Feature Engineering

1. Feature Selection
2. Regression
3. Multicollinearity Problem
4. Factor Analysis
   - 4.1. Example
   - 4.2. Factor Analysis Process
   - 4.3. Principal Component Analysis
   - 4.4. Varimax Rotation
   - 4.5. Factor Analysis Process
   - 4.6. Varimax Rotation
   - 4.7. Factor Scores Regression
   - 4.8. Factor Scores Regression Problem
   - 4.9. Factor Scores Regression Solution
   - 4.10. Factor Scores Regression Implementation

Feature Engineering

Feature Selection

Feature selection is used to select those features that contribute most to the prediction variable that you are interested in.

Given a multidimensional dataset, extract information based on interrelations among variables.

Temperature	Time	Weight	Location	Wedding	Weatherd	Sales
30	5	100	1	1	1	1000
40	6	120	2	2	2	1200
50	7	150	3	3	3	1500
60	8	180	4	4	4	1800

Is there any interrelation among the variables? Does an interrelation among the variables, that has sold at the bar?

So to solve the right conclusion from a statistical analysis and to find out the interrelation among the variables, it is very important to understand the relationship between variables. **Predictive Solutions**

Statistical relationship among variables depend on **regression** and **correlation**.

Regression

Multicollinearity Problem

Factor Analysis

Example

Factor Analysis Process

Direction of Maximum Variance

First Principal Component (PC1)

Varimax or PCA pictures

Varimax Rotation

Problem Statement: The dataset you are going to use in this practice is the Iris flowers Iris data set. The dataset consists of 150 records of 5 attributes - sepal length, sepal width, petal length, petal width, and species. Species will be the target variable, and the other four features will be used to predict the species. The records have been sorted by the species class, first 50 records belong to "Iris-setosa", the 50 "Iris-versicolor", and the "Iris-virginica".

Objective

- Take the results of original number of features
- Rotate the factors to maximize the variance of the original variables
- Review the most important features from the dataset that are responsible for maximum variance in the output.

```
# Load the required libraries
library(psych)
library(GPArotation) #needed for load_data
# Read the data set
data <- read.csv("iris.data")
dim(data) # 150 5
head(data)
# Create a matrix
data_matrix <- as.matrix(data[, c("sepal_length", "sepal_width", "petal_length", "petal_width")])
# Run the principal component analysis
fit <- principal(data_matrix, rotate="varimax")
print(fit)
# Look at the loadings
loadings <- fit$loadings[,c("sepal_length", "sepal_width", "petal_length", "petal_width")]
# Plot the loadings
plot(loadings, type="n") # set up plot
text(loadings, labels=names(data_matrix), cex=.7) # add variable names
# Alternatively, you can use the biplot function
biplot(fit, pc1=1, pc2=2, cex=c(.7,.7))
# Or, you can use the ggbiplot function
# install.packages("devtools")
# library(devtools)
# install_github("ggbiplot", "vqv")
# library(ggbiplot)
# ggbiplot(fit, pc1=1, pc2=2, ellipse=TRUE, label.size=3, varname.size=3)
# Or, you can use the ggscatmat function
# ggscatmat(data_matrix, alpha=.5)
```

Run the above program and you will see the following output:

