Parallelisation of R with h2o

Lab I: Introduction

Dr. Simon Caton

Introduction

The idea of this first session is to get used to some of the basic h2o functions, and start with a simple example: a parallel PCA.

Setting up h2o

To initialise h2o, you need at least 2 things:

1. the Java Development Kit, accessible here: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. If you can open a terminal, type "java" and not get an error, you are good.
2. you need to run the following code:

```
# The following two commands remove any previously installed H2O packages for R.
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
pkgs <- c("RCurl","jsonlite")
for (pkg in pkgs) {
  if (! (pkg %in% rownames(installed.packages()))) { install.packages(pkg) }
}

# Now we download, install and initialize the H2O package for R.
install.packages("h2o", type="source",
                 repos="http://h2o-release.s3.amazonaws.com/h2o/rel-wolpert/4/R")
```

Also available here: http://h2o-release.s3.amazonaws.com/h2o/rel-wolpert/4/index.html – in the INSTALL IN R tab.

This is about a 300MB install, so may take some time.

Datasets

Throughout this session, we'll be using the following datasets:

- the MNIST dataset: handwritten digit recognition – this dataset is large enough to cause problems, but not too large that it will break your machine. It also has a well-established history in applications for both statistical as well as machine learning.
- the Movie Reviews dataset: classifying the nature of a review from its review text – this dataset although small (5000 reviews) is still large enough to require a reasonable amount of compute time for simple optimisations. Also, as text data, it highlights some of the challenges in using sparse matrices with h2o and their solutions.

MNIST: an introduction

The dataset is accessible from here: https://www.kaggle.com/c/digit-recognizer/data

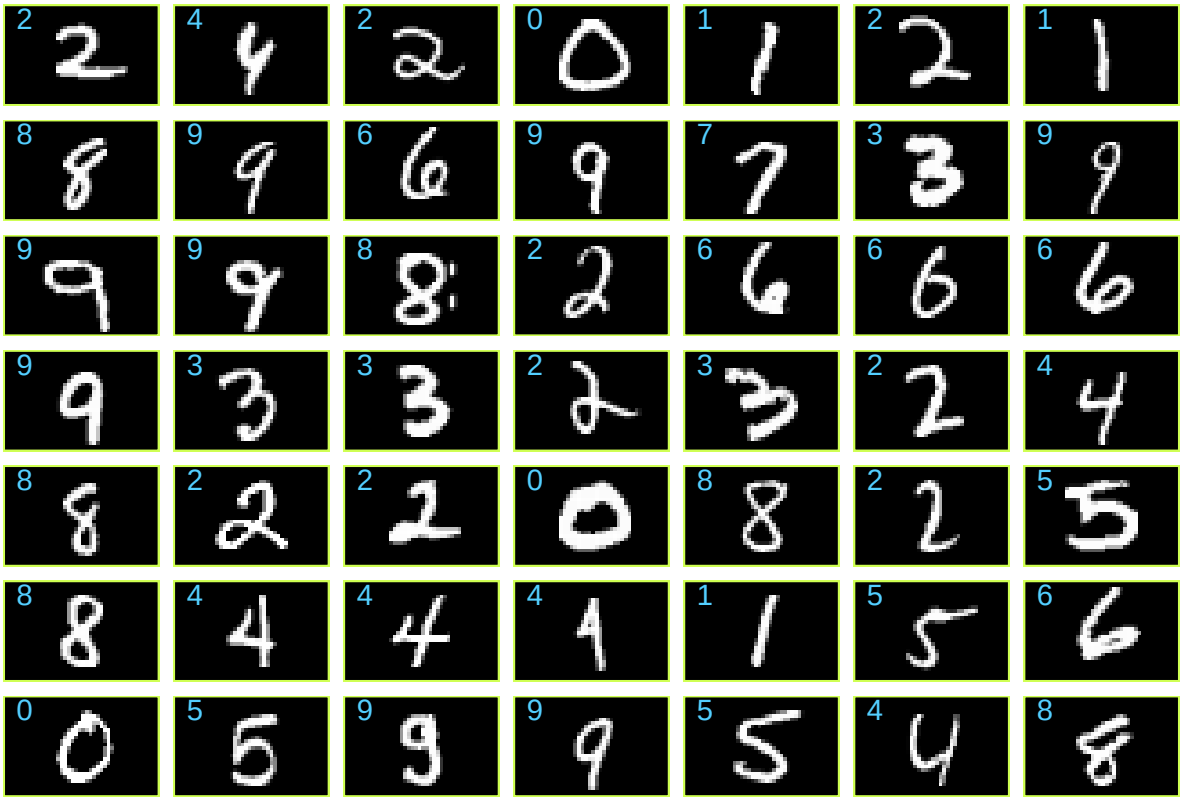
```
mnist <- read.csv("MNISTTrain.csv", header=T)
mnist$label <- factor(mnist$label)
table(mnist$label)
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188
```

So we have 785 features, corresponding to:

- 1 dependent (called label): a value between 0 and 9.
- 784 grayscale pixel values corresponding to a 28 x 28 image.

