Data Mining: BeeViva Challenges

MARCO BELLÒ, University of Padua, Italy

All R source code, images, LaTeX sources and datasets (in csv format), can be found at the following repository: https://github.com/mhetacc/DataMiningChallenges/.

ACM Reference Format:
Marco Bellò. 2025. Data Mining: BeeViva Challenges. 1, 1 (December 2025), 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 RICE VARIETIES

1.1 Preliminary Observations

1.1.1 Dataset. All following considerations are made using the datasets provided on the challenge page (*rice_test.csv* and *rice_train.csv*), but it is worth noting that they both stem from an original one that can be found at the dataset source page. I used it to compute an alternative version of the test dataset that contains the true attribute for *Class* (listing 1).

Listing 1. Merge datasets

```
df1 = pd.read_csv("rice_test.csv").round(10)
df2 = pd.read_csv("Rice_Cammeo_Osmancik.csv").round(10)

keys = [col for col in df1.columns]
merged = df1.merge(df2, on=keys, how="inner")
```

1.1.2 Scatter Plot. From the *scatter plot* (figure 1) we can infer that there are four features (*Area*, *Perimeter*, *Convex_Area* and *Major_Axis_Length*) that seem to be extremely correlated

1.1.3 Correlation Matrix. We can use a correlation matrix (table 1) to see exactly how related to each other the features are. As suspected, all four features before-mentioned have a high degree of correlation (over ninety percent). Once way to handle this is to either use models robust against collinearity, or to either combine or remove some of the correlated features.

1.1.4 Features' Importance. To get a better idea on the importance of each feature, I trained a simple linear regression model, and looked at the result with `summary(fit)`, which can be seen in table 2. The significance stars tells us visually that, with the exception of *Minor_Axis_Length*, the features that are highly correlated to each other contributes strongly to the model, while the *Eccentricity* and *Extent* contribute very little.

Author's address: Marco Bellò, marco.bello.3@studenti.unipd.it, University of Padua, Padua, Italy.

Manuscript submitted to ACM

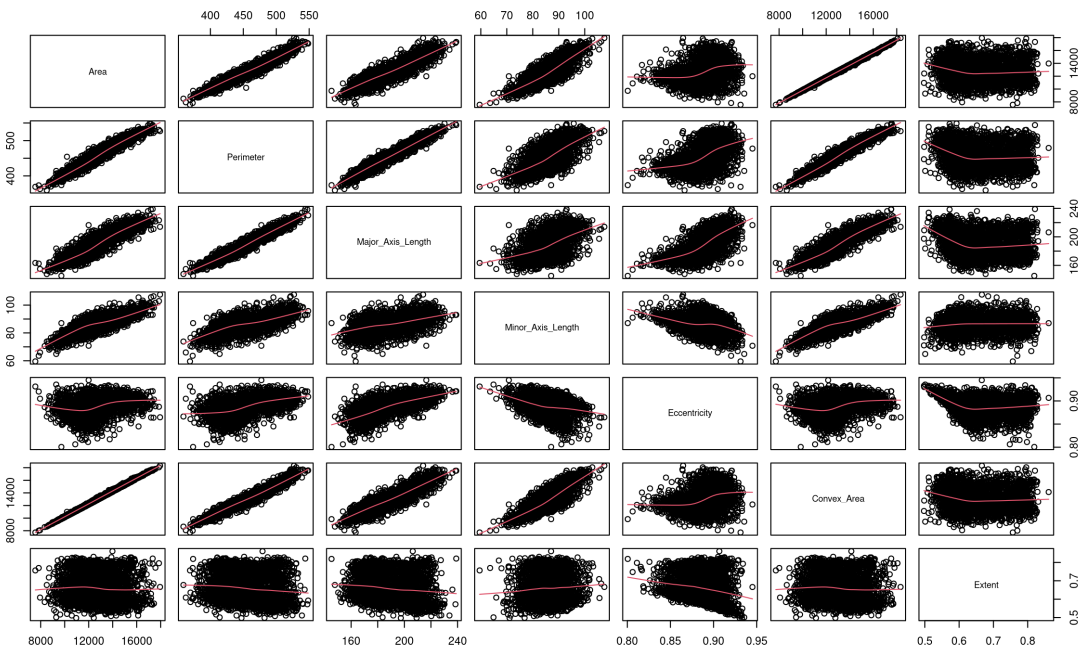


Fig. 1. Scatter Plot for the *rice_train* dataset with LOESS smoothing lines for each class.

