Natural Language Processing and Information Retrieval

Kernel Methods

Alessandro Moschitti

Department of information and communication technology
University of Trento
Email: moschitti@dit.unitn.it
Linear Classifier

- The equation of a hyperplane is
  \[ f(\vec{x}) = \vec{x} \cdot \vec{w} + b = 0, \quad \vec{x}, \vec{w} \in \mathbb{R}^n, b \in \mathbb{R} \]
- \( \vec{x} \) is the vector representing the classifying example
- \( \vec{w} \) is the gradient of the hyperplane
- The classification function is
  \[ h(x) = \text{sign}(f(x)) \]
The main idea of Kernel Functions

- Mapping vectors in a space where they are linearly separable \( \tilde{x} \rightarrow \phi(\tilde{x}) \)
A mapping example

- Given two masses $m_1$ and $m_2$, one is constrained
- Apply a force $f_a$ to the mass $m_1$
- Experiments
  - Features $m_1$, $m_2$ and $f_a$
- We want to learn a classifier that tells when a mass $m_1$ will get far away from $m_2$
- If we consider the Gravitational Newton Law
  \[ f(m_1, m_2, r) = C \frac{m_1 m_2}{r^2} \]
- we need to find when $f(m_1, m_2, r) < f_a$
A mapping example (2)

\[ \tilde{x} = (x_1, \ldots, x_n) \rightarrow \phi(\tilde{x}) = (\phi_1(\tilde{x}), \ldots, \phi_n(\tilde{x})) \]

- The gravitational law is not linear so we need to change space

\[ (f_a, m_1, m_2, r) \rightarrow (k, x, y, z) = (\ln f_a, \ln m_1, \ln m_2, \ln r) \]

- As

\[ \ln f(m_1, m_2, r) = \ln C + \ln m_1 + \ln m_2 - 2 \ln r = c + x + y - 2z \]

- We need the hyperplane

\[ \ln f_a - \ln m_1 - \ln m_2 + 2 \ln r - \ln C = 0 \]

\[ (\ln m_1, \ln m_2, -2 \ln r) \cdot (x, y, z) - \ln f_a + \ln C = 0 \], we can decide without error if the mass will get far away or not
A kernel-based Machine Perceptron training

\[
\begin{align*}
\tilde{w}_0 & \leftarrow 0; b_0 \leftarrow 0; k \leftarrow 0; R \leftarrow \max_{1 \leq i \leq l} \| x_i \|
\end{align*}
\]

\begin{algorithm}
do
  \textbf{for} i = 1 \textbf{to} \ell
  \begin{aligned}
  & \text{if } y_i (\tilde{w}_k \cdot \tilde{x}_i + b_k) \leq 0 \text{ then} \\
  & \quad \tilde{w}_{k+1} = \tilde{w}_k + \eta y_i \tilde{x}_i \\
  & \quad b_{k+1} = b_k + \eta y_i R^2 \\
  & \quad k = k + 1 \\
  \end{aligned}
\textbf{endfor}
\textbf{while} an error is found
\textbf{return} k, (\tilde{w}_k, b_k)
\end{algorithm}
Dual Representation for Classification

- Each step of perceptron only training data is added with a certain weight
  \[ \tilde{w} = \sum_{j=1..\ell} \alpha_j y_j \tilde{x}_j \]

- So the classification function
  \[ \text{sgn}(\tilde{w} \cdot \tilde{x} + b) = \text{sgn}\left( \sum_{j=1..\ell} \alpha_j y_j \tilde{x}_j \cdot \tilde{x} + b \right) \]

- Note that data only appears in the scalar product
Dual Representation for Learning

- as well as the updating function

\[
\text{if } y_i \left( \sum_{j=1}^{\ell} \alpha_j y_j \bar{x}_j \cdot \bar{x}_i + b \right) \leq 0 \text{ then } \alpha_i = \alpha_i + \eta
\]

- The learning rate \( \eta \) only affects the re-scaling of the hyperplane, it does not affect the algorithm, so we can fix \( \eta = 1 \).
Dual Perceptron algorithm and Kernel functions

- We can rewrite the classification function as

\[ h(x) = \text{sgn}(\vec{w}_\phi \cdot \phi(\vec{x}) + b_\phi) = \text{sgn}(\sum_{j=1..\ell} \alpha_j y_j \phi(\vec{x}_j) \cdot \phi(\vec{x}) + b_\phi) = \]

\[ = \text{sgn}(\sum_{i=1..\ell} \alpha_j y_j k(\vec{x}_j, \vec{x}) + b_\phi) \]

- As well as the updating function

\[ \text{if } y_i \left( \sum_{j=1..\ell} \alpha_j y_j k(\vec{x}_j, \vec{x}_i) + b_\phi \right) \leq 0 \text{ allora } \alpha_i = \alpha_i + \eta \]

- The learning rate \( \eta \) does not affect the algorithm so we set it to \( \eta = 1 \).
Dual optimization problem of SVMs

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j + \frac{1}{C} \delta_{ij})$$

$$\alpha_i \geq 0, \quad \forall i = 1, \ldots, m$$

$$\sum_{i=1}^{m} y_i \alpha_i = 0$$
Kernels in Support Vector Machines

- In Soft Margin SVMs we maximize:

\[
\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j + \frac{1}{C} \delta_{ij})
\]

- By using kernel functions we rewrite the problem as:

\[
\left\{ \begin{array}{l}
\text{maximize} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j (k(o_i, o_j) + \frac{1}{C} \delta_{ij}) \\
\alpha_i \geq 0, \quad \forall i = 1, \ldots, m \\
\sum_{i=1}^{m} y_i \alpha_i = 0
\end{array} \right. 
\]
Kernel Function Definition

**Def. 2.26** A kernel is a function \( k \), such that \( \forall \vec{x}, \vec{z} \in X \)

\[
k(\vec{x}, \vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})
\]

where \( \phi \) is a mapping from \( X \) to an (inner product) feature space.

