SVM Practical Session

(How to get good results without cheating)

Jason Weston, Arthur Gretton and Andre Elisseeff
Department of Biological Cybernetics
Max-Planck Institute, Tübingen, Germany.
weston@tuebingen.mpg.de
Getting started

In this session we will be looking at how to get good experimental results (that don’t involve cheating!) with support vector machines.

We will look at three key aspects:

- Understanding SVMs to get good results (tricks of the trade).
- Special kernels for special problems (encoding prior knowledge).
- Proper experimental design (to avoid cheating/biased results).
Matlab and the Spider Toolbox

We will use matlab and a toolbox of machine learning methods called Spider (freely available at http://www.kyb.tuebingen.mpg.de/bs/people/spider/).

Run matlab and type:

```matlab
X=randn(5)
Y=[1 1 1 -1 -1]’
d=data(X,Y) % make data
a=svm;
[tr,a]=train(a,d) % train svm
```

You’ve now trained an SVM on a dataset!
Spider II

Actually you can use a lot of different algorithms in the same way, e.g replace svm with knn in the code from before. Type ”help spider” for a list, and e.g. ”help svm”.

After training one can use ”loss” to calculate the error rate.

```
load colon.mat % colon cancer microarray data
d1=data(X(1:30,:),Y(1:30));
d2=data(X(31:end,:),Y(31:end));

[tr a]=train(svm,d1); % train on one part
r=test(a,d2); % test on another

loss(tr) % calculate train
loss(r) % & test error
```
**Spider III**

The output should look something like this:

```plaintext
>> ...  
>> loss(tr)  
data -> svm kernel linear -> class_loss=0  
>> loss(r)  
data -> svm kernel linear -> class_loss=0.1562
```

This measures the error rate between the estimate \( r.X \) and the truth \( r.Y \).

You can type `help spider` to see what other things spider can do.

... Now we’re ready to do start the SVM session proper ...
SVM Practical Session I

Tricks of the Trade
Outline of this section

1. Understanding parameters: how $\sigma$, $C$ and $\alpha$ work.
3. Understanding the kernel matrix.
4. Preprocessing data (scaling)
5. When SVMs fail: noisy data.
6. Understanding the regularizer ($L_1$ versus $L_2$ norm.)
**Remembering SVM**

SVM: separate with maximal margin in high dimensional space defined by kernel:

Minimize

\[ ||w||^2 + C \sum_i \xi_i \]

subject to

\[ y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \]

Dual form: maximize

\[ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j), \quad \text{s.t.} \ 0 \leq \alpha_i \leq C \text{ and } \sum_i \alpha_i y_i = 0 \]

They can be linked with \( w = \sum \alpha_i y_i \phi(x_i) \).

RBF kernel: \( K(x_i, x_j) = (\phi(x_i) \cdot \phi(x_j)) = \exp(-||x_i - x_j||^2/2\sigma^2) \)
Understanding parameters: $\sigma$ and $\alpha$

\[ w = \sum_i \alpha_i y_i \phi(x_i). \]

RBF kernel: \[ K(x_i, x_j) = \exp(-||x_i - x_j||^2 / 2\sigma^2) \]

Decision rule: \[ f(x) = \sum \alpha_i y_i K(x_i, x) + b \]

This amounts to putting bumps of various sizes on the training set:

Let’s try to understand what $\sigma$ does with some examples.
Understanding parameters II: $\sigma$ and $\alpha$

- Make a dataset with two overlapping Gaussians belonging to two classes (20 pts each) using the `randn` function with means: (-1,-1) and (1,1) and standard deviation 1.

- Train an SVM with an RBF kernel on this dataset: use
  
  ```
  a=svm(kernel(‘rbf’,sigma_value))
  ```

- Plot the results with the `plot(a)` function.

Now look at the different results for different values of $\sigma$.

E.g. try $\sigma = 0.1, 1$ and 10.

