QSAR and Virtual Screening with Support Vector Machines

Jean-Philippe Vert
Jean-Philippe.Vert@ensmp.fr

Center for Computational Biology
Ecole des Mines de Paris, ParisTech

Colloquium Chemiometricum Mediterraneum, Septembre 7th, 2007
More formally...

Objective

Build models to predict biochemical properties $Y$ of small molecules from their structures $X$, using a training set of $(X, Y)$ pairs.

Structures $X$

$C_{15}H_{14}ClN_3O_3$

Properties $Y$

• binding to a therapeutic target,
• pharmacokinetics (ADME),
• toxicity...
Classical approaches

Two steps

1. Map each molecule to a **vector of fixed dimension using molecular descriptors**
   - Global properties of the molecules (mass, logP...)
   - 2D and 3D descriptors (substructures, fragments, ....)

2. Apply an algorithm for **regression or pattern recognition**.
   - PLS, ANN, ...

Example: 2D structural keys
Which descriptors?

**Difficulties**

- Many descriptors are needed to characterize various features (in particular for 2D and 3D descriptors)
- But too many descriptors are harmful for memory storage, computation speed, statistical estimation

**Our approach**

Work implicitly with many descriptors:

- Allows to consider a large number (potentially infinite) of potentially important features.
- Computation trick: no need to compute and store explicitly the vectors
- Statistical trick: use regularized statistical algorithm to handle the problem of large dimension
Difficulties

- Many descriptors are needed to characterize various features (in particular for 2D and 3D descriptors)
- But too many descriptors are harmful for memory storage, computation speed, statistical estimation

Our approach

Work implicitly with many descriptors:

- Allows to consider a large number (potentially infinite) of potentially important features.
- Computation trick: no need to compute and store explicitly the vectors
- Statistical trick: use regularized statistical algorithm to handle the problem of large dimension
Outline

1. Support Vector Machines and kernels
2. 2D Kernel
3. 3D Pharmacophore Kernel
4. Conclusion
Outline

1. Support Vector Machines and kernels
2. 2D Kernel
3. 3D Pharmacophore Kernel
4. Conclusion
Simplest SVM
Simplest SVM
Simplest SVM
Choose the linear separator which is as far as possible to the closest point (maximize the margin $\gamma$).

Computationally: this boils down to a simple convex quadratic optimization problem (next slide). No local minima, efficient algorithms for up to 100,000 points.

Statistically: this allows resistance to overfitting in large dimension (statistical learning theory).
After some algebra it is obtained by solving in $\alpha \in \mathbb{R}^n$ the following quadratic program:

$$\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j x_i^\top x_j - \sum_{i=1}^{n} \alpha_i \\
\text{subject to} & \quad \alpha_i \geq 0, \quad i = 1, \ldots, n, \\
& \quad \sum_{i=1}^{n} \alpha_i y_i = 0.
\end{align*}$$

Once $\alpha$ is found, the classification function is the sign of:

$$f(x) = \sum_{i=1}^{n} \alpha_i x_i^\top x + b.$$
Linear SVM: non-separable case

Implementation

- Solution: find a trade-off between large margin and few misclassification
- Simple and elegant mathematical translation: replace $0 \leq \alpha_i$ by $0 \leq \alpha_i \leq C$, for some constant $C > 0$, in the optimization problem.
Linear SVM: non-separable case

Implementation

- Solution: find a trade-off between large margin and few misclassification
- Simple and elegant mathematical translation: replace $0 \leq \alpha_i$ by $0 \leq \alpha_i \leq C$, for some constant $C > 0$, in the optimization problem.
Linear SVM: non-separable case

Implementation

- Solution: find a trade-off between large margin and few misclassification.
- Simple and elegant mathematical translation: replace $0 \leq \alpha_i$ by $0 \leq \alpha_i \leq C$, for some constant $C > 0$, in the optimization problem.
Linear SVM: non-separable case

Implementation

- Solution: find a trade-off between large margin and few misclassification
- Simple and elegant mathematical translation: replace $0 \leq \alpha_j$ by $0 \leq \alpha_j \leq C$, for some constant $C > 0$, in the optimization problem.
The idea

- Define a (nonlinear) mapping

\[ \phi : \mathcal{X} \to \mathcal{F} \subset \mathbb{R}^P. \]

- Run a linear SVM in the feature space.
The idea

- Define a (nonlinear) mapping

\[ \phi : \mathcal{X} \rightarrow \mathcal{F} \subset \mathbb{R}^p. \]

- Run a linear SVM in the feature space.
The idea

- Define a (nonlinear) mapping

\[ \phi : \mathcal{X} \rightarrow \mathcal{F} \subset \mathbb{R}^p. \]

