Dimension Reduction: Principle Components and the Autoencoder

Mladen Kolar and Rob McCulloch

1. Introduction

We are in the "unsupervised learning" world.

We have a vector $x \in R^p$ of numeric measurements.

If p is big, then we can have a very hard time understanding x.

In dimension reduction, we try to map

$$x \to \tilde{x} \in R^q, \quad q << p$$

with a minimal loss of information !!

What does *minimal loss of information mean*??

How could this possibly work??

Basically, if x_i are “close” to some lower dimensional object in R^p then we may be ok.

We will consider the basic method called principal components which assumes the “lower dimension object” is a linear subspace.

We will also look at the autoencoder approach which, quite remarkably, looks for a nonlinear subset (a manifold). The autoencoder is based on neural networks.

Dimension Reduction and Supervised Learning:

Suppose you succeed and $\tilde{x}$ has the same information as x.

Now suppose you have a supervised problem in which you are trying to predict y from x.

Then, using our intuition from the bias-variance tradeoff, we would be better off building a model

$$y|\tilde{x}$$

than a model,

$$y|x,$$

since it is much easier to work in lower dimensions.

Of course, “succeed” can be hard to define, and it is unlikely you can map down without losing some information.

Nevertheless is is quite common to try

$$y|\tilde{x}$$

or

$$y|\ (x, \tilde{x})$$

in the hopes of finding a simple model.

This often works in practice!!!

2. Principal Components

Our simple underlying model is the $X = (X_1, X_2, \ldots, X_p)'$ is a random vector with

$$\Sigma = E((X - \mu)(X - \mu)').$$

So, as usual Σ_{ij} is the covariance between X_i and X_j and Σ_{ii} is the variance of X_i.

Given data $\{x_i\}_{i=1}^n,\ x_i \in R^p$ we can use the usual (or any other) estimate of Σ:

$$\hat{\Sigma} = \frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})'$$

That is, $\hat{\Sigma}_{ij}$ is the sample covariance between X_i and X_j and $\hat{\Sigma}_{ii}$ is the sample variance of X_i.

We shall see the principal components involves a fundamental scaling issue.

Usually we will also want to rescale to put the components of x on a common footing.

If you standardize by doing the z-score thing $x \to (x - \bar{x})/s_x$ then $\hat{\Sigma}$ is just the sample correlation matrix.

Note:

Each principal component is just a linear combination of x.

- the first principal component is the linear combination with the largest variance subject to L2 norm of the coefficents $=1$.

$$\max_{a,||a||=1} a'\Sigma a$$

- the second principal component is the linear combination, uncorrelated with the first, with the largest variance subject to L2 norm of the coefficents $=1$.
- the j^{th} principal component is the linear combination, uncorrelated with the 1,2,..(j-1) components with the largest variance subject to L2 norm of the coefficents $=1$.

Scaling:

If x_1 is distance in hundred of miles and we measure distance distance to downtown from homes in the metro area, we might feel x_1 does not vary very much.

But, if we measure distance in inches, then, we might feel x_1 does vary.

If we are looking for variability, the scale *obviously* matters.

Again, often people standardize

$$x \to (x - \bar{x})/s_x.$$

Really you should pick a scale for each variable such that you feel the variability is comparable, but this may not be obvious.

All four plots on the same scale!!

(1,1) plot: original (x1,x2) data.
most of the variation is along one axis.

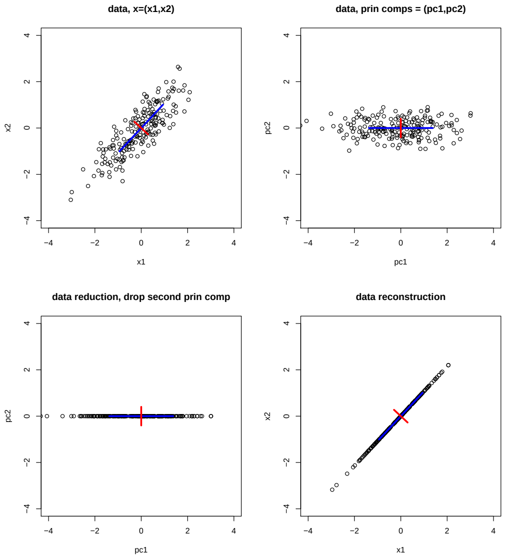
(1,2):
x axis is the first principal component.
y axis is the second principal component.

