Introduction to Mushroom Learning

Andreas Klaß, Cornelia Gruber, Felix Langer, Viktoria Szabo

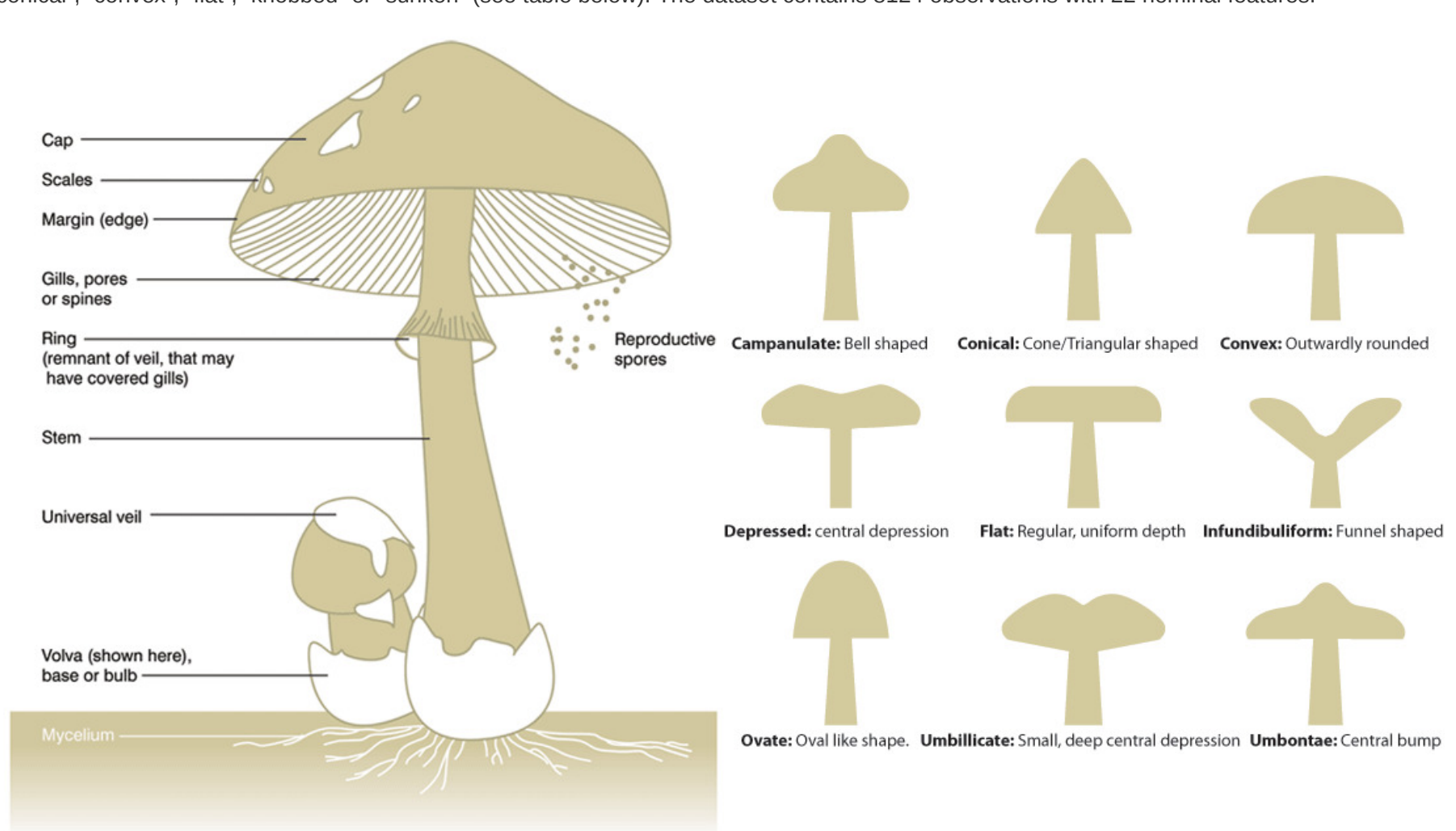
15.05.2020

source: https://blog.goodeggs.com/blog/cooking-different-types-of-mushrooms

Collecting mushrooms has gained considerable popularity - which you can see on Instagram and mushroom picking blogs. Since mushrooms come in many different colors and shapes, their edibility for humans is not necessarily obvious. Naturally, instead of asking your grandparents for advice, a statistical analysis is the way to go to find out whether a mushroom is edible or poisonous given certain attributes.

