Humboldt University to Berlin

Analysis of a FIFA dataset

Submitted To:
Alla Petukhina
Course : Statistical
Programming Language
Ladislaus von Bortkiewicz
Chair of Statistics

Submitted By :
Arna Wömmel
Daniel Vecera
Hsieh Pui Man
Julien Kraemer
Kseniia Ovadenko
Group 4
Winter term 2017/2018

Contents

List of Figures

List of Tables

1 Introduction

2 Theory and Design

2.1 Principal Component Analysis (PCA)

2.1.1 Notation

Consider a ($n \times p$) zero -mean matrix X(if it's not the case we can center X by multiplying X from the left by $H := I_n - \frac{1}{n}1_n 1_n^T$) with n the number of individuals and p the number of variables.

2.1.2 General idea/Aim

A

The PCA helps to summarize a quantitative dataset with many variables:

- see the correlations between the variables
- represent the p-dimensional point cloud of indivuals (here the players) by projecting them on spaces of smaller dimension
- construct new variables called principal components that are uncorrelated and that synthesize information

In the end, we aim at having **the maximal proportion of the overall variance captured by only a few principal components**(see below for the definition)

ADD REFERENCE for section 2.1.2

2.1.3 Computation

Define $S := \frac{1}{n}X^T HX$ the empirical covariance matrix and $R := S^{\frac{-1}{2}}SS^{\frac{-1}{2}}$ the empirical correlation matrix and $S^{\frac{-1}{2}} = diag(S)^{\frac{-1}{2}}$.

We also denote $D = \frac{1}{n}I_n$ the matrix of the individuals weights and $Q = I_p$ (non-normed PCA) or $Q = diag(S)^{-1}$ (normed PCA) (The PC technique is sensitive to scale changes, hence the PC transformation should be applied to data that have approximately the same scale in each variable). Reference for last 2 pargraphs

Remark 2.1.3.1 *That's why in pratice, we prefer to use the normed PCA*

In a normed PCA, each variable has the same weight (in the non normed PCA, the variable with the largest variance has the largest weight)

Reference

We diagonalize then SQ.

Eigenvectors of $S_I := SQ$, a_k,$k = 1, \ldots, r$ (where $rk(X) = r$) are called **Principal Components (PC)** and satisfy $S_I a_k = \lambda_k a_k$ with $\lambda_1, \ldots, \lambda_r$ eigenvalues of S_I.

Main properties of the PCs

The PCs have zero means, variance $\mathrm{Var}(a_k) = \lambda_k$, and are uncorrelated. From $\lambda_1 \geq \cdots \geq \lambda_r$,it follows that $\mathrm{Var}(a_1) \geq \cdots \geq \mathrm{Var}(a_r)$ where Var denotes here the empirical variance ant not the usual variance.
Furthemore it holds that $\sum_{i=1}^{r} \mathrm{Var}(a_i) = \mathrm{tr}(S_I)$

Principal factors u_k $(k = 1, \ldots, r)$ are defined as $u_k = Qa_k$.

Vectors F^k $(\in \mathbf{R}^n)$ representing the individuals coordinates on the k-th axis satisfy $F^k = Xu_k$.

Vectors G^k $(\in \mathbf{R}^p)$ representing the variables coordinates on the k-th axis are given by $G^k = \sqrt{\lambda_k} a_k$.

The graphical representation of the PCs is obtained by plotting the first PC vs. the second (and eventually vs. the third).

Reference section Main Preperties PC

Remark 2.1.3.2 *The components of the the eigenvectors are the weights of the original variables in the PCs.*

2.1.4 Interpretation of the PCs

Variance explained by the first q PCs is

$$\frac{\sum_{i=1}^{q} \lambda_i}{\sum_{i=1}^{r} \lambda_i}$$

2.1.5 Quality of the PCA outcomes

The well represented variables are those which are near the edge of the correlation circle and the variables near (0,0) aren't well represented.

The well represented individuals are those which are close from the main components (angle between the indivual and the principal component close to 0).

The contribution of the i-th indivual to the k-th axis is given by : reference

$$ctr_k(i) = \frac{(F^k(i))^2}{n\lambda_k}$$

Rule of thumb : We consider only the contributions greather than the average contribution ie $\frac{1}{n}$.