- Run a linear SVM in the feature space.
The idea

- Define a (nonlinear) mapping

\[ \phi : \mathcal{X} \rightarrow \mathcal{F} \subset \mathbb{R}^p. \]

- Run a linear SVM in the feature space.
Nonlinear SVM: implementation

Solve in $\alpha \in \mathbb{R}^n$:

$$\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \Phi(x_i)^\top \Phi(x_j) - \sum_{i=1}^{n} \alpha_i \\
\text{subject to} & \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n, \\
& \quad \sum_{i=1}^{n} \alpha_i y_i = 0.
\end{align*}$$

Once $\alpha$ is found, the classification function is the sign of:

$$f(x) = \sum_{i=1}^{n} \alpha_i \Phi(x_i)^\top \Phi(x) + b$$
The kernel tricks

Important idea!

- To any mapping $\Phi: \mathcal{X} \rightarrow \mathcal{F}$ corresponds a kernel function $K$:

$$K(x, x') = \Phi(x)\top \Phi(x').$$

- SVM only need $K$, rather than $\Phi$:

$$\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i \\
\text{subject to} & \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n, \\
& \quad \sum_{i=1}^{n} \alpha_i y_i = 0.
\end{align*}$$
Example: polynomial kernel

For \( x = (x_1, x_2) \top \in \mathbb{R}^2 \), let \( \Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3 \):

\[
K(x, x') = x_1^2 x_1'^2 + 2x_1 x_2 x_1' x_2' + x_2^2 x_2'^2
= (x_1 x_1' + x_2 x_2')^2
= (x \top x')^2.
\]
SVM and kernel: summary

**Data representation**

- Data do not need to be explicitly vectors
- A similarity function $K(x, x')$ between data is enough
- $K$ must be symmetric and positive definite to be a valid kernel
- Allows nonlinear function estimation or working with non-vectorial data without any change in the algorithm!

**Performance**

- State-of-the-art in many real-world applications
- Resistant to large dimensions
SVM and kernel: summary

Data representation
- Data do not need to be explicitly vectors
- A similarity function \( K(x, x') \) between data is enough
- \( K \) must be symmetric and positive definite to be a valid kernel
- Allows nonlinear function estimation or working with non-vectorial data without any change in the algorithm!

Performance
- State-of-the-art in many real-world applications
- Resistant to large dimensions
SVM and kernel: summary

Data representation

- Data do not need to be explicitly vectors
- A similarity function $K(x, x')$ between data is enough
- $K$ must be symmetric and positive definite to be a valid kernel
- Allows nonlinear function estimation or working with non-vectorial data without any change in the algorithm!

Performance

- State-of-the-art in many real-world applications
- Resistant to large dimensions
Other algorithms

Kernel methods

You don’t want to try SVM, or your problem is not binary classification, but you would like to benefit from the kernel trick (nonlinearity, structured data etc...)? Try other kernel methods that extend your favorite algorithm to handle kernels:

- kernel PLS,
- kernel PCA,
- kriging,
- kernel perceptron,
- kernel logistic regression,
- and many more!
In order to use kernel methods for QSAR / virtual screening, all we need are kernels for molecules:

\[ K(\text{molecule}_1, \text{molecule}_2). \]

**Strategy 1**: use well-known molecular descriptors to represent molecules \( m \) as vectors \( \Phi(m) \), and then use kernels for vectors, e.g.:

\[ K(m_1, m_2) = \Phi(m_1)^\top \Phi(m_2). \]

**Strategy 2**: invent new kernels to do things you can not do with strategy 1, such as using an infinite number of descriptors. We will now see two examples of this strategy, extending 2D and 3D molecular descriptors.
In order to use kernel methods for QSAR / virtual screening, all we need are kernels for molecules:

\[ K(\text{molecule}_1, \text{molecule}_2). \]

**Strategy 1:** use well-known molecular descriptors to represent molecules \( m \) as vectors \( \Phi(m) \), and then use kernels for vectors, e.g.:

\[ K(m_1, m_2) = \Phi(m_1)^\top \Phi(m_2). \]

**Strategy 2:** invent new kernels to do things you can not do with strategy 1, such as using an infinite number of descriptors. We will now see two examples of this strategy, extending 2D and 3D molecular descriptors.
In order to use kernel methods for QSAR / virtual screening, all we need are kernels for molecules:

\[ K(\text{molecule}_1, \text{molecule}_2). \]

**Strategy 1:** use well-known molecular descriptors to represent molecules \( m \) as vectors \( \Phi(m) \), and then use kernels for vectors, e.g.:

\[ K(m_1, m_2) = \Phi(m_1)^\top \Phi(m_2). \]

**Strategy 2:** invent new kernels to do things you can not do with strategy 1, such as using an infinite number of descriptors. We will now see two examples of this strategy, extending 2D and 3D molecular descriptors.
Outline

