Non-linear SVMs & libSVM

Advanced Statistical Methods in NLP
Ling 572
February 23, 2012
Roadmap

- Non-linear SVMs:
  - Motivation: Non-linear data

- The kernel trick

- Linear $\rightarrow$ Non-linear SVM models

- LibSVM:
  - `svm-train` & `svm-predict`
  - Models

- HW #7
Non-Linear SVMs

- Problem:
  - Sometimes data really isn’t linearly separable
Non-Linear SVMs

- **Problem:**
  - Sometimes data really isn’t linearly separable

- **Approach:**
  - Map data non-linearly into higher dimensional space
    - Data is separable in the higher dimensional space
Non-Linear SVMs

- **Problem:**
  - Sometimes data really isn’t linearly separable

- **Approach:**
  - Map data non-linearly into higher dimensional space
  - Data is separable in the higher dimensional space

Figure from Hearst et al ‘98
Feature Space

- Basic approach:
  - Original data is not linearly separable
Feature Space

- Basic approach:
  - Original data is not linearly separable

- Map data into 'feature space'
  - Higher dimensional dot product space
  - Mapping via non-linear map: $\Phi$
Feature Space

• Basic approach:
  • Original data is not linearly separable

• Map data into ‘feature space’
  • Higher dimensional dot product space
  • Mapping via non-linear map: $\Phi$

• Compute separating hyperplane
  • In higher dimensional space
Issues with Feature Space

- Mapping idea is simple,
- But has some practical problems
Issues with Feature Space

- Mapping idea is simple,
  - But has some practical problems

- Feature space can be very high – infinite? – dimensional
Issues with Feature Space

- Mapping idea is simple,
  - But has some practical problems

- Feature space can be very high – infinite? – dimensional

- Approach depends on computing similarity (dot product)
  - Computationally expensive
Issues with Feature Space

- Mapping idea is simple,
  - But has some practical problems

- Feature space can be very high – infinite? – dimensional

- Approach depends on computing similarity (dot product)
  - Computationally expensive

- Approach depends on mapping:
  - Also possibly intractable to compute
Solution

- “Kernel trick”:
  - Use a kernel function $K: X \times X \to \mathbb{R}$
  
  \[ K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \]
Solution

• “Kernel trick”:
  • Use a kernel function $K: X \times X \rightarrow \mathbb{R}$
    $$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$
  • Computes similarity measure on images of data points
Solution

- “Kernel trick”:
  - Use a kernel function \( K: X \times X \rightarrow \mathbb{R} \)
    \[
    K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle
    \]
  - Computes similarity measure on images of data points
    - Can often compute similarity efficiently even on high (or infinite) dimensional space
Solution

- “Kernel trick”:
  - Use a kernel function $K: X \times X \rightarrow \mathbb{R}$
    $$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$
  - Computes similarity measure on images of data points
    - Can often compute similarity efficiently even on high (or infinite) dimensional space
  - Choice of $K$ equivalent to selection of $\Phi$
Figure 20.27  (a) A two-dimensional training with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown. (b) The same data after mapping into a three-dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions.
Example (cont’d)

- Original 2-D data: \( \mathbf{x} = (x_1, x_2) \)
Example (cont’d)

- Original 2-D data: \( \mathbf{x} = (x_1, x_2) \)
- Mapping to new values in 3-D feature space \( F(x) \):
  - \( f_1 = x_1^2 \)
  - \( f_2 = x_2^2 \)
  - \( f_3 = \sqrt{2} x_1 x_2 \)
Example (cont’d)

- Original 2-D data: $\mathbf{x}=(x_1,x_2)$

- Mapping to new values in 3-D feature space $F(x)$:
  - $f_1=x_1^2$; $f_2=x_2^2$; $f_3=\sqrt{2}x_1x_2$

  $\mathbf{x}=(1,2); \mathbf{z}=(-2,3)$
Example (cont’d)

- Original 2-D data: \( \mathbf{x} = (x_1, x_2) \)

- Mapping to new values in 3-D feature space \( F(\mathbf{x}) \):
  - \( f_1 = x_1^2; \quad f_2 = x_2^2; \quad f_3 = \sqrt{2}x_1x_2 \)

\[
\mathbf{x} = (1, 2); \quad \mathbf{z} = (-2, 3) \\
\phi(\mathbf{x}) =
\]
Example (cont’d)

- Original 2-D data: \( \mathbf{x} = (x_1, x_2) \)

- Mapping to new values in 3-D feature space \( F(x) \):
  - \( f_1 = x_1^2; \quad f_2 = x_2^2; \quad f_3 = \sqrt{2} x_1 x_2 \)

\[
\begin{align*}
\tilde{x} &= (1, 2); \quad \tilde{z} = (-2, 3) \\
\phi(\tilde{x}) &= (1, 4, 2\sqrt{2}); \quad \phi(\tilde{z}) =
\end{align*}
\]
Example (cont’d)