The contribution of the j-th variable to the k-th axis is given by : reference

$$ctr_k(j) = \frac{(G^k(j))^2}{n\lambda_k} \text{if } Q = I_p.$$

Rule of thumb : We consider only the contributions greather than the average contribution ie $\frac{1}{p}$. reference

Remark 2.1.5.1 *For further details ,see ref (2014) (in French).*

2.2 Clustering

2.2.1 K-Means clustering

K-means clustering is a way of partitioning a set of data points into "clusters," or sets of data points which are similar to one another. It works by iteratively reassigning data points to clusters and computing cluster centers based on the average of the point locations. reference

Formally :
It's based on the LLoyd's algorithm : reference

Consider $x_1, \ldots x_{(m)} \in \mathbf{R}^n$ and fix K the number of clusters.

Step 1:
Initialize cluster centroids $\mu_1^{(0)}, \ldots, \mu_K^{(0)} \in \mathbf{R}^n$ randomly. We obtain a first corresponding partition $C_i^{(0)}$ $(i = 1 \ldots K)$.

Step 2:
Repeat until convergence what follows :

Assign each point to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean.
Mathematically :
Assignement step :

A new partition $C_i^{(t)}$ is defined at each iteration (where t denotes the t-th iteration) :

$$C_i^{(t)} = \{x_p : \left\| x_p - \mu_i^{(t)} \right\|^2 \leq \left\| x_p - \mu_j^{(t)} \right\|^2 \forall j, 1 \leq j \leq K\} \text{ where } x_p \text{ is assigned to exactly one } C_i^{(t)}\text{, even if it could be assigned to two or more of them.}$$

The i-th cluster $C_i^{(t)}$ contains all points closer of $\mu_i^{(t)}$ than all other centroids.
Update step: Calculate the new means to be the centroids of the observations in the new clusters.

$$\mu_i^{(t+1)} = \frac{1}{|C_i^{(t)}|} \sum_{x_j \in C_i^{(t)}} x_j \text{ where } |.| \text{ denotes the size of a set}$$

The algorithm has converged when the assignments no longer change

Remark 2.2.1.1 *One can use an other distance such as the Manhattan distance. It doesn't depend on the distance in finite dimensions (ie if we work in some $\in \mathbf{R}^n$) since the norms are equivalent)* ref

Remark 2.2.1.2 *We want to minimize the Within-cluster variation (or equivalently to maximize the Between-cluster variation)* ref

Remark 2.2.1.3 *To find the number of clusters in the data, the user needs to run the K-means clustering algorithm for a range of K values and compare the results.* ***In general, there is no method for determining exact value of K.*** ref

One of the metrics that is commonly used to compare results across different values of K is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing K will always decrease this metric, to the extreme of reaching zero when K is the same as the number of data points which is not satisfying.

Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K. ref

2.2.2 Agglomerative clustering

This method builds the hierarchy from the individual elements by progressively merging clusters.
The algorithm can be divided into the several steps : ref

We start with all observations : n groups.

Step 1:
The first step is to determine which elements to merge in a cluster. Usually, we want to take the two closest elements $\{i_1, i_2\}$, according to the chosen distance (e.g Manhattan distance , Euclidean distance , Maximum distance , Mahalanobis distance) $d(i_1, i_2) \Rightarrow G^{(1)}(i_1, i_2)$
There remain n-2 isolated observations.

Step 2:
Look for the second smallest distance between the observations. There are two possibilities :

- two different observations from the first ones $\Rightarrow$ a new cluster $G^{(2)}(i_3, i_4)$, there remain n-4 isolated observations.
- one of the two observations is i_1 or i_2 , in this case, the observation i_3 is aggregated to the first group to form a new group $G^{(2)}(i_1, i_2, i_3)$.
  There still remain n-3 isolated observations and define

$$d_{linkage}(G^{(1)}(i_1, i_2), i_3) = linkage(d(i_1, i_3)), d(i_2, i_3))$$

Step 3: Repeat the above instructions for the isolated observations until all observations belong to the same group.

We progressively build a so-called dendrogram (see [4] for an example).

