Support Vector Machines and Kernel Methods: Status and Challenges

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at K. U. Leuven Optimization in Engineering Center,
January 15, 2013
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Support Vector Classification

- **Training vectors**: \( x_i, i = 1, \ldots, l \)
- Feature vectors. For example,
  A patient = [height, weight, ...]\( ^T \)
- Consider a simple case with **two classes**:
  Define an **indicator vector** \( y \)
  \[ y_i = \begin{cases} 
  1 & \text{if } x_i \text{ in class 1} \\
  -1 & \text{if } x_i \text{ in class 2} 
  \end{cases} \]
- A hyperplane which separates all data
A separating hyperplane: \( \mathbf{w}^T \mathbf{x} + b = 0 \)

\[
\begin{align*}
(w^T x_i) + b & \geq 1 \quad \text{if } y_i = 1 \\
(w^T x_i) + b & \leq -1 \quad \text{if } y_i = -1
\end{align*}
\]

Decision function \( f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x} + b) \), \( \mathbf{x} \): test data

Many possible choices of \( \mathbf{w} \) and \( b \)
Maximal Margin

- Distance between $\mathbf{w}^T \mathbf{x} + b = 1$ and $-1$:
  \[ 2/||\mathbf{w}|| = 2/\sqrt{\mathbf{w}^T \mathbf{w}} \]

- A quadratic programming problem (Boser et al., 1992)

\[
\begin{align*}
\min_{\mathbf{w}, b} & \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} \\
\text{subject to} & \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \\
& \quad i = 1, \ldots, l.
\end{align*}
\]
Data May Not Be Linearly Separable

- An example:

- Allow training errors
- Higher dimensional (maybe infinite) feature space

\[ \phi(x) = [\phi_1(x), \phi_2(x), \ldots]^T. \]
Standard SVM (Boser et al., 1992; Cortes and Vapnik, 1995)

\[
\begin{align*}
\min_{\mathbf{w}, b, \xi} & \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{l} \xi_i \\
\text{subject to} & \quad y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\
& \quad \xi_i \geq 0, \ i = 1, \ldots, l.
\end{align*}
\]

Example: \( \mathbf{x} \in \mathbb{R}^3, \phi(\mathbf{x}) \in \mathbb{R}^{10} \)

\[
\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2, \ x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3]^T
\]
Finding the Decision Function

- \( \mathbf{w} \): maybe infinite variables
- The dual problem: finite number of variables

\[
\begin{align*}
\min_{\mathbf{\alpha}} \quad & \frac{1}{2} \mathbf{\alpha}^T \mathbf{Q} \mathbf{\alpha} - \mathbf{e}^T \mathbf{\alpha} \\
\text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l \\
& \mathbf{y}^T \mathbf{\alpha} = 0,
\end{align*}
\]

where \( Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j) \) and \( \mathbf{e} = [1, \ldots, 1]^T \)

- At optimum

\[
\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i)
\]

- A finite problem: \#variables = \#training data
Kernel Tricks

- \( Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j) \) needs a closed form
- Example: \( x_i \in \mathbb{R}^3, \phi(x_i) \in \mathbb{R}^{10} \)

\[
\phi(x_i) = [1, \sqrt{2}(x_i)_1, \sqrt{2}(x_i)_2, \sqrt{2}(x_i)_3, (x_i)_1^2, (x_i)_2^2, (x_i)_3^2, \sqrt{2}(x_i)_1(x_i)_2, \sqrt{2}(x_i)_1(x_i)_3, \sqrt{2}(x_i)_2(x_i)_3]^T
\]

Then \( \phi(x_i)^T \phi(x_j) = (1 + x_i^T x_j)^2 \).

