

Data Science: Capstone CYO Project - Mushroom

Elvin Tam

25 June 2021



CREDIT: GETTY IMAGES

Introduction

In this report, our goal is to predict the edibility (class: edible / poisonous) of mushroom basing on attribution information. Data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). The reason of selecting this dataset is that this problem is related to classification which is a large part of application in data science. And, it is also a complement to project – MovieLens that we can cover each part of what we have learnt from the course.

The mushroom dataset has already been well formatted. Process of data cleaning is only removing 2 attributes prior to splitting the data to training set and test set. 10 algorithms are applied and an ensemble model combining the prior 10 different algorithms to see if it can provide improvement to our predictions.

1. Data Cleaning

Mushroom data set contains 23 columns of 1 class and 22 attributes related to cap, bruises, odor, gill, stalk, veil, ring, spore color, population and habitat of 8,124 observations.

```
## 'data.frame':    8124 obs. of  23 variables:
## $ class          : Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...
## $ cap_shape      : Factor w/ 6 levels "b","c","f","k",...: 6 6 1 6 6 6 1 1 6 1 ...
## $ cap_surface    : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...
## $ cap_color      : Factor w/ 10 levels "b","c","e","g",...: 5 10 9 9 4 10 9 9 9 10 ...
## $ bruises        : Factor w/ 2 levels "f","t": 2 2 2 2 1 2 2 2 2 2 ...
## $ odor           : Factor w/ 9 levels "a","c","f","l",...: 7 1 4 7 6 1 1 4 7 1 ...
## $ gill_attachment : Factor w/ 2 levels "a","f": 2 2 2 2 2 2 2 2 2 2 ...
```

```
## $ gill_spacing      : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...
## $ gill_size         : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...
## $ gill_color        : Factor w/ 12 levels "b","e","g","h",...: 5 5 6 6 5 6 3 6 8 3 ...
## $ stalk_shape       : Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...
## $ stalk_root        : Factor w/ 5 levels "?","b","c","e",...: 4 3 3 4 4 3 3 3 4 3 ...
## $ stalk_surface_above_ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
## $ stalk_surface_below_ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
## $ stalk_color_above_ring : Factor w/ 9 levels "b","c","e","g",...: 8 8 8 8 8 8 8 8 8 8 ...
## $ stalk_color_below_ring : Factor w/ 9 levels "b","c","e","g",...: 8 8 8 8 8 8 8 8 8 8 ...
## $ veil_type         : Factor w/ 1 level "p": 1 1 1 1 1 1 1 1 1 1 ...
## $ veil_color        : Factor w/ 4 levels "n","o","w","y": 3 3 3 3 3 3 3 3 3 3 ...
## $ ring_number       : Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...
## $ ring_type         : Factor w/ 5 levels "e","f","l","n",...: 5 5 5 5 1 5 5 5 5 5 ...
## $ spore_print_color  : Factor w/ 9 levels "b","h","k","n",...: 3 4 4 3 4 3 3 4 3 3 ...
## $ population        : Factor w/ 6 levels "a","c","n","s",...: 4 3 3 4 1 3 3 4 5 4 ...
## $ habitat           : Factor w/ 7 levels "d","g","l","m",...: 6 2 4 6 2 2 4 4 2 4 ...
```

```
## class      cap_shape cap_surface  cap_color  bruises      odor
## e:4208      b: 452    f:2320      n          :2284    f:4748    n          :3528
## p:3916      c:   4    g:   4      g          :1840    t:3376    f          :2160
##           f:3152    s:2556      e          :1500          s          : 576
##           k: 828    y:3244      y          :1072          y          : 576
##           s:  32          w          :1040          a          : 400
##           x:3656          b          : 168          l          : 400
##                                     (Other): 220      (Other): 484
## gill_attachment gill_spacing gill_size  gill_color  stalk_shape stalk_root
## a: 210          c:6812    b:5612    b          :1728    e:3516    ?:2480
## f:7914          w:1312    n:2512    p          :1492    t:4608    b:3776
##                                     w          :1202          c: 556
##                                     n          :1048          e:1120
##                                     g          : 752          r: 192
##                                     h          : 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_type veil_color ring_number ring_type
## w          :4384          p:8124    n: 96      n: 36      e:2776
## p          :1872          o: 96      o:7488    f:  48
## g          : 576          w:7924    t: 600    l:1296
## n          : 512          y:  8      n: 36
## b          : 432          p:3968
## o          : 192
## (Other): 156
## spore_print_color population habitat
## w          :2388          a: 384      d:3148
## n          :1968          c: 340      g:2148
## k          :1872          n: 400      l: 832
## h          :1632          s:1248      m: 292
```

```
## r      : 72      v:4040      p:1144
## b      : 48      y:1712      u: 368
## (Other): 144      w: 192
```

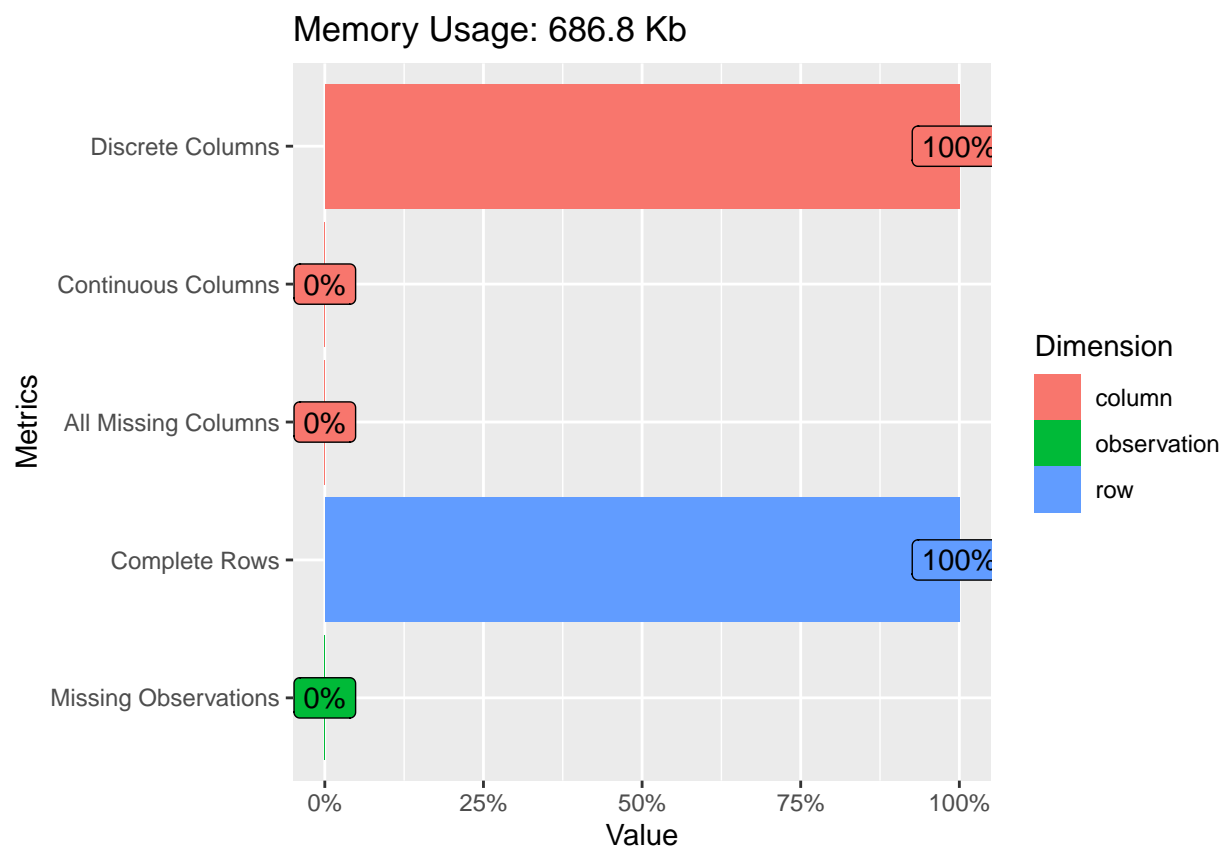
According to description from the source, there is data missing in the attribute of stalk_root. The missing data point is marked “?” from the source already. On the other hand, veil_type is a constant. Both stalk_root and veil_type are removed before we start data exploration & modeling.