Mushroom attributes

Our main goal is to classify each mushroom into one of the two classes "edible" or "poisonous" by using machine learning methods. For this task, we can use various mushroom attributes regarding the cap, gill, stalk, odor, population or habitat. The cap shape, e.g., can be either "bell", "conical", "convex", "flat", "knobbed" or "sunken" (see table below). The dataset contains 8124 observations with 22 nominal features.



source: https://www.mushroomdiary.co.uk/mushroom-identification/

Dataset Overview

Features	Encoding
class	edible=e, poisonous=p
cap-shape	bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
cap-color	brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e ,white=w,yellow=y
bruises	bruises=t,no=f
odor	almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n ,pungent=p,spicy=s
gill-attachment	attached=a,descending=d,free=f,notched=n
gill-spacing	close=c,crowded=w,distant=d
gill-size	broad=b,narrow=n
gill-color	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p, purple=u,red=e,white=w,yellow=y
stalk-shape	enlarging=e,tapering=t
stalk-root	bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
stalk-surface-above-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-surface-below-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-color-above-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
stalk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
veil-type	partial=p,universal=u
veil-color	brown=n,orange=o,white=w,yellow=y
ring-number	none=n,one=o,two=t
ring-type	cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p, sheathing=s,zone=z
spore-print-color	black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
population	abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
habitat	grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

