Statistical Learning Project 4

Laimei Yip Lundstrom, Anton Holm Klang

INTRODUCTION

Erythemato-squamous diseases is a group of common skin diseases that are categorised into six classes. However, when diagnosing patients suffering from these diseases, it is hard for clinicians to distinguish which one of the six classes the patients belongs to. This project sets out to mine a set of data using decision trees with the aim of giving clinicians a method to distinguish between these diseases as well as lessen their workload and simplify their workflow. Two models using decision trees, one simpler and the other more complex, are constructed and their prediction accuracy are investigated. We get the result that both models are equally competitive in terms of prediction errors. For the more complex model, an analysis on the relative importance of the input variables is done with the aim of helping clinicians cut down on unnecessary testing. Finally, the pros and cons of the two models are discussed and suggestions for improvements in future studies are put forward.

DATA

The data for this project is retrieved from the ICU Machine Learning Repository and is named "Dermatology Data Set". The original data set consists of 366 observations of which each is a tuple of 35 elements with 1 response variable and 34 input variables. The response variable is a discrete variable with 6 nominal levels, each corresponding to a disease class. 32 of the predictor variables are ordinal with levels between 0 and 3 where 0 indicates that the symptom is not present and 3 the largest amount of the symptom is present. Input variable *family history* is binary while *age* is integer-valued. There are 8 observations with missing values in the input variable *age*. Since this number is not large, we take the approach of removing them completely from the data set. The table in Figure 1 shows the number of observations in each disease class and a complete list of input variables is available in the appendix. The upper right panel in Figure 1 shows the empirical distribution of the six disease classes and the lower left panel shows the correlations amongst the input variables. While we do not perform a dimension reduction, we use the correlation matrix plot as a form of sanity check. From now on, we say *class* to denote *disease class*.

MODELS

We explore a range of models and select one that we think is compatible with the nature of the data set as well as most suited to the needs of the clinicians.

KNN: Due to the fact that we have a large amount of predictors, we will end up with sparse data and due to the curse of dimensionality, the data will be inclined to lie close to the boundary of the sample space and as such, lie far away from each other (see Hastie, Tibshirani and Friedman [1] Equation (2.24) to explicitly calculate the smallest distance between two points). As such, when using KNN we have two choices, either select a small K in order to avoid using points far away from our observation when estimating the response resulting in a model with high variance or choose K to be large to avoid the high variance, forcing the model to use points far away the observation resulting in a model with high bias. For these reasons, KNN is not used.

Ridge & Lasso: By using squared-error or absolute-error on a classification dataset we will penalize correctly classified observations which is not a property we want in our model. Hence, the use of ridge regression and lasso is not to prefer in this case.

LDA & QDA: Due to the fact that our predictors are all ordered factors, we can draw the conclusion that they are not gaussian and as such, the use of LDA or QDA is prohibited.

Logistic Regression: The degrees of freedom of a model using logistic regression is equal to $(K-1)(p+1)$ where K is the number of classes and p is the number of predictors. In our case, even if we would be able to reduce the dimension of the predictor space by half, we would end up with a degree of freedom around 100 on a dataset of only around 300 observations. At the same time, the sparsity in contingency tables will be immense due to the fact that we have many predictors, each with 4 levels. Many cells will have very few, if any, observations and as such, small changes in the data could result in very different models, and as such a model using logistic regression would in our case suffer from high variance.

Based on our analysis of the different learning methods above, we come to the conclusion that a decision tree is most suited due to its interpretablility. Since the aim of this project is partly to simplify the workflow of clinicians, the simple use of decision trees should be well appreciated. Following this line of thought, we investigate two methods: 1) a simple classification tree where a full tree is fitted and then pruned; 2) gradient boosted trees. The former has the advantage of being easy to interpret and visualize as well as naturally implementing dimension reduction, which implies less tests might be needed to arrive at a diagnosis. The latter, however, does not have these properties but has better prediction accuracy. We try to make up for the loss of interpretability by providing relative variable importance plots and partial dependence plots (see Section Model 2 for details). Our intention is to accomodate to varying levels of medical knowledge of clinicians, testing capabilities of medical institutions and access to information technology. Specifically, we have in mind the simpler model for clinicians in less developed regions where they most likely have limited access to medical training and testing facilities.

In training our models, we use a 10-fold cross validation. We reason that with a 10-fold we get about 29 data points in each validation set, which is quite a reasonable quantity (with reference to what we did in Project 1). Leave-one-out cross validation is not an option for us since we do not have a huge dataset.

Model 1: Simple Classification tree

Here, we deploy what we have learnt in Project 1 and use the `rpart` package to construct the tree.

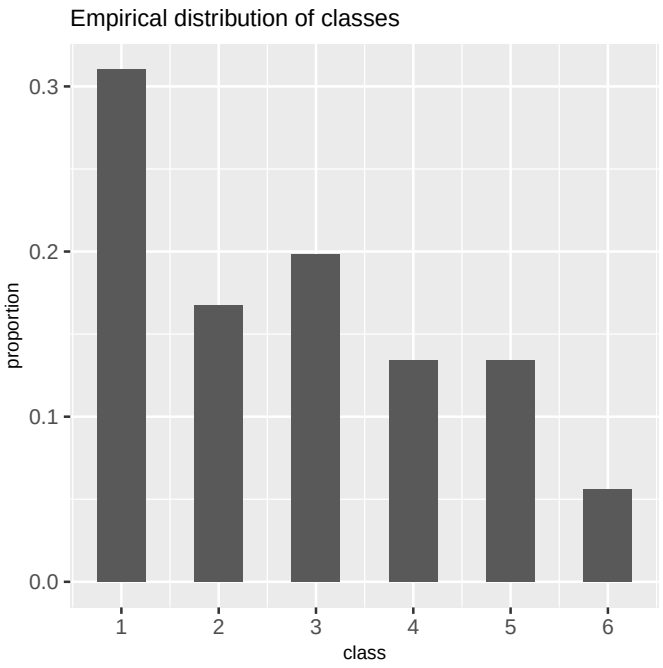
Using the algorithm suggested in [1], we first build a "big" tree T_0 for every training set. Then for a fixed value of pruning parameter α, we let the `prune` function find the subtree $T \in T_0$ that minimizes the cost-complexity criterion as in Equation (9.16) in [1]. According to the package documentation of `rpart` by Therneau and Atkinson [2], the impurity measure $Q_m(T)$ used is the Gini index. We use a 10-fold cross validation to find the optimal α. Specifically, our algorithm is as follows:

- Construct a sequence of tuning parameter $\alpha = \{\alpha_i\}$, for $i = 1, 2, ..., m$. After some initial testing, we decided to test the range of $\alpha = [0, 0.2]$ with 30 values.
- For each training set, grow a large tree T_{0j}, for $j = 1, 2, ..., 10$. We set the control point as minsplit = 10.
- For every large tree, run the `prune` function on every α_i, $i = 1, 2, ..., m.$, to find a corresponding subtree T_{ji}. Then compute the mean misclassification error of each training set using its corresponding validation set.
- Compute the CV error for every α_i by taking the average of all mean misclassification errors from the 10 training sets.