Table 1. Correlation matrix of features

	Area	Perimeter	Major Axis Length	Minor Axis Length	Eccentricity	Convex Area	Extent
Area	1.00000000	0.96704011	0.90356325	0.79022701	0.34653177	0.99895272	-0.06002223
Perimeter	0.96704011	1.00000000	0.97163180	0.63486482	0.53784650	0.97046788	-0.12718015
Major Axis Length	0.90356325	0.97163180	1.00000000	0.45675159	0.70625660	0.90390912	-0.13562592
Minor Axis Length	0.79022701	0.63486482	0.45675159	1.00000000	-0.29393931	0.78975480	0.06192558
Eccentricity	0.34653177	0.53784650	0.70625660	-0.29393931	1.00000000	0.34707340	-0.19301157
Convex Area	0.99895272	0.97046788	0.90390912	0.78975480	0.34707340	1.00000000	-0.06397213
Extent	-0.06002223	-0.12718015	-0.13562592	0.06192558	-0.19301157	-0.06397213	1.00000000

We can also measure the total variance explained by all predictors combined using $R^2 = 0.6953849$, and we can check the standardized coefficients to better compare predictors' importance, as shown in table 3. Once again, we can see that Extent and Eccentricity contribute very little to the overall model.

1.1.5 Principal Components. A good way to aggregate and simplify data is by decomposing it into its principal components (centered in zero and scaled to have unit variance). Components' summary can be seen in table 4, and it shows us that more than ninety nine percent of overall variance can be explained with just three components, so a 2D visualization with only the first two components should give us a good idea of the whole dataset (figure 2). The plot suggests that data is well separated, with similar classes clustering nicely. This should make classification easier.

Table 2. Regression coefficients and coefficients' importance

Parameter	Estimate	Std. Error	t value	Pr(>|t|)	Importance
(Intercept)	-2.152e+00	1.990e+00	-1.081	0.279633	
Area	5.601e-04	1.023e-04	5.474	4.79e-08	***
Perimeter	8.440e-03	2.221e-03	3.800	0.000148	***
Major_Axis_Length	-2.198e-02	5.709e-03	-3.851	0.000120	***
Minor_Axis_Length	4.669e-02	1.213e-02	3.850	0.000121	***
Eccentricity	4.534e+00	1.828e+00	2.481	0.013170	*
Convex_Area	-8.609e-04	9.418e-05	-9.141	< 2e-16	***
Extent	7.146e-02	7.144e-02	1.000	0.317237	

Table 3. Standardized regression coefficients

Parameter	Standardized Coefficient
(Intercept)	NA
Area	1.95970669
Perimeter	0.60605955
Major_Axis_Length	-0.77272038
Minor_Axis_Length	0.54256082
Eccentricity	0.18931314
Convex_Area	-3.09099064
Extent	0.01114328

Table 4. Summary of principal components

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Standard deviation	2.2924	1.2436	0.9562	0.51393	0.10621	0.07758	0.04519	0.02052
Proportion of Variance	0.6569	0.1933	0.1143	0.03302	0.00141	0.00075	0.00026	0.00005
Cumulative Proportion	0.6569	0.8502	0.9645	0.99753	0.99894	0.99969	0.99995	1.00000

1.2 Modeling

After exploring the dataset, I tried to find the best performing model for predicting the target.

1.2.1 Linear Regression. First I split the training dataset into training and validation sets, fitted the model and compute its *MSE*. From this basic regression I got $MSE = 0.07354557$. Since we know from the preliminary observations that some features are highly correlated (so the dataset suffers from collinearity), I tried to transform the dataset to reduce it. A straightforward solution is to compute the average of all of them (listing 2), but this yield a slightly worse result: $MSE \approx 0.074$.

Another approach would be to drop features altogether, but first we want to measure their variance inflation factor. As a rule of thumb, a *VIF* value above then indicates a problematic amount of collinearity, so I tired to drop features iteratively to see if the model improves, as shown in table 5. For each dropped variable I computed the *VIFs* and the *MSE* of the resulting model. As we can see, dropping *Area* resulted in performances similar to combining the variables, while dropping the other features worsened prediction performances even further.

Fig. 2. Plot with first two principal components of *rice_train* dataset.

Table 5. Effect of variable removal on variance inflation factors (VIF) and mean squared error (MSE)

Variable Dropped	VIF Perimeter	VIF Major_A_L	VIF Minor_A_L	VIF Ecc.	VIF Convex_A	VIF Extent	MSE
Area	114	206	133	53	332	1	0.0735629
Convex Area	108	134	40	49	–	1	0.07572877
Major Axis Length	47	–	37	29	–	1	0.07718435
Perimeter	–	–	1	1	–	1	0.0806868

Listing 2. Average of highly correlated features

```
rice.train$Combined <- rowMeans(rice.train[, c("Area","Perimeter","Major_Axis_Length","Convex_Area")])

linear_fit = lm(
  Class ~
  Minor_Axis_Length
  + Eccentricity
  + Extent
  + Combined,
  data=rice_train,
  subset = train
  )
```

```

pred <- predict(linear_fit, newdata = rice_train[test, ])
mean((pred - y.test)^2)     # [1] 0.07439239
```

1.2.2 Ridge Regression. I used the *glmnet* library, which takes care of the variable standardization for us, has a built-in cross-validation functionionality to compute the best possible λ. With code 3 we get a result worse than linear regression, since $MSE = 0.07924703$. That being said, we can see in plot 3 an effective shrinkage of most coefficients towards zero, with the exception of Eccentricity, which we already recognized as not very impactful on the overall variance and prediction.

