Computational analysis of biological structures and networks

Kernel-based approaches for structured objects

Manuele Bicego

Computer Science Dept.
University of Verona
Introduction: how to implement the Bayes rule

- Generative approaches: estimation of the posterior probabilities of each class via the estimation of the \textit{conditional} probabilities and the \textit{prior} probabilities
  - More explicitly: estimation of a "\textit{model} for every class"

- Discriminative approaches: direct estimation of the posterior probabilities
  - More explicitly: direct estimation of the \textit{boundary} of the classifier
From slides of chapter 1 (Basics)
Discriminative Classification

• Many possibilities to derive directly the contour: linear discriminant, quadratic discriminant, ...

• Widely used tool: Support Vector Machines
Support Vector Machines

- Most famous and employed discriminative classifier:
  - Conceptually simple
  - Powerful and mathematically elegant
  - Fast and accurate
- SVM is a binary classifier: the feature space is divided in two regions
Support Vector Machines: the separable case

- First case: the two classes are **linearly separable**
  - There exists a hyperplane (in 2D this is a line) which permits the **perfect** separation between the two classes
Support Vector Machines: the separable case

- Many different hyperplanes allow the perfect separation of the two classes

Which is the **best** hyperplane?
Support Vector Machines: 
the separable case

- For Support Vector Machines, the best hyperplane is 
  the one which maximizes the \textbf{margin}

- The MARGIN of an hyperplane can be seen as the 
  \textbf{minimum distance} between the two classes
Support Vector Machines: the separable case

- Given a training set $D = \{x_i, y_i\}$, $x_i$ i-th point, $y_i$ label of that point (+1/-1)

- Following a “geometric” reasoning, we can explicitly write the **margin** and find the best hyperplane by solving a **constrained optimization** problem

\[
y = w^T \cdot x + b
\]
Support Vector Machines: the separable case

\[
\begin{align*}
\{ \mathbf{w}^{opt}, b^{opt} \} &= \arg \max_{\mathbf{w}, b} \frac{2}{\|\mathbf{w}\|} \\
\text{s.t. } \forall i \quad y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1
\end{align*}
\]

Maximization of margin

Constraints to ensure that the points are all in the right side of the hyperplane

or, equivalently

\[
\begin{align*}
\{ \mathbf{w}^{opt}, b^{opt} \} &= \arg \min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\| = \arg \min_{\mathbf{w}, b} \frac{1}{2}\mathbf{w}^T \mathbf{w} \\
\text{s.t. } \forall i \quad y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1
\end{align*}
\]
Support Vector Machines: the separable case

- This results in a **quadratic constrained optimization problem**, a kind of problem for which many techniques have been studied.

- Main characteristic: there exists a **single optimum**!
  - (there are not optimization problems like in Maximum Likelihood estimation)

- To solve it: transform the constrained problem in an **unconstrained** one using **Lagrange multipliers**

\[
L(w, b, \alpha_1, \cdots \alpha_N) = \frac{1}{2}w^T \cdot w - \sum_{i=1}^{N} \alpha_i (y_i(w^T \cdot x_i + b) - 1)
\]
Support Vector Machines: the separable case

- After the optimization:

\[ w^{opt} = \sum_{i=1}^{N} \alpha_i y_i x_i \]

The solution is obtained as a linear combination of the points of the training set.

\( \alpha_i = 0 \) means that the i-th point does not contribute to the solution.

Points for which \( \alpha_i \neq 0 \) are called SUPPORT VECTORS.
Support Vector Machines: the separable case

- Another view: the Support Vectors are the points which are needed to define the margin and the hyperplane
- Other points are not needed to define the margin
Support Vector Machines: the non-separable case

It is impossible to find an hyperplane for which all red points are on one side and all blue points are on the other!
Support Vector Machines: the non-separable case

- From a mathematical point of view it is not possible to satisfy all these constraints

\[
\{w^{\text{opt}}, b^{\text{opt}}\} = \arg \max_{w,b} \frac{2}{||w||}
\]

\[s.t. \quad \forall i \quad y_i (w^T \cdot x_i + b) \geq 1\]