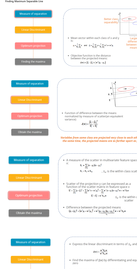
```
$ ggbiplot(fit, pc1=1, pc2=2, ellipse=TRUE, label.size=3, varname.size=3)
Loading ggplot2
Loading plyr
Loading scales
Loading grid
Loading stringr
Loading RColorBrewer
Loading plyr
Loading scales
Loading grid
Loading stringr
Loading RColorBrewer
Loading ellipse
Loading Hmisc
Loading grid
Loading lattice
Loading Formula
Loading cluster
Loading ggplot2

Attaching package: 'Hmisc'

The following object is masked from 'package:psych':

    describe

The following object is masked from 'package:base':

    format.pvalue, round.POSIXt, trunc.POSIXt, units

Warning messages:
1: In .Fortran("dsvdc", as.double(x), as.integer(n), as.integer(n),
    as.integer(p), s = as.double(s), double(n), u = as.double(u),
    v = as.double(v), double(p), as.integer(0), as.integer(11),
    PACKAGE = "base"): NaNs produced
2: In .Fortran("dsvdc", as.double(x), as.integer(n), as.integer(n),
    as.integer(p), s = as.double(s), double(n), u = as.double(u),
    v = as.double(v), double(p), as.integer(0), as.integer(11),
    PACKAGE = "base"): NaNs produced
3: In .Fortran("dsvdc", as.double(x), as.integer(n), as.integer(n),
    as.integer(p), s = as.double(s), double(n), u = as.double(u),
    v = as.double(v), double(p), as.integer(0), as.integer(11),
    PACKAGE = "base"): NaNs produced
4: In .Fortran("dsvdc", as.double(x), as.integer(n), as.integer(n),
    as.integer(p), s = as.double(s), double(n), u = as.double(u),
    v = as.double(v), double(p), as.integer(0), as.integer(11),
    PACKAGE = "base"): NaNs produced
$ head(data)
  sepal_length sepal_width petal_length petal_width species
1          NA          NA           NA          NA  setosa
2          NA          NA           NA          NA  setosa
3          NA          NA           NA          NA  setosa
4          NA          NA           NA          NA  setosa
5          NA          NA           NA          NA  setosa
6          NA          NA           NA          NA  setosa
$ dim(data)
[1] 150   5
$ data_matrix <- as.matrix(data[, c("sepal_length", "sepal_width", "petal_length", "petal_width")])
$ fit <- principal(data_matrix, rotate="varimax")
$ print(fit)
Principal Components Analysis
Call: principal(r = data_matrix, rotate = "varimax")
Standardized loadings (pattern matrix) based upon correlation matrix
     RC1 RC2 RC3 RC4 h2   u2 com
sepal_length   1   0   0   0  1 0.00 1.00
sepal_width    0   1   0   0  1 0.00 1.00
petal_length   0   0   1   0  1 0.00 1.00
petal_width    0   0   0   1  1 0.00 1.00

                       RC1  RC2  RC3  RC4
SS loadings            1.00 1.00 1.00 1.00
Proportion Var         0.25 0.25 0.25 0.25
Cumulative Var         0.25 0.50 0.75 1.00
Proportion Explained   0.25 0.25 0.25 0.25
Mean item complexity   1.00 1.00 1.00 1.00
$ print(loadings, type="n")
Loadings:

Rotation matrix:
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

               [,1] [,2] [,3] [,4]
SS loadings    1.00 1.00 1.00 1.00
Proportion Var 0.25 0.25 0.25 0.25
Cumulative Var 0.25 0.50 0.75 1.00
$ text(loadings, labels=names(data_matrix), cex=.7)
Error in plot.xy(xy.coords(x, y, xlab, ylab, log), type, ...) :
  plot.xy(.) unknown plot type 'n'
$ plot(fit, pc1=1, pc2=2, cex=c(.7,.7))
Error in plot.prc(fit, pc1 = 1, pc2 = 2, cex = c(0.7, 0.7)) :
  Unable to find the scores in 'x'
$ ggscatmat(data_matrix, alpha=.5)
Error in ggscatmat(data_matrix, alpha = .5) :
  Unable to find the scores in 'x'
$ ggbiplot(fit, pc1=1, pc2=2, ellipse=TRUE, label.size=3, varname.size=3)
Error in ggbiplot(fit, pc1 = 1, pc2 = 2, ellipse = TRUE, label.size = 3,
    varname.size = 3) :
  Unable to find the scores in 'x'
$ head(data)
  sepal_length sepal_width petal_length petal_width species
1          NA          NA           NA          NA  setosa
2          NA          NA           NA          NA  setosa
3          NA          NA           NA          NA  setosa
4          NA          NA           NA          NA  setosa
5          NA          NA           NA          NA  setosa
6          NA          NA           NA          NA  setosa
$ dim(data)
[1] 150   5
$ data_matrix <- as.matrix(data[, c("sepal_length", "sepal_width", "petal_length", "petal_width")])
$ fit <- principal(data_matrix, rotate="varimax")
$ print(fit)
Principal Components Analysis
Call: principal(r = data_matrix, rotate = "varimax")
Standardized loadings (pattern matrix) based upon correlation matrix
     RC1 RC2 RC3 RC4 h2   u2 com
sepal_length   1   0   0   0  1 0.00 1.00
sepal_width    0   1   0   0  1 0.00 1.00
petal_length   0   0   1   0  1 0.00 1.00
petal_width    0   0   0   1  1 0.00 1.00

                       RC1  RC2  RC3  RC4
SS loadings            1.00 1.00 1.00 1.00
Proportion Var         0.25 0.25 0.25 0.25
Cumulative Var         0.25 0.50 0.75 1.00
Proportion Explained   0.25 0.25 0.25 0.25
Mean item complexity   1.00 1.00 1.00 1.00
$ print(loadings, type="n")
Loadings:

Rotation matrix:
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

               [,1] [,2] [,3] [,4]
SS loadings    1.00 1.00 1.00 1.00
Proportion Var 0.25 0.25 0.25 0.25
Cumulative Var 0.25 0.50 0.75 1.00
$ text(loadings, labels=names(data_matrix), cex=.7)
Error in plot.xy(xy.coords(x, y, xlab, ylab, log), type, ...) :
  plot.xy(.) unknown plot type 'n'
$ plot(fit, pc1=1, pc2=2, cex=c(.7,.7))
Error in plot.prc(fit, pc1 = 1, pc2 = 2, cex = c(0.7, 0.7)) :
  Unable to find the scores in 'x'
$ ggscatmat(data_matrix, alpha=.5)
Error in ggscatmat(data_matrix, alpha = .5) :
  Unable to find the scores in 'x'
$ ggbiplot(fit, pc1=1, pc2=2, ellipse=TRUE, label.size=3, varname.size=3)
Error in ggbiplot(fit, pc1 = 1, pc2 = 2, ellipse = TRUE, label.size = 3,
    varname.size = 3) :
  Unable to find the scores in 'x'
$
```