- Original 2-D data: \( \mathbf{x} = (x_1, x_2) \)

- Mapping to new values in 3-D feature space \( \mathbf{F}(\mathbf{x}) \):
  - \( f_1 = x_1^2; \quad f_2 = x_2^2; \quad f_3 = \sqrt{2}x_1x_2 \)

\[
\mathbf{x} = (1, 2); \quad \mathbf{z} = (-2, 3)
\]

\[
\phi(\mathbf{x}) = (1, 4, 2\sqrt{2}); \quad \phi(\mathbf{z}) = (4, 9, -6\sqrt{2})
\]

\[
K(\mathbf{x}, \mathbf{z}) = <\phi(\mathbf{x}), \phi(\mathbf{z})>
\]
Example (cont’d)

- Original 2-D data: \( \mathbf{x} = (x_1, x_2) \)
- Mapping to new values in 3-D feature space \( F(x) \):
  - \( f_1 = x_1^2; \ f_2 = x_2^2; \ f_3 = \sqrt{2}x_1x_2 \)

\[
\bar{x} = (1, 2); \ \bar{z} = (-2, 3)
\]

\[
\phi(\bar{x}) = (1, 4, 2\sqrt{2}); \ \phi(\bar{z}) = (4, 9, -6\sqrt{2})
\]

\[
K(\bar{x}, \bar{z}) = \langle \phi(\bar{x}), \phi(\bar{z}) \rangle
\]

\[
= 1 \cdot 4 + 4 \cdot 9 + 2\sqrt{2} \cdot -6\sqrt{2}
\]

\[
= 16
\]
Example (cont’d)

- More generally

\[ \bar{x} = (x_1, x_2); \bar{z} = (z_1, z_2) \]

\[ \phi(\bar{x}) = \]
Example (cont’d)

- More generally

\[ \bar{x} = (x_1, x_2); \bar{z} = (z_1, z_2) \]

\[ \phi(\bar{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2); \phi(\bar{z}) = \]

26
Example (cont’d)

- More generally

\[ \vec{x} = (x_1, x_2); \vec{z} = (z_1, z_2) \]

\[ \phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2); \phi(\vec{z}) = (z_1^2, z_2^2, \sqrt{2}z_1z_2) \]

\[ < \phi(\vec{x}), \phi(\vec{z}) > = \]
Example (cont’d)

- More generally

\[ \bar{x} = (x_1, x_2); \bar{z} = (z_1, z_2) \]

\[
\phi(\bar{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2); \phi(\bar{z}) = (z_1^2, z_2^2, \sqrt{2}z_1z_2)
\]

\[
< \phi(\bar{x}), \phi(\bar{z}) > = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2
\]
Example (cont’d)

- More generally

\[ \vec{x} = (x_1, x_2); \quad \vec{z} = (z_1, z_2) \]

\[ \phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2); \quad \phi(\vec{z}) = (z_1^2, z_2^2, \sqrt{2}z_1z_2) \]

\[ < \phi(\vec{x}), \phi(\vec{z}) > = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \]

\[ = (x_1z_1 + x_2z_2)^2 \]
Example (cont’d)

- More generally

\[ \vec{x} = (x_1, x_2); \vec{z} = (z_1, z_2) \]

\[ \phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2); \phi(\vec{z}) = (z_1^2, z_2^2, \sqrt{2}z_1z_2) \]

\[ < \phi(\vec{x}), \phi(\vec{z}) > = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \]

\[ = (x_1z_1 + x_2z_2)^2 \]

\[ = < x, z >^2 \]
Kernel Trick: Summary

- Avoids explicit mapping to high-dimensional space
- Avoids explicit computation of inner product in feature space
- Avoids explicit computation of mapping function
  - Or even feature vector
- Replace all inner products in SVM train/test with K
Non-Linear SVM Training

- Linear version:
  - Maximize
    \[ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j < \bar{x}_i, \bar{x}_j > \]
  - subject to
    \[ \alpha_i \geq 0; \sum_i \alpha_i y_i = 0 \]
Non-Linear SVM Training

- Linear version:
  - Maximize
    \[ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j < \tilde{x}_i, \tilde{x}_j > \]
  - subject to
    \[ \alpha_i \geq 0; \sum_i \alpha_i y_i = 0 \]

- Non-linear version:
  - Maximize
    \[ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\tilde{x}_i, \tilde{x}_j) \]
Decoding

- Linear SVM:

\[ f(x) = \sum_i \alpha_i y_i \langle \bar{x}_i, \bar{x} \rangle + b \]
Decoding

- Linear SVM:

\[ f(x) = \sum_i \alpha_i y_i < \bar{x}_i, \bar{x} > + b \]

