Introduction to the \textit{PRISMA} package

Tammo Krueger

March 15, 2015

https://github.com/tammok/PRISMA

Introduction

This vignette gives you a first tour to the features of the \textit{PRISMA} package. We will give an overview of the application of the algorithm, yet, the full story is available in the papers [3, 4]. If you use the \textit{PRISMA} package in your research, please cite at least one of these references.

The \textit{PRISMA} package consists essentially out of three parts:

1. Efficiently reading \texttt{sally} output, an extremely fast n-gram processor available at \url{http://www.mlsec.org/sally/}
2. Testing-based feature dimension reduction
3. Optimized matrix factorization of the reduced data exploiting the replicate structure of the data

For the theory behind these parts please consult [3, 4]. We will start this walk-through with the reading of \texttt{sally} data, then showing the inner structure of the resulting data object on which the replicate-aware non-negative matrix factorization can be applied.

Loading the Data

This section serves just as a reference how to apply the processing chain to new data, to get a usable \textit{PRISMA} data set. The generated data set is already prepackaged inside the \textit{PRISMA} package and can be loaded via \texttt{data(asap)}.

Before executing the examples please extract \texttt{asap.tar.gz} located in the \texttt{extdata} path of the \textit{PRISMA} package to find all data necessary to understand the processing chain from the raw data (asap.raw) to the \texttt{sally} file (asap.sally) and the optimized file (asap.fsally). The asap.sally file can be produced as follows:

\texttt{sally -c asap.cfg asap.raw asap.sally}

this call generates asap.sally from the raw data found in asap.raw. To speed up the loading of the data in R, one should apply the \texttt{sallyPreprocessing.py} Python script as follows:

\texttt{python sallyPreprocessing.py asap.sally asap.fsally}

Now the data is ready to be efficiently loaded and processed in R via \texttt{loadPrismaData(“asap”)} which also executes the feature dimension reduction step.

The \textit{PRISMA} Data Set

As an example we use the prepackages ASAP toy data set as described in [4]:

\begin{verbatim}
> data(asap)
> asap

PRISMA data asap
Unprocessed data: # features: 10034 # entries: 10000
Processed data: # features: 12 # entries: 24
\end{verbatim}
We see that the feature reduction step worked quite well. Let’s have a look behind the scenes:

```r
> asap$data
12 x 24 sparse Matrix of class "dgCMatrix"
rv Gecko 1.8.1.3 Mozilla Firefox 2.0.0.3 5.0 20070309 1 1 1 1 1 1 1 1 1 1 1 1 1
admin.php par action 1 1 1 1 1 1 1 1 1 1 1 1 1
show 1 1 1 1 1 1 1 1 1 1 1 1 1
s search.php . . . . . . . . . . . . 1
move . . . . . . . . . . . . 1
static . . . . . . . . . . . . 1
6.0 1 1 1 1 1 1 1 1 1 1 1 1 1
9.20 Opera . . . . . . . . . . . . 1
delete . . . . . . . . . . . . 1
5.1 . . . . . . . . . . . . 1
cgi 1 1 1 1 1 1 1 1 1 1 1 1 1
rename . . . . . . . . . . . . 1
```

This shows us the reduced form of the initial data matrix in a features \( \times \) documents representation, i.e. this is a replicate-free version of it. We can see that the features partly consists of grouped tokens (for instance `admin.php par action` contains 3 tokens, which always co-occurred in the data) and how these tokens are present in the different documents. We can see the initial tokens before the grouping and their corresponding group assignment in the `group` variable:

```r
> asap$group
rv admin.php show s search.php par Gecko 1.8.1.3 move static 6.0
1 1 1 1 1 1 1 1 1 1 1 1 1
1 5 6 7 2 8 8
1.8.1.3 move static 6.0 action 9.20 Opera
Mozilla delete Firefox 2.0.0.3 5.0 5.1 20070309
1 9 1 1 1 1 10 1
```

The member variable `unprocessed` contains the initial data matrix before the feature selection and grouping step. If we want to reconstruct all replicates in the reduced feature space, we need the `getDuplicateData` function:

```r
> dim(getDuplicateData(asap))
[1] 12 10000
> dim(asap$unprocessed)
[1] 10034 10000
```