- Kernels are the product of mapping functions such as

\[
\vec{x} \in \mathbb{R}^n, \quad \phi(\vec{x}) = (\phi_1(\vec{x}), \phi_2(\vec{x}), ..., \phi_m(\vec{x})) \in \mathbb{R}^m
\]
The Kernel Gram Matrix

With KM-based learning, the **sole** information used from the training data set is the Kernel Gram Matrix

\[
K_{\text{training}} = \begin{bmatrix}
    k(x_1, x_1) & k(x_1, x_2) & \ldots & k(x_1, x_m) \\
    k(x_2, x_1) & k(x_2, x_2) & \ldots & k(x_2, x_m) \\
    \vdots & \vdots & \ddots & \vdots \\
    k(x_m, x_1) & k(x_m, x_2) & \ldots & k(x_m, x_m)
\end{bmatrix}
\]

If the kernel is valid, K is symmetric definite-positive.
Def. B.11 Eigen Values
Given a matrix $A \in \mathbb{R}^{m} \times \mathbb{R}^{n}$, an egeinvalue $\lambda$ and an egeinvector $\vec{x} \in \mathbb{R}^{n} - \{\vec{0}\}$ are such that

$$A\vec{x} = \lambda \vec{x}$$

Def. B.12 Symmetric Matrix
A square matrix $A \in \mathbb{R}^{n} \times \mathbb{R}^{n}$ is symmetric iff $A_{ij} = A_{ji}$ for $i \neq j$ $i = 1, \ldots, m$ and $j = 1, \ldots, n$, i.e. iff $A = A'$. 

Def. B.13 Positive (Semi-) definite Matrix
A square matrix $A \in \mathbb{R}^{n} \times \mathbb{R}^{n}$ is said to be positive (semi-) definite if its eigenvalues are all positive (non-negative).
Valid Kernels cont’d

Proposition 2.27 (Mercer’s conditions)
Let $X$ be a finite input space with $K(\bar{x}, \bar{z})$ a symmetric function on $X$. Then $K(\bar{x}, \bar{z})$ is a kernel function if and only if the matrix

$$k(\bar{x}, \bar{z}) = \phi(\bar{x}) \cdot \phi(\bar{z})$$

is positive semi-definite (has non-negative eigenvalues).

- If the matrix is positive semi-definite then we can find a mapping $\phi$ implementing the kernel function
Mercer’s Theorem (finite space)

- Let us consider $K = \left(K(\tilde{x}_i, \tilde{x}_j)\right)^n_{i,j=1}$

- $K$ symmetric $\Rightarrow \exists V: K = V\Lambda V'$ for Takagi factorization of a complex-symmetric matrix, where:
  - $\Lambda$ is the diagonal matrix of the eigenvalues $\lambda_t$ of $K$
  - $\tilde{v}_t = (v_{ti})^n_{i=1}$ are the eigenvectors, i.e. the columns of $V$

- Let us assume lambda values non-negative

$$\phi: \tilde{x}_i \rightarrow \left(\sqrt{\lambda_t}v_{ti}\right)^n_{t=1} \in \mathbb{R}^n, \ i = 1,..,n$$
Mercer’s Theorem  
(sufficient conditions)

Therefore

\[ \Phi(\bar{x}_i) \cdot \Phi(\bar{x}_j) = \sum_{t=1}^{n} \lambda_t v_t^i v_t^j = (V \Lambda V')_{ij} = K_{ij} = K(\bar{x}_i, \bar{x}_j) \]

which implies that \( K \) is a kernel function
Mercer’s Theorem
(necessary conditions)

- Suppose we have negative eigenvalues $\lambda_s$ and eigenvectors $\vec{v}_s$ the following point

$$\vec{z} = \sum_{i=1}^{n} v_{si} \Phi(\vec{x}_i) = \sum_{i=1}^{n} v_{si} \left( \sqrt{\lambda_t} v_{ti} \right)_t = \sqrt{\Lambda} V' \vec{v}_s$$

has the following norm:

$$\left\| \vec{z} \right\|^2 = \vec{z} \cdot \vec{z} = \sqrt{\Lambda} V' \vec{v}_s \sqrt{\Lambda} V' \vec{v}_s = \vec{v}_s' V \sqrt{\Lambda} \sqrt{\Lambda} V' \vec{v}_s =$$

$$\vec{v}_s' K \vec{v}_s = \vec{v}_s' \lambda_s \vec{v}_s = \lambda_s \left\| \vec{v}_s \right\|^2 < 0$$

this contradicts the geometry of the space.
Is it a valid kernel?

- It may not be a kernel so we can use $M' \cdot M$

**Proposition B.14** Let $A$ be a symmetric matrix. Then $A$ is positive (semi-) definite iff for any vector $\vec{x} \neq 0$

$$\vec{x}' A \vec{x} > 0 \quad (\geq 0).$$

From the previous proposition it follows that: If we find a decomposition $A$ in $M' M$, then $A$ is semi-definite positive matrix as

$$\vec{x}' A \vec{x} = \vec{x}' M' M \vec{x} = (M \vec{x})'(M \vec{x}) = M \vec{x} \cdot M \vec{x} = ||M \vec{x}||^2 \geq 0.$$
Valid Kernel operations

- $k(x,z) = k_1(x,z) + k_2(x,z)$
- $k(x,z) = k_1(x,z) * k_2(x,z)$
- $k(x,z) = \alpha k_1(x,z)$
- $k(x,z) = f(x)f(z)$
- $k(x,z) = k_1(\phi(x), \phi(z))$
- $k(x,z) = x'Bz$
Basic Kernels for unstructured data