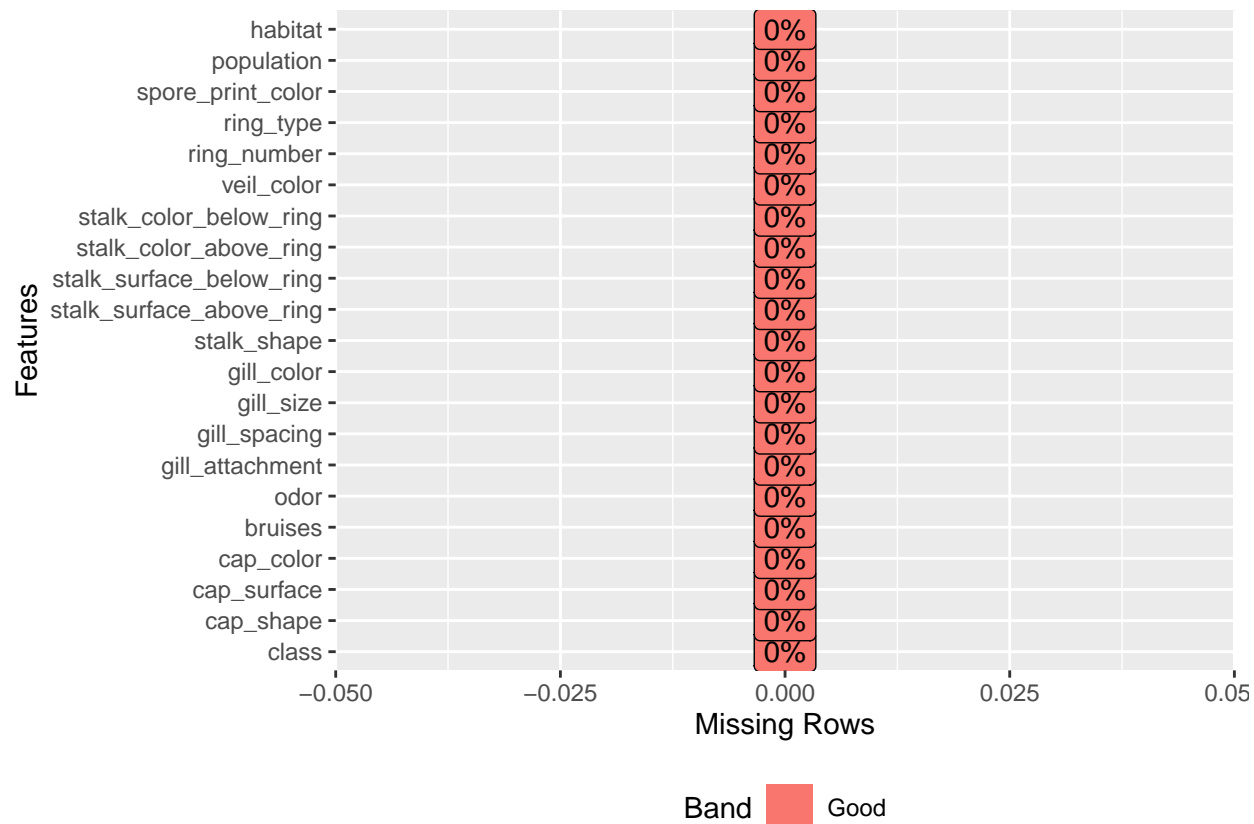
```
mushroom <- mushroom %>% select(-veil_type, -stalk_root)
```

2. Data Exploration

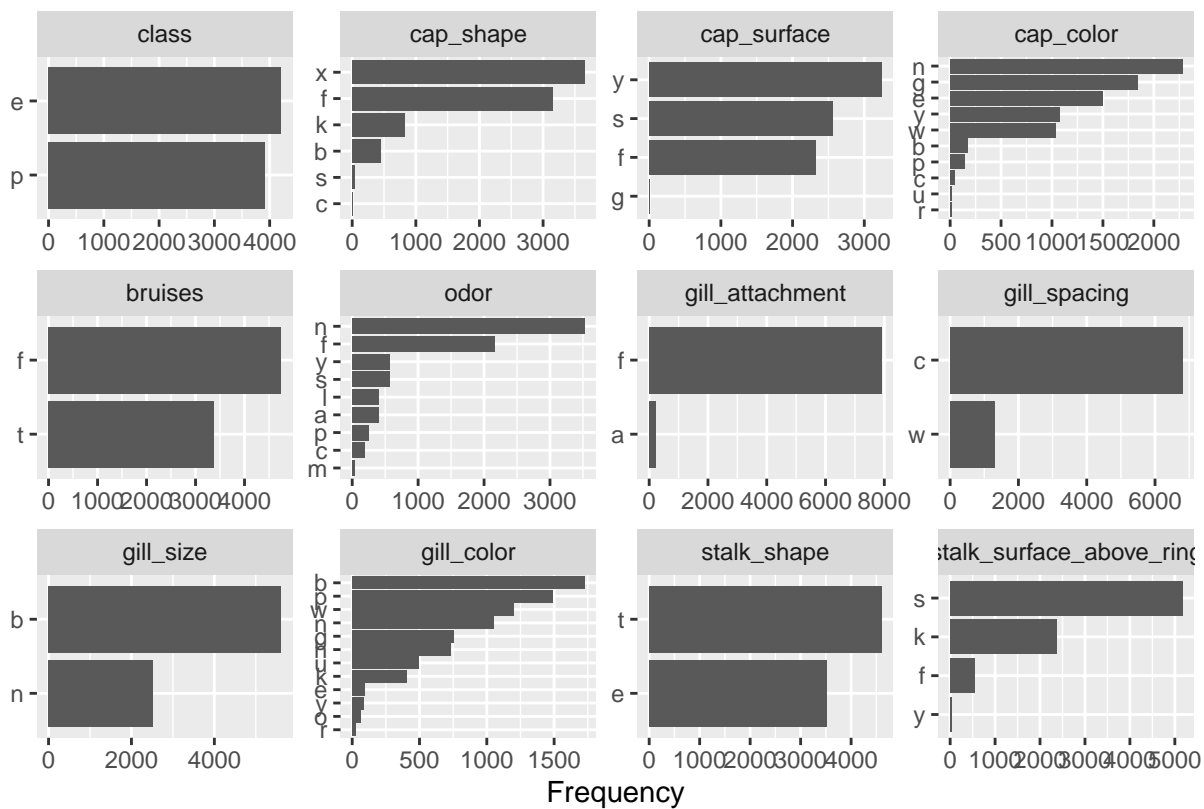
We will use Data Explorer package in the process of data exploration. By using this package, it provides a standardized method to get the insights from the dataset.

