Machine Learning Software: Design and Practical Use

Chih-Jen Lin

National Taiwan University

eBay Research Labs

Talk at Machine Learning Summer School, Kyoto, August 31, 2012

Machine Learning Software

- Most machine learning works focus on developing algorithms
- Researchers didn't pay much attention to software
- Recently, some think software is important. For example, "The need for open source software in machine learning" by Sonnenburg et al. (2007)
- One reasons is for replicating and evaluating research results
- However, a good software package is beyond that

Machine Learning Software (Cont'd)

- In this talk, I will share our experiences in developing LIBSVM and LIBLINEAR.
- LIBSVM (Chang and Lin, 2011):

  One of the most popular SVM packages; cited more than $10,000$ times on Google Scholar
- LIBLINEAR (Fan et al., 2008):

  A library for large linear classification; popular in Internet companies for document classification and NLP applications

Machine Learning Software (Cont'd)

- This talk will contain two parts:
- First, we discuss practical use of SVM as an example to see how users apply a machine learning method
- Second, we discuss design considerations for a good machine learning package.
- The talk is biased toward SVM and logistic regression, but materials are useful for other machine learning methods.

Outline

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

Outline

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

Outline

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

Support Vector Classification

- Training data $(\mathbf{x}_i, y_i), i = 1, \ldots, l, \mathbf{x}_i \in R^n, y_i = \pm 1$
- Maximizing the margin (Boser et al., 1992; Cortes and Vapnik, 1995)

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l} \max(1 - y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b), 0)$$

- High dimensional (maybe infinite) feature space

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots).$$

- $\mathbf{w}$: maybe infinite variables

Support Vector Classification (Cont'd)

- The dual problem (finite # variables)

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

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

- At optimum

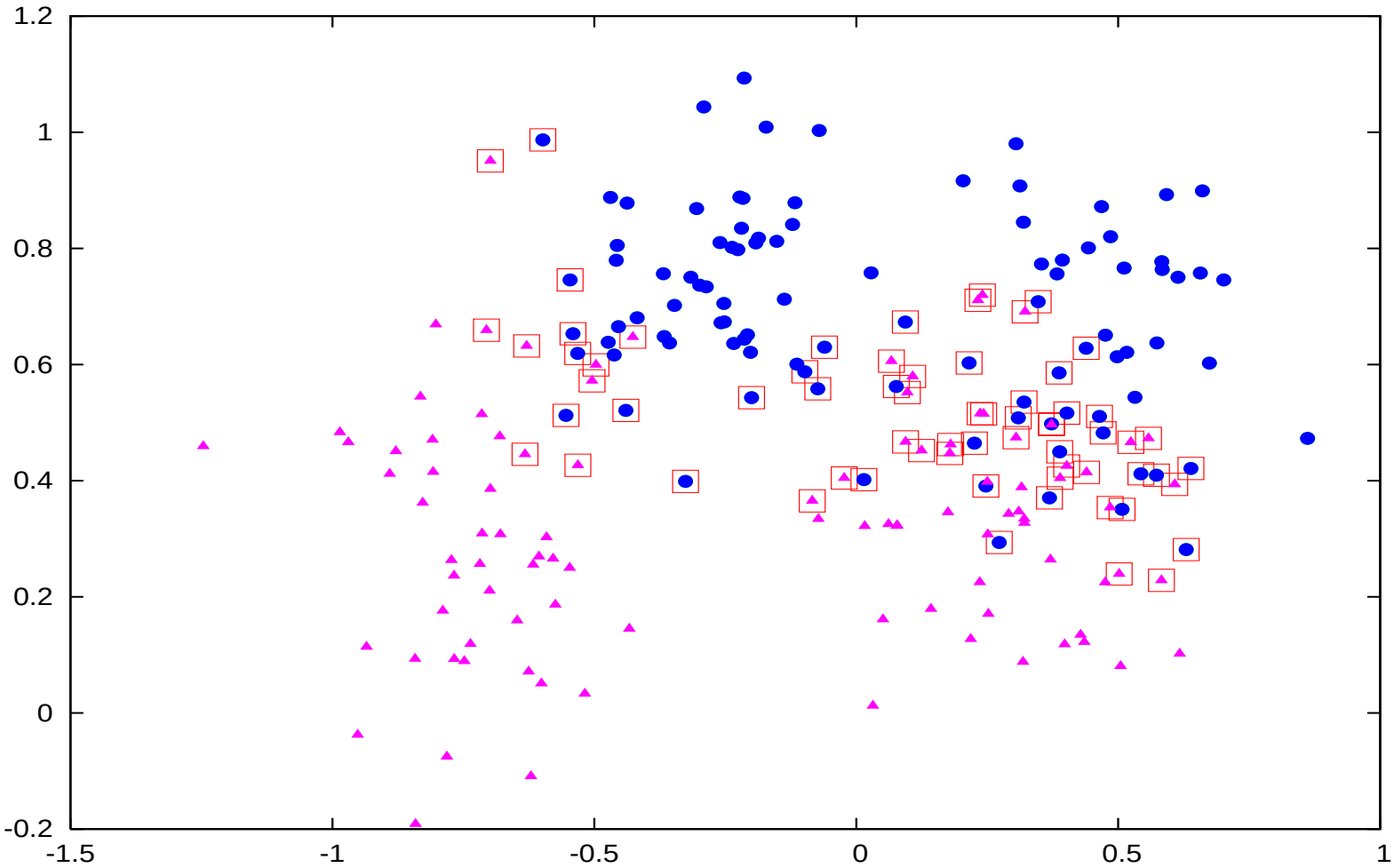
$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\mathbf{x}_i)$$

