# LedPred: Learning from DNA to Predict enhancers

Elodie Darbo, Denis Seyres, Aitor Gonzalez

Tuesday 1st November, 2022

## Contents

**1 Introduction** **1**

**2 Description** **2**

2.1 Data . . . . . . . . . . 2

2.2 Learning from CRM-contained information to predict new regulatory features . . . . . . . . . . 2

2.2.1 Building the training set . . . . . . . . . . 2

2.2.2 Optimization of support vector machine model . . . . . . 3

2.2.3 Definition of the optimal SVM parameters ($\gamma$ and C) . . . 3

2.2.4 Sorting and selecting features according to their importance in the traing set description . . . . . . . . . . 4

2.2.5 Plotting the performances of the model . . . . . . . . . . 4

2.3 Function description . . . . . . . . . . 5

2.3.1 From bed CRM genomic coordinates to training set matrix 5

2.3.2 Scaling of the CRM feature matrix . . . . . . . . . . 5

2.3.3 SVM parameter optimization . . . . . . . . . . 5

2.3.4 Features ranking . . . . . . . . . . 6

2.3.5 Selecting features . . . . . . . . . . 7

2.3.6 Creating the best model . . . . . . . . . . 7

2.3.7 Plotting model perfomance . . . . . . . . . . 8

2.3.8 Using the model to score unknown sequences . . . . . . . 8

2.3.9 From the matrix to the model in one function . . . . . . . 9

**3 Session Information** **10**

## 1 Introduction

Biological processes, such as cell/tissue differentiation, involve several regulatory networks of transcription factors and signalling inputs. The complexity of these networks and what these pathways control in terms of downstream gene networks remains largely unknown.

Using the information contained in few known regulatory elements driving the expression in a specific biological process, this package predicts new cis-regulatory modules (CRMs) using support vector machine (SVM) learning of

transcription factor binding sites (TFBS) motifs, next-generation sequencing (NGS) data signal and any set of pre-calculated values. This vignette uses Drosophila data, but the package can be used with any species and has been tested with human data.

# 2 Description

## 2.1 Data

The data were generated by collecting experimentally validated CRMs and random genomic coordinates. In this example, we focused in CRMs involved in heart differentiation in Drosophila melanogater and ortholog regions in other Drosophila species. The genomic sequences were scanned and scored with 259 Drosophila Position-Specific Scoring Matrices (PSSMs) using *matrix-scan* [1]. From these results, a feature matrix (from our *mapFeaturesToCRMs* function) containing the training set has been built. It contains a line per genomic region and a column per feature plus a binary response vector column (1 and -1 for positive and negative sequences respectively). The files used to produce this example are stored in the *extdata* folder of this package.

```
> library(LedPred)
> data(crm.features)
```

## 2.2 Learning from CRM-contained information to predict new regulatory features

### 2.2.1 Building the training set

**Positive Regions**
The positive regions define the template from which new regions will be predicted. These regions are known to be involved in the process of interest and we assume that they contain the information specific to this process.

**Negative Regions**
The negative regions allow the model to learn which information are relevant to describe the positive regions and which are found randomly. These regions can be provided by the user or computed by shuffling the positive regions coordinates over a set of background sequences.

**Descriptive features: Position-Specific Scoring Matrices**
We retrieve the DNA sequences corresponding to the genomic coordinates and scan them with transcription factor matrices using *matrix-scan* [1]. This tools computes a p-value with each matrix for each position of the sequence depending on the likelihood of the site to belong to the matrix over the likelihood to belong to the background. Both likelihoods are computed using a markov model. The output is a list of significant p-values per sequence and region corresponding to binding sites of the transcription factor (TF) in the sequence. To compute a score per region and TF, we calculate the sum of the negative logarithm to base 10 of the p-values of binding sites in a sequence, i.e. $\sum_i -log_{10}(\text{p-val}_i)$ where $i$ is the index of the significant p-values corresponding to the binding sites.

**Descriptive features: NGS data**

These data can be given in two different formats: (i) Peak files containing the genomic coordinates in Bed format of regions significantly enriched in signal. These regions are usually defined by using a peak-calling algorithm. In this case a binary value is attributed to the region: 1 if there is an overlap between the region and one or several peaks, 0 otherwise. The minimal overlap fraction of the region is controlled by the *bed.overlap* argument. The default value is 1bp of the region. (ii) Signal files (wig or bigwig formats, see UCSC help for details) containing the genome-wide affinity signal. In this case, we compute the average number of reads in the region.