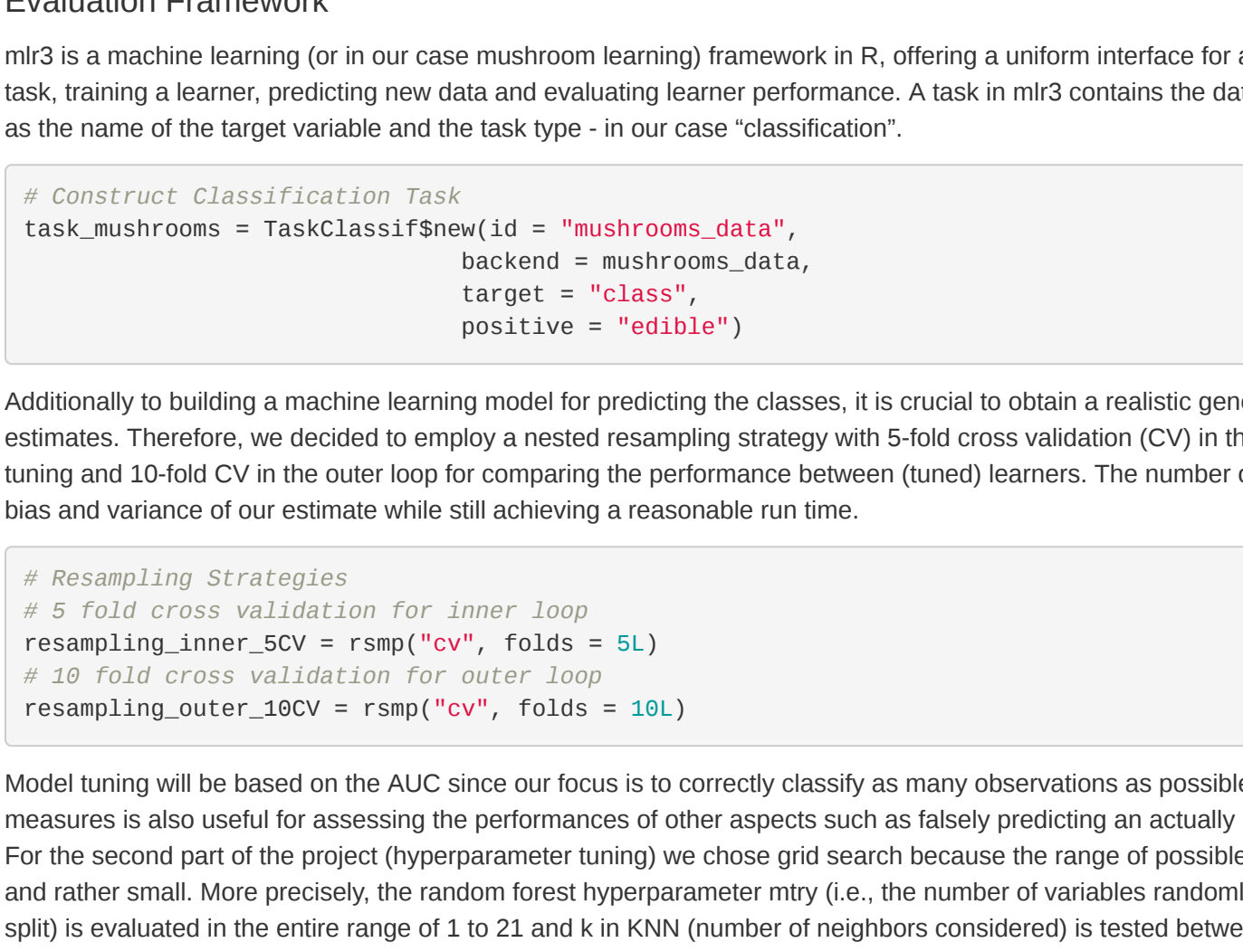
```
# Overview of data
tibble::glimpse(mushrooms_data)
```

```
## Rows: 8,124
## Columns: 22
## $ class                    <fct> poisonous, edible, edible, poisonous, edib...
## $ cap.shape                <fct> x, x, b, x, x, x, b, b, x, b, x, x, b, x, ...
## $ cap.surface              <fct> s, s, s, y, s, y, s, y, y, s, y, y, s, y, ...
## $ cap.color                <fct> n, y, w, w, g, y, w, w, w, y, y, y, y, w, ...
## $ bruises                  <fct> t, t, t, t, f, t, t, t, t, t, t, t, t, t, ...
## $ odor                     <fct> pungent, almond, anise, pungent, none, alm...
## $ gill.attachment          <fct> f, f, f, f, f, f, f, f, f, f, f, f, f, f, ...
## $ gill.spacing             <fct> c, c, c, c, w, c, c, c, c, c, c, c, c, c, ...
## $ gill.size                <fct> n, b, b, n, b, b, b, b, n, b, b, b, b, n, ...
## $ gill.color               <fct> black, black, brown, brown, black, brown, ...
## $ stalk.shape              <fct> e, e, e, e, t, e, e, e, e, e, e, e, e, e, ...
## $ stalk.root               <fct> e, c, c, e, e, c, c, c, e, c, c, c, c, e, ...
## $ stalk.surface.above.ring <fct> s, s, s, s, s, s, s, s, s, s, s, s, s, s, ...
## $ stalk.surface.below.ring <fct> s, s, s, s, s, s, s, s, s, s, s, s, s, s, ...
## $ stalk.color.above.ring   <fct> w, w, w, w, w, w, w, w, w, w, w, w, w, w, ...
## $ stalk.color.below.ring   <fct> w, w, w, w, w, w, w, w, w, w, w, w, w, w, ...
## $ veil.color               <fct> w, w, w, w, w, w, w, w, w, w, w, w, w, w, ...
## $ ring.number              <fct> o, o, o, o, o, o, o, o, o, o, o, o, o, o, ...
## $ ring.type                <fct> p, p, p, p, e, p, p, p, p, p, p, p, p, p, ...
## $ spore.print.color        <fct> k, n, n, k, n, k, k, n, k, k, n, k, n, n, ...
## $ population               <fct> s, n, n, s, a, n, n, s, v, s, n, s, s, v, ...
## $ habitat                  <fct> u, g, m, u, g, g, m, m, g, m, g, m, g, u, ...
```

```
summary(mushrooms_data)
```

```
##        class      cap.shape cap.surface   cap.color    bruises       odor
##  edible   :4208   b: 452    f:2320      n      :2284   f:4748   none   :3528
##  poisonous:3916   c:   4    g:   4      g      :1840   t:3376   foul   :2160
##                   f:3152    s:2556      e      :1500            spicy  : 576
##                   k: 828    y:3244      y      :1072            fishy  : 576
##                   s:  32                w      :1040            almond : 400
##                   x:3656                b      : 168            anise  : 400
##                                         (Other): 220            (Other): 484
##  gill.attachment gill.spacing gill.size     gill.color   stalk.shape stalk.root
##  a: 210          c:6812       b:5612    buff     :1728   e:3516      ?:2480
##  f:7914          w:1312       n:2512    pink     :1492   t:4608      b:3776
##                                         white    :1202               c: 556
##                                         brown    :1048               e:1120
##                                         gray     : 752               r: 192
##                                         chocolate: 732
##                                         (Other)  :1170
##  stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
##  f: 552                   f: 600                   w      :4464
##  k:2372                   k:2304                   p      :1872
##  s:5176                   s:4936                   g      : 576
##  y:  24                   y: 284                   n      : 448
##                                                    b      : 432
##                                                    o      : 192
##                                                    (Other): 140
##  stalk.color.below.ring veil.color ring.number ring.type spore.print.color
##  w      :4384           n:  96     n:  36      e:2776    w      :2388
##  p      :1872           o:  96     o:7488      f:  48    n      :1968
##  g      : 576           w:7924     t: 600      l:1296    k      :1872
##  n      : 512           y:   8                 n:  36    h      :1632
##  b      : 432                                  p:3968    r      :  72
##  o      : 192                                            b      :  48
##  (Other): 156                                            (Other): 144
##  population habitat
##  a: 384     d:3148
##  c: 340     g:2148
##  n: 400     l: 832
##  s:1248     m: 292
##  v:4040     p:1144
##  y:1712     u: 368
##             w: 192
```

In order to get a better feeling for our mushrooms, let us do some plotting. As can be seen in the bar plot below, odor seems to be quite a good indicator whether a mushroom is poisonous or edible. According to this simple descriptive visualization, pungent, spicy or fishy odors clearly identify poisonous mushrooms.



Mushroom Learning

Machine learning analyses focus in particular on two aspects: First, on comparing performances of different machine learning methods such as k-nearest neighbors (KNN) or Random Forests on a specific dataset and second, tuning hyperparameters for some of the used methods.