Listing 3. Ridge regression on rice dataset

```
library(glmnet)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
plot(cv.out)
bestlam <- cv.out$lambda.min
cv_mse <- min(cv.out$cvm)     # [1] 0.07924703
cv_rmse <- sqrt(cv_mse)       # [1] 0.2815085
```

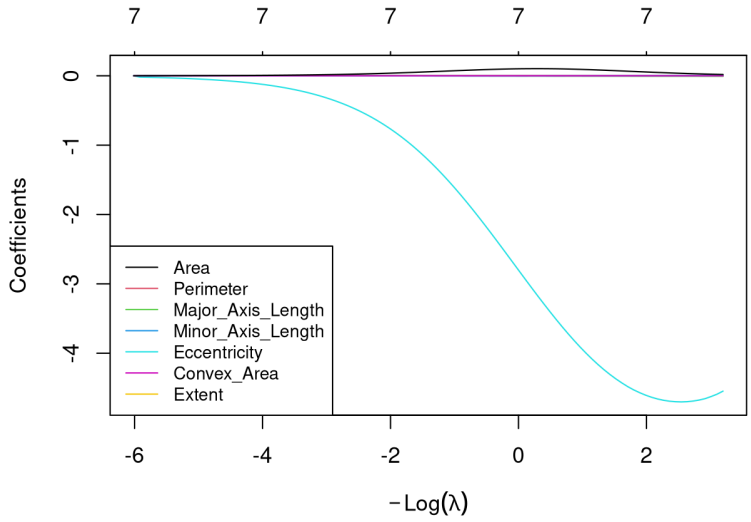


Fig. 3. Ridge coefficients for *rice_train* dataset.

1.2.3 Lasso Regression. Very similar to ridge regression, except *glmnet* gets called with `alpha = 1`. We get a slightly better result than ridge, with $MSE = 0.07348723$. From the plot (in figure 4), we can see that once again Eccentricity seems to resist shrinkage, but overall coefficient reduction is much more accentuated compare to ridge, since lasso can set them to zero.

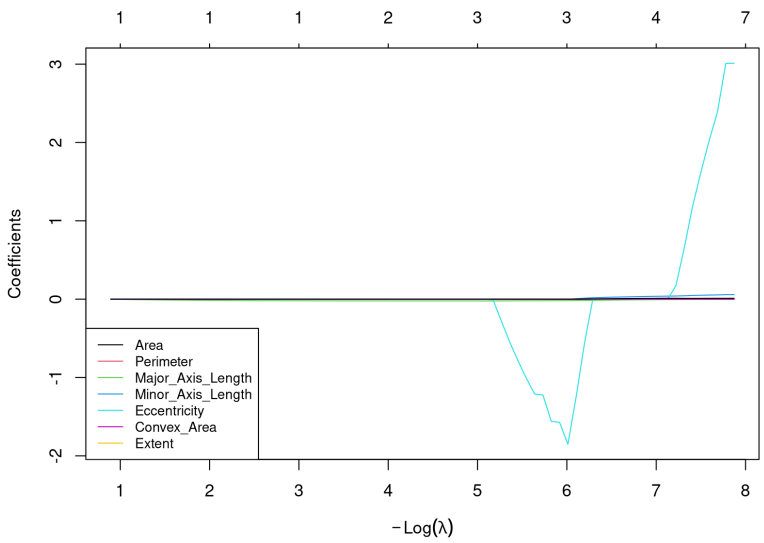


Fig. 4. Lasso coefficients for *rice_train* dataset.

1.2.4 Principal Components Regression. I fitted the model with *pls* library built-in cross-validation, as seen in code 4. The lowest error (figure 5) is achieved with seven components: $MSE = 0.07354557$, slightly worse than lasso regression.

Listing 4. Principal components regression on rice dataset

```
pcr.fit <- pcr(
  Class ~ .,
  data = rice_train,
  subset = train,
  scale = TRUE,
  validation = "CV"
  )
```

1.2.5 Partial Least Squares. I used the same *pls* library to fit the partial least squares model on the dataset. The best result is obtained with just two components (figure 8), yielding $MSE = 0.07609556$, which is slightly worse than *PCR* but needs a third of the components, potentially leading to less overfitting and computational cost.