You can look at the alphas using `a.alpha`
Understanding parameters I: code

```matlab
randn('seed',1);

m=20; d=1; s=1; % make data
x1=[randn(m,1)*s-d randn(m,1)*s-d]
x2=[randn(m,1)*s+d randn(m,1)*s+d]
d=data([x1;x2],[ones(m,1); -ones(m,1)]);

a=svm; sigma=10; % train svm
a.child=kernel('rbf',sigma);
[r a]=train(a,d);

plot(a) % plot results
```
Parameters: $\sigma = 0.1, 0.25, 0.5, 0.75, 1, 10$
Understanding parameters II: $C$ and $\xi$

- Now change the standard deviation $\sigma^*$ of the two Gaussians to make the datasets more overlapping, e.g. $\sigma^* = 2$.
- Use an svm with an RBF kernel again, with $\sigma = 1$ and $\sigma = 10$.
- Now look at the different results for different amounts of overlap and different values of $C$ (e.g. use a.C=1).

$$a = \text{svm(kernel('rbf', sigma_value))}; a.C=1$$

- Try to find the best $C'$ (e.g. try $C' = 1, 10, 100$).

*Also note how $\xi_i$ changes: see r.X and r.Y in results object.*
Understanding parameters II: code

```matlab
randn('seed',1);

m=20; d=1; s=2; % make data
x1=[randn(m,1)*s-d randn(m,1)*s-d]
x2=[randn(m,1)*s+d randn(m,1)*s+d]
d=data([x1;x2],[ones(m,1);-ones(m,1)]);

a=svm; sigma=10; a.C=1; % train svm
a.child=kernel('rbf',sigma);
[r a]=train(a,d);

plot(a) % plot results
```
Parameters: $\sigma^* = 2, \sigma = 1, \ C = \text{Inf, 100, 10, 1}$
\( \sigma^* = 2, \sigma = 10, C = 100, 10, 1 \) and \( \sigma = 100, C = 1 \)
Heuristics for finding parameters

First heuristics for finding $C$ and $\sigma$ (motivation: faster than CV):

- divide all $d$ components by their standard deviation
- take RBF kernel, $\sigma^2 = d$ (value exponentiated of order 1)
- Take $C = 1$.

Alternative heuristic for $\sigma$: Take $\sigma =$ median value of smallest distances between all +ve points and opposite class.
Heuristics for finding parameters II

Heuristic $\sigma$ and $C$ for dataset with std.dev. $\sigma^* = 1, 2$. 
Heuristics for finding parameters III

Issues:

- $C$ and $\sigma$ are on an exponential scale
- If data multiplied by 10, $C$ divided by 100 (otherwise invariant)
- Too small $C$ doesn’t work, too large $C$ might not work (stalls)
- Can use ridge instead $K \leftarrow K + \lambda I$ equivalent to:
  
  Minimize
  
  \[ ||w||^2 + \frac{1}{2\lambda} \sum \xi_i^2 \]

  subject to
  
  \[ y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \]

- Can optimize the bias ($b$) to minimize training error: can improve results as only the 1- or 2-norm of slacks ($\xi_i$) was minimized before.
Understanding the kernel matrix

- Look at submatrix of kernel when choosing $\sigma$:

```matlab
load simple_data; d=data(X,Y);  % load data
a=kernel('rbf',10); k=calc(a,d);  % calc kernel
y1=find(d.Y==1);  y2=find(d.Y==-1);
imagesc(k([y1;y2],[y1;y2])); colorbar;  % view kernel
k([y1(1:3) y2(1:3)],[y1(1:3) y2(1:3)])
```

- Try the same as above for $\sigma = 0.1, 1, 10$.
- If matrix is all zeros, or all ones $\rightarrow$ may be no good?
- Always try to visualize data to see patterns, & as a sanity check.
Understanding the kernel matrix (picture) $x_t$
The distance measure & Prior knowledge

For any learning algorithm the distance measure is important: it should encode that $x$ and $x^*$ are close if they have the same label $\rightarrow$ depends on the target.

E.g given handwritten text: predicting its category or its author require different measures.

Different ways to give prior knowledge about relationship between $x$ and $y$ to an algorithm:

- scaling (pre-processing)
- choosing a kernel
- choosing a regularizer
Preprocessing data

SVMs are not scale insensitive: minimize $||w||^2$ → feature that has large values tends to be preferred as requires small $w_i$.

Some methods of scaling:

- Mean 0, Std 1: good when attributes are differently scaled (age, weight, height), uninformative prior (equal weight).