This will blow up the reduced matrix to the full 10,000 initial data points in the reduced feature space. To see, how often a specific entry in the reduced data matrix was present, we can have a look at the duplicate count:
The Replicate-Aware Non-Negative Matrix Factorization (NMF)

The replicate-aware NMF is a matrix factorization method which describes the data according to a new base vector system, i.e. each data point is described as a weighted sum of these base vectors. Thus, the base vectors can be seen as the parts of which a document is constructed. Furthermore, the new coordinates of a document (the base weights) can also be interpreted as a soft clustering. But before we can apply the NMF we need to specify the inner dimension of the factorization. This could either be supplied by a number (which should be even, if pca.init is TRUE), or a prismaDimension object generated by the fully automatized dimension estimation method:

```r
> asapDim = estimateDimension(asap)
> asapDim
```

Estimated data dimension for positive matrix factorization via simulated noise level: 8

Equipped with this object, we can now apply the NMF to the data:

```r
> asapNMF = prismaNMF(asap, asapDim, time=60)
```

Errors: 3771.392
Error: 3113.138
Error: 2855.863
Error: 2810.286
Error: 2765.763
Error: 2755.29
Error: 2752.505

```r
> asapLabels = getMatrixFactorizationLabels(asapNMF)
> table(asapLabels)
```

<table>
<thead>
<tr>
<th>asapLabels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>623 607 602 660 1696 2473 817 2522</td>
</tr>
</tbody>
</table>

We can look at the results via `plot(asapNMF)` which is shown in Figure 1. We can see that the NMF extracts a search template, then the four `admin.php`-action templates, a Firefox template and two static templates, which reproduces the results in [4], Section 3.1., with added user agents as “noise”.

Interface to the *tm* Package

To allow the application of the replicate-aware NMF to corpora generated by the *tm* package [1], the PRISMA package contains a converter function which maps a *tm* corpus object to a PRISMA data object. We exemplify this procedure with an already stemmed and cleansed version of the 15 subsections of [2]:

```r
> data(thesis)
> thesis
A corpus with 15 text documents
> thesis = corpusToPrisma(thesis, NULL, TRUE)
> thesis
PRISMA data tm-Corpus
Unprocessed data: # features: 2002 # entries: 15
Processed data: # features: 2002 # entries: 15
> thesisNMF = prismaNMF(thesis, 3, pca.init=FALSE)
```

Errors: 1329.73
Error: 1310.481
Error: 1295.959
Error: 1295.509
Since we have just 15 documents, the application of the feature reduction step and the correlation analysis suffers from too less data, which also holds true for the PCA-based initialization scheme. Thus, we ignore all these processings and apply the NMF directly on the data with three components as a sophisticated guess. To analyze the result we look at the top 20 words of the resulting base matrix:

```r
> isQuantile = (t(thesisNMF$B) > apply(thesisNMF$B, 2, quantile, prob=.99))
> maxFeatures = apply(isQuantile, 1, function(r) which(r == 1))
> rownames(thesis$data)[maxFeatures[, 1]]
[1] "add" "align" "associ" "cluster" "communic" "correct"
[7] "extract" "fill" "format" "inner" "machin" "messag"
[13] "obvious" "preserv" "reflect" "return" "simul" "templat"
[19] "trace" "transit" "tri"
> rownames(thesis$data)[maxFeatures[, 2]]
[1] "behavior" "chang" "configur" "crossvalid" "drop"
[6] "fast" "figur" "follow" "lead" "learn"
[11] "lower" "observ" "optim" "overal" "procedur"
[16] "process" "relat" "shown" "speed" "statist"
[21] "use"
> rownames(thesis$data)[maxFeatures[, 3]]
[1] "addit" "applic" "approach" "attack" "base" "construct"
[7] "content" "exploit" "method" "model" "network" "normal"
[13] "protocol" "server" "similar" "simpl" "structur" "techniqu"
[19] "token" "traffic" "use"
```

These word stems accurately describe the contents of the three chapters of [2] which concludes the analysis of this section.
References


