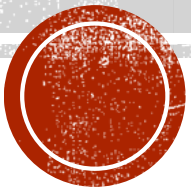
SVM & KERNELS



CS 412 Introduction to Machine Learning

Prof. Zheleva

February 27, 2017

Reading Assignment: CIML: 11.5-11.6, ISL: 9.3-9.4

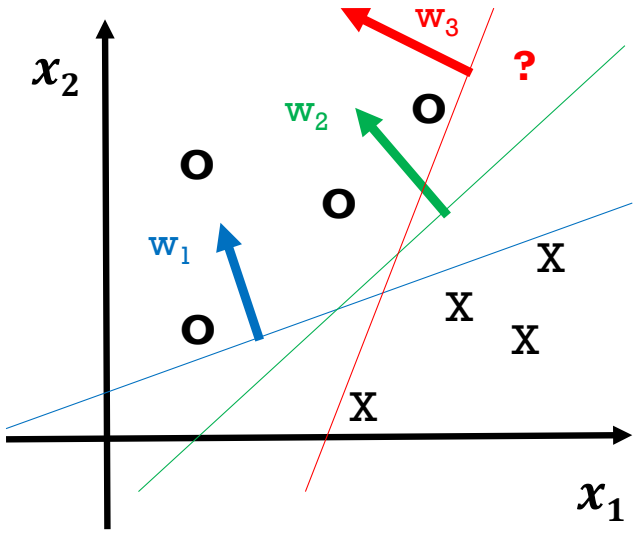
LAST TIME: KERNELS

- Two lectures ago, Linear SVM
  - Computing and optimizing the margin
- Kernel functions
  - Keep advantages of linear models, but make them capture non-linear patterns in data!
  - Map data to higher dimensions where it exhibits linear patterns
  - Rewrite linear models so that the mapping never needs to be explicitly computed
    - Kernelized perceptron

LINEAR MODELS

- A linear classifier is a hyperplane that separates positive from negative examples (**w**,b)
  - The prediction is a linear combination of feature values x
  - Examples: perceptron, linear SVM
- At test time, we check on what side of the hyperplane examples fall

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

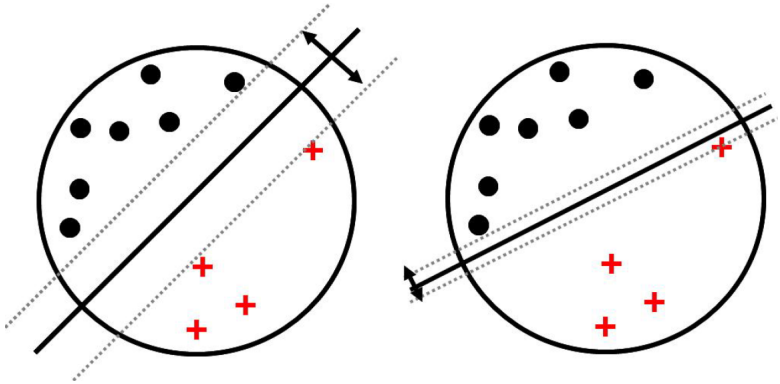


HARD-MARGIN SVM

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

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

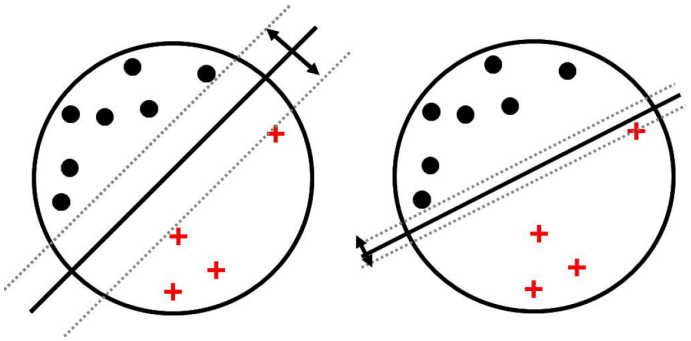
Subject to $y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 \ \ (\forall n)$