From the output, the result obtained has indicated considerably stunning 90% of the variance.

Now, let's try to see what the result would be if we rotate the factors. The rotated data again can be transformed dataset and check the factors.

```
$ print(fit,cutoff=0.3,sort=TRUE)
$
$
$
```

Finding Maximum Separable Line



- Mean vector within each class of p and q

$$\mu_p = \frac{1}{n_p}\sum_{i=1}^{n_p} x_i \qquad \mu_q = \frac{1}{n_q}\sum_{i=1}^{n_q} x_i$$

- Scatter matrix for the dataset
  - Covariance matrix for each class

$$\Sigma_p = \sum_{i=1}^{n_p}(x_i - \mu_p)(x_i - \mu_p)^T$$

$$\Sigma_q = \sum_{i=1}^{n_q}(x_i - \mu_q)(x_i - \mu_q)^T$$

- Function of difference between the means
  - Measure of separation between the projected means

$$\Delta\mu = \frac{\mu_p - \mu_q}{\Sigma_p + \Sigma_q}$$

Variables from same class are projected very close to each other and at the same time, the projected means are at farther apart as possible

- A measure of the scatter in multivariate feature space

$$s_p = \sum_{i=1}^{n_p}(x_i - \mu_p)^T \qquad s_q = \sum_{i=1}^{n_q}(x_i - \mu_q)^T$$

s_p is the within class scatter matrix

- Scatter of the projections can be expressed as a function of the scatter in feature space

$$\tilde{s}_p = \sum_{i=1}^{n_p}(w^T x_i - w^T \mu_p)^2 = w^T s_p w \qquad \tilde{s}_q = \sum_{i=1}^{n_q}(w^T x_i - w^T \mu_q)^2 = w^T s_q w$$