1.2.6 LOESS. I then tried some non-parametric methods, starting with *LOESS* regression. This method allows up to four predictors, so I used the most useful ones previously identified: *Area*, *Perimeter*, *Convex_Area* and *Major_Axis_Length*. Predictors are normalized by default. Then, fitted three models with three different values for `span`. Predictions must be sanitized since there some test points can fall outside the local neighborhood, meaning being `Na`. With $span = 0.1$ we have five such predictions. The code can be seen at 5. The results were disastrous, so I tried to include only two predictors. This yielded the best results yet out of all models: with $span = 0.1$, $MSE = 0.06099749$. All results are summarized in table 6

Listing 5. LOESS regression on rice dataset with four predictors and $span = 1$

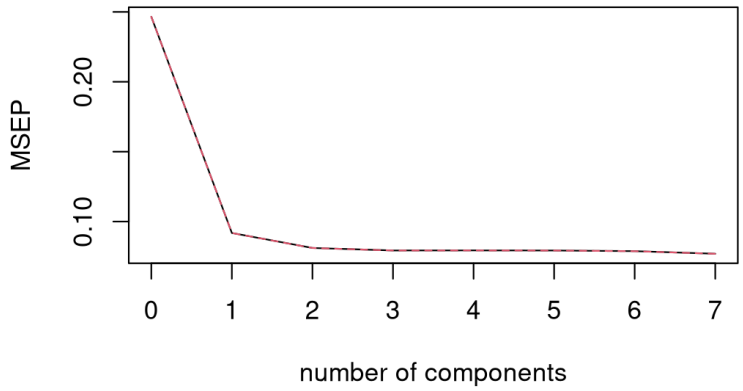


Fig. 5. PCR cross-validation MSE error for target *Class* on *rice_train* dataset.

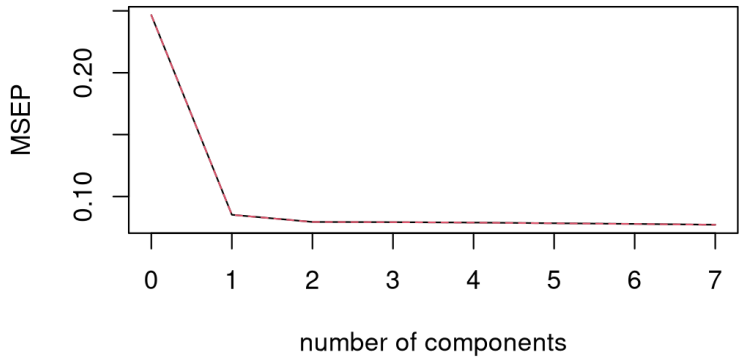


Fig. 6. PLS cross-validation MSE error for target *Class* on *rice_train* dataset.

```r
loess_fit1 <- loess(
  Class ~ Area
  + Perimeter
  + Convex_Area
  + Major_Axis_Length,
  data = rice_train,
  subset = train,
  span = 0.1
  )

pred1 <- predict(loess_fit1, newdata = rice_train[test, ])
mean((pred1 - y.test)^2, na.rm = TRUE)    # [1] 46.0226
```

1.2.7 K-Nearest Neighbors. I used the *caret* library to fit *KNN* classification both for regression and classification. The former to get a comparable prediction with the other models, the latter to be more logically sound, since *Class* is a categorical target. The code for classification can be seen at 6, for regression the only difference is the `metric` parameter:

Table 6. MSE for LOESS with different *span* values and number of predictors

Span	Number of Predictors	MSE
0.1	Four (4)	46.0226
0.5	Four (4)	1.799422
0.9	Four (4)	0.2276324
0.1	Two (2)	0.06099749
0.5	Two (2)	0.06514592
0.9	Two (2)	0.06836381

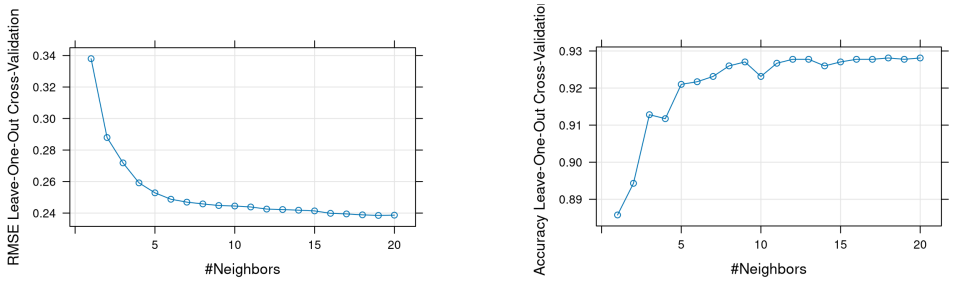
"Accuracy" for classification, "RMSE" for regression. For both methods I used Leave-One-Out-Cross-Validation, and both RMSE plots can be seen in figure 7.

For regression, the best result is achieved with nineteen neighbors, yielding a mean square error even better than LOESS: $MSE = 0.05688406$. For classification, the best accuracy is achieved with twenty neighbors, yielding an accuracy of almost 93%. This last model is the one I ultimately used to predict the target values on the test dataset.

