Introduction to Linear and Kernel Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at NCTU Summer School, July 9, 2014

Outline

- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- Discussion and conclusions

Outline

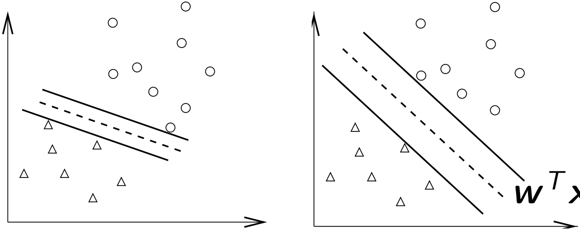
- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- Discussion and conclusions

Support Vector Classification

- Training vectors : $\boldsymbol{x}_i, i = 1, \ldots, l$
- Feature vectors. For example,
  A patient $=$ [height, weight, $\ldots]^T$
- Consider a simple case with two classes:
  Define an indicator vector $\mathbf{y}$

$$y_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \text{ in class 1} \\ -1 & \text{if } \boldsymbol{x}_i \text{ in class 2} \end{cases}$$

- A hyperplane which separates all data



$$w^T x + b = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$$

- A separating hyperplane: $\boldsymbol{w}^T \boldsymbol{x} + b = 0$

$$\begin{aligned} (\boldsymbol{w}^T \boldsymbol{x}_i) + b \geq 1 & \quad \text{if } y_i = 1 \\ (\boldsymbol{w}^T \boldsymbol{x}_i) + b \leq -1 & \quad \text{if } y_i = -1 \end{aligned}$$

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

  Many possible choices of $\boldsymbol{w}$ and b

Maximal Margin

- Distance between $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ and -1:

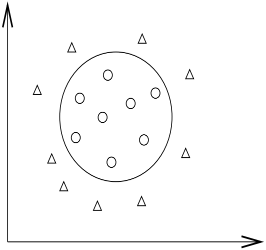
$$2/\|\boldsymbol{w}\| = 2/\sqrt{\boldsymbol{w}^T\boldsymbol{w}}$$

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

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \\ & i = 1, \ldots, l. \end{aligned}$$

Data May Not Be Linearly Separable

- An example:



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

$$\phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots]^T.$$

- Standard SVM (Boser et al., 1992; Cortes and Vapnik, 1995)

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

- Example: $\boldsymbol{x} \in R^3, \phi(\boldsymbol{x}) \in R^{10}$

$$
\begin{aligned}
\phi(\boldsymbol{x}) \ = \ & [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2, \\
& x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3]^T
\end{aligned}
$$

Finding the Decision Function

- $\boldsymbol{w}$: maybe infinite variables
- The dual problem: finite number of variables

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

where $Q_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ and $\boldsymbol{e} = [1, \ldots, 1]^T$

- At optimum

$$\boxed{\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i)}$$

- A finite problem: #variables = #training data

Kernel Tricks

- $Q_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ needs a closed form
- Example: $\boldsymbol{x}_i \in R^3, \phi(\boldsymbol{x}_i) \in R^{10}$

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

Then $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T \boldsymbol{x}_j)^2$.

- Kernel: $K(\boldsymbol{x}, \mathbf{y}) = \phi(\boldsymbol{x})^T \phi(\mathbf{y})$; common kernels:

$$e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}, \text{ (Radial Basis Function)}$$

$$(\boldsymbol{x}_i^T \boldsymbol{x}_j / a + b)^d \text{ (Polynomial kernel)}$$

Can be inner product in infinite dimensional space

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

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

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

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}}x_i \cdot \sqrt{\frac{2\gamma}{1!}}x_j + \sqrt{\frac{(2\gamma)^2}{2!}}x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}}x_j^2 \right.$$

$$\left. + \sqrt{\frac{(2\gamma)^3}{3!}}x_i^3 \cdot \sqrt{\frac{(2\gamma)^3}{3!}}x_j^3 + \cdots\right) = \phi(x_i)^T \phi(x_j),$$

