A transfer learning algorithm for spatial proteomics

Lisa M. Breckels and Laurent Gatto*
Computational Proteomics Unit
University of Cambridge, UK

May 5, 2017

Abstract

This vignette illustrates the application of a transfer learning algorithm to assign proteins to sub-cellular localisations. The knnClassification algorithm combines primary experimental spatial proteomics data (LOPIT, PCP, etc.) and an auxiliary data set (for example binary data based on Gene Ontology terms) to improve the sub-cellular assignment given an optimal combination of these data sources.

Keywords: Bioinformatics, organelle, spatial proteomics, machine learning, transfer learning

*lg390@cam.ac.uk
Contents

1 Introduction ....................................................... 3

2 Preparing the auxiliary data .................................... 3
   2.1 The Gene Ontology .......................................... 3
       2.1.1 Preparing the query parameters ................... 4
       2.1.2 Preparing the auxiliary data from the GO ontology 5
       2.1.3 A note on reproducibility ............................ 6
   2.2 The Human Protein Atlas .................................... 6
   2.3 Protein-protein interactions ................................ 9

3 Support vector machine transfer learning .................. 11

4 Nearest neighbour transfer learning .......................... 11
   4.1 Optimal weights ........................................... 11
   4.2 Choosing weights ........................................... 15
   4.3 Applying best \theta \text{ weights} ............................ 17

5 Conclusions ..................................................... 17
1 Introduction

Our main data source to study protein sub-cellular localisation are high-throughput mass spectrometry-based experiments such as LOPIT, PCP and similar designs (see [1] for an general introduction). Recent optimised experiments result in high quality data enabling the identification of over 6000 proteins and discriminate numerous sub-cellular and sub-organellar niches [2]. Supervised and semi-supervised machine learning algorithms can be applied to assign thousands of proteins to annotated sub-cellular niches [3, 4] (see also the pRoloc-tutorial vignette). These data constitute our main source for protein localisation and are termed thereafter primary data.

There are other sources of data about sub-cellular localisation of proteins, such as the Gene Ontology [5] (in particular the cellular compartment name space), quantitative features derived from protein sequences (such as pseudo amino acid composition) or the Human Protein Atlas [6] to cite a few. These data, while not optimised to a specific system at hand and, in the case of annotation feature, not as reliable as our experimental data, constitute an invaluable, often plentiful source of auxiliary information.

The aim of a transfer learning algorithm is to combine different sources of data to improve overall classification. In particular, the goal is to support/complement the primary target domain (experimental data) with auxiliary data (annotation) features without compromising the integrity of our primary data. In this vignette, we describe the application of transfer learning algorithms for the localisation of proteins from the pRoloc package, as described in


Two algorithms were developed: a transfer learning algorithm based on the \( k \)-nearest neighbour classifier, coined kNN-TL hereafter, described in section 4, and one based on the support vector machine algorithm, termed SVM-TL, described in section 3.

> library("pRoloc")

2 Preparing the auxiliary data

2.1 The Gene Ontology

The auxiliary data is prepared from the primary data’s features. All the GO terms associated to these features are retrieved and used to create a binary matrix where a one (zero) at position \((i, j)\) indicates that term \(j\) has (not) been used to annotate feature \(i\).

The GO terms are retrieved from an appropriate repository using the biomaRt package. The specific Biomart repository and query will depend on the species under study and the type of features. The first step is to prepare annotation parameters that will enable to perform the query. The pRoloc package
provides a dedicated infrastructure to set up the query to the annotation resource and prepare the GO data for subsequent analyses. This infrastructure is composed of:

1. define the annotation parameters based on the species and feature types;
2. query the resource defined in (1) to retrieve relevant terms and use the terms to prepare the auxiliary data.

We will demonstrate these steps using a LOPIT experiment on Human Embryonic Kidney (HEK293T) fibroblast cells [3], available and documented in the *pRolocdata* experiment package as andy2011.

```r
> library("pRolocdata")
> data(andy2011)
```

### 2.1.1 Preparing the query parameters

The query parameters are stored as *AnnotationParams* objects that are created with the `setAnnotationParams` function. The function will present a first menu with 215. Once the species has been selected, a set of possible identifier types is displayed.