Figure 2 shows the corresponding CV error for each α. While the easiest choice is to choose the α that returns the lowest CV error, we need to bear in mind that α controls the size of a tree and the bigger the tree, the higher the risk of overfitting. Hence, we use the one-standard-error-bar rule and choose the largest possible α that is still within standard error bar of the lowest *alpha*, which in this case is the fourth smallest *alpha* value of 0.0207. Figure 3 shows the resulting pruned tree, which is also our final classification tree. We test its prediction power using the held-out test set and get a misclassification error of 0.042.

As we can see in Figure 3, of all the 34 predictors, the final tree uses only 6 of them of which 2 are clinical and the rest are histopathological. As we already know, decision trees are very easy to interpret and has the inherent capability to sieve out predictors that are not relevant. Figure 3 clearly demonstrates these nice properties and from a medical practitioner's point of view, carrying out 6 instead of 34 checks can really help to streamline workflow and improve patient flow. Further, cutting down the time to diagnosis allows patients to receive treatment earlier and lead better patient outcomes.

Class	Name	No. of instances
1	Psoriasis	112
2	Seboreic Dermatitis	61
3	Lichen Planus	72
4	Pityriasis Rosea	49
5	Chronic Dermatitis	52
6	Pityriasis Rubra Pilaris	20



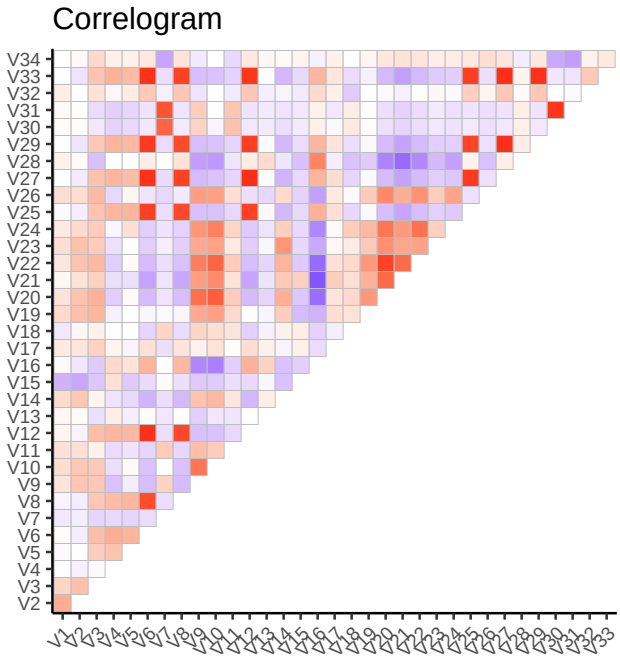


Figure 1: A simple analysis of the data set.

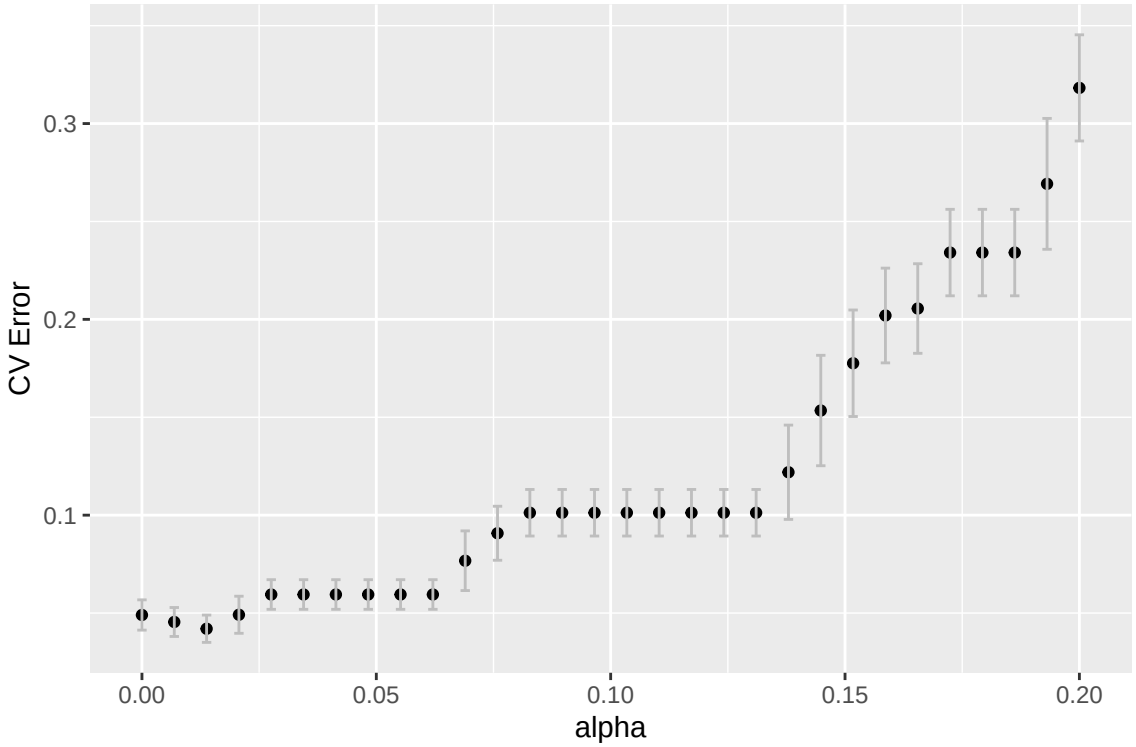


Figure 2: CV errors for all 30 α values.