- Linear Kernel
- Polynomial Kernel
- Lexical kernel
- String Kernel
Linear Kernel

- In Text Categorization documents are word vectors

\[ \Phi(d_x) = \tilde{x} = (0,..,1,..,0,..,0,..,1,..,0,..,0,..,1,..,0,..,1,..,0,..,1) \]

buy acquisition stocks sell market

\[ \Phi(d_z) = \tilde{z} = (0,..,1,..,0,..,1,..,0,..,0,..,0,..,1,..,0,..,1,..,0) \]

buy company stocks sell

- The dot product \( \tilde{x} \cdot \tilde{z} \) counts the number of features in common

- This provides a sort of similarity
Feature Conjunction (polynomial Kernel)

- The initial vectors are mapped in a higher space
  \[ \Phi(<x_1, x_2>) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1) \]

- More expressive, as \((x_1x_2)\) encodes
  \textbf{Stock+Market vs. Downtown+Market} features

- We can smartly compute the scalar product as
  \[
  \Phi(\tilde{x}) \cdot \Phi(\tilde{z}) = \\
  = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2, \sqrt{2}z_1, \sqrt{2}z_2, 1) = \\
  = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1 = \\
  = (x_1z_1 + x_2z_2 + 1)^2 = (\tilde{x} \cdot \tilde{z} + 1)^2 = K_{\text{Poly}}(\tilde{x}, \tilde{z})
  \]
Document Similarity

Doc 1

industry

telephone

market

Doc 2

company

product
Lexical Semantic Kernel [CoNLL 2005]

The document similarity is the SK function:

$$SK(d_1, d_2) = \sum_{w_1 \in d_1, w_2 \in d_2} s(w_1, w_2)$$

where $s$ is any similarity function between words, e.g. WordNet [Basili et al., 2005] similarity or LSA [Cristianini et al., 2002]

Good results when training data is small
Using character sequences

\[
\phi("bank") = \tilde{x} = (0,\ldots,1,\ldots,0,\ldots,1,\ldots,0,\ldots,1,\ldots,0,\ldots,1,\ldots,0,\ldots,1,\ldots,0)
\]

\[
\begin{array}{cccccccc}
\text{bank} & \text{ank} & \text{bnk} & \text{bk} & \text{b} \\
\end{array}
\]

\[
\phi("rank") = \tilde{z} = (1,\ldots,0,\ldots,0,\ldots,1,\ldots,0,\ldots,1,\ldots,0,\ldots,1,\ldots,0,\ldots,1)
\]

\[
\begin{array}{cccccccc}
\text{rank} & \text{ank} & \text{rnk} & \text{rk} & \text{r} \\
\end{array}
\]

\[\tilde{x} \cdot \tilde{z} \text{ counts the number of common substrings}\]

\[\tilde{x} \cdot \tilde{z} = \phi("bank") \cdot \phi("rank") = k("bank","rank")\]
String Kernel

- Given two strings, the number of matches between their substrings is evaluated.
  - E.g. Bank and Rank
    - B, a, n, k, Ba, Ban, Bank, Bk, an, ank, nk,..
    - R, a, n, k, Ra, Ran, Rank, Rk, an, ank, nk,..

- String kernel over sentences and texts
- Huge space but there are efficient algorithms.
Formal Definition

\[ s = s_1, \ldots, s_{|s|} \]

\[ \vec{I} = (i_1, \ldots, i_{|u|}) \quad u = s[\vec{I}] \]

\[ \phi_u(s) = \sum_{\vec{I}: u = s[\vec{I}]} \lambda^{l(\vec{I})}, \text{ where } l(\vec{I}) = i_{|u|} - i_1 + 1 \]

\[ K(s, t) = \sum_{u \in \Sigma^*} \phi_u(s) \cdot \phi_u(t) = \sum_{u \in \Sigma^*} \sum_{\vec{I}: u = s[\vec{I}]} \sum_{\vec{J}: u = t[\vec{J}]} \lambda^{l(\vec{I})} \lambda^{l(\vec{J})} = \]

\[ = \sum_{u \in \Sigma^*} \sum_{\vec{I}: u = s[\vec{I}]} \sum_{\vec{J}: u = t[\vec{J}]} \lambda^{l(\vec{I}) + l(\vec{J})}, \text{ where } \Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \]
Kernel between Bank and Rank

B, a, n, k, Ba, Ban, Bank, an, ank, nk, Bn, Bnk, Bk and ak are the substrings of Bank.

R, a, n, k, Ra, Ran, Rank, an, ank, nk, Rn, Rnk, Rk and ak are the substrings of Rank.
An example of string kernel computation

- $\phi_a(Bank) = \phi_a(Rank) = \lambda^{(i_1-i_1+1)} = \lambda^{(2-2+1)} = \lambda,$

- $\phi_n(Bank) = \phi_n(Rank) = \lambda^{(i_1-i_1+1)} = \lambda^{(3-3+1)} = \lambda,$

- $\phi_k(Bank) = \phi_k(Rank) = \lambda^{(i_1-i_1+1)} = \lambda^{(4-4+1)} = \lambda,$

- $\phi_{an}(Bank) = \phi_{an}(Rank) = \lambda^{(i_2-i_1+1)} = \lambda^{(3-2+1)} = \lambda^2,$

- $\phi_{ank}(Bank) = \phi_{ank}(Rank) = \lambda^{(i_3-i_1+1)} = \lambda^{(4-2+1)} = \lambda^3,$

- $\phi_{nk}(Bank) = \phi_{nk}(Rank) = \lambda^{(i_2-i_1+1)} = \lambda^{(4-3+1)} = \lambda^2$

- $\phi_{ak}(Bank) = \phi_{ak}(Rank) = \lambda^{(i_2-i_1+1)} = \lambda^{(4-2+1)} = \lambda^3$

