

# 1 Data Mining: BeeViva Challenges

---

2  
3 MARCO BELLÒ, University of Padua, Italy  
4

5 All R source code, images, L<sup>A</sup>T<sub>E</sub>X sources and datasets (in csv format), can be found at the following repository: <https://github.com/mhetacc/DataMiningChallenges/>.  
6  
7

8 **ACM Reference Format:**

9 Marco Bellò. 2025. Data Mining: BeeViva Challenges. 1, 1 (December 2025), 10 pages. <https://doi.org/XXXXXX.XXXXXXXX>  
10

11 **1 RICE VARIETIES**

12 **1.1 Preliminary Observations**

13 *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).  
14  
15

20 Listing 1. Merge datasets  
21

```
22 1 df1 = pd.read_csv("rice_test.csv").round(10)
23 2 df2 = pd.read_csv("Rice_Cammeo_Osmancik.csv").round(10)
24 3
25 4 keys = [col for col in df1.columns]
26 5 merged = df1.merge(df2, on=keys, how="inner")
```

---

27  
28 *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  
29  
30

31 *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.  
32  
33

34 *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.  
35  
36

---

37 Author's address: Marco Bellò, marco.bello.3@studenti.unipd.it, University of Padua, Padua, Italy.  
38

---

39 Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not  
40 made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components  
41 of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on  
42 servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.  
43 © 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
44 Manuscript submitted to ACM  
45

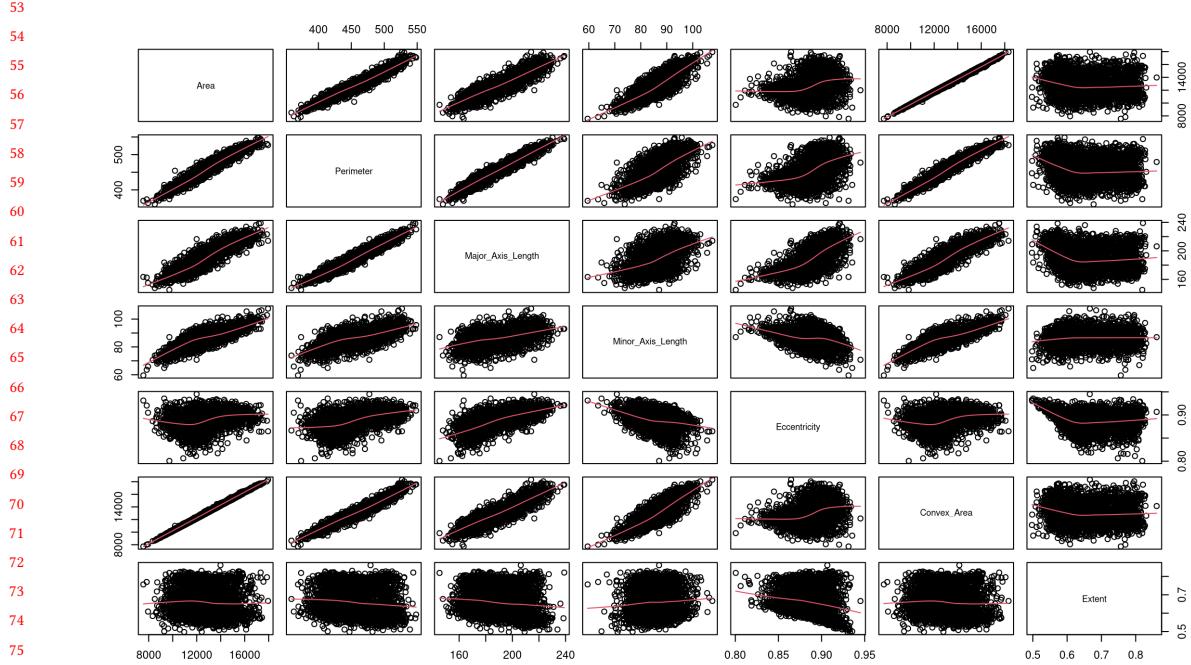


Fig. 1. Scatter Plot for the *rice\_train* dataset with LOESS smoothing lines for each class.

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.