Each principal component is a linear combination of (x1,x2).
Looks like a rotation!

The first principal component varies much more than the second.

(2,1): data reduction: drop second since it does not do much.

(2,2): data reconstruction, rotate back.
Same as if you orthogonally projected the points onto the blue axis in the (1,1) plot.



With (x1,x2,x3), if the points cling to a plane we can rotate and drop one dimension.

The first two principal components will capture a lot of the variation and we can dimension reduce from R^3 to R^2.

With (x1,x2,x3), if the points cling to a line we can rotate and drop two dimensions.

The first principal component will capture a lot of the variation and we can dimension reduce from R^3 to R^1.

USArrests Data

```
Description:

     This data set contains statistics, in arrests per 100,000
     residents for assault, murder, and rape in each of the 50 US
     states in 1973.  Also given is the percent of the population
     living in urban areas.

Usage:

     USArrests

Format:

     A data frame with 50 observations on 4 variables.

       [,1]  Murder    numeric  Murder arrests (per 100,000)
       [,2]  Assault   numeric  Assault arrests (per 100,000)
       [,3]  UrbanPop  numeric  Percent urban population
       [,4]  Rape      numeric  Rape arrests (per 100,000)
```

```
> ad = USArrests
> states = row.names(ad)
>
> head(ad)
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
> summary(ad)
     Murder          Assault         UrbanPop          Rape
 Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
 1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
 Median : 7.250   Median :159.0   Median :66.00   Median :20.10
 Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
 3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
 Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

```
> pcres = prcomp(ad,scale=TRUE)
> P = pcres$rotation
> P
                PC1        PC2        PC3         PC4
Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

P gives the coefficients of the linear combinations.

How do you interpret the first principal component?
Note that you can multiply a column by -1 if you like.

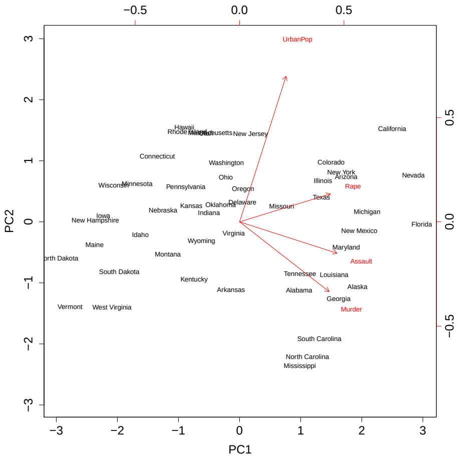
The bi-plot tries to plot the first two principal components and their weights in the same plot.

```
##biplot
pcres$rotation= - pcres$rotation
pcres$x = -pcres$x
biplot(pcres,scale=0,cex.lab=1.5,cex.axis=1.5,cex=.6)
```

Two different scales, one for the principal components, and one for the weights.

bottom scale: first principal component
left scale: second principal component
top scale: weights of first component across our 4 variables
right scale: weights of second component across our 4 variables

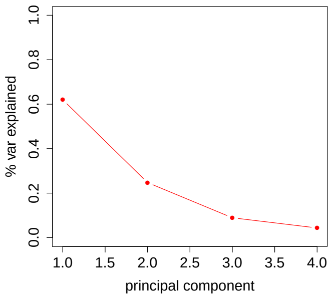
Notice the two principal components are on the same scale so that you can see how much more variable the first component is than the second.