#### 2.2.2 Optimization of support vector machine model

Machine learning approaches are used to build models that are able to discriminate between positive and negative sets combining and weighting information given by some predictors, here we use the presence of instance of PSSMs in the regions. This model is then used to predict new positive features. The negative and positive regions are labeled by a binary response vector. The resulting matrix is named the training set. The goal is to define a function as close as possible to the response vector. The method assigns to each descriptive feature a weight corresponding to the impact of the feature in the discrimination between positive and negative regions. Here we propose a method using Support Vector Machine (SVM) and tested with the linear or radial kernels. This package is based on the libsvm and can be used in principle with other kernels of this library [9].

- The SVM with linear kernel assumes that there exists a linear hyper plane that separates positive and negative data. As it exists an infinity of such planes, the SVM will maximize the margin between the positive/negative sets and the plane. The $\beta$ parameters variance is constrained by the cost parameter C ($1/\lambda$). This parameter allows to avoid overfitting by relaxing this constrain.
- The radial kernel function is a similarity function which transforms the descriptive feature space in a space of higher dimension. It assumes that there is no linear hyperplane able to separate the positive and negative regions. Two variables are very important: the cost parameter C and the $\gamma$ parameter ($\gamma/\sigma^2$) which allows to smoothen the similarity function in order to be more or less stringent. The similarity function returns a probability between 0 and 1.

The aim of this package is to compute a predictive model general enough to avoid the overfitting the training set. We propose a sequential pipeline to achieve this purpose:

#### 2.2.3 Definition of the optimal SVM parameters ($\gamma$ and C)

We test the different combinations of C and $\gamma$ parameters given by the user and return the prediction error computed using cross-validation. At this step, each set (positive and negative) is partitioned in subsets. One subset of

each group is used as a training set to predict in the remaining subsets the positive and negative regions, the error of prediction is then calculated. This error represents overfitting, which means that the model is too specific to the training set and fails to generalise and predict new regions.

#### 2.2.4 Sorting and selecting features according to their importance in the traing set description

The ranked list of features based on decreasing "importance" is computed by recursive feature elimination (RFE) [2, 3]. During this step, SVM is train on subsets of the *feature.matrix* and ranked according to the weight given by the resulting classifier. A mean rank ("importance") is attributed to each feature.

To select the optimal number of features, we compute the Cohen's kappa coefficient [4] of models built from increasing number of features from the sorted list of features, the step of the incrementation is defined by the *step.nb* parameter.
The kappa coefficient is defined as:

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \tag{1}$$

where $Pr(a)$ is the observed accuracy, that is the proportion of well predicted regions (True Positive TP, True Negative TN) among the total number of regions (N).

$$Pr(a) = \frac{TP + TN}{N} \tag{2}$$

$Pr(e)$ is the expected accuracy that represents the accuracy that any random classifier would be expected to achieve. $Pr(e)$ is defined as:

$$Pr(e) = \frac{(TP + FN) * (TP + FP)}{N} + \frac{(FP + TN) * (FN + TN)}{N} \tag{3}$$

where, $TP$: True Positive; $TN$: True Negative; $FP$: False Positive; $FN$: False Negative; $N$ the total number of objects.

Given the optimal parameters and features, we can create now the model.

#### 2.2.5 Plotting the performances of the model

We can evaluate the selected model by plotting the precision/recall, the precision/cut off, the recall/cut off and the ROC curves. This is performed using cross-validation results. We used the R package ROCR [5].

$$precision = \frac{TP}{TP + FP} \tag{4}$$

$$recall = \text{true positive rate} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{false positive rate} = \frac{FP}{TN + FP} \tag{6}$$

The *precision* indicate the proportion of true positive among all the regions predicted as positive. The *recall* indicates the proportion of true positive detected among all the true positive. Thanks to these curves we can choose the most appropriate cut off on the probability according to the question. Here we want to select regions in order to test them experimentally so we want high precision to avoid false positive. The ROC curve represents the false positive rate against the *truepositiverate*, a high area under the resulting curve means that the model can return high number of positive regions with a low false positive rate.