$K(Bank, Rank) = (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3) \cdot (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3) = 3\lambda^2 + 2\lambda^4 + 2\lambda^6$
Efficient Evaluation

- Dynamic Programming technique
- Evaluate the spectrum string kernels
  - Substrings of size $p$
- Sum the contribution of the different spectra
Efficient Evaluation

Given two sequences $s_1a$ and $s_2b$, we define:

$$D_p(|s_1|, |s_2|) = \sum_{i=1}^{s_1} \sum_{r=1}^{s_2} \lambda^{s_1-i} |s_2|^{-r} \times SK_{p-1}(s_1[1 : i], s_2[1 : r]),$$

$s_1[1 : i]$ and $s_2[1 : r]$ are their subsequences from 1 to $i$ and 1 to $r$.

$$SK_p(s_1a, s_2b) = \begin{cases} \lambda^2 \times D_p(|s_1|, |s_2|) & \text{if } a = b; \\ 0 & \text{otherwise.} \end{cases}$$

$D_p$ satisfies the recursive relation:

$$D_p(k, l) = SK_{p-1}(s_1[1 : k], s_2[1 : l]) + \lambda D_p(k, l - 1) +$$

$$+ \lambda D_p(k - 1, l) - \lambda^2 D_p(k - 1, l - 1)$$
An example: \( SK(\text{“Gatta”,”Cata”}) \)

- First, evaluate the SK with size \( p=1 \), i.e. \( “a” \), \( “a”,“t”,“t”,“a”,“a” \)
- Store this in the table

\[
SK_{p=1} = \begin{array}{cccccc}
& g & a & t & t & a \\
\hline
c & 0 & 0 & 0 & 0 & 0 \\
a & 0 & \lambda^2 & 0 & 0 & \lambda^2 \\
t & 0 & 0 & \lambda^2 & \lambda^2 & 0 \\
a & 0 & \lambda^2 & 0 & 0 & \lambda^2 \\
\end{array}
\]
Evaluating DP2

- Evaluate the weight of the string of size $p$ in case a character will be matched
- This is done by multiplying the double summation by the number of substrings of size $p-1$

$$D_p(|s_1|, |s_2|) = \sum_{i=1}^{\lfloor s_1 \rfloor} \sum_{r=1}^{\lfloor s_2 \rfloor} \lambda^{s_1-i+s_2-r} \times SK_{p-1}(s_1[1:i], s_2[1:r])$$
Evaluating the Predictive DP on strings of size 2 (second row)

- Let’s consider substrings of size 2 and suppose that:
  - we have matched the first “a”
  - we will match the next character that we will add to the two strings

- We compute the weights of matches above at different string positions with some not-yet known character “?”

- If the match occurs immediately after “a” the weight will be $\lambda^{1+1} \times \lambda^{1+1} = \lambda^4$ and we store just $\lambda^2$ in the DP entry in [“a”,”a”]

<table>
<thead>
<tr>
<th>DP₂</th>
<th>g</th>
<th>a</th>
<th>t</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>$\lambda^2$</td>
<td>$\lambda^3$</td>
<td>$\lambda^4$</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>$\lambda^3$</td>
<td>$\lambda^4 + \lambda^2$</td>
<td>$\lambda^5 + \lambda^3 + \lambda^2$</td>
</tr>
</tbody>
</table>
Evaluating the DP wrt different positions (second row)

- If the match for “gatta” occurs after “t” the weight will be $\lambda^{1+2}$ ($x \lambda^2 = \lambda^5$) since the substring for it will be with “a□?”
- We write such prediction in the entry [“a”,”t”]
- Same rationale for a match after the second “t”: we have the substring “a□□?” (matching with “a?” from “catta”) for a weight of $\lambda^{3+1}$ ($x \lambda^2$)

### Table

<table>
<thead>
<tr>
<th>$DP_2$</th>
<th>g</th>
<th>a</th>
<th>t</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>$\lambda^2$</td>
<td>$\lambda^3$</td>
<td>$\lambda^4$</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>$\lambda^3$</td>
<td>$\lambda^4 + \lambda^2$</td>
<td>$\lambda^5 + \lambda^3 + \lambda^2$</td>
</tr>
</tbody>
</table>
Evaluating the DP wrt different positions (third row)

- If the match occurs after “t” of “cata”, the weight will be $\lambda^{2+1}$ ($\times \lambda^2 = \lambda^5$) since it will be with the string “a□?” with a weight of $\lambda^3$

- If the match occurs after “t” of both “gatta” and “cata”, there are two ways to compose substring of size two: “a□?” with weight $\lambda^4$ or “t?” with weight $\lambda^2$ $\Rightarrow$ the total is $\lambda^2+\lambda^4$

<table>
<thead>
<tr>
<th>DP_2</th>
<th>g</th>
<th>a</th>
<th>t</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>$\lambda^2$</td>
<td>$\lambda^3$</td>
<td>$\lambda^4$</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>$\lambda^3$</td>
<td>$\lambda^4+\lambda^2$</td>
<td>$\lambda^5+\lambda^3+\lambda^2$</td>
</tr>
</tbody>
</table>
The final case is a match after the last “t” of both “cat” and “gatta”

There are three possible substrings of “gatta”:
- “a☐☐?”, “t☐?” “t?” for “gatta” with weight $\lambda^3$, $\lambda^2$ or $\lambda$, respectively.

