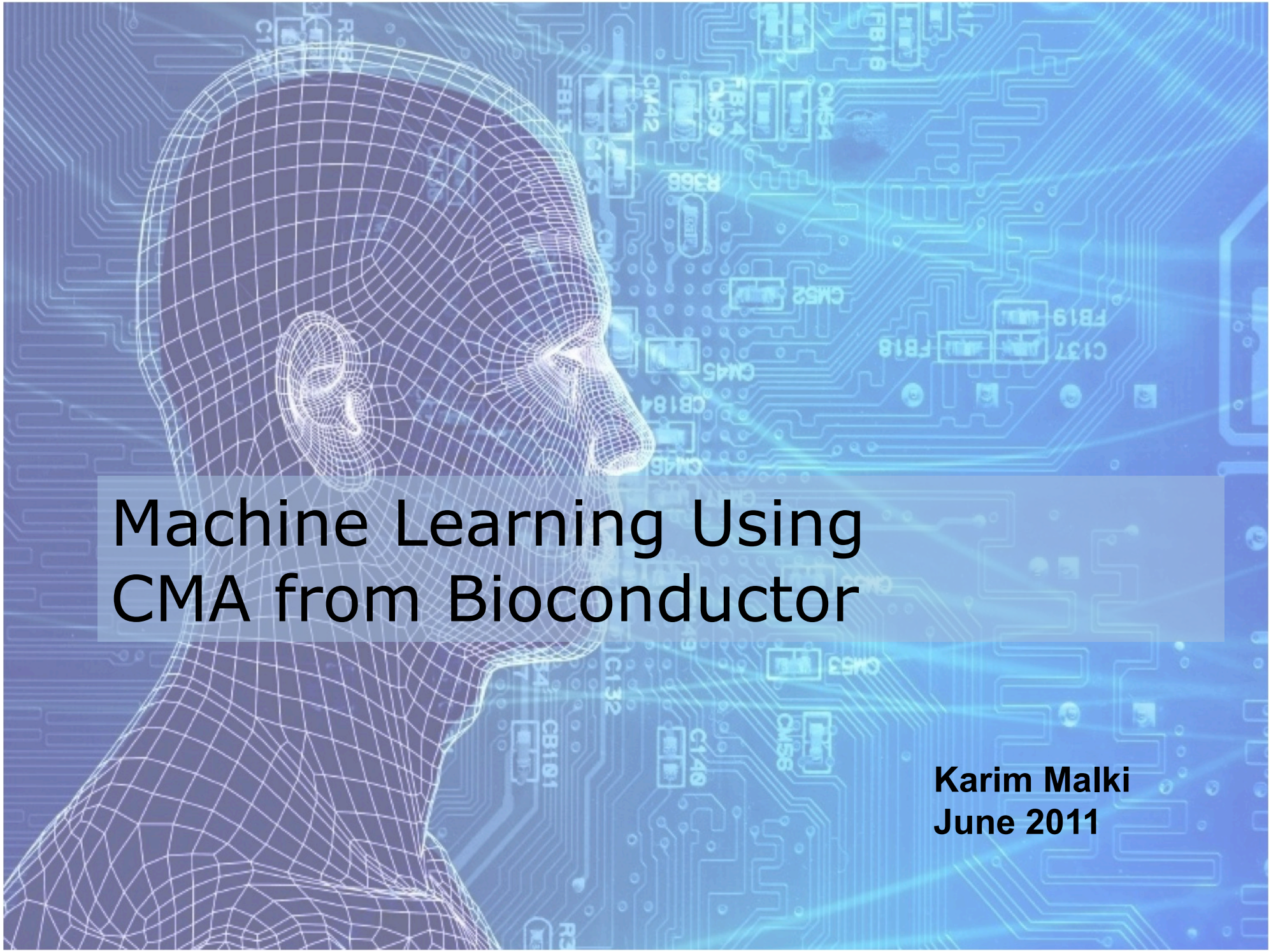

Machine Learning Using CMA from Bioconductor

Karim Malki
June 2011

Install CMA on your computer

```
###install package from bioconductor website##
source("http://bioconductor.org/biocLite.R")

biocLite("CMA”)
```

Recommended packages

###To fully exploit CMA capabilities consider installing the following ###

MASS , class , nnet , glmpath , e1071 , randomForest , plsgenomics , gbm , mgcv , corpcor

Start CMA

```
#########load CMA########
library(CMA)
####Consider also###
library(e1071)
library(glmpath)
```

Load your data

####data from Golub et al######

data(golub)

Leukemia data set

Explore you data – Garbage in = Garbage out

How many observations in total?