- By correlation score: after normalizing above, scale feature $i$ by

$$f_i = \frac{(\mu_i^+ - \mu_i^-)^2}{(\sigma_i^+)^2 + (\sigma_i^-)^2}$$

- Using a sigmoid to make robust outliers: $x_d \leftarrow \tanh(s \times x_d)$

- Normalizing the kernel in feature space: $K_{ij} \leftarrow K_{ij} / \sqrt{K_{ii}K_{jj}}$
  - equivalent to scaling to unit length
Choosing the kernel & regularizer

Later in the session we will look at special kernels that encode prior knowledge to improve generalization.

The regularizer (e.g. $||w||^2$) can also encode knowledge.

- Importance of features: trivally add weights $f$ to regularizer: 
  $$\min \sum_i f_i w_i^2.$$  
- One can choose different regularizers, e.g. $||w||_1 = \sum_i |w_i|$.  

It turns out that SVMs are good because the 2-norm is a ”safe bet when you dont know anything” → leaves encoding task for kernel.
Comparing the 1- and 2- norms

• Make a dataset with two 200-dimensional overlapping Gaussians belonging to two classes using the `randn` function with means: (-1,-1,..-1) and (1,1..1) and standard deviation 3.

• Take 5 training sets of 20 pts, and 200 test points.

• Train a linear SVM (2-norm) and the 1-norm on this dataset:
  use

  a1=svm;      % 2-norm minimization
  a2=l1;       % 1-norm minimization

• Calculate the mean error over the 5 trials.
1- and 2- norm: code

```matlab
res=[];
for loops=1:5
        m=10; m2=100; d=1; s=3;
        x1=[]; x2=[]; xt1=[]; xt2=[];

    for i=1:200 % make data
        x1=[x1 randn(m,1)*s-d]; x2=[x2 randn(m,1)*s+d];
        xt1=[xt1 randn(m2,1)*s-d]; xt2=[xt2 randn(m2,1)*s+d];
    end

d=data([x1;x2],[ones(m,1) ; -ones(m,1)]);
dtst=data([xt1;xt2],[ones(m2,1) ; -ones(m2,1)]);
a=svm; [tr a]=train(a,d); [r]=test(a,dtst);
res1=loss(r); % train 2-norm

    a=l1; [tr a]=train(a,d); [r]=test(a,dtst);
res2=loss(r); % train 1-norm

res=[res ; res1.Y res2.Y]; % remember results
end
mean(res)
```
When SVMs succeed

SVMs succeed when there is a large number of weakly correlated features.

The results for the last experiment look like this:

<table>
<thead>
<tr>
<th>classifier</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$|w|_2$</td>
<td>0</td>
</tr>
<tr>
<td>$|w|_1$</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Why are SVMs better?

Because $\|w\|_1$ selects a few features, and ignores the others which were also relevant.
Comparing the 1- and 2- norms, part II

- Now make the same dataset, but change the standard deviation in each dimension: make the first $n$ features have $\sigma = 0.5$ and the rest have $\sigma = 20$.

- Compare l1 and svm in the same way as before for $n = 1, 5, 10, 20$.

- After that try repeating the experiment first scaling the data to have Mean 0, Std 1. Use:
  
  ```matlab
  [d,a]=train(normalize,d);  dtst=test(a,dtst);
  ```

- Then try by scaling with the correlation coefficients:
  
  ```matlab
  [d,a]=train(normalize(‘scale_type=4’),d);  dtst= ...
  ```
### When SVMs fail

Without scaling:

<table>
<thead>
<tr>
<th>classifier</th>
<th>n=1</th>
<th>n=5</th>
<th>n=10</th>
<th>n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td></td>
<td>w</td>
<td></td>
<td>_2^2$</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>w</td>
<td></td>
<td>_1$</td>
</tr>
</tbody>
</table>

With Mean 0 Std 1:

<table>
<thead>
<tr>
<th>classifier</th>
<th>n=1</th>
<th>n=5</th>
<th>n=10</th>
<th>n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td></td>
<td>w</td>
<td></td>
<td>_2^2$</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>w</td>
<td></td>
<td>_1$</td>
</tr>
</tbody>
</table>

**CONCLUSION:** *Normalization stops the noisy features dominating, then the 1-norm then selects correct features.*
When SVMs.. don’t fail (again)

With correlation coefficient scaling:

<table>
<thead>
<tr>
<th>classifier</th>
<th>n=1</th>
<th>n=5</th>
<th>n=10</th>
<th>n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td></td>
<td>w</td>
<td></td>
<td>^2_2$</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>w</td>
<td></td>
<td>_1$</td>
</tr>
</tbody>
</table>

Now both methods perform well.