- Kernel: \( K(x, y) = \phi(x)^T \phi(y) \); common kernels:

\[
e^{-\gamma \|x_i - x_j\|^2}, \text{ (Radial Basis Function)}
\]

\[
(x_i^T x_j / a + b)^d \text{ (Polynomial kernel)}
\]
Can be inner product in \textbf{infinite} dimensional space

Assume \( x \in R^1 \) and \( \gamma > 0 \).

\[
e^{-\gamma \|x_i - x_j\|^2} = e^{-\gamma (x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2}
\]

\[
e^{-\gamma x_i^2 - \gamma x_j^2} \left( 1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots \right)
\]

\[
e^{-\gamma x_i^2 - \gamma x_j^2} \left( 1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}} x_i \cdot \sqrt{\frac{2\gamma}{1!}} x_j + \sqrt{\frac{(2\gamma)^2}{2!}} x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}} x_j^2 + \cdots \right) = \phi(x_i)^T \phi(x_j),
\]

where

\[
\phi(x) = e^{-\gamma x^2} \left[ 1, \sqrt{\frac{2\gamma}{1!}} x, \sqrt{\frac{(2\gamma)^2}{2!}} x^2, \sqrt{\frac{(2\gamma)^3}{3!}} x^3, \cdots \right]^T
\]
Issues

- So what kind of kernel should I use?
- What kind of functions are valid kernels?
- How to decide kernel parameters?
- Some of these issues will be discussed later
Decision function

At optimum

\[ w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i) \]

Decision function

\[ w^T \phi(x) + b \]

\[ = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i)^T \phi(x) + b \]

\[ = \sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b \]

Only \( \phi(x_i) \) of \( \alpha_i > 0 \) used \( \Rightarrow \) support vectors
Support Vectors: More Important Data

Only $\phi(x_i)$ of $\alpha_i > 0$ used $\Rightarrow$ support vectors
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Deriving the Dual

- For simplification, consider the problem without $\xi_i$

  $\begin{align*}
  \min_{w,b} & \quad \frac{1}{2} w^T w \\
  \text{subject to} & \quad y_i (w^T \phi(x_i) + b) \geq 1, \ i = 1, \ldots, l.
  \end{align*}$

- Its dual is

  $\begin{align*}
  \min_{\alpha} & \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
  \text{subject to} & \quad 0 \leq \alpha_i, \quad i = 1, \ldots, l, \\
  & \quad y^T \alpha = 0.
  \end{align*}$
Lagrangian Dual

\[
\max_{\alpha \geq 0} \min_{w, b} \left( \min_{L(w, b, \alpha)} \right),
\]

where

\[
L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i \left( y_i (w^T \phi(x_i) + b) - 1 \right)
\]

Strong duality (be careful about this)

\[
\min \text{ Primal} = \max_{\alpha \geq 0} \left( \min_{w, b} L(w, b, \alpha) \right)
\]
Simplify the dual. When $\alpha$ is fixed,

$$\min_{w, b} L(w, b, \alpha) =$$

$$\begin{cases} 
-\infty & \text{if } \sum_{i=1}^l \alpha_i y_i \neq 0, \\
\min_{w} \frac{1}{2} w^T w - \sum_{i=1}^l \alpha_i [y_i (w^T \phi(x_i) - 1)] & \text{if } \sum_{i=1}^l \alpha_i y_i = 0.
\end{cases}$$

If $\sum_{i=1}^l \alpha_i y_i \neq 0$, we can decrease

$$-b \sum_{i=1}^l \alpha_i y_i$$

in $L(w, b, \alpha)$ to $-\infty$. 

Chih-Jen Lin (National Taiwan Univ.)
If $\sum_{i=1}^{l} \alpha_i y_i = 0$, optimum of the strictly convex function

$$\frac{1}{2} w^T w - \sum_{i=1}^{l} \alpha_i [y_i (w^T \phi(x_i) - 1]$$

happens when

$$\nabla_w L(w, b, \alpha) = 0.$$ 

Thus,

$$w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i).$$
Note that

\[ w^T w = \left( \sum_{i=1}^{l} \alpha_i y_i \phi(x_i) \right)^T \left( \sum_{j=1}^{l} \alpha_j y_j \phi(x_j) \right) \]

\[ = \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \]

The dual is

\[
\max_{\alpha \geq 0} \left\{ \begin{array}{ll}
\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) & \text{if } \sum_{i=1}^{l} \alpha_i y_i = 0, \\
-\infty & \text{if } \sum_{i=1}^{l} \alpha_i y_i \neq 0.
\end{array} \right. 
\]
- Lagrangian dual: \( \max_{\alpha \geq 0} \left( \min_{w, b} L(w, b, \alpha) \right) \)
- \(-\infty\) definitely not maximum of the dual
  Dual optimal solution not happen when
  \[
  \sum_{i=1}^{l} \alpha_i y_i \neq 0
  \]

