.

CLOSED FORM VS. GRADIENT DESCENT

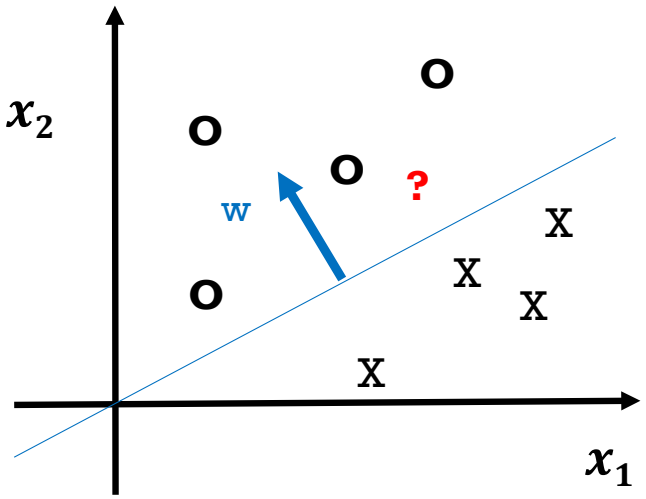
- Matrix solution gives exact solution (no iterations!)
- Gradient descent gives progressively better solutions
  - Converges to the optimum at a rate 1/k
- Is the exact solution always more efficient?
  - One step of gradient descent takes O(ND), K steps take O(KND)
  - Matrix solution takes $O(D^3+D^2N)$ (when N>D, then $O(D^2N)$)
    - reconstructing X^TX takes $O(D^2N)$,
    - inversion takes $O(D^3)$
    - final multiplication takes O(ND)
  - If K>D, then matrix solution is more efficient

RECAP: CLASSIFICATION VIA HYPERPLANES

- A classifier is a hyperplane that separates positive from negative examples (**w**,b)
- At test time, we check on what side of the hyperplane examples fall

$$\hat{y} = sign(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b})$$

- This is a linear classifier model
  - Because the prediction is a linear combination of feature values x
  - Perceptron is one example



REVISITING DATASET MARGINS

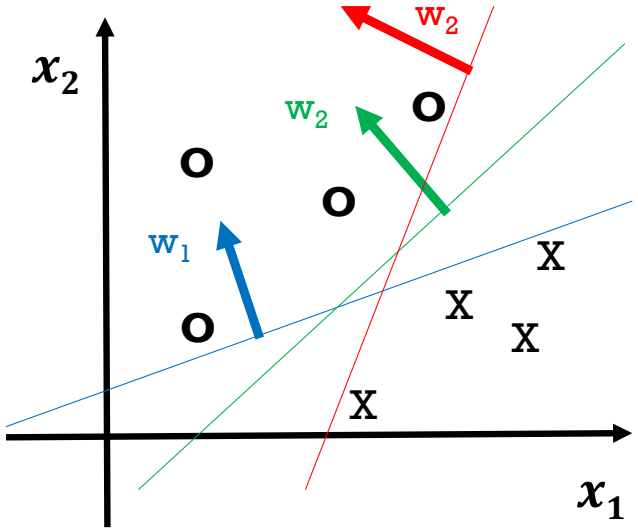
- Hyperplane margin on D: Distance between the hyperplane (w,b) and the nearest point in D

$$\text{margin}(\mathbf{D}, \mathbf{w}, \mathrm{b}) = \begin{cases} \min_{(x,y)\in \mathrm{D}} y(\boldsymbol{w}\cdot\boldsymbol{x}+b) & if\ \boldsymbol{w}\ separates\ \boldsymbol{D} \\ -\infty & otherwise \end{cases}$$