The linkage criterion determines the distance between sets of observations as a function of the pairwise distances between observations.
Some commonly used linkage criteria(or measure of dissimilarities) between two sets of observations A and B are : ref

- Maximum or complete-linkage clustering $d(\{a,b\},c) = max(d(a,c),d(b,c))$
- Minimum or single-linkage clustering $d(\{a,b\},c) = min(d(a,c),d(b,c))$
- Mean or average linkage clustering $d(\{a,b\},c) = mean(d(a,c),d(b,c))$

2.3 Entropy

The concept of entropy is used in many areas such as probability theory, statistical physics and computer science. Entropy is a formalization of an intuitive concept of information. **Information entropy** is defined as the average amount of information produced by a stochastic source of data. This concept was originally presented by Claude Shannon in "A Mathematical Theory of Communication"(1948).
Suppose some random variable p is given. Information function I(p) represents the average amount of 'information' that we get after getting the observation result of this variable. I(p) should satisfy the following properties that hold for casual 'information':

- $I(p) > 0$ (a new observation can't reduce information)
- $I(1) = 0$ where 1 is a constant variable (if some variable is always the same such observation adds no information)
- $I(p_1,p_2) = I(p_1) + I(p_2)$ where p_1 and p_2 are independent random variables (if two observations are independent then resulting information is a sum of information from individual observations)

It can be shown that only one function following those requirements exists (up to multiplication by a constant):

- in case of discrete random variable

$$S = -\sum_i p_i log(p_i)$$

- in case of continuous variable

$$-\int p(x) log(p(x)\, dx$$

As we can see, uniformly distributed random variable maximizes this function. If a random variable is a constant this function turns to zero. As mentioned before, the information function is unique up to a constant. If a particular function $I(p)$ holds aforementioned properties then $I_1(p) = c * I(p)$ holds those properties as well. This constant can be chosen for convenience. In computer science 1 bit is an information of the random variable with two equally probable outcomes.

3 Implementation

3.1 Quantlet 1

Motivation : We have a large number of quantitative variables in our dataset and we want to reduce the dimension of our study to make easier possible predictions.

We follow the same procedure as explained in the theoretical part about the PCA (see [2.1]).

We decided to create our own PCA function rather than simply use a package function such as *dudi.pca* from the package *ade4*. The function *PCA* takes as arguments a **table** (either a matrix or a data.frame), a binary variable **norm** indicating if we want a normed or a non normed PCA ,an integer variable **order** (by default 2, it's the number of PCs, we put 2 as default value because it's easier to interpret graphs in a two dimensional space than in higher dimensions) and an integer variable **desiredvariance** : it's the proportion of the total variance explained by the returned PCs.

Remark 3.1.0.1 *Thereafter we compare our results which these obtained by package functions of R.*

Remark 3.1.0.2 *We apply a normed PCA to our data.frame since our variables don't have the same scale.*

First, we need to check that we only deal with quantitative variables (Otherwise, it's not possible to use the usual algebra). If it's not the case, the function stops and we return an error message "The argument table must be a matrix/-dataframe of exclusive quantitative values".

We use the notations from [2.1.3] for the number of individuals, the number of variables, the weight matrix , the sample covariance matrix, the correlation matrix and the proportion of the total variance explained by the principal components.

The number of principal components is the minimum between order and the minimal number of axes needed to explain 80% (**desiredvariance**) of the overall variance.

Then we compute the principal components (eigenvectors of the matrix S_I denoted by **FF**) and the coordinates of the variables on these axes (matrix **G**).