- Dual simplified to

  \[
  \max_{\alpha \in R^l} \left\{ \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \right\}
  \]

  subject to

  \[
  y^T \alpha = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, l.
  \]
More about Dual Problems

- After SVM is popular
  Quite a few people think that for any optimization problem
  \[ \Rightarrow \] Lagrangian dual exists and strong duality holds

- Wrong! We usually need
  Convex programming; Constraint qualification

- We have them
  SVM primal is convex; Linear constraints
Our problems may be infinite dimensional
Can still use Lagrangian duality
See a rigorous discussion in Lin (2001)
Primal versus Dual

Recall the dual problem is

\[
\min_{\alpha} \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha
\]

subject to

\[
0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l
\]

\[
y^T \alpha = 0
\]

and at optimum

\[
w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i)
\]  

(1)
Primal versus Dual (Cont’d)

- What if we put (1) into primal

\[
\min_{\alpha, \xi} \quad \frac{1}{2} \alpha^T Q \alpha + C \sum_{i=1}^{l} \xi_i \\
\text{subject to} \quad (Q \alpha + b y)_i \geq 1 - \xi_i \quad (2) \\
\xi_i \geq 0
\]

- If \( Q \) is positive definite, we can prove that the optimal \( \alpha \) of (2) is the same as that of the dual

- So dual is not the only choice to solve when we use kernels
Other Variants

- A general form for binary classification

$$\min_w r(w) + C \sum_{i=1}^l \xi(w; x_i, y_i)$$

- $r(w)$: regularization term
- $\xi(w; x, y)$: loss function: we hope $y w^T x > 0$
- $C$: regularization parameter
Loss Functions

- Some commonly used loss functions:

\[ \xi_{L1}(\mathbf{w}; \mathbf{x}, y) \equiv \max(0, 1 - y\mathbf{w}^T\mathbf{x}), \]  
\[ \xi_{L2}(\mathbf{w}; \mathbf{x}, y) \equiv \max(0, 1 - y\mathbf{w}^T\mathbf{x})^2, \quad \text{and} \]  
\[ \xi_{LR}(\mathbf{w}; \mathbf{x}, y) \equiv \log(1 + e^{-y\mathbf{w}^T\mathbf{x}}). \]

- We omit the bias term \( b \) here.
- SVM (Boser et al., 1992; Cortes and Vapnik, 1995): (3)-(4)
- Logistic regression (LR): (5)
Loss Functions (Cont’d)

\[ \xi(w; x, y) \]

\[ -y w^T x \]

- Indeed SVM and logistic regression are very similar
If we use square loss function

\[ \xi(w; x, y) \equiv (1 - yw^T x)^2 \]

it becomes least-square SVM (Suykens and Vandewalle, 1999) or Gaussian process.
Regularization

- L1 versus L2
  \[ \|w\|_1 \text{ and } w^T w / 2 \]
- \( w^T w / 2 \): smooth, easier to optimize
- \( \|w\|_1 \): non-differentiable
  - sparse solution; possibly many zero elements
- Possible advantages of L1 regularization:
  - Feature selection
  - Less storage for \( w \)
The main issue is to solve the dual problem:

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

subject to

$$0 \leq \alpha_i \leq C, \ i = 1, \ldots, l$$

$$y^T \alpha = 0$$

This will be discussed in Thursday’s lecture, which talks about the connection between optimization and machine learning.
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- **Practical use of SVM**
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Let’s Try a Practical Example

A problem from astroparticle physics

1 2.61e+01 5.88e+01 -1.89e-01 1.25e+02
1 5.70e+01 2.21e+02 8.60e-02 1.22e+02
1 1.72e+01 1.73e+02 -1.29e-01 1.25e+02
0 2.39e+01 3.89e+01 4.70e-01 1.25e+02
0 2.23e+01 2.26e+01 2.11e-01 1.01e+02
0 1.64e+01 3.92e+01 -9.91e-02 3.24e+01