- How many belong to acute lymphoblastic leukemia ?
- How many belong to acute myeloid leukemia?

Extract expression and labels

Assign to golubY the labels

Assign to GolubX the expression, coerce it as.matrix

Extract expression and labels

Assign to golubY the labels

Assign to GolubX the expression, coerce it as.matrix

```
golubY <- golub[, 1]
golubX <- as.matrix(golub[, -1])
```

Generate Training Data Set

Five-fold cross-validation

```
set.seed(321)
fiveCVdat <- GenerateLearningsets(y = golubY,
    method = "CV", fold = 5, strat = TRUE)
```

stratified learning sets are generated by setting the argument strat =T. The random seed should be set for reproducibility

Generate Training Data Sets

MonteCarloCrossValidation

```
set.seed(456)
MCCVdat <- GenerateLearningsets(y = golubY,
   method = "MCCV", niter = 15, ntrain = floor(2/3
   *length(golubY)), strat = TRUE)
```

Bootstrap

```
set.seed(651)
bootstrap <- GenerateLearningsets(y = golubY,
   method = "bootstrap", niter = 15, strat = TRUE)
```

Support Vector Machine

We are going to use Support Vector Machine with linear kernel as classification method:

VARIABLE SELECTION

Variable selection is not strictly necessary, but performance of the SVM method improves

When noise features are removed (Hastie et al. [2001])

method name	**CMA function name**	**Package**	**Reference**
Componentwise Boosting	`compBoostCMA`	`CMA`	Bühlmann and Yu [2003]
Diagonal Discriminant Analysis	`dldaCMA`	`CMA`	McLachlan [1992]
Elastic Net	`ElasticNetCMA`	`glmpath`	Zhou and Hastie [2004]
Fisher's Discriminant Analysis	`fdaCMA`	`CMA`	Ripley [1996]
Flexible Discriminant Analysis	`flexdaCMA`	`mgcv`	Ripley [1996]
Tree-based Boosting	`gbmCMA`	`gbm`	Friedman [2001]
k-nearest neighbours	`knnCMA`	`class`	Ripley [1996]
Linear Discriminant Analysis $*$	`ldaCMA`	`MASS`	McLachlan [1992]
Lasso	`LassoCMA`	`glmpath`	Young-Park and Hastie [2007]
Feed-Forward Neural Networks	`nnetCMA`	`nnet`	Ripley [1996]
Probalistic nearest neighbours	`pknnCMA`	`CMA`	–
Penalized Logistic Regression	`plrCMA`	`CMA`	Zhu [2004]
Partial Least Squares $\star + *$	`pls_ldaCMA`	`plsgenomics`	Boulesteix and Strimmer [2007]
$\star +$ logistic regression	`pls_lrCMA`	`plsgenomics`	Boulesteix and Strimmer [2007]
$\star +$ Random Forest	`pls_rfCMA`	`plsgenomics`	Boulesteix and Strimmer [2007]
Probabilistic Neural Networks	`pnnCMA`	`CMA`	Specht [1990]
Quadratic Discriminant Analysis $*$	`qdaCMA`	`MASS`	McLachlan [1992]
Random Forest	`rfCMA`	`randomForest`	Breiman [2001]
PAM	`scdaCMA`	`CMA`	Tibshirani et al. [2002]
Shrinkage Discriminant Analysis	`shrinkldaCMA`	`CMA`	–
Support Vector Machine	`svmCMA`	`e1071`	Schölkopf and Smola [2002]

Ranking of Variables

- For simplicity, we choose the distribution free Wicloxon-Test to rank the variables, separately for each learning sample.

```
varsel_fiveCV <- GeneSelection(X = golubX, y =
  golubY, learningsets = fiveCVdat, method =
  "wilcox.test")
```

Repeat the same procedure for the other training sets

Ranking of Variables

```
varsel_MCCV <- GeneSelection(X = golubX, y =
   golubY, learningsets = MCCVdat, method =
   "wilcox.test”)
varsel_boot <- GeneSelection(X = golubX, y =
   golubY, learningsets = bootdat, method =
   "wilcox.test")
```

