STSCI 4740 PROJECT REPORT

Cancer Data Prediction

Hanqing Wu, Yuanhong Cao, Shaohan Wang

ABSTRACT

Starting the treatment early is a great way for keeping the cancer at the bay. The purpose of this project is to use machine to predict the cancer level, based on patients' vital indexes(Age, Obesity, Alcohol use, etc.).We used variety kinds of models to do this multi-class classification, including Linear Discriminant Analysis, K-Nearest Neighbors, Support Vector Machine, Naive Bayes, Multinomial Regression, Decision Tree and Random Forest. As result, we compare the performance of each models, which are close to perfect, and evaluate the prediction accuracy to select the best model for this project. We think the great result is because of a clean and non-noise dataset, in the future we will consider do this project on a more real-world dataset.

1. Data Preparation and Descriptive Statistics

In the original dataset, each case accounts for the detailed physical information recorded of an individual cancer patient. To favor the application of machine to predict the respective cancer levels with the included predictors, we split the 1000-observation dataset to train set and test set with a proportion of 80% and 20% respectively. Now, we could inspect and compare these two datasets separately with respect to some descriptive statistics. Based on the bar-plots Figure 1 Figure 2, the training set and test set have the similar trend in the number of patients having each respective cancer levels. Besides, because of the difference in total data points included, the relationship or trend between cancer levels and ages are different between these datasets according to the box-plots Figure 3 Figure 4. In test set, patients with level "High" has the lowest average

age and in training set, patients with level "Low" has the lowest average age. We also plot the density graphs Figure 5 Figure 6 of `Coughing_of_Blood` which is proved to be the most important variable in the random forest method with respect to these two sets. This time we could find that the patients with level "High" has the strictly more coughing of blood than that in the other two levels for both train and test sets.

2. Modelling and Prediction

In this section, we selected 7 different machine learning models, training them on the training set. And then we perform the prediction task on the test set. The criterion used for comparing different methods are test error rate which is computed as follows

$$\#\{\text{level}_{\text{predict}} \neq \text{level}_{\text{true}}\}/\#\{\text{observations in the test set}\}$$

2.1. Linear Discriminant Analysis

In class, we have learned Linear Discriminant Analysis based on Bayes' theorem to analyze the classification problem. Recall Bayes' theorem,

$$\mathbb{P}(Y = k|X = x) = \mathbb{P}(X = x|Y = k) * \mathbb{P}(Y = k)/\mathbb{P}(X = x).$$

And we can slightly rewrite it as

$$\mathbb{P}(Y = k|X = x) = \frac{\pi_k * f_k(x)}{\sum_{l=1}^{K} \pi_l * f_l(x)}.$$

where π_k is the prior prob, $f_k(x)$ is the density function of X and $\mathbb{P}_k(x)$ is the posterior prob with respect to the kth class in Y.

Then, we classify the new point X=x by determining which class has the largest postrior prob or we can say that the Bayes classifier is to assign X=x to

the class with the largest $\delta_k(x)$ which is called the discriminant function of X=x.

Now, we can apply LDA method to our dataset "cancer data".We fit LDA to all the variables except our response variable "Level" with respect to training set and we could take a glimpse to our LDA model for some values such as prior prob π_k and model coefficients. We then calculate the test error which is the fraction of patient in the test set whose cancer level is misclassified based on our test set. First, we use our LDA model to predict the test cancer level value and create a table which contains the true value and the predicted value. We could simply calculate the test error rate by checking how many predicted cancer level value are missclassified to wrong level with respect to the true value.

In conclusion, we have obtained a very good test error rate around 0.035 by applying LDA method here and the reason is probably that this original dataset `cancer_data` might be well-separated and enough cleaned.

2.2. K-Nearest Neighbors

K-nearest neighbors is another clear and useful method that could be directly applied to classification problem. The intuition behind KNN method is that we classify or estimate $\mathbb{P}_j(x)$ by calculating the fraction of points in jth class around all the data points around x. The formula for KNN is quite straightforward

$$\mathbb{P}(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} l(y_i = j).$$

And we have different cluster N with K points in the training data that are closest to our desired data point.

Now, choosing a good value for K is critical because of the effect of K. If we choose K=1, our KNN model will be too flexible and might overfit the data points. Besides, we also cannot choose a very large K which may lead to biased prediction. Therefore, when determining K, we should consider bias-variance trade-off effect.

Here, we can apply KNN method to our dataset ‘cancer data‘ and we select K=3. Following a similar training and testing procedure as that of LDA, we have obtained a perfect test error rate 0 by applying KNN method here, so we have the reason to believe that KNN method perfectly predict the cancer level value based on test set using the corresponding data of the predictors.

2.3. Support Vector Machine

In machine learning, support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N the number of features) that distinctly classifies the data points. In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

In this cancer project, we factorize the "Level" variable for classification. Next step is to train the SVM model. We set the kernel as "Polynomial" since it is a multi-class classification with multiple features; After that, the SVM is trained with the 80% training set, and ready for testing. By using the 20% testing set, we acquired the accuracy of 100%, which is a perfect test error rate 0. Therefore we have reason to believe that the SVM model works perfectly on prediction on the cancer data.

2.4. Naive Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where

the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. Using Bayes' theorem, the conditional probability can be decomposed as

$$P(C_k|x) = \frac{P(C_k)P(x|C_k)}{P(x)}$$

The Naive Bayes Classifier is to computes the conditional posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule. Next step is to train the Naive Bayes model with the 80% training set, and ready for testing. By using the 20% testing set, we acquired the accuracy of 86%, which is a relatively good test error rate around 0.14. In conclusion, the Naive Bayes classifier may not have great performance as other models.

2.5. Multinomial regression

In class, we have already learnt that we could use generalized linear regression models like the logistic regression model for classification tasks. Although in the slides, the logistic regression model is only applied to binary classification problem, it also works for multi-class problems if we change the link function.

We know that for the binary problem, the link function could be written as $\log\left(\frac{p}{1-p}\right) = \mathbf{X}\boldsymbol{\beta}$, where p is the probability of being of the reference level.

Now, in order to perform the multiclass classification task, we can propose the following link function

$$\log\left(\frac{p_i}{p_K}\right) = \mathbf{X}\boldsymbol{\beta}_i$$

where p_i is the probability of being of level i and K is the number of classes. We choose the `nnet` package which has function `multinom` that fits multinomial

log-linear models via neural networks.