Training and testing sets available: 3,089 and 4,000
Data available at LIBSVM Data Sets
Training and Testing

Training the set svmguide1 to obtain svmguide1.model

$./svm-train svmguide1$

Testing the set svmguide1.t

$./svm-predict svmguide1.t svmguide1.model out$

Accuracy = 66.925% (2677/4000)

We see that training and testing accuracy are very different. Training accuracy is almost 100%

$./svm-predict svmguide1 svmguide1.model out$

Accuracy = 99.7734% (3082/3089)
Why this Fails

- Gaussian kernel is used here
- We see that most kernel elements have

\[ K_{ij} = e^{-\frac{\|x_i - x_j\|^2}{4}} \begin{cases} = 1 & \text{if } i = j, \\ \to 0 & \text{if } i \neq j. \end{cases} \]

because some features in large numeric ranges

- For what kind of data,

\[ K \approx I? \]
Why this Fails (Cont’d)

- If we have training data

\[ \phi(x_1) = [1, 0, \ldots, 0]^T \]
\[ \vdots \]
\[ \phi(x_l) = [0, \ldots, 0, 1]^T \]

then

\[ K = I \]

- Clearly such training data can be correctly separated, but how about testing data?

- So overfitting occurs
Overfitting

- See the illustration in the next slide
- In theory
  
  You can easily achieve 100% training accuracy
- This is useless
- When training and predicting a data, we should
  
  Avoid **underfitting**: small training error
  
  Avoid **overfitting**: small testing error
○ and ▲: training; ○ and △: testing
Data Scaling

- Without scaling, the above overfitting situation may occur.
- Also, features in greater numeric ranges may dominate.
- A simple solution is to linearly scale each feature to [0, 1] by:
  \[
  \text{feature value} - \text{min} \\
  \frac{\text{max} - \text{min}}{
  \text{max} - \text{min}}
  \]
- There are many other scaling methods.
- Scaling generally helps, but not always.
Data Scaling: Same Factors

A common mistake

```
./svm-scale -l -1 -u 1 svmguide1 > svmguide1.scale
./svm-scale -l -1 -u 1 svmguide1.t > svmguide1.t.scale
```

-1 -1 -u 1: scaling to $[-1, 1]$  

We need to use same factors on training and testing

```
./svm-scale -s range1 svmguide1 > svmguide1.scale
./svm-scale -r range1 svmguide1.t > svmguide1.t.scale
```

Later we will give a real example
After Data Scaling

Train scaled data and then predict

```
./svm-train svmguide1.scale
./svm-predict svmguide1.t.scale svmguide1.scale.model	svmguide1.t.predict
```

Accuracy = 96.15%

Training accuracy is now similar

```
./svm-predict svmguide1.scale svmguide1.scale.model	svmguide1.t.predict
```

Accuracy = 96.439%

For this experiment, we use parameters $C = 1, \gamma = 0.25$, but sometimes performances are sensitive to parameters
Practical use of SVM

Parameters versus Performances

- If we use $C = 20, \gamma = 400$
  
  ```
  $./svm-train -c 20 -g 400 svmguide1.scale
  $./svm-predict svmguide1.scale svmguide1.scale.model
  Accuracy = 100% (3089/3089)
  ```

- 100% training accuracy but
  
  ```
  $./svm-predict svmguide1.t.scale svmguide1.scale.model
  Accuracy = 82.7% (3308/4000)
  ```

- Very bad test accuracy

- Overfitting happens
For SVM, we may need to select suitable parameters.

They are $C$ and kernel parameters.

Example:

$\gamma$ of $e^{-\gamma \|x_i - x_j\|^2}$

$a, b, d$ of $(x_i^T x_j / a + b)^d$

How to select them so performance is better?
Performance Evaluation

- Available data $\Rightarrow$ training and validation
- Train the training; test the validation to estimate the performance
- A common way is $k$-fold cross validation (CV):
  - Data randomly separated to $k$ groups
  - Each time $k - 1$ as training and one as testing
- Select parameters/kernels with best CV result
- There are many other methods to evaluate the performance
Contour of CV Accuracy
The good region of parameters is quite large.

