catalogue

1.DATA DESCRIPTION 1

2.DATA PROCESS 1

3.CLASSIFICATION EXPERIMENT 4

4.REPRODUCE IN R 5

5.CONCLUSTION 6

# 1.DATA DESCRIPTION

A dataset of the product with bad or good quality, classified into several types. Each bad type was marked by code, respectively. The goal was to get a feature or a set of features that had the greatest impact on the quality of the product.

The output of the product was in column 25, and column 26 was its illustration. Column 25 and 26 were the second inspection result, and column 23 and 24 were the first result, which were useless in the final output.

Beside the output columns, there were 209 input columns, which were so-called “features”.

# 2.DATA PROCESS

There existed data which could not be processed directly, in that the classifiers could only use real numbers as input. So the dataset had to be filtered and cleaned.

Attention:

There are different ways to process the dataset, and the following process in either python or R is just reference.

Here is the python code to do this job.

|  |
| --- |
| import csv  def convert(s):  sum = 0  index = 1  length = len(s)  for i in reversed(range(length)):  sum = sum + index \* (ord(s[i]) - 64)  index = index \* 26  return sum - 1  csvRead = file('factory.csv', 'rb')  reader = csv.reader(csvRead)  dataValid = 0  data = []  for line in reader:  if(line[24] != 'NA' and line[24] != 'CONFIRM\_RESULT\_CODE'):  dataValid += 1  data.append(line)  for i in range(dataValid):  data[i].append(data[i][24])  data[i][24] = 0  #del(data[i][24])  check = []  l = len(data[0])  for i in range(l):  flag = True  for j in range(dataValid):  if(data[j][i] != data[0][i]):  flag = False  check.append(flag)  dataWrite = []  for i in range(dataValid):  temp = []  for j in range(l):  if(j == l - 1):  if(data[i][j] != '0'):  temp.append(-1)  else:  temp.append(1)  continue    if(convert('G') == j):  data[i][j].replace('-', '.')  data[i][j].replace('\_', '')  if(convert('X') == j or convert('Z') == j or convert('AL') == j or convert('AM') == j \  or convert('AW') == j or convert('AX') == j or convert('BM') == j or convert('EQ') == j \  or convert('FA') == j or convert('GG') == j):  data[i][j] = 0  if(convert('AD') == j):  data[i][j] = data[i][j][-2:]  if(convert('AJ') == j or convert('AK') == j):  data[i][j] = float(data[i][j][:2])\*60 + float(data[i][j][3:])  if(convert('G') == j or convert('AO') == j or convert('AU') == j or convert('AY') == j \  or convert('EP') == j or convert('EZ') == j or convert('GF') == j):  data[i][j] = data[i][j].replace('-', '').replace('\_', '')  #data[i][j].replace('\_', '')  if(convert('AR') == j or convert('AS') == j):  data[i][j] = 1  if(convert('FU') == j or convert('FW') == j or convert('FX') == j):  if(data[i][j] == '(1.00)'):  data[i][j] = 1  else:  data[i][j] = 0  if(convert('S') == j):  if(data[i][j].startswith('10088U')):  data[i][j] = 1  else:  data[i][j] = 0  if(check[j] == True):  temp.append(0)  else:  if(data[i][j] != 'NA'):  temp.append(data[i][j])  else:  temp.append(-1)  dataWrite.append(temp)  csvWrite = file('factory\_trans.csv', 'wb')  writer = csv.writer(csvWrite)  writer.writerows(dataWrite)  csvRead.close() |

Our main process are as follows.

1. we processed the factory data column by column and then wrote the cleaned data into a new file, as the input of the regression and classification experiment.
2. according to the data in the output column, we discarded the invalid rows of data whose output field was ‘NA’ and counted the number of rows of valid data. (about 90 lines of data were discarded)
3. in some columns, the data were totally the same, so these columns were useless in our experiment. We just filled these columns with ‘0’. (about 12 columns)
4. after discarding invalid rows and columns, we process the data in some selected columns.

* symbol ‘-’ and ‘\_’ in column G were replaced to form a valid number to process.
* several columns were useless explained by below, such as column X and Z.
* retained the last two digits of AD, in that the data in column AD were different with each other by last two digits.
* transformed the time span into real number to process, such as the column AJ and AK.
* some columns were specially processed because their unique format, for example, replaced ‘(1.00)’ with ‘1’ and ‘0’ otherwise in column S.
* discarded all date & time data in that we believe they were of little use.