![Figure 1: Selecting species (left) and feature type (right) to create an AnnotationParams instance for the human andy2011 data.](image)

It is also possible to pass patterns\(^1\) to match against the species ("Homo sapiens") and feature type ("UniProtKB/Swiss-Prot ID").

```r
> ap <- setAnnotationParams(inputs =
+                        c("Homo sapiens",
+                          "UniProtKB/Swiss-Prot ID"))
```

*Using species Homo sapiens genes (GRCh38.p5)*

*Using feature type UniProtKB/Swiss-Prot ID(s) [e.g. A0A0758B5P]*

Connecting to Biomart...

```r
> ap
```

\(^1\)These patterns must match uniquely or an error will be thrown.
Object of class "AnnotationParams"
Using the 'ENSEMBL_MART_ENSEMBL' BioMart database
Using the 'hsapiens_gene_ensembl' dataset
Using 'uniprotswissprot' as filter
Created on Fri May  5 19:24:20 2017

The `setAnnotationParams` function automatically sets the annotation parameters globally so that the `ap` object does not need to be explicitly set later on. The default parameters can be retrieved with `getAnnotationParams`.

2.1.2 Preparing the auxiliary data from the GO ontology

The feature names of the `andy2011` data are UniProt identifiers, as defined in the `ap` accession parameters.

```r
data(andy2011)
head(featureNames(andy2011))
```

```
[1] "O00767" "P51648" "Q2TAA5" "Q9UKV5" "Q12797"
[6] "P16615"
```

The `makeGoSet` function takes an `MSnSet` class (from which the feature names will be extracted) or, directly a vector of characters containing the feature names of interest to retrieve the associated GO terms and construct an auxiliary `MSnSet`. By default, it downloads *cellular component* terms and does not do any filtering on the terms evidence codes (see the `makeGoSet` manual for details). Unless passed as argument, the default, globally set `AnnotationParams` are used to define the Biomart server and the query.

```r
andygoset <- makeGoSet(andy2011)
```

```
MSnSet (storageMode: lockedEnvironment)
assayData: 1371 features, 796 samples
  element names: exprs
protocolData: none
phenoData: none
featureData
  featureNames: O00767 P51648 ... 075312 (1371 total)
  fvarLabels: Accession.No. Protein.Description ... UniProtKB.entry.name (10 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: 
```

--- Processing information ---

---

2 The annotation parameters could also be passed explicitly through the `params` argument.
We now have a primary data set, composed of 1371 protein quantitative profiles for 8 fractions along the density gradient and an auxiliary data set for 796 cellular compartment GO terms for the same 1371 features.

2.1.3 A note on reproducibility

The generation of the auxiliary data relies on specific Biomart server Mart instances in the AnnotationParams class and the actual query to the server to obtain the GO terms associated with the features. The utilisation of online servers, which undergo regular updates, does not guarantee reproducibility of feature/term association over time. It is recommended to save and store the AnnotationParams and auxiliary MSnSet instances. Alternatively, it is possible to use other Bioconductor infrastructure, such as specific organism annotations and the GO.db package to use specific versioned (and thus traceable) annotations.