Maximization of margin

Constraints to ensure that the points are all in the right side of the hyperplane
Support Vector Machines: the non-separable case

- Solution: relax the constraints by inserting in the formulation the so-called **slack variables**
- Variables which permit an “uncorrect” position of some points

\[ \xi_i = \max(0, -y_i(w^T \cdot x_i + b)) \]

This is not 1 anymore, it is less than 1 (can also be negative if the slack variable is greater than 1)
Support Vector Machines: the non-separable case

- Idea: permit to some points to go over the hyperplane, by paying a price (encoded in the slack variable)
- Now the goal is to maximize the margin and to minimize the “errors”

\[
\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i
\]

\[
\text{s.t. } \forall i \quad y_i (w^T \cdot x_i + b) \geq 1 - \xi_i, \quad \xi \geq 0
\]

\(C\) represents a parameter (to be set by the user) which regulates the impact of this cost

High \(C\): try to avoid the “errors” as much as possible (risk: overtraining)
Support Vector Machines: the non-separable case

- The value of $\xi_i$ indicates the position of the point $x_i$ with respect to the hyperplane.

- $\xi_i = 0$: correct classification
- $0 < \xi_i \leq 1$: correct classification but in the margin zone
- $\xi_i > 1$: wrong classification
Support Vector Machines: the non-separable case

- Again the solution is obtained as a linear combination of the points of the training set:

\[ \mathbf{w}^{opt} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \]

The support vectors are the points for which \( \alpha_i \) is different than zero:

- points defining the hyperplane (as before)
- points in the wrong side of the margin
Support Vector Machines: the kernel trick

- There are cases for which an hyperplane represents a too simplistic solution
Support Vector Machines: the kernel trick

- Solution: to project the points in a higher dimensional space, where discrimination is easier
Support Vector Machines: the kernel trick

But... how to select the mapping?

and... more importantly, how to deal with the curse of dimensionality of the space induced by the mapping?

with the **Kernel Trick!**
Support Vector Machines: the kernel trick

Observation 1: the optimization problem

\[
\begin{align*}
\min_{\mathbf{w}, b} & \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} \\
\text{s.t.} & \quad \forall i \quad y_i (\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1
\end{align*}
\]

can be rewritten as

\[
\begin{align*}
\max_{\alpha_i} \left( \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \right) \\
\text{s.t.} \quad \forall i \quad \sum_i \alpha_i y_i = 0 \quad \alpha_i \geq 0
\end{align*}
\]
Support Vector Machines: the kernel trick

Observation 2: The testing phase (i.e. how to decide to which class a given point $x$ belongs) is done via the sign function:

$$\text{class}(x) = \text{sign}(w^T \cdot x + b) = \begin{cases} +1 & \text{if } w^T \cdot x + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

In other words we have to evaluate:

$$w^T \cdot x + b = \sum_{i=1}^{N} \alpha_i y_i x_i \cdot x$$

Remember that:

$$w^{opt} = \sum_{i=1}^{N} \alpha_i y_i x_i$$
Support Vector Machines: the kernel trick

- In both cases, the points do not appear alone, but always in relation with other points (via a dot product)
Support Vector Machines: the kernel trick

If we apply the mapping $\Phi$ to the points (the mapping which permits to go to the high dimensional space), we need to compute the dot product in the high dimensional space

$$\max_{\alpha_i} \left( \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi(x_j) \right)$$

$$\sum_{i=1}^{N} \alpha_i y_i \Phi(x_i) \cdot \Phi(x)$$
Support Vector Machines: the kernel trick

- **Therefore:** to perform training and testing, we don't need to *explicitly define* the mapping $\Phi$, but only to define a way to compute the dot product $\Phi(x_i)\Phi(x_j)$.