There are two possible substrings of “cata”
- “a☐?” “t?” with weight $\lambda^2$ and $\lambda$
- Their match gives weights: $\lambda^5$, $\lambda^3$, $\lambda^2 \Rightarrow$ by summing: $\lambda^5 + \lambda^3 + \lambda^2$

<table>
<thead>
<tr>
<th>DP$_2$</th>
<th>g</th>
<th>a</th>
<th>t</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>$\lambda^2$</td>
<td>$\lambda^3$</td>
<td>$\lambda^4$</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>$\lambda^3$</td>
<td>$\lambda^4 + \lambda^2$</td>
<td>$\lambda^5 + \lambda^3 + \lambda^2$</td>
</tr>
</tbody>
</table>
Evaluating SK of size 2 using DP2

\[ SK_p(s_1a, s_2b) = \begin{cases} \lambda^2 \times D_p(|s_1|, |s_2|) & \text{if } a = b; \\ 0 & \text{otherwise.} \end{cases} \]

The number (weight) of substrings of size 2 between “gat” and “cat” is \( \lambda^4 = \lambda^2 \) ([“a”,”a”] entry of DP) \( \times \lambda^2 \) (cost of one character), where \( a = “t” \) and \( b = “t” \).

Between “gattta” and “catata” is \( \lambda^7 + \lambda^5 + \lambda^4 \), i.e the matches of “a☐☐a”, “t☐a”, “ta” with “a☐a” and “ta”.

<table>
<thead>
<tr>
<th>DP2</th>
<th>g</th>
<th>a</th>
<th>t</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>( \lambda^2 )</td>
<td>( \lambda^3 )</td>
<td>( \lambda^4 )</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>( \lambda^3 )</td>
<td>( \lambda^4 + \lambda^2 )</td>
<td>( \lambda^5 + \lambda^3 + \lambda^2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SK ( p=2 )</th>
<th>g</th>
<th>a</th>
<th>t</th>
<th>t</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>0</td>
<td>( \lambda^4 )</td>
<td>( \lambda^5 )</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \lambda^7 + \lambda^5 + \lambda^4 )</td>
</tr>
</tbody>
</table>
String Kernels for OCR
Pixel Representation

Figure 6: Resampling of an image from $16 \times 16$ to $8 \times 8$ format
Sequence of bits

$\begin{align*}
\text{L1} & \quad 00011100 \\
. & \quad 00111100 \\
. & \quad 00101100 \\
. & \quad 00001100 \\
\text{L8} & \quad 00001100 \\
\end{align*}$

$SK(im_a, im_b) = \sum_{i=1..8} SK(L^i_a, L^i_b)$
## Results

- Using columns+rows+diagonals

<table>
<thead>
<tr>
<th>Digit</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>97.78</td>
<td>97.78</td>
<td>97.78</td>
</tr>
<tr>
<td>1</td>
<td>95.45</td>
<td>93.33</td>
<td>94.38</td>
</tr>
<tr>
<td>2</td>
<td>93.62</td>
<td>97.78</td>
<td>95.65</td>
</tr>
<tr>
<td>3</td>
<td>93.33</td>
<td>93.33</td>
<td>93.33</td>
</tr>
<tr>
<td>4</td>
<td>97.83</td>
<td>100.00</td>
<td>98.90</td>
</tr>
<tr>
<td>5</td>
<td>97.67</td>
<td>93.33</td>
<td>95.45</td>
</tr>
<tr>
<td>6</td>
<td>100.00</td>
<td>97.78</td>
<td>98.88</td>
</tr>
<tr>
<td>7</td>
<td>91.84</td>
<td>100.00</td>
<td>95.74</td>
</tr>
<tr>
<td>8</td>
<td>93.18</td>
<td>91.11</td>
<td>92.13</td>
</tr>
<tr>
<td>9</td>
<td>93.02</td>
<td>88.89</td>
<td>90.91</td>
</tr>
<tr>
<td>Multiclass accuracy</td>
<td>95.33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tree kernels

- Subtree, Subset Tree, Partial Tree kernels
- Efficient computation
Main Idea of Tree Kernels

$\phi(T_1) = [2, 1, 1, 1, 1, 0, 0]$

$\phi(T_2) = [0, 0, 0, 0, 1, 1, 1]$

$TK(T_1, T_2) = \langle \phi(T_1), \phi(T_2) \rangle = 1$
Example of a syntactic parse tree

“John delivers a talk in Rome”
The Syntactic Tree Kernel (STK)
[Collins and Duffy, 2002]
The overall fragment set
The overall fragment set

Children are not divided
Explicit kernel space

\[ \phi(T_x) = \bar{x} = (0, ..., 1, ..., 0, ..., 1, ..., 0, ..., 1, ..., 0, ..., 1, ..., 0) \]

\[ \phi(T_z) = \bar{z} = (1, ..., 0, ..., 0, ..., 1, ..., 0, ..., 1, ..., 0, ..., 0, ..., 0) \]

\[ \bar{x} \cdot \bar{z} \] counts the number of common substructures
Efficient evaluation of the scalar product

\[ \tilde{x} \cdot \tilde{z} = \phi(T_x) \cdot \phi(T_z) = K(T_x, T_z) = \]
\[ = \sum_{n_x \in T_x} \sum_{n_z \in T_z} \Delta(n_x, n_z) \]
Efficient evaluation of the scalar product

\[ \vec{x} \cdot \vec{z} = \phi(T_x) \cdot \phi(T_z) = K(T_x, T_z) = \]

\[ \sum_{n_x \in T_x} \sum_{n_z \in T_z} \Delta(n_x, n_z) \]

- [Collins and Duffy, ACL 2002] evaluate \( \Delta \) in \( O(n^2) \):

\[ \Delta(n_x, n_z) = 0, \quad \text{if the productions are different} \]
\[ \Delta(n_x, n_z) = 1, \quad \text{if pre-terminals} \]
\[ \Delta(n_x, n_z) = \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j))) \]
Other Adjustments

- Decay factor

\[ \Delta(n_x, n_z) = \lambda, \quad \text{if pre-terminals} \]
\[ \Delta(n_x, n_z) = \lambda \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j))) \]