Remark 3.1.0.3 *We don't use the letter F to refer to the principal components because this letter is dedicated to "FALSE".*

Finally, **we return the principal components, the new variables coordinates ,the correlation matrix and the explained variance.**

```r
PCA = function(table, norm = T, order = 2, desiredvariance = 0.8) {
  check = apply(table, 2, is.numeric)  #We check if we have only quantitative values
  if (mean(check) == 1) {
      table = scale(table, center = TRUE, scale = norm)  #We center the dataset (but we don't scale)
      n = dim(table)[1]  #number of indivuals (rows)
      p = dim(table)[2]  #number of variables (columns)
      D = 1/n * diag(rep(1, n))  #matrix with the weights of indivuals
      Q = diag(rep(1, p))
      S = t(table) %*% D %*% table  #sample covariance matrix
      u = diag(S)
      u = 1/sqrt(u)
      D1s = diag(u)
      R = D1s %*% S %*% D1s  #sample correlation matrix
      eigenvalues = eigen(S)$values
      eigenvectors = eigen(S)$vectors
      inertia = sum(diag(S))  #inertia is the sum of the eigenvalues
      cum = cumsum(eigen(S)$values)/inertia  # cumulative energy /inertia
      i = 1
      while (cum[i] < desiredvariance) {
          i = i + 1
      }
      i = min(order, i)
      # We project the table on the i first vectors We compute the main factors and the coordinates of the individuals on the i first axes Main
      # factors are main axes since Q=Ip Gk for 1<=k<=i fulfill : Fk=table*uk where (uk)_k are the main axes
      FF = matrix(rep(0, n * i), ncol = i)  #zero matrix for storing the Fk's
      G = matrix(rep(0, p * i), ncol = i)  #zero matrix for storing the Gk's
      nor = matrix(rep(0, p * i), ncol = i)
```

```r
        cor = matrix(rep(0, i * p), ncol = p)
        for (k in 0:i) {
            FF[, k] = table %*% eigenvectors[, k]
            G[, k] = sqrt(eigenvalues[k]) * eigenvectors[, k]
        }
        for (m in 1:i) {
            for (j in 1:p) {
                cor[m, j] = G[j, m]/sqrt(1/n * t(table[, j]) %*% table[, j])
            }
        }
        return(list(FF = FF, G = G, cor = cor, cum = cum, R = R))  #We return FF,G,cor,the inertia proportion and the correlation matrix
        # Coordinates of points in the area of variables
    } else {
        stop("The argument table must be a matrix/dataframe of exclusive quantitative values")
    }
}
```

See also PCA1

3.2 Quantlet 2

3.3 Quantlet 3

3.4 Quantlet 4

3.5 Quantlet 5

3.6 Quantlet 6

3.7 Quantlet 7

3.8 Quantlet 8

3.9 Quantlet 9 and 10

In this quantlets we realised the analysis of predictability of matches and leagues (Quantlet 9), as well as plotted graphis depicting the its results (Quantlet 10).

Motivation and idea: The original idea was taken from one of the Kaggle's kernels . This analysis was originally performed in Python, so we replicated it in R.
The idea was to calculate the predictability of chosen leagues. This predictability is regarded as a measure of competitiveness, and, hence, uncertainty of outcomes of matches. Predictability of matches and leagues are calculated on the basis of probabilities, which has different outcomes of matches. So, we take match as an event. In this case, the event has three possible outcomes - a win of home team, a lose of a home team (which os obviously the same as a win of a guest team), or a draw. The probability of each outcome is calculated via the odds. These gambling odds are provided by the agencies´ analytics who are aiming at predicting the outcome of the game with the most possible accuracy. So we can assume that this analytics' predictions take into account the historical data of a certain team´s performance, players attributes and some other figures. Hence, these bidding odds reflect the chances of each team to win. We are tranforing these odds into probabilitities of a certain outcome. On the basis of calculated probabilities we calculate the entropy, which is used as a measure of (un)predictability of the event.
The idea of uncertainty is intuitively understandable: if we have a certain set of event, and the corresponding to them probabilities are equal, we cannot say, which outcome is more or less probable then others, so the result of realisation is unex-

pected for us. When it comes to a football, we are talking about match results and how . We would rather say that the match is more interesting to watch and the league is more competitive, when both teams are known to have approximately equal level, then when there is a favourite and a underdog.