- **Kernel trick:** to define a specific function, called *kernel*, which represents a dot product in a space induced by a mapping $\Phi$.

\[ K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \]
Support Vector Machines:
the kernel trick

Example:

\[ x \in \mathbb{R}^2, x = [x_1 \ x_2] \]

\[ \Phi(x) = [1, \ x_1^2, \ \sqrt{2}x_1x_2, \ x_2^2, \ \sqrt{2}x_1, \ \sqrt{2}x_2] \]

\[ \Phi(x_i) \cdot \Phi(x_j) = (1 + x_i \cdot x_j)^2 = K(x_i, x_j) \]

Exercise: derive the proof!
Support Vector Machines: the kernel trick

• Please note that the kernel is a function working in the original space of $\mathbf{x}$ (it is a function of $\mathbf{x}_i$ and $\mathbf{x}_j$)

\[ \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_j) \]

• We don’t have anymore the problem of the curse of dimensionality in the space induced by $\Phi$!!
The kernel trick: summary

- To solve complex problems we can use an hyperplane in a **higher dimensional** space defined with a mapping $\Phi$

- Since training and testing of SVM only involve dot products of points, we don’t have to explicitly define the mapping $\Phi$ but only the dot product in the space induced by $\Phi$

- We can use a **kernel**, a function which defines a dot product in the space induced by $\Phi$
The kernel trick: summary

Advantages:

1. we don't have to define the mapping $\Phi$, but only the kernel $k(x_i, x_j)$ (training and testing are performed by using the kernel)
   \[\rightarrow\text{problem of defining the mapping: no!}\]

2. The kernel is a function working in the original space of $x_i$ (it is a function of $x_i$ and $x_j$)
   \[\rightarrow\text{problem of curse of dimensionality of the space induced by } \Phi: \text{ no!}\]
The kernel trick: summary

3. The SVM is now a plane in a high dimensional space, which can be enough to perform discrimination

→ The hyperplane in the high dimensional space represents a **highly non linear** boundary in the original space
How to define a kernel

- The Kernel is a function which represents a dot product in a higher dimensional space

\[ K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \]

- But... what is a dot product?
  - The dot product is a measure of the relation between two points.
The dot product:

For similar points, the dot product is high and positive, for dissimilar points, the dot product is negative.

In summary: the dot product represents a measure of similarity between vectors.
How to define a kernel

- Therefore, to define a kernel we have to define a similarity measure between points

- This similarity measure, to properly represent a kernel (i.e. dot product in a space induced by the mapping \( \Phi \)) should obey to some mathematical conditions (the **Mercer's conditions**).
Example: the Radial Basis Function (RBF) kernel

\[ K(x, z) = e^{-\frac{||x - z||^2}{2\sigma^2}} \]
Examples of kernels

Linear

Polynomial

Radial basis function

Sigmoid

\[ K(x, z) = x \cdot z \]

\[ K(x, z) = (x \cdot z + 1)^p \]

\[ K(x, z) = e^{-\frac{\|x - z\|^2}{2\sigma^2}} \]

\[ K(x, z) = \tanh(a x \cdot z + b) \]
Kernels for structured objects
SVM for structured objects

- Question: can we use Support Vector Machines with structured objects?
- In principle it is not possible: E.g. how can we derive an hyperplane passing by two sequences?
- But... if we use the kernelized version of SVM, we only have to define a kernel between sequences
- This permits to project sequences in a high dimensional vectorial space where we can derive an hyperplane!!
Kernels for structured objects

Therefore...

- To apply SVM to structured objects we simply have to **define kernels between structured objects**
  - i.e. to define similarities between structured objects obeying to some mathematical conditions (Mercer's conditions)
Kernels for structured objects

- NOTE: recently it has been shown that, even if some of the mathematical conditions specified by the Mercer theorem are not fulfilled, we can still use SVMs

  - we loose some mathematical properties, (e.g. no guarantee of convergence of the training or unique minimum), but we can still effectively use them!