Listing 6. KNN regression on rice dataset with Leave-One-Out-Cross-Validation

```
train.control <- trainControl(method  = "LOOCV")
train$Class <- as.factor(train$Class) # necessary for classification

knn_fit <- train(
  Class ~ .,
  method     = "knn",
  tuneGrid   = expand.grid(k = 1:20),
  trControl  = train.control,
  preProcess = c("center","scale"),    # normalized
  metric     = "Accuracy",
  data       = train
  )
```



(a) RMSE of KNN for regression

(b) Accuracy of KNN for classification

Fig. 7. RMSE and Accuracy respectively of KNN for regression and for classification with *LOOCV* on *rice_train* dataset.

1.2.8 Random Forest. The last method I tried is *random forest* with Leave-One-Out-Cross-Validation, using the *caret* library still. Since this method is extremely computationally expensive, I had to leverage the library *doParallel* to use

all available CPU cores, and I used the *ranger* method instead, which is a faster random forest implementation. I also reduced the amount of predictors by combining the highly correlated ones (we already saw that dropping them yield similar to worse results). The code can be seen at **??**.

The optimal model, with $mtry = 2$, $splitrule = extratress$ and $min.node.size = 5$, yielded a slighly worse result than KNN trained on all features: $MSE = 0.0589397$. The plot for *RMSE* can be seen in figure **??**

Listing 7. Random Forest regression on *rice_train* dataset with Leave-One-Out-Cross-Validation

```
train.control <- trainControl(method  = "LOOCV")

rf_fit <- train(
  Class ~ Minor_Axis_Length + Eccentricity + Extent + Combined,
  method     = "ranger",
  tuneLength = 10,
  trControl  = train.control,
  metric     = "RMSE",
  data       = rice_train,
  allowParallel = TRUE
)
```

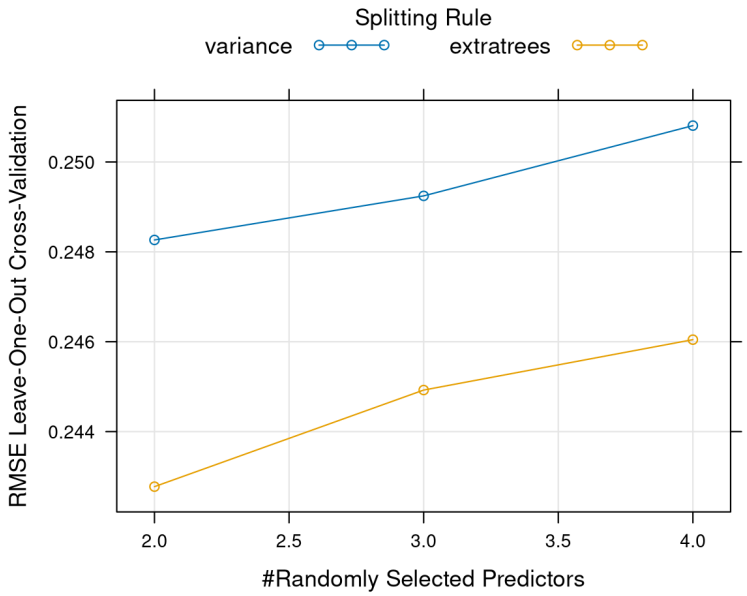


Fig. 8. RMSE error for random forest on *rice_train* dataset.

1.3 Rice Varieties Conclusions

The best model so far is KNN with $K = [19, 20]$. While slightly more computationally expensive than other methods (with the exception of random forest) it improves MSE by 0.00411343 over the second best method (LOESS with $span = 0.1$). I decided to use predict the target with a KNN fit in classification mode, since I felt it more logically sound.

I decided not to try random forest with all predictors for time constraints: it required a 10-Cores machine (i7-13620H @ 4.90 GHz) twenty-five minutes to fit the model.

2 PHONE USERS

2.1 Preliminary Observations

2.1.1 Dataset. Let's first briefly analyze the dataset.

- **For each user:**
  - Plan type
  - Payment method
  - Sex
  - Activation zone
  - Activation channel
  - Value-added service 1
  - Value-added service 2
  - **For each month:**
    - |expensive calls|
    - |cheap calls|
    - time(expensive calls)
    - time(cheap calls)
    - cost(expensive calls)
    - cost(cheap calls)
    - |incoming calls|
    - time(incoming calls)
    - |SMS|
    - |calls to call center|

Considering that the target is monthly call time we can make the hypothesis that some features do not interest us, for example any monthly feature not concerning with call time itself, leaving us with only $time(expensivecalls)$ and $time(cheapcalls)$, which we could further assume can be combined since overall cost is not a concern. It goes without saying that all of the above will have to be be proven empirically.

2.1.2 Scatter Plot. I filtered out all not call time-related monthly data. The result (figure 9) is a bit hard to read due to its sheer size. Looking at it more up close we can infer some things: first of all we see that call data looks positively skewed (figure 10), and that there are some activation channels that see more call time than others, while activation regions seem to differ lightly between one another (figure 11a). On the other hand, plan and sex seem to have an impact on call time (figure 11b), while age does not seem to have a big impact, with the exception of the very young and very old (figure 11c).