The `decay` parameter, which is used in updating the learning rate in neural network, is tuned. It helps the optimization process and prevents over-tting. We used 10-folds cross-validation to search the best decay parameter among decays $= (0, 0.01, 0.1, 1, 10)$. The corresponding CV-error is $(0.00125, 0.00125, 0.00125, 0.0175, 0.08)$. This means that it is optimal to set `decay` as 0.

Under this setting, the `multinom` model predicts the test set perfectly. Maybe this is because the provided predictors' combination are located separately enough for sample of different cancer levels.

2.6. Decision Tree

Decision tree is a simple but useful model to solve classification problem. We made use of the package `rpart` to conduct decision tree analysis and Gini index is evaluated as the splitting criterion.

How is Gini index calculated and what does it mean? Gini Index, also known as Gini impurity, calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly. The Gini index of a dataset D is $\text{Gini}(D) = 1 - \sum_{i=1}^{K} p_i^2$, where p_i is the probability of a randomly selected observation belonging to class i. Clearly, if all observation in the dataset belongs to the same class, the Gini index would be zero. And if the observation in the dataset are approximately distributed then the Gini index will be relatively large. The Gini index of a predictor x is calculated similarly,

$$\text{Gini}(D, x) = \sum_{v=1}^{V} \frac{|D_V|}{|D|} \text{Gini}(D)$$

Where V is the number of distinct possible value of predictor x. $|D_V|$ and $|D|$ are the number of observations of the sub-dataset where $x = v$ and the number of observation of the whole dataset respectively. A predictor with a higher Gini

index tends to provide more information related to classification for us.

Therefore, each time when we add branches to our decision tree, we build up the criterion based on the predictor with the highest Gini index. In our practice, there is a parameter in `rpart` that is called `cp` which could be viewed as the decrease of total lack of fit. Any improvement that is less than `cp` will be ignored and therefore no new branch will be generated. We selected 5 candidate value of `cp` that is $(0, 0.01, 0.05, 0.1, 0.2)$. And the corresponding error of cross-validation is $(0.00625, 0.00625, 0.05250, 0.13875, 0.13875)$. Therefore, the best `cp` is 0.

We then generated a decision tree that is depicted by Figure 7. Its structure is quite simple, only a few predictors are actually taken into consideration. The decision process is quite intuitive. Basically, the more features indicating unhealthiness a patient has, the higher level of cancer he or she tends to suffer. The performance of this tree on the test set returns a very small test error, 0.025.

2.7. Random Forest

The tree does not generate perfect prediction. Is there any way to enhance the performance of tree-based model? The answer is yes. One such powerful method is called random forest. Its idea is that if a single tree does not solve all problems, more trees could help. Moreover, in addition to the bootstrapping sampling for each tree, it also has something to do with feature selection. When training a random forest model, each time when a split in a tree is considered, we randomly select some predictors and force the split based on only these predictors. The final prediction of a test sample's class is determined by a majority vote.

We just fit a random forest model using the R package `randomForest` with the default setting. This powerful bagging method does not let us down and we also have a perfect prediction on the test set – zero test error!

From Figure 8 we could see, just as the tree plot in the previous section, the most important predictor is `Coughing_Of_Blood`. However, the second most important `Passive_Smoker` does not even appear in the plot of single decision tree. This does illustrate the importance of random predictor selection – some-

times in the training process key predictors could be overlooked so we need to give them a chance.

3. Conclusion

In this project, we utilized 7 distinctive machine learning models to conduct prediction task on the cancer data. To have a straightforward look, see the following table comparing the test error of the 7 methods.

Method	Test Error	Method	Test Error
LDA	0.035	KNN	0
SVM	0	Naive Bayes	0.140
Multinomial Regression	0	Decision Tree	0.025
Random Forest	0		

Table 1.: Comparing the test error of 7 methods

As we can see, KNN, SVM, Multinomial Regression and Random Forest are tie for the first. And the most inferior method seems to be Naive Bayes. This phenomenon offers two insights regarding this project.

First, the cancer data is somewhat "well-split" in the data space. This means that with proper method like SVM, we can generate three non-intersecting hyper-dimensional balls called "low", "medium", "high" to wrap all the test data points of level "low", "medium", "high" inside respectively.

Second, a more complicated model tends to give more accurate prediction. This is not always true for all data prediction tasks. But in our case, models with more parameters make use of more information from the training data and because of the training data and test data share a quite similar pattern there is no need to worry about over-fitting.

This project provide us a good chance to put what we have learnt in the class into practice. It enhances our understanding of the theoretical models and also strengthen our ability of using R to implement data science.

