BiocSklearn – exposing python Scikit machine learning elements for Bioconductor

Vincent J. Carey, stvjc at channing.harvard.edu, Shweta Gopaulakrishnan, reshg at channing.harvard.edu, Samuela Pollack, spollack at jimmy.harvard.edu

July 31, 2017

Contents

1 Introduction

Scientific computing in python is well-established. This package takes advantage of new work at Rstudio that fosters python-R interoperability. Identifying good practices of interface design will require extensive discussion and experimentation, and this package takes an initial step in this direction.

A key motivation is experimenting with an incremental PCA implementation with very large out-of-memory data.

2 Basic concepts

2.1 Module references

The package includes a list of references to python modules.

```
library(BiocSklearn)
## Loading required package: reticulate
## Loading required package: knitr
SklearnEls()
## $np
## Module(numpy)
##
## $pd
## Module(pandas)
```

```
##
## $h5py
## Module(h5py)
##
## $skd
## Module(sklearn.decomposition)
```

2.2 Python documentation

We can acquire python documentation of included modules with reticulate's `py_help`:

```
# py_help(SklearnEls()$skd)
Help on package sklearn.decomposition in sklearn:

NAME
    sklearn.decomposition

FILE
    /Users/stvjc/anaconda2/lib/python2.7/site-packages/sklearn/decomposition/__init__.py

DESCRIPTION
    The :mod:`sklearn.decomposition` module includes matrix decomposition
    algorithms, including among others PCA, NMF or ICA. Most of the algorithms of
    this module can be regarded as dimensionality reduction techniques.

PACKAGE CONTENTS
    _online_lda
    base
    cdnmf_fast
    dict_learning
    factor_analysis
    fastica_
    incremental_pca
...
```

2.3 Importing data for direct handling by python functions

The reticulate package is designed to limit the amount of effort required to convert data from R to python for natural use in each language.

```
irloc = system.file("csv/iris.csv", package="BiocSklearn")
irismat = SklearnEls()$np$genfromtxt(irloc, delimiter=',')
```

To examine a submatrix, we use the take method from numpy. The bracket format notifies us that we are not looking at data native to R.

```
SklearnEls()$np$take(irismat, 0:2, 0L )
## [[ 5.1  3.5  1.4  0.2]
##  [ 4.9  3.   1.4  0.2]
##  [ 4.7  3.2  1.3  0.2]]
```

3 Dimension reduction with sklearn: illustration with iris dataset

We'll use R's prcomp as a first test to demonstrate performance of the sklearn modules with the iris data.

```
fullpc = prcomp(data.matrix(iris[,1:4]))$x
```

3.1 PCA

We have a python representation of the iris data. We compute the PCA as follows:

```
ppca = skPCA(irismat)
ppca
## SkDecomp instance, method:  PCA
## retrieve transformed data with getTransformed(),
##  python reference with pyobj()
```

This returns an object that can be reused through python methods. The numerical transformation is accessed via `getTransformed`.

```
tx = getTransformed(ppca)
dim(tx)
## [1] 150   4
head(tx)
##           [,1]       [,2]        [,3]         [,4]
## [1,] -2.684126 -0.3193972  0.02791483  0.002262437
## [2,] -2.714142  0.1770012  0.21046427  0.099026550
## [3,] -2.888991  0.1449494 -0.01790026  0.019968390
## [4,] -2.745343  0.3182990 -0.03155937 -0.075575817
## [5,] -2.728717 -0.3267545 -0.09007924 -0.061258593
## [6,] -2.280860 -0.7413304 -0.16867766 -0.024200858
```

The native methods can be applied to the `pyobj` output.

```
pyobj(ppca)$fit_transform(irismat)[1:3,]
##           [,1]       [,2]        [,3]        [,4]
## [1,] -2.684126 -0.3193972  0.02791483 0.002262437
## [2,] -2.714142  0.1770012  0.21046427 0.099026550
## [3,] -2.888991  0.1449494 -0.01790026 0.019968390
```