SVM is sensitive to parameters, but not that sensitive.

Sometimes default parameters work but it’s good to select them if time is allowed.
Example of Parameter Selection

Direct training and test

$./svm-train svmguide3
$./svm-predict svmguide3.t svmguide3.model o

→ Accuracy = 2.43902%

After data scaling, accuracy is still low

$./svm-scale -s range3 svmguide3 > svmguide3.scale
$./svm-scale -r range3 svmguide3.t > svmguide3.t.scale
$./svm-train svmguide3.scale
$./svm-predict svmguide3.t.scale svmguide3.scale.model o

→ Accuracy = 12.1951%
Example of Parameter Selection (Cont’d)

Select parameters by trying a grid of \((C, \gamma)\) values

$ python grid.py svmguide3.scale
...
128.0 0.125 84.8753

(Best \(C=128.0, \gamma=0.125\) with five-fold cross-validation rate=84.8753%)

Train and predict using the obtained parameters

$ ./svm-train -c 128 -g 0.125 svmguide3.scale
$ ./svm-predict svmguide3.t.scale svmguide3.scale.model svmguide3.t.predict

→ Accuracy = 87.8049%
Selecting Kernels

- RBF, polynomial, or others?
- For beginners, use RBF first
- Linear kernel: special case of RBF
  
  Accuracy of linear the same as RBF under certain parameters (Keerthi and Lin, 2003)

- Polynomial kernel:

  \[(x_i^T x_j / a + b)^d\]

  Numerical difficulties: \((< 1)^d \rightarrow 0, (> 1)^d \rightarrow \infty\)

  More parameters than RBF
Selecting Kernels (Cont’d)

- Commonly used kernels are Gaussian (RBF), polynomial, and linear.
- But in different areas, special kernels have been developed. Examples
  1. $\chi^2$ kernel is popular in computer vision
  2. String kernel is useful in some domains
A Simple Procedure for Beginners

After helping many users, we came up with the following procedure:

1. Conduct simple **scaling** on the data
2. Consider **RBF** kernel \( K(x, y) = e^{-\gamma \|x-y\|^2} \)
3. Use cross-validation to find the **best parameter** \( C \) and \( \gamma \)
4. Use the best \( C \) and \( \gamma \) to train the **whole** training set
5. Test

In LIBSVM, we have a python script `easy.py` implementing this procedure.
We proposed this procedure in an “SVM guide” (Hsu et al., 2003) and implemented it in LIBSVM.

From research viewpoints, this procedure is not novel. We never thought about submitting our guide somewhere.

But this procedure has been tremendously useful. Now almost the standard thing to do for SVM beginners.
A Real Example of Wrong Scaling

Separately scale each feature of training and testing data to $[0, 1]$.

$$/svm-scale -l 0 \text{svmguide4} > \text{svmguide4.scale}$$

$$/svm-scale -l 0 \text{svmguide4.t} > \text{svmguide4.t.scale}$$

$python easy.py \text{svmguide4.scale} \text{svmguide4.t.scale}$$

Accuracy = 69.2308\% (216/312) (classification)

The accuracy is low even after parameter selection

$$/svm-scale -l 0 -s \text{range4} \text{svmguide4} > \text{svmguide4.scale}$$

$$/svm-scale -r \text{range4} \text{svmguide4.t} > \text{svmguide4.t.scale}$$

$python easy.py \text{svmguide4.scale} \text{svmguide4.t.scale}$$

Accuracy = 89.4231\% (279/312) (classification)
A Real Example of Wrong Scaling (Cont’d)

With the correct setting, the 10 features in the test data `svmguide4.t.scale` have the following maximal values:

0.7402, 0.4421, 0.6291, 0.8583, 0.5385, 0.7407, 0.3982, 1.0000, 0.8218, 0.9874

Scaling the test set to [0, 1] generated an erroneous set.
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Multi-class Classification