2.1.3 Call Features Over Everything Else. Let's try to see if calls amount and time are related to other features. First I plotted total calls taken and total calls time over nine months (figure 13). We can see that the graphs are almost identical from a trend standpoint, hinting at a strong correlation between number of calls and time spent calling.

Then I wanted to see whether sex plays any role in total call time. Using the library *dplyr* I computed all data in table 7. On average, men's calls seem to last 8% longer. To mitigate the effect of outliers I filtered out the top 1%, which still shows the same trend, albeit reduced to 5%. Cutting even further the top 10% shows a bigger increase in men's

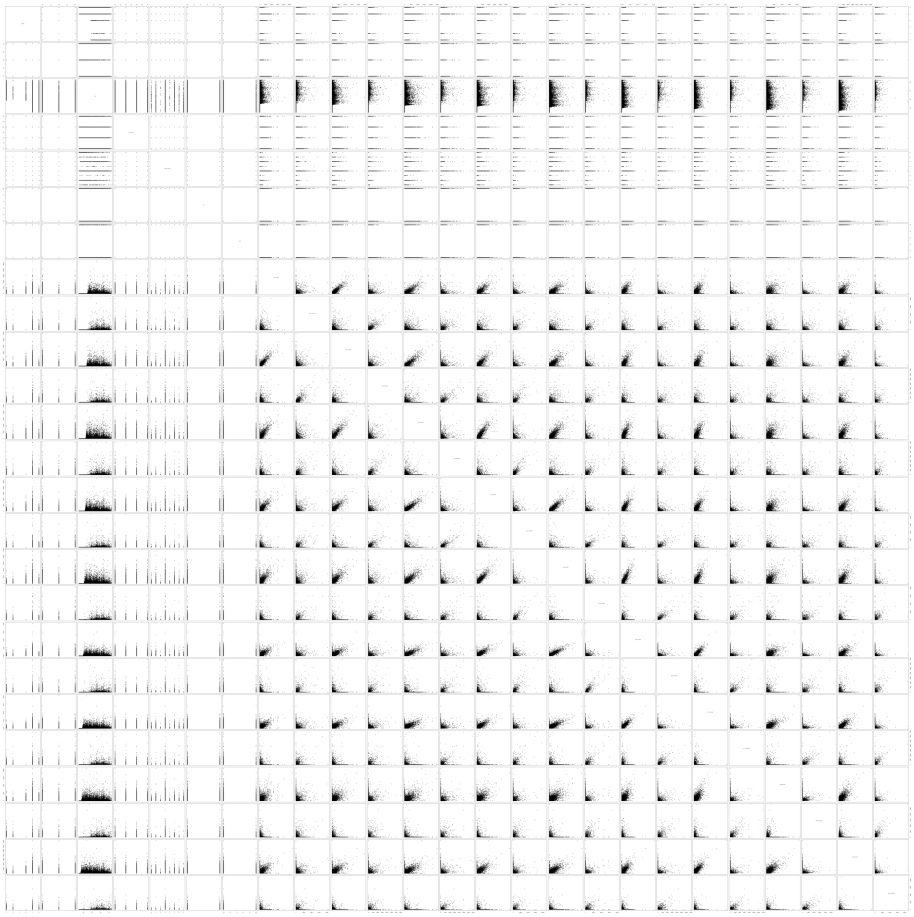


Fig. 9. Scatter Plot for the filtered *phone_train* dataset.

average call times of about 10%. Lastly, I tired to log-transform the data. Even in this situation we see that men have longer calls, hinting that this predictor may be useful. I did exactly the same with payment methods, activation zones, activation channels and both value-added services, always cutting off the top 1%,. All data can be seen in table 8. There seem to be quite a bit of variance in call time compared to the tariff plan, which would make sense: some plans may be geared toward calling, while others are more suited sending SMS. We can see some variance on payment method too. Activation zone and activation channel seem to have some influence on call times, with the exception of zone eight, whose call times are way lower. On the other hand, customers that have activated either first or second added-value services seem to do longer calls.

2.1.4 Skewness. Call time data is usually positively skewed, with many users having low usage and a small number of user with very high usage (e.g.,50.000 seconds). This is easily verifiable at a glance if we look at the density graph in figure 13a. This effect can be mitigated by log-transforming the data, as shown in figure 13b. Moreover, with the *e1071* library I computed a skewness matrix for all features (appendix 8): we can see that all monthly-based data (calls, call