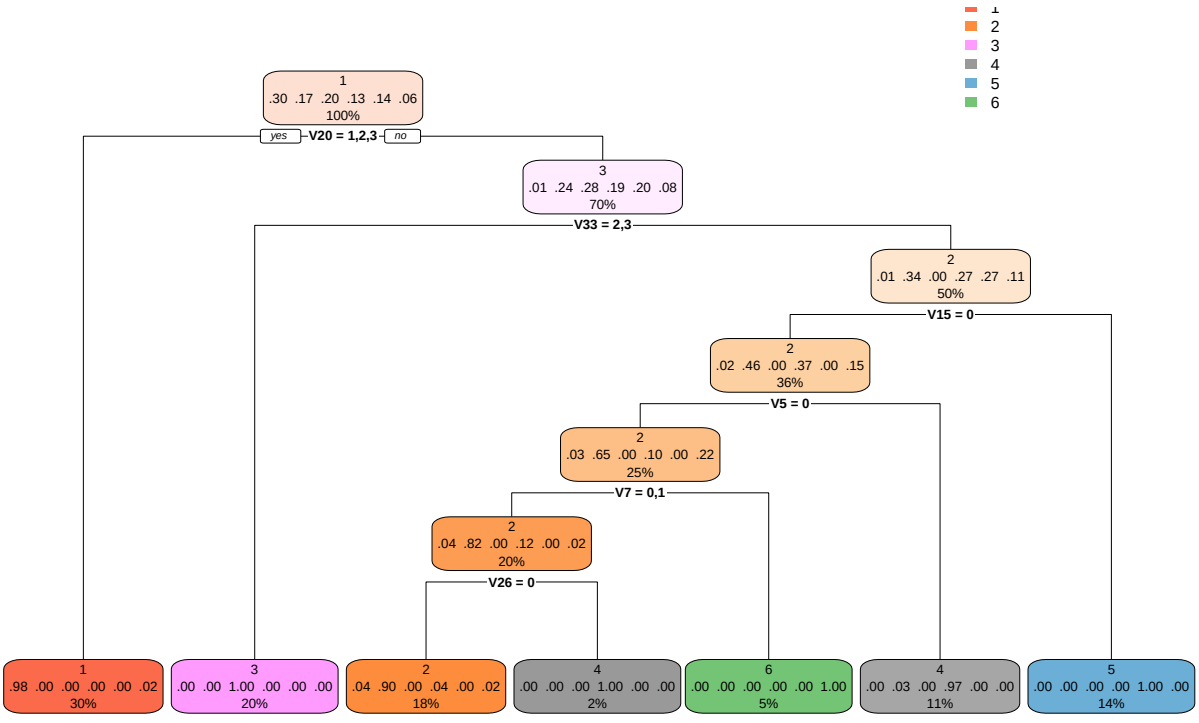


Figure 3: The resulting classification tree pruned using cross validation.

Model 2: Gradient Boosted Trees

The motivation behind this second model is to circumvent a major disadvantage of decision trees, which is high variance [1] (Section 9.2.4). Boosting decision trees can help to improve their accuracy [1] (Section 10.7), although at the price of speed and interpretability.

We use the method of gradient boosting that is implemented in the `R gbm` package. For a k class classification problem, we implement Algorithm 10.4 [1] by inducing a classification tree for each class where they are related through Equation (10.21). So what we get is no longer a simple decision tree but a "distribution" that tells us what is the probability that this data point of interest belongs to each of the class labels. We then assign the class label of the data point to the mode of this distribution. The loss function implemented in the `gbm` function for a binary classification tree is the binomial deviance. The method of gradient boosting works by updating the value of $\hat{f}(x)$ at each iteration step by adding the estimated value of the leaf in the fitted regression tree that the observation with predictors x ends up in. To estimate the value of a leaf node in this process is not a simple task, and as such, the `gbm` function uses a one-step Newton-Raphson method in order to numerically approximate this estimate. See Ridgeway [3] for full details of what the `gbm` package implements.

There are two important tuning parameters in gradient boosting, namely the number of iterations M and the size of the regression tree J_m (number of terminal nodes) at each iteration step. Using the strategy suggested by [1] (Section 10.11) and also after discussion with our lecturer, we decide to restrict all trees to be the same size at each iteration step, i.e. $J_m = J \ \forall \ m$. In particular, we investigate two values of J_m, namely $J_m = 2$ and $J_m = 4$, and find the optimal number of iterations M by cross validation. Controlling the number of iterations is to prevent the model from overfitting the training data and ends up memorising instead of learning from it. The `gbm` function has a built-in capability of doing cross validation on the training data and outputs the optimal number of iterations. To illustrate, Figure 4 shows that the CV errors start to climb up after a certain number of iterations for the model that is trained using stumps (i.e. $J_m = 2$).

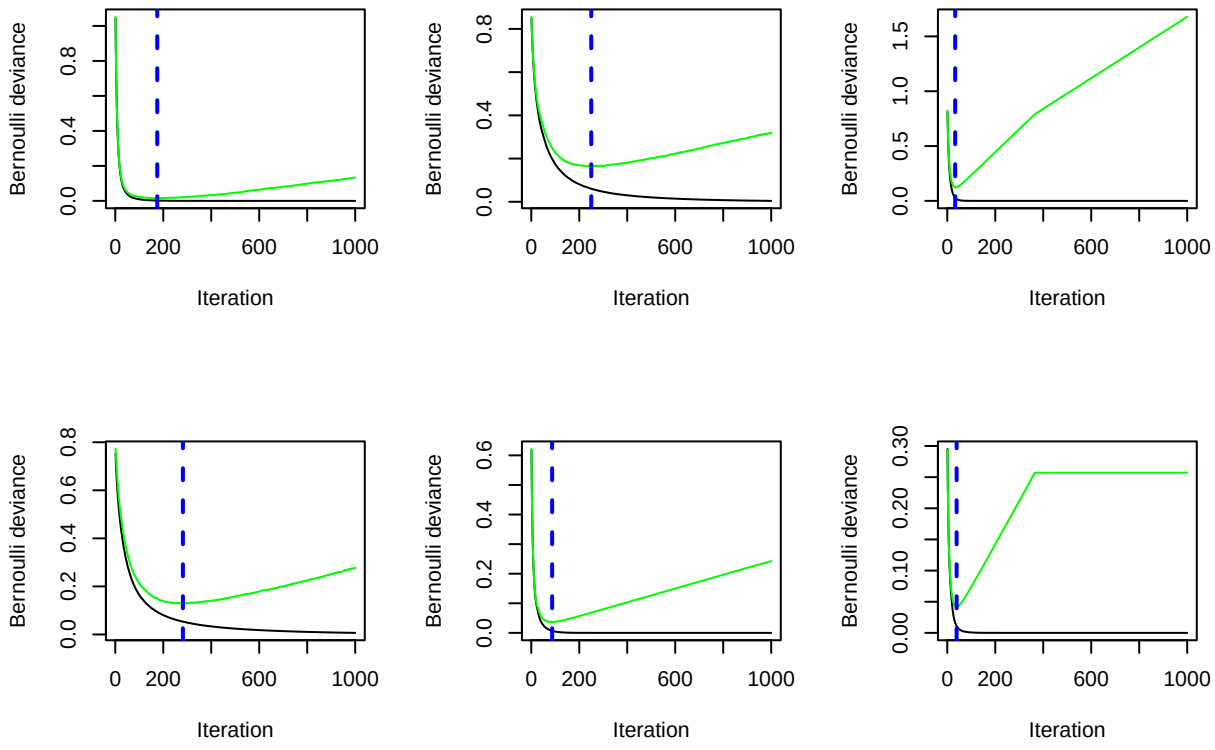


Figure 4: Number of iterations against cv error (green) and training error (black) for each class label. Top three (left to right): class 1, class 2, class 3; bottom three (left to right): class 4, class 5, class 6. Dotted line is the optimal number of iterations.

After training the two submodels with their respective optimal M, we compute the test errors with the held-out test set and obtain

- $J_m = 2$: 0.028
- $J_m = 4$: 0.056

The model that is trained using stumps returns a smaller test error and hence is our chosen gradient boosted tree model. We can also see that the test errors for both model 1 and 2 do not differ that much from each other, indicating that they are actually quite competitive.