- Normalization

\[ K'(T_x, T_z) = \frac{K(T_x, T_z)}{\sqrt{K(T_x, T_x) \times K(T_z, T_z)}} \]
SubTree (ST) Kernel [Vishwanathan and Smola, 2002]
Evaluation

- Given the equation for STK

\[ \Delta(n_x, n_z) = 0, \text{ if the productions are different else} \]
\[ \Delta(n_x, n_z) = 1, \text{ if pre-terminals else} \]

\[ \Delta(n_x, n_z) = \prod_{j=1}^{nc(n_x)} (1 + \Delta(ch(n_x, j), ch(n_z, j))) \]
Given the equation for STK

\[ \Delta(n_x, n_z) = \begin{cases} 0, & \text{if the productions are different} \\ 1, & \text{if pre-terminals} \end{cases} \]

\[ \Delta(n_x, n_z) = \prod_{j=1}^{nc(n_x)} (\Delta(ch(n_x, j), ch(n_z, j))) \]
Fast Evaluation of STK [Moschitti, EACL 2006]

\[ K(T_x, T_z) = \sum_{\langle n_x, n_z \rangle \in NP} \Delta(n_x, n_z) \]

\[ NP = \left\{ \langle n_x, n_z \rangle \in T_x \times T_z : \Delta(n_x, n_z) \neq 0 \right\} = \]

\[ = \left\{ \langle n_x, n_z \rangle \in T_x \times T_z : P(n_x) = P(n_z) \right\}, \]

where \( P(n_x) \) and \( P(n_z) \) are the production rules used at nodes \( n_x \) and \( n_z \).
function Evaluate_Pair_Set(Tree \( T_1, T_2 \)) returns NODE_PAIR_SET;
LIST \( L_1, L_2 \);
NODE_PAIR_SET \( N_p \);
begin
\( L_1 = T_1.\text{ordered\_list} \);
\( L_2 = T_2.\text{ordered\_list} \); /*the lists were sorted at loading time*/
n_1 = extract(\( L_1 \)); /*get the head element and*/
n_2 = extract(\( L_2 \)); /*remove it from the list*/
while (\( n_1 \) and \( n_2 \) are not NULL)
    if (production_of(\( n_1 \)) > production_of(\( n_2 \)))
        then \( n_2 = \text{extract}(L_2) \);
    else if (production_of(\( n_1 \)) < production_of(\( n_2 \)))
        then \( n_1 = \text{extract}(L_1) \);
    else
        while (production_of(\( n_1 \)) == production_of(\( n_2 \)))
            while (production_of(\( n_1 \)) == production_of(\( n_2 \)))
                add(\( \langle n_1, n_2 \rangle, N_p \));
                \( n_2 = \text{get\_next\_elem}(L_2) \); /*get the head element and move the pointer to the next element*/
        end
        \( n_1 = \text{extract}(L_1) \);
        reset(\( L_2 \)); /*set the pointer at the first element*/
    end
    \( n_1 = \text{extract}(L_1) \);
    \( \text{reset}(L_2) \); /*set the pointer at the first element*/
end
return \( N_p \);
end
Running Time Complexity

- We order the production rules used in $T_x$ and $T_z$, at loading time
- At learning time we may evaluate NP in $|T_x| + |T_z|$ running time
- If $T_x$ and $T_z$ are generated by only one production rule $\Rightarrow O(|T_x| \times |T_z|)$...
Running Time Complexity

- We order the production rules used in $T_x$ and $T_z$, at loading time
- At learning time we may evaluate NP in $|T_x| + |T_z|$ running time
- If $T_x$ and $T_z$ are generated by only one production rule $\Rightarrow O(|T_x|\times|T_z|)$ … Very Unlikely!!!!
Labeled Ordered Tree Kernel

- STK satisfies the constraint “remove 0 or all children at a time”.
- If we relax such constraint we get more general substructures [Kashima and Koyanagi, 2002]
Weighting Problems

- Both matched pairs give the same contribution.
- Gap based weighting is needed.
- A novel efficient evaluation has to be defined.
Partial Trees, [Moschitti, ECML 2006]

- STK + String Kernel with weighted gaps on Nodes’ children
Partial Tree Kernel

- if the node labels of $n_1$ and $n_2$ are different then 
  $\Delta(n_1, n_2) = 0$;
- else
  $\Delta(n_1, n_2) = 1 + \sum_{\bar{J}_1, \bar{J}_2, l(\bar{J}_1) = l(\bar{J}_2)} \prod_{i=1}^{\ell(\bar{J}_1)} \Delta(c_{n_1}[\bar{J}_{1i}], c_{n_2}[\bar{J}_{2i}])$

By adding two decay factors we obtain:

$$\mu\left(\lambda^2 + \sum_{\bar{J}_1, \bar{J}_2, l(\bar{J}_1) = l(\bar{J}_2)} \lambda^{d(\bar{J}_1) + d(\bar{J}_2)} \prod_{i=1}^{\ell(\bar{J}_1)} \Delta(c_{n_1}[\bar{J}_{1i}], c_{n_2}[\bar{J}_{2i}])\right)$$
Efficient Evaluation (1)

- In [Taylor and Cristianini, 2004 book], sequence kernels with weighted gaps are factorized with respect to different subsequence sizes.
- We treat children as sequences and apply the same theory

\[
\Delta(n_1, n_2) = \mu(\lambda^2 + \sum_{p=1}^{l} \Delta_p(c_{n_1}, c_{n_2})),
\]