- Kernel: $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$; closed form

  Example: Gaussian (RBF) kernel: $e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$

Support Vector Classification (Cont'd)

Only $\mathbf{x}_i$ of $\alpha_i > 0$ used $\Rightarrow$ support vectors



Status of Support Vector Machines

- SVM was introduced with kernels
- But ironically, the linear setting also helps SVM to become popular
- By linear we mean data are not mapped to a higher dimensional space
- There are many differences between using kernel or not
- Many people are confused about this; so I will clarify the differences in the next few slides

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

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

  $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ easily calculated; little control on $\phi(\cdot)$
- Linear classification + feature engineering:

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

We refer to them as kernel and linear classifiers

Linear and Kernel Classification (Cont'd)

- Let's check the prediction cost

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

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

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

- Linear is much cheaper
- Deciding whether using kernel is cost-effective is still a big issue

Linear and Kernel Classification (Cont'd)

- For certain problems, accuracy by linear is as good as nonlinear

  But training and testing are much faster
- Especially document classification

  Number of features (bag-of-words model) very large
- Recently linear classification is a popular research topic. Sample works in 2005-2008: Joachims (2006); Shalev-Shwartz et al. (2007); Hsieh et al. (2008)

  They focus on large sparse data
- There are many other recent papers and software

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

Summary: Support Vector Machines

- The significant differences between linear and kernel motivated us to separately develop LIBSVM and LIBLINEAR
- For recent advances of linear classification, you may check a survey by Yuan et al. (2012)
- Today I will discuss only the situation of using kernels
- Another thing I didn't mention is online and offline training. We consider only offline training here

Summary: Support Vector Machines (Cont'd)

- So far I only gave a very brief introduction of support vector classification
- Today I am not giving a tutorial on SVM. For details of this method, you can check my other tutorial talks (e.g., the one at ACML 2010)

Outline

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

How People Use SVM?

- For most users, what they hope is
  1. Prepare training and testing sets
  2. Run a package and get good results
- But things are not that simple
- Let's start with a practical example from a user

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

The Story Behind this Data Set

- User:

  I am using libsvm in a astroparticle physics application .. First, let me congratulate you to a really easy to use and nice package. Unfortunately, it gives me astonishingly bad results...
- OK. Please send us your data
- I am able to get 97% test accuracy. Is that good enough for you ?
- User:

  You earned a copy of my PhD thesis

The Story Behind this Data Set (Cont'd)

What we have seen over the years is that

- Users expect good results right after using a method
- If method A doesn't work, they switch to B
- They may inappropriately use most methods they tried

But isn't it machine learning people's responsibility to make their methods easily give reasonable results?

The Story Behind this Data Set (Cont'd)

In my opinion

- Machine learning packages should provide some simple and automatic/semi-automatic settings for users

  These setting may not be the best, but easily give users some reasonable results
- If such settings are not enough, users many need to consult with machine learning experts.