- Non-linear SVM:

\[ f(x) = \sum_i \alpha_i y_i K(\bar{x}_i, \bar{x}) + b \]
Common Kernel Functions

- Implemented in most packages

- Linear: \( K(\bar{x}, \bar{z}) = \langle \bar{x}, \bar{z} \rangle \)

- Polynomial: \( K(\bar{x}, \bar{z}) = (\gamma < \bar{x}, \bar{z} > + c)^d \)

- Radial Basis Function (RBF): \( K(\bar{x}, \bar{z}) = e^{-\gamma \|\bar{x} - \bar{z}\|^2} \)

- Sigmoid \( K(\bar{x}, \bar{z}) = \tanh(\gamma < \bar{x}, \bar{z} > + c) \)

\[
\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]
\[ \vec{x} = (x_1, \ldots, x_n) \]
\[ \vec{z} = (z_1, \ldots, z_n) \]
\[ \vec{x} - \vec{z} \]
\[ = (x_1 - z_1, \ldots, x_n - z_n) \]
\[ \|\vec{x} - \vec{z}\| \]
\[ = \sqrt{(x_1 - z_1)^2 + \ldots + (x_n - z_n)^2} \]
Kernels

- Many conceivable kernels:
  - Function is a kernel if obeys Mercer’s theorem:
    - Is symmetric, continuous, and matrix is positive definite
Kernels

• Many conceivable kernels:
  • Function is a kernel if obeys Mercer’s theorem:
    • Is symmetric, continuous, and matrix is positive definite

• Selection of kernel can have huge impact
  • Dramatic differences in accuracy
Kernels

• Many conceivable kernels:
  • Function is a kernel if obeys Mercer’s theorem:
    • Is symmetric, continuous, and matrix is positive definite

• Selection of kernel can have huge impact
  • Dramatic differences in accuracy

• Knowledge about ‘shape’ of data can help select
  • Ironically, linear SVMs perform well on many tasks
Summary

- Find decision hyperplane that maximizes margin
Summary

- Find decision hyperplane that maximizes margin

- Employ soft-margin to support noisy data
Summary

- Find decision hyperplane that maximizes margin
- Employ soft-margin to support noisy data
- For non-linearly separable data, use non-linear SVMs
  - Project to higher dimensional space to separate
  - Use kernel trick to avoid intractable computation of
    - Projection or inner products
MaxEnt vs SVM

<table>
<thead>
<tr>
<th></th>
<th>MaxEnt</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling</td>
<td>Maximize ( P(Y</td>
<td>X, \lambda) )</td>
</tr>
<tr>
<td>Training</td>
<td>Learn ( \lambda ) for each feature function</td>
<td>Learn ( \alpha ) for each training instance</td>
</tr>
<tr>
<td>Decoding</td>
<td>Calculate ( P(y</td>
<td>x) )</td>
</tr>
<tr>
<td>Things to decode</td>
<td>Features</td>
<td>Kernel</td>
</tr>
<tr>
<td></td>
<td>Regularization</td>
<td>Regularization</td>
</tr>
<tr>
<td></td>
<td>Training algorithm</td>
<td>Training algorithm</td>
</tr>
<tr>
<td></td>
<td>Binarization</td>
<td>Binarization</td>
</tr>
</tbody>
</table>

Due to F. Xia
LibSVM

- Well-known SVM package
  - Chang & Lin, most recent version 2011

- Supports:
  - Many different SVM variants
  - Range of kernels
  - Efficient multi-class implementation
  - Range of language interfaces, as well as CLI
  - Frameworks for tuning – cross-validation, grid search

- Why libSVM? SVM not in Mallet
libSVM Information

- Main website:
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - Documentation & examples
  - FAQ
libSVM Information

- Main website:
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - Documentation & examples
  - FAQ

- Installation on patas:
  - /NLP_TOOLS/ml_tools/svm/libsvm/latest/
  - Calling prints usage
libSVM

• Main components:
  • svm-train:
    • Trains model on provided data

• svm-predict:
  • Applies trained model to test data
svm-train

- Usage:
  - svm-train [options] training_data model_file

- Options:
  - `-t`: kernel type
    - `[0-3]`
  - `-d`: degree – used in polynomial
  - `-g`: gamma – used in polynomial, RBF, sigmoid
  - `-r`: coef0 – used in polynomial, sigmoid
  - lots of others
LibSVM Kernels

- Predefined libSVM kernels
  - Set with –t switch: default 2
  - 0 -- linear: $u'v$
  - 1 -- polynomial: $(\gamma u'v + \text{coef0})^\text{degree}$
  - 2 -- radial basis function: $\exp(-\gamma |u-v|^2)$
  - 3 -- sigmoid: $\tanh(\gamma u'v + \text{coef0})$
**svm-predict**

- **Usage:**
  - `svm-predict testing_file model_file output_file`

- Prints results to `output_file`
  - Accuracy printed to `stderr`

- **HW #8:** Implement SVM decoder using SVM model
Training File Format

- Training file:
  - Sequence of lines:
    - Each line represents a training instance
Training File Format

• Training file:
  • Sequence of lines:
    • Each line represents a training instance

• Mallet format:
  • instanceID classLabel f1 v1 ...