2.2 The Human Protein Atlas

The feature names of our LOPIT experiment are UniProt identifiers, while the Human Protein Atlas uses Ensembl gene identifiers. This first code chunk matches both identifier types using the Ensembl Biomart server.

```r
> fvarLabels(andy2011)[1] <- "accession" ## for left_join matching
> ## convert protein accession numbers to ensembl gene identifiers
> library("biomaRt")
> mart <- useMart("ensembl",dataset="hsapiens_gene_ensembl")
> filter <- "uniprotswissprot"
> attrib <- c("uniprot_gn", "uniprotswissprot", "ensembl_gene_id")
> bm <- getBM(attributes = attrib,

3This section has been updated on the 30th November to reflect changes in Biomart.
+     filters = filter,
+   values = fData(andy2011)[, "accession"],
+   mart = mart)
> head(bm)

<table>
<thead>
<tr>
<th>uniprot_gn</th>
<th>uniprotswissprot</th>
<th>ensembl_gene_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0FGR8</td>
<td>ENSG00000117868</td>
<td></td>
</tr>
<tr>
<td>A1L0T0</td>
<td>A1L0T0 ENSG00000105135</td>
<td></td>
</tr>
<tr>
<td>A2RRP1</td>
<td>A2RRP1 ENSG00000151779</td>
<td></td>
</tr>
<tr>
<td>A4FU01</td>
<td>A4FU01 ENSG0000014914</td>
<td></td>
</tr>
<tr>
<td>A6NC56</td>
<td>A6NC56 ENSG00000204128</td>
<td></td>
</tr>
<tr>
<td>A6NKG5</td>
<td>A6NKG5 ENSG00000254656</td>
<td></td>
</tr>
</tbody>
</table>

> ## HPA data
> library("hpar")

This is hpar version 1.18.0, based on the Human Protein Atlas
Version: 16.1
Release data: 2017.01.31
Ensembl build: 83.38
See '?hpar' or 'vignette('hpar')' for details.

> ## using old version for traceability
> setHparOptions(hpadata = "hpaSubcellularLoc14")
> hpa <- getHpa(bm$ensembl_gene_id)
> hpa$Reliability <- droplevels(hpa$Reliability)
> colnames(hpa)[1] <- "ensembl_gene_id"
>
> library("dplyr")
> hpa <- left_join(hpa, bm)

Joining, by = "ensembl_gene_id"

Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining character vector and factor, coercing into character vector

> hpa <- hpa[!duplicated(hpa$uniprotswissprot), ]
>
> ## match HPA/LOPIT
> colnames(hpa)[7] <- "accession"
> fd <- left_join(fData(andy2011), hpa)

Joining, by = "accession"

Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining character vector and factor, coercing into character vector

> rownames(fd) <- featureNames(andy2011)
Below, we deparse the multiple ‘;’-delimited locations contained in the Human Protein sub-cellular Atlas, create the auxiliary binary data matrix (only localisations with reliability equal to Supportive are considered; Uncertain assignments are ignored - see http://www.proteinatlas.org/about/quality+scoring#ifr for details) and filter proteins without any localisation data.

```r
## HPA localisation
hpalocs <- c(as.character(fData(lopit)$Main.location),
              as.character(fData(lopit)$Other.location))
hpalocs <- hpalocs[!is.na(hpalocs)]
hpalocs <- unique(unlist(strsplit(hpalocs, ";")))

makeHpaSet <- function(x, score2, locs = hpalocs) {
  hpamat <- matrix(0, ncol = length(locs), nrow = nrow(x))
  colnames(hpamat) <- locs
  rownames(hpamat) <- featureNames(x)
  for (i in 1:nrow(hpamat)) {
    loc <- unlist(strsplit(as.character(fData(x)[i, "Main.location"]), ";"))
    loc2 <- unlist(strsplit(as.character(fData(x)[i, "Other.location"]), ";"))
    score <- score2[fData(x)[i, "Reliability"]]
    hpamat[i, loc] <- score
    hpamat[i, loc2] <- score
  }
  new("MSnSet", exprs = hpamat,
       featureData = featureData(x))
}

hpaset <- makeHpaSet(lopit,
                      score2 = c(Supportive = 1, Uncertain = 0))
hpaset <- filterZeroRows(hpaset)

Removing 319 columns with only 0s.

> dim(hpaset)
[1] 668 18
> exprs(hpaset)[c(1, 6, 200), 1:3]
          Endoplasmic reticulum Cytoplasm Vesicles
    000767            1          0        0
    095302            0          0        1
```
2.3 Protein-protein interactions

Protein-protein interaction data can also be used as auxiliary data input to the transfer learning algorithm. Several sources can be used to do so directly from R:

- The **PSICQUIC** package provides an R interface to the HUPO Proteomics Standard Initiative (HUPO-PSI) project, which standardises programmatic access to molecular interaction databases. This approach enables to query great many resources in one go but, as noted in the vignettes, for bulk interactions, it is recommended to directly download databases from individual PSICQUIC providers.
- The **STRINGdb** package provides a direct interface to the STRING protein-protein interactions database. This package can be used to generate a table as the one used below. The exact procedure is described in the STRINGdb vignette and involves mapping the protein identifiers with the `map` and retrieve the interaction partners with the `get_neighbors` method.
- Finally, it is possible to use any third-party PPI inference results and adequately prepare these results for transfer learning. Below, we will described this case with PPI data in a tab-delimited format, as retrieved directly from the STRING database.

Below, we access the PPI spreadsheet file for our test data, that is distributed with the *pRolocdata* package.