- **k** classes
- One-against-the rest: Train **k** binary SVMs:
  1st class vs. \((2, \cdots, k)\)th class
  2nd class vs. \((1, 3, \ldots, k)\)th class
  ...
- **k** decision functions

\[
(w^1)^T \phi(x) + b_1 \\
\vdots \\
(w^k)^T \phi(x) + b_k
\]
Prediction:

$$\arg\max_j (w^j)^T \phi(x) + b_j$$

Reason: If $x \in 1st$ class, then we should have

$$(w^1)^T \phi(x) + b_1 \geq +1$$
$$(w^2)^T \phi(x) + b_2 \leq -1$$
$$\vdots$$
$$(w^k)^T \phi(x) + b_k \leq -1$$
Multi-class Classification (Cont’d)

- One-against-one: train $k(k - 1)/2$ binary SVMs
  $(1, 2), (1, 3), \ldots, (1, k), (2, 3), (2, 4), \ldots, (k - 1, k)$

- If 4 classes $\Rightarrow$ 6 binary SVMs

<table>
<thead>
<tr>
<th>Class 1</th>
<th>Class 2</th>
<th>Class 3</th>
<th>Class 4</th>
<th>Decision functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_i = 1$</td>
<td>$f^{12}(x) = (w^{12})^T x + b^{12}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_i = 1$</td>
<td>$f^{13}(x) = (w^{13})^T x + b^{13}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_i = 1$</td>
<td>$f^{14}(x) = (w^{14})^T x + b^{14}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_i = -1$</td>
<td>$f^{23}(x) = (w^{23})^T x + b^{23}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_i = -1$</td>
<td>$f^{24}(x) = (w^{24})^T x + b^{24}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_i = -1$</td>
<td>$f^{34}(x) = (w^{34})^T x + b^{34}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For a testing data, predicting all binary SVMs

<table>
<thead>
<tr>
<th>Classes</th>
<th>winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1</td>
</tr>
<tr>
<td>1 3</td>
<td>1</td>
</tr>
<tr>
<td>1 4</td>
<td>1</td>
</tr>
<tr>
<td>2 3</td>
<td>2</td>
</tr>
<tr>
<td>2 4</td>
<td>4</td>
</tr>
<tr>
<td>3 4</td>
<td>3</td>
</tr>
</tbody>
</table>

Select the one with the largest vote

<table>
<thead>
<tr>
<th>class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td># votes</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

May use decision values as well
More Complicated Forms

- Solving a **single** optimization problem (Weston and Watkins, 1999; Crammer and Singer, 2002; Lee et al., 2004)
- There are many other methods
- A comparison in Hsu and Lin (2002)
- RBF kernel: accuracy similar for different methods
  - But 1-against-1 is the fastest for training
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
SVM doesn’t Scale Up

Yes, if using kernels

- Training millions of data is time consuming
- Cases with many support vectors: quadratic time bottleneck on $Q_{SV}, sv$

- For noisy data: # SVs increases linearly in data size (Steinwart, 2003)

Some solutions

- Parallelization
- Approximation
Parallelization

Multi-core/Shared Memory/GPU

- **One line change of LIBSVM**

<table>
<thead>
<tr>
<th>Multicore</th>
<th>Shared-memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

50,000 data (kernel evaluations: 80% time)

- GPU (Catanzaro et al., 2008); Cell (Marzolla, 2010)

Distributed Environments

- Chang et al. (2007); Zanni et al. (2006); Zhu et al. (2009).
Can be done in many aspects
Data level: sub-sampling
Optimization level:
  Approximately solve the quadratic program
Other non-intuitive but effective ways
  I will show one today
Many papers have addressed this issue
Approximately Training SVM (Cont’d)

Subsampling

- Simple and often effective

More advanced techniques

- Incremental training: (e.g., Syed et al., 1999)
  Data $\Rightarrow$ 10 parts
  train 1st part $\Rightarrow$ SVs, train SVs + 2nd part, ...
- Select and train good points: KNN or heuristics
  For example, Bakır et al. (2005)
Approximately Training SVM (Cont’d)

- **Approximate the kernel**; e.g., Fine and Scheinberg (2001); Williams and Seeger (2001)
- **Use part of the kernel**; e.g., Lee and Mangasarian (2001); Keerthi et al. (2006)
- **Early stopping** of optimization algorithms
  Tsang et al. (2005) and others
- **And many more**
  Some simple but some sophisticated
Sophisticated techniques may not be always useful