I will illustrate the first point by a procedure we developed for SVM beginners

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

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

  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

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

  then

$$K = I$$

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

Overfitting

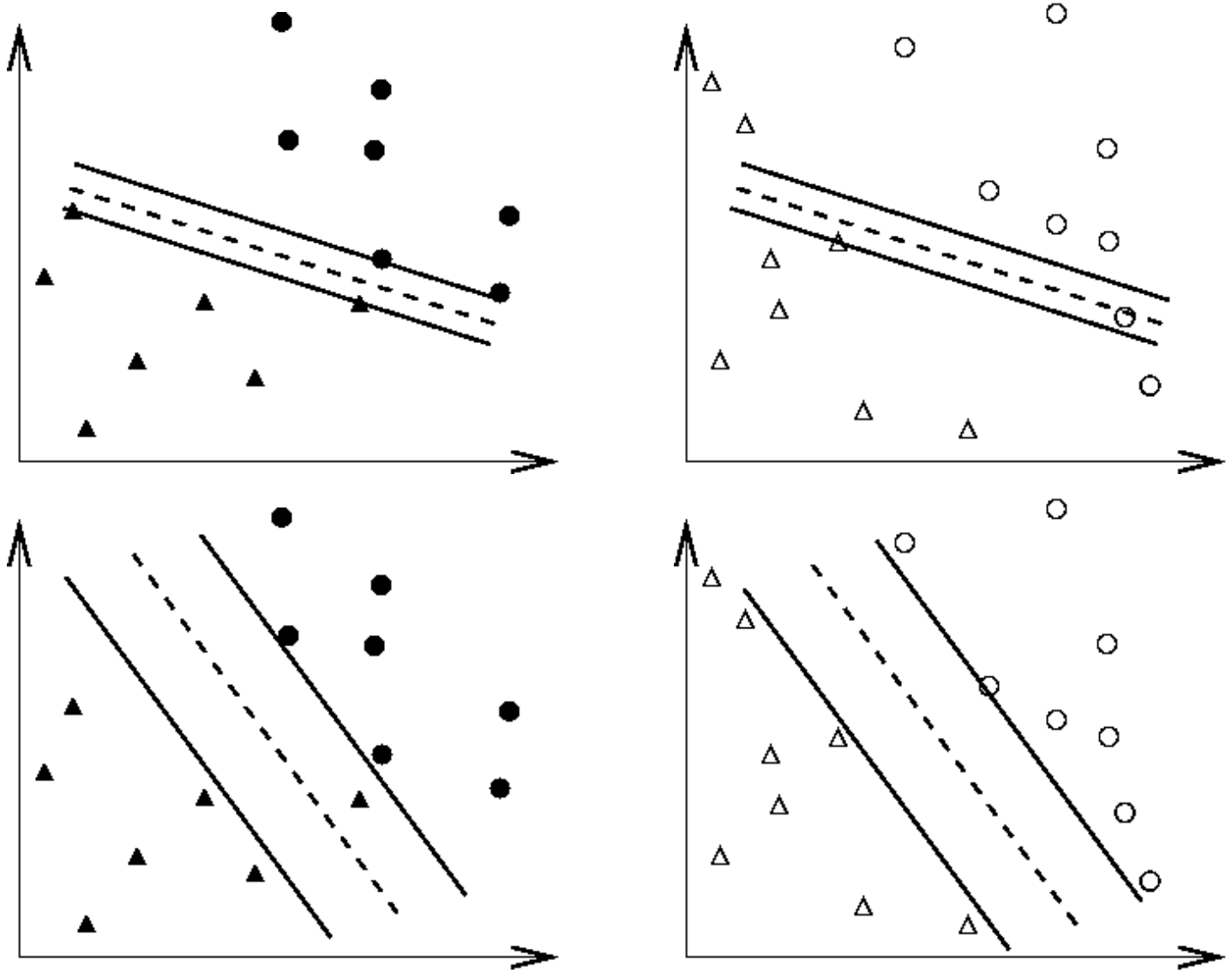
- See the illustration in the next slide
- In theory

  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.s
$./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.sc
$./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

Outline

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

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:

$$\gamma \text{ of } e^{-\gamma\|\mathbf{x}_i-\mathbf{x}_j\|^2}$$

$$a, b, d \text{ of } (\mathbf{x}_i^T\mathbf{x}_j/a + b)^d$$

- How to select them so performance is better?