```r
> ppif <- system.file("extdata/tabdelimited._gHentss2F9k.txt.gz", package = "pRolocdata")
> ppidf <- read.delim(ppif, header = TRUE, stringsAsFactors = FALSE)
> head(ppidf)

<table>
<thead>
<tr>
<th>X.node1</th>
<th>node2</th>
<th>node1_string_id</th>
<th>node2_string_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUDT5</td>
<td>IMPDH2</td>
<td>1861432</td>
<td>1850365</td>
</tr>
<tr>
<td>NOP2</td>
<td>RPL23</td>
<td>1858730</td>
<td>1858184</td>
</tr>
<tr>
<td>HSPA4</td>
<td>ENO1</td>
<td>1848476</td>
<td>1843405</td>
</tr>
<tr>
<td>RPS13</td>
<td>DKC1</td>
<td>1862013</td>
<td>1855055</td>
</tr>
<tr>
<td>RPL35A</td>
<td>DDOST</td>
<td>1859718</td>
<td>1856225</td>
</tr>
<tr>
<td>RPL13A</td>
<td>RPS6</td>
<td>1857955</td>
<td>1857216</td>
</tr>
</tbody>
</table>
```

```
node1_external_id node2_external_id neighborhood
1 ENSP00000419628 ENSP00000321584 0.000
2 ENSP00000382392 ENSP00000377865 0.000
3 ENSP00000302961 ENSP00000234590 0.000
4 ENSP00000435777 ENSP00000358563 0.462
5 ENSP00000393393 ENSP00000364188 0.000
6 ENSP00000375730 ENSP00000369757 0.000

fusion cooccurence homology coexpression
1 0 0 0 0.112
2 0 0 0 0.064
3 0 0 0 0.109
```
The file contains 18623 pairwise interactions and the STRING combined interaction score. Below, we create a contingency matrix that uses these scores to encode and weight interactions.

```r
> uid <- unique(c(ppidf$X.node1, ppidf$node2))
> ppim <- diag(length(uid))
> colnames(ppim) <- rownames(ppim) <- uid

> for (k in 1:nrow(ppidf)) {
+   i <- ppidf[k, "X.node1"]
+   j <- ppidf[k, "node2"]
+   ppim[i, j] <- ppidf[k, "combined_score"]
+
+
> ppim[1:5, 1:8]

NUDT5 NOP2 HSPA4 RPS13 RPL35A RPL13A CPS1
NUDT5  1  0  0  0  0.000  0.000  0
NOP2   0  1  0  0  0.000  0.000  0
HSPA4  0  0  1  0  0.000  0.000  0
RPS13  0  0  0  1  0.997  0.998  0
RPL35A 0  0  0  0 1.000  0.999  0
CTNNB1

NUDT5  0
NOP2   0
HSPA4  0
RPS13  0
RPL35A 0
```

We now have a contingency matrix reflecting a total of 19910 interactions between 1287 proteins. Below, we only keep proteins that are also available in our spatial proteomics data, subset the PPI and LOPIT data, create the appropriate MSnSet object, and filter out proteins without any interaction scores.