Where example images are as follows:



The code to produce the above figure is available here: https://www.kaggle.com/jameshirschorn/example-handwritten-digits.

So, it goes without saying that this is a dataset of significant size, both in terms of width (size of feature space), and size (no. of instances). Building a model on the full dataset will be computationally expensive. However, some preprocessing is available – we can remove pixels that are always black:

```
columsKeep <- names(which(colSums(mnist[, -1]) > 0))
mnist <- mnist[c("label", columsKeep)]
```

Movie Reviews: an introduction

This is a built-in dataset within the text2vec library:

```
library(text2vec)
data("movie_review")
summary(movie_review)
```

```
##       id              sentiment         review
##  Length:5000        Min.   :0.0000   Length:5000
##  Class :character   1st Qu.:0.0000   Class :character
##  Mode  :character   Median :1.0000   Mode  :character
##                     Mean   :0.5034
##                     3rd Qu.:1.0000
##                     Max.   :1.0000
```

```
prop.table(table(movie_review$sentiment))
```

```
##
##      0      1
## 0.4966 0.5034
```

```
movie_review$sentiment <- factor(movie_review$sentiment)
```

It is a labeled dataset consisting of 5000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of the reviews is binary, meaning an IMDB rating < 5 results in a sentiment score of 0, and a rating $>=7$ has a sentiment score of 1. No individual movie has more than 30 reviews.

An example review:

```
movie_review$review[1]
```

[1] "With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain insight into this guy who i thought was really cool in the eighties just to maybe make up my mind whether he is guilty or innocent. Moonwalker is part biography, part feature film which i remember going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's feeling towards the press and also the obvious message of drugs are bad m'kay.Visually impressive but of course this is all about Michael Jackson so unless you remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ an egotist for consenting to the making of this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really nice of him.The actual feature film bit when it finally starts is only on for 20 minutes or so excluding the Smooth Criminal sequence and Joe Pesci is convincing as a psychopathic all powerful drug lord. Why he wants MJ dead so bad is beyond me. Because MJ overheard his plans? Nah, Joe Pesci's character ranted that he wanted people to know it is he who is supplying drugs etc so i dunno, maybe he just hates MJ's music.Lots of cool things in this like MJ turning into a car and a robot and the whole Speed Demon sequence. Also, the director must have had the patience of a saint when it came to filming the kiddy Bad sequence as usually directors hate working with one kid let alone a whole bunch of them performing a complex dance scene.Bottom line, this movie is for people who like MJ on one level or another (which i think is most people). If not, then stay away. It does try and give off a wholesome message and ironically MJ's bestest buddy in this movie is a girl! Michael Jackson is truly one of the most talented people ever to grace this planet but is he guilty? Well, with all the attention i've gave this subject....hmmm well i don't know because people can be different behind closed doors, i know this for a fact. He is either an extremely nice but stupid guy or one of the most sickest liars. I hope he is not the latter."

A dataset like this serves an interesting purpose, h2o will not always run faster than a standard R implementation (e.g. a logistic regression + elasticnet regularisation via glmnet), but its optimisation routines will outperform a novice user quite easily, which we will explore in sesson III.

Getting started with h2o

Once installed, we start a local h2o cluster as follows:

```
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 hours 35 minutes
##     H2O cluster timezone:       Europe/Dublin
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.18.0.4
##     H2O cluster version age:    24 days
##     H2O cluster name:           H2O_started_from_R_scaton_kpi460
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.63 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.4.3 (2017-11-30)
```

This provides us with a series of information on our cluster. As this is a local cluster, i.e. it's running on your

machine, it automatically adopts the specification of you machine. Here, for me, I have x1 (i.e. H2O cluster total nodes: 1) quad core machine (i.e. H2O cluster total cores: 4). We can also access h2o Flow (a UI) by opening http://localhost:54321 in a browser – we'll come to this later.