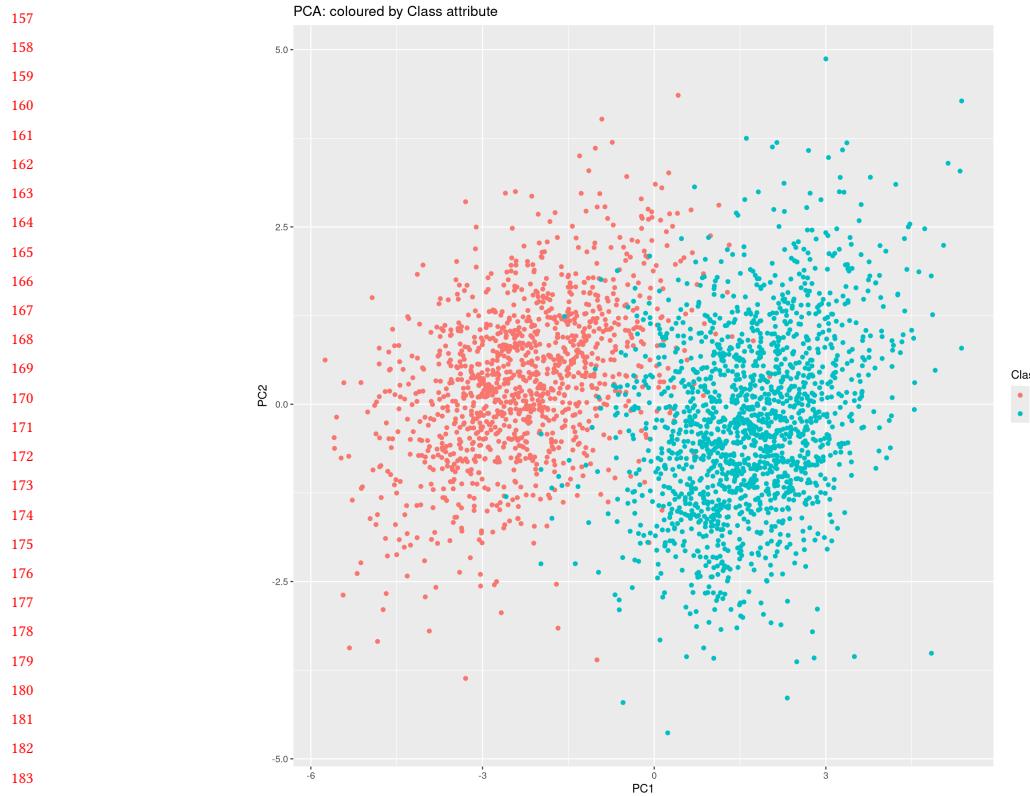


Fig. 2. Plot with first two principal components of *rice\_train* dataset.

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

```
198 rice.train$Combined <- rowMeans(rice.train[, c("Area", "Perimeter", "Major_Axis_Length", "Convex_Area")])
199
200 linear_fit = lm(
201   Class ~
202     Minor_Axis_Length
203     + Eccentricity
204     + Extent
205     + Combined,
206     data=rice_train,
207     subset = train
208   )
209
```

```

209 12
210 13 pred <- predict(linear_fit, newdata = rice_train[test, ])
211 14 mean((pred - y.test)^2)      # [1] 0.07439239
212

```

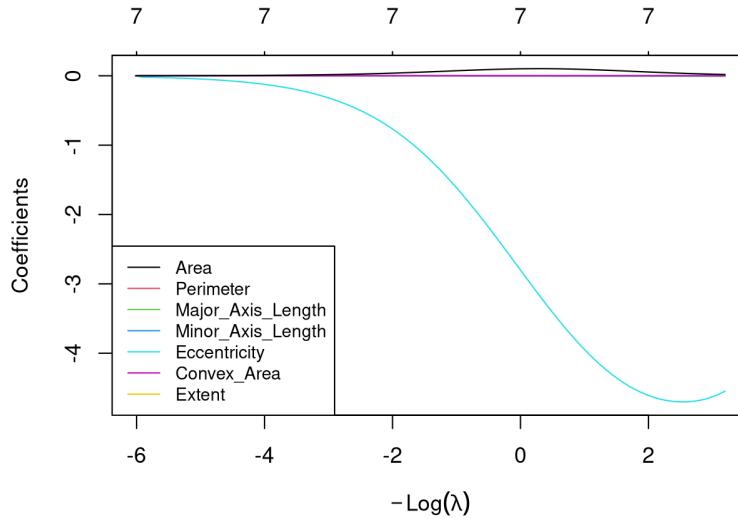
213  
214   1.2.2 *Ridge Regression.* I used the *glmnet* library, which takes care of the variable standardization for us, has a built-in  
215 cross-validation functionality to compute the best possible  $\lambda$ . With code 3 we get a result worse than linear regression,  
216 since  $MSE = 0.07924703$ . That being said, we can see in plot 3 an effective shrinkage of most coefficients towards zero,  
217 with the exception of Eccentricity, which we already recognized as not very impactful on the overall variance and  
218 prediction.  
219