```r
> featureNames(andy2011) <- sub("_HUMAN", "", fData(andy2011)$UniProtKB.entry.name)
> cmn <- intersect(featureNames(andy2011), rownames(ppim))
> ppim <- ppim[cmn, ]
```
We now have two `MSnSet` objects containing respectively 520 primary experimental protein profiles along a sub-cellular density gradient (`andy2001`) and 520 auxiliary interaction profiles (`ppi`).

### 3 Support vector machine transfer learning

The SVM-TL method described in [7] has not yet been incorporated in the `pRoloc` package. The code implementing the method is currently available in its own repository:

https://github.com/ComputationalProteomicsUnit/lpsvm-tl-code

### 4 Nearest neighbour transfer learning

#### 4.1 Optimal weights

The weighted nearest neighbours transfer learning algorithm estimates optimal weights for the different data sources and the spatial niches described for the data at hand with the `knnt1Optimisation` function. For instance, for the human data modelled by the `andy2011` and `andygoset` objects and the 10 annotated sub-cellular localisations (Golgi, Mitochondrion, PM, Lysosome, Cytosol, Cytosol/Nucleus, Nucleus, Ribosome 60S, Ribosome 40S and ER), we want to know how to optimally combine primary and auxiliary data. If we look at figure 2, that illustrates the experimental separation of the 10 spatial classes on a principal component plot, we see that some organelles such as the mitochondrion or the cytosol and cytosol/nucleus are well resolved, while others such as the Golgi or the ER are less so. In this experiment, the former classes are not expected to benefit from another data source, while the latter should benefit from additional information.

Let’s define a set of three possible weights: 0, 0.5 and 1. A weight of 1 indicates that the final results rely exclusively on the experimental data and ignore completely the auxiliary data. A weight of 0 represents the opposite situation, where the primary data is ignored and only the auxiliary data is considered. A weight of 0.5 indicates that each data source will contribute equally to the final results. It is the algorithm’s optimisation step task to identify the optimal combination of class-specific weights for a given primary and auxiliary data pair. The optimisation process can be quite time consuming for many weights and many sub-cellular classes, as all combinations (there are \( \text{number of classes} \times \text{number of weights} \) possibilities; see below). One would generally defined more weights (for example 0, 0.25, 0.5, 0.75, 1).

---

⁴We will use the sub-cellular markers defined in the `markers.tl` feature variable, instead of the default `markers`.
1 or 0, 0.33, 0.67, 1) to explore more fine-grained integration opportunities. The possible weight combinations can be calculated with the `thetas` function:

- 3 classes, 3 weights

```r
> head(thetas(3, by = 0.5))
Weights:
(0, 0.5, 1)
[,1]  [,2]  [,3]
```
The actual combination of weights to be tested can be defined in multiple ways: by passing a weights matrix explicitly (as those generated with `thetas` above) through the `th` argument; or by defining the increment between weights using `by`; or by specifying the number of weights to be used through the `length.out` argument.

Considering the sub-cellular resolution for this experiment, we would anticipate that the mitochondrion, the cytosol and the cytosol/nucleus classes would get high weights, while the ER and Golgi would be assigned lower weights.

As we use a nearest neighbour classifier, we also need to know how many neighbours to consider when classifying a protein of unknown localisation. The `knnOptimisation` function (see the `pRoloc` tutorial vignette and the functions manual page) can be run on the primary and auxiliary data sources independently to estimate the best $k_P$ and $k_A$ values. Here, based on `knnOptimisation`, we use 3 and 3, for $k_P$ and $k_A$ respectively.

Finally, to assess the validity of the weight selection, it should be repeated a certain number of times (default value is 50). As the weight optimisation can become very time consuming for a wide range of weights and many target classes, we would recommend to start with a lower number of iterations, pre-analyse the results, proceed with further iterations and eventually combine the optimisation results data with the `combineThetaRegRes` function before proceeding with the selection of best weights.
The above code chunk would take too much time to be executed in the frame of this vignette. Below, we pass a very small subset of theta matrix to minimise the computation time. The \texttt{knntlOptimisation} function supports parallelised execution using various backends thanks to the \texttt{BiocParallel} package; an appropriate backend will be defined automatically according to the underlying architecture and user-defined backends can be defined through the \texttt{BPPARAM} argument\(^5\). Also, in the interest of time, the weights optimisation is repeated only 5 times below.

\begin{verbatim}
> set.seed(1)
> i <- sample(nrow(th), 12)
> topt <- knntlOptimisation(andy2011, andygoset,
+     th = th[i, ],
+     k = c(3, 3),
+     fcol = "markers.tl",
+     times = 5)

Removing 428 columns with only 0s.

Note: vector will be ordered according to classes: Cytosol Cytosol/Nucleus ER Golgi Lysosome Mitochondrion Nucleus PM Ribosome 40S Ribosome 60S (as names are not explicitly defined)

> topt
Object of class "ThetaRegRes"
Algorithm: theta
Theta hyper-parameters:
  weights: 0 0.3333333 0.6666667 1
  k: 3 3
  nrow: 12
Design:
  Replication: 5 x 5-fold X-validation
  Partitioning: 0.2/0.8 (test/train)
Results
  macro F1:
    Min. 1st Qu. Median Mean 3rd Qu. Max.
  0.7813 0.8021 0.8580 0.8485 0.8829 0.9181
  best theta:
        Cytosol Cytosol.Nucleus ER Golgi Lysosome
weight:0 1 0 3 2 0
\end{verbatim}

\(^5\)Large scale applications of this algorithms (\textit{fixme: add ref}) were run on a cluster using an MPI backend defined with \texttt{SnowParams(256, type="MPI")}.  

The optimisation is performed on the labelled marker examples only. When removing unlabelled non-marker proteins (the unknowns), some auxiliary GO columns end up containing only 0 (the GO-protein association was only observed in non-marker proteins), which are temporarily removed.

The `topt` result stores all the result from the optimisation step, and in particular the observed theta weights, which can be directly plotted as shown on figure 3. These bubble plots give the proportion of best weights for each marker class that was observed during the optimisation phase. We see that the mitochondrion, the cytosol and cytosol/nucleus classes predominantly are scored with height weights (2/3 and 1), consistent with high reliability of the primary data. The Golgi and the ribosomal clusters (and to a lesser extend the ER) favour smaller scores, indicating a substantial benefit of the auxiliary data.

### 4.2 Choosing weights

A set of best weights must be chosen and applied to the classification of the unlabelled proteins (formally annotated as unknown). These can be defined manually, based on the pattern observed in the weights bubble plot (figure 3), or automatically extracted with the `getParams` method. See [getParams](#getParams) for details and the `favourPrimary` function, if it is desirable to systematically favour the primary data (i.e. high weights) when different weight combinations perform equally well.