h2o will run until showdown, for this, you run:

```
h2o.shutdown()
```

h2o handles almost all objects in R via S4 object representations. If you are unfamiliar with S4, see here: http://adv-r.had.co.nz/S4.html.

h2o is still being developed, but has already implemented a lot of methods. A basic R tutorial covering things that we will not cover directly here is available here: http://h2o-release.s3.amazonaws.com/h2o/master/1713/docs-website/Ruser/rtutorial.html. A quick overview of (parallelised) methods available to date are:

- Data import
- Standard R functions like: summary, str, quantile, =,>,<,|,[ ], etc.
- Gradient Boosted Models – very useful for building model Ensembles, and typically do well with a large number of problems
- Generalized Linear Models, including parallelised regularisation and parameter optimisation
- K-Means clustering
- Principal Components Analysis
- Principal Components Regression
- Deep Learning (one of h2o's primary use cases)
- Naive Bayes
- Random Forest

Note that depending on what you are trying to achieve, h2o will not always be "faster" than standard R, it will also be more demanding of battery life on a laptop.

Simple Example: Parallel PCA

To get started, let's look at some very simple examples of h2o; the MNIST dataset is an ideal candidate for PCA.

Standard R:

```
t1 <- Sys.time()
pca <- prcomp(mnist[, -1], scale. = F, center = F)
print( difftime( Sys.time(), t1, units = 'sec'))
```

```
## Time difference of 68.55391 secs
```

We don't need to scale the data; all variables are in the same range 0-255. Centering the data also will not really make a big difference in this case, the first PC will be "better" if we don't center the data.

h2o:

```
h2o.mnist <- h2o.importFile(path=paste0(getwd(), "/MNISTTrain.csv"), header=T)
h2o.mnist <- h2o.mnist[c("label", columsKeep)]
t2 <- Sys.time()
pca.h2o <- h2o.prcomp(h2o.mnist[, -1], k=100)
```

```
print( difftime( Sys.time(), t2, units = 'sec'))
```

```
## Time difference of 11.72307 secs
```

For h2o, we first have to import the dataset into an h2o frame. We could also do this via *as.h2o(mnist)*, however, *h2o.importFile()* is usually faster. The parameter k essentially just instructs h2o how many PCs to

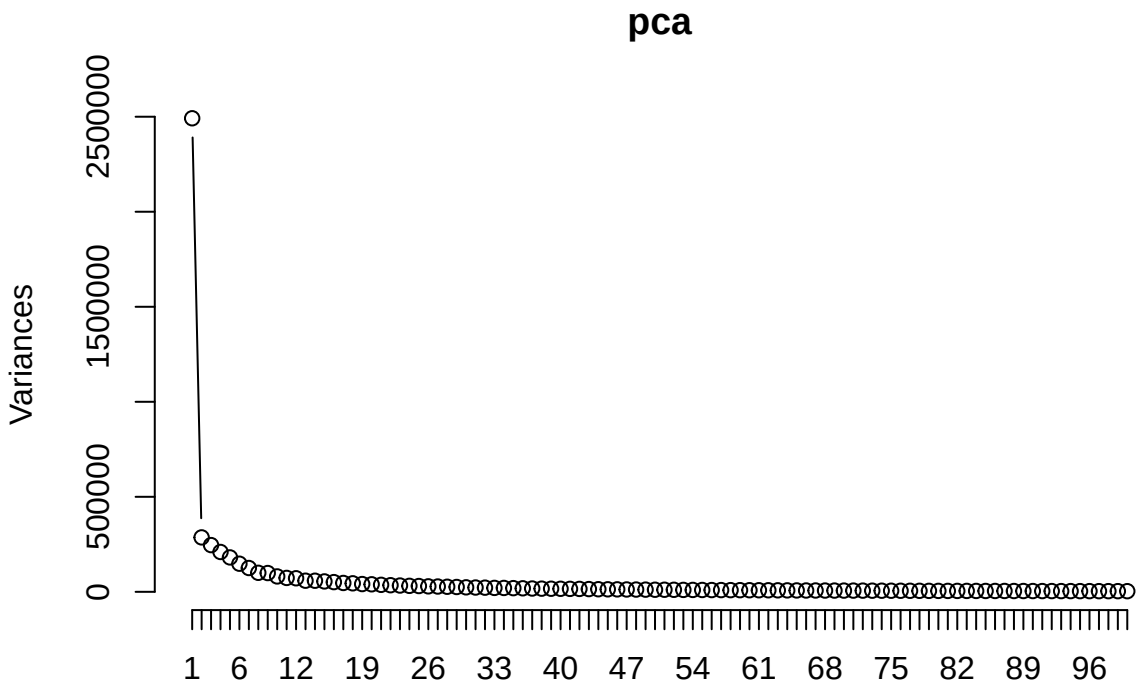
compute. Given that we have 700 odd variables, 100 is probably sufficient, and in fact probably already more than needed.