4. Appendices

Appendix A. Figures

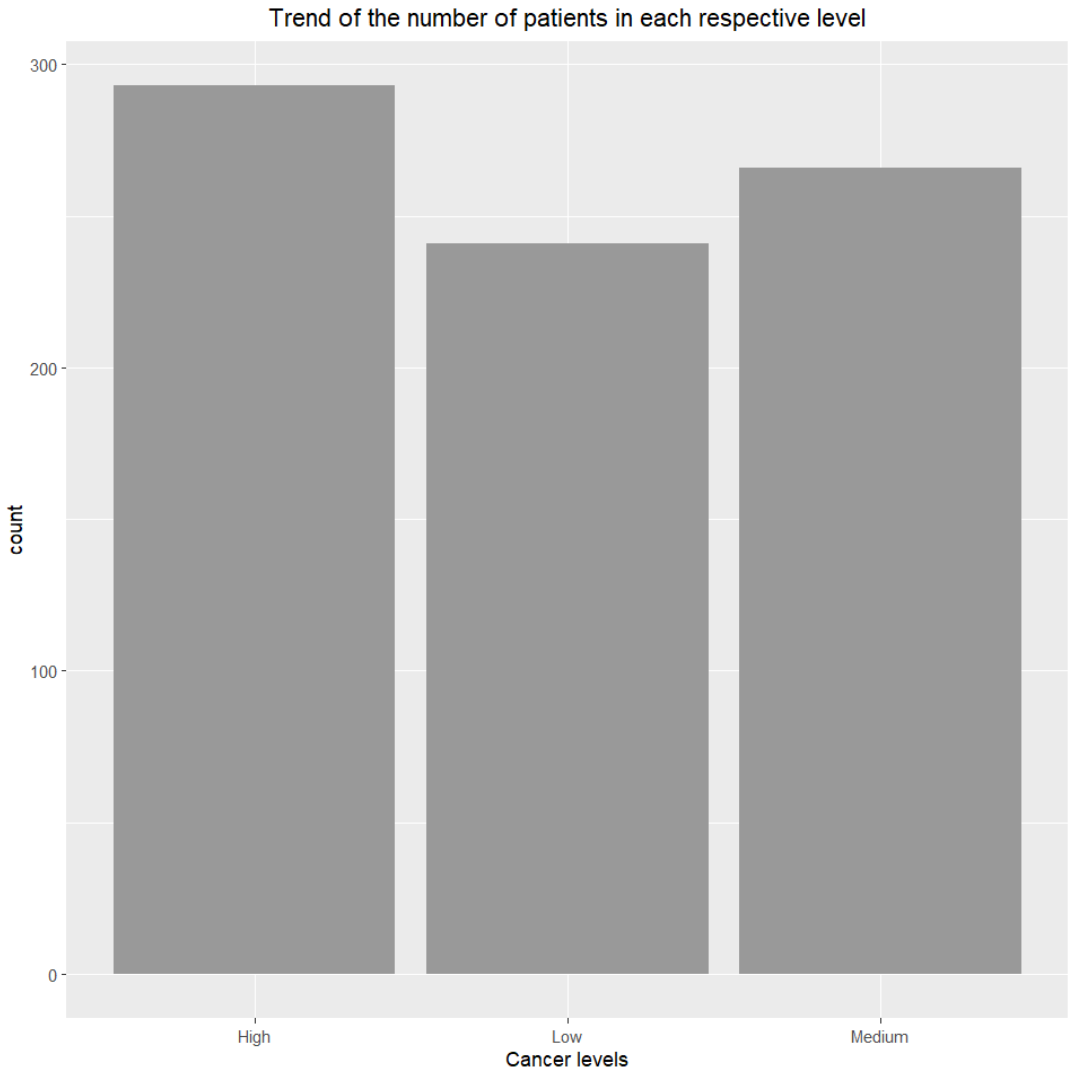

Figure 1.: The Bar Plot of the Training Set

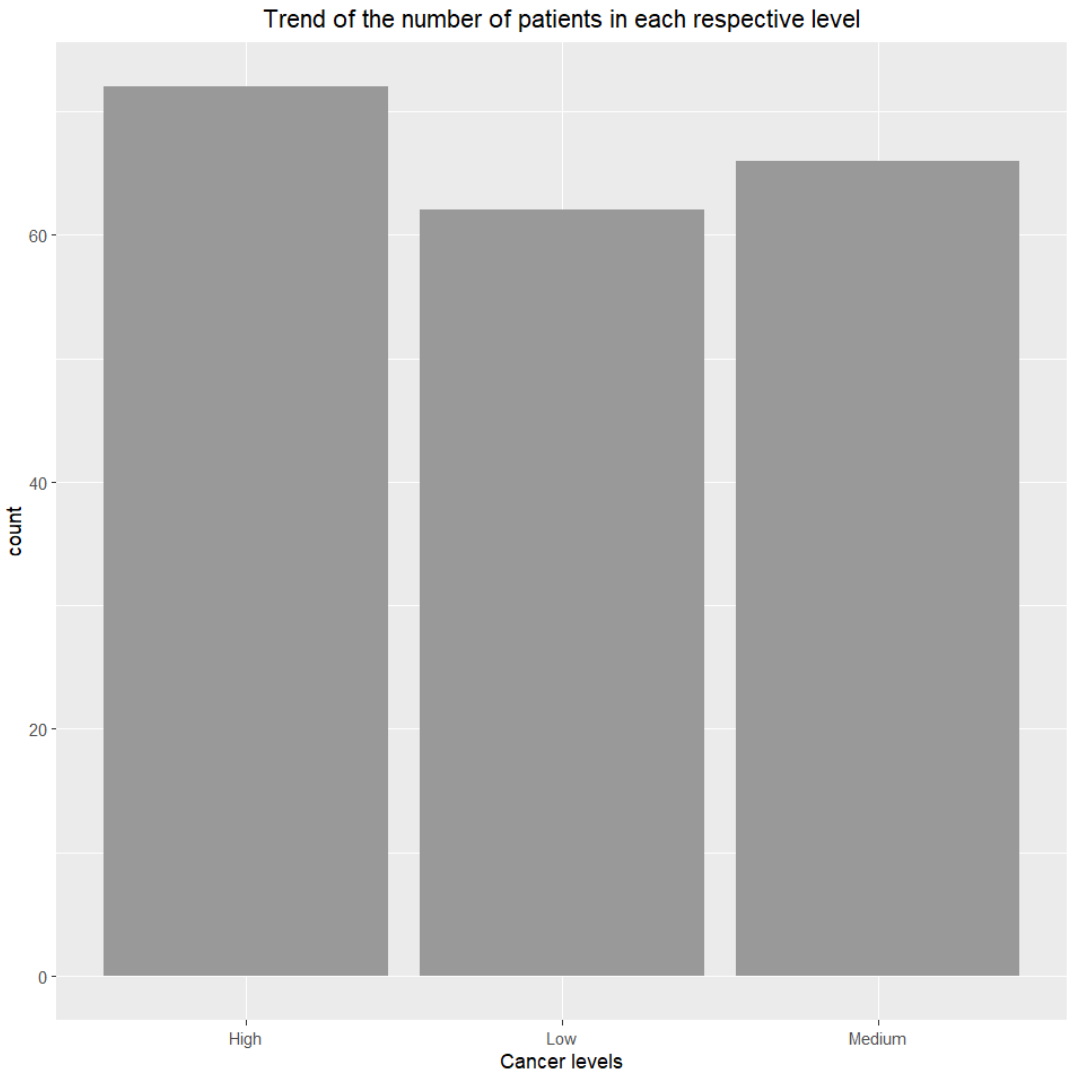

Figure 2.: The Bar Plot of the Test Set

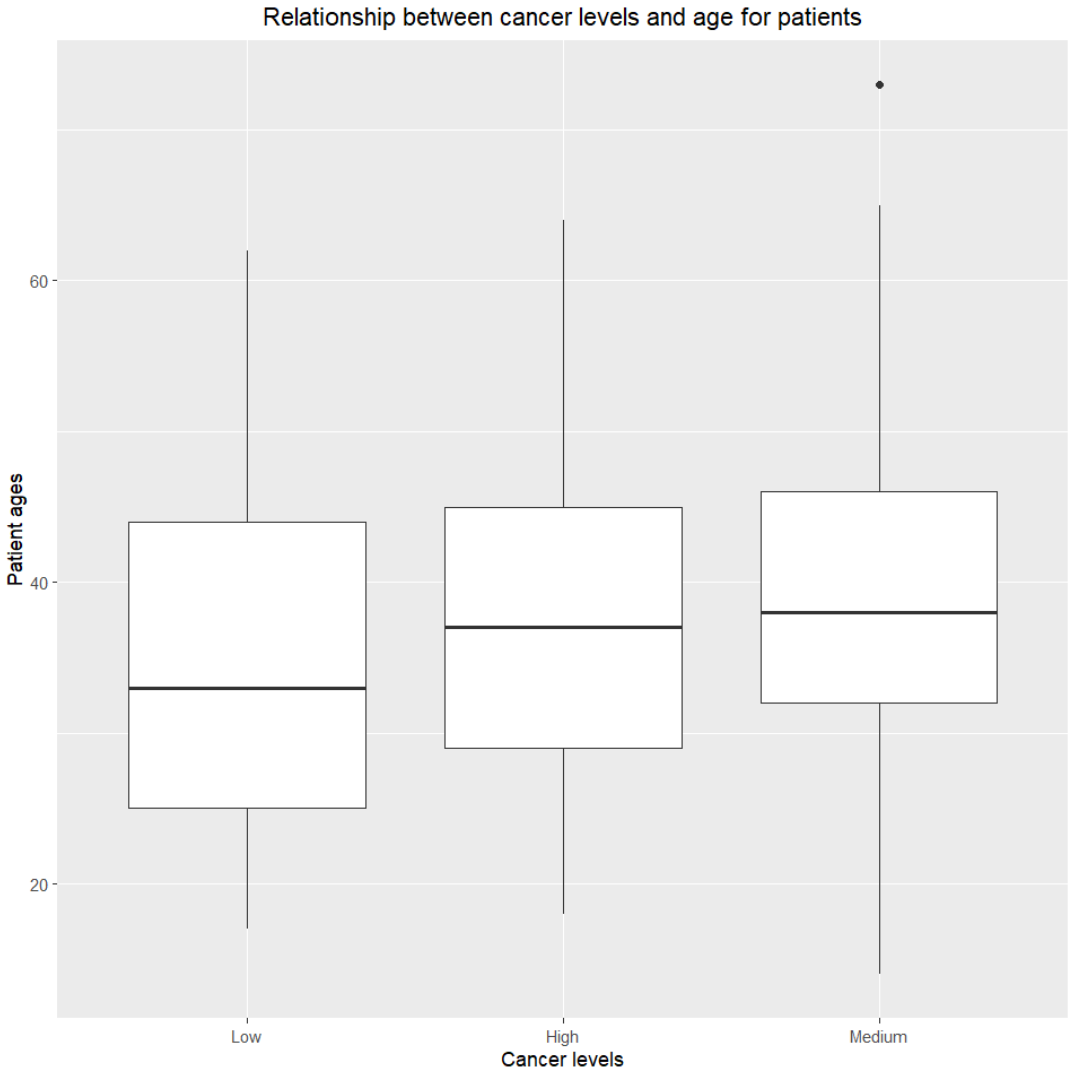

Figure 3.: The Box Plot of the Training Set

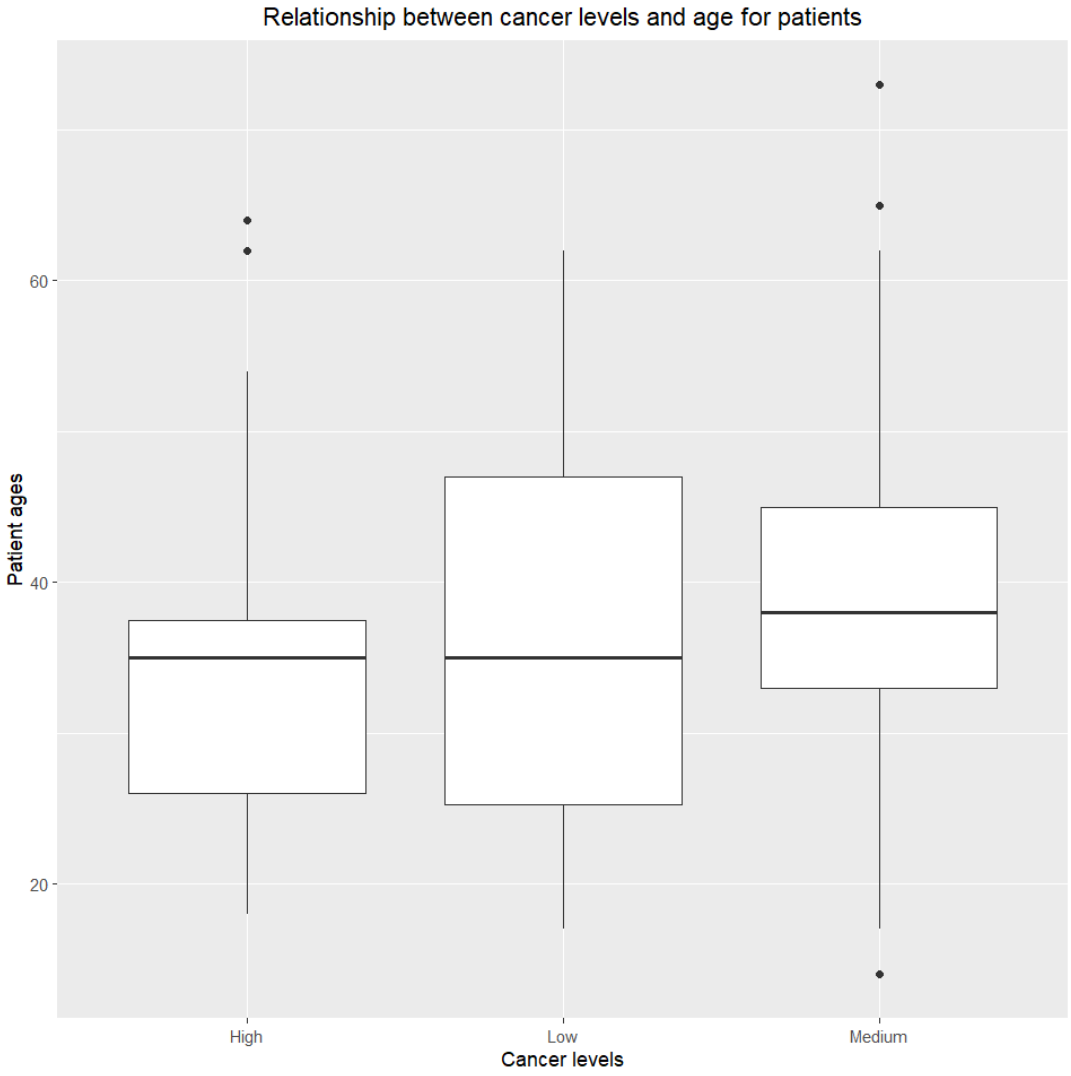

Figure 4.: The Box Plot of the Test Set

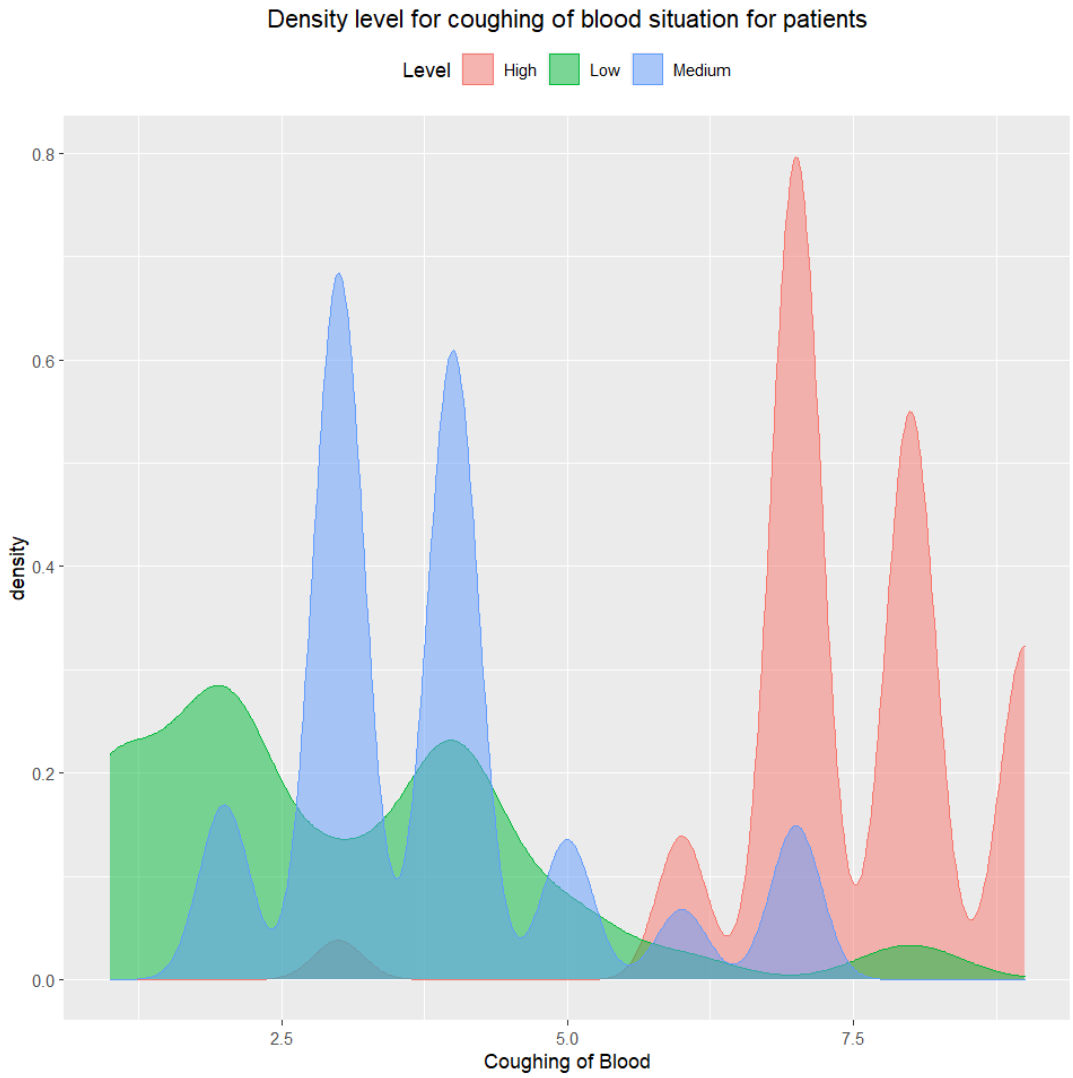

Figure 5.: The Train Density Plot

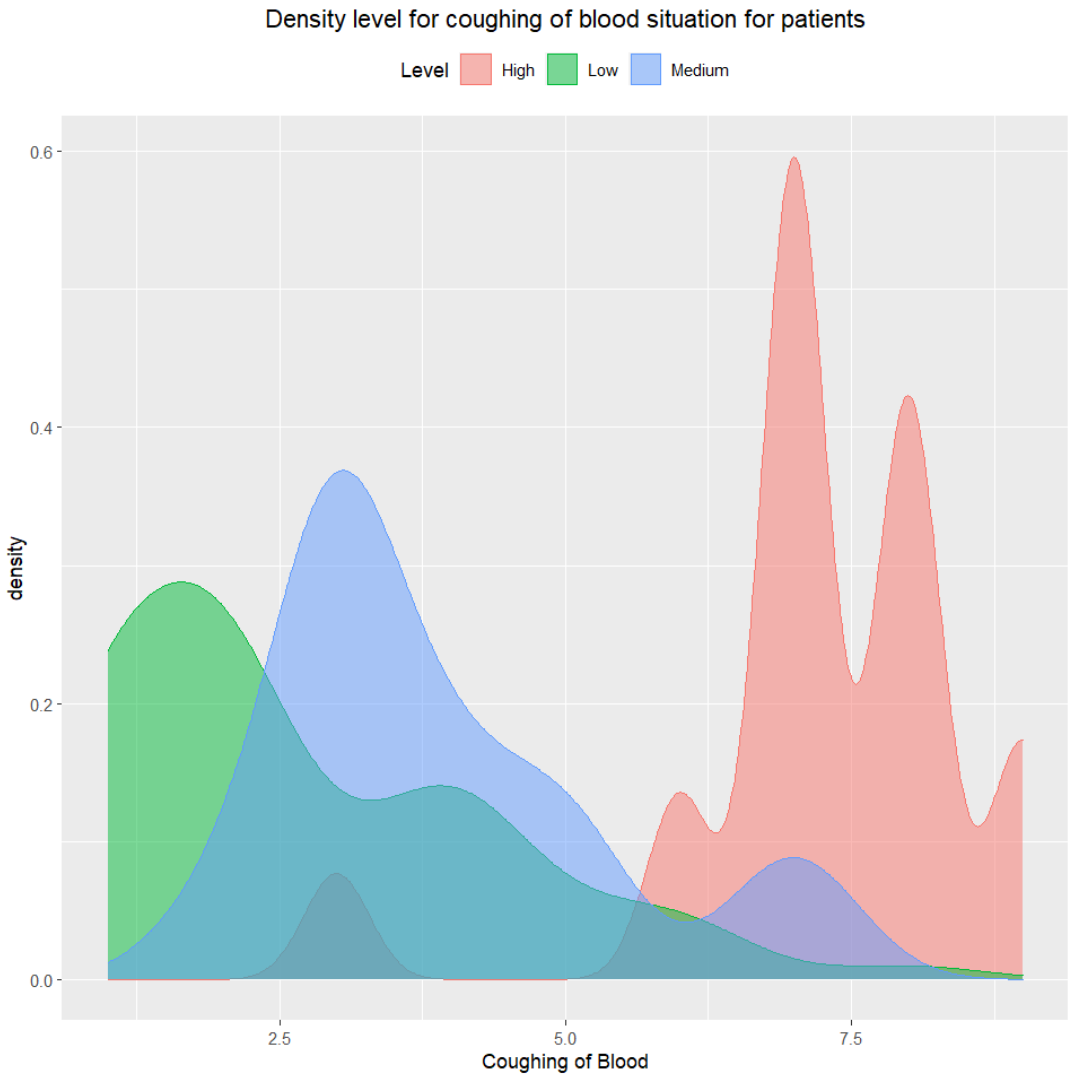

Figure 6.: The Density Plot of the Test Set

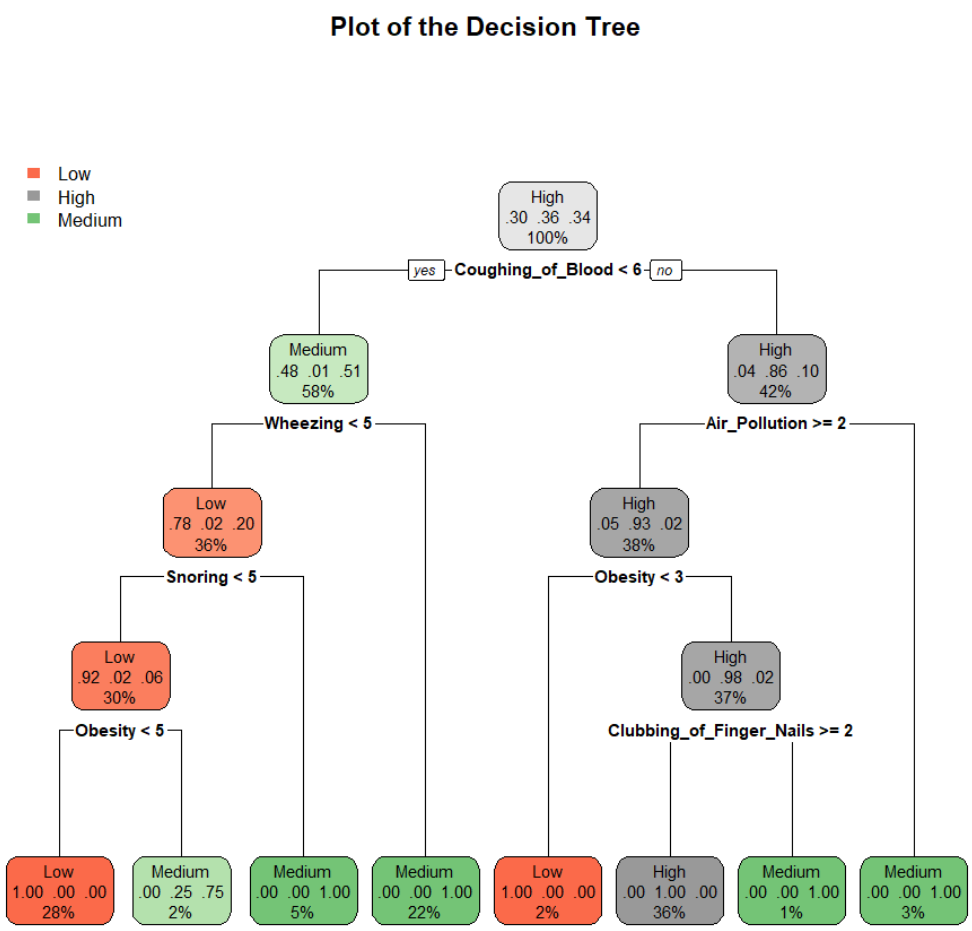

Figure 7.: The Decision Tree

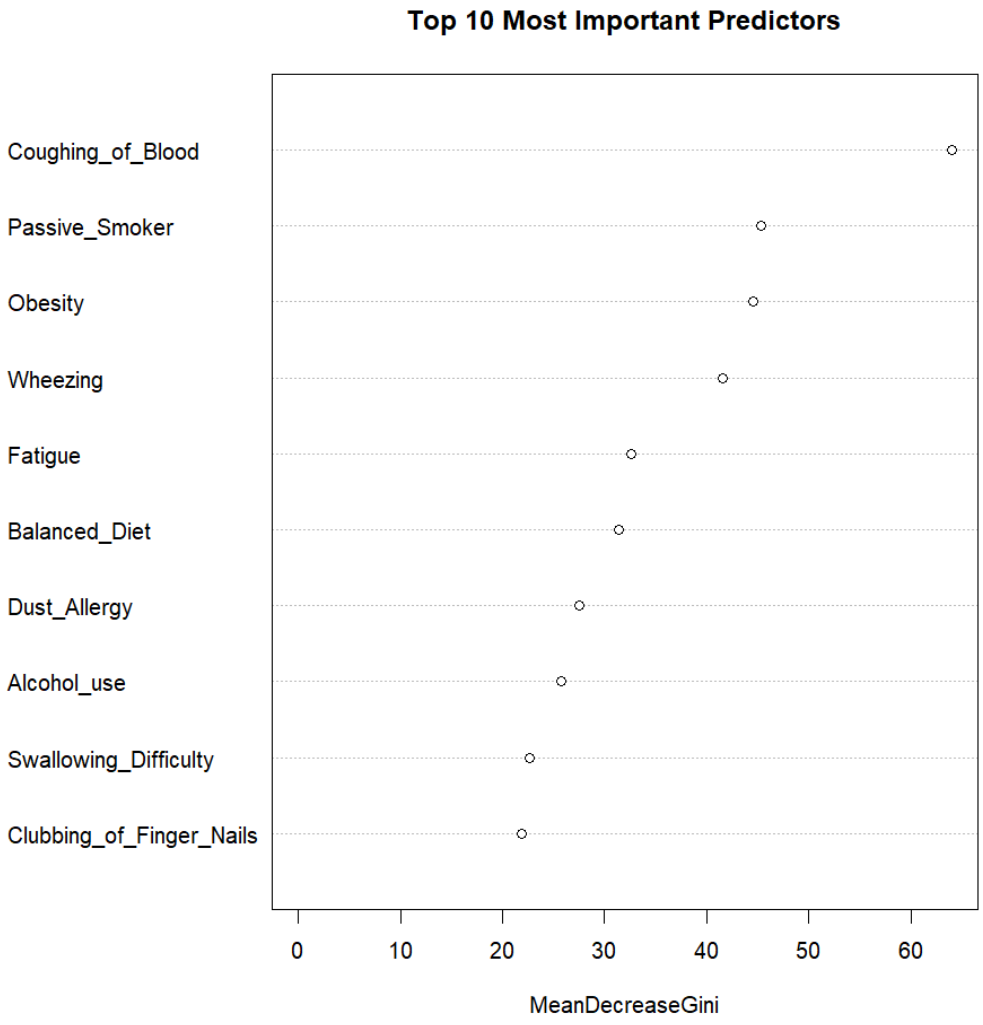

Figure 8.: Top 10 Important Predictors in the Random Forest Method

Appendix B. R Code

```
# The following is the work of Hanqing Wu
rm(list=ls())
setwd("E:\\Cornell\\2020 Fall\\ML\\Project")

library(readxl)
library(nnet)
library(boot)
library(rpart)
library(rpart.plot)
library(randomForest)

set.seed(1)
dataset = read_xlsx("cancer_data.xlsx", sheet =
   "Sheet1")
dataset$Level = as.factor(dataset$Level)
dataset$Gender = as.factor(dataset$Gender)
dataset$Level = relevel(dataset$Level, ref="Low"
   )
for (i in 1:ncol(dataset)) {
  colnames(dataset)[i] = gsub(" ", "_", colnames
     (dataset)[i])
}
# Use 80% training set and 20% test set

# train_idx = sort(sample(seq(1, nrow(dataset),
   by=1), size = 0.8*nrow(dataset)))
# test_idx = setdiff(seq(1, nrow(dataset), by=1)
   , train_idx)
train_set = dataset[train_idx, -1]
test_set = dataset[test_idx, -1]

# write.csv(train_idx, "cancer_train_idx.csv")
# write.csv(test_idx, "cancer_test_idx.csv")
# write.csv(train_set, "cancer_train.csv")
```

```
30 # write.csv(test_set, "cancer_test.csv")
31
32 # The dataset has been split
33 info = list()
34 for (i in 1:ncol(train_set)) {
35   info[colnames(train_set)[i]] = unique(train_