Implementation in R For calculating predictability of the matches across leagues via entropy, we will be using the following tables "Country", "Leagues", and "Matches". In the latter the information about the outcomes of the matches is contained, as well as bidding odds from gambling agencies. For this analysis we have chosen one agency, which is marked as "365" in the Kaggle dataset, and is supposedly a UK-based online gambling agency "bet3565". This gambling agency, like all other, gives bids for three possible outcome: home team is winning, away team is winning, and a draw - columns "365H", "365A", and "365D" of the dataset respectively.
So, as it has already been mentioned, the idea was to take the bidding odds, transform them into probabilities corresponding to each outcome of every match for a certain team - a win, a lose, or a draw - and on the basis of these outcomes calculate the entropy, which is used as a proxy of uncertainty of a certain outcome, and, hence, the competitiveness of the match and league. For calculating the entropy we used the main function of the package "Entropy", which estimate the Shannon entropy of the random variable.
Here is how the code in R for calculating the entropy of matches looks like:

```

  matchEntropy = function(row) {
  odds = row[c('B365H' ,'B365D', 'B365A')]
  probs = vapply(as.numeric(odds), function(x) 1 / x , 1)
  probs_sum = sum(probs, na.rm = TRUE)
  norm_probs = vapply(probs, function(x) x / probs_sum , 1)
  entropy(norm_probs, na.rm = TRUE)
}

matches["entropy"] = apply(matches, 1, matchEntropy)
```

So, the bidding odds were taken, and then converting to probabilities by dividing 1 by them. This step is intuitively evident: the greater is the odd for a certain outcome of the match, the less is the probability that this outcome is going to happen. So odds are just a format to present probabilities as estimated by the bookkeepers. But it is also remember, that these odds are not "true" probabilities, as a gambling agency always adds profit margins in these odds. This ensures that a gambling agency will always receive some profits from gamblers, giving that the original probabilities were calculated correctly. To correct the calculations for this profit bias, we normalise the probabilities, which were received from the bidding odds, by dividing them by the sum of all the three probabilities corresponding to a certain match. After that

we applied entropy function.
To present results in a comprehensive way and spot the trends, we calculated the entropy mean for each team in chosen leagues, and calculated the entropy mean for each league across the seasons. To do that, we needed to rearrange the data we have been using so far. We created a table, which contained the figures of the probabilities entropy for every single team across all the matches. To do that, we split the database into two tables - one includes the matches entropy results for those teams, for which these matches were home matches, and the other one for guest matches - and then merged them into a united table. The following code represents this logic:

```
  home.matches = subset(matches, select=c("league_id", "
    home_team_api_id", "season", "entropy"))
names(home.matches)[2] = 'team_id'

guest.matches = subset(matches, select=c("league_id", "
  away_team_api_id", "season", "entropy"))
names(guest.matches)[2] = 'team_id'
allmatches = rbind(home.matches, guest.matches)
```

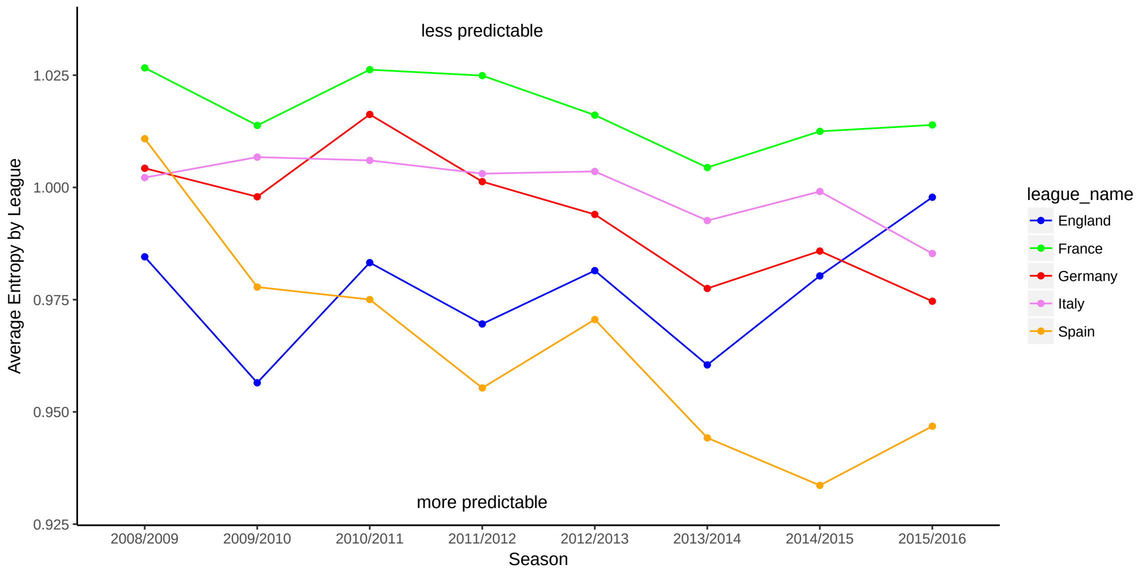
Finally, we calculated mean entropy for each of the chosen leagues and corresponding matches:

```
  team.entropy <- aggregate(allmatches$entropy,
          list(league_id = allmatches$league_id,
               team = allmatches$team_id,
               season = allmatches$season),
          FUN = function(x) mean(x, na.rm=TRUE))

league.entropy <- aggregate(matches$entropy,
          list(league_id = matches$league_id, season =
            matches$season),
          FUN = function(x) mean(x, na.rm=TRUE))
```