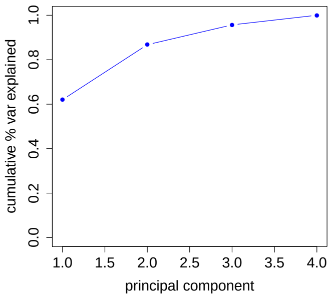


Plot the variance explained by the principal components.

```
##
pcv = pcres$sdev^2
pve = pcv/sum(pcv)

par(mfrow=c(1,2))

plot(pve,xlab="principal component",ylab="% var explained",
          ylim=c(0,1),type="b", cex.axis=1.5,cex.lab=1.5,col="red",pch=16)
plot(cumsum(pve),xlab="principal component",
              ylab="cumulative % var explained",ylim=c(0,1),type="b",
              cex.axis=1.5,cex.lab=1.5,col="blue",pch=16)
```

Left: % explained by each one.
Right: cumulative % explained.





3. Autoencoder

We will use deep neural nets for data reduction.

This is very cool.

The Movie Review Data

Each row corresponds to a movie.

For each movie, we have the text of a review.

The first column is the name of the movie.

```
> print(dim(data))
[1] 100 564
> print(data[20:25,c(1,40:47)])
                            X       area      argu        arm       armi
1                   Star Wars 0.02919095 0.0108981 0.02607413 0.00000000
2 E.T. the Extra-Terrestrial 0.05046324 0.0000000 0.00000000 0.00000000
3      2001: A Space Odyssey 0.00000000 0.0000000 0.03399680 0.00000000
4   The Silence of the Lambs 0.00000000 0.0000000 0.00000000 0.00000000
5                   Chinatown 0.02595432 0.0000000 0.03477463 0.00000000
6   Bridge on the River Kwai 0.01705374 0.0000000 0.00000000 0.05194466
      arrang     arrest      arriv        ask
1 0.00000000 0.01019338 0.02765395 0.02682460
2 0.00000000 0.00000000 0.00000000 0.04173521
3 0.00000000 0.00000000 0.01081700 0.08394073
4 0.00000000 0.08187929 0.04442659 0.05745895
5 0.01207359 0.01359475 0.02950529 0.05724083
6 0.01586633 0.00000000 0.02908046 0.05641664

[6 rows x 9 columns]
```

From the movie reviews a set of *terms* were extracted.

Columns 2 - 564 correspond to the different terms.

The numbers in columns 2-564 are the tf-idf value for a given term in a given movie.

tf-idf

tf_{vd}: term frequency of term v in document d:

% of words in document equal to the given term.

df_v: document frequency of of term v over the documents

% of documents that contain term v

$$\text{tf-idf}_{vd} = tf_{vd} \times log(1/df_v).$$

"term frequency - inverse document freqency".

Intuition: If a term appear a lot in a document that tells you something about the document, but not so much if it apears in many of the other documents.

There are many variants of the tf-idf measure.

So, to get our data someone:

- Processed all the movie reviews to come up with a set of terms.
- Computed the tf-idf$_{vd}$ for each term and document.

Step 1 is not obvious.

Cook Book:
A high value of tf-idf means that word in that document is important.

I'm not sure the version of tf-idf is exactly the one on the previous slide, I just chose a version that is relatively simple to understand so that we can get the idea.

Just for fun, let's try clustering the movies.

k-means in h2o:

```
set.seed(99)
m = h2o.kmeans(data,x=2:564,k=5,standardize=FALSE,init="PlusPlus")
p = h2o.predict(m,data)
pp = as.vector(p$predict)

tapply(as.vector(data[,1]),as.vector(p$predict),print)
```

```
init: Specify the initialization mode.
The options are Random, Furthest, PlusPlus, or User.

Random initialization randomly samples the k-specified value of the rows
of the training data as cluster centers.

PlusPlus initialization chooses one initial center at random and
weights the random selection of subsequent centers so that points
furthest from the first center are more likely to be chosen.

Furthest initialization chooses one initial center at random
and then chooses the next center to be the point furthest
away in terms of Euclidean distance.

User initialization requires the corresponding user_points parameter.
Note that the user-specified points dataset must have
the same number of columns as the training dataset.
```

