Support Vector Machines
(or – more generally – Kernel Machines)

- Motivation
- Linear Classifiers
  - Rosenblatt Learning Rule
- Kernel Methods and Support Vector Machines
  - Dual Representation
  - Maximal Margins
  - Kernels
- Soft Margin Classifiers

Motivation

- Main idea of Kernel Methods
  - Embed data into suitable vector space
  - Find linear classifier (or other linear pattern of interest) in new space
- Needed: a Mapping

$$\Phi : x \in X \mapsto \Phi(x) \in F$$

- Key Assumptions:
  - Information about relative position is often all that is needed by learning methods
  - The inner products between points in the projected space can be computed in the original space using special functions (kernels).

Linear Discriminant

Simple linear, binary classifier:

$$f(\vec{x}) = \langle \vec{x}, \vec{w} \rangle + b = \sum_{i=1}^{n} x_i w_i + b$$

- Class A if $f(\vec{x})$ positive
- Class B if $f(\vec{x})$ negative

e.g. $h(\vec{x}) = \operatorname{sgn}\left(f(\vec{x})\right)$ is the decision function.

Linear Discriminant

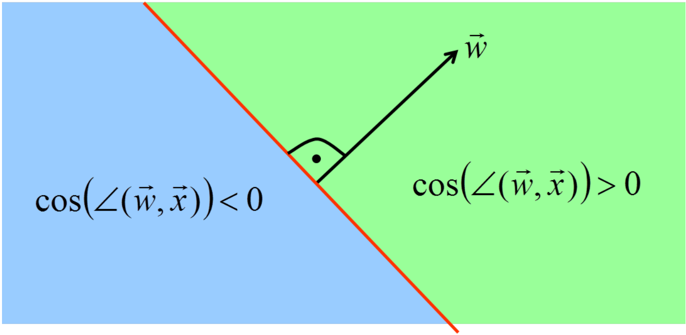
$$f(\vec{x}) = \langle \vec{x}, \vec{w} \rangle + b = \sum_{i=1}^{n} x_i w_i + b$$

Linear Discriminant

$$f(\vec{x}) = \langle \vec{x}, \vec{w} \rangle + b = \sum_{i=1}^{n} x_i w_i + b = b + |\vec{x}|\,|\vec{w}| \cos\left(\angle(\vec{w}, \vec{x})\right)$$

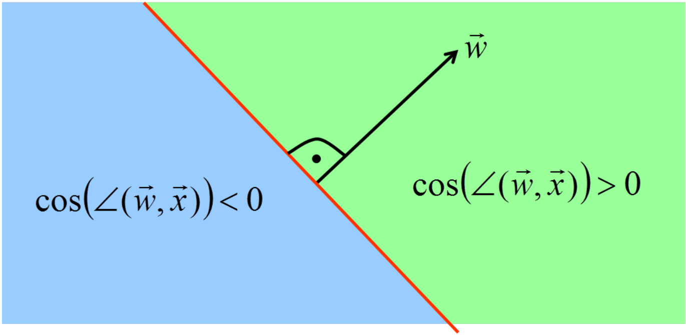
Linear Discriminant

$$f(\vec{x}) = \langle \vec{x}, \vec{w} \rangle + b = \sum_{i=1}^{n} x_i w_i + b = b + |\vec{x}|\,|\vec{w}| \cos(\angle(\vec{w}, \vec{x}))$$



Linear Discriminant

$$f(\vec{x}) = \langle \vec{x}, \vec{w} \rangle + b = \sum_{i=1}^{n} x_i w_i + b = b + |\vec{x}|\,|\vec{w}| \cos\left(\angle(\vec{w}, \vec{x})\right)$$



$\Rightarrow$ Linear discriminants represent hyperplanes in feature space.

Remember? Training a Perceptron

- Classification using a Perceptron
  - Represents a (hyper-) plane: $\sum_{i=1}^{n} w_i \cdot x_i = \theta$
  - Left of hyperplane: class 0
  - Right of hyperplane: class 1
- Training a Perceptron
  - Learn the "correct" weights to distinguish the two classes
  - Iterative adaption of weights w_i
  - Rotation of the hyperplane defined by $\vec{w}$ and θ in small direction of $\vec{x}$ if $\vec{x}$ is not yet on the correct side of the hyperplane.

