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 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 target attribute (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	$(\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c)^d$	γ, `gamma`=1/(data dimension) c, `coef0`=0 d, degree=3
radial	$\exp(-\gamma(\|\|\mathbf{x}_i - \mathbf{x}_j\|\|^2))$	γ, `gamma`=1
sigmoid	$\tanh(\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c)$	γ, `gamma`=1/(data dimension) c, `coef0`=0

SVM – 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 = 10,
                                  cross = 10)
> model.best$accuracies
[1] 26.81 30.90 36.36 28.63 38.18 28.18 37.72 35.90 34.09 30.90
> model.best$tot.accuracy
[1] 32.77
> prediction.best <- predict(model.best, test, type="class")
> mean(prediction.best==test$class)
[1] 33.36
```

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`.

Evaluation of multi-class classification task

target class	True Positive	False Positive	False Negative	class weight	Precision	Recall	F1 score
C_1	TP_1	FP_1	FN_1	w_1	P_1	R_1	F_1
C_2	TP_2	FP_2	FN_2	w_2	P_2	R_2	F_2
...	...	...	...	...	...	...	...
C_k	TP_k	FP_k	FN_k	w_k	P_k	R_k	F_k

- class weight w_i is the relative frequency of C_i class in the data
- macro-averaged F1 score $= \sum_{i=1}^{k} F_i/k$
- weighted-averaged F1 score $= \sum_{i=1}^{k} w_i F_i/k$

Evaluation of multi-class classification task

- In general, if you are working with an imbalanced dataset where all classes are equally important, using the macro average would be a good choice.
- If you have an imbalanced dataset but want to assign greater contribution to classes with more examples in the dataset, then the weighted average is preferred.