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

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i)$$

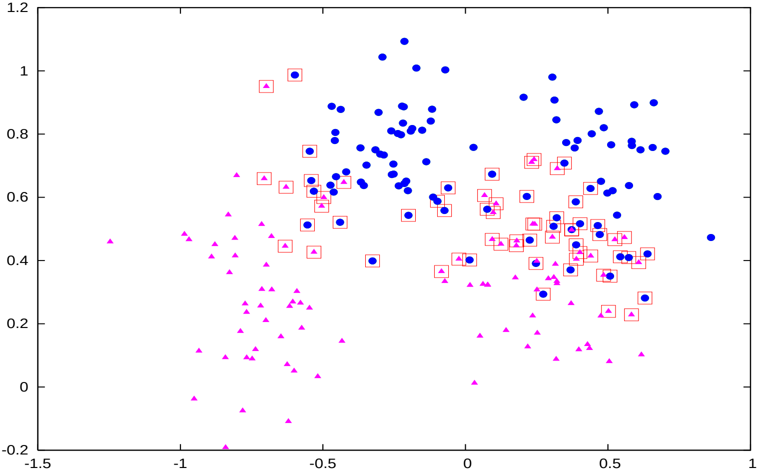
- Decision function

$$\begin{aligned} & \boldsymbol{w}^T \phi(\boldsymbol{x}) + b \\ = & \sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) + b \\ = & \sum_{i=1}^{l} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \end{aligned}$$

- Only $\phi(\boldsymbol{x}_i)$ of $\alpha_i > 0$ used $\Rightarrow$ support vectors

Support Vectors: More Important Data

Only $\phi(\boldsymbol{x}_i)$ of $\alpha_i > 0$ used $\Rightarrow$ support vectors



Large Dense Quadratic Programming

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

- $Q_{ij} \neq 0$, Q : an l by l fully dense matrix
- 50,000 training points: 50,000 variables: $(50,000^2 \times 8/2)$ bytes $=$ 10GB RAM to store Q
- This is a challenging problem though we won't discuss this subject today

Outline

- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- Discussion and conclusions

Deriving the Dual

- For simplification, consider the problem without ξ_i

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i)+b) \geq 1, i = 1,\dots,l. \end{aligned}$$

- Its dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i, \qquad i = 1,\dots,l, \\ & \mathbf{y}^T\boldsymbol{\alpha} = 0. \end{aligned}$$

Lagrangian Dual

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

where

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) - 1 \right)$$

Strong duality (be careful about this)

$$\min \ \text{Primal} = \max_{\boldsymbol{\alpha} \geq 0} \left(\min_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \right)$$

- Simplify the dual. When $\boldsymbol{\alpha}$ is fixed,

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

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

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

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

in $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ to $-\infty$

- If $\sum_{i=1}^{l} \alpha_i y_i = 0$, optimum of the strictly convex function

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

happens when

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0.$$

- Thus,

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i).$$

- Note that

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{w} &= \left(\sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i)\right)^T \left(\sum_{j=1}^{l} \alpha_j y_j \phi(\boldsymbol{x}_j)\right) \\ &= \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \end{aligned}$$

- The dual is

$$\max_{\boldsymbol{\alpha} \geq 0} \begin{cases} \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) & \text{if } \sum_{i=1}^{l} \alpha_i y_i = 0 \\ -\infty & \text{if } \sum_{i=1}^{l} \alpha_i y_i \neq 0 \end{cases}$$

- Lagrangian dual: $\max_{\boldsymbol{\alpha} \geq 0}(\min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}))$
- $-\infty$ 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

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in R^l} \quad & \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \\ \text{subject to} \quad & \mathbf{y}^T \boldsymbol{\alpha} = 0, \\ & \alpha_i \geq 0, i = 1, \ldots, l. \end{aligned}$$

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
- We can still use Lagrangian duality
  See a rigorous discussion in Lin (2001)

Primal versus Dual

- Recall the dual problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0 \end{aligned}$$

  and at optimum

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i) \tag{1}$$

Primal versus Dual (Cont'd)

- What if we put (1) into primal

$$
\begin{aligned}
\min_{\boldsymbol{\alpha},\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + C\sum_{i=1}^{l}\xi_i \\
\text{subject to} \quad & (Q\boldsymbol{\alpha} + b\mathbf{y})_i \geq 1 - \xi_i \\
& \xi_i \geq 0
\end{aligned} \tag{2}
$$