Evaluation Framework

mlr3 is a machine learning (or in our case mushroom learning) framework in R, offering a uniform interface for all necessary steps like defining a task, training a learner, predicting new data and evaluating learner performance. A task in mlr3 contains the data as well as meta information, such as the name of the target variable and the task type - in our case "classification".

```
# Construct Classification Task
task_mushrooms = TaskClassif$new(id = "mushrooms_data",
                               backend = mushrooms_data,
                               target = "class",
                               positive = "edible")
```

Additionally to building a machine learning model for predicting the classes, it is crucial to obtain a realistic generalization error (GE) for our estimates. Therefore, we decided to employ a nested resampling strategy with 5-fold cross validation (CV) in the inner loop for hyperparameter tuning and 10-fold CV in the outer loop for comparing the performance between (tuned) learners. The number of folds was chosen to balance the bias and variance of our estimate while still achieving a reasonable run time.

```
# Resampling Strategies
# 5 fold cross validation for inner loop
resampling_inner_5CV = rsmp("cv", folds = 5L)
# 10 fold cross validation for outer loop
resampling_outer_10CV = rsmp("cv", folds = 10L)
```

Model tuning will be based on the AUC since our focus is to correctly classify as many observations as possible. Nevertheless, printing other measures is also useful for assessing the performances of other aspects such as falsely predicting an actually poisonous mushroom as edible. For the second part of the project (hyperparameter tuning) we chose grid search because the range of possible hyperparameter values is discrete and rather small. More precisely, the random forest hyperparameter mtry (i.e., the number of variables randomly sampled as candidates at each split) is evaluated in the entire range of 1 to 21 and k in KNN (number of neighbors considered) is tested between 1 and 50.

```
# Performance Measures
measures = list(
  msr("classif.auc",
      id = "AUC"),
  msr("classif.fpr",
      id = "False Positive Rate"), # false positive rate especially interesting
  # for our falsely edible (although actually poisonous) classification
  msr("classif.sensitivity",
      id = "Sensitivity"),
  msr("classif.specificity",
      id = "Specificity"),
  msr("classif.ce",
      id = "MMCE")
)

# Choose optimization algorithm:
tuner_grid_search_knn = tnr("grid_search", resolution = 50)
tuner_grid_search_mtry = tnr("grid_search", resolution = 21)

# evaluate performance on AUC:
measures_tuning = msr("classif.auc", id = "AUC")
# Set when to terminate:
terminator_knn = term("evals", n_evals = 50)
# since almost all mtry values lead to very good results we evaluate 1 to 21 features as opposed to a termination
criterion like stagnation
terminator_mtry = term("evals", n_evals = 21)
```

Choosing Algorithms

A relevant property of our dataset is that all features are nominal and thus multinomially distributed. Therefore, linear and quadratic discriminant analyses (LDA and QDA) are not possible due to the violated assumption of normally distributed features. We will compare featureless, naive bayes, KNN, logistic regression, decision tree and random forest models since they are all suitable for binary classification with nominal features.

In the following part, we define our inner loop of the nested resampling process in order to tune the chosen hyperparameters. Only the random forest (mtry) and KNN (k) have hyperparameters which will be tuned in a 5-fold-cv.

```
# Autotune knn -----------------------------------------------------------------
# Define learner:
learner_knn = lrn("classif.kknn", predict_type = "prob")

# tune the chosen hyperparameter k with these boundaries:
param_k = ParamSet$new(
  list(
    ParamInt$new("k", lower = 1L, upper = 50)
  )
)

# Set up autotuner instance with the predefined setups
tuner_knn = AutoTuner$new(
  learner = learner_knn,
  resampling = resampling_inner_5CV,
  measures = measures_tuning,
  tune_ps = param_k,
  terminator = terminator_knn,
  tuner = tuner_grid_search_knn
)


# Autotune Random Forest --------------------------------------------------------
# Define learner:
learner_ranger = lrn("classif.ranger", predict_type = "prob", importance = "impurity")


# we will try all configurations: 1 to 21 features.
param_mtry = ParamSet$new(
  list(
    ParamInt$new("mtry", lower = 1L, upper = 21L)
  )
)

# Set up autotuner instance with the predefined setups
tuner_ranger = AutoTuner$new(
  learner = learner_ranger,
  resampling = resampling_inner_5CV,
  measures = measures_tuning,
  tune_ps = param_mtry,
  terminator = terminator_mtry,
  tuner = tuner_grid_search_mtry
)


(learners = list(lrn("classif.featureless", predict_type = "prob"),
                lrn("classif.naive_bayes", predict_type = "prob"),
                lrn("classif.rpart", predict_type = "prob"),
                lrn("classif.log_reg", predict_type = "prob"),
                tuner_ranger,
                tuner_knn))
```

```
## [[1]]
## <LearnerClassifFeatureless:classif.featureless>
## * Model: -
## * Parameters: method=mode
## * Packages: -
## * Predict Type: prob
## * Feature types: logical, integer, numeric, character, factor, ordered
## * Properties: importance, missings, multiclass, selected_features,
##   twoclass
##
## [[2]]
## <LearnerClassifNaiveBayes:classif.naive_bayes>
## * Model: -
## * Parameters: list()
## * Packages: e1071
## * Predict Type: prob
## * Feature types: logical, integer, numeric, factor
## * Properties: multiclass, twoclass
##
## [[3]]
## <LearnerClassifRpart:classif.rpart>
## * Model: -
## * Parameters: xval=0
## * Packages: rpart
## * Predict Type: prob
## * Feature types: logical, integer, numeric, factor, ordered
## * Properties: importance, missings, multiclass, selected_features,
##   twoclass, weights
##
## [[4]]
## <LearnerClassifLogReg:classif.log_reg>
## * Model: -
## * Parameters: list()
## * Packages: stats
## * Predict Type: prob
## * Feature types: logical, integer, numeric, character, factor, ordered
## * Properties: twoclass, weights
##
## [[5]]
## <AutoTuner:classif.ranger.tuned>
## * Model: -
## * Parameters: importance=impurity
## * Packages: ranger
## * Predict Type: prob
## * Feature types: logical, integer, numeric, character, factor, ordered
## * Properties: importance, multiclass, oob_error, twoclass, weights
##
## [[6]]
## <AutoTuner:classif.kknn.tuned>
## * Model: -
## * Parameters: list()
## * Packages: kknn
## * Predict Type: prob
## * Feature types: logical, integer, numeric, factor, ordered
## * Properties: multiclass, twoclass
```