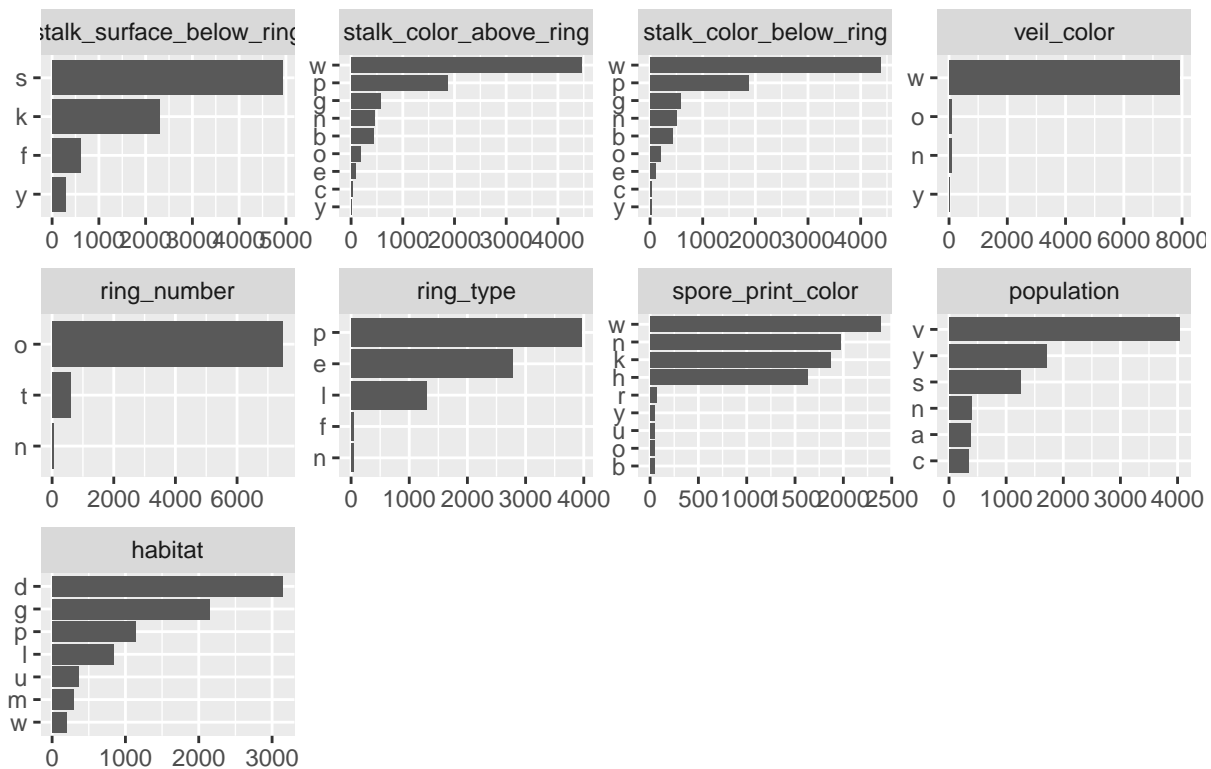
From below 2 charts, we can see that the mushroom data is discrete with no data missing.



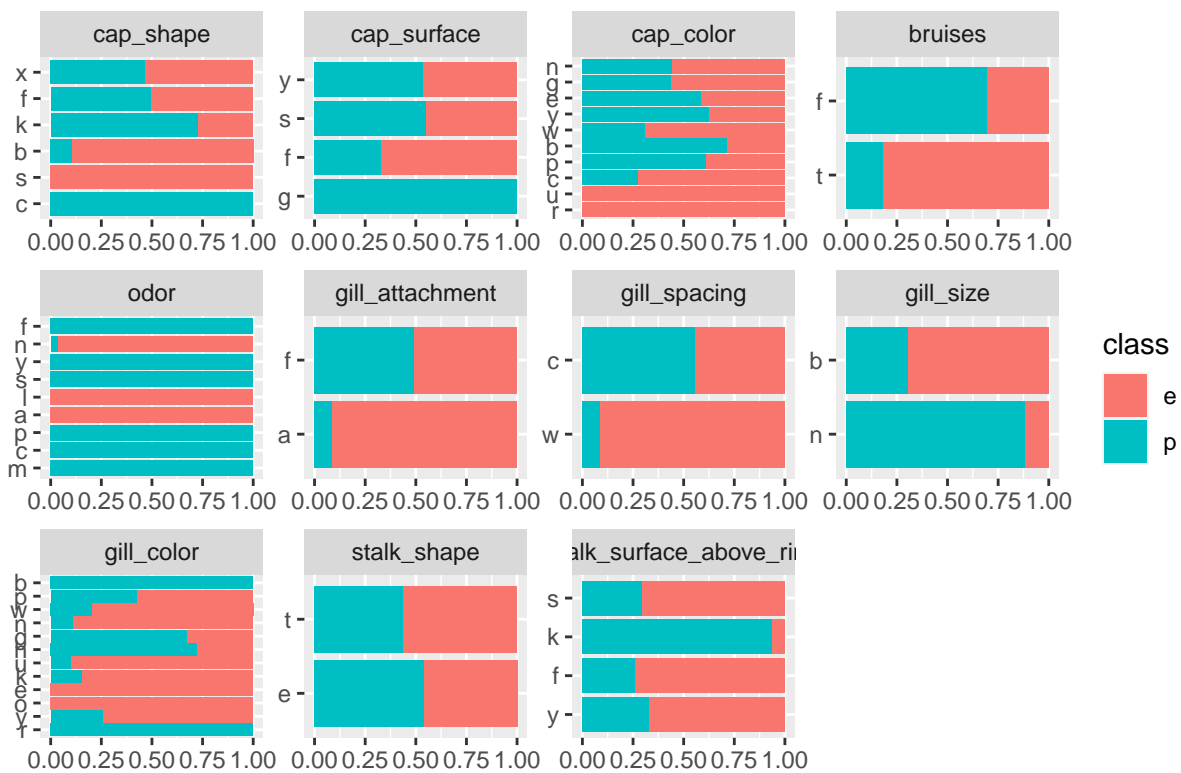


From below frequency and percentage charts, we can see how many observations belong to each category in each attribute and among those how many are edible or poisonous. In class which we are going to predict, we can say the feature is roughly equal distributed. However, observations are mainly clustered in one category in gill_attachment, gill_spacing, veil_color and ring_number.