Primal Perceptron

Rosenblatt (1959) introduced simple learning algorithm for linear discriminants ("perceptrons"):

Primal Perceptron

Rosenblatt (1959) introduced simple learning algorithm for linear discriminants ("perceptrons"):
Given a linearly separable training set S

$\vec{w_0} \leftarrow \mathbf{0}$; $b_0 \leftarrow 0$; $k \leftarrow 0$

$R \leftarrow max_{1 \leq j \leq m} \; ||\vec{x_j}||$

repeat

for $j = 1$ **to** m

if $y_j(\langle \vec{w_k}, x_j \rangle + b_k) \leq 0$ **then**

$\vec{w_{k+1}} \leftarrow \vec{w_k} + y_j \vec{x_j}$

$b_{k+1} \leftarrow b_k + y_j R^2$

$k \leftarrow k + 1$

end if

end for

until no mistakes made within the for loop

return $(\vec{w_k}, b_k)$

Rosenblatt Algorithm

- Algorithm is
  - On-line (pattern by pattern approach)
  - Mistake driven (updates only in case of wrong classification)
- Algorithm converges guaranteed if a hyperplane exists which classifies all training data correctly (data is linearly separable)

Rosenblatt Algorithm

- Algorithm is
  - On-line (pattern by pattern approach)
  - Mistake driven (updates only in case of wrong classification)
- Algorithm converges guaranteed if a hyperplane exists which classifies all training data correctly (data is linearly separable)
- Learning rule:

$$\text{IF } y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) < 0 \text{ THEN } \begin{cases} \vec{w}(t+1) = \vec{w}(t) + y_j \cdot \vec{x}_j \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

Rosenblatt Algorithm

- Algorithm is
  - On-line (pattern by pattern approach)
  - Mistake driven (updates only in case of wrong classification)
- Algorithm converges guaranteed if a hyperplane exists which classifies all training data correctly (data is linearly separable)
- Learning rule:

$$\text{IF } y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) < 0 \text{ THEN } \begin{cases} \vec{w}(t+1) = \vec{w}(t) + y_j \cdot \vec{x}_j \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

- One observation:
  Weight vector (if initialized properly) is simply a weighted sum of input vectors (b is even more trivial).

Dual Representation

- Weight vector is a weighted sum of input vectors:

$$\vec{w} = \sum_{j=1}^{m} \alpha_j \cdot y_j \cdot \vec{x}_j.$$

- "difficult" training patterns have larger alpha
- "easier" ones have smaller or zero alpha

Dual Representation (of discriminant function)

$$f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b$$

Dual Representation (of discriminant function)

$$f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b = \left\langle \sum_{j=1}^{m} \alpha_j \cdot y_j \cdot \vec{x}_j, \vec{x} \right\rangle + b$$

Dual Representation (of discriminant function)

$$f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b = \left\langle \sum_{j=1}^{m} \alpha_j \cdot y_j \cdot \vec{x}_j, \vec{x} \right\rangle + b = \left(\sum_{j=1}^{m} \alpha_j \cdot y_j \cdot \langle \vec{x}_j, \vec{x} \rangle \right) + b.$$

Dual Representation

Dual Representation of Learning Algorithm:
Given a training set S

$\vec{\alpha} \leftarrow \mathbf{0}$; $b \leftarrow 0$

$R \leftarrow max_{1 \leq i \leq m} ||x_i||$

repeat

for $i = 1$ **to** m

if $y_i(\sum_{j=1}^{m} \alpha_j y_j \langle \vec{x_j}, \vec{x_i} \rangle + b) \leq 0$ **then**

$\alpha_i \leftarrow \alpha_i + 1$

$b \leftarrow b + y_i R^2$

end if

end for

until no mistakes made within the for loop

return $(\vec{\alpha}, b)$

Dual Representation

- Learning Rule:

$$\text{IF } y_j \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \langle \vec{x}_i, \vec{x}_j \rangle + b \right) < 0 \text{ THEN } \begin{cases} \alpha_i(t+1) = \alpha_i(t) + 1 \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