       set[, i])
36 }
37
38 # Since the level of each variable are large,
     for simplicity
39 # we may treat each variable as continuous
     random variable
40 # like age. Perform crosstest by hand.
41
42 DECAY = c(0, 0.01, 0.1, 1, 10)
43 MAXIT = 1000
44
45 cv_error_multinom=rep(0,5)
46 set_idx = seq(1, nrow(train_set))
47 cv_train_idx = list()
48 for (i in 1:10){ # 10-folds crosstest
49   train_idx = sort(sample(set_idx, size = 0.1*
       nrow(train_set)))
50   set_idx = setdiff(set_idx, train_idx)
51   cv_train_idx[[i]] = train_idx
52 }
53
54 for (i in 1:10) {
55   train_idx = setdiff(seq(1, nrow(train_set)),
       cv_train_idx[[i]])
56   cv_train_set = train_set[train_idx, ]
57   cv_test_set = train_set[cv_train_idx[[i]], ]
58   for (j in 1:length(DECAY)) {
59     multi_fit = multinom(Level~., data=cv_train_
         set, decay=DECAY[j], maxit=MAXIT)
```

```
60      cv_pred = predict(multi_fit, cv_test_set,
           type = "class")
61      cv_error_multinom[j] = cv_error_multinom[j]
           +
62        sum(cv_pred != unlist(cv_test_set[, ncol(
             cv_test_set)])) / nrow(cv_test_set) /
             10
63    }
64  }
65
66  seq(1, length(DECAY))[cv_error_multinom == min(
      cv_error_multinom)] # c(1,2)
67  z = summary(multi_fit)$coefficients/summary(
      multi_fit)$standard.errors
68  # 2-tailed Wald z tests to test significance of
      coefficients
69  p = (1 - pnorm(abs(z), 0, 1)) * 2
70  p
71
72  # Since the first 2 parameters give the same
      result, We choose the decay to be 0.
73  # It is shocking that the prediction error is 0.
74
75  multi_fit=multinom(Level~., data=train_set,
      decay=DECAY[1], maxit=MAXIT)
76  multinom_pred = predict(multi_fit, test_set,
      type = "class")
77  # multinom_prob = predict(multi_fit, test_set,