Frequency

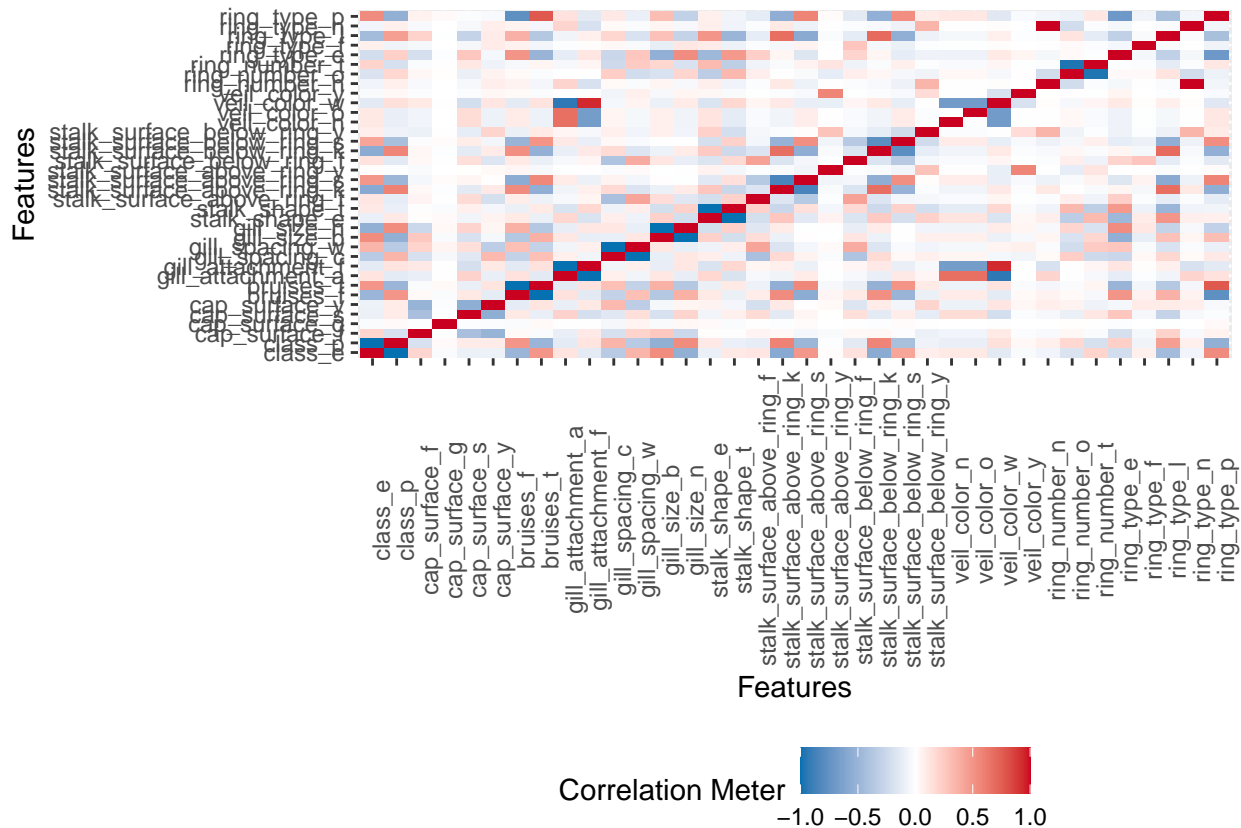




Page 2

From correlation matrix (filtered category less than 5), there is high correlation between veil_color and gill_attachment, stalk_surface_above_ring and bruises.

```
## 9 features with more than 5 categories ignored!
## cap_shape: 6 categories
## cap_color: 10 categories
## odor: 9 categories
## gill_color: 12 categories
## stalk_color_above_ring: 9 categories
## stalk_color_below_ring: 9 categories
## spore_print_color: 9 categories
## population: 6 categories
## habitat: 7 categories
```

3. Modeling Approach

We will use 10 algorithms and 1 ensemble model to evaluate if this can provide improvement to our predictions. Algorithms are listed below.

3-1. GLM 3-2. LDA 3-3. Naïve Bayes 3-4. svmLinear 3-5. KNN 3-6. gamLoess 3-7. Multinom 3-8. Classification Model 3-9. Random Forest 3-10. Adaboost 3-11. Ensemble

3-1. GLM

Generalized Linear Model (GLM) is the most common and general model. This is the starting point of our modeling. Using Caret package, we can simply apply 10 different algorithms in a standardized way.

When we run the GLM algorithms, we get warning message of “Warning: glm.fit: algorithm did not converge”, “Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred” and “prediction from a rank-deficient fit may be misleading”.

```
start_time <- Sys.time()
fit_glm <- train(class ~ ., method = "glm", data = train_set)
time_diff <- Sys.time() - start_time

s <- summary(fit_glm)

s
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.141e-05 -2.364e-06 -8.477e-07  2.286e-06  2.359e-05
##
## Coefficients: (8 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -8.071e+01  2.901e+05  0.000   1.000
## cap_shapec      2.017e+00  1.463e+05  0.000   1.000
## cap_shapef     -7.377e-01  2.543e+04  0.000   1.000
## cap_shapek     -7.998e-01  2.720e+04  0.000   1.000
## cap_shapes     -6.907e-01  8.700e+04  0.000   1.000
## cap_shapex     -6.972e-01  2.482e+04  0.000   1.000
## cap_surfaceg    3.707e+00  1.900e+05  0.000   1.000
## cap_surfaces    1.795e-01  1.492e+04  0.000   1.000
## cap_surfacey    8.440e-03  1.146e+04  0.000   1.000
## cap_colorc     -2.370e+00  6.558e+04  0.000   1.000
## cap_colore     -4.315e-01  3.882e+04  0.000   1.000
## cap_colorg     -8.780e-02  3.679e+04  0.000   1.000
## cap_colorn     -4.820e-01  3.783e+04  0.000   1.000
## cap_colorp     -3.552e-02  4.544e+04  0.000   1.000
## cap_colorrr     4.301e-01  1.354e+05  0.000   1.000
## cap_coloru     4.439e-01  1.430e+05  0.000   1.000
## cap_colorw     4.546e-01  3.800e+04  0.000   1.000
## cap_colory     1.636e-01  3.949e+04  0.000   1.000
## bruiseest     -3.716e+01  1.294e+05  0.000   1.000
## odorc          5.654e+01  8.688e+04  0.001   0.999
## odorf          1.504e+01  1.532e+05  0.000   1.000
## odorl         -4.720e-02  2.694e+04  0.000   1.000
## odorm         -1.072e+02  3.002e+05  0.000   1.000
## odorn         -3.642e+01  1.303e+05  0.000   1.000
## odorp          5.326e+01  6.635e+04  0.001   0.999
## odors          1.501e+01  1.549e+05  0.000   1.000
## odory          1.501e+01  1.549e+05  0.000   1.000
## gill_attachmentf -2.794e-01  1.218e+05  0.000   1.000
## gill_spacingw   -4.076e+01  7.594e+04 -0.001   1.000
## gill_sizen      3.211e+00  7.264e+04  0.000   1.000
## gill_colore      8.503e+01  1.805e+05  0.000   1.000
## gill_colorg      8.529e+01  1.712e+05  0.000   1.000
## gill_colorh      8.524e+01  1.716e+05  0.000   1.000
## gill_colork      8.548e+01  1.720e+05  0.000   1.000
## gill_colorn      8.508e+01  1.724e+05  0.000   1.000
## gill_coloro      8.493e+01  1.834e+05  0.000   1.000
## gill_colorp      8.507e+01  1.718e+05  0.000   1.000
## gill_colorrr     8.531e+01  1.952e+05  0.000   1.000
## gill_coloru      8.499e+01  1.730e+05  0.000   1.000
## gill_colorw      8.504e+01  1.713e+05  0.000   1.000
## gill_colory      8.495e+01  1.808e+05  0.000   1.000
## stalk_shapet     4.153e+01  9.032e+04  0.000   1.000
## stalk_surface_above_ringk 8.935e-01  2.811e+04  0.000   1.000
## stalk_surface_above_rings 3.796e-01  2.117e+04  0.000   1.000
```

