Introduction to Machine Learning
NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

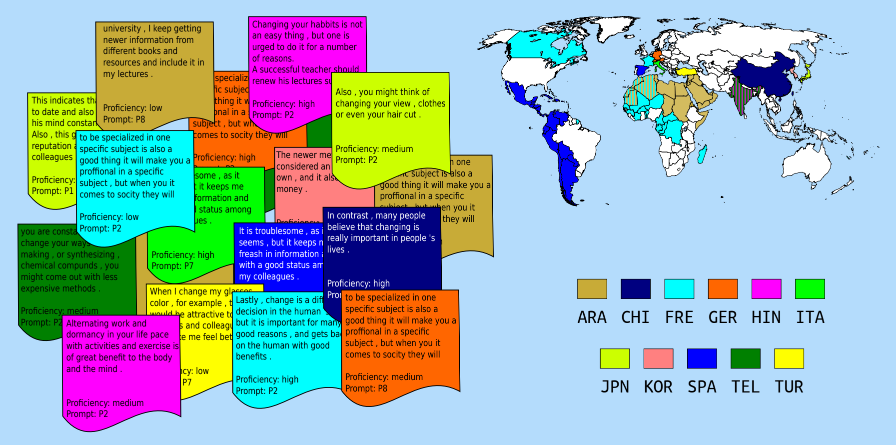
Barbora Hladká **Martin Holub**

{Hladka | Holub}@ufal.mff.cuni.cz

Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Support Vector Machines
Native Language Identification task

Native language identification task (NLI)



NLI

Identifying the native language (L1) of a writer based on a sample of their writing in a second language (L2)

Our data

- **L1s**: Arabic (ARA), Chinese (ZHO), French(FRA), German (DEU) Hindi (HIN), Italian (ITA), Japanese (JPN), Korean (KOR), Spanish (SPA), Telugu (TEL), Turkish (TUR)
- **L2**: English
- **Real-world objects**: For each L1, 1,000 texts in L2 from The ETS Corpus of Non-Native Written English (former TOEFL11), i.e. $\mathit{Train} \cup \mathit{DevTest}$
- **Target class:** L1

More detailed info is available at the course website.

References

- Barbora Hladká, Martin Holub, Vincent Kríž. Feature Engineering in the NLI Shared Task 2013: Charles University Submission Report. 2013. [pdf]
- Pavel Ircing, Jan Švec, Zbyněk Zajíc, Barbora Hladká, Martin Holub. Combining Textual and Speech Features in the NLI Task Using State-of-the-Art Machine Learning Techniques. 2017. [pdf]

NLI
Features used

96 numerical features = relative character frequencies

Example

"Finally having people with many academic broad know"

```
   <SPACE>          a          b          c          d          e
0.17073171 0.14634146 0.02439024 0.04878049 0.04878049 0.07317073
         m          n          o          F          g          h
0.04878049 0.09756098 0.07317073 0.02439024 0.02439024 0.04878049
         i          k          l          p          r          t
0.09756098 0.02439024 0.07317073 0.04878049 0.02439024 0.02439024
         v          w          y
0.02439024 0.04878049 0.04878049
```

Support Vector Machines in R

Online demo

- Java applet at http://svm.dcs.rhbnc.ac.uk/

The implementation of SVMs in R

- `library(e1071)`, but there are also other libraries (**kernlab**, **shogun** ...)
- training: function `svm()`
- prediction: function `predict()`
- `svm()` can work in both classification and regression mode
- if response variable is categorical (factor) the engine switches to classification

SVM in R

```
model = svm(formula, data=, kernel=, cost=, cross=, ...)
```

- `?svm`
- `kernel` defines the kernel used in training and prediction. The options are: linear, polynomial, radial basis and sigmoid (default: radial)
- `cost` – cost of constraint violation (default: 1)
- `cross` – optional, with the value k the k-fold cross-validation is performed

SVM kernels in `e1071`

Kernel name	Formula	Learning parameters and their default values
linear	$\mathbf{x}_i \cdot \mathbf{x}_j$	
polynomial	$(\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c_0)^d$	γ, `gamma`=1/(data dimension) c_0, `coef0`=0 d, degree=3
radial	$\exp(-\gamma(\|\|\mathbf{x}_i - \mathbf{x}_j\|\|^2))$	γ, `gamma`=1
sigmoid	$\tanh(\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c_0)$	γ, `gamma`=1/(data dimension) c_0, `coef0`=0

SVM – kernel functions

Non-linear kernel functions

- polynomial kernel
  - smaller degree can generalize better
  - higher degree can fit (only) training data better
- radial basis
  - very robust
  - you should try and use it when polynomial kernel is weak to fit your data

SVM Parameter tuning with `tune.svm`

- **SVM is a more complicated method in comparison with the previous and usually requires parameter tuning!**
- parameter tuning can take a very long time on big data, use a reasonably smaller part is often recommended

```
> model.tune= tune.svm(class ~ ., data=train.small,
                     kernel = "radial",
                     gamma = c(0.001, 0.005, 0.01, 0.015, 0.02),
                     cost = c(0.5, 1, 5,  10))
> model.tune
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
 0.01    1

- best performance: 0.739
```

Built-in cross-validation

K-fold cross-validation

- parameter cross

```
> model.best <- svm(class ~ ., train.small,
                                   kernel = "radial",
                                   gamma = 0.01,
                                   cost = 1,
                                   cross = 10)
> model.best$accuracies
 [1] 33.0 27.5 31.0 33.5 28.0 29.0 29.0 33.5 33.0 34.5
> model.best$tot.accuracy
# [1] 31.2
> prediction.best <- predict(model.best, test, type="class")
> mean(prediction.best==test$class)
[1] 0.3472727
```

Class weighting

- `class.weights` parameter
  In case of asymmetric class sizes you may want to avoid possibly overproportional influence of bigger classes. Weights may be specified in a vector with named components, like
  `m <- svm(x, y, class.weights = c(A = 0.3, B = 0.7))`

General hints on practical use of `svm()`

- Note that SVMs may be very sensible to the proper choice of parameters, so always check a range of parameter combinations, at least on a reasonable subset of your data.
- Be careful with large datasets as training times may increase rather fast.
- C-classification with the `RBF` kernel (default) can often be a good choice because of its good general performance and the few number of parameters (only two: `cost` and `gamma`).
- When you use C-classification with the `RBF` kernel: try small and large values for `cost` first, then decide which are better for the data by cross-validation, and finally try several `gamma` values for the better `cost`.