Kernels, SVMs

CMSC 422

SOHEIL FEIZI

sfeizi@cs.umd.edu

Slides adapted from MARINE CARPUAT

Today’s topics

- Kernelized Perceptron

- SVMs

The Kernel Trick

- Rewrite learning algorithms so they only depend on **dot products between two examples**

- Replace dot product $\phi(\mathbf{x})^\top \phi(\mathbf{z})$
  by **kernel function** $k(\mathbf{x}, \mathbf{z})$
  which computes the dot product **implicitly**

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:

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

The Kernel Trick

- Rewrite learning algorithms so they only depend on **dot products between two examples**

- Replace dot product $\phi(\mathbf{x})^\top \phi(\mathbf{z})$
  by **kernel function** $k(\mathbf{x}, \mathbf{z})$
  which computes the dot product **implicitly**

"Kernelizing" the perceptron

- Naïve approach: let's explicitly train a perceptron in the new feature space

Algorithm 28 PerceptronTrain($\mathbf{D}$, $MaxIter$)

$\boldsymbol{w} \leftarrow \mathbf{0}$, $b \leftarrow 0$ // initialize weights and bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(\boldsymbol{x}, y) \in \mathbf{D}$ **do**
    $a \leftarrow \boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\phi(\boldsymbol{x})$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $\boldsymbol{w}$, b

Can we apply the Kernel trick?
Not yet, we need to rewrite the algorithm using dot products between examples

“Kernelizing” the perceptron

- Perceptron Representer Theorem

  “During a run of the perceptron algorithm, the weight vector w can always be represented as a linear combination of the expanded training data”

  Proof by induction

  (in CIML)

"Kernelizing" the perceptron

- We can use the perceptron representer theorem to compute activations as a **dot product** between examples

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b = \left(\sum_n \alpha_n \phi(\boldsymbol{x}_n) \right) \cdot \phi(\boldsymbol{x}) + b \qquad \text{definition of } \boldsymbol{w} \tag{9.6}$$

$$= \sum_n \alpha_n \left[\phi(\boldsymbol{x}_n) \cdot \phi(\boldsymbol{x}) \right] + b \qquad \text{dot products are linear} \tag{9.7}$$

"Kernelizing" the perceptron

Algorithm 29 KernelizedPerceptronTrain($\mathbf{D}$, *MaxIter*)

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$, $b \leftarrow 0$ // initialize coefficients and bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(\boldsymbol{x}_n, y_n) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_m \alpha_m \phi(\boldsymbol{x}_m) \cdot \phi(\boldsymbol{x}_n) + b$ // compute activation for this example
    if $y_n a \leq 0$ **then**
      $\alpha_n \leftarrow \alpha_n + y_n$ // update coefficients
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $\boldsymbol{\alpha}$, b

- Same training algorithm, but
doesn't explicitly refers to weights w anymore
only depends on dot products between examples

- We can apply the kernel trick!

Kernel Methods

- Goal: keep advantages of linear models, but make them capture non-linear patterns in data!
- How?
  - By **mapping data to higher dimensions** where it exhibits linear patterns
  - By **rewriting linear models** so that the **mapping never needs to be explicitly computed**

Discussion

- Other algorithms can be kernelized:
  – See CIML for K-means

- Do Kernels address all the downsides of "feature explosion"?
  – Helps reduce computation cost during training
  – But overfitting remains an issue

What you should know

- Kernel functions
  - What they are, why they are useful, how they relate to feature combination
- Kernelized perceptron
  - You should be able to derive it and implement it

Support Vector Machines

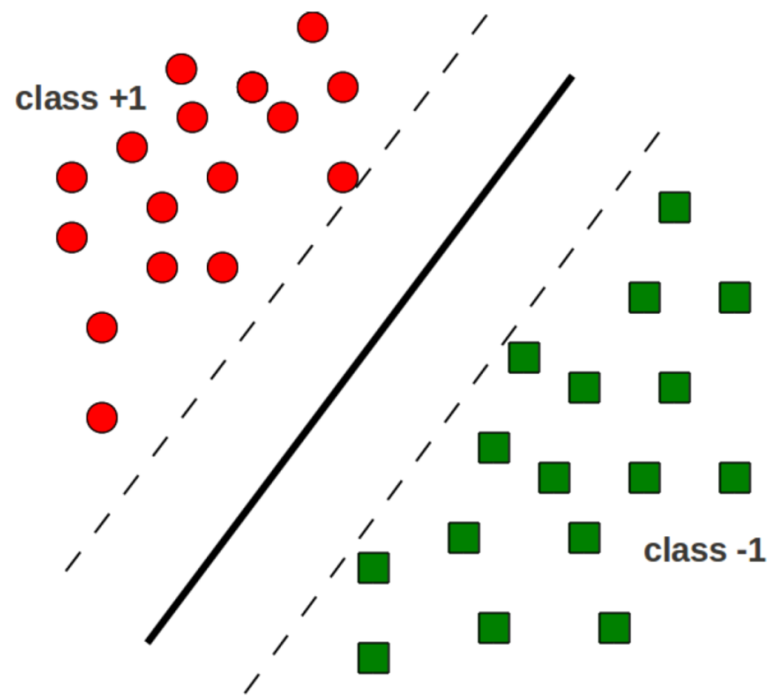
Back to linear classification

- So far: we've seen that kernels can help capture non-linear patterns in data while keeping the advantages of a linear classifier

