A transfer learning algorithm for spatial proteomics

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

June 14, 2016

Abstract

This vignette illustrates the application of a *transfer learning* algorithm to assign proteins to sub-cellular localisations. The *knntlClassification* 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

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

> Breckels LM, Holden S, Wonjar D, Mulvey CM, Christoforou A, Groen A, Kohlbacker O, Lilley KS and Gatto L (2015). "Learning from heterogeneous data sources: an application in spatial proteomics." *bioRxiv*.

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`.

```
> 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.

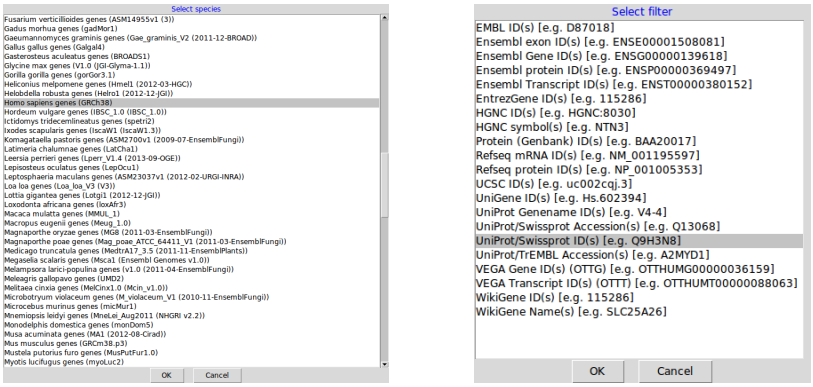

Figure 1: Selecting species (left) and feature type (right) to create an `AnnotationParams` instance for the human `andy2011` data.

It is also possible to pass patterns[1] to match against the species (`"Homo sapiens"`) and feature type (`"UniProt/Swissprot ID"`).

```
> ap <- setAnnotationParams(inputs =
+                             c("Homo sapiens",
+                               "UniProt/Swissprot Accession"))

Using species Homo sapiens genes (GRCh38.p5)

Using feature type UniProt/Swissprot Accession(s) [e.g.  Q13068]

Connecting to Biomart...

> 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 'uniprot_swissprot' as filter
 Created on Tue Jun 14 20:36:14 2016
```

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.

```
> 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[2].

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

MSnSet (storageMode: lockedEnvironment)
assayData: 1371 features, 751 samples
  element names: exprs
protocolData: none
phenoData: none
featureData
  featureNames: O00767 P51648 ... O75312 (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.

```
Constructed GO set using cellular_component namespace: Tue Jun 14 20:36:24 2016
 MSnbase version: 1.20.7
> exprs(andygoset)[1:7, 1:4]
       GO:0005783 GO:0005789 GO:0016020 GO:0016021
O00767          1          1          1          1
P51648          1          1          0          1
Q2TAA5          0          1          1          1
Q9UKV5          0          1          1          1
Q12797          1          1          1          1
P16615          1          1          1          1
Q96SQ9          1          1          0          0
```

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 751 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[3].

```
> 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 <- "uniprot_swissprot"
> attrib <- c("uniprot_genename", "uniprot_swissprot", "ensembl_gene_id")
> bm <- getBM(attributes = attrib,
```

[3]This section has been updated on the 30th November to reflect changes in Biomart.

```
+             filters = filter,
+             values = fData(andy2011)[, "accession"],
+             mart = mart)
> head(bm)

  uniprot_genename uniprot_swissprot ensembl_gene_id
1                              A0FGR8 ENSG00000117868
2           A1L0T0             A1L0T0 ENSG00000105135
3           A2RRP1             A2RRP1 ENSG00000151779
4           A4FU01             A4FU01 ENSG00000014914
5           A6NCS6             A6NCS6 ENSG00000204128
6           A6NKG5             A6NKG5 ENSG00000254656

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

This is hpar 1.14.2.  For more information,
please type '?hpar' or 'vignette('hpar')'.

> getHpaVersion()

[1] "Protein Atlas version 13"

> getHpaDate()

[1] "2014.11.06"

> setHparOptions(hpadata = "SubcellularLoc")
> 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):  joining character vector and factor,
coercing into character vector

> hpa <- hpa[!duplicated(hpa$uniprot_swissprot), ]
>
> ## 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):  joining character vector and factor,
coercing into character vector
```

```
> rownames(fd) <- featureNames(andy2011)
> fData(andy2011) <- fd
> stopifnot(validObject(andy2011))
>
> ## Let's get rid of features without any hpa data
> lopit <- andy2011[!is.na(fData(andy2011)$Main.location), ]
```

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.

```
> ## 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(lopit))
+ }
>
> 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
O00767                     1         0        0
```

```
O95302                    0         0         1
P06493                    0         1         0
```

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 interfaces 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.