Dual Representation

- Learning Rule:

$$\text{IF } y_j \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \langle \vec{x}_i, \vec{x}_j \rangle + b \right) < 0 \text{ THEN } \begin{cases} \alpha_i(t+1) = \alpha_i(t) + 1 \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

- Harder to learn examples have larger alpha (higher information content)

Dual Representation

- Learning Rule:

$$\text{IF } y_j \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \langle \vec{x}_i, \vec{x}_j \rangle + b \right) < 0 \text{ THEN } \begin{cases} \alpha_i(t+1) = \alpha_i(t) + 1 \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

- Harder to learn examples have larger alpha (higher information content)
- The information about training examples enters algorithm only through the inner products

Dual Representation

- Learning Rule:

$$\text{IF } y_j \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \langle \vec{x}_i, \vec{x}_j \rangle + b \right) < 0 \text{ THEN } \begin{cases} \alpha_i(t+1) = \alpha_i(t) + 1 \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

- Harder to learn examples have larger alpha (higher information content)
- The information about training examples enters algorithm only through the inner products (which we could pre-compute!...)

Dual Representation in other spaces

- All we need for training:
  - Computation of inner products of all training examples

Dual Representation in other spaces

- All we need for training:
  - Computation of inner products of all training examples
- If we train in a different space:
  - Computation of inner products in the projected space

Dual Representation in other spaces

- All we need for training:
  - Computation of inner products of all training examples
- If we train in a different space:
  - Computation of inner products in the projected space

Can we compute the inner product in the projected space directly?

Kernel Functions

Definition : A kernel is a function K, such that for all $(\mathbf{x}, \mathbf{y}) \in X$

$$K(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$$

where Φ is a mapping from X to an (inner product) feature space F.

- A kernel allows us (via K) to compute the inner product of two points x and y in the projected space without ever entering that space!

...in Kernel Land...

- The discriminant function in our projected space:

$$f(\vec{x}) = \sum_{j=1}^{n} \alpha_j y_j \langle \Phi(\vec{x}), \Phi(\vec{x}_j) \rangle + b$$

...in Kernel Land...

- The discriminant function in our projected space:

$$f(\vec{x}) = \sum_{j=1}^{n} \alpha_j y_j \langle \Phi(\vec{x}), \Phi(\vec{x}_j) \rangle + b$$

- And, using a kernel K:

$$f(\vec{x}) = \sum_{j=1}^{n} \alpha_j y_j K(\vec{x}, \vec{x}_j) + b$$

The Gram Matrix

All data necessary for:

- the decision function
- the training of the coefficients

can be pre-computed

The Gram Matrix

All data necessary for:

- the decision function
- the training of the coefficients

can be pre-computed using a Kernel or Gram Matrix:

The Gram Matrix

All data necessary for:

- the decision function
- the training of the coefficients

can be pre-computed using a Kernel or Gram Matrix:

$$\mathbf{K} = \begin{pmatrix} K(\vec{x}_1, \vec{x}_1) & K(\vec{x}_1, \vec{x}_2) & \cdots & K(\vec{x}_1, \vec{x}_m) \\ K(\vec{x}_2, \vec{x}_1) & K(\vec{x}_2, \vec{x}_2) & \cdots & K(\vec{x}_2, \vec{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ K(\vec{x}_m, \vec{x}_1) & K(\vec{x}_m, \vec{x}_2) & \cdots & K(\vec{x}_m, \vec{x}_m) \end{pmatrix}$$

What is a Valid Kernel ?

Let X be a nonempty set. A function is a valid kernel in X if for all n and all $x_1, ..., x_n \in X$ it produces a Gram matrix K, which is:

- symmetric

$$K = K^T$$

- positive semi-definite

$$\forall \vec{\alpha} : \vec{\alpha}^T K \vec{\alpha} \geq 0$$

Kernels

- A simple kernel is

$$K(x, y) = (x_1 y_1 + x_2 y_2)^2$$

- And the corresponding projected space:

$$(x_1, x_2) \mapsto \Phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

- Why?