The implementation of main steps in R is as follows.

* Read in data

|  |
| --- |
| >data <- read.csv("factory.csv", header= T, sep = ",") |

* Replace some useless columns and clean the identical data in the same column. (Numbers in the vector c are the order number of the columns to be cleaned.)

|  |
| --- |
| >to\_clear = c(3, 4, 14, 15, 16, 17, 21, 22, 25, 27, 28, 34, 35, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 182, 183, 184, 185, 186, 187, 214, 215, 24, 26, 38, 39, 44, 45, 49, 50, 65, 147, 157, 189)  >for(i in to\_clear){data[, i] = 0} |

* Strip some symbols in the data.

|  |
| --- |
| > to\_replace <- c(7, 41, 47, 51, 146, 156, 188)  > rows <- c(1:1000)  > for(i in rows){for(j in to\_replace){data[i, j] <- sub("\_","",data[i, j])}}  > for(i in rows){for(j in to\_replace){data[i, j] <- sub("-","",data[i, j])}} |

* Get the special part of the data (last two digits).

|  |
| --- |
| > for(i in rows) {data[i, 40] <- substr(x = data[i, 40], start = 6, stop = 7)} |

* Replace data Specially.

|  |
| --- |
| >to\_replace\_special <- c(178, 179, 180)  > for(i in rows){for(j in to\_replace\_special){if(data[i, j] == “(1.00)”){data[i, j] <- 1}else{ data[i, j] <- 0}}}  > for(i in rows){for(j in to\_replace\_special){if(data[i, j] == “(1.00)”){data[i, j] <- 1}else{ data[i, j] <- 0}}} |

* Strip invalid data.

|  |
| --- |
| >data <- na.omit(data) |

# 3.CLASSIFICATION EXPERIMENT

We conducted the classification experiment using the open-source SVM tool kit – libsvm. It could directly run the data as long as we transformed the data into the format required in the shell or bash environment.

# 4.REPRODUCE IN R

e1071 is a package for R. It contains most machine learning tools such as SVM, Naïve Bayes, etc. Alas, it is convenient, fast, and efficient. We implemented the whole process using the SVM module in package e1071. So before run the command, please make sure the package e1071 have been installed and imported by R.

Here are the command to get the accuracy of classifying the dataset.

|  |
| --- |
| data <- read.csv("factory\_trans.csv", header=FALSE)  inputData <- data.frame(data[, c(6, 11, 131)], response = as.factor(data[, c(216)]))  svmfit <- svm(response ~ ., data = inputData, kernel = "radial", cost = 10, gamma = 1000, scale = FALSE)  result = mean(inputData$response == predict(svmfit))  print(result) |

Tips:

1. ”factory\_trans.csv” refers to the file of input dataset, which were processed by the code written in python below.
2. ”c(6, 11, 131)” refers to the features which have been selected by the classification experiment, as the input data. In the FACTORY data, “6” refers to the 6th feature, that is feature F, and “11” refers to K, “131” refers to DA. “c(25)” refers to the output.
3. the parameters of SVM is showed by “kernel = "radial", cost = 10”. The parameter “gamma” can be ignored in this case. We will elaborate the parameters later.
4. we can get the accuracy of classification immediately after running the command in the last line.

Here are the parameter optimization process in R.

|  |
| --- |
| set.seed(100)  rowIndices <- 1 : nrow(inputData)  sampleSize <- 0.8 \* length(rowIndices)  trainingRows <- sample (rowIndices, sampleSize)  trainingData <- inputData[trainingRows, ]  testData <- inputData[-trainingRows, ]  tuned <- tune.svm(response ~., data = trainingData, gamma = 10^(-3:1), cost = 10^(1:3))  summary (tuned) |

We loop the parameters iteratively to find the best combination. In the instance, we tries the gamma by 0.001, 0.01, 0.1, 1, 10 and cost by 10, 100, 1000, and finally get the best are 10 and 10, respectively.

# 5.CONCLUSTION

We first filtered and cleaned the dataset by empirical methods, discarding invalid data and transformed the bad ones in python. We used libsvm to get the combination of features that achieved best accuracy, and then reproduced the result in R. What’s more, we iteratively found the best parameters of SVM to get better accuracy.

The best combination of features were F, K, DA. This subset can get about 75% correct rate using libsvm and over 82% in R. The difference between the results laid in the training set were not the same. While R used the whole dataset as training set, the libsvm used only 4/5 of dataset one time because of initial restriction.