## stalk_surface_above_ringy	-4.073e+01	1.343e+05	0.000	1.000
## stalk_surface_below_ringk	2.964e-01	2.766e+04	0.000	1.000
## stalk_surface_below_rings	3.079e-01	2.118e+04	0.000	1.000
## stalk_surface_below_ringy	4.925e+00	6.413e+04	0.000	1.000
## stalk_color_above_ringc	NA	NA	NA	NA
## stalk_color_above_ringe	1.053e-01	6.444e+04	0.000	1.000
## stalk_color_above_ringg	-5.323e-02	3.421e+04	0.000	1.000
## stalk_color_above_ringn	-3.638e-03	2.708e+04	0.000	1.000
## stalk_color_above_ringo	-7.615e+01	2.274e+05	0.000	1.000
## stalk_color_above_ringp	2.002e-04	2.662e+04	0.000	1.000
## stalk_color_above_ringw	9.813e-02	3.033e+04	0.000	1.000
## stalk_color_above_ringy	6.090e-01	1.991e+05	0.000	1.000
## stalk_color_below_ringc	NA	NA	NA	NA
## stalk_color_below_ringe	6.311e-01	6.577e+04	0.000	1.000
## stalk_color_below_ringg	3.680e-01	3.675e+04	0.000	1.000
## stalk_color_below_ringn	-2.756e-01	2.527e+04	0.000	1.000
## stalk_color_below_ringo	NA	NA	NA	NA
## stalk_color_below_ringp	3.433e-01	2.977e+04	0.000	1.000
## stalk_color_below_ringw	6.088e-01	3.319e+04	0.000	1.000
## stalk_color_below_ringy	2.672e+00	1.039e+05	0.000	1.000
## veil_coloro	1.484e-01	5.204e+04	0.000	1.000
## veil_colorw	NA	NA	NA	NA
## veil_colory	NA	NA	NA	NA
## ring_numero	3.816e+00	1.329e+05	0.000	1.000
## ring_numbert	NA	NA	NA	NA
## ring_typef	-7.608e-01	1.631e+05	0.000	1.000
## ring_type1	4.245e+00	1.251e+05	0.000	1.000
## ring_typen	NA	NA	NA	NA
## ring_typep	-1.077e+00	8.723e+04	0.000	1.000
## spore_print_colorh	NA	NA	NA	NA
## spore_print_colork	-2.625e-02	7.357e+04	0.000	1.000
## spore_print_colorn	-2.430e-02	7.246e+04	0.000	1.000
## spore_print_coloro	-6.728e-02	7.420e+04	0.000	1.000
## spore_print_colorr	9.600e+01	2.104e+05	0.000	1.000
## spore_print_coloru	9.818e-03	1.088e+05	0.000	1.000
## spore_print_colorw	4.527e+01	1.600e+05	0.000	1.000
## spore_print_colory	-2.732e-02	7.353e+04	0.000	1.000
## populationc	8.064e+01	1.355e+05	0.001	1.000
## populationn	4.908e-01	4.759e+04	0.000	1.000
## populations	-3.733e-01	3.111e+04	0.000	1.000
## populationv	-1.275e+00	3.994e+04	0.000	1.000
## populationy	-1.389e+00	4.058e+04	0.000	1.000
## habitatg	5.815e-01	2.579e+04	0.000	1.000
## habitatl	-6.942e-01	2.133e+04	0.000	1.000
## habitatm	1.911e+00	5.579e+04	0.000	1.000
## habitatp	-1.732e-01	1.780e+04	0.000	1.000
## habitatu	5.936e-01	4.228e+04	0.000	1.000
## habitatw	-8.350e+01	1.968e+05	0.000	1.000
##				
## (Dispersion parameter for binomial family taken to be 1)				
##				
## Null deviance: 8.9997e+03 on 6497 degrees of freedom				
## Residual deviance: 4.2953e-08 on 6414 degrees of freedom				
## AIC: 168				

```
##
## Number of Fisher Scoring iterations: 25
```

From the above Summary of Coefficients, we find that 8 coefficients are not defined (NA) because of singularities. They are listed below.

```
s$aliased[which(s$aliased == TRUE)]
```

```
## stalk_color_above_ringc stalk_color_below_ringc stalk_color_below_ringo
##                      TRUE                      TRUE                      TRUE
##          veil_colorw          veil_colory          ring_number
##                      TRUE                      TRUE                      TRUE
##          ring_typen          spore_print_colorh
##                      TRUE                      TRUE
```