Benchmark Results

The final benchmark of all 6 methods, which is the outer loop of our nested resampling with a 10-fold-cv, is shown below. In this step the GE as well as other performance measures of the algorithms are compared.

```
design = benchmark_grid(
  tasks = task_mushrooms,
  learners = learners,
  resamplings = resampling_outer_10CV
)
# lower the logger threshold to suppress info messages:
lgr::get_logger("mlr3")$set_threshold("warn")
bmr = benchmark(design, store_models = TRUE)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
# reset console messages to default
lgr::get_logger("mlr3")$set_threshold("info")
```

When nominal features are included in a logistic regression each category is binary encoded. However, if in one category every single observation points to the same target variable class (edible or poisonous), the model does not converge. In this case the separation leads to the non-existence of - or rather infinitely high - maximum likelihood estimates, since the probability for an observation to belong to a specific class is exactly 1. As this is what happens for some categories such as "spicy odor" the model fit is not quite stable and interpreting the coefficients might be misleading.

Apart from this, no errors or warnings occurred during resampling when printing the benchmark result.

```
print(bmr)
```

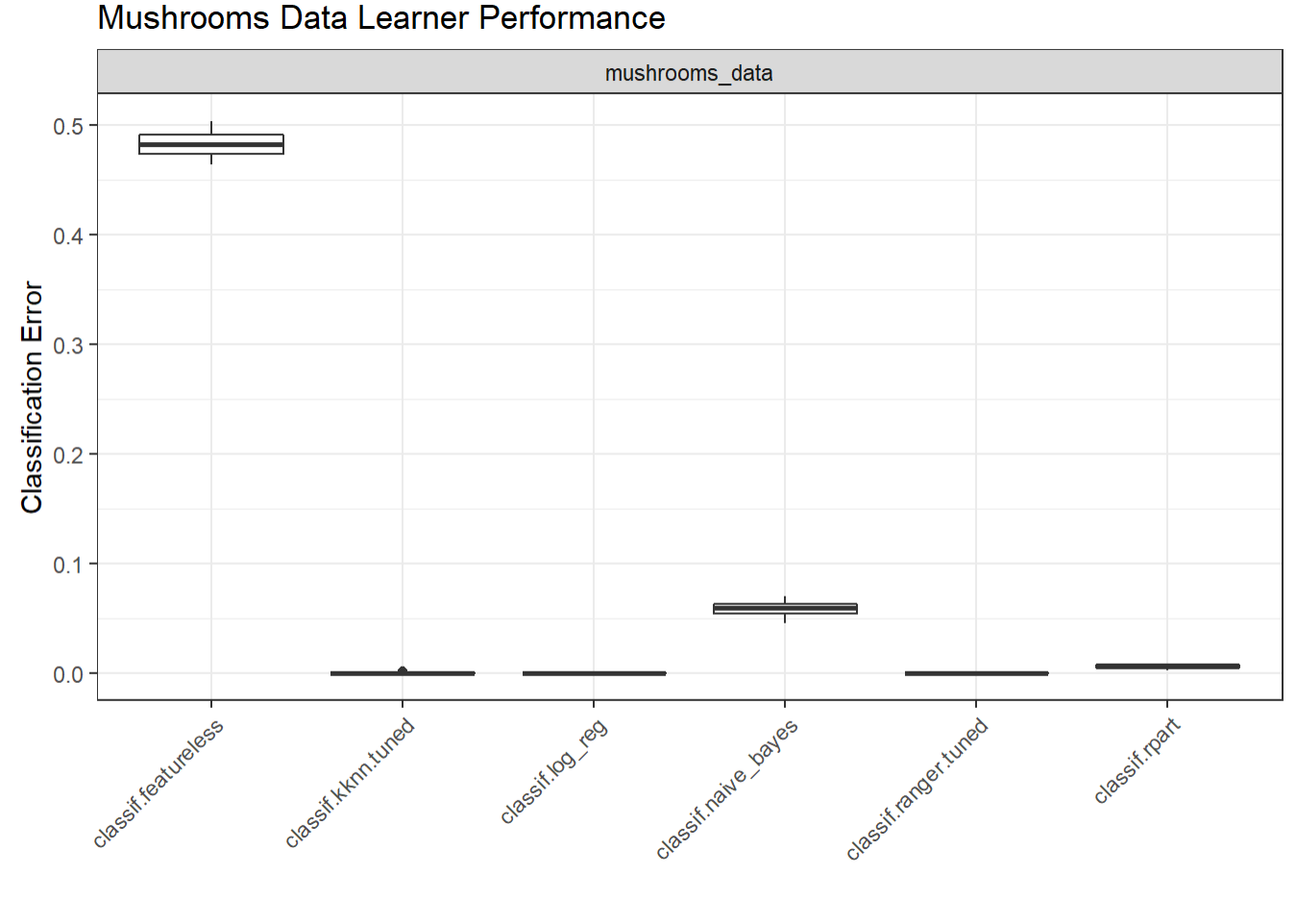
```
## <BenchmarkResult> of 60 rows with 6 resampling runs
##  nr        task_id           learner_id resampling_id iters warnings errors
##   1 mushrooms_data  classif.featureless            cv    10        0      0
##   2 mushrooms_data  classif.naive_bayes            cv    10        0      0
##   3 mushrooms_data        classif.rpart            cv    10        0      0
##   4 mushrooms_data      classif.log_reg            cv    10        0      0
##   5 mushrooms_data classif.ranger.tuned            cv    10        0      0
##   6 mushrooms_data   classif.kknn.tuned            cv    10        0      0
```