• LibSVM format:
  • classNumber featidx1:v1 featidx2:v2.....
Training File Format

- **Training file:**
  - Sequence of lines:
    - Each line represents a training instance

- **Mallet format:**
  - instanceID classLabel f1 v1 ...

- **LibSVM format:**
  - classNumber featidx1:v1 featidx2:v2.....

- featidxes numeric, values numeric
  - Sparse: omit idx:value pair if value = 0
  - Increasing order by idx
Model File Format Example

- `svm_type c_svc`
- `kernel_type linear`
- `nr_class 2`
- `total_sv 535`
- `rho 0.281122`
- `label 0 1`
- `nr_sv 272 263`
- `SV`
- `0.004437478408154137 0:1 1:1 2:1 3:1 4:1 5:1 6:1 7:1`

 Due to F. Xia
Model File Format Example

- svm_type c_svc
- kernel_type linear
- nr_class 2
- total_sv 535
- rho 0.281122 corresponds to -b
- label 0 1
- nr_sv 272 263
- SV
- 0.004437478408154137 0:1 1:1 2:1 3:1 4:1 5:1 6:1 7:1

Due to F. Xia
Model File Format Example

- svm_type c_svc
- kernel_type linear
- nr_class 2
- total_sv 535
- rho 0.281122 \( \text{corresponds to} \cdot b \)
- label 0 1
- nr_sv 272 263
- SV
- Weight: \( \alpha_i y_i \)
- 0.004437478408154137 0:1 1:1 2:1 3:1 4:1 5:1 6:1 7:1

Due to F. Xia
Model File Format Example

- svm_type c_svc
- kernel_type linear
- nr_class 2
- total_sv 535
- rho 0.281122  corresponds to -b
- label 0 1
- nr_sv 272 263
- SV
- 0.004437478408154137 0:1 1:1 2:1 3:1 4:1 5:1 6:1 7:1

Weight:  $$\alpha_i y_i$$  
Corresponding support vector

Due to F. Xia
Applying the Model File

- Classification:
  - \( f(x) = \text{sign} \ (<w,x>+b) \)

- In papers:
  \[ f(x) = \sum_i \alpha_i y_i K(x_i, x) + b \]
  - where \( y_i \) is class of \( x \), \( c_0=+1 \) and \( c_1=-1 \)
Applying the Model File

- Classification:
  - $f(x) = \text{sign} \ (<w.x> + b)$

- In papers:
  - $f(x) = \sum_i \alpha_i y_i K(x_i, x) + b$
  - where $y_i$ is class of $x$, $c_0 = +1$ and $c_1 = -1$

- In libSVM:
  - $f(x) = \sum_i \text{weight}_i K(x_i, x) - \rho$
Applying the Model File

- Classification:
  - \( f(x) = \text{sign} \left( \langle w, x \rangle + b \right) \)

- In papers:
  \[
  f(x) = \sum_i \alpha_i y_i K(x_i, x) + b
  \]
  - where \( y_i \) is class of \( x \), \( c_0=+1 \) and \( c_1=-1 \)

- In libSVM:
  \[
  f(x) = \sum_i \text{weight}_i K(x_i, x) - \rho
  \]

- if \( f(x) > 0 \),
  - label with class \( c_0 \)
  - else \( c_1 \)
Output File Format

- Output format:
  - Sequence of lines
    - One label per line – corresponding to input

- Binary classification: classes 0/1 (not -1,1)
  - 0  $\Rightarrow c_0$
  - 0
  - 1  $\Rightarrow c_1$
  - 1
  - 0
## Notation Mapping

<table>
<thead>
<tr>
<th></th>
<th>Papers</th>
<th>libSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>$\alpha_i$ $y_i$ $x_i$ $b$</td>
<td>$\text{weight}_i$ $x_i$ $\rho$</td>
</tr>
<tr>
<td><strong>Decoding</strong></td>
<td>$f(x) = \sum_i \alpha_i y_i K(x_i, x) + b$</td>
<td>$f(x) = \sum_i \text{weight}_i K(x_i, x) - \rho$</td>
</tr>
<tr>
<td><strong>Labels</strong></td>
<td>+1, -1</td>
<td>0, 1</td>
</tr>
</tbody>
</table>
HW #8

- Run libSVM on binary text classification task
  - Binary in both senses:
    - 2 classes
    - Values are 0/1

- Build SVM decoder using model in step 1
  - Implement different kernel functions