- If Q is positive definite, we can prove that the optimal $\boldsymbol{\alpha}$ of (2) is the same as that of the dual
- So dual is not the only choice to solve when we use kernels

Outline

General Form of Linear Classification

- A general form for binary classification

$$\min_{\boldsymbol{w}} \quad r(\boldsymbol{w}) + C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

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

Loss Functions

- Some commonly used loss functions:

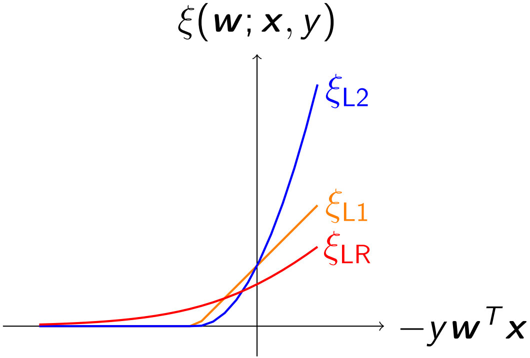
$$\xi_{\text{L1}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}), \tag{3}$$

$$\xi_{\text{L2}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x})^2, \quad \text{and} \tag{4}$$

$$\xi_{\text{LR}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \log(1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}). \tag{5}$$

- 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)



- Indeed SVM and logistic regression are very similar

Regularization

- L1 versus L2

$$\|\boldsymbol{w}\|_1 \text{ and } \boldsymbol{w}^T\boldsymbol{w}/2$$

- $\boldsymbol{w}^T\boldsymbol{w}/2$: smooth, easier to optimize
- $\|\boldsymbol{w}\|_1$: non-differentiable

  sparse solution; possibly many zero elements
- Possible advantages of L1 regularization:

  Feature selection

  Less storage for $\boldsymbol{w}$

Logistic Regression

- Logistic regression is often derived in the following way rather than regularization plus loss
- For a label-feature pair $(\boldsymbol{y}, \boldsymbol{x})$, assume the probability model

$$p(y|\boldsymbol{x}) = \frac{1}{1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}}.$$

- $\boldsymbol{w}$ is the parameter to be decided
- Assume

$$(y_i, \boldsymbol{x}_i), i = 1, \ldots, l$$

  are training instances

Logistic Regression (Cont'd)

- Logistic regression finds $\boldsymbol{w}$ by maximizing the following likelihood

$$\max_{\boldsymbol{w}} \quad \prod_{i=1}^{l} p\left(y_i | \boldsymbol{x}_i\right). \tag{6}$$

- Regularized logistic regression

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \log\left(1 + e^{-y_i \boldsymbol{w}^T\boldsymbol{x}_i}\right). \tag{7}$$

- If $\boldsymbol{x}$ replaced by $\phi(\boldsymbol{x})$, we have kernel logistic regression

Discussion

We see that the same method can be derived from different ways

SVM

- Maximal margin
- Regularization and training losses

LR

- Regularization and training losses
- Maximum likelihood

Outline

- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- 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

$$K_{ij} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/4} \begin{cases} = 1 & \text{if } i = j, \\ \to 0 & \text{if } i \neq j. \end{cases}$$

  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

$$\phi(\boldsymbol{x}_1) = [1, 0, \ldots, 0]^T$$
$$\vdots$$
$$\phi(\boldsymbol{x}_l) = [0, \ldots, 0, 1]^T$$

then

$$K = I$$

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

Overfitting

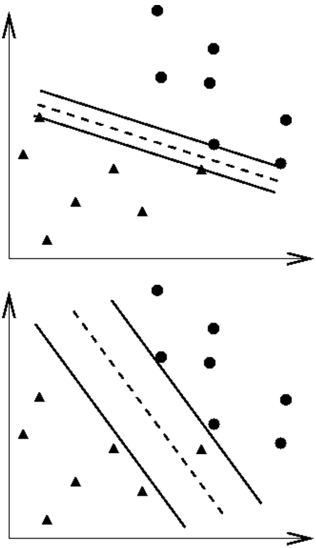
- See the illustration in the next slide
- In theory