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

```
  X.node1  node2 node1_string_id node2_string_id
1   NUDT5 IMPDH2         1861432         1850365
2    NOP2  RPL23         1858730         1858184
3   HSPA4   ENO1         1848476         1843405
4   RPS13   DKC1         1862013         1855055
5  RPL35A  DDOST         1859718         1856225
6  RPL13A   RPS6         1857955         1857216
  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
4      0           0        0        0.202
5      0           0        0        0.000
6      0           0        0        0.931
  experimental knowledge textmining combined_score
1        0.000       0.0      0.370          0.416
2        0.868       0.0      0.000          0.871
3        0.222       0.0      0.436          0.575
4        0.000       0.0      0.354          0.698
5        0.000       0.9      0.265          0.923
6        0.419       0.9      0.240          0.996
```

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.

```
> 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.

```
> featureNames(andy2011) <- sub("_HUMAN", "", fData(andy2011)$UniProtKB.entry.name)
> cmn <- intersect(featureNames(andy2011), rownames(ppim))
```

```
> ppim <- ppim[cmn, ]
> andy2011 <- andy2011[cmn, ]
>
> ppi <- MSnSet(ppim, fData = fData(andy2011),
+               pData = data.frame(row.names = colnames(ppim)))
> ppi <- filterZeroCols(ppi)
```

Removing 178 columns with only 0s.

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 descibed in [2] 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 `knntlOptimisation` function. For instance, for the human data modelled by the `andy2011` and `andygoset` objects[4] 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 $number\ of\ classes^{number\ of\ weights}$

[4] We will use the sub-cellular markers defined in the `markers.tl` feature variable, instead of the default `markers`.

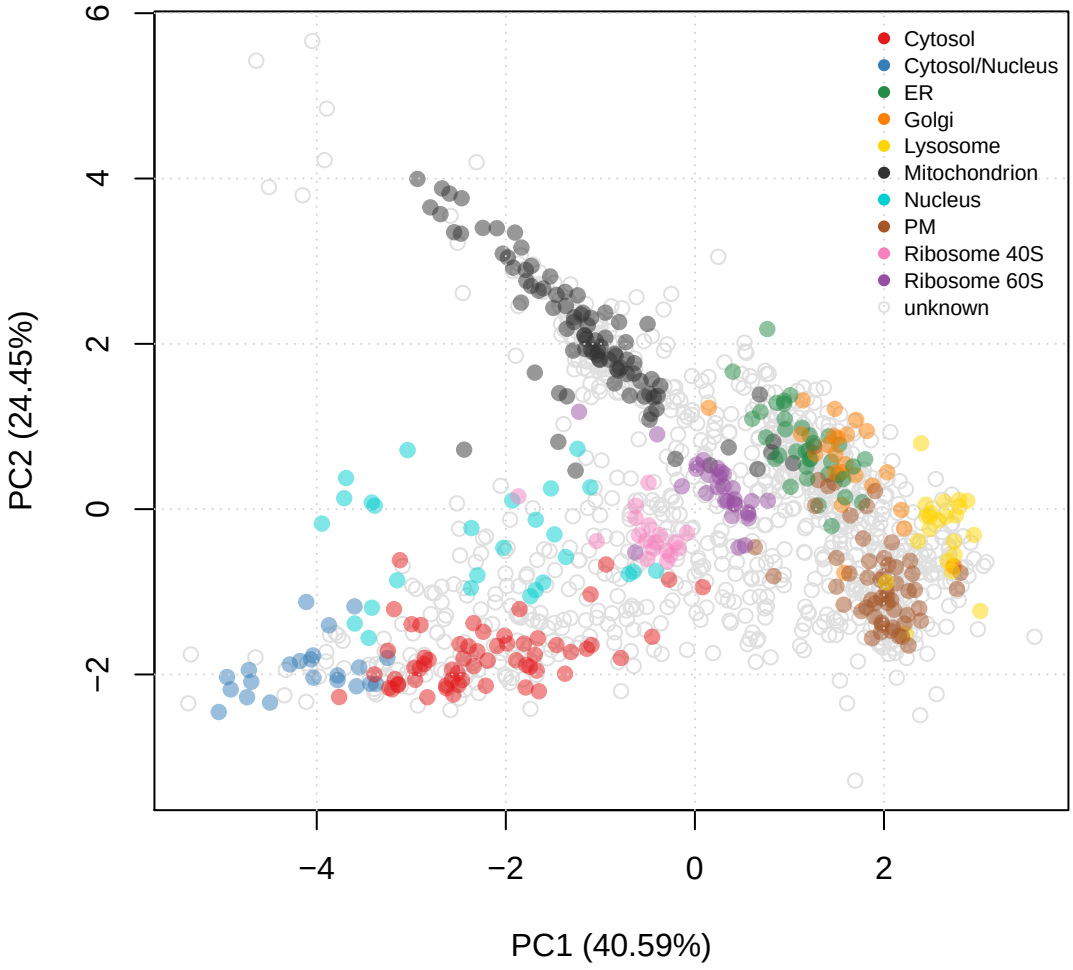

Figure 2: PCA plot of `andy2011`. The multivariate protein profiles are summarised along the two first principal components. Proteins of unknown localisation are represented by empty grey points. Protein markers, which are well-known residents of specific sub-cellular niches are colour-coded and form clusters on the figure.

possibilities; see below). One would generally defined more weights (for example 0, 0.25, 0.5, 0.75, 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