Kernels

- A simple kernel is

$$K(x, y) = (x_1 y_1 + x_2 y_2)^2$$

- And the corresponding projected space:

$$(x_1, x_2) \mapsto \Phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

- Why?

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y} \rangle^2 &= \langle (x_1, x_2), (y_1, y_2) \rangle^2 \\ &= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2), (y_1^2, y_2^2, \sqrt{2} y_1 y_2) \rangle \\ &= x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2 \\ &= (x_1 y_1 + x_2 y_2)^2 \end{aligned}$$

Kernels

- A few (slightly less) simple kernels are $K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle^d$

Kernels

- A few (slightly less) simple kernels are $K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle^d$
- And the corresponding projected spaces are of dimension $\binom{n+d-1}{d}$

Kernels

- A few (slightly less) simple kernels are $K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle^d$
- And the corresponding projected spaces are of dimension $\binom{n+d-1}{d}$
- but computing the inner products in the projected space becomes pretty expensive rather quickly ...

More Kernels

More Kernels

- Gaussian Kernel :

$$K(\vec{x}, \vec{y}) = e^{-\frac{||\vec{x}-\vec{y}||^2}{2\sigma^2}}$$

More Kernels

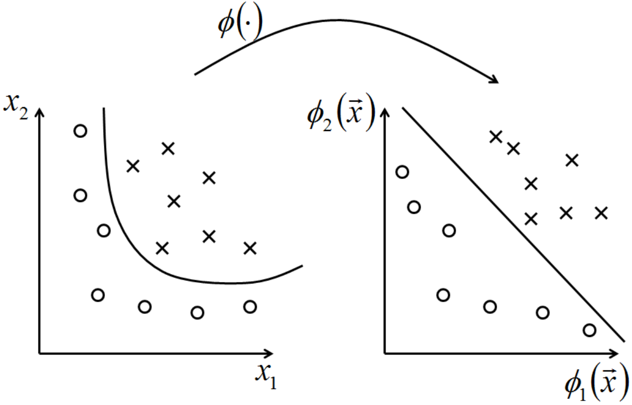
- Gaussian Kernel :

$$K(\vec{x},\vec{y}) = e^{-\frac{||\vec{x}-\vec{y}||^2}{2\sigma^2}}$$

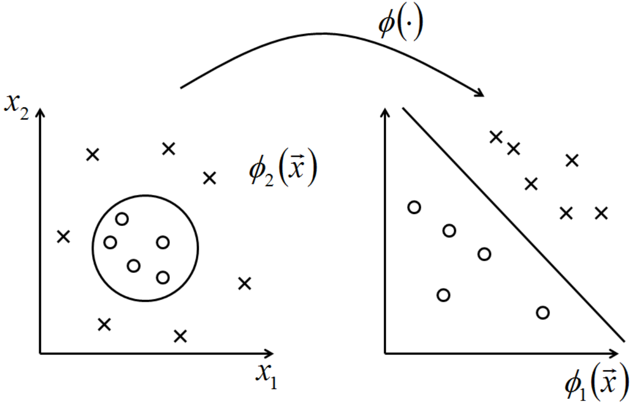
- Polynomial Kernel of degree d:

$$K(\vec{x},\vec{y}) = (\langle\vec{x},\vec{y}\rangle + 1)^d$$

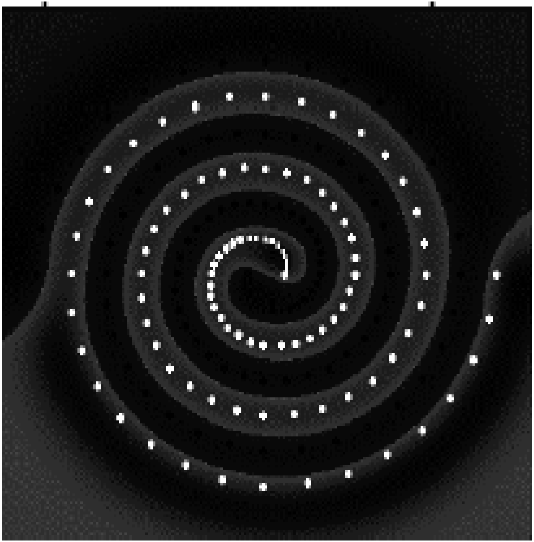
Projections...