Given the two child sequences \(s_1 a = c_{n_1}\) and \(s_2 b = c_{n_2}\) (\(a\) and \(b\) are the last children), \(\Delta_p(s_1 a, s_2 b) = \sum_{i=1}^{\frac{|s_1|}{|s_2|}} \sum_{r=1}^{\frac{|s_1|}{|s_2|}} \lambda^{|s_1| - i + |s_2| - r} \times \Delta_{p-1}(s_1[1:i], s_2[1:r])\)
Efficient Evaluation (2)

\[ \Delta_p(s_1a, s_2b) = \begin{cases} 
\Delta(a, b)D_p(|s_1|, |s_2|) & \text{if } a = b; \\
0 & \text{otherwise.}
\end{cases} \]

Note that \( D_p \) satisfies the recursive relation:

\[ D_p(k, l) = \Delta_{p-1}(s_1[1:k], s_2[1:l]) + \lambda D_p(k, l - 1) + \lambda D_p(k - 1, l) + \lambda^2 D_p(k - 1, l - 1). \]

- The complexity of finding the subsequences is \( O(p|s_1||s_2|) \)
- Therefore the overall complexity is \( O(p\rho^2|N_{T_1}||N_{T_2}|) \)
  where \( \rho \) is the maximum branching factor (\( p = \rho \))
Running Time of Tree Kernel Functions

- FTK-SST
- QTK-SST
- FTK-PT

μseconds vs. Number of Tree Nodes
SVM-light-TK Software

- Encodes ST, STK and combination kernels in SVM-light [Joachims, 1999]
- Available at http://dit.unitn.it/~moschitt/
- Tree forests, vector sets
- The new SVM-Light-TK toolkit will be released asap (email me to have the current version)
Practical Example on Question Classification

- **Definition**: What does HTML stand for?
- **Description**: What's the final line in the Edgar Allan Poe poem "The Raven"?
- **Entity**: What foods can cause allergic reaction in people?
- **Human**: Who won the Nobel Peace Prize in 1992?
- **Location**: Where is the Statue of Liberty?
- **Manner**: How did Bob Marley die?
- **Numeric**: When was Martin Luther King Jr. born?
- **Organization**: What company makes Bentley cars?
Question Classifier based on Tree Kernels

- Question dataset (http://l2r.cs.uiuc.edu/~cogcomp/Data/QA/QC/)
  [Lin and Roth, 2005]
  - Distributed on 6 categories: Abbreviations, Descriptions, Entity, Human, Location, and Numeric.

- Fixed split 5500 training and 500 test questions

- Cross-validation (10-folds)

- Using the whole question parse trees
  - Constituent parsing
  - Example

  “What is an offer of direct stock purchase plan?”
What is an offer of direct stock purchase plan?
Data Format

■ “What does HTML stand for?”

■ 1 |BT| (SBARQ (WHNP (WP What))(SQ (AUX does)(NP (NNP S.O.S.))(VP (VB stand)(PP (IN for)))))(. ?))|ET|
Trees + Feature Vectors

“What does HTML stand for?”

1  |BT| (SBARQ (WHNP (WP What))(SQ (AUX does)(NP (NNP S.O.S.))(VP (VB stand)(PP (IN for)))))(. ?))|ET|
Basic Commands

- Training and classification
  - ./svm_learn -t 5 train.dat model
  - ./svm_classify test.dat model
Conclusions

- Dealing with noisy and errors of NLP modules require robust approaches

- SVMs are robust to noise and Kernel methods allows for:
  - Syntactic information via STK
  - Shallow Semantic Information via PTK
  - Word/POS sequences via String Kernels

- When the IR task is complex, syntax and semantics are essential

  ⇒ Great improvement in Q/A classification

- SVM-Light-TK: an efficient tool to use them
SVM-light-TK Software

- Encodes ST, SST and combination kernels in SVM-light [Joachims, 1999]
- Available at http://dit.unitn.it/~moschitt/
- Tree forests, vector sets
- New extensions: the PT kernel will be released asap
“What does Html stand for?”

1. **BT** (SBARQ (WHNP (WP What))(SQ (AUX does)(NP (NNP S.O.S.))(VP (VB stand)(PP (IN for))))(. ?))

   |BT| (BOW (What *)(does *)(S.O.S. *)(stand *)(for *)(? *))

   |BT| (BOP (WP *)(AUX *)(NNP *)(VB *)(IN *)(. *))

   |BT| (PAS (ARG0 (R-A1 (What *))(ARG1 (A1 (S.O.S. NNP)))(ARG2 (rel stand))))


Basic Commands

- Training and classification
  - ./svm_learn -t 5 -C T train.dat model
  - ./svm_classify test.dat model

- Learning with a vector sequence
  - ./svm_learn -t 5 -C V train.dat model

- Learning with the sum of vector and kernel sequences
  - ./svm_learn -t 5 -C + train.dat model
Custom Kernel

- Kernel.h

```c
double custom_kernel(KERNEL_PARM *kernel_parm, DOC *a, DOC *b);

if(a->num_of_trees && b->num_of_trees && a->forest_vec[i]!=NULL && b->forest_vec[i]!=NULL){ // Test if one the i-th tree of instance a and b is an empty tree
```
Custom Kernel: tree-kernel

- \( k_1 = \text{summation of tree kernels} \)
  
  \[
  \text{tree_kernel(kernel_parm, a, b, i, i)}/
  \]
  
  Evaluate tree kernel between the two \( i \)-th trees.

  \[
  \sqrt{\text{tree_kernel(kernel_parm, a, a, i, i) \ast \text{tree_kernel(kernel_parm, b, b, i, i)}}};
  \]
  
  Normalize respect to both \( i \)-th trees.
Custom Kernel: Polynomial kernel

- if(a->num_of_vectors && b->num_of_vectors && a->vectors[i]!=NULL && b->vectors[i]!=NULL){ Check if the i-th vectors are empty.
- k2=  // summation of vectors
   basic_kernel(kernel_parm, a, b, i, i)/
Compute standard kernel (selected according to the "second_kernel" parameter).
Custom Kernel: Polynomial kernel