```
> head(thetas(3, by = 0.5))
Weigths:
(0, 0.5, 1)
```

```
     [,1] [,2] [,3]
[1,]    0  0.0  0.0
[2,]    0  0.0  0.5
[3,]    0  0.0  1.0
[4,]    0  0.5  0.0
[5,]    0  0.5  0.5
[6,]    0  0.5  1.0
> dim(thetas(3, by = 0.5))
Weigths:
(0, 0.5, 1)
[1] 27  3
```

- 5 classes, 4 weights

```
> dim(thetas(5, length.out = 4))
Weigths:
(0, 0.333333333333333, 0.666666666666667, 1)
[1] 1024    5
```

- for the human andy2011 data, considering 4 weights, there are very many combinations:

```
> ## marker classes for andy2011
> m <- unique(fData(andy2011)$markers.tl)
> m <- m[m != "unknown"]
> th <- thetas(length(m), length.out=4)
Weigths:
(0, 0.333333333333333, 0.666666666666667, 1)
> dim(th)
[1] 1048576      10
```

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.

```
> topt <- knntlOptimisation(andy2011, andygoset,
+                           th = th,
+                           k = c(3, 3),
+                           fcol = "markers.tl",
+                           times = 50)
```

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 `knntlOptimisation` function supports parallelised execution using various backends thanks to the *BiocParallel* package; an appropriate backend will be defined automatically according to the underlying architecture and user-defined backends can be defined through the `BPPARAM` argument[5]. Also, in the interest of time, the weights optimisation is repeated only 5 times below.

```
> 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 406 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.
 0.8530  0.8531  0.8632  0.8726  0.8781  0.9159
 best theta:
            Cytosol Cytosol.Nucleus ER Golgi Lysosome
```

[5] Large scale applications of this algorithms (*fixme: add ref*) were run on a cluster using an MPI backend defined with `SnowParams(256, type="MPI")`.

```
weight:0          0               0  3     0        3
weight:0.33       0               0  2     5        0
weight:0.67       3               5  0     0        0
weight:1          2               0  0     0        2
            Mitochondrion Nucleus PM Ribosome.40S
weight:0                3       0  2            0
weight:0.33             0       0  3            3
weight:0.67             0       0  0            2
weight:1                2       5  0            0
            Ribosome.60S
weight:0               3
weight:0.33            2
weight:0.67            0
weight:1               0
Use getWarnings() to see warnings.
```

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[6]. See `?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.