Polynomial Kernel



Gauss Kernel



- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones
  - Kernels can be defined over non numerical objects

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones
  - Kernels can be defined over non numerical objects
    - text: e.g. string matching kernel

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones
  - Kernels can be defined over non numerical objects
    - text: e.g. string matching kernel
    - images, trees, graphs,...

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones
  - Kernels can be defined over non numerical objects
    - text: e.g. string matching kernel
    - images, trees, graphs,...
- Note also: A good Kernel is crucial

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones
  - Kernels can be defined over non numerical objects
    - text: e.g. string matching kernel
    - images, trees, graphs,...
- Note also: A good Kernel is crucial
  - Gram Matrix diagonal: classification easy and useless

Kernels

- Note that we do not need to know the projection Φ.
  It is sufficient to prove that $K(\cdot)$ is a Kernel.
- A few notes:
  - Kernels are modular and closed: we can compose new Kernels based on existing ones
  - Kernels can be defined over non numerical objects
    - text: e.g. string matching kernel
    - images, trees, graphs,...
- Note also: A good Kernel is crucial
  - Gram Matrix diagonal: classification easy and useless
  - No Free Kernel: too many irrelevant attributes: Gram Matrix diagonal.

Finding Linear Discriminants

- Finding the hyperplane (in any space) still leaves lots of room for variations?

Finding Linear Discriminants

- Finding the hyperplane (in any space) still leaves lots of room for variations?
- We can define "margins" of individual training examples:

$$\gamma_i = y_i \left(\langle \vec{w}, \vec{x}_i \rangle + b\right)$$

(appropriately normalized this is a "geometrical" margin)

Finding Linear Discriminants

- Finding the hyperplane (in any space) still leaves lots of room for variations?
- We can define "margins" of individual training examples:

$$\gamma_i = y_i \left(\langle \vec{w}, \vec{x}_i \rangle + b \right)$$

  (appropriately normalized this is a "geometrical" margin)
- The margin of a hyperplane (with respect to a training set):

$$\min_{i=1 \cdots n} \gamma_i$$

Finding Linear Discriminants

- Finding the hyperplane (in any space) still leaves lots of room for variations?
- We can define "margins" of individual training examples:

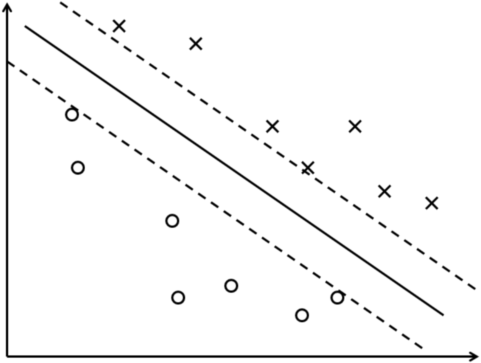
$$\gamma_i = y_i \left(\langle \vec{w}, \vec{x}_i \rangle + b \right)$$

  (appropriately normalized this is a "geometrical" margin)
- The margin of a hyperplane (with respect to a training set):

$$\min_{i=1 \cdots n} \gamma_i$$

- And a maximal margin of all training examples indicates the maximum margin over all hyperplanes.

(maximum) Margin of a Hyperplane



Finding Linear Discriminants

The original objective function

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 0$$

Finding Linear Discriminants

The original objective function

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 0$$

is reformulated slightly:

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1$$

The original objective function

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 0$$

is reformulated slightly:

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1$$

The decision line is still defined by:

$$\langle \vec{w}, \vec{x} \rangle + b = 0$$

Finding Linear Discriminants

The original objective function

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 0$$

is reformulated slightly:

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1$$

The decision line is still defined by:

$$\langle \vec{w}, \vec{x} \rangle + b = 0$$

and in addition the upper and lower margin are defined by

$$\langle \vec{w}, \vec{x} \rangle + b = \pm 1$$

The distance between those two hyperplanes is $2/||\vec{w}||$.