## 2.3 Function description

### 2.3.1 From bed CRM genomic coordinates to training set matrix

The *mapFeaturesToCRMs* function takes BED files containing the coordinates of the positive regions and for the negative set a bed file or a set of background sequences for sampling randomly. Regarding the features, a transfac file with PSSMs and another file with background frequences, BED or WIG files for NGS data and a BED files with arbitrary values in the first column. This function requires a very diverse palette of tools for this computation, namely *RSAT matrix-scan* [1], *BedTools* [7] and IntervalStats [8]. To simplify its use, the *mapFeaturesToCRMs* function will connect to a REST server, send the data, and get the results of the calculations. The result is a list with the matrix as a *data.frame* object, and the standard and error logs of the computation.

```
> dirPath <- system.file("extdata", package="LedPred")
> file.list <-   list.files(dirPath, full.names=TRUE)
> background.freq <- file.list[grep("freq", file.list)]
> positive.regions <-  file.list[grep("positive", file.list)]
> negative.regions <-  file.list[grep("negative", file.list)]
> TF.matrices <-  file.list[grep("tf", file.list)]
> ngs.path <- system.file("extdata/ngs", package="LedPred")
> ngs.files=list.files(ngs.path, full.names=TRUE)
> #crm.feature.list <- mapFeaturesToCRMs(URL = 'http://ifbprod.aitorgonzalezlab.org/map_fe
> #names(crm.feature.list)
> #class(crm.feature.list$crm.features)
```

### 2.3.2 Scaling of the CRM feature matrix

The PSSM calculation returns feature vectors with values organized similar to negative exponential distributions and average of around 0.08. The intersection of BED files with the sequences return mostly zero values and and some ones. To bring both values to the same range all values are divided internally with the Euclidean length of the feature vector.

### 2.3.3 SVM parameter optimization

The *mcTune* function is a modified version of the function *tune* from package *e1071* that uses parallel computation [6]. It tests the different combinations of cost C and gamma parameters given as vectors in a list and will return the

prediction error computed during the cross-validation step. The results is a list containing the best parameters (Fig. 1)..

```
> #crm.features=crm.feature.list$crm.features
> #cost.vector <- c(1,3,10)
> #gamma.vector <- c(1,3,10)
> #c.g.obj <- mcTune(data = crm.features, ranges = list(cost=cost.vector, gamma=gamma.vect
> #names(c.g.obj)
> #cost <- c.g.obj$e1071.tune.obj$best.parameters$cost
> #gamma <- c.g.obj$e1071.tune.obj$best.parameters$gamma
```

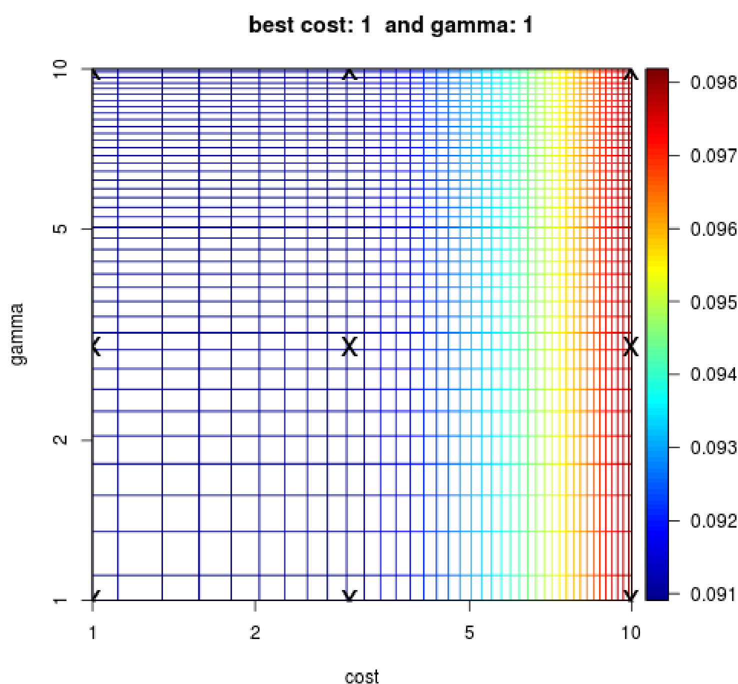


Figure 1: Error prediction by SVM linear kernel method. We use the same plot for linear and radial kernel mcTune optimization but gamma does not give any information for the linear kernel.