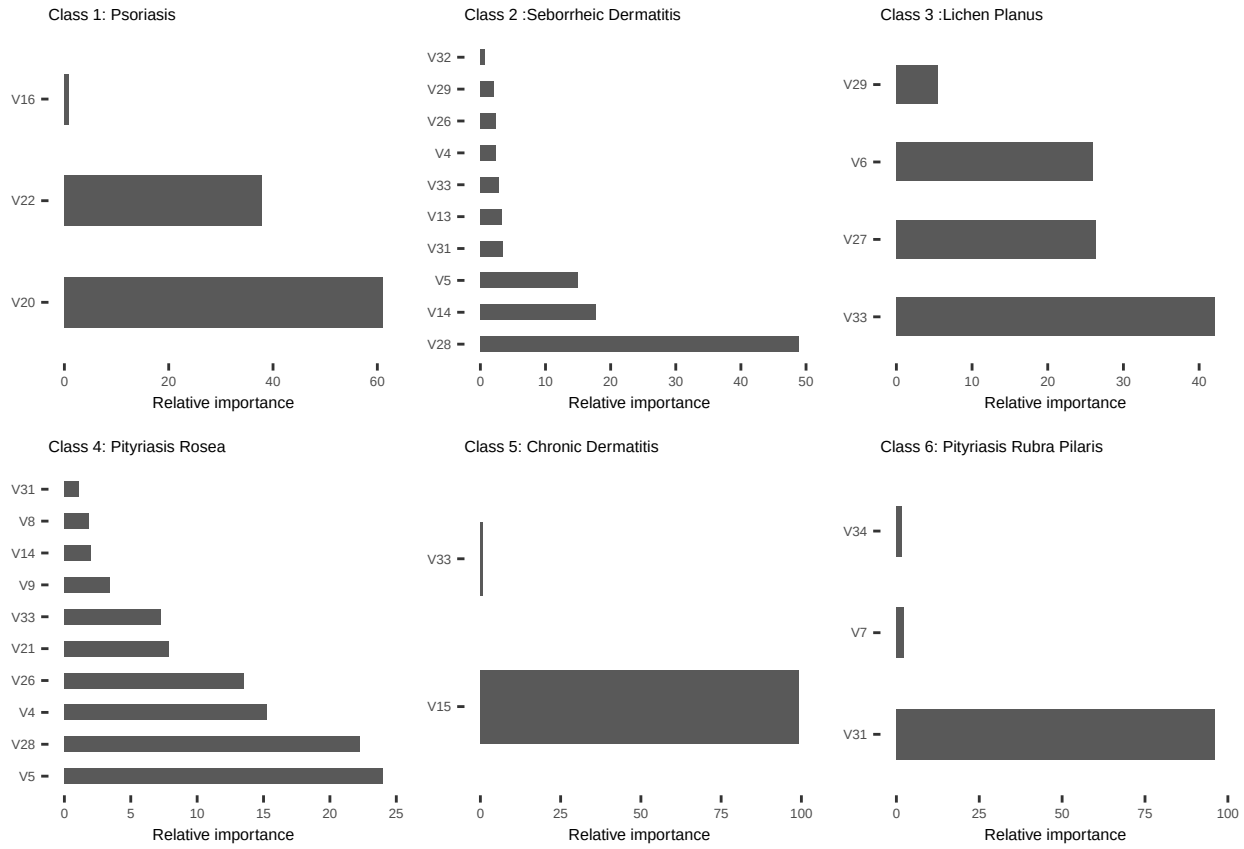


Figure 5: Predictor variable importances for each of the 6 classes.

As mentioned earlier, gradient boosting helps to enhance the accuracy of decision trees but it makes interpretability alot more difficult. Hence, we make up for this by looking at the relative importance of the predictors, i.e. the contribution of each input variable in predicting the response. Using [1] (Section 10.13) as our main reference (as well as [3] (Section 2.4)), at each internal node t, one of the input variables is chosen to split the region into two sub-regions according to criteria Equation (9.13) in [1].Then we can look at the estimated improvement in the squared error over that of a constant fit for the entire region. The squared relative importance of a particular input variable is then the sum of these squared error improvements across all the internal nodes for which it was chosen as the splitting variable. Then by the boosted tree model, which is a sum of trees, we can average out these squared error improvements and obtain an average importance measure for each input variable. Finally, we scale them on an interval of $[0, 100]$ to facilitate comparison. Fortunately for us, this algorithm is implemented in the `gbm` package and we can retrieve this information by calling the `summary` function on our model.

Figure 5 shows the relative variables importance for each class. With these relative importance graphs, the clinicians can exercise their own judgements and decide which tests they think they can omit across all 6 classes. For example, some histopathological tests that do not have high importance across all 6 classes may be omitted since they cost more and take more time to conduct. Further, we provide also partial dependence plots that show the dependence of the prediction on the relatively important predictor variables. These plots aim to help clinicians better understand and explain to their patients the mechanisms behind the model's predictions. These plots are available in the appendix.

CONCLUSION AND DISCUSSION

While we do not favour one model over the other, a major advantage of model 1 is its simplicity and interpretability. It should be even easy for non-mathematicians, like clinicians, to understand it. Also, the number of tests that clinicians need to conduct is greatly reduced, which is ideal for those working in medical institutions that have less access to medical training and testing capabilities. Model 2, on the other hand, is not easy to understand and visualize. It will seem like a "black-box" solution for the clinicians. But its prediction accuracy is higher. We also need to bear in mind that these two models are trained using this specific set of data and we do not have any information if it is collected from just one (or a few) or many sources. This has an impact on how well our models can generalize. However, all in all, our motivation is to help clinicians streamline their workflow and hopefully result in better patient outcomes.

An area of improvement for the simple classification tree is to do bagging to reduce its variance. Another model improvement for Model 2 is to introduce a loss-matrix that penalizes incorrect classification. For this, we need to go back to the clinicians to obtain information about the varying severity of the different classes of erythemato squamous disease. We also want to point out that it is possible to do shrinkage in gradient boosting but it is beyond the scope of this report.

From a data collection point of view, future researchers can consider collecting the duration of symptoms onset. Perhaps one can conduct a longitudinal study to follow the patients over a period of time to see how the symptoms evolve since one of the clinical observations is that the disease may show the features of another disease at the beginning stage and may have the characteristic features at later stages.

REFERENCES

1. *Hastie, T., Tibshirani and R., Friedman, J.*. 2017. The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Springer Series in Statistics). Springer.

2. *Therneau, T. M. and Atkinson, E. J.*. 2019. An Introduction to Recursive Partitioning Using the RPART Routines. https://cran.r-project.org/web/packages/rpart/index.html (retrieved 28 December 2020).

3. *Ridgeway, G.*. 2020. Generalized Boosted Models: A guide to the gbm package. https://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf (retrieved 28 December 2020).