Listing 3. Ridge regression on rice dataset

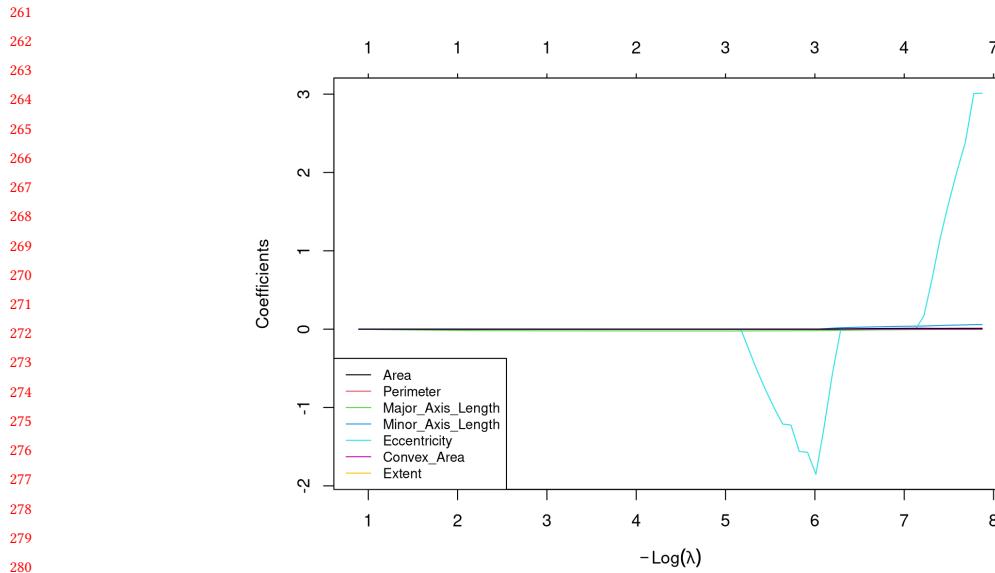
```

220
221
222 1 library(glmnet)
223 2 cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
224 3 plot(cv.out)
225 4 bestlam <- cv.out$lambda.min
226 5 cv_mse <- min(cv.out$cvm)      # [1] 0.07924703
227 6 cv_rmse <- sqrt(cv_mse)       # [1] 0.2815085
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260

```

Fig. 3. Ridge coefficients for *rice\_train* dataset.

255   1.2.3 *Lasso Regression.* Very similar to ridge regression, except *glmnet* gets called with  $\alpha = 1$ . We get a slightly  
256 better result than ridge, with  $MSE = 0.07348723$ . From the plot (in figure 4), we can see that once again Eccentricity  
257 seems to resist shrinkage, but overall coefficient reduction is much more accentuated compare to ridge, since lasso can  
258 set them to zero.  
259

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

```

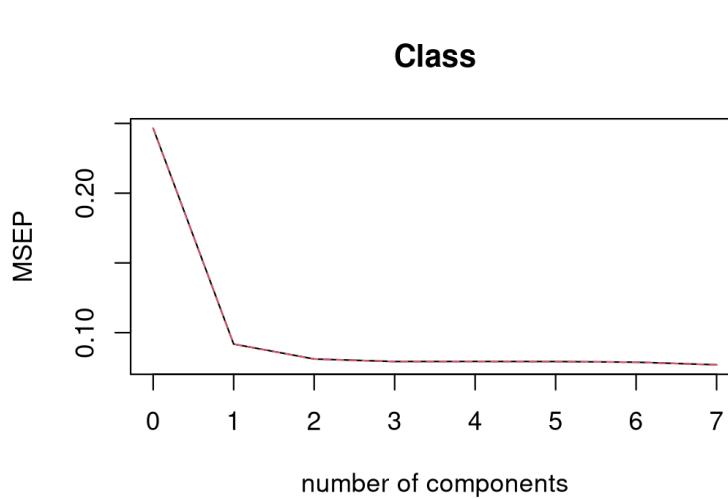
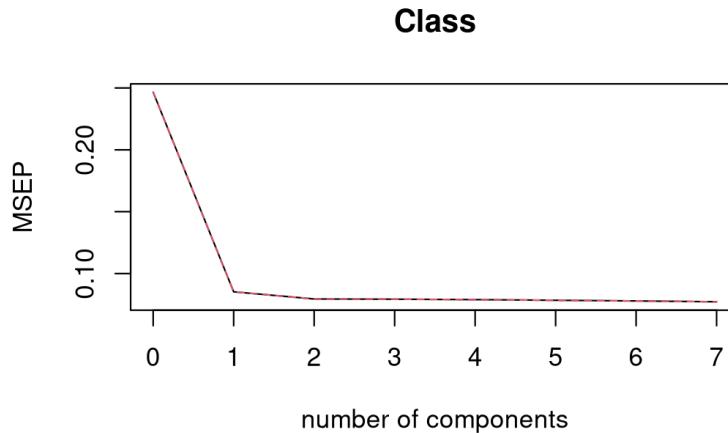
1 pcr.fit <- pcr(
2   Class ~ .,
3   data = rice_train,
4   subset = train,
5   scale = TRUE,
6   validation = "CV"
7 )