#### 2.3.4 Features ranking

The *rankFeatures* function performs a Recursive Feature Elimination (RFE) and return a *data.frame* object containing the ranked features list, their ID (number of the column in the feature matrix) and the average rank.

```
> #feature.ranking <- rankFeatures(data=crm.features, cost=cost, gamma=gamma, kernel='line
> #head(feature.ranking)
```

#### 2.3.5 Selecting features

The *tuneFeatureNb* function computes the kappa measure on the feature matrix in which the features are ranked according to their decreasing importance. It starts with one feature and increases the number of features included in the matrix by an step (nb.step argument) until all features are included. This step depends *feature.ranking* object from the step before. The returned object is a list contaning a data frame summarizing the kappa coefficient for the different number of features included in the model. We keep the number corresponding to the highest kappa for the following steps (Fig. 2).

```
> #feature.nb.tuner.obj <- tuneFeatureNb(data=crm.features, feature.ranking=feature.rankin
> #names(feature.nb.tuner.obj)
> #feature.nb.tuner.obj$best.feature.nb
```

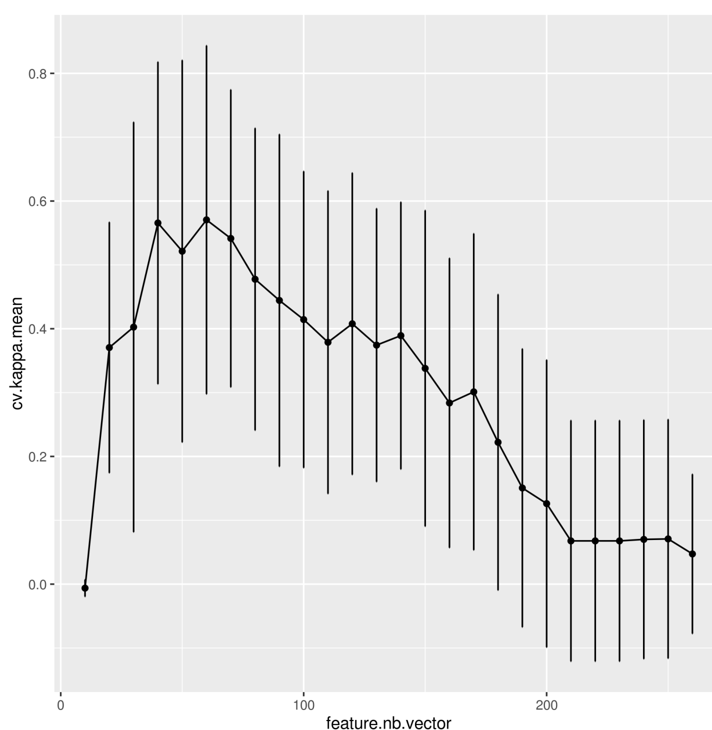


Figure 2: Kappa coefficient showing the performance of the model as a function of the number of features.

#### 2.3.6 Creating the best model

The *createModel* function takes the tuned SVM parameters, the ranked features and the optimal number of features and returns a SVM model.

```
> #feature.nb <- 60
> #model.obj <- createModel(data=crm.features, cost=cost, gamma=gamma, feature.ranking=fea
> #names(model.obj)
> #model.obj$model
```

An interesting information is the sign of the feature weights, which tells whether the feature is over- or under-represented in the positive or negative sets. It can be shown for each feature with this command:

```
> #feature.weights <- as.data.frame(t(t(model.obj$model$coefs) %*% model.obj$model$SV))
> #head(feature.weights)
```

### 2.3.7 Plotting model perfomance

The *evaluateModelPerformance* function returns a list of prediction probabilities computed during the rounds of cross-validation and plots the performance results (Fig 3).

```
> #probs.labels.list <- evaluateModelPerformance(data=crm.features, feature.ranking=featur
```