BONUS PROBLEM: design an algorithm that switches between 1- and 2- norm automatically.
SVM Practical Session II
Designing Special Kernels
Using invariances

If you know prior information about your problem, you should use this to improve performance. Especially good when you don’t have much data.
Kernel design with invariance: problem

Can you design a kernel to take into account the (assumed) invariance in the previous classification problem?

- Write a function to compute an inner product matrix:
  \[
  [K]=\text{invar\_dot(pattern1, pattern2)};
  \]
  where the 2-D input vectors are \( \mathbf{x} = [u \; v] \), and thus \( \text{pattern1, pattern2} \)
  are matrices (maybe not of the same length), of the form
  \[
  \begin{bmatrix}
  \mathbf{x}_1 \\
  \mathbf{x}_2 \\
  \vdots
  \end{bmatrix}
  =
  \begin{bmatrix}
  u_1 & v_1 \\
  u_2 & v_2 \\
  \vdots & \vdots
  \end{bmatrix}
  \]

- To check your function, run the code
  \text{invariant.m}
Kernel design with invariance: results

The results you get should look like this:
Kernel design with invariance: code

Here is some sample code to achieve this invariance

```matlab
function [K]=invar_dot(patterns1,patterns2);

features1 = sqrt(patterns1(:,1).^2 + patterns1(:,2).^2);
features2 = sqrt(patterns2(:,1).^2 + patterns2(:,2).^2);

size1=length(features1);
size2=length(features2);

K=zeros(size1,size2);
for k=1:size1(1)
    for l=1:size2(1)
        K(k,l)=features1(k)*features2(l);
    end
end
end
```
Another method: virtual SVs

1. Train a support vector machine, find the SVs

2. Apply small transforms to the SVs to which the solution should be invariant: get *virtual support vectors* (VSVs)
   - Why small transformations: since then new points should also be close to decision boundary and become SVs.

3. Train a new support vector machine with both the SVs and VSVs

**Note 1:** Virtual examples generated by non-SVs do not generally improve the performance.

**Note 2:** This approach yields the best current performance for hand written digit recognition.
Virtual SVs: toy example

Taken from Schölkopf and Smola p. 338

- Problem assumes invariance to horizontal transformations
- Given the training data, point ? is hard to classify
- Adding VSVs makes the identity of ? clear
Virtual SVs: hand written digits

Taken from Schölkopf and Smola p. 339

The identification of the digit should not change through small translations, rotations, or line thickness changes.
String kernel: description

- How do we find a kernel between strings? For example
  - Documents (internet, books,...)
  - Proteins (look like qvqlqqpgselsevrgasvkl..., where each letter is a different amino acid)

- A simple spectrum kernel: features count the number of times groups (substrings) of $m$ contiguous letters appear.

- Example: Protein $aababc$ with individual letters $a, b, c$ and $m = 2$:

\[
x = \begin{bmatrix}
1 & 2 & 0 & 1 & 0 & 1 & \ldots
\end{bmatrix}
\]

- The kernel is just $k(x_1, x_2) = x_1^T x_2$
Mismatch string kernel: description

- A less sparse kernel: counts slightly mismatched strings as being similar
- We allow a single letter of mismatch (but it is possible to use more)
- **Example:** Protein **aabac** with individual letters **a, b, c**, and \( m = 2 \)
  - The substring **aa** increments the features **aa, ab, ac, ba, ca** in the feature vector \( x \).
  - In other words: one element in the substring is a wildcard, and can take any value in the alphabet. So **aa** could be written as a combination of \( a^* \) and \( ^*a \).
String kernel: Demonstration I

Empirical comparison of the spectrum and mismatch kernels.

- Run the code `stringKernelDemo`: computes spectrum, mismatch Gram matrices. *Use only letters ’a’-’z’!*

- Example: compare the strings *hellio* and *helluo* with $m = 3$. Note ratio of diagonal to off-diagonal elements is less for the mismatch kernel.

- Try different values of substring length!
String kernel: Demonstration II

Run the code stringKernelTest2: computes both the spectrum and mismatch kernels for protein classification

The test file uses $m = 4$: try different substring lengths to observe the effects!