1. Support Vector Machines and kernels
2. 2D Kernel
3. 3D Pharmacophore Kernel
4. Conclusion
Motivation: 2D Fingerprints

Features

A vector indexed by a large set of molecular fragments

Pros
- Many features
- Easy to detect

Cons
- Too many features?
- Hashing $\Rightarrow$ clashes
Motivation: 2D Fingerprints

Features

A vector indexed by a large set of molecular fragments

Pros

- Many features
- Easy to detect

Cons

- Too many features?
- Hashing $\Rightarrow$ clashes
Let $\Phi(x)$ the vector of fragment counts:

- Long fragments lead to large dimensions:
  SVM can learn in high dimension

- $\Phi(x)$ is too long to be stored, and hashes induce clashes:
  SVM do not need $\Phi(x)$, they just need the kernel

$$K(x, x') = \phi(x)\top \phi(x').$$
2D fingerprint kernel

Definition

- For any $d > 0$ let $\phi_d(x)$ be the vector of counts of all fragments of length $d$:

  $\phi_1(x) = \begin{pmatrix} \#(C), \#(O), \#(N), \ldots \end{pmatrix}^T$

  $\phi_2(x) = \begin{pmatrix} \#(C-C), \#(C=O), \#(C-N), \ldots \end{pmatrix}^T$ etc...

- The 2D fingerprint kernel is defined, for $\lambda < 1$, by

  \[
  K_{2D}(x, x') = \sum_{d=1}^{\infty} \lambda^d \phi_d(x)^T \phi_d(x').
  \]

- This is an inner product in the space of 2D fingerprints of infinite length.
Theorem

The 2D fingerprint kernel between two molecules $x$ and $x'$ can be computed with a worst-case complexity $O\left((|x| \times |x'|)^3\right)$ (much faster in practice).

Remarks

- The complexity is not related to the length of the fragments considered (although faster computations are possible if the length is limited).
- Solves the problem of clashes and memory storage.
- Allows to work with infinite-length fingerprints without computing them!
Theorem

The 2D fingerprint kernel between two molecules $x$ and $x'$ can be computed with a worst-case complexity $O\left((|x| \times |x'|)^3\right)$ (much faster in practice).

Remarks

- The complexity is not related to the length of the fragments considered (although faster computations are possible if the length is limited).
- Solves the problem of clashes and memory storage.
- Allows to work with infinite-length fingerprints without computing them!
Rephrase the kernel computation as that as counting the number of walks on a graph (the product graph)

\[
\begin{align*}
X &= \lambda A + \lambda^2 A^2 + \lambda^3 A^3 + \ldots = (I - \lambda A)^{-1} - I.
\end{align*}
\]
Extensions 1: label enrichment

Atom relabeling with the Morgan index

Compromise between fingerprints and structural keys features.
Other relabeling schemes are possible.
Faster computation with more labels (less matches implies a smaller product graph).
Extensions 2: filter out tottering fragments

Tottering fragments

Solution: graph transform
Extensions 3: tree-like fragments

\[
\begin{align*}
\text{t} &= \begin{array}{l}
\text{N} - \text{N} - \text{C} - \text{C} - \text{C} \\
\end{array} : 1 \times \sqrt{w(t)} \\
\text{t} &= \begin{array}{l}
\text{N} - \text{N} = \text{O} - \text{C} \\
\end{array} : 1 \times \sqrt{w(t)} \\
\text{t} &= \begin{array}{l}
\text{N} - \text{N} = \text{O} - \text{C} \\
\end{array} : 1 \times \sqrt{w(t)} \\
\text{t} &= \begin{array}{l}
\text{N} - \text{N} - \text{C} - \text{C} - \text{C} \\
\end{array} : 1 \times \sqrt{w(t)}
\end{align*}
\]
Experiments

MUTAG dataset

- aromatic/hetero-aromatic compounds
- high mutagenic activity / no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compounds: 125 + / 63 -

Results

10-fold cross-validation accuracy

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progol1</td>
<td>81.4%</td>
</tr>
<tr>
<td>2D kernel</td>
<td>91.2%</td>
</tr>
</tbody>
</table>

Jean-Philippe Vert (ParisTech)  QSAR and Virtual Screening with SVM
AUC as a function of the branching factors for different tree depths (from Mahé et al., 2007).
1. Support Vector Machines and kernels
2. 2D Kernel
3. 3D Pharmacophore Kernel
4. Conclusion
A set of 3 atoms, and 3 inter-atom distances:

\[ T = \{ ((x_1, x_2, x_3), (d_1, d_2, d_3)) \mid x_i \in \{\text{atom types}\}; d_i \in \mathbb{R} \} \]
3D fingerprint kernel

1. **Discretize** the space of pharmacophores $\mathcal{T}$ (e.g., 6 atoms or groups of atoms, 6-7 distance bins) into a finite set $\mathcal{T}_d$

2. Count the number of occurrences $\phi_t(x)$ of each pharmacophore bin $t$ in a given molecule $x$, to form a **pharmacophore fingerprint**.