Results and Graphics

The first graphics represents the average entropy by league across different season, so we can see the dynamics.



The second gra

Predictability of the leagues and graphics

4 Empirical study

Figure 1: Example see related quantlets essai

1	2
3	4

Table 1: Table 1

5 Conclusion

Example : For further explanations, see ref (2017)

6 Appendices

6.1 Quantlet 1 : full code

```r
PCA = function(table, norm = T, order = 2, desiredvariance = 0.8) {
  check = apply(table, 2, is.numeric)  #We check if we have only quantitative values
  if (mean(check) == 1) {
      table = scale(table, center = TRUE, scale = norm)  #We center the dataset (but we don't scale)
      n = dim(table)[1]  #number of indivuals (rows)
      p = dim(table)[2]  #number of variables (columns)
      D = 1/n * diag(rep(1, n))  #matrix with the weights of indivuals
      Q = diag(rep(1, p))
      S = t(table) %*% D %*% table  #sample covariance matrix
      u = diag(S)
      u = 1/sqrt(u)
      D1s = diag(u)
      R = D1s %*% S %*% D1s  #sample correlation matrix
      eigenvalues = eigen(S)$values
      eigenvectors = eigen(S)$vectors
      inertia = sum(diag(S))  #inertia is the sum of the eigenvalues
      cum = cumsum(eigen(S)$values)/inertia  # cumulative energy /inertia
      i = 1
      while (cum[i] < desiredvariance) {
          i = i + 1
      }
      i = min(order, i)
      # We project the table on the i first vectors We compute the main factors and the coordinates of the individuals on the i first axes Main
      # factors are main axes since Q=Ip Gk for 1<=k<=i fulfill : Fk=table*uk where (uk)_k are the main axes
      FF = matrix(rep(0, n * i), ncol = i)  #zero matrix for storing the Fk's
      G = matrix(rep(0, p * i), ncol = i)  #zero matrix for storing the Gk's
      nor = matrix(rep(0, p * i), ncol = i)
      cor = matrix(rep(0, i * p), ncol = p)
      for (k in 0:i) {
          FF[, k] = table %*% eigenvectors[, k]
          G[, k] = sqrt(eigenvalues[k]) * eigenvectors[, k]
      }
      for (m in 1:i) {
            for (j in 1:p) {
                cor[m, j] = G[j, m]/sqrt(1/n * t(table[, j]) %*% table[, j])
            }
        }
        return(list(FF = FF, G = G, cor = cor, cum = cum, R = R)) #We return FF,G,cor,the inertia proportion and the correlation matrix
        # Coordinates of points in the area of variables
    } else {
        stop("The argument table must be a matrix/dataframe of exclusive quantitative values")
    }
}
```

References

(1999). *Classification*. Chapman and Hall.

(2001). *Cluster Analysis*. Oxford University Press Inc.

(2014). *Exploration de données et Méthodes statistiques : data analysis and data mining avec R*. Références Sciences Ellipses.

(2017). *Hybrid Framework for the Estimation of Rare Failure Event Probability*. American Society of Civil Engineers.

Norusis, M. J. (1993). *SPSS: SPSS for Windows, base system user's guide release 6.0*. SPSS Inc.