      type = "prob")
78  multinom_error = sum(multinom_pred != unlist(
      test_set[, ncol(test_set)])) / nrow(test_set)
79
80  # The second method is tree-based method.
81  CP = c(0, 0.01, 0.05, 0.1, 0.2)
82  cv_error_tree=rep(0, 5)
83  for (i in 1:10){ # 10-folds crosstest
```

```
84    train_idx = setdiff(seq(1, nrow(train_set)),
         cv_train_idx[[i]])
85    cv_train_set = train_set[train_idx, ]
86    cv_test_set = train_set[cv_train_idx[[i]], ]
87    for (j in 1:length(DECAY)) {
88      tree_fit = rpart(Level~., data=train_set,
           method="class", cp=CP[j])
89      cv_pred = predict(tree_fit, cv_test_set,
           type = "class")
90      cv_error_tree[j] = cv_error_tree[j] +
91        sum(cv_pred != unlist(cv_test_set[, ncol(
             cv_test_set)])) / nrow(cv_test_set) /
             10
92    }
93  }
94  seq(1, length(CP))[cv_error_tree == min(cv_error
      _tree)] # c(1,2)
95
96  tree_fit = rpart(Level~., data=train_set, method
      ="class", cp=CP[1])
97  tree_pred = predict(tree_fit, test_set, type = "
      class")
98  tree_error = sum(tree_pred != unlist(test_set[,
      ncol(test_set)])) / nrow(test_set)
99  View(cbind(tree_pred, unlist(test_set[, ncol(
      test_set)])))
100 tree_pred != unlist(test_set[, ncol(test_set)])
101 png(filename="TreePlot.png", width=960, height
      =960, res=120)
102 rpart.plot(tree_fit, main="Plot of the Decision
      Tree")
103 graphics.off()
104
105 # The third method could be randomForest.
106 rf = randomForest(Level~., data=train_set)
107 rf_pred = predict(rf, test_set, type = "class")
```

```
rf_error = sum(rf_pred != unlist(test_set[, ncol
    (test_set)])) / nrow(test_set)
png(filename="RFImp.png", width=960, height=960,
     res=120)
varImpPlot(rf, n.var=10, main = "Top 10 Most
    Important Predictors")
graphics.off()