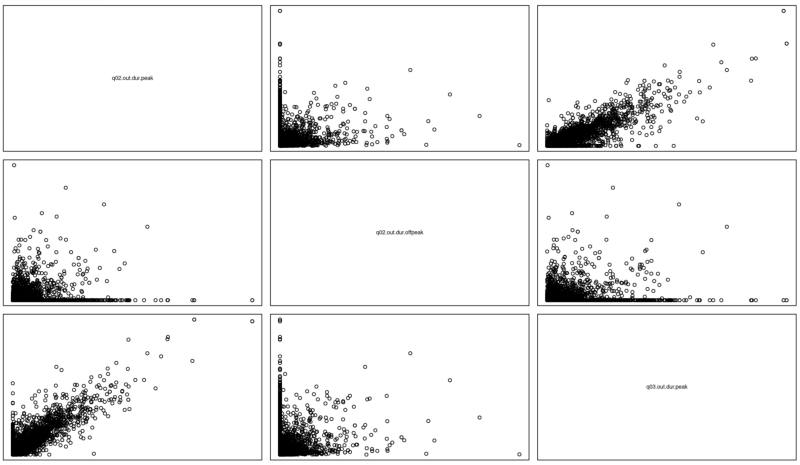
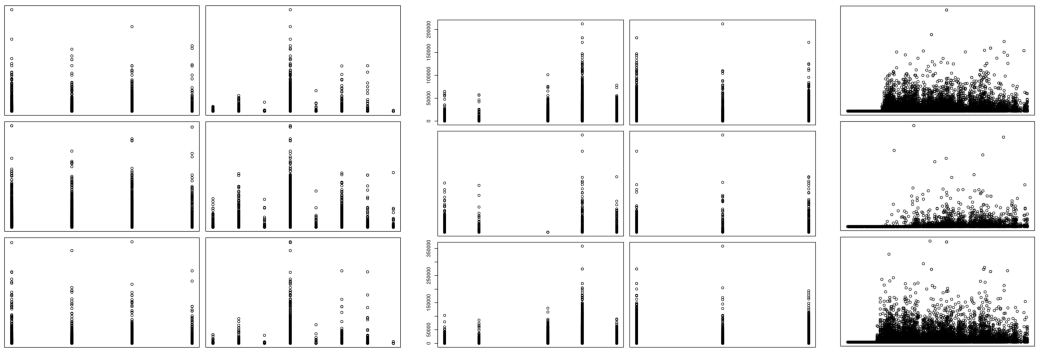


Fig. 10. Scatter Plot for call time for *phone_train* dataset.



(a) Activation zone (X-asis, L) and activation channel (X-axis, R) over call time (Y axis)

(b) Plan type (X-axis, L) and sex (X-axis, R) over call time (Y axis)

(c) Age (X-axis) over call time (Y-axis)

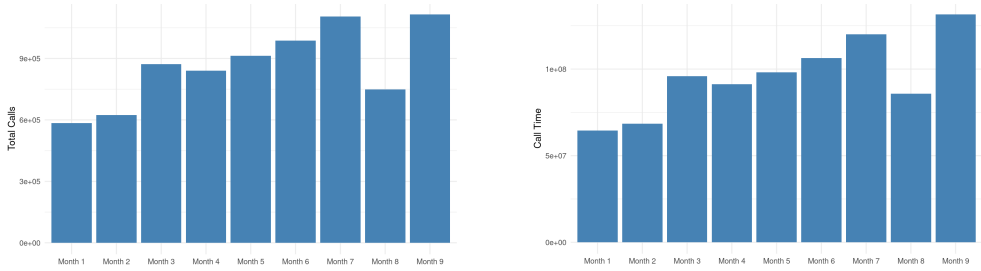
Fig. 11. Different features plotted over call time.

times, SMS, costs, etc) are highly positively skewed. To manage this problem we can either transform them (log, square root or Box-Cox transform), or use more robust models (such as random forest, Gamma/Poisson or quantile regression). Methods like linear. ridge or lasso regression, and even some non-parametric ones like KNN are not ideal.

2.1.5 *Correlation Matrix.*

2.2 Modeling

2.3 Phone Users Conclusions



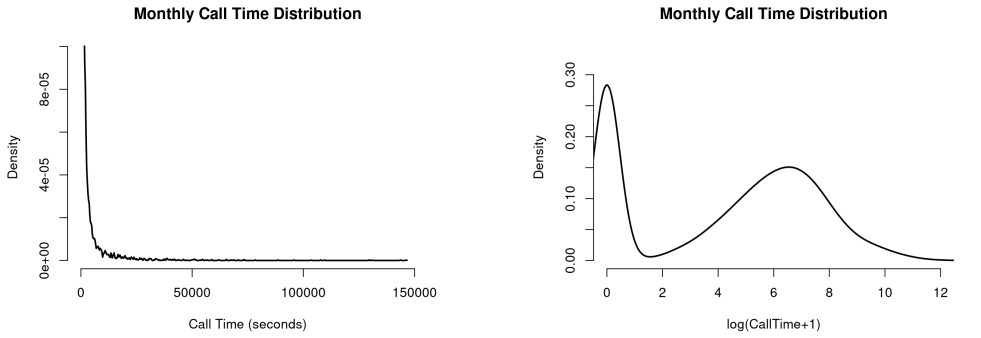
(a) Total amount of calls per month

(b) Total call time per month

Fig. 12. Total calls and total call time per month.