$\tilde{s}_p$ is the within class scatter

- Difference between the projected means

$$(\tilde{\mu}_p - \tilde{\mu}_q)^2 = (w^T \mu_p - w^T \mu_q)^2 = w^T S_B w$$

- Express the linear discriminants in terms of s_p and s_q

$$\Delta\mu = \frac{S_B}{S_W}$$

- Find the inverse of taking differentiating and equating to zero

- Solving the problem of finding $\Delta\mu$ is equivalent to

$$w = (s_p + s_q)^{-1}(\mu_p - \mu_q)$$

Obtain the optimal direction for projection

Name: Labeled Feature Reduction

Problem Statement: Describe the 3-dimensional pentagonal within the mentioned library. Segregate the data and labels accordingly and perform Linear Discriminant Analysis.

Objective: Perform LDA as a dimensionality reduction technique.

```
from sklearn.datasets import load_iris
from sklearn.decomposition import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
X, y = iris.data, iris.target
X = X[y != 0, :2]
y = y[y != 0]

n_samples, n_features = X.shape
mean = np.mean(X, axis=0)
cov = np.dot(X.T, X) / n_samples
for i in range(3):
    print('class', i, 'mean', mean[i])
    print('class', i, 'cov', cov[i, i])
```

```
class 0 mean 0.9666666666666666
class 0 cov 0.7266666666666667
class 1 mean 1.0666666666666667
class 1 cov 0.7266666666666667
class 2 mean 1.0666666666666667
class 2 cov 0.7266666666666667
```

```
X = np.dot(X, np.dot(np.sqrt(n_samples), np.linalg.inverse(np.sqrt(cov))))
print('X[:5, :]\n', X[:5, :])
print('X.shape', X.shape)
```

```
X[:5, :]
 [[-0.90453403 -1.31357418]
 [-0.90453403 -1.31357418]
 [-0.90453403 -1.31357418]
 [-0.90453403 -1.31357418]
 [-0.90453403 -1.31357418]]
X.shape (100, 2)
```

Finally, we have transformed our data using LDA.

Practice Program: LDA Transformation

Problem Statement: Solve team scores with and loaded datasets. Load the iris(in dataset from sklearn (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris)) and perform Linear Discriminant Analysis on the mentioned dataset. Transform the dataset using the eigenvectors and observe the accuracy. Compare it with the previous model created in PCA in terms of accuracy and model complexity.

Objective: Load the mentioned dataset and perform LDA using Scikit-learn.

Import Data

Import the load_iris dataset from scikit-learn and split it accordingly.

```
import numpy
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=109, stratify=y)
print('X_train.shape', X_train.shape)
print('X_test.shape', X_test.shape)
```

```
X_train.shape (105, 4)
X_test.shape (45, 4)
```

Number of Components in Transformed Shape

We will use the training data.

```
from sklearn.linear_model import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None, solver='svd',
    store_covariance=False, tol=0.0001)
X_train = lda.fit(X_train, y_train).transform(X_train)
print('Number of Components in transformed shape:', X_train.shape[1])
```

```
Number of Components in transformed shape: 2
```

Model Transformation

Transforming the test data

```
X_test = lda.transform(X_test)
print('Transformed test data shape', X_test.shape)
print('X_test.shape', X_test.shape)
```

```
Transformed test data shape (45, 2)
X_test.shape (45, 2)
```

Fitting Logistic Regression Model

```
lr = LogisticRegression(multi='multinomial',penalty='l2',
C=1.0,dual=False,fit_intercept=True,
intercept_scaling=1,class_weight=None,
random_state=None,solver='newton-cg',
max_iter=100,verbose=0,warm_start=False,
n_jobs=1)
lr.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          penalty='l2', random_state=None, solver='newton-cg',
          tol=0.0001, verbose=0, warm_start=False, n_jobs=1)
```