- Sometimes *slower than sub-sampling*

- **covtype**: 500k training and 80k testing
- **rcv1**: 550k training and 14k testing

<table>
<thead>
<tr>
<th>Training size</th>
<th>Accuracy</th>
<th>Training size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>50k</td>
<td>92.5%</td>
<td>50k</td>
<td>97.2%</td>
</tr>
<tr>
<td>100k</td>
<td>95.3%</td>
<td>100k</td>
<td>97.4%</td>
</tr>
<tr>
<td>500k</td>
<td>98.2%</td>
<td>550k</td>
<td>97.8%</td>
</tr>
</tbody>
</table>
Sophisticated techniques may not be always useful

- Sometimes **slower than sub-sampling**

<table>
<thead>
<tr>
<th>covtype</th>
<th>Training size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50k</td>
<td>92.5%</td>
</tr>
<tr>
<td></td>
<td>100k</td>
<td>95.3%</td>
</tr>
<tr>
<td></td>
<td>500k</td>
<td>98.2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rcv1</th>
<th>Training size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50k</td>
<td>97.2%</td>
</tr>
<tr>
<td></td>
<td>100k</td>
<td>97.4%</td>
</tr>
<tr>
<td></td>
<td>550k</td>
<td>97.8%</td>
</tr>
</tbody>
</table>
Discussion: Large-scale Training

- We don’t have many large and well labeled sets
  Expensive to obtain true labels
- Specific properties of data should be considered
  We will illustrate this point using linear SVM
- The design of software for very large data sets should be application different
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Linear and Kernel Classification

Methods such as SVM and logistic regression can used in two ways

- Kernel methods: data mapped to a higher dimensional space
  \[
  x \Rightarrow \phi(x)
  \]
  \[
  \phi(x_i)^T \phi(x_j) \text{ easily calculated; little control on } \phi(\cdot)
  \]

- Linear classification + feature engineering:
  We have \( x \) without mapping. Alternatively, we can say that \( \phi(x) \) is our \( x \); full control on \( x \) or \( \phi(x) \)

We refer to them as kernel and linear classifiers.
Linear and Kernel Classification

- Let's check the prediction cost

$$\mathbf{w}^T \mathbf{x} + b$$  versus  $$\sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- If $K(\mathbf{x}_i, \mathbf{x}_j)$ takes $O(n)$, then

$$O(n)$$  versus  $$O(nl)$$

- Linear is much cheaper
Linear and Kernel Classification (Cont’d)

- Also, linear is a special case of kernel
- Indeed, we can prove that accuracy of linear is the same as Gaussian (RBF) kernel under certain parameters (Keerthi and Lin, 2003)
- Therefore, roughly we have
  \[ \text{accuracy: kernel } \geq \text{ linear} \]
  \[ \text{cost: kernel } \gg \text{ linear} \]
- **Speed** is the reason to use linear
For some problems, accuracy by linear is as good as nonlinear
But training and testing are much faster
This particularly happens for document classification
Number of features (bag-of-words model) very large
Data very sparse (i.e., few non-zeros)
Recently linear classification is a popular research topic. Sample works in 2005-2008: Joachims (2006); Shalev-Shwartz et al. (2007); Hsieh et al. (2008)
## Linear SVM

**Comparison Between Linear and Kernel (Training Time & Testing Accuracy)**

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear</th>
<th>RBF Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Accuracy</td>
</tr>
<tr>
<td>MNIST38</td>
<td>0.1</td>
<td>96.82</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.6</td>
<td>91.81</td>
</tr>
<tr>
<td>covtype</td>
<td>1.4</td>
<td>76.37</td>
</tr>
<tr>
<td>news20</td>
<td>1.1</td>
<td>96.95</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.1</td>
<td>92.63</td>
</tr>
<tr>
<td>webspam</td>
<td>25.7</td>
<td>93.35</td>
</tr>
</tbody>
</table>