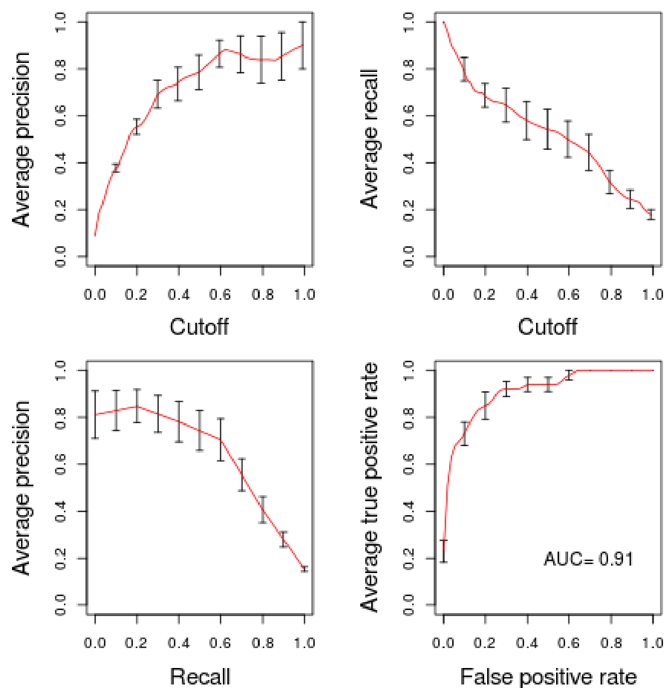


Figure 3: Model performance. Four plots and the AUC are given to evaluate the performance of the model: Precision and recall as a function of the score, precision versus recall and the ROCR curve with the area under the curve (AUC).

### 2.3.8 Using the model to score unknown sequences

In the last step, we want the model to score unknown sequences. With this aim, we need first to create a numerical matrix for the unknown sequences with the best features. This matrix can be created by our *mapFeaturesToCRMs* function using the *feature.nb* and the *feature.rank* arguments to select the relevant

features, by setting *negative.bed* argument at *NULL* and *shuffling* argument at 0 to produce a feature matrix that does not contain negative sequences.

```
> dirPath <- system.file("extdata", package="LedPred")
> file.list <-   list.files(dirPath, full.names=TRUE)
> background.freqs <- file.list[grep("freq", file.list)]
> positive.bed <-  file.list[grep("prediction_small", file.list)]
> TF.matrices <-  file.list[grep("259_matrices_lightNames", file.list)]
> ngs.path <- system.file("extdata/ngs", package="LedPred")
> ngs.files=list.files(ngs.path, full.names=TRUE)
> #prediction.crm.features.list <- mapFeaturesToCRMs(URL = 'http://ifbprod.aitorgonzalezla
> #  pssm=TF.matrices, background.freqs=background.freqs,
> #  genome='dm3', ngs=ngs.files, feature.ranking=feature.ranking, feature.nb=feature.nb,
> #    crm.feature.file = "vignette_pred.crm.features.tab",
> #    stderr.log.file = "vignette_pred.stderr.log", stdout.log.file = "vignette_pred.stdo
> #names(prediction.crm.features.list)
> #prediction.crm.features <- prediction.crm.features.list$crm.features
```

This creates a matrix for the unknown sequences that can be used by the *scoreData* function with the *model* argument to score the sequences of the matrix.

```
> #pred.test <- scoreData(data=prediction.crm.features, model=model.obj, score.file="vigne
> #pred.test
```

#### 2.3.9 From the matrix to the model in one function

The *LedPred* function takes the numerical matrix created by *mapFeaturesToCRMs* and runs sequentially all functions needed to create the model. It returns a list containing all the objects produced at each each step.

```
> crm.features=data(crm.features)
> cost.vector <- c(1,3,10)
> gamma.vector <- c(1,3,10)
> #ledpred.obj=LedPred(data=crm.features, cl=1, ranges = list(cost=cost.vector, gamma=gamm
> #names(ledpred.obj)
```