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

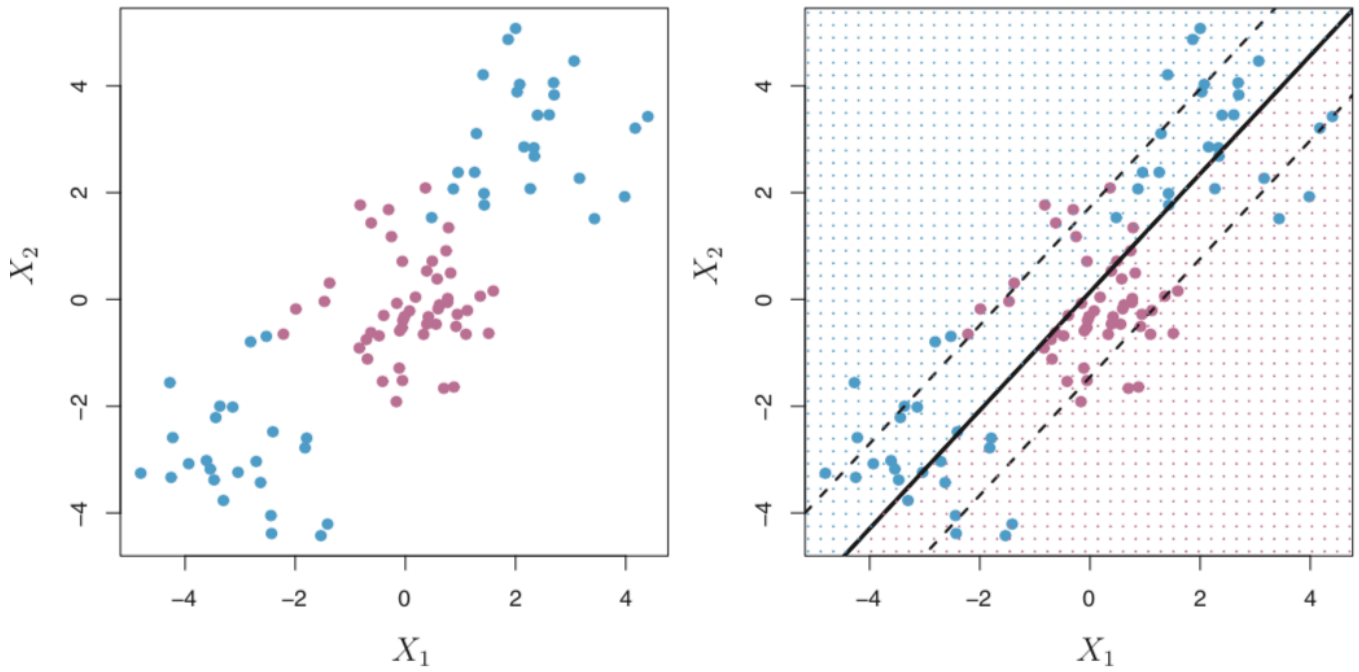
$$\min_{w,b,\xi} \underbrace{\frac{1}{\gamma(w,b)}}_{\text{Large margin}} + C \underbrace{\sum_n \xi_n}_{\text{Small slack:}}$$



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

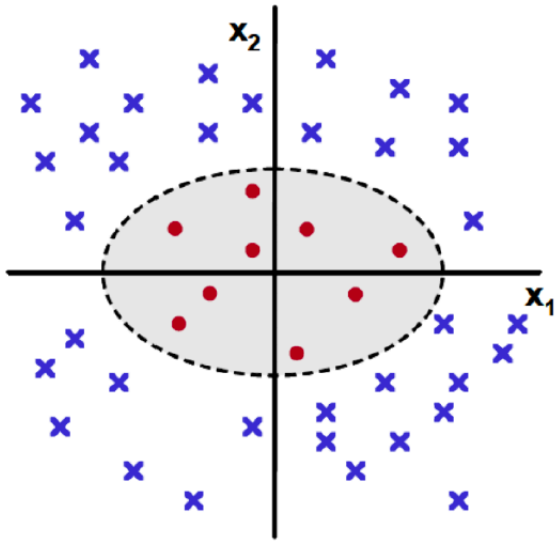
$$\text{Subject to } y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 - \xi_n \quad (\forall n)$$
$$\xi_n \geq 0 \quad (\forall n)$$