Given that PCA is a fairly standard method, that native R took about 6 times longer to run, is noticable for a dataset of this size. Of course, if you are using a much smaller dataset, then you may not really notice the difference in runtime.

Building the standard scree plot or cumulative importance plots, also are not significantly different to do, we just need to keep in mind that h2o uses an S4 representation, which requires a little more massaging if you are not used to it.

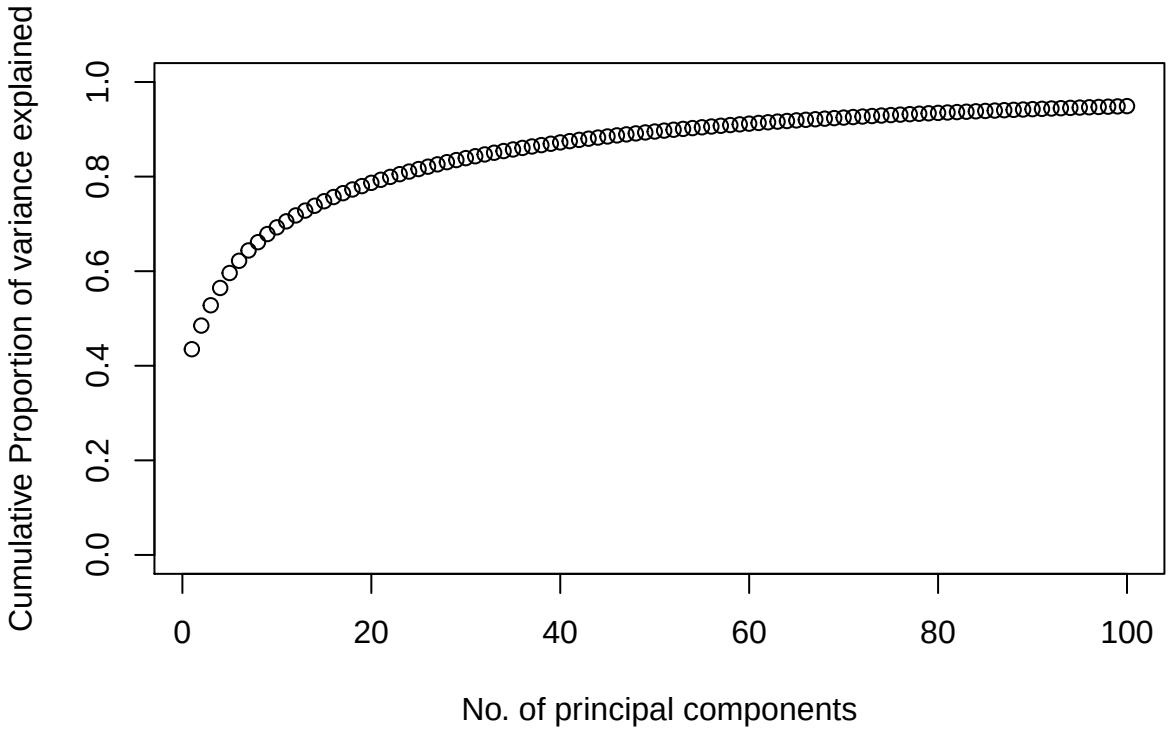
Standard R:

```
screeplot(pca, type="lines", npcs = 100)
```