Simple kernel: $\text{class} \_ \text{loss}=0.12995+-0.040455$
Mismatch kernel: $\text{class} \_ \text{loss}=0.036087+-0.0054204$

- The mismatch kernel is less diagonally dominant, and shows greater similarity to other points within the same class.
- The cross-validation error associated with the mismatch kernel is much smaller.
String kernel: Results II (continued)

- Basic string kernel
- Mismatch kernel

Cross validation index vs Error

Basic string kernel matrix

Mismatch kernel matrix
Some other string kernels

- Strings that allow gaps: thus $aaba$ would include the feature $aaa$.
  - Larger numbers of letters in the gaps can be made to incur a greater penalty, so they don’t increment the corresponding entry in $x$ as much.
  - You need a more efficient method of calculating $k(x_1, x_2)$ (loops take too long!)

- Fisher kernels between the HMMs that generate the strings
SVM practical session III

Proper Experimental Design in Machine Learning

*How to perform experiments as unbiased as possible when limited amount of data is available*
Outline of the section

1. Designing algorithm using cross validation error:
   Example of a common practice to avoid.

2. Comparing classifiers:
   Statistical tests and their limits.

3. How to misuse cross validation:
   An example of a classical mistake when using cross validation.
Algorithm Design Strategy
How to design a learning algorithm

1. Consider a \textit{dataset} $D$

2. Apply an \textit{algorithm of your choice} on $D$

3. \textbf{Estimate} its generalization error

4. \textbf{If}: generalization error smaller than what exists in the literature for the same dataset:

   End of the process. \textbf{Outcome}: Publication

5. \textbf{Else}:

   Go back to step 2 with another algorithm

\textit{Estimate for the generalization error = 10 fold cross validation}
Cross validation

Definition 1 (n-fold cross validation)

1. Divide the training set (of size $m$) into $n$ disjoint sets $S_1, \ldots, S_n$ of equal size $n/m$

2. For each $S_i$:
   
   Train a classifier on $S \setminus S_i$

   Test it on $S_i \rightarrow$ error($i$)

3. Output the average error

This gives an estimate of the generalization error of the classifier when trained on $n - n/m$ data (Usually, $n = 10$).
A toy design process

- **Dataset**
  ionosphere dataset: 34 features, 175 training points.

- **Set of algorithms**
  gaussian SVM with hyperparameters \((\sigma, C)\)

- **Estimate of the generalization error**
  2-fold Cross Validation error

- **Outcome**
  A model whose test error is ”supposed” to be small.

Analogy with the algorithm design process:

- **Set of algorithms that you consider in the process = Gaussian SVM**

- **Considering a new algorithm = changing \((\sigma, C)\)**
Exercise:

Run the script ocve and try to find the best \((\sigma, C)\) value that minimize the CV error on the ionosphere dataset.

Before starting it note that:

- The design process is limited to 5 minutes (think about your strategy before).
- Try to use a non-random strategy as this search is supposed to model your search of a best algorithm.
- A sound is produced at each step of the search process in order to motivate you to find the best CV error (it is supposed to model the impact of your publication).
Mr. X’s result:

Test errors estimated on 176 points.
What’s happening?

My best bet: \( \sigma=4, \ C=6, \text{ 2-CV error}=6.31\%, \text{ test error}=7.95\% \)

Another result: \( \sigma=2, \ C=0.4, \text{ 2-CV error}=6.88\%, \text{ test error}=5.68\% \)
Overfitting the cross validation error:

The algorithm is chosen in order to minimize the cross validation error on the training set, not to have the lowest test error.

Cross validation error is close to test error but is not equal to it ⇒ optimizing too much leads to overfitting.

How to avoid overfitting:

- Do not optimize too much on cross validation.
- Try to keep a test set that you will use at the very end of your designing process.
- Do not compare your algorithm with others on the dataset you have used to design it.
Other flaws of the design process

1. **Number precision**: when testing on 176 points, one mistake represents 0.5% ⇒ no need to give too precise numbers.