```
> getParams(topt)

       Cytosol Cytosol/Nucleus              ER
     0.6666667       0.6666667       0.0000000
         Golgi        Lysosome   Mitochondrion
     0.3333333       0.0000000       0.0000000
       Nucleus              PM    Ribosome 40S
```

[6]Note that the scores extracted here are based on the random subsest of weights.

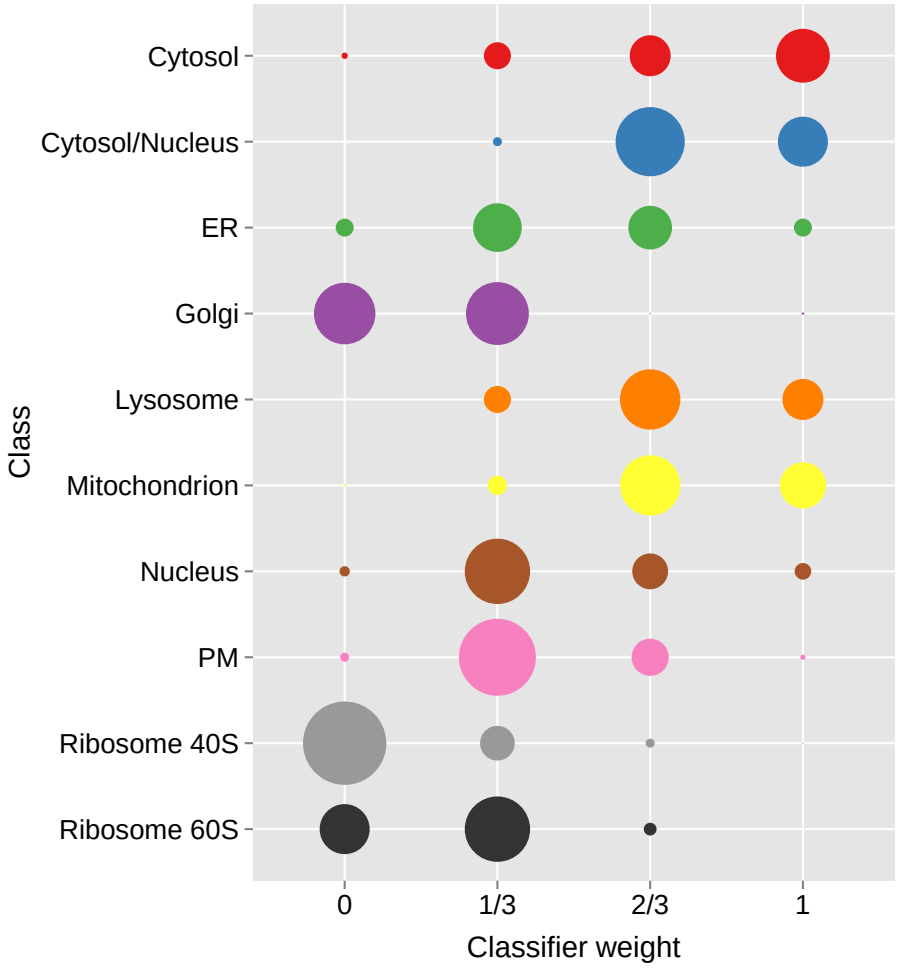

Figure 3: Results obtained from an extensive optimisation on the primary `andy2011` and auxiliary `andygoset` data sets, as produced by `plot(topt)`. This figure is not the result for the previous code chunk, where only a random subset of 10 candidate weights have been tested.

```
      1.0000000       0.3333333       0.3333333
   Ribosome 60S
      0.0000000
```

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

```
> (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.

```
> 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.

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

ans
Chromatin associated              Cytosol
                  11                  305
     Cytosol/Nucleus                   ER
                  48                  200
            Endosome                Golgi
                  12                   60
            Lysosome         Mitochondrion
                  71                  262
             Nucleus                   PM
                 109                  217
        Ribosome 40S          Ribosome 60S
                  19                   57
```

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 [2] on third-party quantitative auxiliary data.