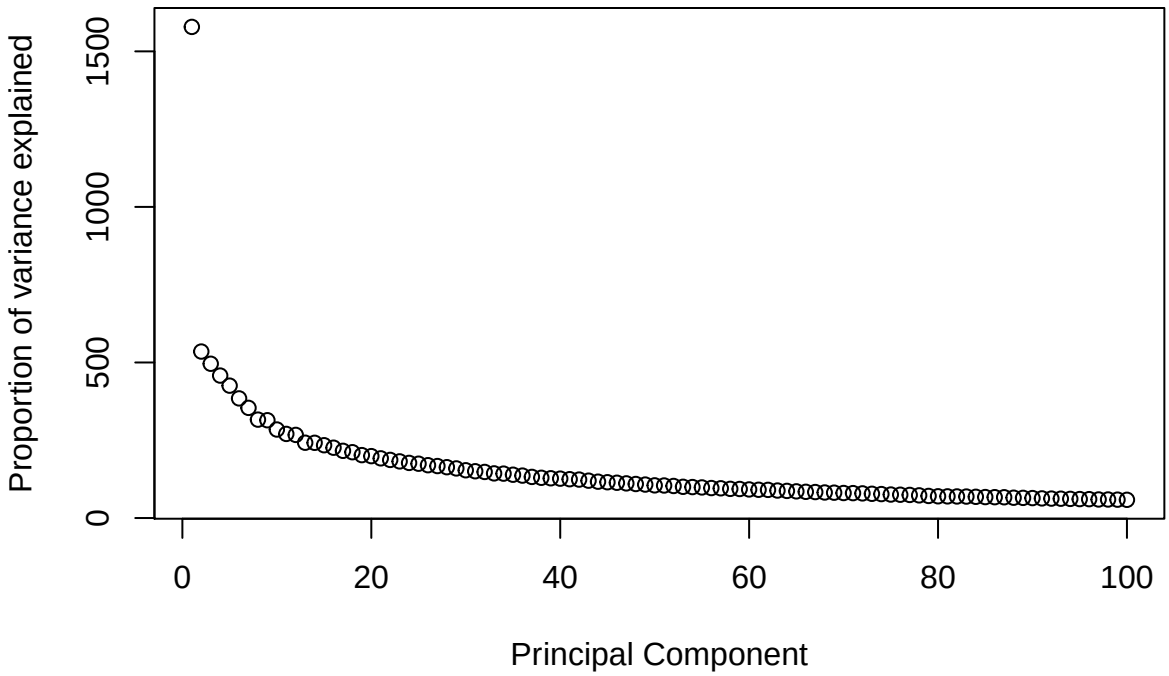

```
var.pca <- pca$sdev ^ 2
x.var.pca <- var.pca / sum(var.pca)
cum.var.pca <- cumsum(x.var.pca)

plot(cum.var.pca[1:100],xlab="No. of principal components",
     ylab="Cumulative Proportion of variance explained", ylim=c(0,1), type='b')
```