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features
## Comparison Between Linear and Kernel (Training Time & Testing Accuracy)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear</th>
<th>RBF Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Accuracy</td>
</tr>
<tr>
<td>MNIST38</td>
<td>0.1</td>
<td>96.82</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.6</td>
<td>91.81</td>
</tr>
<tr>
<td>covtype</td>
<td>1.4</td>
<td>76.37</td>
</tr>
<tr>
<td>news20</td>
<td>1.1</td>
<td>96.95</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.1</td>
<td>92.63</td>
</tr>
<tr>
<td>webspam</td>
<td>25.7</td>
<td>93.35</td>
</tr>
</tbody>
</table>

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features
## Comparison Between Linear and Kernel (Training Time & Testing Accuracy)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear Time</th>
<th>Linear Accuracy</th>
<th>RBF Kernel Time</th>
<th>RBF Kernel Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST38</td>
<td>0.1</td>
<td>96.82</td>
<td>38.1</td>
<td>99.70</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.6</td>
<td>91.81</td>
<td>26.8</td>
<td>98.69</td>
</tr>
<tr>
<td>covtype</td>
<td>1.4</td>
<td>76.37</td>
<td>46,695.8</td>
<td>96.11</td>
</tr>
<tr>
<td>news20</td>
<td>1.1</td>
<td>96.95</td>
<td>383.2</td>
<td>96.90</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
<td>938.3</td>
<td>97.82</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.1</td>
<td>92.63</td>
<td>20,955.2</td>
<td>93.31</td>
</tr>
<tr>
<td>webspam</td>
<td>25.7</td>
<td>93.35</td>
<td>15,681.8</td>
<td>99.26</td>
</tr>
</tbody>
</table>

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features
Extension: Training Explicit Form of Nonlinear Mappings

Linear-SVM method to train $\phi(x_1), \ldots, \phi(x_l)$

- Kernel not used
- Applicable only if dimension of $\phi(x)$ not too large

Low-degree Polynomial Mappings

$$K(x_i, x_j) = (x_i^T x_j + 1)^2 = \phi(x_i)^T \phi(x_j)$$

$$\phi(x) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, x_1^2, \ldots, x_n^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{n-1}x_n]^T$$

- When degree is small, train the explicit form of $\phi(x)$
## Testing Accuracy and Training Time

<table>
<thead>
<tr>
<th>Data set</th>
<th>Degree-2 Polynomial</th>
<th></th>
<th></th>
<th>Accuracy diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training time (s)</td>
<td>LIBLINEAR</td>
<td>LIBSVM</td>
<td>Accuracy</td>
</tr>
<tr>
<td>a9a</td>
<td>1.6</td>
<td>89.8</td>
<td>85.06</td>
<td>0.07</td>
</tr>
<tr>
<td>real-sim</td>
<td>59.8</td>
<td>1,220.5</td>
<td>98.00</td>
<td>0.49</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>10.7</td>
<td>64.2</td>
<td>97.84</td>
<td>5.63</td>
</tr>
<tr>
<td>MNIST38</td>
<td>8.6</td>
<td>18.4</td>
<td>99.29</td>
<td>2.47</td>
</tr>
<tr>
<td>covtype</td>
<td>5,211.9</td>
<td>NA</td>
<td>80.09</td>
<td>3.74</td>
</tr>
<tr>
<td>webspam</td>
<td>3,228.1</td>
<td>NA</td>
<td>98.44</td>
<td>5.29</td>
</tr>
</tbody>
</table>

Training $\phi(x_i)$ by linear: faster than kernel, but sometimes competitive accuracy.
Discussion: Directly Train $\phi(x_i), \forall i$

- See details in our work (Chang et al., 2010)
- Useful for certain applications
Outline

- Basic concepts: SVM and kernels
- Dual problem and SVM variants
- Practical use of SVM
- Multi-class classification
- Large-scale training
- Linear SVM
- Discussion and conclusions
Discussion and conclusions

Extensions of SVM

- Multiple Kernel Learning (MKL)
- Learning to rank
- Semi-supervised learning
- Active learning
- Cost sensitive learning
- Structured Learning
Conclusions

- SVM and kernel methods are rather mature areas
- But still quite a few interesting research issues
- Many are extensions of standard classification
- It is possible to identify more extensions through real applications
References I


References II


References III


Discussion and conclusions

References IV