These 8 coefficients belong to 6 attributes. We will remove them and run the glm algorithm again.

```
train_set <- train_set %>% select(-stalk_color_above_ring, -stalk_color_below_ring,
                                -veil_color, -ring_number, -ring_type, -spore_print_color)
test_set <- test_set %>% select(-stalk_color_above_ring, -stalk_color_below_ring,
                               -veil_color, -ring_number, -ring_type, -spore_print_color)

start_time <- Sys.time()
fit_glm <- train(class ~ ., method = "glm", data = train_set)
time_diff <- Sys.time() - start_time

s <- summary(fit_glm)

s
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.525e-05 -2.226e-06 -2.100e-08  2.116e-06  3.727e-05
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.664e+02  1.260e+05  -0.001    0.999
## cap_shapec      9.738e-01  1.680e+05   0.000    1.000
## cap_shapef    -1.087e+00  1.563e+04   0.000    1.000
## cap_shapek    -1.114e+00  1.817e+04   0.000    1.000
## cap_shapes    -2.264e+00  7.429e+04   0.000    1.000
## cap_shapex    -1.290e+00  1.603e+04   0.000    1.000
## cap_surfaceg   4.889e-01  1.975e+05   0.000    1.000
## cap_surfaces   2.038e-01  1.226e+04   0.000    1.000
## cap_surfacey  -5.661e-02  1.007e+04   0.000    1.000
## cap_colorc    -3.651e+00  5.749e+04   0.000    1.000
## cap_colore    -1.623e+00  3.065e+04   0.000    1.000
## cap_colorg    -1.348e+00  2.810e+04   0.000    1.000
## cap_colorn    -1.558e+00  2.873e+04   0.000    1.000
```

```

## cap_colorp          -4.781e-01  2.632e+04  0.000  1.000
## cap_colorr          -4.012e+00  9.612e+04  0.000  1.000
## cap_coloru          -3.856e+00  1.064e+05  0.000  1.000
## cap_colorw          -4.874e-01  2.605e+04  0.000  1.000
## cap_colory          -6.139e-01  3.063e+04  0.000  1.000
## bruise            3.360e+01  4.344e+04  0.001  0.999
## odorc             1.242e+02  6.851e+04  0.002  0.999
## odorf             1.570e+02  5.843e+04  0.003  0.998
## odorl            -5.699e-02  2.344e+04  0.000  1.000
## odorm             1.120e+02  1.420e+05  0.001  0.999
## odorn             7.573e+01  4.411e+04  0.002  0.999
## odorp             9.330e+01  7.136e+04  0.001  0.999
## odors             1.570e+02  6.288e+04  0.002  0.998
## odory             1.570e+02  6.293e+04  0.002  0.998
## gill_attachmentf   -1.578e+01  1.311e+05  0.000  1.000
## gill_spacingw      -1.321e+01  3.629e+04  0.000  1.000
## gill_sizen         4.659e+01  5.007e+04  0.001  0.999
## gill_colore         4.181e+01  8.219e+04  0.001  1.000
## gill_colorg         4.280e+01  6.187e+04  0.001  0.999
## gill_colorh         4.220e+01  6.089e+04  0.001  0.999
## gill_colork         4.301e+01  6.495e+04  0.001  0.999
## gill_colorn         4.196e+01  6.102e+04  0.001  0.999
## gill_coloro         4.250e+01  9.426e+04  0.000  1.000
## gill_colorp         4.177e+01  6.012e+04  0.001  0.999
## gill_colorr         4.590e+01  9.472e+04  0.000  1.000
## gill_coloru         4.144e+01  6.195e+04  0.001  0.999
## gill_colorw         4.184e+01  6.160e+04  0.001  0.999
## gill_colory         4.262e+01  8.040e+04  0.001  1.000
## stalk_shapet       -3.092e+00  2.553e+04  0.000  1.000
## stalk_surface_above_ringk  9.114e-01  3.141e+04  0.000  1.000
## stalk_surface_above_rings -2.646e-02  2.650e+04  0.000  1.000
## stalk_surface_above_ringy  3.885e-01  7.873e+04  0.000  1.000
## stalk_surface_below_ringk  1.380e+01  3.349e+04  0.000  1.000
## stalk_surface_below_rings  1.385e+01  3.114e+04  0.000  1.000
## stalk_surface_below_ringy  4.556e+01  3.492e+04  0.001  0.999
## populationc         1.098e+01  1.063e+05  0.000  1.000
## populationn        -3.988e+00  4.403e+04  0.000  1.000
## populations        -3.685e+00  4.244e+04  0.000  1.000
## populationv        -2.998e+00  4.746e+04  0.000  1.000
## populationy        -3.846e+00  4.706e+04  0.000  1.000
## habitatg           4.285e+01  2.448e+04  0.002  0.999
## habitatl          -4.733e-01  2.120e+04  0.000  1.000
## habitatm           7.001e+01  6.209e+04  0.001  0.999
## habitatp          -3.721e-01  1.468e+04  0.000  1.000
## habitatu          -1.595e+01  3.134e+04  -0.001  1.000
## habitatw          -1.857e+01  9.844e+04  0.000  1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 8.9997e+03  on 6497  degrees of freedom
## Residual deviance: 6.6182e-08  on 6439  degrees of freedom
## AIC: 118
##
## Number of Fisher Scoring iterations: 25

```

```
s$aliased[which(s$aliased == TRUE)]
```

```
## named logical(0)
```

GLM results are tabulated below.

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1	1	1	1	18.5792 secs

3-2. LDA

Linear Discriminant Analysis (LDA) is an algorithm for predictive classification modeling problems. LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made.

```
start_time <- Sys.time()
fit_lda <- train(class ~ ., method = "lda", data = train_set)
time_diff <- Sys.time() - start_time

s <- summary(fit_lda)

s
```

```
##           Length Class      Mode
## prior          2  -none-  numeric
## counts          2  -none-  numeric
## means        182  -none-  numeric
## scaling        91  -none-  numeric
## lev            2  -none-  character
## svd             1  -none-  numeric
## N               1  -none-  numeric
## call           3  -none-    call
## xNames         91  -none-  character
## problemType     1  -none-  character
## tuneValue        1 data.frame list
## obsLevels        2  -none-  character
## param            0  -none-    list
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.00000	1.0000000	1	1.000000	18.579198 secs
lda	0.99877	0.9975366	1	0.997449	7.453696 secs

3-3. Naïve Bayes

Naïve Bayes algorithm is based on Bayes theorem and used for solving classification problems. It is a probabilistic classifier basing on the probability of an object to make prediction.

```

start_time <- Sys.time()
fit_nb <- train(class ~ ., method = "naive_bayes", data = train_set)
time_diff <- Sys.time() - start_time

summary(fit_nb)

##
## ===== Naive Bayes =====
##
## - Call: naive_bayes.default(x = x, y = y, laplace = param$laplace, usekernel = FALSE)
## - Laplace: 0
## - Classes: 2
## - Samples: 6498
## - Features: 91
## - Conditional distributions:
##   - Gaussian: 91
## - Prior probabilities:
##   - e: 0.518
##   - p: 0.482
##
## -----

```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs

3-4. svmLinear

svmLinear stands for Support Vector Machine (SVM) Linear Model. It fits a linear SVM model by identifying the optimal decision boundary that separates data points from different classes, and then predicts the class of new observations based on this separation boundary.

```

start_time <- Sys.time()
fit_svmLinear <- train(class ~ ., method = "svmLinear", data = train_set)
time_diff <- Sys.time() - start_time

fit_svmLinear["finalModel"]

## $finalModel
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc (classification)
## parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 172
##
## Objective Function Value : -0.5854

```

```
## Training error : 0
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs

3-5. KNN

The k-nearest neighbors (KNN) algorithm predicts the outcome of a new observation by comparing it to k similar cases in the training data set. The best tune here is $k = 5$.

```
start_time <- Sys.time()
fit_knn <- train(class ~ ., method = "knn", data = train_set)
time_diff <- Sys.time() - start_time

fit_knn["finalModel"]
```

```
## $finalModel
## 5-nearest neighbor model
## Training set outcome distribution:
##
##      e      p
## 3366 3132
```

```
fit_knn$bestTune
```

```
##      k
## 1 5
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs

3-6. gamLoess

gamLoess stands for Generalized Additive Model using LOESS (Local weighted regression). Comparing to the bin smoother approach with constant assumptions in KNN, Loess considers larger window size (span = 0.5 here) with fitting a line within that window than with a constant.