2. **These numbers are estimates**: we only considered average here but variance information is important as well.
   ⇒ no statistical testing has been performed here.
Use of simple Statistical Tests
Comparison on a test set

When reporting result of an algorithm on a test set, it is worth providing the 95% confidence interval of a t-test:

1. Compute the average error \( (m_t = \text{size of the test set}) \):

\[
\mu = \frac{1}{m_t} \sum_{i=1}^{m_t} \ell(f(x_i), y_i)
\]

2. Compute the standard error \( s = \frac{\mu(1-\mu)}{\sqrt{m_t}} \)

3. And report:

\[
\mu \pm s \ p(0.95) \approx \mu \pm 2.2s
\]

\( p(0.95) \) is the p-value of a t-test with \( m_t - 1 \) degree of freedom

Check whether 5.68\% is different from 7.95\% when we tested on 176 points
Comparison with Cross Validation error

Let $\mu_i$ be the generalization error estimates on the set $S_i$ during the cross validation procedure. Assuming that the $\mu_i$’s are independent, we can use the t-test as before and report:

$$cv \ error \ \pm \ 2.2s$$

where $s = \frac{1}{n} \sqrt{\sum_{i=1}^{n} (\mu_i - \mu)^2}$ and $\mu = 1/n \sum_i \mu_i$.

Use the get_cv_error(sigma,C) function to check the confidence interval for the guess $sigma=2, \ C=0.4$ and $sigma=4, \ C=6$
Limitations of statistical testing

- The statistical tests tell you when two results are not statistical significant not when they are:
  - If two numbers are significantly different
    ⇒ maybe they really are
  - If two numbers are not significantly different
    ⇒ they certainly are not

- The hypothesis underlying the statistical tests are not valid in most cases.

- Even if two numbers are significantly different, it does not mean that this difference is meaningful.
Example of an improper use of Cross Validation

Out of the loop feature selection
The CV object

s = svm;
a = cv(s); %folds is set to 5 by default
a = cv(s,'folds=10');

[r,a]=train(a,d);
%run the CV procedure on s with the dataset d

tmp = get_mean(r);
%compute the average of the test and training errors

tmp = tmp.Y;
%get the averaged test error (and its std)

tmp = tmp(1);
%get the average only (tmp(2) is the std)
Feature selection

Feature selection object: 10 runs a feature selection algorithm and applies a SVM afterwards. It has the following parameters:

- **feat**: number of desired features (integer)
- **output_rank**: output a ranking list of feature if set to 1 (boolean)

Use the following code to perform a simple feature selection over the lymphoma dataset:

```matlab
a=10;
a.feat=10;
[r,a]=train(a,d);
```

The following code builds a new dataset from `d`:

```matlab
dnew = get(d,[],a.rank(1:10));
```
Exercise

1. Use the previous code to apply a feature selection on the lymphoma dataset (select 10 features) and to run a 5-fold CV afterwards.

What do you observe?

2. Run the following code

```matlab
a=10; % that’s an "L" not a "one"
a.output_rank=0; % find features, then output estimate
a.feat=10; % choose only 10 features
[r,a] = train(cv(a),d);
get_mean(r)
```

How different is the error from before? What was done wrong?
Discussion

- The cross validation loop should be at the top of everything else (including feature selection, model selection, etc.).
- If not possible or difficult, try only a small set of values/features: keep it simple to avoid overfitting.

Other remarks:

- Different splits of CV can lead to different error estimates ⇒ perform many CV’s with different split and average the results.
- Identifying the reasons why an algorithm works better than another one is difficult. Many unreported tricks (see first part of this practical session) can have a big impact although the basic principles of the learning algorithm are still the same.
Discussion on methodology:

- Performing a good experiment is really difficult.
  ⇒ Build your set-up so that it can be re-used for other datasets/algorithms.

Toward better experiments:

- **Share code and data:** provides a way to reproduce experiments
  ⇒ better comparisons, discussion possible.
- **Build a dedicated web page** containing all the experimental details: shows all the tricks that have been used.
- **Build re-usable algorithm:** can check the impact of the method on other datasets.

Create a directory/ Write down *everything* even if it won’t be published
Practical case
Exercise: compare two algorithms on the Wisconsin dataset

Load the wisconsin dataset and compare a SVM with a k-Nearest Neighbor method. You can optimize with respect to whatever you want (hyperparameter optimization, feature selection, choosing different kernels, etc.).

- Report errors with confidence interval/error bars.

Related objects: normalize, knn, param.