Concordance with the R computation can be checked:

```
round(cor(tx, fullpc),3)
##      PC1 PC2 PC3 PC4
## [1,]   1   0   0   0
## [2,]   0   1   0   0
## [3,]   0   0   1   0
## [4,]   0   0   0   1
```

3.2 Incremental PCA

A computation supporting *a priori* bounding of memory consumption is available. In this procedure one can also select the number of principal components to compute.

```
ippca = skIncrPCA(irismat) #
ippcab = skIncrPCA(irismat, batch_size=25L)
```

```
round(cor(getTransformed(ippcab), fullpc),3)
##         PC1    PC2    PC3    PC4
## [1,]  1.000  0.000  0.000  0.000
## [2,] -0.008 -1.000  0.002  0.000
## [3,] -0.002 -0.005 -1.000 -0.001
## [4,]  0.001 -0.002 -0.002  1.000
```

3.3 Manual incremental PCA with explicit chunking

This procedure can be used when data are provided in chunks, perhaps from a stream. We iteratively update the object, for which there is no container at present. Again the number of components computed can be specified.

```
ta = SklearnEls()$np$take # provide slicer utility
ipc = skPartialPCA_step(ta(irismat,0:49,0L))
ipc = skPartialPCA_step(ta(irismat,50:99,0L), obj=ipc)
ipc = skPartialPCA_step(ta(irismat,100:149,0L), obj=ipc)
ipc$transform(ta(irismat,0:5,0L))
##           [,1]       [,2]        [,3]         [,4]
## [1,] -2.684165  0.3190092 -0.02858225  0.002103429
## [2,] -2.714065 -0.1773644 -0.21124965  0.098808454
## [3,] -2.888975 -0.1453761  0.01709173  0.019793665
## [4,] -2.745300 -0.3187041  0.03078118 -0.075743907
## [5,] -2.728785  0.3263410  0.08941582 -0.061392703
## [6,] -2.281012  0.7409675  0.16819933 -0.024277215
fullpc[1:5,]
##            PC1        PC2         PC3          PC4
## [1,] -2.684126 -0.3193972  0.02791483  0.002262437
## [2,] -2.714142  0.1770012  0.21046427  0.099026550
## [3,] -2.888991  0.1449494 -0.01790026  0.019968390
## [4,] -2.745343  0.3182990 -0.03155937 -0.075575817
## [5,] -2.728717 -0.3267545 -0.09007924 -0.061258593
```

4 Interoperation with HDF5 matrix

We have extracted methylation data for the Yoruban subcohort of CEPH from the yriMulti package. Data from chr6 and chr17 are available in an HDF5 matrix in this BiocSklearn package. A reference to the dataset through the h5py File interface is created by `H5matref`.

```
fn = system.file("ban_6_17/assays.h5", package="BiocSklearn")
ban = H5matref(fn)
ban
## <HDF5 dataset "assay001": shape (64, 44560), type "<f8">
```

We will explicitly define the numpy matrix.

```
np = import("numpy", convert=FALSE) # ensure
ban$shape
## [[1]]
## [1] 64
##
## [[2]]
## [1] 44560
```

We'll treat genes as records and individuals as features.

```
ban2 = np$matrix(ban)$T
```

We'll define three chunks of the data and update the partial PCA contributions in the object `st`.

```
st = skPartialPCA_step(ta(ban2, 0:999, 0L))
st = skPartialPCA_step(ta(ban2, 1000:10999, 0L), obj=st)
st = skPartialPCA_step(ta(ban2, 11000:44559, 0L), obj=st)
sss = st$transform(ban2)
```

Verify against the standard PCA, checking correlation between the projections to the first four PCs.

```
iii = skPCA(ban2)
dim(getTransformed(iii))
## [1] 44560    64
round(cor(sss[,1:4], getTransformed(iii)[,1:4]),3)
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0   -1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

5 Conclusions

We need more applications and profiling.