Session information

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

- R version 3.3.0 (2016-05-03), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.34.3, Biobase 2.32.0, BiocGenerics 0.18.0, BiocParallel 1.6.2, BiocStyle 2.0.2, GO.db 3.3.0, IRanges 2.6.0, MLInterfaces 1.52.0, MSnbase 1.20.7, ProtGenerics 1.4.0, Rcpp 0.12.5, S4Vectors 0.10.1, XML 3.98-1.4, annotate 1.50.0, biomaRt 2.28.0, class 7.3-14, cluster 2.0.4, dplyr 0.4.3, hpar 1.14.2, knitr 1.13, mzR 2.6.2, pRoloc 1.12.4, pRolocdata 1.10.0, xtable 1.8-2
- Loaded via a namespace (and not attached): BiocInstaller 1.22.2, DBI 0.4-1, DEoptimR 1.0-4, FNN 1.1, MALDIquant 1.14, MASS 7.3-45, Matrix 1.2-6, MatrixModels 0.4-1, R6 2.1.2, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, SparseM 1.7, affy 1.50.0, affyio 1.42.0, assertthat 0.1, base64enc 0.1-3, bitops 1.0-6, car 2.1-2, caret 6.0-70, codetools 0.2-14, colorspace 1.2-6, digest 0.6.9, diptest 0.75-7, doParallel 1.0.10, e1071 1.6-7, evaluate 0.9, flexmix 2.3-13, foreach 1.4.3, formatR 1.4, fpc 2.1-10, gbm 2.1.1, gdata 2.17.0, genefilter 1.54.2, ggplot2 2.1.0, ggvis 0.4.2, grid 3.3.0, gtable 0.2.0, gtools 3.5.0, highr 0.6, htmltools 0.3.5, htmlwidgets 0.6, httpuv 1.3.3, hwriter 1.3.2, impute 1.46.0, iterators 1.0.8, jsonlite 0.9.21, kernlab 0.9-24, lattice 0.20-33, limma 3.28.8, lme4 1.1-12, lpSolve 5.6.13, magrittr 1.5, mclust 5.2, mgcv 1.8-12, mime 0.4, minqa 1.2.4, mlbench 2.1-1, modeltools 0.2-21, munsell 0.4.3, mvtnorm 1.0-5, mzID 1.10.2, nlme 3.1-128, nloptr 1.0.4, nnet 7.3-12, pbkrtest 0.4-6, pcaMethods 1.64.0, pls 2.5-0, plyr 1.8.4, prabclus 2.2-6, preprocessCore 1.34.0, proxy 0.4-15, quantreg 5.26, randomForest 4.6-12, rda 1.0.2-2, reshape2 1.4.1, rmarkdown 0.9.6, robustbase 0.92-6, rpart 4.1-10, sampling 2.7, scales 0.4.0, sfsmisc 1.1-0, shiny 0.13.2, splines 3.3.0, stringi 1.1.1, stringr 1.0.0, survival 2.39-4, threejs 0.2.2, tools 3.3.0, trimcluster 0.1-2, vsn 3.40.0, yaml 2.1.13, zlibbioc 1.18.0

References

[1] Laurent Gatto, Juan Antonio Vizcaíno, Henning Hermjakob, Wolfgang Huber, and Kathryn S Lilley. Organelle proteomics experimental designs and analysis. *Proteomics*, 2010. doi:10.1002/pmic.201000244.

[2] L.M. Breckels, S. Holden, D. Wonjar, C.M Mulvey, A. Christoforou, A. Groen, O. Kohlbacker, K.S. Lilley, and L. Gatto. Learning from heterogeneous data sources: an application in spatial

proteomics. *bioRxiv*, 2015. URL: http://biorxiv.org/content/early/2015/07/07/022152, doi:10.1101/022152.

[3] Lisa M Breckels, Laurent Gatto, Andy Christoforou, Arnoud J Groen, Kathryn S Lilley, and Matthew W B Trotter. The effect of organelle discovery upon sub-cellular protein localisation. *J Proteomics*, Mar 2013. doi:10.1016/j.jprot.2013.02.019.

[4] L Gatto, L M Breckels, T Burger, D J Nightingale, A J Groen, C Campbell, N Nikolovski, C M Mulvey, A Christoforou, M Ferro, and K S Lilley. A foundation for reliable spatial proteomics data analysis. *Mol Cell Proteomics*, 13(8):1937–52, Aug 2014. doi:10.1074/mcp.M113.036350.

[5] M Ashburner, C A Ball, J A Blake, D Botstein, H Butler, J M Cherry, A P Davis, K Dolinski, S S Dwight, J T Eppig, M A Harris, D P Hill, L Issel-Tarver, A Kasarskis, S Lewis, J C Matese, J E Richardson, M Ringwald, G M Rubin, and G Sherlock. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nat Genet*, 25(1):25–9, May 2000. doi:10.1038/75556.