2D DATA AND A LINEAR SVM



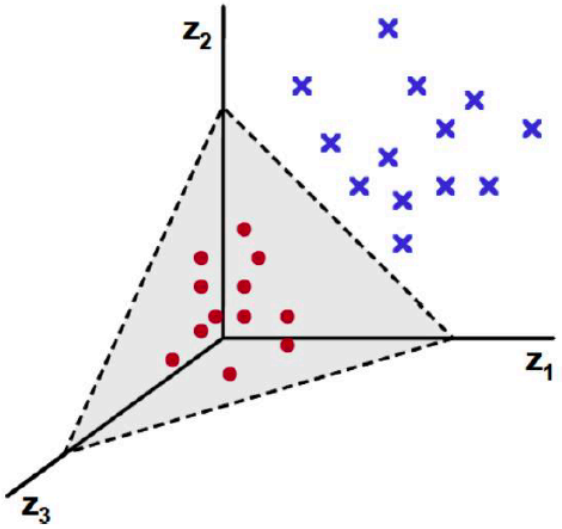
CLASSIFYING NON-LINEARLY SEPARABLE DATA WITH A LINEAR CLASSIFIER

- Example 2: Non-linearly separable data in 2D



- Becomes linearly separable in 3D defined by following mapping:

$$\boldsymbol{x} = (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



COMMONLY USED KERNEL FUNCTIONS

Linear (trivial) Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z} \text{ (mapping function } \phi \text{ is identity - no mapping)}$$

Quadratic Kernel:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2 \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^2$$

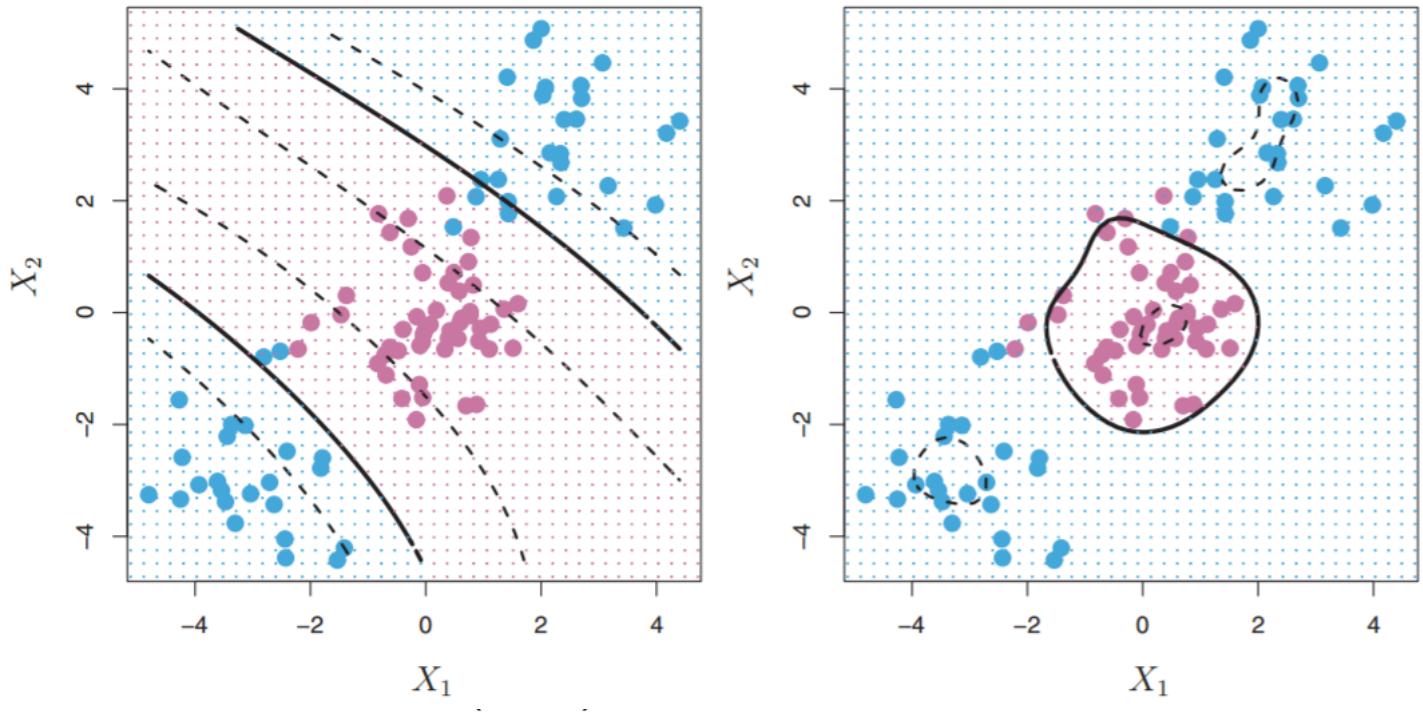
Polynomial Kernel (of degree d):