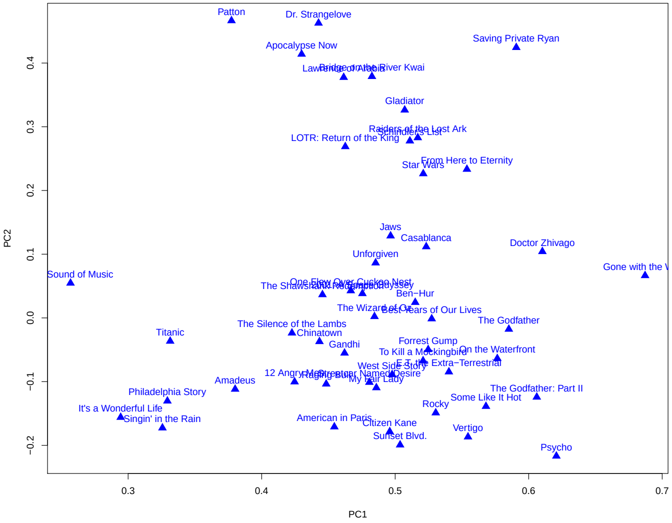
```
$`0`
[1] "Gone with the Wind"    "To Kill a Mockingbird" "Braveheart"
[4] "High Noon"             "Rain Man"              "Rebel Without Cause"

$`1`
 [1] "The Shawshank Redemption"   "One Flew Over Cuckoo Nest"
 [3] "The Wizard of Oz"           "Psycho"
 [5] "Vertigo"                    "E.T. the Extra-Terrestrial"
 [7] "The Silence of the Lambs"   "Some Like It Hot"
 [9] "The Exorcist"               "The French Connection"
[11] "Fargo"                      "Close Encounters 3rd Kind"
[13] "American Graffiti"          "A Clockwork Orange"
[15] "Rear Window"                "The Third Man"
[17] "North by Northwest"

$`2`
 [1] "The Godfather"            "Casablanca"
 [3] "Lawrence of Arabia"       "On the Waterfront"
 [5] "West Side Story"          "Star Wars"
 [7] "Chinatown"                "12 Angry Men"
 [9] "Dr. Strangelove"          "Apocalypse Now"
[11] "LOTR: Return of the King" "Gladiator"
[13] "From Here to Eternity"    "Unforgiven"
[15] "Raiders of the Lost Ark"  "My Fair Lady"
[17] "Ben-Hur"                  "Doctor Zhivago"
[19] "Jaws"                     "Treasure of Sierra Madre"
[21] "The Pianist"              "City Lights"
[23] "Giant"                    "The Grapes of Wrath"
[25] "Shane"                    "The Green Mile"
[27] "Pulp Fiction"             "Stagecoach"
[29] "The Maltese Falcon"       "Double Indemnity"
```

```
$‘3‘
 [1] "Schindler’s List"           "Forrest Gump"
 [3] "Bridge on the River Kwai"   "Saving Private Ryan"
 [5] "Patton"                     "Butch Cassidy & Sundance"
 [7] "Platoon"                    "Dances with Wolves"
 [9] "The Deer Hunter"            "All Quiet on Western Front"
[11] "Mutiny on the Bounty"       "Yankee Doodle Dandy"

$‘4‘
 [1] "Raging Bull"              "Citizen Kane"
 [3] "Titanic"                  "The Godfather: Part II"
 [5] "Sunset Blvd."             "The Sound of Music"
 [7] "2001: A Space Odyssey"    "Singin’ in the Rain"
 [9] "It’s a Wonderful Life"    "Amadeus"
[11] "Gandhi"                   "Rocky"
[13] "Streetcar Named Desire"   "Philadelphia Story"
[15] "American in Paris"        "Best Years of Our Lives"
[17] "Good, Bad and Ugly"       "The Apartment"
[19] "Goodfellas"               "The King’s Speech"
[21] "It Happened One Night"    "A Place in the Sun"
[23] "Midnight Cowboy"          "Mr. Smith Goes Washington"
[25] "Annie Hall"               "Out of Africa"
[27] "Good Will Hunting"        "Terms of Endearment"
[29] "Tootsie"                  "Network"
[31] "Nashville"                "The Graduate"
[33] "The African Queen"        "Taxi Driver"
[35] "Wuthering Heights"
```