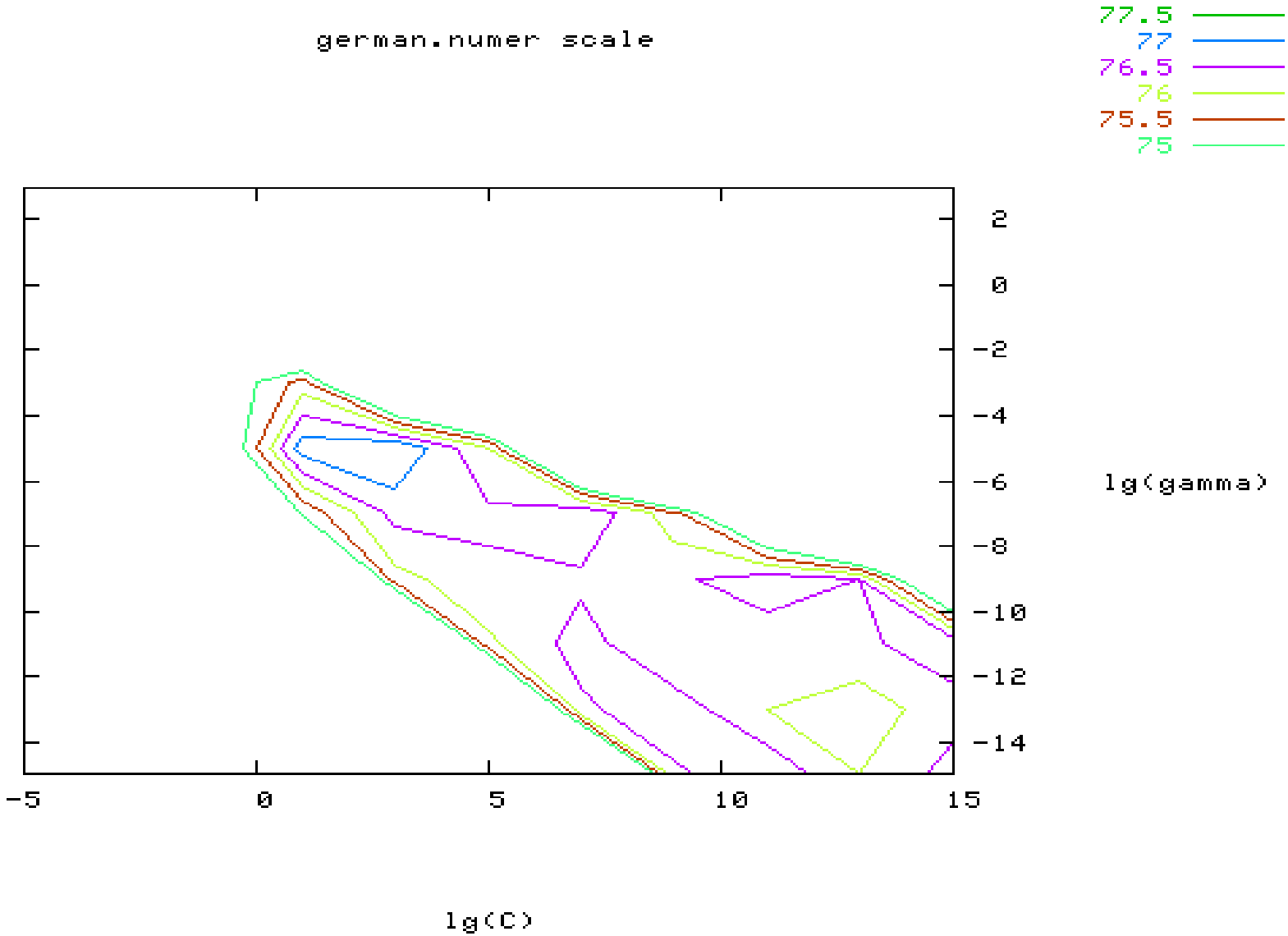
Performance Evaluation

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

→ 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:

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

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

  More parameters than RBF

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 $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
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 submiting 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.s
$ 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 > svmguide4
$ 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

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

Outline

1 Practical use of SVM
- SVM introduction
- A real example
- Parameter selection

2 Design of machine learning software
- Users and their needs
- Design considerations

3 Discussion and conclusions

Users' Machine Learning Knowledge

- When we started developing LIBSVM, we didn't know who our users are or whether we will get any
- Very soon we found that many users have zero machine learning knowledge
- It is unbelievable that many asked what the difference between training and testing is

Users' Machine Learning Knowledge (Cont'd)

- A sample mail

```
From:
To: cjlin@csie.ntu.edu.tw
Subject: Doubt regarding SVM
Date: Sun, 18 Jun 2006 10:04:01

Dear Sir,
        sir what is the difference between
testing data and training data?
```

- Sometimes we cannot do much for such users.

Users' Machine Learning Knowledge (Cont'd)

- Fortunately, more people have taken machine learning courses (or attended MLSS)
- On the other hand, because users are not machine learning researchers, some automatic or semi-automatic settings are helpful
- The simple procedure discussed earlier is an example
- Also, your target users affect your design.

  For example, we assume LIBLINEAR users are more experienced.

We are Our Own Users

- You may ask why we care non-machine learning users so much
- The reason is that we were among them before
- My background is in optimization. When we started working on SVM, we tried some UCI sets.
- We failed to obtain similar accuracy values in papers
- Through a painful process we learned that scaling may be needed

We are Our Own Users (Cont'd)

- Machine learning researchers sometimes failed to see the difficulties of general users.
- As users of our own software, we constantly think about difficulties others may face

Users are Our Teachers

- While we criticize users' lack of machine learning knowledge, they help to point out many useful directions
- Example: LIBSVM supported only binary classification in the beginning. From many users' requests, we knew the importance of multi-class classification
- There are many possible approaches for multi-class SVM. Assume data are in k classes

Users are Our Teachers (Cont'd)

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

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

- One-against-one: train $k(k-1)/2$ binary SVMs

$$(1,2), (1,3), \ldots, (1,k), (2,3), (2,4), \ldots, (k-1,k)$$

- We finished a study in Hsu and Lin (2002), which is now well cited.
- Currently LIBSVM supports one-vs-one approach

Users are Our Teachers (Cont'd)

- LIBSVM is among the first SVM software to handle multi-class data.