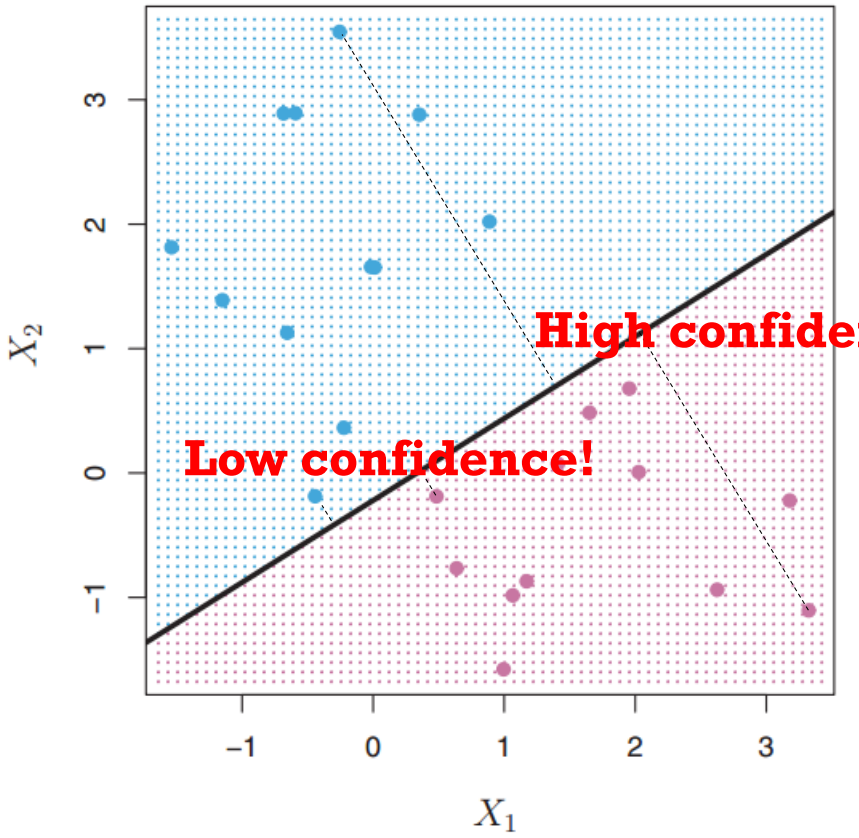
- Dataset margin: Largest attainable margin on D

$$\text{margin}(\mathbf{D}) = \sup_{w,b} \text{margin}(\mathbf{D}, \mathbf{w}, \mathrm{b})$$

- Which hyperplane is best?