```
start_time <- Sys.time()
fit_gamLoess <- train(class ~ ., method = "gamLoess", data = train_set)
time_diff <- Sys.time() - start_time
```



```
fit_gamLoess$bestTune
```

```
## span degree
## 1 0.5 1
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs
gamLoess	1.0000000	1.0000000	1.0000000	1.0000000	35.879180 secs

3-7. multinom

Multinomial Regression is an extension of the logistic regression. It is specially designed for the nominal data. The target dependent variable can have more than two classes. Although we only have two classes, we still apply the algorithm to check the result.

```
fit_multinom["finalModel"]
```

```
## $finalModel
## Call:
## nnet::multinom(formula = .outcome ~ ., data = dat, decay = param$decay)
##
## Coefficients:
## (Intercept) cap_shapec cap_shapef
## -2.819550073 3.400077256 0.056124886
## cap_shapek cap_shapes cap_shapex
## 0.595182807 -3.858049706 -0.217264263
## cap_surfaceg cap_surfaces cap_surfacey
## 4.277432807 0.165351663 0.516287764
## cap_colorc cap_colore cap_colorg
## -1.554741951 -0.968901419 -0.336063206
## cap_colorn cap_colorp cap_colorr
## -1.352387071 1.076368615 -1.828651491
## cap_coloru cap_colorw cap_colory
## -1.000274808 0.060028113 -0.739873349
## bruiseest odorc odorf
## -0.781497354 26.895628665 8.124700816
## odorl odorm odorn
## -2.571585248 0.834948770 -0.101635753
## odorp odors odory
## 22.237652295 6.033750203 6.022079029
## gill_attachmentf gill_spacingw gill_sizen
## 0.804375609 -7.723352407 8.541999723
## gill_colore gill_colorg gill_colorh
## -0.575874375 0.022998163 0.618067815
## gill_colork gill_colorn gill_coloro
## -0.827936096 -0.667897362 -0.318572945
```

```

##          gill_colorp          gill_colorr          gill_coloru
##      -0.068225672          3.583573583      -0.016924303
##          gill_colorw          gill_colory          stalk_shapet
##      -0.157046819      -0.253858638          6.688129515
## stalk_surface_above_ringk stalk_surface_above_rings stalk_surface_above_ringy
##      -0.729293484      -1.907337107      -5.818469519
## stalk_surface_below_ringk stalk_surface_below_rings stalk_surface_below_ringy
##      -0.118124466      -0.057282952          7.252676693
## stalk_color_above_ringc stalk_color_above_ringe stalk_color_above_ringg
##          0.834948770          0.009633426          0.793719654
## stalk_color_above_ringn stalk_color_above_ringo stalk_color_above_ringp
##      -0.163158535      -5.908794691          0.483518525
## stalk_color_above_ringw stalk_color_above_ringy stalk_color_below_ringc
##          1.137406075          6.861897586          0.834948770
## stalk_color_below_ringe stalk_color_below_ringg stalk_color_below_ringn
##      -0.049935321          1.029710528      -0.316001912
## stalk_color_below_ringo stalk_color_below_ringp stalk_color_below_ringw
##      -5.908794691          0.768317531          1.390583561
## stalk_color_below_ringy          veil_coloru          veil_colorw
##          5.573980063      -0.694488978      -3.772652967
##          veil_colory          ring_numbero          ring_numbert
##          6.861897586      -0.730881909      -2.923616934
##          ring_typef          ring_typed          ring_typedn
##      -15.931078906          6.691989157          0.834948770
##          ring_typep          spore_print_colorh          spore_print_colork
##          0.789896366          8.407304566      -6.495861580
## spore_print_colorn          spore_print_coloru          spore_print_colorr
##      -7.008289123      -0.695150205          20.157854537
## spore_print_coloru          spore_print_colorw          spore_print_colory
##      -8.242631542          2.822777713      -0.487666495
##          populationc          populationn          populations
##          8.201781704      -1.253888584      -3.126985385
##          populationv          populationy          habitatg
##      -3.473488709      -3.915874701          1.501166468
##          habitatl          habitatm          habitatp
##      -1.009552876          2.501467651      -1.019150609
##          habitatu          habitatw
##      -1.377527471      -12.451930648
##
## Residual Deviance: 0.8177269
## AIC: 168.8177

```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs
gamLoess	1.0000000	1.0000000	1.0000000	1.0000000	35.879180 secs
multinom	1.0000000	1.0000000	1.0000000	1.0000000	51.942764 secs

3-8. Classification Model

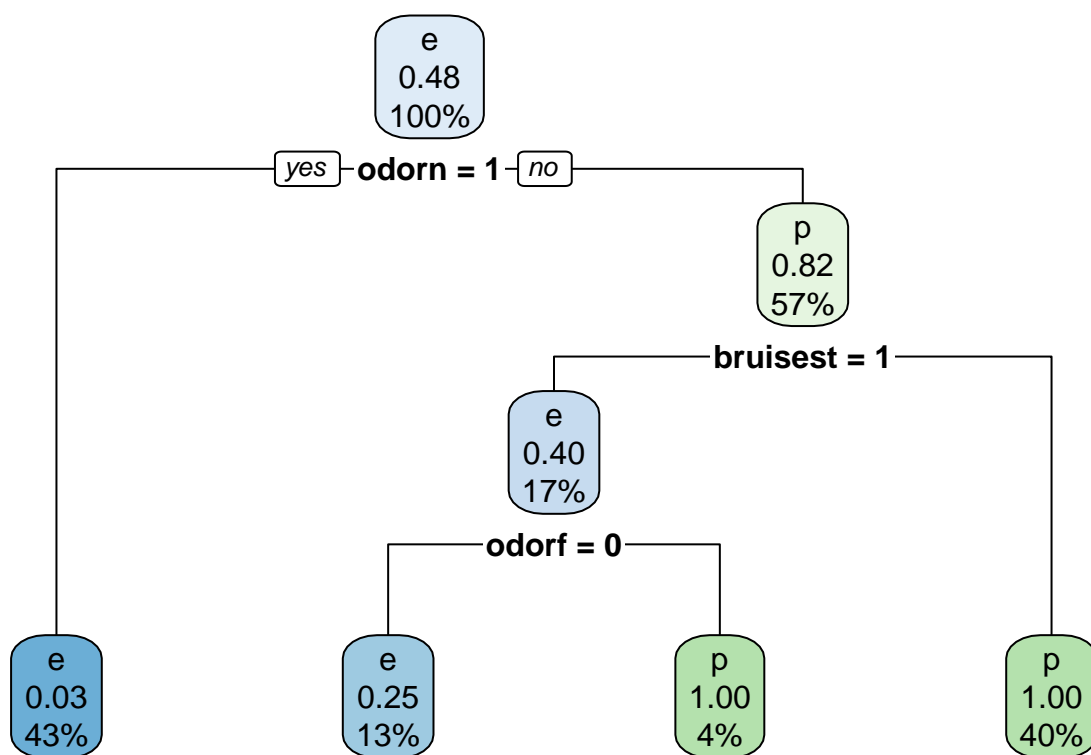
Classification Model is basing decision tree. It works by repeatedly partitioning data into multiple sub-spaces. The outcome in each final sub-space is as homogeneous as possible. Both text and visualized binary tree are presented here.

```
start_time <- Sys.time()
fit_rpart <- train(class ~ ., method = "rpart", data = train_set)
time_diff <- Sys.time() - start_time

fit_rpart["finalModel"]
```

```
## $finalModel
## n= 6498
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
##  1) root 6498 3132 e (0.5180055 0.4819945)
##    2) odorn>=0.5 2812   96 e (0.9658606 0.0341394) *
##    3) odorn< 0.5 3686  650 p (0.1763429 0.8236571)
##      6) bruiseest>=0.5 1089  439 e (0.5968779 0.4031221)
##        12) odorf< 0.5 861  211 e (0.7549361 0.2450639) *
##        13) odorf>=0.5 228    0 p (0.0000000 1.0000000) *
##        7) bruiseest< 0.5 2597    0 p (0.0000000 1.0000000) *
```

```
rpart.plot(fit_rpart$finalModel)
```



Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs
gamLoess	1.0000000	1.0000000	1.0000000	1.0000000	35.879180 secs
multinom	1.0000000	1.0000000	1.0000000	1.0000000	51.942764 secs
classif. model	0.9575646	0.9147627	1.0000000	0.9119898	3.620821 secs

3-9. Random Forest

Random Forest algorithm addresses the shortcomings of decision tree with a special type of bagging to the data and bootstrap sampling at each split (bootstrap aggregating). This means that at each splitting step of the tree algorithm, a random sample of n predictors is chosen as split candidates from the full set of the predictors. As suggested in the course, we used “Rborist” method rather than “rf” method because of time efficiency. Variable Importance shows that $\text{Odorn} = 1$ is the most important. This is in-line with the decision tree in Classification Model.

```

start_time <- Sys.time()
fit_rf <- train(class ~ ., method = "Rborist", data = train_set)
time_diff <- Sys.time() - start_time

fit_rf$bestTune

```

```
## predFixed minNode
## 2          46          2
```

```
varImp(fit_rf)
```

```
## Rborist variable importance
##
## only 20 most important variables shown (out of 91)
##
## Overall
## odorn 100.000
## odorf 39.022
## gill_sizen 33.929
## bruise 16.142
## stalk_surface_above_ringk 11.979
## odorp 9.422
## odorl 8.110
## spore_print_colorr 8.103
## stalk_surface_below_ringk 7.364
## ring_typep 5.826
## spore_print_colorh 4.896
## spore_print_colorw 4.170
## cap_colory 3.337
## ring_numbert 3.282
## habitatm 2.316
## stalk_surface_below_ringy 2.285
## odorc 1.631
## populationv 1.489
## habitatu 1.480
## populationn 1.369
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs
gamLoess	1.0000000	1.0000000	1.0000000	1.0000000	35.879180 secs
multinom	1.0000000	1.0000000	1.0000000	1.0000000	51.942764 secs
classif. model	0.9575646	0.9147627	1.0000000	0.9119898	3.620821 secs
random forest	1.0000000	1.0000000	1.0000000	1.0000000	219.782776 secs

3-10. Adaboost

Instead of growing tree randomly, Adaptive Boosting algorithm grows trees using information from previously grown trees, with the aim to minimize the error of the previous models.

```
start_time <- Sys.time()
fit_adaboost <- train(class ~ ., method = "adaboost", data = train_set)
time_diff <- Sys.time() - start_time
```

```
fit_adaboost["finalModel"]
```

```
## $finalModel
## fastAdaboost::real_adaboost(formula = .outcome ~ ., data = dat,
##   nIter = param$nIter)
## .outcome ~ .
## <environment: 0x0000000044b2af58>
## Dependent Variable: .outcome
## No of trees:2
```

```
varImp(fit_adaboost)
```

```
## ROC curve variable importance
##
##               Importance
## gill_color      100.000
## gill_size       81.984
## bruises         81.889
## ring_type       74.291
## stalk_surface_above_ring 68.214
## stalk_surface_below_ring 63.288
## gill_spacing    41.236
## population      38.514
## habitat         37.529
## stalk_color_above_ring 31.225
## stalk_color_below_ring 28.406
## cap_surface     24.677
## odor           21.152
## ring_number     17.526
## stalk_shape     16.432
## cap_shape        7.066
## cap_color        6.969
## veil_color       6.083
## gill_attachment  4.806
## spore_print_color 0.000
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs
gamLoess	1.0000000	1.0000000	1.0000000	1.0000000	35.879180 secs
multinom	1.0000000	1.0000000	1.0000000	1.0000000	51.942764 secs
classif. model	0.9575646	0.9147627	1.0000000	0.9119898	3.620821 secs
random forest	1.0000000	1.0000000	1.0000000	1.0000000	219.782776 secs
adaboost	1.0000000	1.0000000	1.0000000	1.0000000	109.758860 secs

3-11. ensemble

Ensemble model combines the results from previous 10 algorithms. If more than 50% algorithms predict edible, it will predict edible.

```
start_time <- Sys.time()
y_hat_results <- bind_cols(y_hat_glm, y_hat_lda, y_hat_nb,
                           y_hat_svmLinear, y_hat_rpart,
                           y_hat_knn, y_hat_gamLoess, y_hat_multinom,
                           y_hat_rf, y_hat_adaboost)
time_diff <- Sys.time() - start_time

y_hat_ensemble <- ifelse(rowMeans(y_hat_results == "e") > 0.5, "e", "p")
```

Method	Accuracy	Kappa	Sensitivity	Specificity	Train_Time
glm	1.0000000	1.0000000	1.0000000	1.0000000	18.579198 secs
lda	0.9987700	0.9975366	1.0000000	0.9974490	7.453696 secs
navie bayes	0.9440344	0.8877932	0.9619952	0.9247449	11.266552 secs
svmLinear	1.0000000	1.0000000	1.0000000	1.0000000	11.871701 secs
knn	1.0000000	1.0000000	1.0000000	1.0000000	97.885378 secs
gamLoess	1.0000000	1.0000000	1.0000000	1.0000000	35.879180 secs
multinom	1.0000000	1.0000000	1.0000000	1.0000000	51.942764 secs
classif. model	0.9575646	0.9147627	1.0000000	0.9119898	3.620821 secs
random forest	1.0000000	1.0000000	1.0000000	1.0000000	219.782776 secs
adaboost	1.0000000	1.0000000	1.0000000	1.0000000	109.758860 secs
ensemble	1.0000000	1.0000000	1.0000000	1.0000000	0.006001 secs

Result

From the above table, the lowest accuracy is coming from Naïve Bayes followed by Classification Model and LDA. Other algorithms are all 100% accurate. The Kappa of these 3 algorithms are lower than 1 (from 0.88 to 0.99), which means that it is slightly affected by randomness. For Sensitivity, only Naïve Bayes is lower than 1. For Specificity, these 3 algorithms are lower than 100% but they are in high tier of 9x%. For ensemble model, we can not find improvement because 7 out of 10 algorithms obtain the 100% accuracy in test set already.

In terms of performance (basing on my PC, 6 cores, 3.4GHz, 32G Ram), it takes more than 200 secs to finish the training process by Random Forest and more than 100 secs by Adaboost. The more trees created in the forest the lowest performance it is. KNN takes about 90 secs to finish because it best-tunes the results in CARET package. Ensemble model consumes less than 0.01 sec by only using the in RAM results from other algorithms with one line of IF code.

Conclusion

In this report, we run through 10 basic machine learning algorithms with the mushroom data set. The accuracy of the prediction is at least 94% and 7 out of 10 algorithms are 100% accurate. In terms of performance, algorithms basing on bootstrap aggregating decision tree (Random Forecast / Adaboost) and neighbors (KNN / Loess) consume longer training time than linear regression and probability based models. PCA (Principal Components Analysis) can be a next step for future study.