  This helps to attract many users.
- Users help to identify useful things for the software
- They even help to identify important research directions
- The paper (Hsu and Lin, 2002) was rejected by many places because it's a detailed comparison without new algorithms
- But from users we knew its results are important. This paper is now one of my most cited papers

Outline

1. Practical use of SVM
   - SVM introduction
   - A real example
   - Parameter selection
2. Design of machine learning software
   - Users and their needs
   - Design considerations
3. Discussion and conclusions

One or Many Options

- Sometimes we received the following requests

  1. In addition to "one-vs-one," could you include other multi-class approaches such as "one-vs-the rest?"

  2. Could you extend LIBSVM to support other kernels such as χ^2 kernel?

- Two extremes in designing a software package

  1. One option: reasonably good for most cases

  2. Many options: users try options to get best results

One or Many Options (Cont'd)

- From a research viewpoint, we should include everything, so users can play with them
- But

$$\text{more options} \Rightarrow \text{more powerful}$$
$$\Rightarrow \text{more complicated}$$

- Some users have no abilities to choose between options

  Example: Some need χ^2 kernel, but some have no idea what it is

One or Many Options (Cont'd)

- Users often try all options even if that's not needed
- Example: LIBLINEAR has the following solvers

```
options:
-s type : set type of solver (default 1)
0 -- L2-regularized logistic regression (primal)
...
7 -- L2-regularized logistic regression (dual)
```

- Some users told me:
  I have tried all solvers, but accuracy is similar
- But wait, doesn't solvers 0 and 7 always give same accuracy? No need to run both

One or Many Options (Cont'd)

- For LIBSVM, we basically took the "one option" approach

  We are very careful in adding things to LIBSVM
- However, users do have different needs. For example, some need precision/recall rather than accuracy
- We end up with developing another web site "LIBSVM Tools" to serve users' special needs

One or Many Options (Cont'd)

- Sample code in LIBSVM tools

  - Cross Validation with Different Criteria (AUC, F-score, etc.)

  - ROC Curve for Binary SVM

  - LIBSVM for string data
- Not sure if this is the best way, but seems ok so far
- Another advantage is we can maintain high quality for the core package. Things in LIBSVM Tools are less well maintained.

Simplicity versus Better Performance

- This issue is related to "one or many options" discussed before
- Example: Before, our cross validation (CV) procedure is not stratified

  - Results less stable because data of each class not evenly distributed to folds

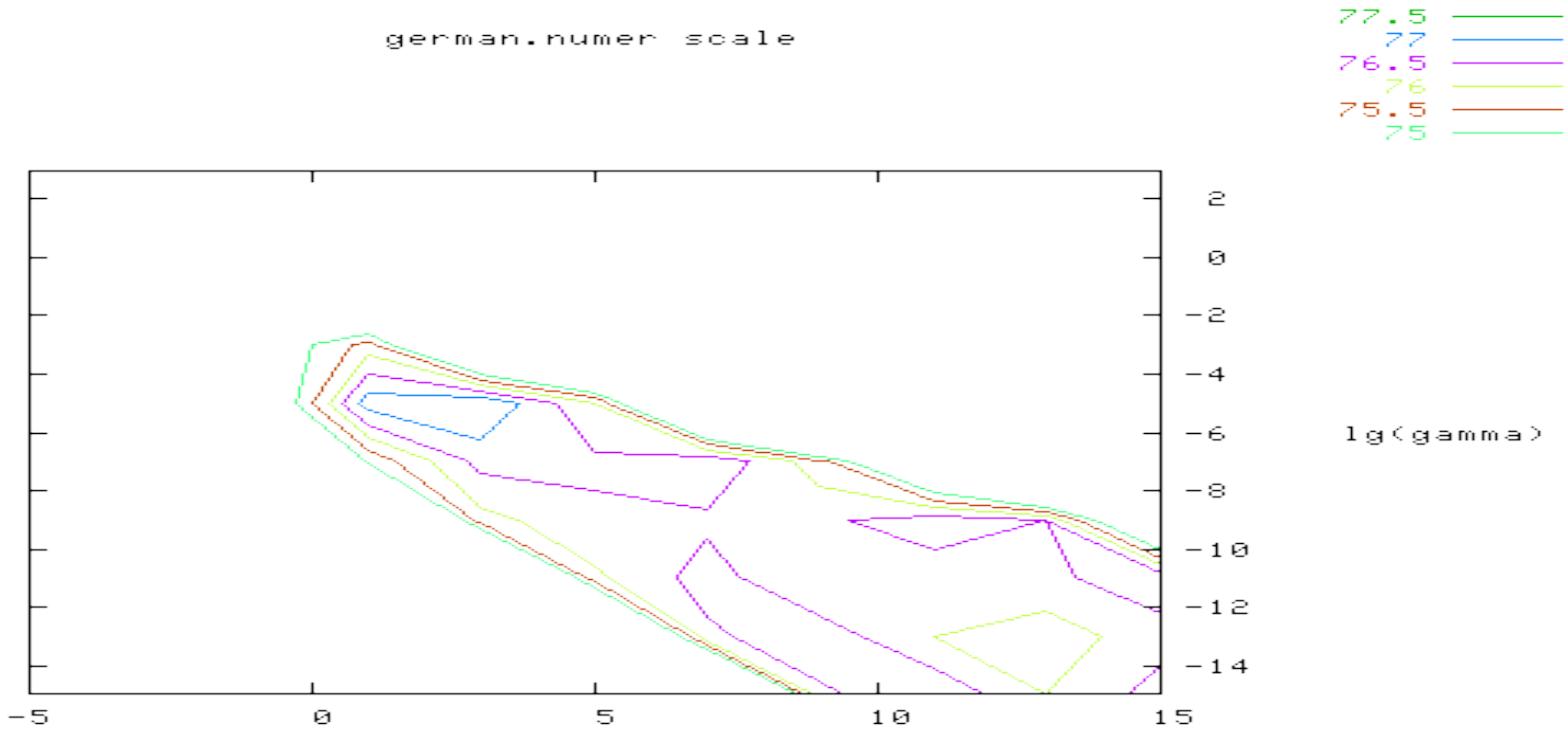
  - We now support stratified CV, but code becomes more complicated
- In general, we avoid changes for just marginal improvements