MARGIN AS "CONFIDENCE" OF PREDICTIONS

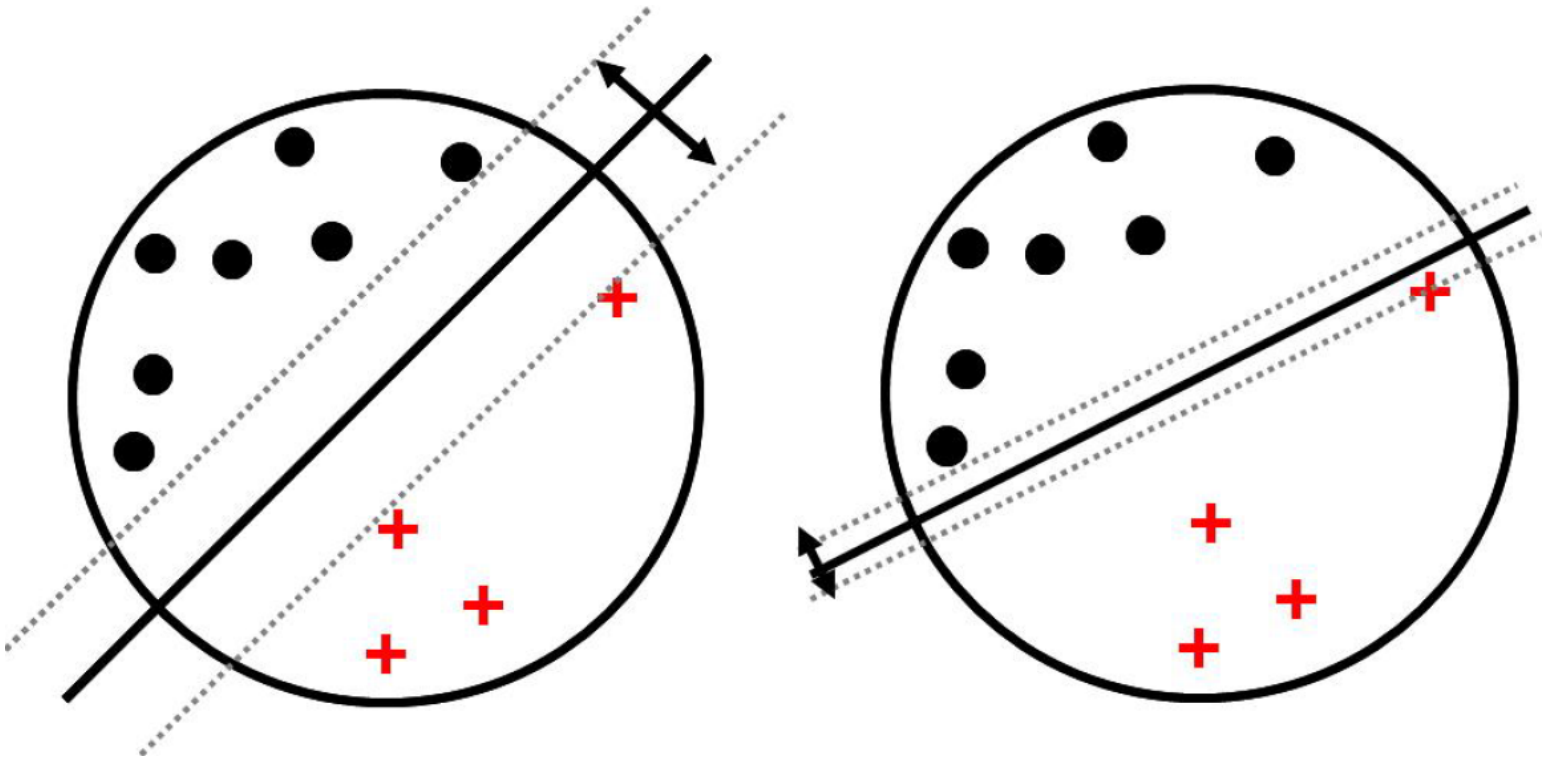


"Confidence" =

$$y_n(w_1 x_{n1} + \cdots w_d x_{nd} + b)$$

Margin is the minimum "confidence"

Let's maximize this!

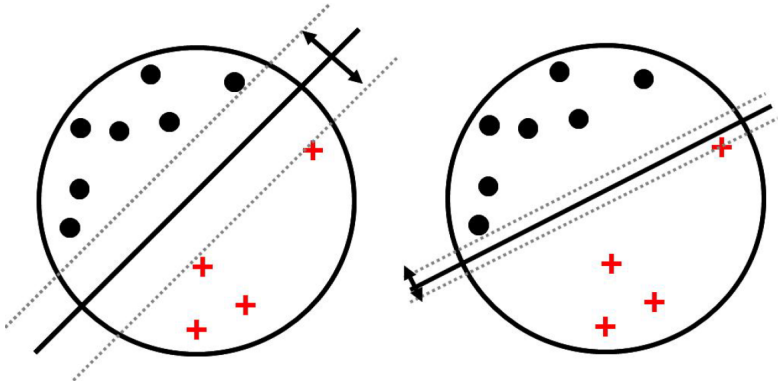


SUPPORT VECTOR MACHINES

- Goal: find a hyperplane with largest possible margin
- Constrained optimization problem:

$$\min_{w,b} \frac{1}{\gamma(w,b)}$$

Subject to $y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 \ \ (\forall n)$



- What's the problem with this formulation?

HARD-MARGIN VS SOFT-MARGIN SVM

- Hard-margin SVM: if *data not linearly separable*, feasible region is empty
  - No parameters can satisfy the constraints
- Soft-margin SVM: use slack parameters

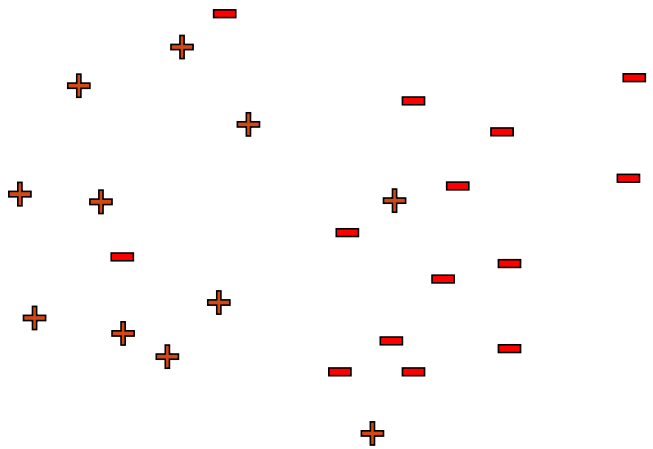
$$\min_{w,b,\xi} \underbrace{\frac{1}{\gamma(w,b)}}_{\text{Large margin}} + C \underbrace{\sum_n \xi_n}_{\text{Small slack:}}$$

Amount you need to pay, so a point is not misclassified

$$\text{Subject to } y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 - \xi_n \quad (\forall n)$$
$$\xi_n \geq 0 \quad (\forall n)$$

"SUPPORT VECTORS" OF SVM

- Which examples influence the margin and decision boundaries?