```r
> getParams(topt)
     Cytosol Cytosol/Nucleus ER
  1.0000000  1.0000000  0.0000000
Golgi Lysosome Mitochondrion
  0.3333333  0.6666667  0.3333333
Nucleus PM Ribosome 40S
  0.3333333  0.3333333  0.6666667
Ribosome 60S
  0.6666667
```

Note that the scores extracted here are based on the random subset of weights.
Figure 3: Results obtained from an extensive optimisation on the primary andy2011 and auxiliary andygoset data sets, as produced by plot(topt). This figure is not the result for the previous code chunk, where only a random subset of 10 candidate weights have been tested.

We provide the best parameters for the extensive parameter optimisation search, as provided by getParams:

```r
> (bw <- experimentData(andy2011)@other$knntl$thetas)

    Cytosol Cytosol/Nucleus    ER
    0.6666667   0.6666667 0.3333333
    Golgi  Lysosome   Mitochondrion
    0.3333333  0.6666667 0.6666667
   Nucleus    PM  Ribosome 40S
   0.3333333 0.3333333          0.0000000
Ribosome 60S
   0.3333333
```
4.3 Applying best theta weights

To apply our best weights and learn from the auxiliary data accordingly when classifying the unlabelled proteins to one of the sub-cellular niches considered in markers.tl (as displayed on figure 2), we pass the primary and auxiliary data sets, best weights, best k’s (and, on our case the marker’s feature variable we want to use, default would be markers) to the knntlClassification function.

```r
> andy2011 <- knntlClassification(andy2011, andygoset,
+                                   bestTheta = bw,
+                                   k = c(3, 3),
+                                   fcol = "markers.tl")
```

This will generate a new instance of class MSnSet, identical to the primary data, including the classification results and classifications scores of the transfer learning classification algorithm (as knntl and knntl.scores feature variables respectively). Below, we extract the former with the getPrediction function and plot the results of the classification.