We annotate test data again

```
> dirPath <- system.file("extdata", package="LedPred")
> file.list <-   list.files(dirPath, full.names=TRUE)
> background.freqs <- file.list[grep("freq", file.list)]
> positive.bed <-  file.list[grep("prediction_small", file.list)]
> TF.matrices <-  file.list[grep("259_matrices_lightNames", file.list)]
> ngs.path <- system.file("extdata/ngs", package="LedPred")
> ngs.files=list.files(ngs.path, full.names=TRUE)
> #prediction.crm.features.obj2 <- mapFeaturesToCRMs(URL = 'http://ifbprod.aitorgonzalezla
> #  pssm=TF.matrices, background.freqs=background.freqs,
> #  genome='dm3', ngs=ngs.files, feature.ranking=ledpred.obj$feature.ranking, feature.nb=
> #    crm.feature.file = "vignette2_pred.crm.features.tab",
> #    stderr.log.file = "vignette2_pred.stderr.log", stdout.log.file = "vignette2_pred.st
> #names(prediction.crm.features.obj2)
> #prediction.crm.features2 <- prediction.crm.features.obj2$crm.features
```

Now we can score test data

```
> #pred.test2 <- scoreData(data=prediction.crm.features2, model=ledpred.list$model.obj, sc
> #pred.test2
```

## 3 Session Information

Here is the output of *sessionInfo* on the system on which this document was compiled:

```
> sessionInfo()

R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.5 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.16-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.16-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_GB              LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] LedPred_1.32.0 e1071_1.7-12

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.9        plyr_1.8.7        pillar_1.8.1      compiler_4.2.1
 [5] bitops_1.0-7      class_7.3-20      tools_4.2.1       testthat_3.1.5
 [9] jsonlite_1.8.3    lifecycle_1.0.3   tibble_3.1.8      gtable_0.3.1
[13] lattice_0.20-45   pkgconfig_2.0.3   rlang_1.0.6       cli_3.4.1
[17] DBI_1.1.3         parallel_4.2.1    akima_0.6-3.4     irr_0.84.1
[21] dplyr_1.0.10      generics_0.1.3    vctrs_0.5.0       plot3D_1.4
[25] grid_4.2.1        tidyselect_1.2.0  glue_1.6.2        R6_2.5.1
[29] fansi_1.0.3       tcltk_4.2.1       sp_1.5-0          ROCR_1.0-11
[33] ggplot2_3.3.6     magrittr_2.0.3    scales_1.2.1      assertthat_0.2.1
[37] misc3d_0.9-1      lpSolve_5.6.17    colorspace_2.0-3  utf8_1.2.2
[41] proxy_0.4-27      RCurl_1.98-1.9    munsell_0.5.0     brio_1.1.3
```

## References


[1] Turatsinze JV, Thomas-Chollier M, Defrance M, van Helden J.: **Using RSAT to scan genome sequences for transcription factor binding sites and cis-regulatory modules.** *Nature Protocole* 2008, **3**: (10) 1578-88.

[2] Duan K.-B., Rajapakse J. C., Wang H., and Azuaje F.: **Multiple SVM-RFE for gene selection in cancer classification with expression data**. *IEEE transactions on nanobioscience* 2005, **4**: (3) 228-234.

[3] Colby J: **R implementation of the (multiple) Support Vector Machine Recursive Feature Elimination - mSVM-RFE.** `http://github.com/johncolby/SVM-RFE` 2011.

[4] Byrt T., Bishop J and Carlin J.B.: **Bias, prevalence and kappa.** *Journal of Clinical Epidemiology*, 1993 **46**: 423–429.

[5] Sing T., Sander O, Beerenwinkel N. and Lengauer T.: **ROCR: visualizing classifier performance in R.** *Bioinformatics*, 2005 **21**: (20) 7881.

[6] Meyer D., Dimitriadou E., Hornik K., Weingessel A., Leisch F., Chang C.C., Lin C.C.: **e1071: Misc Functions of the Department of Statistics (e1071)** `http://cran.r-project.org/web/packages/e1071/`, 2014

[7] Quilan A.R. and Hall I.M.: **BEDTools: a flexible suite of utilities for comparing genomic features** *Bioinformatics*, 2010 **26**: (6) 841-2

[8] Chikina M.D. and Troyanskaya O.G.: **An effective statistical evaluation of ChIPseq dataset similarity** *Bioinformatics*, 2012 **28**: (5) 607-13

[9] Chang, Chih-Chung and Lin, Chih-Jen : **LIBSVM: A library for support vector machines** *ACM Transactions on Intelligent Systems and Technology* 2011 **2**: (5) 27:1-27:27