  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:

$$\frac{\text{feature value} - \min}{\max - \min},$$

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

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

We need to use same factors on training and testing

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

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
  svmguide1.t.predict
Accuracy = 96.15%
```

Training accuracy is now similar

```
$./svm-predict svmguide1.scale svmguide1.scale.m
Accuracy = 96.439%
```

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

Parameters versus Performances

- If we use $C = 20, \gamma = 400$

```
$./svm-train -c 20 -g 400 svmguide1.scale
$./svm-predict svmguide1.scale svmguide1.sca
Accuracy = 100% (3089/3089)
```

- 100% training accuracy but

```
$./svm-predict svmguide1.t.scale svmguide1.s
Accuracy = 82.7% (3308/4000)
```

- Very bad test accuracy
- Overfitting happens

Parameter Selection

- For SVM, we may need to select suitable parameters
- They are C and kernel parameters
- Example:

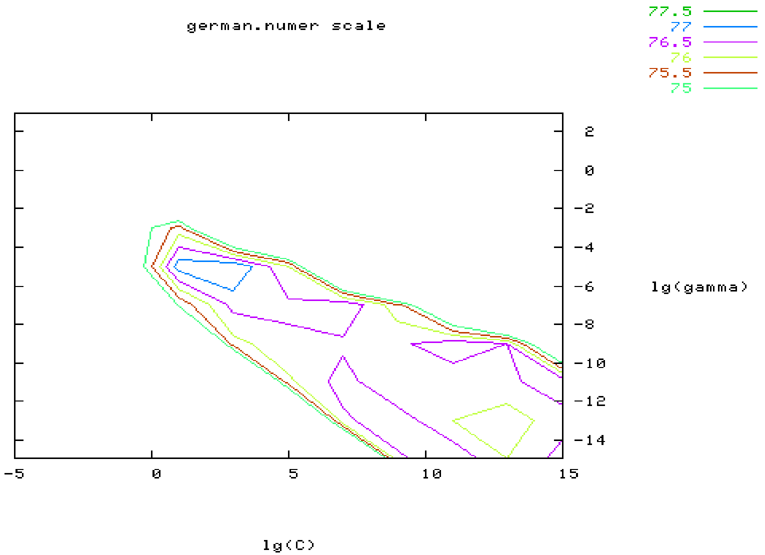
$$\gamma \text{ of } e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$$
$$a, b, d \text{ of } (\boldsymbol{x}_i^T \boldsymbol{x}_j / a + b)^d$$

- 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
```

$\rightarrow$ Accuracy $=$ 2.43902%

After data scaling, accuracy is still low

```
$./svm-scale -s range3 svmguide3 > svmguide3.sca
$./svm-scale -r range3 svmguide3.t > svmguide3.t
$./svm-train svmguide3.scale
$./svm-predict svmguide3.t.scale svmguide3.scale
```

$\rightarrow$ Accuracy $=$ 12.1951%

Example of Parameter Selection (Cont'd)

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

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

(Best C=128.0, γ=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.scal
```

$\rightarrow$ 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:

$$(\boldsymbol{x}_i^T \boldsymbol{x}_j / a + b)^d$$

  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. χ^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 $K(\boldsymbol{x}, \mathbf{y}) = e^{-\gamma\|\boldsymbol{x}-\mathbf{y}\|^2}$
3. Use cross-validation to find the best parameter C and γ
4. Use the best C and γ to train the whole training set
5. Test

In LIBSVM, we have a python script `easy.py` implementing this procedure.

A Simple Procedure for Beginners (Cont'd)

- 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 svmguide4 > svmguide4.scale
$ ../svm-scale -l 0 svmguide4.t > svmguide4.t.sc
$ python easy.py svmguide4.scale svmguide4.t.sca
Accuracy = 69.2308% (216/312) (classification)
```