APPENDIX

Predictor variables list

Clinical Predictors (take values 0, 1, 2, 3 unless otherwise stated)

V1: erythema

V2: scaling

V3: definite borders

V4: itching

V5: koebner phenomenon

V6: polygonal papules

V7: follicular papules

V8: oral mucosal involvement

V9: knee and elbow involvement

V10: scalp involvement

V11: family history (0 or 1)

V34: Age (linear)

Histopathological Predictors (take values 0, 1, 2, 3)

V12: melanin incontinence

V13: eosinophils in the infiltrate

V14: PNL infiltrate

V15: fibrosis of the papillary dermis

V16: exocytosis

V17: acanthosis

V18: hyperkeratosis

V19: parakeratosis

V20: clubbing of the rete ridges

V21: elongation of the rete ridges

V22: thinning of the suprapapillary epidermis

V23: spongiform pustule

V24: munro microabcess

V25: focal hypergranulosis

V26: disappearance of the granular layer

V27: vacuolisation and damage of basal layer

V28: spongiosis

V29: saw-tooth appearance of retes

V30: follicular horn plug

V31: perifollicular parakeratosis

V32: inflammatory monoluclear inflitrate

V33: band-like infiltrate

Partial Dependence plots

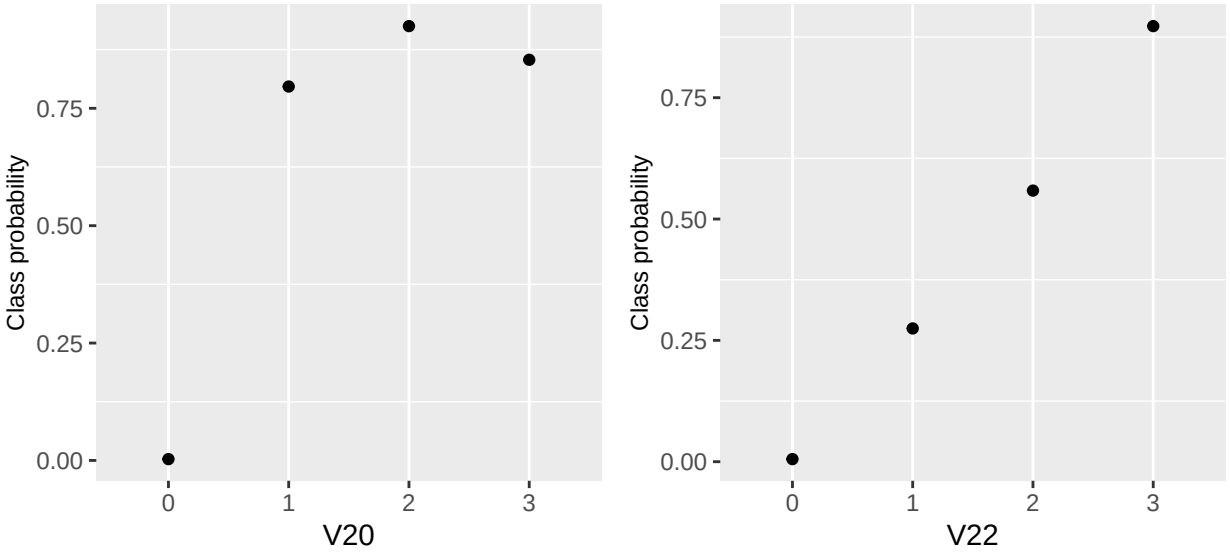


Figure 6: Class 1: Partial dependence plots for the leading two predictor variables.

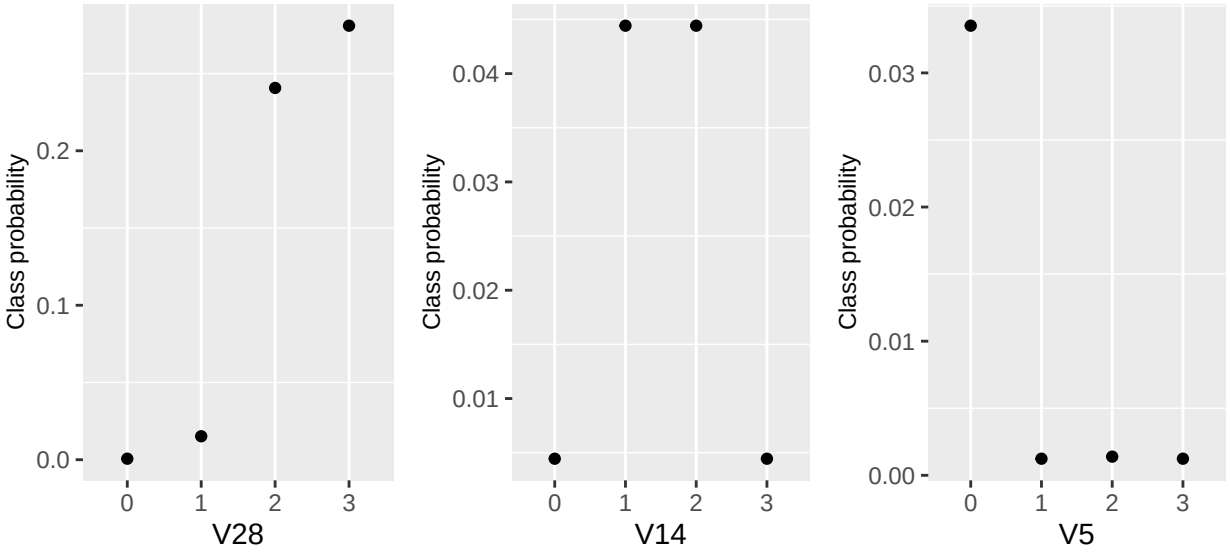


Figure 7: Class 2: Partial dependence plots for the leading three predictor variables.