Simplicity versus Better Performance (Cont'd)

- A recent Google research blog "Lessons learned developing a practical large scale machine learning system" by Simon Tong
- From the blog, "It is perhaps less academically interesting to design an algorithm that is slightly worse in accuracy, but that has greater ease of use and system reliability. However, in our experience, it is very valuable in practice."
- That is, a complicated method with a slightly higher accuracy may not be useful in practice

Simplicity versus Better Performance (Cont'd)

Example: LIBSVM uses a grid search to find two parameters C and γ. We may think this is simple and naive



Simplicity versus Better Performance (Cont'd)

- Indeed, we studied loo bound in detail. That is, a function of parameters $f(C, \gamma)$ is derived to satisfy

$$\text{leave-one-out error} \leq f(C, \gamma)$$

  Then we solve

$$\min_{C,\gamma} f(C, \gamma)$$

- Results not very stable because $f(C, \gamma)$ is only an approximation. Implementation is quite complicated.
- For only two parameters, a simple grid search may be a suitable choice

Numerical and Optimization Methods

- Many classification methods involve numerical and optimization procedures
- A key point is that we need to take machine learning properties into the design of your implementation
- An example in Lieven's talk yesterday: L1-regularized least-square regression
- I will discuss some SVM examples

Numerical and Optimization Methods (Cont'd)

- Let's consider SVM dual

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

- $Q_{ij} \neq 0$, 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
- Traditional optimization methods:
  Newton or gradient cannot be directly applied

Numerical and Optimization Methods (Cont'd)

- One workaround is to work on some variables each time (e.g., Osuna et al., 1997; Joachims, 1998; Platt, 1998)
- Working set B, $N = \{1, \ldots, l\} \backslash B$ fixed
- Sub-problem at the kth iteration:

$$\min_{\boldsymbol{\alpha}_B} \quad \frac{1}{2} \begin{bmatrix} \boldsymbol{\alpha}_B^T & (\boldsymbol{\alpha}_N^k)^T \end{bmatrix} \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N^k \end{bmatrix} - \begin{bmatrix} \mathbf{e}_B^T & (\mathbf{e}_N^k)^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N^k \end{bmatrix}$$

$$\text{subject to} \quad 0 \le \alpha_t \le C, t \in B, \ \mathbf{y}_B^T \boldsymbol{\alpha}_B = -\mathbf{y}_N^T \boldsymbol{\alpha}_N^k$$

Numerical and Optimization Methods (Cont'd)

- The new objective function

$$\frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B + (-\mathbf{e}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T\boldsymbol{\alpha}_B + \text{ constant}$$

- Only B columns of Q needed ($|B| \geq 2$)
- Calculated when used

  Trade time for space
- But is such an approach practical?

Numerical and Optimization Methods (Cont'd)

- Convergence not very fast
- But, no need to have very accurate $\boldsymbol{\alpha}$

$$\text{decision function: } \sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

  Prediction may still be correct with a rough $\boldsymbol{\alpha}$
- Further, in some situations, # support vectors $\ll$ # training points
  Initial $\boldsymbol{\alpha}^1 = 0$, some instances never used
- So special properties of SVM did contribute to the viability of this method

Numerical and Optimization Methods (Cont'd)

- An example of training 50,000 instances using LIBSVM

```
$svm-train -c 16 -g 4 -m 400 22features
Total nSV = 3370
Time 79.524s
```

- On a Xeon 2.0G machine
- Calculating the whole Q takes more time
- #SVs $= 3{,}370 \ll 50{,}000$

  A good case where some remain at zero all the time

Numerical and Optimization Methods (Cont'd)

- Another example: training kernel and linear SVM should be done by different methods
- What's the difference between kernel and linear?

  for linear, K can be written as XX^T

  where

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_l^T \end{bmatrix} \in R^{l \times n}$$

  is the training matrix
- Note that n: # features, l: # data

Numerical and Optimization Methods (Cont'd)