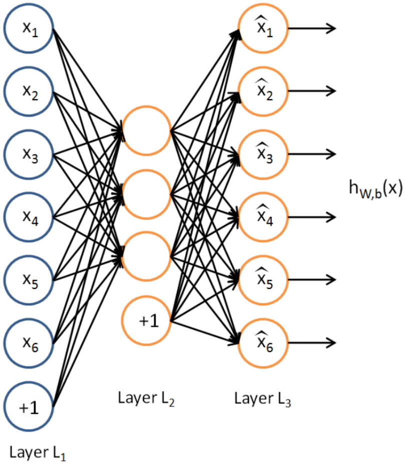
Now let's try principal components:

```
m = h2o.prcomp(data,2:564,k=2)
p = h2o.predict(m,data)
pR = as.matrix(p)

nmov = 50
labels = as.vector(data[1:nmov,1])
plot(pR[1:nmov,],pch=17,col="blue",cex=1.5)
text(pR[1:nmov,],labels,col="blue",pos=3) #pos=3 means above
```

Plot of first two principal components labeled with the movie name.



The Autoencoder



Make you outputs your inputs, but have an internal hidden layer with a small number of units.

Loss function

- For real valued inputs, try to find weights such that

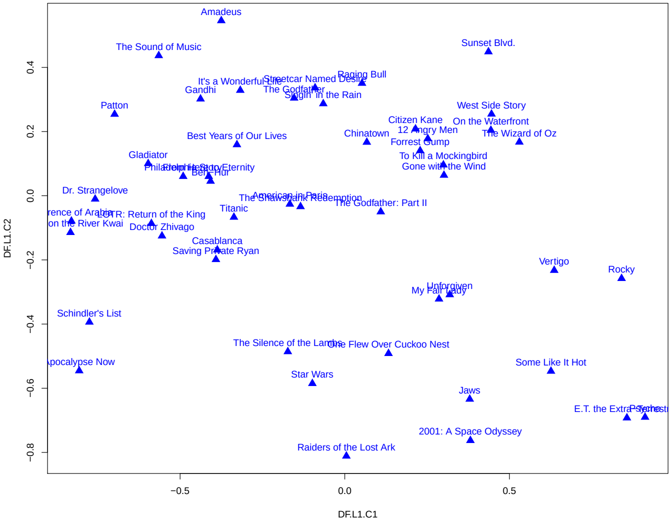
$$\frac{1}{2}\sum_k (x_k - \hat{x}_k)^2$$

  is minimized
- For binary input cross entropy is used, which is similar to deviance

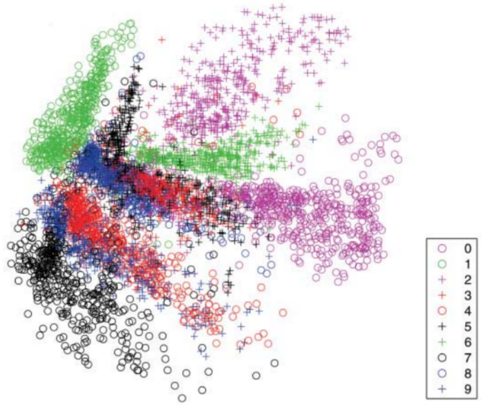
Autoencoder in h2o for the movies data:

```
m = h2o.deeplearning(2:564,training_frame=data,hidden=c(2),
             autoencoder=T,activation="Tanh")
f= h2o.deepfeatures(m,data,layer=1)
fR = as.matrix(f)

#perfm = h2o.performance(m)
#cat("rmse: ",perfm@metrics$RMSE,"\n")

nmov = 50
labels = as.vector(data[1:nmov,1])
plot(fR[1:nmov,],pch=17,col="blue",cex=1.5)
text(fR[1:nmov,],labels,col="blue",pos=3) #pos=3 means above
```

Plot of the 2 deep features summarizing the tf-idf values for movies.



Autoencoder for the MNIST digits problem.



Autoencoder structure: 784 — 1000 — 500 — 250 — 2

Easy to see a green circle (1).
Blue circles (8) and red plus (3) are confused in the middle.