save.image(file="project_hw.RData")
# load("project_hw.RData")

# The following is the work of Yuanhong Cao and
    Shaohan Wang
# clean up workspace environment
rm(list = ls())
# load library
library(class)
library(boot)
library(MASS)
library(glmnet)
library(e1071)
library(forecast)
library(caret)
library(ggplot2)
library(dplyr)

# Load dataset
library(readxl)
cancer_data <- read_xlsx("cancer_data.xlsx",
    sheet = "Sheet1")

# Clean dataset
cancer_data = na.omit(cancer_data)
glimpse(cancer_data)

# Use 80% training set and 20% test set
set.seed(1)
```

```
train_idx = sort(sample(seq(1, nrow(cancer_data)
   , by=1), size = 0.8*nrow(cancer_data)))
test_idx = setdiff(seq(1, nrow(cancer_data), by
   =1), train_idx)
training_set = cancer_data[train_idx, -1]
test_set = cancer_data[test_idx, -1]

# Exploratory Data Analysis (EDA)
# inspect our dataset
summary(training_set)
summary(validation_set)
ncol(training_set)
ncol(validation_set)
nrow(training_set) # case of training_set
nrow(validation_set) # case of validation_set

# Diagnostic plot
# png(filename="barplot_train.png", width=960,
   height=960, res=120)
training_set %>%
  ggplot(aes(Level)) +
  geom_bar(fill = 'grey60')+
  xlab('Cancer levels') +
  ggtitle('Trend of the number of patients in
     each respective level') +
  theme(plot.title = element_text(hjust = 0.5))
# graphics.off()

# png(filename="barplot_test.png", width=960,
   height=960, res=120)
validation_set %>%
  ggplot(aes(Level)) +
  geom_bar(fill = 'grey60')+
  xlab('Cancer levels') +
  ggtitle('Trend of the number of patients in
     each respective level') +
```

```
169   theme(plot.title = element_text(hjust = 0.5))
170 # graphics.off()
171
172 # png(filename="boxplot_train.png", width=960,
      height=960, res=120)
173 training_set %>%
174   ggplot(aes(x = reorder(Level, Age, mean), y =