- $\sqrt{\text{basic}_\text{kernel}(\text{kernel}\_\text{parm}, a, a, i, i) \times \text{basic}_\text{kernel}(\text{kernel}\_\text{parm}, b, b, i, i)}$; //normalize vectors
- $\text{return } k1+k2$;
Conclusions

- Kernel methods and SVMs are useful tools to design language applications
- Kernel design still require some level of expertise
- Engineering approaches to tree kernels
  - Basic Combinations
  - Canonical Mappings, e.g.
    - Node Marking
  - Merging of kernels in more complex kernels
- State-of-the-art in SRL and QC
- An efficient tool to use them
Thank you
References


Roberto Basili
Alessandro Moschitti

Automatic Text Categorization

From Information Retrieval to Support Vector Learning
References


References

References

- Alessandro Moschitti and Roberto Basili, 

- Ana-Maria Giuglea and Alessandro Moschitti, 

- Roberto Basili, Marco Cammisa and Alessandro Moschitti, 
  *Effective use of wordnet semantics via kernel-based learning*. In Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL 2005), Ann Arbor(MI), USA, 2005
References


References

References

- AN INTRODUCTION TO SUPPORT VECTOR MACHINES (and other kernel-based learning methods)
  N. Cristianini and J. Shawe-Taylor Cambridge University Press


function Evaluate_Pair_Set(Tree $T_1$, $T_2$) returns NODE_PAIR_SET;
LIST $L_1$, $L_2$;
NODE_PAIR_SET $N_p$;
begin

$L_1 = T_1$.ordered_list;
$L_2 = T_2$.ordered_list; /*the lists were sorted at loading time*/
n_1 = extract($L_1$); /*get the head element and*/
n_2 = extract($L_2$); /*remove it from the list*/
while (n_1 and n_2 are not NULL)
if (production_of($n_1$) > production_of($n_2$))
  then $n_2 = extract(L_2)$;
else if (production_of($n_1$) < production_of($n_2$))
  then $n_1 = extract(L_1)$;
else
  while (production_of($n_1$) == production_of($n_2$))
    while (production_of($n_1$) == production_of($n_2$))
      add($\langle n_1, n_2 \rangle$, $N_p$);

$n_2$=get_next_elem($L_2$); /*get the head element
  and move the pointer to the next element*/
end
$n_1 = extract(L_1)$;
reset($L_2$); /*set the pointer at the first element*/
end

return $N_p$ ;
end
The Impact of SSTK in Answer Classification

![Graph showing the F1-measure for various configurations of SSTK.](image)
Mercer’s conditions (1)

**Def. B.11 Eigen Values**

Given a matrix \( A \in \mathbb{R}^{m \times n} \), an eigenvalue \( \lambda \) and an eigenvector \( \overline{x} \in \mathbb{R}^n - \{0\} \) are such that

\[
A \overline{x} = \lambda \overline{x}
\]

**Def. B.12 Symmetric Matrix**

A square matrix \( A \in \mathbb{R}^{n \times n} \) is symmetric iff \( A_{ij} = A_{ji} \) for \( i \neq j \) \( i = 1, \ldots, m \) and \( j = 1, \ldots, n \), i.e. iff \( A = A' \).

**Def. B.13 Positive (Semi-) definite Matrix**

A square matrix \( A \in \mathbb{R}^{n \times n} \) is said to be positive (semi-) definite if its eigenvalues are all positive (non-negative).
Mercer’s conditions (2)

Proposition 2.27 (Mercer’s conditions)
Let $X$ be a finite input space with $K(\bar{x}, \bar{z})$ a symmetric function on $X$. Then $K(\bar{x}, \bar{z})$ is a kernel function if and only if the matrix

$$k(\bar{x}, \bar{z}) = \phi(\bar{x}) \cdot \phi(\bar{z})$$

is positive semi-definite (has non-negative eigenvalues).

- If the Gram matrix: $G = k(\bar{x}_i, \bar{x}_j)$
  is positive semi-definite there is a mapping $\phi$ that produces the target kernel function
The lexical semantic kernel is not always a kernel

- It may not be a kernel so we can use $M' \cdot M$, where $M$ is the initial similarity matrix.

**Proposition B.14** Let $A$ be a symmetric matrix. Then $A$ is positive (semi-) definite iff for any vector $\vec{x} \neq 0$

$$\vec{x}' A \vec{x} > \lambda \vec{x} \quad (\geq 0).$$

From the previous proposition it follows that: If we find a decomposition $A$ in $M' M$, then $A$ is semi-definite positive matrix as

$$\vec{x}' A \vec{x} = \vec{x}' M' M \vec{x} = (M \vec{x})' (M \vec{x}) = M \vec{x}' \cdot M \vec{x} = \|M \vec{x}\|^2 \geq 0.$$
Efficient Evaluation (1)

- In [Taylor and Cristianini, 2004 book], sequence kernels with weighted gaps are factorized with respect to different subsequence sizes.
- We treat children as sequences and apply the same theory

\[
\Delta(n_1, n_2) = \mu(\lambda^2 + \sum_{p=1}^{ln} \Delta_p(c_{n_1}, c_{n_2}))
\]

Given the two child sequences \(s_1a = c_{n_1}\) and \(s_2b = c_{n_2}\) (\(a\) and \(b\) are the last children), \(\Delta_p(s_1a, s_2b) = \)

\[
\Delta(a, b) \times \sum_{i=1}^{\vert s_1 \vert} \sum_{r=1}^{\vert s_2 \vert} \lambda^{\vert s_1 \vert - i + \vert s_2 \vert - r} \times \Delta_{p-1}(s_1[1:i], s_2[1:r])
\]