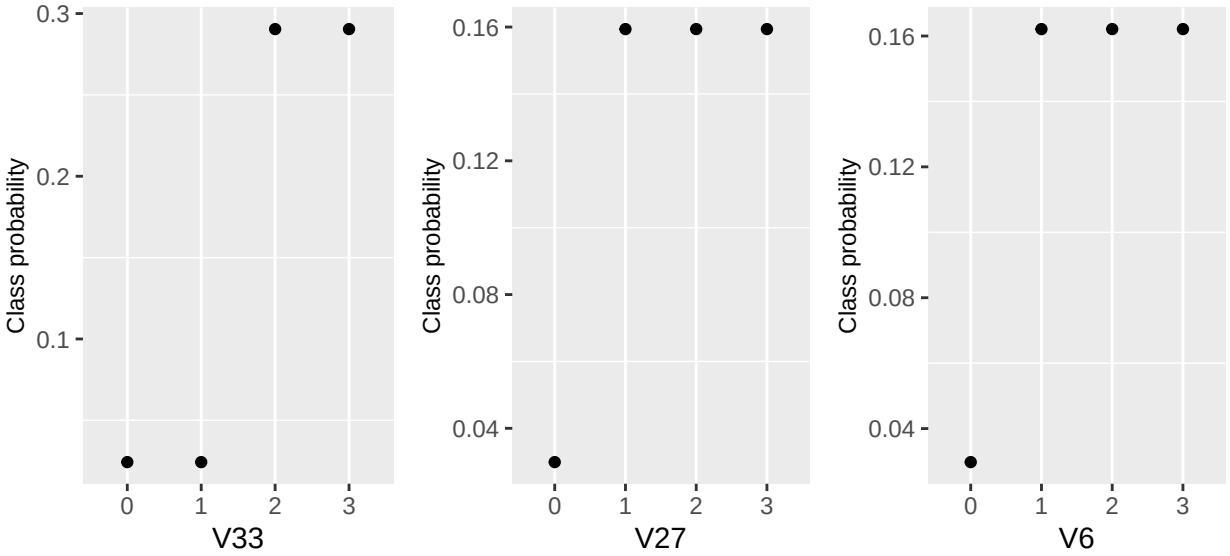


Figure 8: Class 3: Partial dependence plots for the leading three predictor variables.

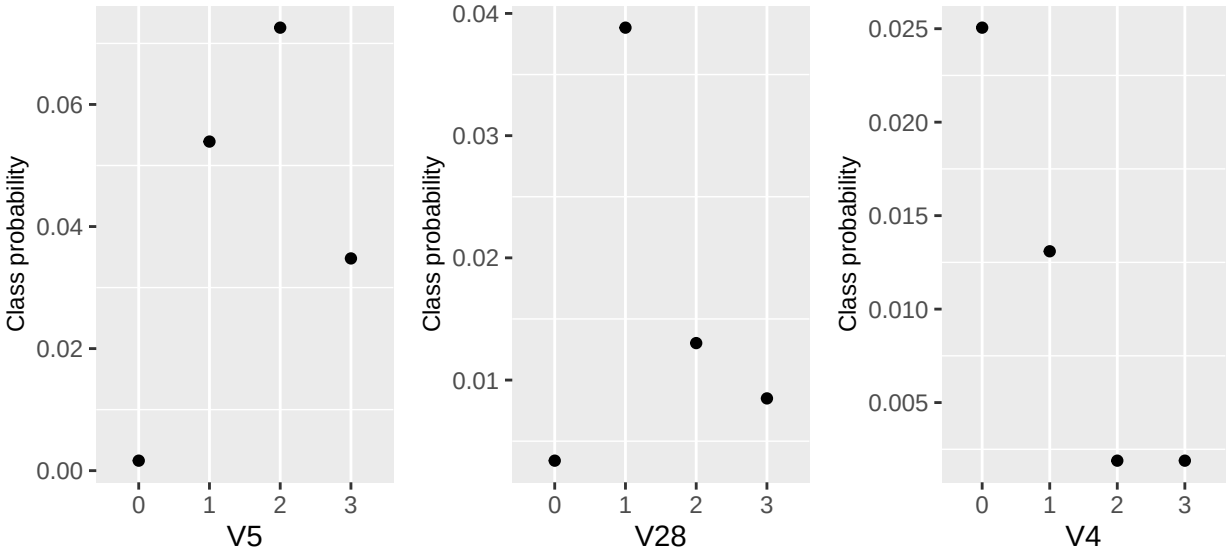


Figure 9: Class 4: Partial dependence plots for the leading three predictor variables.

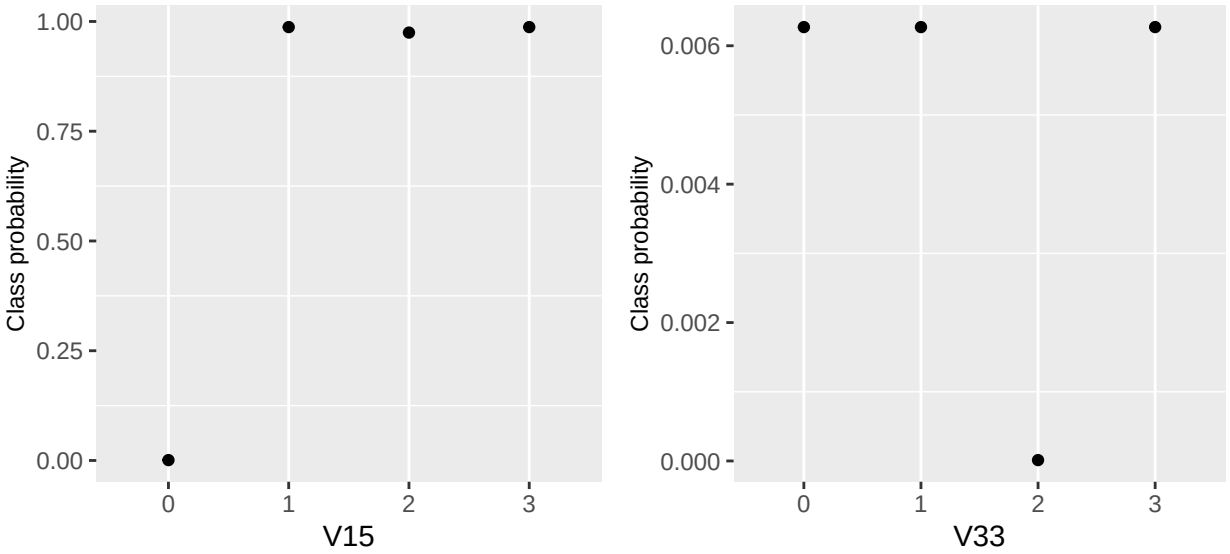


Figure 10: Class 5: Partial dependence plots for the leading two predictor variables.