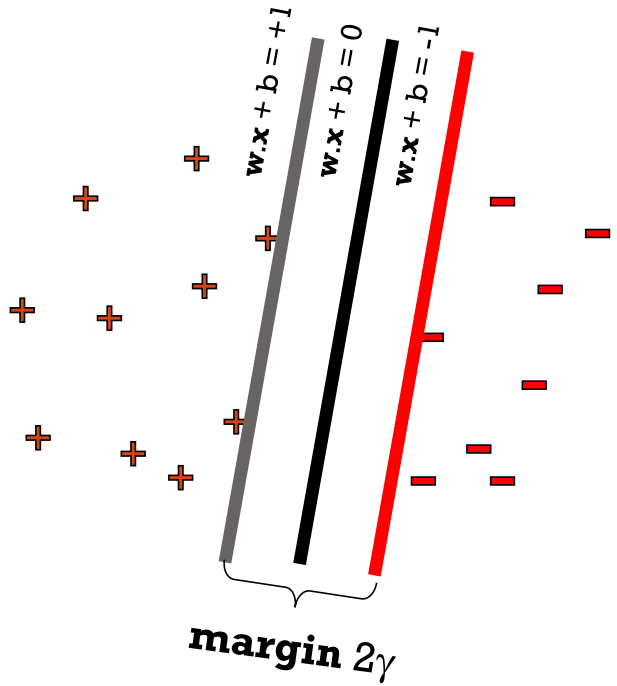


SVM OPTIMIZATION

- Need to be able to measure margin size
  - Hard-margin SVM: optimal hyperplane halfway between closest positive and negative points

$$d^+ = \frac{1}{||w||} w \cdot x^+ + b - 1$$

$$d^- = -\frac{1}{||w||} w \cdot x^- - b + 1$$



COMPUTING THE MARGIN

$$d^+ = \frac{1}{||w||} w \cdot x^+ + b - 1$$

$$d^- = -\frac{1}{||w||} w \cdot x^- - b + 1$$

$$\begin{aligned}
\gamma &= \frac{1}{2}\left[d^+ + d^-\right] \\
&= \frac{1}{2}\left[\frac{1}{||w||} w \cdot x^+ + b - 1 - \frac{1}{||w||} w \cdot x^- - b + 1\right] \\
&= \frac{1}{2}\left[\frac{1}{||w||} w \cdot x^+ - \frac{1}{||w||} w \cdot x^-\right] \\
&= \frac{1}{2}\left[\frac{1}{||w||}(+1) - \frac{1}{||w||}(-1)\right] \\
&= \frac{1}{||w||}
\end{aligned}$$

Maximizing the margin is equivalent to minimizing the l_2 norm!

SVM OPTIMIZATION

- Maximizing the margin is equivalent to minimizing the l_2 norm

$$\min_{w,b,\xi} \underbrace{\frac{1}{2}||\boldsymbol{w}|\Big|^2}_{\text{Large margin}} + \underbrace{C\sum_n \xi_n}_{\text{Small slack:}}$$

Does this remind you of something you have seen?

$$\text{Subject to } y_n(\boldsymbol{w}\cdot\boldsymbol{x_n}+b) \geq 1-\xi_n \quad (\forall n)$$
$$\xi_n \geq 0 \quad (\forall n)$$

- If you were given **w** and b, can you recover the slacks?

SVM OPTIMIZATION

- Recovering slacks when given **w** and b

$$\xi_n = \begin{cases} 0 & if \;\; y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 \\ 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) & otherwise \end{cases}$$

Does this remind you of something you have seen?

- SVM as unconstrained optimization problem

$$\min_{w,b,\xi} \; \frac{1}{2}||\boldsymbol{w}|\Big|^2 + C \sum_n l^{(hinge)}(\; y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b))$$

Large margin

Small slack:

SUMMARY

- Subgradient descent
  - When functions are not differentiable
- Closed form solutions
- Support vector machines

ANNOUNCEMENTS

- Homework
  - Academic honesty expectation

ACKNOWLEDGEMENTS

- These slides use materials by Marine Carpuat and Brian Ziebart