The accuracy is low even after parameter selection

```
$ ../svm-scale -l 0 -s range4 svmguide4 > svmgu
$ ../svm-scale -r range4 svmguide4.t > svmguide
$ python easy.py svmguide4.scale svmguide4.t.sca
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
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- 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

$$Q_{\text{SV, SV}}$$

- 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

Multicore		Shared-memory	
1	80	1	100
2	48	2	57
4	32	4	36
8	27	8	28

  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).

Approximately Training SVM

- 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

Approximately Training SVM (Cont'd)

- Sophisticated techniques may not be always useful
- Sometimes slower than sub-sampling
- covtype: 500k training and 80k testing

  rcv1: 550k training and 14k testing

covtype		rcv1	
Training size	Accuracy	Training size	Accuracy
50k	92.5%	50k	97.2%
100k	95.3%	100k	97.4%
500k	98.2%	550k	97.8%

Approximately Training SVM (Cont'd)

- Sophisticated techniques may not be always useful
- Sometimes slower than sub-sampling
- covtype: 500k training and 80k testing

  rcv1: 550k training and 14k testing

covtype		rcv1	
Training size	Accuracy	Training size	Accuracy
50k	92.5%	50k	97.2%
100k	95.3%	100k	97.4%
500k	98.2%	550k	97.8%

Outline

- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- 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

$$\boldsymbol{x} \Rightarrow \phi(\boldsymbol{x})$$

  $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ easily calculated; little control on $\phi(\cdot)$

- Linear classification + feature engineering:

  We have $\boldsymbol{x}$ without mapping. Alternatively, we can say that $\phi(\boldsymbol{x})$ is our $\boldsymbol{x}$; full control on $\boldsymbol{x}$ or $\phi(\boldsymbol{x})$

We refer to them as kernel and linear classifiers

Linear and Kernel Classification

- Let's check the prediction cost

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

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

$$O(n) \quad \text{versus} \quad O(nl)$$

- 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

$$\begin{array}{ll} \text{accuracy:} & \text{kernel} \geq \text{linear} \\ \text{cost:} & \text{kernel} \gg \text{linear} \end{array}$$

- Speed is the reason to use linear

Linear and Kernel Classification (Cont'd)

- 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)

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

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

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

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

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

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

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

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

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Extension: Training Explicit Form of Nonlinear Mappings

Linear-SVM method to train $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_l)$

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

Low-degree Polynomial Mappings

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1)^2 = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

$$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, x_1^2, \ldots, x_n^2, \sqrt{2}x_1 x_2, \ldots, \sqrt{2}x_{n-1}x_n]^T$$

- When degree is small, train the explicit form of $\phi(\boldsymbol{x})$

Testing Accuracy and Training Time

Data set	Degree-2 Polynomial Training time (s) LIBLINEAR	LIBSVM	Accuracy	Accuracy diff. Linear	RBF
a9a	1.6	89.8	85.06	0.07	0.02
real-sim	59.8	1,220.5	98.00	0.49	0.10
ijcnn1	10.7	64.2	97.84	5.63	−0.85
MNIST38	8.6	18.4	99.29	2.47	−0.40
covtype	5,211.9	NA	80.09	3.74	−15.98
webspam	3,228.1	NA	98.44	5.29	−0.76

Training $\phi(\boldsymbol{x}_i)$ by linear: faster than kernel, but sometimes competitive accuracy

Discussion: Directly Train $\phi(\boldsymbol{x}_i), \forall i$

- See details in our work (Chang et al., 2010)
- A related development: Sonnenburg and Franc (2010)
- Useful for certain applications

Outline

- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- Discussion and conclusions

Multi-class Classification

- k classes
- One-against-the rest: Train k binary SVMs:

$$\begin{array}{lll} \text{1st class} & \text{vs.} & (2, \cdots, k)\text{th class} \\ \text{2nd class} & \text{vs.} & (1, 3, \ldots, k)\text{th class} \\ & \vdots & \end{array}$$

- k decision functions

$$(\boldsymbol{w}^1)^T \boldsymbol{x} + b_1$$

$$\vdots$$

$$(\boldsymbol{w}^k)^T \boldsymbol{x} + b_k$$