```r
> andy2011 <- getPredictions(andy2011, fcol = "knntl")
```

<table>
<thead>
<tr>
<th></th>
<th>Chromatin associated</th>
<th>Cytosol</th>
<th>Cytosol/Nucleus</th>
<th>ER</th>
<th>Endosome</th>
<th>Golgi</th>
<th>Lysosome</th>
<th>Mitochondrion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proteins</td>
<td>11</td>
<td>277</td>
<td>56</td>
<td>210</td>
<td>12</td>
<td>65</td>
<td>61</td>
<td>258</td>
</tr>
<tr>
<td>PM</td>
<td>108</td>
<td>236</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ribosome 40S</td>
<td>18</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ribosome 60S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please read the pRoloc-tutorial vignette, and in particular the classification section, for more details on how to proceed with exploration the classification results and classification scores.

5 Conclusions

This vignette describes the application of a weighted k-nearest neighbour transfer learning algorithm and its application to the sub-cellular localisation prediction of proteins using quantitative proteomics data as primary data and Gene Ontology-derived binary data as auxiliary data source. The algorithm can be used with various data sources (we show how to compile binary data from the Human Protein Atlas in section 2.2) and have successfully applied the algorithm [7] on third-party quantitative auxiliary data.
Session information

All software and respective versions used to produce this document are listed below.

- R version 3.4.0 (2017-04-21), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Running under: Ubuntu 16.04.2 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.38.0, Biobase 2.36.2, BiocGenerics 0.22.0, BiocParallel 1.10.1, BiocStyle 2.4.0, GO.db 3.4.1, IRanges 2.10.0, MLInterfaces 1.56.0, MSnbase 2.2.0, ProtGenerics 1.8.0, Rcpp 0.12.10, S4Vectors 0.14.0, XML 3.98-1.7, annotate 1.54.0, biomaRt 2.32.0, class 7.3-14, cluster 2.0.6, dplyr 0.5.0, hpar 1.18.0, knitr 1.15.1, mzR 2.10.0, pRoloc 1.16.1, pRolocdata 1.14.0, xtable 1.8-2
- Loaded via a namespace (and not attached): BiocInstaller 1.26.0, DBI 0.6-1, DEoptimR 1.0-8, FNN 1.1, MALDIquant 1.16.2, MASS 7.3-47, Matrix 1.2-10, MatrixModels 0.4-1, ModelMetrics 1.1.0, R6 2.2.0, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.1-2, SparseM 1.77, affy 1.54.0, affyio 1.46.0, assertthat 0.2.0, backports 1.0.5, base64enc 0.1-3, bitops 1.0-6, car 2.1-4, caret 6.0-76, codetools 0.2-15, colorspace 1.3-2, compiler 3.4.0, dextrase 1.5.2, digest 0.6.12, doParallel 1.0.10, e1071 1.6-8, evaluate 0.10, flexmix 2.3-14, foreach 1.4.3, fpc 2.1-10, gbm 2.1.3, gdata 2.17.0, genefilter 1.58.1, ggplot2 2.2.1, ggvis 0.4.3, grid 3.4.0, gridExtra 2.2.1, gtable 0.2.0, gtools 3.5.0, hexbin 1.27.1, highr 0.6, htmltools 0.3.6, htmlwidgets 0.8, httpuv 1.3.3, hwriter 1.3.2, impute 1.50.1, iterators 1.0.8, jsonlite 1.4, kernlab 0.9-25, lattice 0.20-35, lazyeval 0.2.0, limma 3.32.2, lme4 1.1-13, lpSolve 5.6.13, magrittr 1.5, mclust 5.2.3, memoise 1.1.0, mgcv 1.8-17, mime 0.5, minqa 1.2.4, mlbench 2.1-1, modeltools 0.2-21, munsell 0.4.3, mvtnorm 1.0-6, mzdID 1.14.0, nlme 3.1-131, nloptr 1.0.4, nnet 7.3-12, pbkrtest 0.4-7, pcaMethods 1.68.0, pls 2.6-0, plyr 1.8.4, prabclus 2.2-6, preprocessCore 1.38.1, proxy 0.4-17, quantreg 5.33, randomForest 4.6-12, rda 1.0-2-2, reshape2 1.2.2, rmarkdown 1.5, rmeta 0.9-2, rpart 4.1-11, rprojroot 1.2, sampling 2.8, scales 0.4.1, sfsmisc 1.1-0, shiny 1.0.3, splines 3.4.0, stringi 1.1.5, stringr 1.2.0, survival 2.41-3, threejs 0.2.2, tibble 1.3.0, tools 3.4.0, trimcluster 0.1-2, viridis 0.4.0, viridisLite 0.2.0, vsn 3.44.0, whisker 0.3-2, yam 2.1.14, zlibbioc 1.22.0

References


> setStockcol(paste0(getStockcol(), "80"))
> ptsze <- exp(fData(andy2011)$knntl.scores) - 1
> plot2D(andy2011, fcol = "knntl", cex = ptsze)
> setStockcol(NULL)
> addLegend(andy2011, where = "topright",
+   fcol = "markers.tl",
+   bty = "n", cex = .7)

Figure 4: PCA plot of andy2011 after transfer learning classification. The size of the points is proportional to the classification scores.