        Age)) +
175   geom_boxplot(stat = 'boxplot') +
176   xlab('Cancer levels') +
177   ylab('Patient ages') +
178   ggtitle('Relationship between cancer levels
        and age for patients') +
179   theme(plot.title = element_text(hjust = 0.5))
180 # graphics.off()
181
182 # png(filename="boxplot_test.png", width=960,
      height=960, res=120)
183 validation_set %>%
184   ggplot(aes(x = reorder(Level, Age, mean), y =
        Age)) +
185   geom_boxplot(stat = 'boxplot') +
186   xlab('Cancer levels') +
187   ylab('Patient ages') +
188   ggtitle('Relationship between cancer levels
        and age for patients') +
189   theme(plot.title = element_text(hjust = 0.5))
190 # graphics.off()
191
192 # png(filename="densityplot_train.png", width
      =960, height=960, res=120)
193 training_set %>%
194   ggplot(aes(x = `Coughing of Blood`)) +
195   geom_density(aes(fill = Level, color = Level),
        alpha = 0.5) +
196   ggtitle('Density level for coughing of blood
```

```
      situation for patients') +
197   theme(legend.position = "top") +
198   theme(plot.title = element_text(hjust = 0.5))
199 # graphics.off()
200
201 # png(filename="densityplot_test.png", width
      =960, height=960, res=120)
202 validation_set %>%
203   ggplot(aes(x = `Coughing of Blood`)) +
204   geom_density(aes(fill = Level, color = Level),
        alpha = 0.5) +
205   ggtitle('Density level for coughing of blood
      situation for patients') +
206   theme(legend.position = "top") +
207   theme(plot.title = element_text(hjust = 0.5))
208 # graphics.off()
209
210 # Linear Discriminant Analysis
211 # use set.seed(1) to make results reproducible
212 set.seed(1)
213
214 # perform linear discriminant analysis using all
      the variables in training set
215 lda_model = lda(Level~ ., data = training_set)
216 lda_model