Finding Linear Discriminants

Finding the maximum margin now turns into a minimization problem:

minimize (in $\vec{w}, b$)

$$||\vec{w}||$$

subject to (for any $j = 1, \ldots, n$)

$$y_j \left(\langle \vec{w}, \vec{x} \rangle - b \right) \geq 1$$

Finding the maximum margin now turns into a minimization problem:

minimize (in $\vec{w}, b$)

$||\vec{w}||$

subject to (for any $j = 1, \ldots, n$)

$y_j \left(\langle \vec{w}, \vec{x} \rangle - b \right) \geq 1$

Solution sketch:

- solutions depends on $||\vec{w}||$, the norm of $\vec{w}$ which involves a square root.
- convert into a quadratic form by substituting $||\vec{w}||$ with $\frac{1}{2}||\vec{w}||^2$ without changing the solution.
- Using Lagrange multipliers this turns into a standard quadratic programming problems.

Notes: Soft and Hard Margin Classifiers

What can we do if no linear seperating hyperplane exists?

Notes: Soft and Hard Margin Classifiers

What can we do if no linear seperating hyperplane exists?

Instead of focusing on find a hard margin, allow minor violations

- Introduce (positive) slack variables (patterns with slack are allowed to lie in margin)

$$\forall j = 1, \ldots n : y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1 - \epsilon_j$$

Notes: Soft and Hard Margin Classifiers

What can we do if no linear seperating hyperplane exists?

Instead of focusing on find a hard margin, allow minor violations

- Introduce (positive) slack variables (patterns with slack are allowed to lie in margin)

$$\forall j = 1, \ldots n : y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1 - \epsilon_j$$

- Misclassifications are allowed if slack $\epsilon_j > 1$ is allowed.

Notes: Soft and Hard Margin Classifiers

What can we do if no linear seperating hyperplane exists?

Instead of focusing on find a hard margin, allow minor violations

- Introduce (positive) slack variables (patterns with slack are allowed to lie in margin)

$$\forall j = 1, \ldots n : y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1 - \epsilon_j$$

- Misclassifications are allowed if slack $\epsilon_j > 1$ is allowed.
- and we need to introduce an additional penalty term to punish non-zero ϵ_j:

$$\arg\min \frac{1}{2}||\vec{w}||^2 + C \sum_j \epsilon_j$$

subject to

$$y_j \cdot (\langle \vec{w}, \vec{x}_j \rangle + b) \geq 1 - \epsilon_j \text{ for } 1 \leq j \leq n.$$

- also solvable using Lagrange multipliers.

Notes: Multi Class Support Vector Machines

How do we separare more than two classes?

- transforms the problem into a set of binary classification problems:
  - one-against-other classifiers for each class
  - classA-against-classB classifiers for all class pairs
- use distance from hyper plane as weight.

Notes: Support Vector Regression

The key idea: change the optimization

minimize (in $\vec{w}, b$)

$$\frac{1}{2}||\vec{w}||^2$$

subject to (for any $j = 1, \ldots, n$)

$$y_j - (\langle \vec{w}, \vec{x}_j \rangle + b) \leq \epsilon$$

requires prediction error to stay under certain ϵ.
(Slack variables can allow larger errors).

Notes: Support Vectors represent what, exactly?

- Can we interpret support vectors?
  - In the kernel induced space, no less!
- Do the support vectors (number, class) tell us anything at all?
- Whats the version space of SVMs?

Support Vector Machines

- Dual Representation
  - Classifier as weighted sum over inner products of training pattern (or only support vectors) and the new pattern.
  - Training analog
- Kernel-Induced feature space
  - Transformation into higher-dimensional space (where we will hopefully be able to find a linear separation plane).
  - Representation of solution through few support vectors ($\alpha > 0$).
- Maximum Margin Classifier
  - Reduction of Capacity (Bias) via maximization of margin (and not via reduction of degrees of freedom).
  - Efficient parameter estimation.
- Relaxations
  - Soft Margin for non separable problems.

Summary

Today:

- Finding Predictors: Support Vector Machines
- Linear Classifiers
  - Rosenblatt Learning Rule
- Kernel Methods and Support Vector Machines
  - Dual Representation
  - Maximal Margins
  - Kernels
- Soft Margin Classifiers