```

**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.

```
1 loess_fit1 <- loess(
2   Class ~ Area
3   + Perimeter
4   + Convex_Area
5   + Major_Axis_Length,
6   data = rice_train,
7   subset = train,
8   span = 0.1
```

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

	Span	Number of Predictors	MSE
368	0.1	Four (4)	46.0226
369	0.5	Four (4)	1.799422
370	0.9	Four (4)	0.2276324
371			
372	0.1	Two (2)	0.06099749
373	0.5	Two (2)	0.06514592
374	0.9	Two (2)	0.06836381
375			

```

377   9   )
378  10
379  11 pred1 <- predict(loess_fit1, newdata = rice_train[test, ])
380  12 mean((pred1 - y.test)^2, na.rm = TRUE) # [1] 46.0226
381

```

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: "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

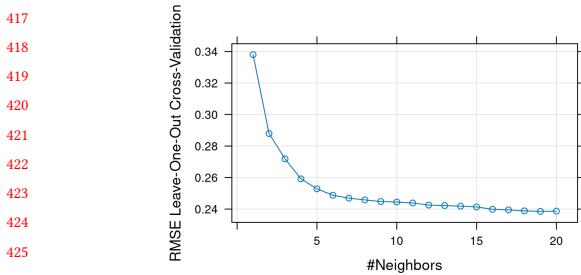
```

395 1 train.control <- trainControl(method = "LOOCV")
396 2 train$Class <- as.factor(train$Class) # necessary for classification
397
398 4 knn_fit <- train(
399 5   Class ~ .,
400 6   method      = "knn",
401 7   tuneGrid    = expand.grid(k = 1:20),
402 8   trControl   = train.control,
403 9   preProcess  = c("center", "scale"),    # normalized
404 10  metric       = "Accuracy",
405 11  data        = train
406 12 )

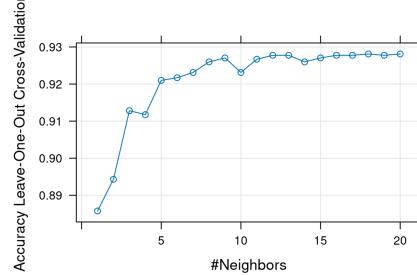
```

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 slightly worse result than KNN trained on all features: *MSE* = 0.0589397. The plot for *RMSE* can be seen in figure ??



(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.Listing 7. Random Forest regression on *rice\_train* dataset with Leave-One-Out-Cross-Validation

```
433 1 train.control <- trainControl(method = "LOOCV")
434
435 2 rf_fit <- train(
436 3   Class ~ Minor_Axis_Length + Eccentricity + Extent + Combined,
437 4   method      = "ranger",
438 5   tuneLength = 10,
439 6   trControl  = train.control,
440 7   metric      = "RMSE",
441 8   data        = rice_train,
442 9   allowParallel = TRUE
443 10 )
444
```

### 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.

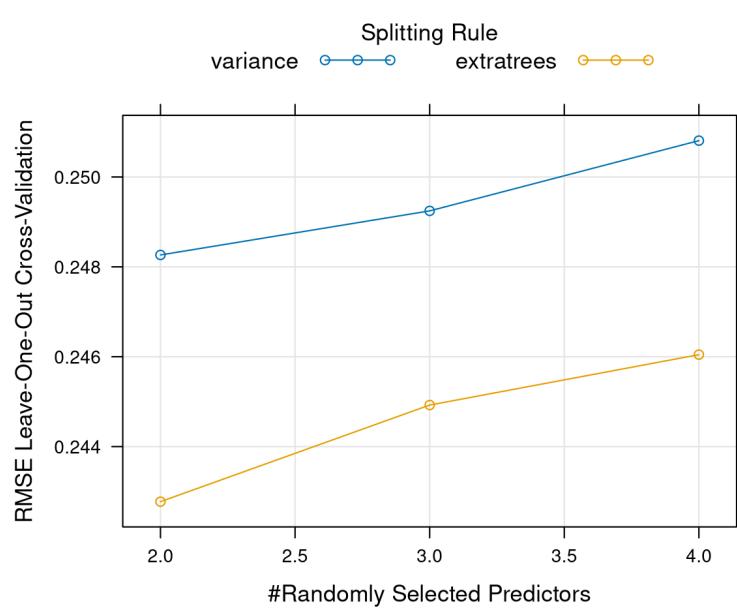


Fig. 8. RMSE error for random forest on *rice\_train* dataset.