217 plot(lda_model)
218
219 # compute the test set error which is the
      fraction of patient in the test set whose
      cancer level is misclassified
220 lda_predict = predict(lda_model, test_set)
221 lda_predict$class # prediction result
222 lda_class = lda_predict$class
223 table(lda_class, test_set$Level)
224 mean(lda_class != test_set$Level) # estimate the
      test error
```

```
225
226 # K-Nearest Neighbors
227 # use set.seed(1) to make results reproducible
228 set.seed(1)
229
230 # Transform variables in dataset
231 train_X = training_set[,-24] #X for training
      data
232 test_X = test_set[,-24] #X for testing data
233 train_Y = training_set$Level #Y for training
      data
234
235 # perform K-Nearest Neighbors analysis using all
       the variables in training set and
236 knn_predict = knn(train_X,test_X,train_Y,k=3)
237 knn_predict
238 plot(knn_predict)
239
240 # compute the test set error which is the
      fraction of patient in the test set whose
      cancer level is misclassified
241 table(knn_predict, test_set$Level)
242 mean(knn_predict != test_set$Level) # estimate
      the test error
243
244 # Support Vector Machines
245 predictfeat <- c("Level")
246 x_train = training_set[ , !(names(training_set)
      %in% predictfeat)]
247 y_train = as.factor(training_set$Level)
248 x_test = test_set[ , !(names(test_set) %in%
      predictfeat)]
249 y_test = as.factor(test_set$Level)
250
251 svmmodel <- svm(x_train, y_train, kernel ="
      polynomial", gamma =1,cost =1)
```

```
252 print(svmmodel)
253 summary(svmmodel)
254
255 y_pred <- predict(svmmodel, x_test)
256 confusionMatrix(y_pred, y_test)
257
258 # Naive Bayes
259 set.seed(120)  # Setting Seed
260 nbmodel <- naiveBayes(x_train, y_train)
261
262 y_pred2 <- predict(nbmodel, x_test)
263 confusionMatrix(y_pred2, y_test)
```