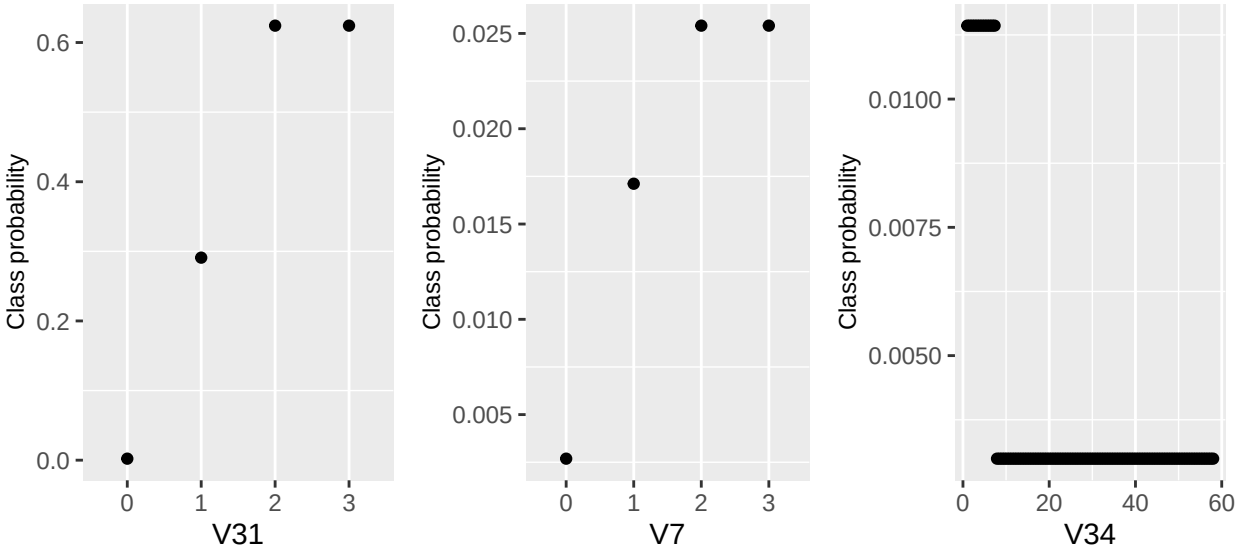


Figure 11: Class 6: Partial dependence plots for the leading three predictor variables.

Code chunks

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE)
library(tidyverse)
library(ggplot2)
library(knitr)
library(ggcorrplot)
library(e1071)
library(gridExtra)
library(rpart)
library(rpart.plot)
library(gbm)
#####################################################################
# read data
#####################################################################
data <- read.table("dermatology.data", sep = ",") %>%
  rename(class = V35) %>%
  select(class, everything()) %>%# remove missing values. 8 of them.
  filter(V34 != "?") %>%
  mutate(ID = row_number(),
         V34 = as.integer(V34))
#####################################################################
# preprocessing of data
# Split into training set and test set
#####################################################################
set.seed(790327)
# Split into training set and test set
raw_train <- data %>%
  slice_sample(prop = 0.8, replace = FALSE) #%>%
  # change all predictor to factors except for age
  #mutate_at(vars(matches("V")), as.factor) %>%
  #mutate(V34 = as.integer(V34))

raw_test <- data %>%
  anti_join(., raw_train, by = "ID") %>%
  mutate_at(vars(matches("V")), as.factor) %>%
  mutate(V34 = as.integer(V34)) %>%
  select(-ID)
#####################################################################
# exploratory analysis
#####################################################################
# table of class names
class.names <- tibble(Class = seq(1,6),
                      Name = c("Psoriasis", "Seboreic Dermatitis ",
                               "Lichen Planus", "Pityriasis Rosea",
                               "Chronic Dermatitis", "Pityriasis Rubra Pilaris"),
                      `No. of instances` = c(112, 61, 72, 49, 52, 20)) %>%
  tableGrob(theme=ttheme_minimal(base_size = 10), rows = NULL)

# distribution of classes
empi.class <- ggplot(data = data) +
  geom_bar(aes(x = class, y = ..prop..), width = 0.5) +
  theme(plot.background = element_blank(),
        plot.title = element_text(size=10),
        axis.title = element_text(size=8)) +
  labs(title = "Empirical distribution of classes",
```

```
       y = "proportion")+
  scale_x_continuous(breaks = seq(1,6))

# distribution of combined symptoms score and
#   by class
symp.dist <- raw_train %>%
  rowwise() %>%
  mutate(sym.score = sum(c_across(cols = V1:V33))) %>%
  ungroup() %>%
  mutate(V34 = as.integer(V34),
         sym.score = as.numeric(sym.score),
         class = as.factor(class)) %>%
  group_by(class, sym.score) %>%
  summarise(counts = n()) %>%
  #mutate(class = as.factor(class)) %>%
  ggplot() +
  geom_col(aes(fill = class, x = sym.score, y = counts)) +
  theme(plot.background = element_blank(),
        plot.title = element_text(size=10),
        axis.title = element_text(size=8),
        legend.key.size = unit(0.5, "cm"),
        legend.key.width = unit(0.5,"cm")) +
  labs(title = "Empirical distribution of total symptoms score",
       x = "total symptoms score")
# correlation matrix
# extract row and col nos of correlation > 0.7
# extract the values of w for which cor > 0.7
corr.mat <- cor(raw_train[2:35], method = "pearson")
sig.corr <- which(corr.mat >= 0.9 & corr.mat != 1, arr.ind = TRUE)
z <- corr.mat[corr.mat >= 0.9 & corr.mat != 1]
cb <- cbind(sig.corr,z)

# create dataframe that contains var name and cor values
df <- data.frame(cbind(unlist(cb),
    rownames(corr.mat)[sig.corr[,1]],
    colnames(corr.mat)[sig.corr[,2]])) %>%
  rename(cor.val = z, rowname = V4, colname = V5)
rownames(df) <- NULL
df <- arrange(df, rowname)

corr.mat.plot <- ggcorrplot(corr.mat, type = "upper", tl.cex = 8,
                            ggtheme = theme_classic(),
                            show.legend = FALSE,
                            title = "Correlogram")
grid.arrange(class.names, empi.class, corr.mat.plot, nrow = 2)
corr.mat.plot
#################################################################
#  function that splits data into n folds
#  returns a nested dataframe with training set and validation
#  set for each fold
#################################################################
cv_fold <- function(data, n_fold){
    # fold_id denotes in which fold the observation
    # belongs to the test set
    data <- mutate(data, fold_id = rep_len(1:n_fold, length.out = n()))
    # Two functions to split data into train and test sets
    cv_train <- function(fold, data){
```

```
        filter(data, fold_id != fold) %>%
            select(- fold_id)
    }
    cv_test <- function(fold, data){
        filter(data, fold_id == fold) %>%
            select(- fold_id)
    }
    # Folding
    tibble(fold = 1:n_fold) %>%
        mutate(train = map(fold, ~cv_train(.x, data)),
               test = map(fold, ~cv_test(.x, data)),
               fold = as.factor(fold))
}
#################################################################
# split data into 5 folds and 10 folds
# try out different fold numbers
#################################################################
n_fold.list <- list(5,10)

# train set for rpart.
train.rpart <- raw_train %>%
  # change all predictor to factors except for age
  mutate_at(vars(matches("V")), as.factor) %>%
  mutate(V34 = as.integer(V34),
         class = as.factor(class)) %>%
  select(-ID)

cv.df.list <- map(n_fold.list, ~cv_fold(train.rpart, n_fold = .x))
cv5 <- cv.df.list[[1]]
cv10 <- cv.df.list[[2]]
#################################################################
#  Construct a vector of alpha values
#################################################################
alpha.vec <- seq(0, 0.2, length.out = 30 )
#################################################################
#  Grow a large tree for each of the training sets
#  Fold = 5 and Fold = 10
#################################################################
lrg.trees <- map(cv.df.list, ~.x %>%
  mutate(lrg_tree = map(train, ~rpart(formula = class ~ ., data = .x,
                                      method = "class",
                                      control = rpart.control(minsplit = 10, cp = 0)))))
lrg.trees.5 <- lrg.trees[[1]]
lrg.trees.10 <- lrg.trees[[2]]
#################################################################
# For fold= 5 and fold =10,
#   prune each of the large trees using the different
#   alpha values.
# Compute train error for every combination of tree and alpha
#################################################################
trees.df <- lrg.trees.10 %>%
  crossing(alpha = alpha.vec) %>%
  mutate(prune_trees = map2(lrg_tree, alpha, ~prune(tree = .x, cp = .y)),
         predicted = map2(prune_trees, test, ~predict(.x, .y, type = "class")),
         actual = map(test, ~(model.frame(formula = class ~ ., .x) %>%
                                model.extract("response"))),
         train.error = map2_dbl(predicted, actual, ~mean((.x != .y))))
```