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^d \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^d$$

Radial Basis Function (RBF) Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \exp[-\gamma ||\mathbf{x} - \mathbf{z}||^2]$$

DEGREE 3 POLYNOMIAL AND RADIAL KERNELS



$$k(\mathbf{x}, \mathbf{z}) = \exp[-\gamma||\mathbf{x} - \mathbf{z}||^2]$$

THE KERNEL TRICK

1. Rewrite learning algorithms so they only depend on dot products between two examples
2. Replace dot product $\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z})$

   by kernel function k(x,z)

   which computes the dot product implicitly

- Intuition: kernel function quantifies the similarity of two examples
  - For example, the linear kernel $\mathbf{k}(\mathbf{x}, \mathbf{z}) = \boldsymbol{x} \cdot \boldsymbol{z}$ is equivalent to Pearson correlation

KERNELS: FORMALLY DEFINED

- Each kernel has an associated feature mapping $\boldsymbol{\phi}$
- $\boldsymbol{\phi}$ takes input $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ (input space) and maps it to $\mathcal{F}$ (feature space)
- Kernel k(**x**,**z**) takes two inputs and gives their similarity in $\mathcal{F}$ space

$$\phi: \boldsymbol{\mathcal{X}} \rightarrow \mathcal{F}$$
$$k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}, \qquad k(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$$

- $\mathcal{F}$ needs to be a vector space with a dot product defined in it
  - Also called Hilbert space

"KERNELIZING" THE PERCEPTRON

Algorithm 30 KERNELIZEDPERCEPTRONTRAIN(**D**, *MaxIter*)

```
1:  α ← 0, b ← 0                                   // initialize coefficients and bias
2:  for iter = 1 ... MaxIter do
3:      for all (x_n, y_n) ∈ D do
4:          a ← Σ_m α_m φ(x_m) · φ(x_n) + b        // compute activation for this example
5:          if y_n a ≤ 0 then
6:              α_n ← α_n + y_n                    // update coefficients
7:              b ← b + y                          // update bias
8:          end if
9:      end for
10: end for
11: return α, b
```

Same training algorithm but no explicit reference to weights
Only depends on dot products between examples

We can apply the kernel trick!

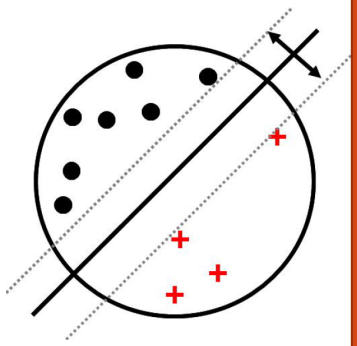
TODAY: KERNELIZED SVM

- Applying kernel trick to SVM
- Method of Lagrange multipliers
- Primal to dual transformation

RECAP: SOFT-MARGIN SVM

- Goal: find a hyperplane with largest possible margin
- Soft-margin SVM: use slack parameters

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



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

C hyperparameter dictates which term dominates the minimization

• Small C => prefer large margins and allows more misclassified examples

• Large C => prefer small number of misclassified examples, but at the expense of a small margin

RECAP: LINEAR SVM OPTIMIZATION

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

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

- SVM as unconstrained optimization problem

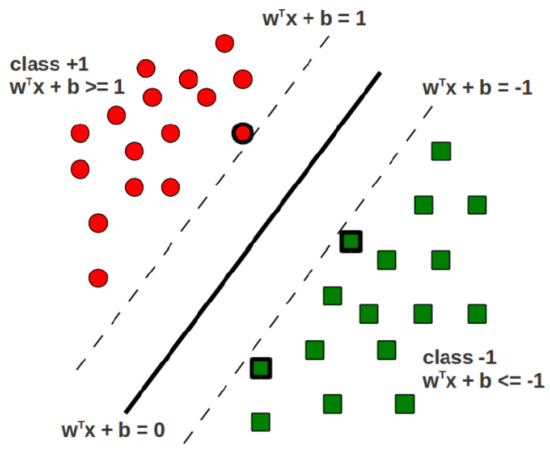
$$\min_{w,b,\xi} \; \frac{1}{2}||\boldsymbol{w}|\Big|^2 + C \sum_n l^{(hinge)}(\; y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b))$$