- Support Vector Machines
  - A hyperplane-based classification algorithm
  - Highly influential
  - Backed by solid theoretical grounding (Vapnik & Cortes, 1995)
  - Easy to kernelize

The Maximum Margin Principle

- Find the hyperplane with maximum separation margin on the training data



Margin of a data set D

$$margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(x,y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x} + b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases} \quad (3.8)$$

Distance between the hyperplane (w,b) and the nearest point in D

$$margin(\mathbf{D}) = \sup_{\boldsymbol{w},b} margin(\mathbf{D}, \boldsymbol{w}, b) \quad (3.9)$$

Largest attainable margin on D

Support Vector Machine (SVM)

A hyperplane based linear classifier defined by $\mathbf{w}$ and b

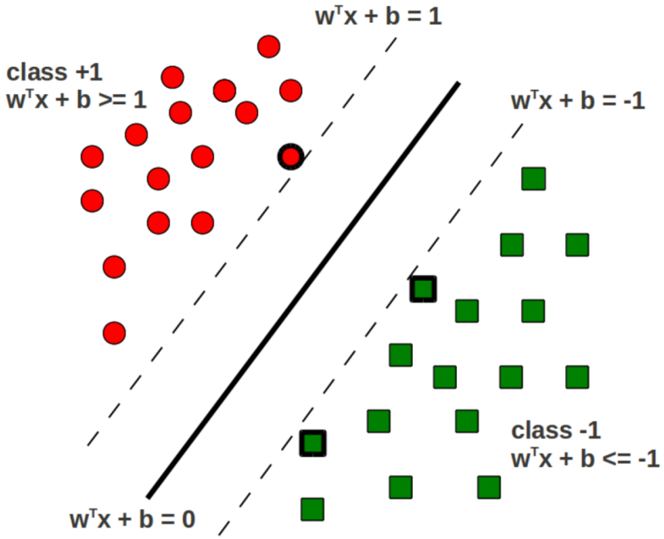
Prediction rule: $y = sign(\mathbf{w}^T\mathbf{x} + b)$

Given: Training data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

Goal: Learn $\mathbf{w}$ and b that achieve the maximum margin

Characterizing the margin

Let's assume the entire training data is correctly classified by (**w**,b) that achieve the maximum margin

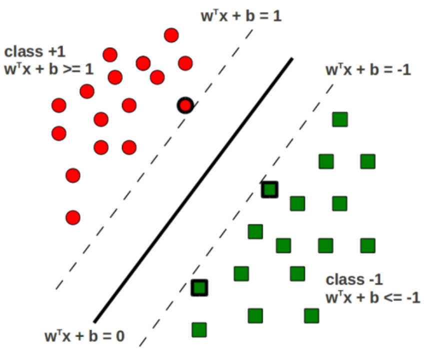


- Assume the hyperplane is such that
  - $\mathbf{w}^T\mathbf{x}_n + b \geq 1$ for $y_n = +1$
  - $\mathbf{w}^T\mathbf{x}_n + b \leq -1$ for $y_n = -1$
  - Equivalently, $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
    $\Rightarrow \min_{1 \leq n \leq N} |\mathbf{w}^T\mathbf{x}_n + b| = 1$
  - The hyperplane's margin:

$$\gamma = \min_{1 \leq n \leq N} \frac{|\mathbf{w}^T\mathbf{x}_n + b|}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||}$$

The Optimization Problem

We want to maximize the margin $\gamma = \frac{1}{||\mathbf{w}||}$



Maximizing the margin γ = minimizing $||\mathbf{w}||$ (the norm)

Our optimization problem would be:

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2}$$
$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

Large Margin = Good Generalization

- Intuitively, large margins mean good generalization
  - Large margin => small $||\mathbf{w}||$
  - small $||\mathbf{w}||$ => regularized/simple solutions

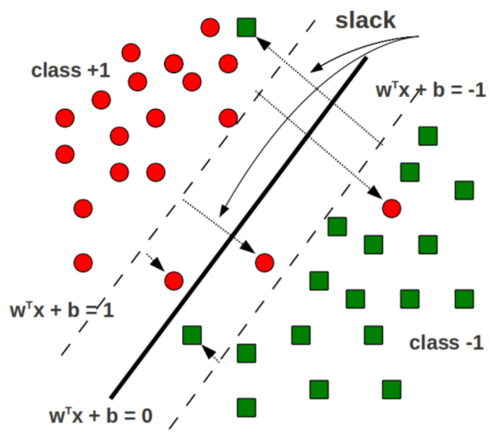
- (Learning theory gives a more formal justification)

SVM in the non-separable case

- no hyperplane can separate the classes perfectly

- We still want to find the max margin hyperplane, but
  - We will allow some training examples to be **misclassified**
  - We will allow some training examples to fall **within** the margin region

SVM in the non-separable case



Recall: For the separable case (training loss = 0), the constraints were:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

For the non-separable case, we relax the above constraints as:

$$\boxed{y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \quad \forall n}$$

ξ_n is called slack variable (distance $\mathbf{x}_n$ goes past the margin boundary)

$\xi_n \geq 0, \forall n$, misclassification when $\xi_n > 1$

SVM Optimization Problem

Non-separable case: We will allow misclassified training examples

- .. but we want their number to be minimized
  $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)

The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C \sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

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