[6] Mathias Uhlen, Per Oksvold, Linn Fagerberg, Emma Lundberg, Kalle Jonasson, Mattias Forsberg, Martin Zwahlen, Caroline Kampf, Kenneth Wester, Sophia Hober, Henrik Wernerus, Lisa Björling, and Fredrik Ponten. Towards a knowledge-based Human Protein Atlas. *Nature biotechnology*, 28(12):1248–1250, December 2010. URL: http://dx.doi.org/10.1038/nbt1210-1248, doi:10.1038/nbt1210-1248.

```
> 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)
```

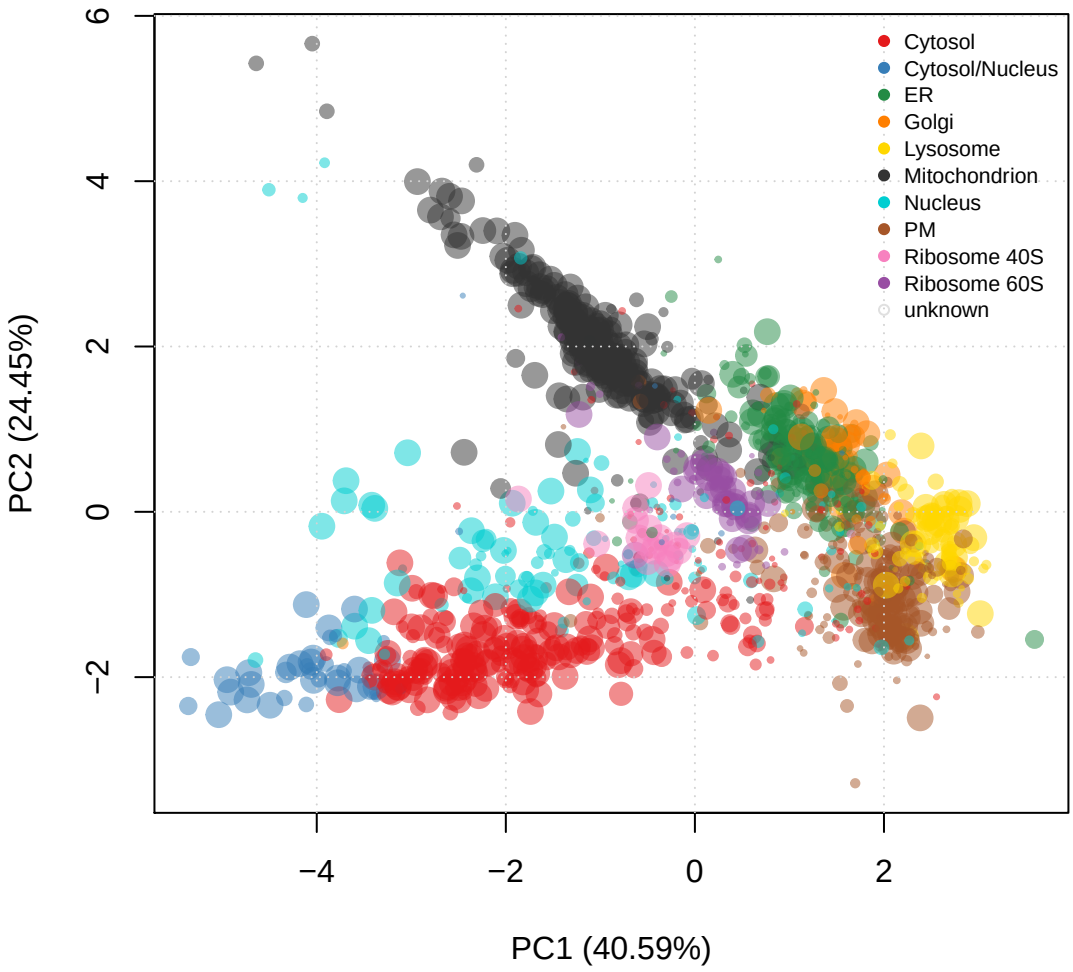

Figure 4: PCA plot of andy2011 after transfer learning classification. The size of the points is proportional to the classification scores.