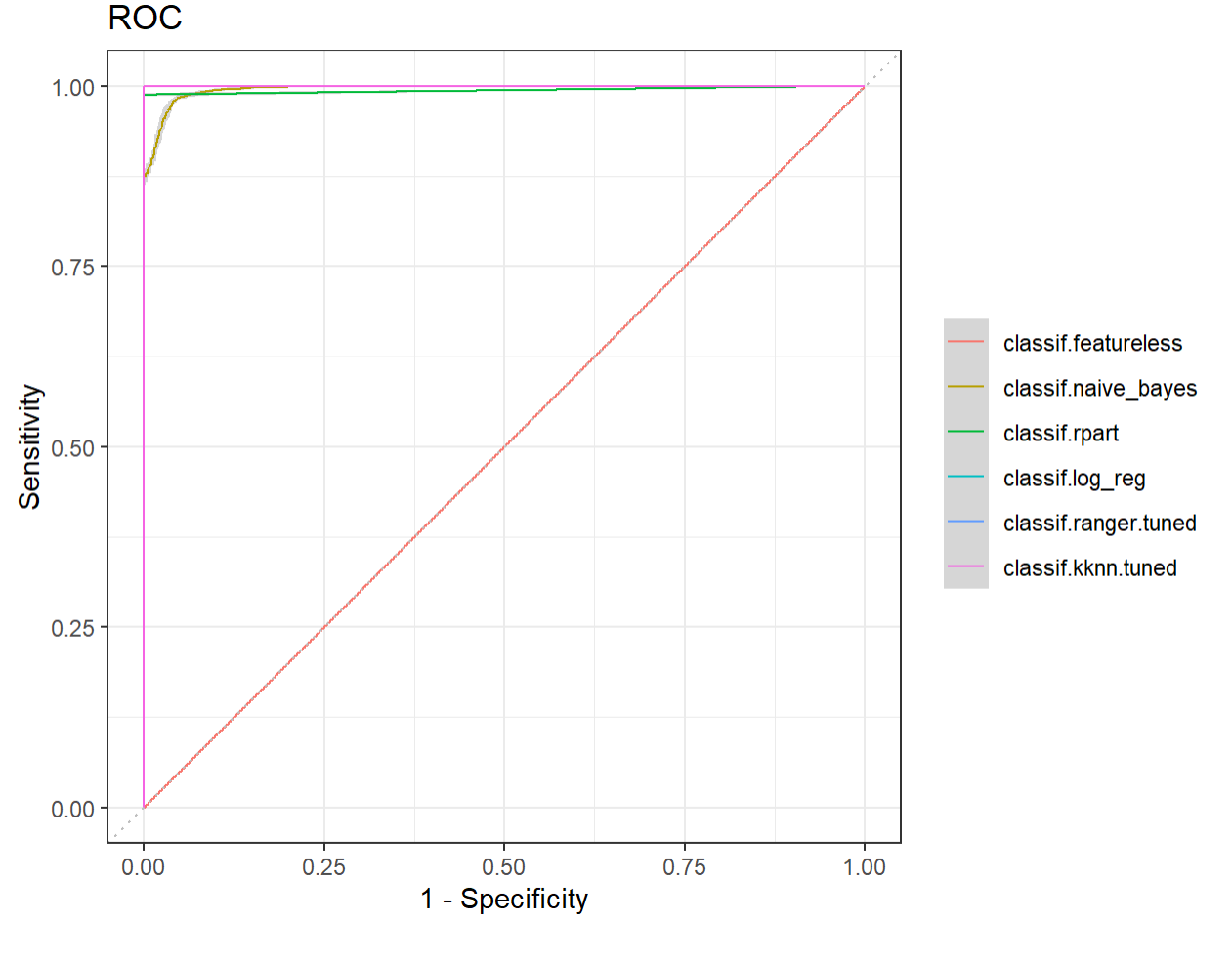
In the following table we can see how every algorithm (except for the featureless baseline) leads to nearly perfect classification results and thus to an AUC to 1 or almost 1. As expected, a featureless model stays at a classification error (CE) of close to 0.5 (see mean misclassification error MMCE) since the most common class label will be predicted for every observation regardless of features and our classes are almost equally distributed (51.8% edible vs. 48.2% poisonous). The logistic regression (log_reg), random forest (ranger) and KNN (kknn) all have an AUC of 1.0000000. However, if you look at the False Positive Rate (FPR) of KNN, it is not exactly zero (due to a threshold of 0.5). Even a single, usually rather unstable, decision tree (rpart) leads to incredible results and a FPR of only 1.2%. Since there are several features which can separate the classes very well, these results should not be too surprising, though. Therefore, we conclude that there is no necessity to further tune hyperparameters.

```
tab_learner_performance = bmr$aggregate(measures)
# tab_learner_performance[, learner_id:MMCE]
tab_learner_performance %>% select(learner_id:MMCE)
```

```
##              learner_id resampling_id iters       AUC False Positive Rate
## 1:  classif.featureless            cv    10 0.5000000        1.0000000000
## 2:  classif.naive_bayes            cv    10 0.9960508        0.1156206472
## 3:        classif.rpart            cv    10 0.9938725        0.0122550286
## 4:      classif.log_reg            cv    10 1.0000000        0.0000000000
## 5: classif.ranger.tuned            cv    10 1.0000000        0.0000000000
## 6:   classif.kknn.tuned            cv    10 1.0000000        0.0002564103
##    Sensitivity Specificity         MMCE
## 1:    1.000000   0.0000000 0.4820289447
## 2:    0.993594   0.8843794 0.0589604578
## 3:    1.000000   0.9877450 0.0059081490
## 4:    1.000000   1.0000000 0.0000000000
## 5:    1.000000   1.0000000 0.0000000000
## 6:    1.000000   0.9997436 0.0001231527
```