  - These “not proper” kernels are referred to as Indefinite Kernels
Kernels for structured objects

... let’s see some kernels for structured objects

• Kernels from distances: general formulation
• Histogram Intersection Kernel
• Generative Kernels
  • Fisher Kernel
• Convolution Kernels
Kernels from distances: a general formulation

- We studied many distances/dissimilarities suitable for structured objects
- Given a distance $d(o_i,o_j)$, we can derive a kernel from it with this formula:

$$K(o_i, o_j) = e^{-\frac{d(o_i, o_j)}{\gamma}}$$

Where $\gamma$ represents a scaling parameter
Kernels from distances: a general formulation

- If \( d \) is a **true metric**, then the function \( K \) is a proper kernel.
- If the distance \( d \) is not a metric (common case when dealing with structured objects and bioinformatics), then the function is not a kernel anymore.
- But... it can be used as well inside the SVM as an indefinite kernel!
Histogram intersection kernel

- Widely employed kernel in image modelling
  - Appropriate also when images have different size

Main idea: compare the histograms of the two images
Histogram intersection kernel

Histogram

Every bin measures how many pixels of that color are present in the image.
Histogram intersection kernel

- The kernel is defined as the intersection between the histograms of the two images
Histogram intersection kernel

• Main idea: a large intersection indicates that there are many pixels in the two images with a similar color, so images are possibly similar!

• Mathematically:

\[
A = \{a_1, \cdots, a_m\} \quad \text{histogram of the image } I_A \\
B = \{b_1, \cdots, b_m\} \quad \text{histogram of the image } I_B \\
(m \text{ bins})
\]

\[
K(I_A, I_B) = \sum_{i=1}^{m} \min(a_i, b_i)
\]
Generative kernels

- Hybrid generative-discriminative strategy, which tries to put together the best of generative and discriminative paradigms
  - Description capabilities of generative models
  - Discrimination skills of discriminative approaches
- Main idea: to exploit a generative model to compute a kernel between objects
  - The kernel is obtained through the generative model
A generative model able to describe the objects (e.g. HMMs for proteins) is used to derive a similarity between the objects. The generative model is K(Object 1, Object 2).

Two structured objects.
The Fisher Kernel

- Most famous generative kernel:
  - First step: define a generative model for the structured objects (for example by learning a Bayesian Net from a training set)
  - The kernel between two objects is computed starting from the \textbf{derivative of the likelihood} of the two objects with respect to the model

\[
K(X_i, X_j) = U_{X_i}^T \mathcal{I}^{-1} U_{X_j}
\]

\[
\mathcal{I} = E_X \{ U_X U_X^T \} \quad \rightarrow \quad \text{Fisher Information Matrix}
\]

\[
U_X = \nabla_\theta \log P(X|\theta) \quad \rightarrow \quad \text{Fisher Score Vector}
\]
The Fisher Kernel

- It can be approximated with

\[
K(X_i, X_j) = \nabla_\theta \log P(X_i|\theta) \cdot \nabla_\theta \log P(X_j|\theta)
\]

→ the similarity (dot product) between the value of the derivative of the likelihood in the two different points

*Leaving math aside, the main concept is that we can define the kernel by looking at the way the generative model explains the two objects (e.g. the likelihood, which can be different for every object)*
Main idea:
- Decompose structured objects into comparable parts.
- Aggregate the values of similarity measures for individual parts.

\[
K(x, y) = \sum_{(\text{sets of parts of } x)} \sum_{(\text{sets of parts of } y)} \prod_{d} K_d(x_d, y_d)
\]

Similarity between corresponding parts
Convolution Kernel

- Example: to compare two cars we compare all parts of the two cars (1. decompose the cars into parts, 2. compare parts, 3. aggregate comparison)
Convolution Kernels

- Widely used for strings, graphs, trees
  - Strings: subparts are substrings
  - Graphs: subparts are subgraphs
  - Trees: subparts are subtrees

- Special data structures and algorithms are needed for efficiency.
A final note

- SVMs are not the sole method based on the kernel-trick
- Actually, in recent years, many classical techniques have been revisited to derive a **kernel formulation**:
  - Kernel PCA, Kernel ICA, Kernel K-means, ...

- **Kernel formulation**: formulate the problem so that training and testing are defined only in terms of dot products
  - this permits to exploit the kernel trick!
A final note

- Once defined a kernel between structured objects, we can apply all these techniques (providing an answer to questions like: *how can I use PCA for strings?*)

- Clearly the problem is to define *informative* kernels, i.e. kernels which permit to capture the *true similarity* between structured objects
  - Still an open problem in the scientific community
Further readings