Explore Ranking for each generated set

- The toplist methods provides easy access to the top-ranked variables:

```
show(varsel_fiveCV)
toplist(varsel_fiveCV, iter = 1)
seliter <- numeric()
for (i in 1:5) seliter <- c(seliter, toplist(varsel_fiveCV, iter
    = i, top = 10, show = FALSE)$index)
sort(table(seliter), dec = TRUE)
```

Repeat for the other data sets

HyperParameter Tuning

```
set.seed(351)
tuningstep <- CMA:::tune(X = golubX, y = golubY,
    learningsets = fiveCVdat, genesel = varsel_fiveCV,
    nbgene = 100, classifier = svmCMA, grids = list(cost = c
    (0.1, 1, 10, 100, 200)), probability = T)
```

Run Classification Algorithm

- class_fiveCV <- classification(X = golubX, y = golubY, learningsets = fiveCVdat, genesel = varsel_fiveCV, nbgene = 100, classifier = svmCMA, cost = 0.1, probability = T)

Run SVM using the other two training sets.

Run Classification Algorithm

```
class_MCCV <- classification(X = golubX, y = golubY,
    learningsets = MCCVdat,genesel = varsel_MCCV,
    nbgene = 100, classifier = svmCMA,cost = 0.1,
    probability = T)

class_boot <- classification(X = golubX, y = golubY,
    learningsets = bootdat, genesel = varsel_boot, nbgene
    = 0.1, classifier = svmCMA, cost = 100, probability = T)
```

SVM Results

Results of classification are lists generated from each learning sample. It is possible to join the results into a single “big” object for visualisation purposes.

```
resultlist <- list(class_fiveCV, class_MCCV, class_boot)
result <- lapply(resultlist, join)
```

SVM Results

A common visualisation method is the “voting” plot

```
schemes <- c("five-fold CV", "MCCV", "bootstrap")
par(mfrow = c(3, 1))
for (i in seq(along = result)) plot(result[[i]], main =
    schemes[i])
```

Summarise results in a “confusion matrix”

```
invisible(lapply(result, ftable))
```

Empirical ROC Curve

Compare Performance of classifiers

```
par(mfrow = c(2, 2))
for (i in seq(along = result)) roc(result[[i]])
```

Aggregate Results

We can now join again and aggregate the results over the different splitting rules:

```
totalresult<-join(result)
ftable(totalresult)
```

Summarise results in a “confusion matrix”

```
 invisible(lapply(result, ftable))
roc(totalresult)
plot(totalresult)
```

Aggregate Results

We can now join again and aggregate the results over the different splitting rules:

```
totalresult<-join(result)
ftable(totalresult)
```

Summarise results in a “confusion matrix”

```
 invisible(lapply(result, ftable))
roc(totalresult)
plot(totalresult)
```

Interpreting Classification Results

"Confusion Matrixes" quantify performances via misclassication. However you can use the function "evaluation" for more advance performance evaluation.

```
av_MCCV <- evaluation(class_MCCV, measure =
  "average probability")
show(av_MCCV)
boxplot(av_MCCV)
summary(av_MCCV)
```

Compare with results using different training sets

Interpreting Classification Results

"Confusion Matrixes" quantify performances via misclassication. However you can use the function "evaluation" for more advance performance evaluation.

```
av_MCCV <- evaluation(class_MCCV, measure = "average probability")
show(av_MCCV)
boxplot(av_MCCV)
summary(av_MCCV)
```

Average probability is the average predicted probability for the correct class:

Compare with results using different training sets

Interpreting Classification Results

?evaluation

A number of options are available using the evaluation function, including applying the evaluation scheme to observations as opposed to iterations.

```
av_obs_MCCV <- evaluation(class_MCCV, measure =
  "average probability", scheme = "obs")
show(av_obs_MCCV)
```

Misclassifying Observations

Some observations are harder to classify than others. It is frequently of interest to know which observations are consistently misclassified; these are candidates for outliers or wrong class labels.

```
obsinfo(av_obs_MCCV, threshold = 0.6)
```

References:

Slawski, M. Daumer, M. Boulesteix, A.-L. (2008) CMA - A comprehensive Bioconductor package for supervised classification with high dimensional data. *BMC Bioinformatics 9: 439*