
A bit about Vapnik

- Started SVM study in late 70s
- Fully developed in late 90s
- While at AT&T lab

http://en.wikipedia.org/wiki/Vladimir_Vapnik
Generalization and Capacity

- For a given learning task, with a given finite amount of training data, the best generalization performance will be achieved if the right balance is struck between the accuracy attained on that particular training set, and the “capacity” of the machine.
- Capacity – the ability of the machine to learn any training set without error
  - Too much capacity - overfitting

Bounds on the Balance

- Under what circumstances, and how quickly, the mean of some empirical quantity converges uniformly, as the number of data point increases, to the true mean.
- True mean error (or actual risk)
- One of the bounds

\[
\begin{align*}
R(\alpha) &= \int \frac{1}{2} \left| y - f(x) \right| dx \\
R_{\text{emp}}(\alpha) &= \frac{1}{m} \sum_{i=1}^{m} \left| y_i - f(x_i) \right|
\end{align*}
\]

Principled method: choose a learning machine that minimizes the RHS with a sufficiently small \( \eta \).
**VC Dimension**

- For a given set of \( i \) points, there can be \( 2^i \) ways to label them. For each labeling, if a member of the set \( \{f(x)\} \) can be found that correctly classifies them, we say that set of points is **shattered** by that set of functions.
- VC dimension of that set of functions \( \{f(x)\} \) is defined as the maximum number of training points that can be shattered by \( \{f(x)\} \).
- We should minimize \( h \) in order to minimize the bound.

**Example (f(\( \alpha \)) is perceptron)**

![Image](image1.png)

*Figure 1: Three points in \( \mathbb{R}^2 \) shattered by oriented lines.*

**Linear SVM – The Separable Case**

![Image](image2.png)

*Figure 2: Linear SVM – The Separable Case.*

- \( w \cdot x + b = 1 \)
- Decision boundary: \( w \cdot x + b = 0 \)
- Support vectors
- Margin
- Origin
- \( H_1 \)
- \( H_2 \)
- \( b \)
\[
\begin{align*}
\begin{cases}
    x \cdot w + b \geq 1 & \text{for } y = +1 \\
    x \cdot w + b \leq -1 & \text{for } y = -1
\end{cases}
\end{align*}
\]

Minimizing \( \|w\| \)

s.t. \( y(x \cdot w + b) = 1 \geq 0 \)

Minimize \( L_w = \frac{1}{2} \|w\|^2 + \sum \alpha_i y_i (x_i \cdot w + b) + \sum \xi_i \)

\[
\frac{\partial L_w}{\partial w} = 0 \Rightarrow w = \sum \alpha_i y_i x_i
\]

\[
\frac{\partial L_w}{\partial b} = 0 \Rightarrow \sum \alpha_i y_i = 0
\]

Maximize \( L_\alpha = -\frac{1}{2} \sum \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum \xi_i \)

---

Non-separable Cases

- SVM with soft margin
- Kernel trick

---

Non-separable Case – Soft Margin

\[
\begin{align*}
\begin{cases}
    x \cdot w + b \geq 1 - \xi & \text{for } y = +1 \\
    x \cdot w + b \leq -1 + \xi & \text{for } y = -1
\end{cases}
\end{align*}
\]

Minimizing \( \|w\| \)

s.t. \( y(x \cdot w + b) = 1 - \xi \geq 0 \)

Minimize \( L_w = \frac{1}{2} \|w\|^2 + C \left( \sum \xi_i \right) \)

Maximize \( L_\alpha = -\frac{1}{2} \sum \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum \alpha_i \)

s.t. \( \alpha_i \leq C, \quad \sum \alpha_i y_i = 0 \)
Non-separable Cases – Kernel Trick

If there were a "kernel function", \( K \), s.t.

\[
K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) = e^{-\frac{|x_i - x_j|^2}{2\sigma^2}}
\]

Gaussian Radial Basis Function (RBF)

Comparison - XOR

Limitation

Need to choose parameters
Packages

- **libSVM**
  - Use one-against-one (1a)
- **SVMlight**

Package Installation

- Installation (Three choices)
  - On Unix systems, type `make` to build the `svm-train` and `svm-predict` programs.
  - On other systems, consult `Makefile` to build them.
  - Use the pre-built binaries (Windows binaries are in the directory `windows`).
  - More details pls refer to the README file

Steps

- **Step 1**: Transform the data to the format of an SVM package
- **Step 2**: Conduct simple scaling on the data
- **Step 3**: Consider the RBF kernel $k(x, y) = e^{-|x-y|^2}$
- **Step 4**: Select the best parameter $c$ and $\gamma$ to train the whole training set
- **Step 5**: Test
Example