Large margin=$\frac{1}{||\boldsymbol{w}||}$

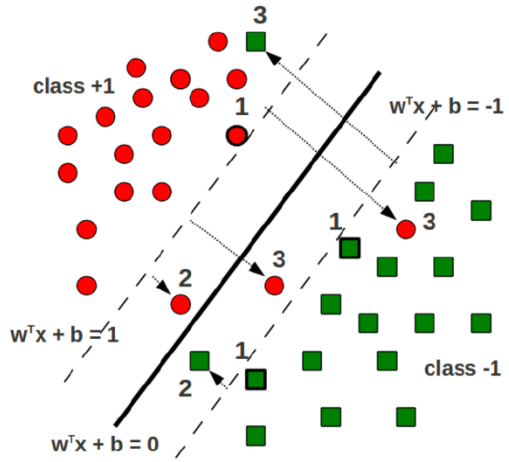
Small slack:

SVM

- Linearly separable case
  - Two support vectors



- Non-separable case
  - Three types of support vectors



(1) Lying on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$ ($\xi_n = 0$)

(2) Lying within the margin region ($0 < \xi_n < 1$) but still on the correct side

(3) Lying on the wrong side of the hyperplane ($\xi_n \geq 1$)

RECAP: THE KERNEL TRICK

1. Rewrite learning algorithms so they only depend on dot products between two examples
2. Replace dot product $\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z})$

   by kernel function k(x,z)

   which computes the dot product implicitly

- Intuition: kernel function quantifies the similarity of two examples
  - For example, the linear kernel $\mathbf{k}(\mathbf{x}, \mathbf{z}) = \boldsymbol{x} \cdot \boldsymbol{z}$ is equivalent to Pearson correlation

"KERNELIZING" SVM

- Goal: re-write SVM in a way that it no longer depends on **w** explicitly

$$\min_{w,b,\xi} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_n \xi_n$$

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

$$\xi_n \geq 0 \quad (\forall n)$$

- Solve using method of Lagrange multipliers
  - Add new variables that correspond to each constraint
  - 2N constraints

"KERNELIZING" SVM

- To use Lagrange multipliers, first re-write all constraints as ≥ 0

$$\min_{w,b,\xi} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_n \xi_n$$

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

$$\xi_n \geq 0 \quad (\forall n)$$

- Incorporate constraints through Lagrange multipliers

$$\min_{w,b,\xi} \max_{\boldsymbol{\alpha}\geq 0} \max_{\boldsymbol{\beta}\geq 0} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_n \xi_n - \sum_n \beta_n \xi_n - \sum_n \alpha_n \left[y_n(\boldsymbol{w}\cdot\boldsymbol{x_n} + b) - 1 + \xi_n\right]$$

"KERNELIZING" SVM

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_n \xi_n - \sum_n \beta_n \xi_n - \sum_n \alpha_n \left[y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) - 1 + \xi_n\right]$$

- Solve using gradients

$$\nabla_w \mathcal{L} = \boldsymbol{w} - \sum_n \alpha_n y_n \boldsymbol{x}_n = 0 \iff \boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n$$

  - Same as perceptron!
  - Plug **w** back in objective function $\mathcal{L}$

"KERNELIZING" SVM

$$\mathcal{L}(b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} || \sum_m \alpha_m y_m \boldsymbol{x}_m ||^2 + C \sum_n \xi_n - \sum_n \beta_n \xi_n - \sum_n \alpha_n \left[y_n (\sum_n \alpha_m y_m \boldsymbol{x}_m \cdot \boldsymbol{x_n} + b) - 1 + \xi_n \right]$$

$$= \frac{1}{2} \sum_n \sum_m \alpha_m \alpha_n y_m y_n \boldsymbol{x}_m \boldsymbol{x}_n + \sum_n (C - \beta_n) \xi_n$$

$$- \sum_n \sum_m \alpha_m \alpha_n y_m y_n \boldsymbol{x}_m \boldsymbol{x}_n - \sum_n \alpha_n \ (y_n b - 1 + \xi_n)$$