3D kernel

A simple 3D kernel is the **inner product of pharmacophore fingerprints**:

$$K(x, x') = \sum_{t \in \mathcal{T}_d} \phi_t(x) \phi_t(x').$$
Discretization of the pharmacophore space

Common issues

1. If the bins are **too large**, then they are **not specific enough**
2. If the bins are **too large**, then they are **too specific**

In all cases, the **arbitrary position of boundaries between bins** affects the comparison:

\[
\rightarrow d(x_1, x_3) < d(x_1, x_2)
\]

**BUT**

\[
\text{bin}(x_1) = \text{bin}(x_2) \neq \text{bin}(x_3)
\]
Kernels between pharmacophores

A small trick

\[
K(x, y) = \sum_{t \in \mathcal{I}_d} \phi_t(x) \phi_t(y)
\]

\[
= \sum_{t \in \mathcal{I}_d} \left( \sum_{p_x \in \mathcal{P}(x)} 1(\text{bin}(p_x) = t) \right) \left( \sum_{p_y \in \mathcal{P}(y)} 1(\text{bin}(p_y) = t) \right)
\]

\[
= \sum_{p_x \in \mathcal{P}(x)} \sum_{p_y \in \mathcal{P}(y)} 1(\text{bin}(p_x) = \text{bin}(p_y))
\]

General pharmacophore kernel

\[
K(x, y) = \sum_{p_x \in \mathcal{P}(x)} \sum_{p_y \in \mathcal{P}(y)} K_P(p_x, p_y)
\]
New pharmacophore kernels

- Discretizing the pharmacophore space is equivalent to taking the following kernel between individual pharmacophores:

\[ K_P(p_1, p_2) = 1 \left( \text{bin}(p_x) = \text{bin}(p_y) \right) \]

- For general kernels, there is no need for discretization!
- For example, if \( d(p_1, p_2) \) is a Euclidean distance between pharmacophores, take:

\[ K_P(p_1, p_2) = \exp(-\gamma d(p_1, p_2)) . \]
Experiments

4 public datasets

- BZR: ligands for the benzodiazepine receptor
- COX: cyclooxygenase-2 inhibitors
- DHFR: dihydrofolate reductase inhibitors
- ER: estrogen receptor ligands

<table>
<thead>
<tr>
<th></th>
<th>TRAIN</th>
<th></th>
<th>TEST</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pos</td>
<td>Neg</td>
<td>Pos</td>
<td>Neg</td>
</tr>
<tr>
<td>BZR</td>
<td>94</td>
<td>87</td>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>COX</td>
<td>87</td>
<td>91</td>
<td>61</td>
<td>64</td>
</tr>
<tr>
<td>DHFR</td>
<td>84</td>
<td>149</td>
<td>42</td>
<td>118</td>
</tr>
<tr>
<td>ER</td>
<td>110</td>
<td>156</td>
<td>70</td>
<td>110</td>
</tr>
</tbody>
</table>
## Experiments

### Results (accuracy)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>BZR</th>
<th>COX</th>
<th>DHFR</th>
<th>ER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D (Tanimoto)</td>
<td>71.2</td>
<td>63.0</td>
<td>76.9</td>
<td>77.1</td>
</tr>
<tr>
<td>3D fingerprint</td>
<td>75.4</td>
<td>67.0</td>
<td>76.9</td>
<td>78.6</td>
</tr>
<tr>
<td>3D not discretized</td>
<td><strong>76.4</strong></td>
<td><strong>69.8</strong></td>
<td><strong>81.9</strong></td>
<td><strong>79.8</strong></td>
</tr>
</tbody>
</table>
Outline

1. Support Vector Machines and kernels
2. 2D Kernel
3. 3D Pharmacophore Kernel
4. Conclusion
SVM is a powerful and flexible machine learning algorithm. The kernel trick allows the manipulation of non-vectorial objects at the cost of defining a kernel function.

The 2D kernel for molecule extends classical fingerprint-based approaches. It solves the problem of bit clashes, allows infinite fingerprints and various extensions.

The 3D kernel for molecule extends classical pharmacophore fingerprint-based approaches. It solves the problems of bit clashes and of discretization.

Both kernels improve upon their classical counterparts, and provide competitive results on benchmark datasets.
Acknowledgements

- Pierre Mahé (CBIO)
- Tatsuya Akutsu, Nobuhisa Ueda, Jean-Luc Perret (Kyoto University)
- Liva Ralaivola (U Marseille)
References