Table 7. Call time statistics by sex under different data cuts and transformations

Type of cut	Sex	Number of customers	Total call time	Average call time
None (raw)	B	5266	557266412	105823
None (raw)	F	1030	62493290	60673
None (raw)	M	3704	242226228	65396
Top 1% cut	B	5201	498846876	95914
Top 1% cut	F	1025	57203392	55808
Top 1% cut	M	3674	215801665	58738
Top 10% cut	B	4595	276063673	60079
Top 10% cut	F	960	32400501	33751
Top 10% cut	M	3445	130747836	37953
Log-transformed	B	5266	557266412	10.00
Log-transformed	F	1030	62493290	8.24
Log-transformed	M	3704	242226228	8.58



(a) Call time density distribution

(b) Call time density distribution (log-transformed)

Fig. 13. Call time density distributions.

Table 8. Call time over different categorical features

Feature	Number of customers	Total call time	Average call time
Tariff plan 3	781	44811041	57376
Tariff plan 4	83	10575009	127410
Tariff plan 6	724	92214268	127368
Tariff plan 7	3564	489635465	137384
Tariff plan 8	4748	134616150	28352
Activation zone1	3507	288670845	82313
Activation zone2	3130	213438509	68191
Activation zone3	2342	197941555	84518
Activation zone4	921	71801024	77960
Activation channel 2	130	10643060	81870
Activation channel 3	408	36663577	89862
Activation channel 4	31	2369822	76446
Activation channel 5	7135	623961079	87451
Activation channel 6	47	4352853	92614
Activation channel 7	652	62777153	96284
Activation channel 8	111	12074180	108776
Activation channel 9	1386	19010209	13716
Value-added one: No	7450	526620961	70687
Value-added one: Yes	2450	245230972	100094
Value-added two: No	9279	695404226	74944
Value-added two: Yes	621	76447707	123104

A APPENDICES

A.1 Phone call time skewness

Listing 8. Skweness matrix

```
        tariff.plan  activation.channel     q01.out.ch.peak    q01.out.dur.peak    q01.out.val.peak
          -2.026605            1.073333            4.930308            5.883900           10.158610
q01.out.dur.offpeak q01.out.val.offpeak       q01.in.ch.tot      q01.in.dur.tot          q01.ch.sms
           9.336163           31.977903            4.322467            3.615376           37.220847
    q02.out.ch.peak    q02.out.dur.peak    q02.out.val.peak  q02.out.ch.offpeak q02.out.dur.offpeak
           4.011558            5.026925            6.120357            9.636244            9.202850
      q02.in.ch.tot      q02.in.dur.tot          q02.ch.sms          q02.ch.cc     q03.out.ch.peak
           3.657302            3.444048           38.464828           10.697464            3.742071
   q03.out.val.peak  q03.out.ch.offpeak q03.out.dur.offpeak q03.out.val.offpeak       q03.in.ch.tot
           4.144170            8.709344            9.948054           25.074765            3.359255
         q03.ch.sms          q03.ch.cc     q04.out.ch.peak    q04.out.dur.peak    q04.out.val.peak
          35.231226           12.146903            3.726583            4.415209            3.988589
q04.out.dur.offpeak q04.out.val.offpeak       q04.in.ch.tot      q04.in.dur.tot          q04.ch.sms
          12.366396           10.517232            3.362859            3.660378           33.513288
    q05.out.ch.peak    q05.out.dur.peak    q05.out.val.peak  q05.out.ch.offpeak q05.out.dur.offpeak
           3.578910            4.056863            4.242749           12.129119           12.598793
      q05.in.ch.tot      q05.in.dur.tot          q05.ch.sms          q05.ch.cc     q06.out.ch.peak
           3.450009            3.165951           38.804264            9.085293            3.364608
   q06.out.val.peak  q06.out.ch.offpeak q06.out.dur.offpeak q06.out.val.offpeak       q06.in.ch.tot
```

```
            4.037063           15.035554           12.511158           10.149751            3.386507
          q06.ch.sms          q06.ch.cc     q07.out.ch.peak    q07.out.dur.peak    q07.out.val.peak
           28.668984           13.747535            3.157588            4.088846            3.830199
q07.out.dur.offpeak q07.out.val.offpeak       q07.in.ch.tot      q07.in.dur.tot          q07.ch.sms
           12.907377           10.128761            3.237636            3.192192           28.628610
    q08.out.ch.peak    q08.out.dur.peak    q08.out.val.peak  q08.out.ch.offpeak q08.out.dur.offpeak
            3.963771            4.131527            4.117853            9.256204           11.536043
      q08.in.ch.tot      q08.in.dur.tot          q08.ch.sms          q08.ch.cc     q09.out.ch.peak
            3.972012            3.953185           20.317293            8.946827            2.927636
   q09.out.val.peak  q09.out.ch.offpeak q09.out.dur.offpeak q09.out.val.offpeak       q09.in.ch.tot
            3.646469           10.344301           15.974234           11.674903            2.728376
          q09.ch.sms          q09.ch.cc                   y
           16.048910           13.034136           10.820427
```