- Dataset: pima.tr and pima.te
- Step 1: Transform the data to the format of an SVM package
  - \( P_{tr} \in \mathbb{R}^{n \times l} \) (training data: every row is a feature vector)
  - \( P_{te} \in \mathbb{R}^{n \times l} \) (testing data: every row is a feature vector)
  - \( l_{tr} \) (label vector for training data pima.tr)
  - \( l_{te} \) (label vector for testing data pima.te)

Example

- Step 2: Data scaling
  - Avoid attributes in greater numeric ranges dominating those in smaller numeric ranges
  - Avoid numerical difficulties during the calculation
- How?
  - Calculate the min and max for every feature from the training dataset
  - For every feature (train or test) \( f \), the scaled feature \( f_{s} \) can be calculated by \( f_{s} = (f - \text{min}) / (\text{max} - \text{min}) \)

Details pls refer to:

Example

- Step 3: Train SVM on given parameters
  \[
  \text{model} = \text{svmtrain}(l_{tr}, \text{Sparse(pima_trs)}, '-c 16 , -g 0.1');
  \]
  - Trained model
  - Label vector for training dataset
  - Penalty parameter
  - Parameter for the kernel function RBF
  - 1. Pima_trs is the matrix for the scaled features of training dataset
  - 2. Sparse(pima_trs) is an operation to generate a sparse matrix in matlab, required by the libsvm package
Step 4: Test on the trained model

```matlab
[predict_labels, accuracy, dec_values] = svmpredict(lte, sparse(pima_tes), model);
```

1. `Pima_tes` is the matrix for the scaled features of testing dataset.
2. `Sparse(pima_tes)` is an operation to generate a sparse matrix in MATLAB, required by the libsvm package.

Result:
- **Scaled:**
  - Accuracy = 80.1205% (266/332)
- **Non-scaled:**
  - Accuracy = 66.8675% (222/332)

Matlab Code

```matlab
% Demonstration of the usage of libSVM on pima data set
% Jiajia Luo

clear all;
clc;

% add the path
addpath('E:\My Code\SourceCode\Internet\libsvm-3.1\windows');

% Step 1: Collect the training and testing dataset
fid = fopen('pima.tr.txt');
tr = textscan(fid,'%f %f %f %f %f %f %f %s','HeaderLines',1);
fclose(fid);
fid = fopen('pima.te.txt');
te = textscan(fid,'%f %f %f %f %f %f %f %s','HeaderLines',1);
fclose(fid);

% Step 2: Generate the feature vectors and labels for tr and te
pima_tr = [tr{1} tr{2} tr{3} tr{4} tr{5} tr{6} tr{7}];
Ntr = size(pima_tr,1);
Pima_tes = [te{1} te{2} te{3} te{4} te{5} te{6} te{7}];
Nte = size(Pima_tes,1);
ltr = [tr{8}];
lte = [te{8}];
label_tr = zeros(Ntr,1);
label_te = zeros(Nte,1);
for i = 1:Ntr
    if strcmp(ltr{i},'Yes')
        label_tr(i,1) = 1;
    elseif strcmp(ltr{i},'No')
        label_tr(i,1) = 2;
    end
end
```
for i = 1:Nte
    if strcmp(lte{i},'Yes')
        label_te(i,1) = 1;
    elseif strcmp(lte{i},'No')
        label_te(i,1) = 2;
    end
end

% Step 3: scale the data
pima_trs = (pima_tr - repmat(min(pima_tr,[],1),size(pima_tr,1),1))*...
    spdiags(1./(max(pima_tr,[],1)
    - min(pima_tr,[],1))',0,size(pima_tr,2),size(pima_tr,2));
pima_tes = (pima_te - repmat(min(pima_tr,[],1),size(pima_te,1),1))*...
    spdiags(1./(max(pima_tr,[],1)
    - min(pima_tr,[],1))',0,size(pima_te,2),size(pima_te,2));

% Step 4: train the data
model = svmtrain(label_tr, sparse(pima_trs), ...
    '-c 16 -g 0.1');

% Step 5: test the data
[predict_labels, accuracy, dec_value_s] = svmpredict(label_te, sparse(pima_tes), model);

% Step 6: Evaluate the performance of using unscaled data
model = svmtrain(label_tr, sparse(pima_tr), ...
    '-c 2 -g 0.1');
[predict_labels, accuracy, dec_value] = svmpredict(label_te, sparse(pima_te), model);