- Recall in the kernel case, we need

$$(Q\boldsymbol{\alpha})_i - 1, i \in B$$

- The cost is $O(nl)$

$$\sum_{j=1}^{l} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\alpha_j - 1$$

Numerical and Optimization Methods (Cont'd)

- For linear, if we have

$$\mathbf{w} \equiv \sum_{j=1}^{l} y_j \alpha_j \mathbf{x}_j$$

then

$$\sum_{j=1}^{l} y_i y_j \mathbf{x}_i^T \mathbf{x}_j \alpha_j - 1 = y_i \mathbf{w}^T \mathbf{x}_i - 1$$

costs only $O(n)$. Details of maintaining $\mathbf{w}$ are not discussed here; the cost is also $O(n)$

- Again, the key is that properties of machine learning problems should be considered

Numerical Stability

- Quality of the numerical programs is important for a machine learning package
- Numerical analysts have a high standard on their code, but unfortunately we machine learning people do not
- This situation is expected:
  If we put efforts on implementing method A and one day method B gives higher accuracy $\Rightarrow$ Efforts are wasted

Numerical Stability (Cont'd)

- We will give an example to discuss how to improve numerical stability in your implementation
- In LIBSVM's probability outputs, we calculate

$$-\left(t_i \log(p_i) + (1 - t_i) \log(1 - p_i)\right),$$

where

$$p_i \equiv \frac{1}{1 + \exp(\Delta)}$$

- When Δ is small, $p_i \approx 1$
- Then $1 - p_i$ is a catastrophic cancellation

Numerical Stability (Cont'd)

- Catastrophic cancellation (Goldberg, 1991): when subtracting two nearby numbers, the relative error can be large so most digits are meaningless.
- In a simple C++ program with double precision,

$$\Delta = -64 \quad \Rightarrow \quad 1 - \frac{1}{1 + \exp(\Delta)} \text{ returns zero}$$

but

$$\frac{\exp(\Delta)}{1 + \exp(\Delta)} \text{ gives more accurate result}$$

Numerical Stability (Cont'd)

- Catastrophic cancellation may be resolved by reformulation
- Another issue is that log and exp could easily cause an overflow

$$\text{If } \Delta \text{ is large } \Rightarrow \exp(\Delta) \to \infty$$

- Then

$$p_i = \frac{1}{1+\exp(\Delta)} \approx 0 \quad \Rightarrow \quad \log(p_i) \to -\infty$$

- We can use the following reformulation

$$\begin{aligned} & \log(1 + \exp(\Delta)) \\ = \ & \log(\exp(\Delta) \cdot (\exp(-\Delta) + 1)) \\ = \ & \Delta + \log(1 + \exp(-\Delta)) \end{aligned}$$

- When Δ is large

$$\exp(-\Delta) \approx 0 \text{ and } \log(1 + \exp(\Delta)) \approx \Delta$$

- Less like to get overflow because of no $\exp(\Delta)$

Numerical Stability (Cont'd)

- In summary,

$$
\begin{aligned}
& -(t_i \log p_i + (1 - t_i) \log(1 - p_i)) && (1)\\
= & (t_i - 1)\Delta + \log(1 + \exp(\Delta)) && (2)\\
= & \, t_i(\Delta) + \log(1 + \exp(-\Delta)) && (3)
\end{aligned}
$$

- We implement (1) with the following rule:

$$\text{If } \Delta \geq 0 \text{ then use (3); Else use (2).}$$

- This handles both issues of overflow and catastrophic cancellation

Legacy Issues

- The compatibility between earlier and later versions is an issue
- Such legacy issues restrict developers to conduct certain changes.
- We face a similar situation. For example, we chose "one-vs-one" as the multi-class strategy. This decision affects subsequent buildups.
- Example: in LIBSVM, multi-class probability outputs must follow the one-vs-one structure. For classes i and j, we obtain

$$P(\mathbf{x} \text{ in class } i \mid \mathbf{x} \text{ in class } i \text{ or } j),$$

Legacy Issues (Cont'd)

- Then we need to couple all $\binom{k}{2}$ results (k: the number of classes) and obtain

$$P(\mathbf{x} \text{ in class } i), i = 1, \ldots, k.$$

- If we further develop multi-label methods, we are restricted to extend from one-versus-one multi-class strategy
- What if one day we would like to use a different multi-class method?