- Prediction:

$$\arg\max_{j} \; (\boldsymbol{w}^j)^T \boldsymbol{x} + b_j$$

- Reason: If $\boldsymbol{x} \in$ 1st class, then we should have

$$\begin{aligned}
(\boldsymbol{w}^1)^T \boldsymbol{x} + b_1 &\geq +1 \\
(\boldsymbol{w}^2)^T \boldsymbol{x} + b_2 &\leq -1 \\
&\vdots \\
(\boldsymbol{w}^k)^T \boldsymbol{x} + b_k &\leq -1
\end{aligned}$$

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

$y_i = 1$	$y_i = -1$	Decision functions
class 1	class 2	$f^{12}(\boldsymbol{x}) = (\boldsymbol{w}^{12})^T\boldsymbol{x} + b^{12}$
class 1	class 3	$f^{13}(\boldsymbol{x}) = (\boldsymbol{w}^{13})^T\boldsymbol{x} + b^{13}$
class 1	class 4	$f^{14}(\boldsymbol{x}) = (\boldsymbol{w}^{14})^T\boldsymbol{x} + b^{14}$
class 2	class 3	$f^{23}(\boldsymbol{x}) = (\boldsymbol{w}^{23})^T\boldsymbol{x} + b^{23}$
class 2	class 4	$f^{24}(\boldsymbol{x}) = (\boldsymbol{w}^{24})^T\boldsymbol{x} + b^{24}$
class 3	class 4	$f^{34}(\boldsymbol{x}) = (\boldsymbol{w}^{34})^T\boldsymbol{x} + b^{34}$

- For a testing data, predicting all binary SVMs

Classes		winner
1	2	1
1	3	1
1	4	1
2	3	2
2	4	4
3	4	3

- Select the one with the largest vote

class	1	2	3	4
# votes	3	1	1	1

- 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

  However, 1-against-1 is the fastest for training

Outline

- Basic concepts: SVM and kernels
- Dual problem
- Regularization, loss functions, and logistic regression
- Practical use of SVM
- Large-scale training of kernel classifiers
- Linear classification with fast training/prediction
- Multi-class classification
- Discussion and conclusions

Extensions of Linear and Kernel Classification

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

Conclusions

- Linear and kernel classification are rather mature areas
- However, there are still interesting research issues
- Many are extensions of standard classification
- It is possible to identify more extensions through real applications

References I

G. H. Bakır, L. Bottou, and J. Weston. Breaking svm complexity with cross-training. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 81–88. MIT Press, Cambridge, MA, 2005.

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008.

E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Parallelizing support vector machines on distributed computers. In *NIPS 21*, 2007.

Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490, 2010. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/lowpoly_journal.pdf`.

C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, (2–3):201–233, 2002.

References II

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf`.

C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

S. S. Keerthi and C.-J. Lin. Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation*, 15(7):1667–1689, 2003.

S. S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. *Journal of the American Statistical Association*, 99(465):67–81, 2004.

References III

Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the First SIAM International Conference on Data Mining*, 2001.

C.-J. Lin. Formulations of support vector machines: a note from an optimization point of view. *Neural Computation*, 13(2):307–317, 2001.

M. Marzolla. Optimized training of support vector machines on the cell processor. Technical Report UBLCS-2010-02, Department of Computer Science, University of Bologna, Italy, Feb. 2010. URL `http://www.cs.unibo.it/pub/TR/UBLCS/ABSTRACTS/2010.bib?ncstrl.cabernet//BOLOGNA-UBLCS-2010-02`.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *Proceedings of the Twenty Fourth International Conference on Machine Learning (ICML)*, 2007.

S. Sonnenburg and V. Franc. COFFIN : A computational framework for linear SVMs. In *Proceedings of the Twenty Seventh International Conference on Machine Learning (ICML)*, pages 999–1006, 2010.

I. Steinwart. Sparseness of support vector machines. *Journal of Machine Learning Research*, 4: 1071–1105, 2003.

N. A. Syed, H. Liu, and K. K. Sung. Incremental learning with support vector machines. In *Workshop on Support Vector Machines, IJCAI99*, 1999.

References IV

I. Tsang, J. Kwok, and P. Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

J. Weston and C. Watkins. Multi-class support vector machines. In M. Verleysen, editor, *Proceedings of ESANN99*, pages 219–224, Brussels, 1999. D. Facto Press.

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7: 1467–1492, 2006.

Z. A. Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen. P-packSVM: Parallel primal gradient descent kernel SVM. In *Proceedings of the IEEE International Conference on Data Mining*, 2009.