In the first plot we can see how the CE is distributed among the different CV folds for the various algorithms and in the second plot the ROC curve is displayed.





Final model

All steps so far answered the question which model with which hyperparameters yield good results with a now known GE. However, in order to obtain the best possible predictions, training with all available data points is needed. Our previous models showed that:

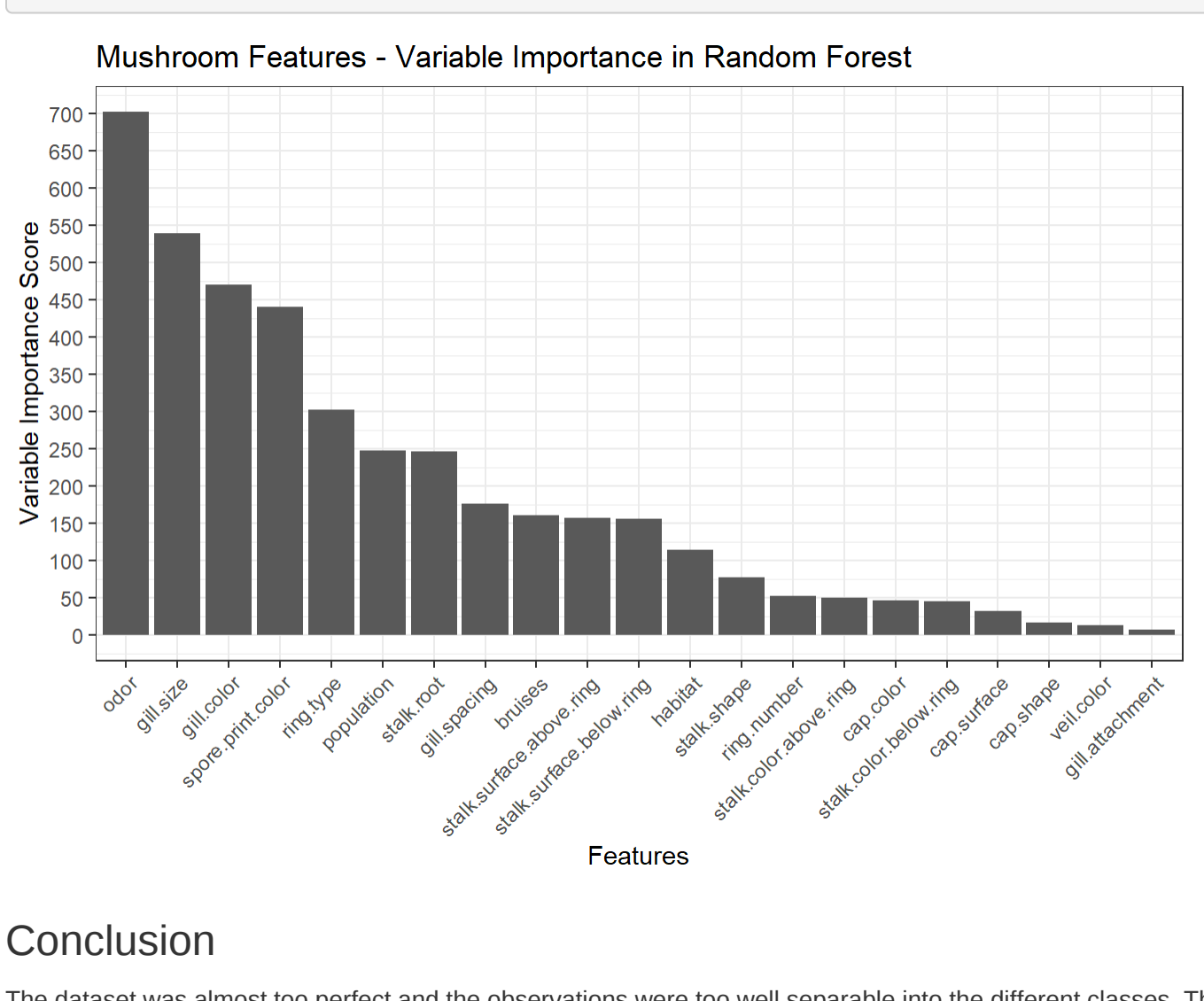
- Naive Bayes has a worse FPR than other models, which is relevant considering a mushroom which is falsely classified as edible might lead to poisoning or even death
- Perfect classification results for random forest and logistic regression
- Logistic regression has convergence issues
- Almost perfect classification for KNN and decision tree

Even though a single tree would be the most useful and applicable model when picking mushrooms in real life, we chose the random forest as final model since it can perfectly classify all observations and has no stability issues such as the logistic regression.

```
# Train tuner_ranger once again using the same specs as before
tuner_ranger$tuning_instance
# show only warnings:
lgr::get_logger("mlr3")$set_threshold("warn")
tuner_ranger$train(task_mushrooms)
# reset console outputs to default:
lgr::get_logger("mlr3")$set_threshold("info")

# AUC performance for parameter combinations:
tuner_ranger$tuning_instance$archive(unnest = "params") %>%
  select(mtry, AUC) %>%
  arrange(mtry)
# Pretty much every combination works perfectly

tuner_ranger$tuning_result # Winning mtrz is 11 although we could use
# 2-21 and achieve the same perfect performance

# use these parameters for our final winner model with winner specs:
learner_final = lrn("classif.ranger",predict_type = "prob")
learner_final$param_set$values = tuner_ranger$tuning_result$params

# Fit winner model to entire data set
learner_final$train(task_mushrooms)
```

To prevent the random forest from being too much of a black box algorithm, the variable importance is shown in the plot below. As suspected in the descriptive part, odor is clearly a good indicator for mushroom edibility.

```
# construct filter to extract variable importance in previously set up winning learner
# Gini index for classification used in 'impurity' measure
filter_ranger = flt("importance", learner = learner_final)
# rerun learner on entire data set and store variable importance results
filter_ranger$calculate(task_mushrooms)

feature_scores <- as.data.table(filter_ranger)

ggplot(data = feature_scores,
       aes(x = reorder(feature, -score),
           y = score)) +
  geom_bar(stat = "identity") +
  ggtitle(label = "Mushroom Features - Variable Importance in Random Forest") +
  labs(x = "Features", y = "Variable Importance Score") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+
  scale_y_continuous(breaks = pretty(1:feature_scores$score[1],10))
```



Conclusion

The dataset was almost too perfect and the observations were too well separable into the different classes. This made machine learning methods and especially hyperparameter tuning almost unnecessary. Nevertheless, it was very interesting to look at this uncommon case of "perfect" data and it showed that you should critically question every analysis and not just blindly follow the standard machine learning procedure. In borderline cases such as ours, a careful examination of the outputs are especially important. The non-convergence of the logistic regression estimates illustrates this perfectly.