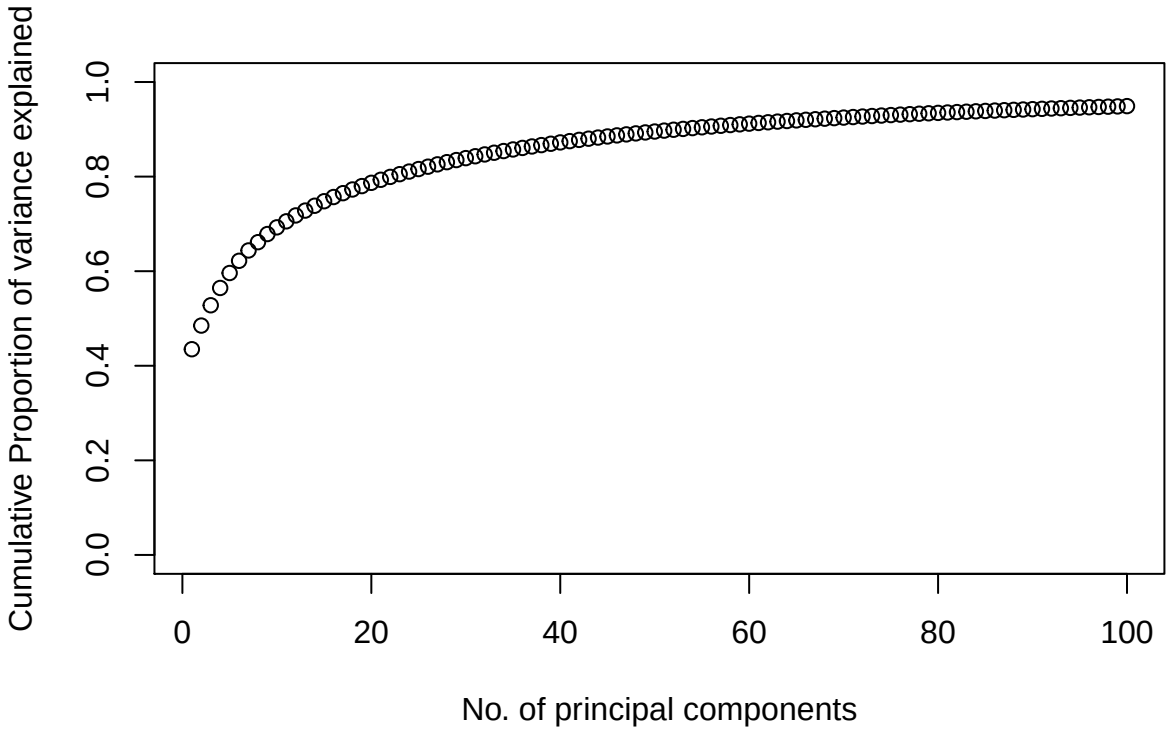


h2o:

```
plot(x=t(pca.h2o@model$importance[1,]), xlab="Principal Component",
     ylab="Proportion of variance explained")
```




```
plot(x=t(pca.h2o@model$importance[3,]), xlab="No. of principal components",
     ylab="Cumulative Proportion of variance explained", ylim=c(0,1), type='b')
```



pca.h2o@model$importance returns us a data.frame, which for plotting we need to subset and transpose. Note that the out of the box *screeplot* function will not work here, as it expects an object containing a sdev component, such as that returned by *princomp()* and *prcomp()*, engineering this is possible, but unnecessary.

Similarly, if we wanted to get the loadings for each PC, e.g. apply the first 35 PCs to the dataset:

Standard R:

```
mnist.pca <- as.matrix(mnist[,-1]) %*% pca$rotation[, 1:35]
```

Using h2o's PCA:

```
mnist.h2o.pca <- as.matrix(mnist[,-1]) %*%
  as.matrix(pca.h2o@model$eigenvectors[, 1:35])
```

We could cast the h2o frame *h2o.mnist* to a matrix, but this will give us the same result, and in this case, will take slightly longer to compute. However, if we were to use *mnist.h2o.pca* with h2o, we would have to cast it back to an h2o frame using *as.h2o()*.

Using Cloud resources

Obviously, the example used here hasn't been very demanding on your machine. This can change quite quickly, however, as we'll see in Session II, we can handle this by building an h2o cluster using Cloud resources, i.e. one or more virtual machines, and:

1. leverage h2o via RServer and a browser.
2. add additional parameters to *h2o.init()* to "point" at our Cloud VM.

Also note, that if you are on the move (e.g. hotspotting your phone), or in a location with poor internet, option 1 (with or without h2o) is a really useful way of coping with a low-bandwith connection. For option 1 all data transfers happen on the cloud infrastructure, for option 2 most large transfers happen on the cloud infrastructure, unless you pull data to your machine.

Note that we can also build multi-node h2o clusters, i.e. where we use more than one (virtual) machine. For this, we just need to start up h2o slightly differently, and point the different nodes at each other. See here: http://h2o-release.s3.amazonaws.com/h2o/rel-lambert/5/docs-website/deployment/multinode.html

Building a Ubuntu-based single node instance of h2o

Assuming you had access to either a linux box with Ubuntu 16.04 (Ubuntu Xenial), or you provisioned cloud resources again with Ubuntu 16.04 (Ubuntu Xenial), you could set-up a vm for both cases above as follows:

sudo apt-key adv –keyserver keyserver.ubuntu.com –recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9

sudo add-apt-repository 'deb [arch=amd64,i386] https://cran.rstudio.com/bin/linux/ubuntu xenial/'

sudo apt-get update

sudo apt-get -y install gdebi-core libcurl4-openssl-dev openjdk-8-jdk openssl libssl-dev libxml2-dev libjpeg62 r-base

sudo R CMD javareconf

wget https://download2.rstudio.org/rstudio-server-1.1.442-amd64.deb

sudo gdebi -n rstudio-server-1.1.442-amd64.deb

Omit the last 2 lines, if you do not want to use RServer. In your firewall settings, you need to open the following ports: 8787 (for RServer), and 54321 (for h2o).

Using NCI cloud resources

For the spring school, I've made each of you an 8-core virtual machine set up as above. If you find that your laptop is struggling, or that the examples are consuming too much battery, use the login provided for scenario 1 above and the remaining labs. Do, however, note that the 8-core VM may be slower than the timings noted in labs here, so please adjust the size of your training samples (i.e. make them smaller). In optimisation sections, you can also reduce the number of permutations (e.g. nfolds) and/or potentially the number of parameters being optimised.