Legacy Issues (Cont'd)

- In LIBSVM, we understand this legacy issue in the beginning
- Example: we did not make the trained model a public structure

  Encapsulation in object-oriented programming
- Here is the C code to train and test

```
#include <svm.h>
...
model = svm_train(...);
...
predict_label = svm_predict(model,x);
```

- `svm.h` includes all public functions and structures

Legacy Issues (Cont'd)

- We decided not to put model structure in `svm.h`
  Instead we put it in `svm.cpp`
- User can call

```
model = svm_train(...);
```

  but cannot directly access a model's contents

```
int y1 = model.label[1];
```

- We provide functions so users can get some model information

```
svm_get_svm_type(model);
svm_get_nr_class(model);
svm_get_labels(model, ...);
```

Documentation and Support

- Any software needs good documents and support
- I cannot count how many mails my students and I replied. Maybe 20,000 or more.
- How to write good documents is an interesting issue

  Users may not understand what you wrote

Documentation and Support (Cont'd)

- Here is an example: some users asked if LIBSVM supported multi-class classification
- I thought it's well documented in `README`
- Finally I realized that users may not read the whole README.
- Instead, some of them check only the "usage"

Documentation and Support (Cont'd)

- They didn't see "multi-class" in the usage

```
$ svm-train

Usage: svm-train [options] training_set_file
options:
-s svm_type : set type of SVM (default 0)
   0 -- C-SVC
   1 -- nu-SVC
   2 -- one-class SVM
   3 -- epsilon-SVR
   4 -- nu-SVR
...
```

Documentation and Support (Cont'd)

- In the next version we will change the usage to

```
-s svm_type : set type of SVM (default 0)
    0 -- C-SVC          (multi-class classificati
    1 -- nu-SVC         (multi-class classificati
    2 -- one-class SVM
    3 -- epsilon-SV (regression)
    4 -- nu-SVR         (regression)
```

- I am going to see how many asked "why LIBSVM doesn't support two-class SVM"

Outline

1 Practical use of SVM
- SVM introduction
- A real example
- Parameter selection

2 Design of machine learning software
- Users and their needs
- Design considerations

3 Discussion and conclusions

Software versus Experiment Code

- Many researchers now release experiment code used for their papers

  Reason: experiments can be reproduced
- This is important, but experiment code is different from software
- Experiment code often includes messy scripts for various settings in the paper – useful for reviewers

  Example: to check an implementation trick in a proposed algorithm, need to run with/without the trick

Software versus Experiment Code (Cont'd)

- Software: for general users

  One or a few reasonable settings with a suitable interface are enough
- Many are now willing to release their experimental code

  Basically you clean up the code after finishing a paper
- But working on and maintaining high-quality software take much more work

Software versus Experiment Code (Cont'd)

- Reproducibility different from replicability (Drummond, 2009)

  Replicability: make sure things work on the sets used in the paper

  Reproducibility: ensure that things work in general
- In my group, we release experiment code for every paper $\Rightarrow$ for replicability

  And carefully select and modify some results to our software $\Rightarrow$ (hopefully) for reproducibility

Software versus Experiment Code (Cont'd)

- The community now lacks incentives for researchers to work on high quality software
- JMLR recently started "open source software" section (4-page description of the software)
- This is a positive direction
- How to properly evaluate such papers is an issue
- Some software are very specific on a small problem, but some are more general

Research versus Software Development

Shouldn't software be developed by companies?

Two issues

1. Business models of machine learning software
2. Research problems in developing software

Research versus Software Development (Cont'd)

Business model

- It is unclear to me what a good model should be
- Machine learning software are basically "research" software
- They are often called by some bigger packages
  For example, LIBSVM and LIBLINEAR are called by Weka and Rapidminer through interfaces
- These data mining packages are open sourced and their business is mainly on consulting
- Should we on the machine learning side use a similar way?

Research versus Software Development (Cont'd)

Research issues

- A good machine learning package involves more than the core machine learning algorithms
- There are many other research issues
  - Numerical algorithms and their stability
  - Parameter tuning, feature generation, and user interfaces
  - Serious comparisons and system issues
- These issues need researchers rather than engineers
- Currently we lack a system to encourage machine learning researchers to study these issues.

Machine Learning and Its Practical Use

- I just mentioned that a machine learning package involves more than algorithms
- Here is an interesting conversation between me and a friend

  Me: Your company has system A for large-scale training and now also has system B. What are their differences?

  My friend: A uses ... algorithms and B uses ...

  Me: But these algorithms are related

  My friend: Yes, but you know sometimes algorithms are the least important thing

Machine Learning and Its Practical Use (Cont'd)

- As machine learning researchers, of course we think algorithms are important
- But we must realize that in industry machine learning is often only part of a big project
- Moreover, it rarely is the most important component
- We academic machine learning people must try to know where machine learning is useful. Then we can develop better algorithms and software

Conclusions

- From my experience, developing machine learning software is very interesting.

  In particular, you get to know how your research work is used
- Through software, we have seen different application areas of machine learning
- We should encourage more researchers to develop high quality machine learning software

Acknowledgments

- All users have greatly helped us to make improvements

  Without them we cannot get this far
- We also thank all our past group members

References I

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.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

C. Drummond. Replicability is not reproducibility: Nor is it good science. In *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*, 2009.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

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`.

References II

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. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 169–184, Cambridge, MA, 1998. MIT Press.

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.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 130–136, 1997.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

References III

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, M. Braun, C. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K. Müller, F. Pereira, C. Rasmussen, et al. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent advances of large-scale linear classification. *Proceedings of IEEE*, 2012. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/survey-linear.pdf`. To appear.