$$= -\frac{1}{2} \sum_n \sum_m \alpha_m \alpha_n y_m y_n \boldsymbol{x}_m \boldsymbol{x}_n + \sum_n (C - \beta_n) \xi_n$$

$$- b \sum_n \alpha_n y_n - \sum_n \alpha_n \ (\xi_n - 1)$$

- We got rid of **w!**
- Next: remove b and $\boldsymbol{\xi}$
  - Derivative wrt b is $-\sum_n \alpha_n y_n$=0 (cannot substitute b but term goes to 0 at optimum)
  - Derivative wrt ξ_n is $C - \beta_n - \alpha_n$=0 allows us to substitute $C - \beta_n = \alpha_n$ subject to constraint $C \geq \alpha_n$
- Final Dual Lagrangian form

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_n \alpha_n - \frac{1}{2} \sum_n \sum_m \alpha_m \alpha_n y_m y_n \boldsymbol{x}_m \boldsymbol{x}_n$$

subject to constraint $0 \leq \alpha_n \leq C$

"KERNELIZING" SVM

- That was a lot of Math!
  - What did we gain?
- Transformed problem from primal Lagrangian form

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{||\mathbf{w}||^2}{2} + +C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n\{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) - \xi_n\} - \sum_{n=1}^{N}\beta_n\xi_n$$

$$\text{subject to} \quad \alpha_n, \beta_n \geq 0; \quad n = 1, \ldots, N$$

to dual Lagrangian form

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n y_m y_n(\mathbf{x}_m^T\mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N}\alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N$$

Can replace with kernel!

SOME INTUITION

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n K_{mn}$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N$$

- Goal: make first term as large as possible, second term as small as possible
  - Constraint ensures α_n cannot get larger than C, so α_n tends to go to C
  - If K is large (two samples very similar) and same label, then either α_n or α_m would need to go to 0
    - Ensures no redundant support vectors get included
  - If K is large and different labels, then both α_n and α_m will be encouraged to go to C

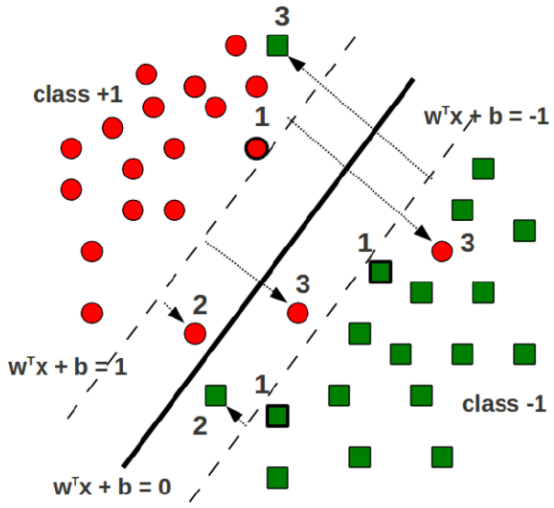
NOTES ON TRAINING

- Solving the problem is $O(N^3)$
  - Can be prohibitive for large datasets
- But many options to speed up training
  - Approximate solvers
  - Learn from what we know about training linear models

SVM PREDICTION

- What is the meaning of the dual Lagrangian?
  - α is sparse, nonzero α_n's correspond to support vectors
  - α encodes "importance" of support vector
  - All we need to predict are the support vectors!
    - Compare to perceptron

$$f(\hat{\boldsymbol{x}}) = \text{sign}(\textstyle\sum_n \alpha_n y_n K(\boldsymbol{x}_n, \hat{\boldsymbol{x}}))$$



USING SVM: CUPCAKES VS MUFFINS

https://www.youtube.com/watch?v=N1vOgolbjSc&t=372s

SUMMARY

- What are Support Vector Machines
- How to train SVMs
  - Which optimization problem we need to solve
- Geometric interpretation
  - What are support vectors and what is their relationship with parameters **w**,b?
- How do SVM relate to the general formulation of linear classifiers
- Why/how can SVMs be kernelized

ANNOUNCEMENTS

- Homework 3
  - Released on Monday
  - Due March 15

ACKNOWLEDGEMENTS

- These slides use materials by Marine Carpuat and Piyush Rai