```r
#rpart.plot(trees.df$prune_trees[[1]])
options(dplyr.summarise.inform = FALSE)
################################################################
# Pull out optimal alpha that give low CV error
################################################################
# cve = mean training error
best_alpha <- trees.df %>%
    group_by(alpha) %>%
    summarise(cve = mean(train.error))%>%
    top_n(1, -cve) %>%
    pull(alpha)
################################################################
# Plot CV error for every alpha
################################################################
trees.df %>%
  group_by(alpha) %>%
    summarise(`CV Error` = mean(train.error), se = sd(train.error)/sqrt(10)) %>%
    ggplot(aes(x = alpha, y = `CV Error`)) +
    geom_point() +
    geom_line(aes(group = alpha)) +
    geom_errorbar(aes(ymin = `CV Error` - se, ymax = `CV Error` + se), width = 0.002, color = "grey")
################################################################
#  Construct trees on whole train set with chosen alpha
################################################################
# alpha choice by 1-std-error bar rule
alpha.choice <- round(alpha.vec[4], 4)

# Grow a large tree for the test set using the same control settings during cross validation
lrg.tree.test <- rpart(formula = class ~ ., data = train.rpart,
                       control = rpart.control(minsplit = 10, cp = 0))

# Prune the large test tree with chosen alphas
best_tree <- prune(tree = lrg.tree.test, cp = alpha.choice)
# This is how the best tree looks like
rpart.plot(best_tree)
################################################################
#  compute test error for best alpha
################################################################
# dataframe for test in rpart
test.rpart <- raw_test %>%
  mutate(class = as.factor(class))

test.tree <- tibble(predicted = predict(best_tree, test.rpart[,2:35], type = "class"),
                        actual = model.frame(formula = class ~ ., test.rpart) %>%
                          model.extract("response")) %>%
  mutate(miss = map2_dbl(predicted, actual, ~.x != .y)) %>%
  summarize(miss.error = sum(miss) / nrow(test.rpart)) %>%
  mutate(miss.error = round(miss.error, 3))# fraction of missclassifed
                                                    #  points.
################################################################
# Prepare data for gradient boosting
################################################################
# For k=1,...,6, let y_ik be 1 if obs i is in class k and
#   0 otherwise. We get 6 training sets, one for each class.
classnr <- seq(1, 6)

# train set for gbm. gbm call does not accept class as factor
```

```
train.gbm <- raw_train %>%
  # change all predictor to factors except for age
  mutate_at(vars(matches("V")), as.factor) %>%
  mutate(V34 = as.integer(V34)) %>%
  select(-ID)

# split gbm training set into 6 sets, one for every class
train.gbm1 <- map(classnr, ~mutate(train.gbm,
                                   class = if_else(class == .x, 1, 0)))
################################################################
# Train the model for each class as a binary classification
#   problem.
# Outcome is 6 coupled p_k(x)
# Tuning para: n.trees = nr of iterations
#              shrinkage para = learning rate
#              bag.fraction = subsampling (stoch grad descent)
#              cv.folds = number of cv folds
#              interaction.depth = J_m
################################################################
# train the model with J_m = 2 (a stump)
gbm1 <- map(train.gbm1, ~gbm(class ~ . , data = .x,
            distribution = "bernoulli",
            n.trees = 1000,
            interaction.depth = 1,
            shrinkage = 0.1,
            bag.fraction = 1,
            cv.folds = 10))

# train the model with J_m = 4 (4 terminal nodes)
gbm2 <- map(train.gbm1, ~gbm(class ~ ., data = .x,
            distribution = "bernoulli",
            n.trees = 1000,
            interaction.depth = 3,
            shrinkage = 0.1,
            bag.fraction = 1,
            cv.folds = 10))
################################################################
# plot M against CV error and training error
################################################################
# shows how too high a M can cause CV error to climb up again,
#   indicating overfitting
# enough to illustrate with J=2.
par(mfrow = c(2, 3))
best.iter1 <- map(gbm1, ~gbm.perf(.x, method = "cv"))
################################################################
# compute optimal M for the case of J=4
################################################################
best.iter2 <- map(gbm2, ~gbm.perf(.x, method = "cv"))
################################################################
# function that makes prediction, i.e. this is the machine
# output: a dataframe that contains
#     1) predicted class label for each test datapoint,
#     2) class assignment probabilities for each test datapoint
#     3) actual class labels of each test datapoint
# Paras: test.df = a new set of data points
#        test.resp = the class labels of the new data points
#        mymodel = the gbm model that was trained
```

```
#        opt.M = list(the optimal no. of iterations for each class)
#################################################################
gbm.machine <- function (test.df, mymodel, opt.M) {
  # no. of classes
  classnr <- seq(1, 6)
  cols.names <- map_chr(classnr, ~paste("p", .x))
  # each class has its optimal no. of iterations learnt from
  #   training.
  map2_dfc(classnr, opt.M, ~tibble(p = predict.gbm(mymodel[[.x]], newdata = test.df,
                         n.trees = .y,
                         type = "response"))) %>%
    # map(classnr, ~tibble(p = predict.gbm(mymodel[[.x]], newdata = test.df,
    #                       type = "response"))) %>%
    # map2_dfc(., classnr, ~.x %>%
    #         mutate(class = if_else(p > 0.5, .y, as.integer(0)))) %>%
     mutate(tot.prob.raw = rowSums(.[1:6]),
           across(starts_with("p"), function(x) {x/tot.prob.raw})) %>%
     rename_with(., function(x){x <- cols.names}, starts_with("p...")) %>%
    mutate(class.pred = max.col(.[,1:6]),
           actual = model.frame(formula = class ~., test.df) %>%
             model.extract("response")) %>%
    mutate(shannon = pmap_dbl(list(`p 1`, `p 2`, `p 3`,
                                   `p 4`, `p 5`, `p 6`), lift_vd(function(x) {-sum(x*log(x))})))
}
#################################################################
# test the machine with held-out data (raw_test dataframe)
#################################################################
stump <- gbm.machine(test.df = raw_test, mymodel = gbm1, opt.M = best.iter1)
J4 <- gbm.machine(test.df = raw_test, mymodel = gbm2, opt.M = best.iter2)
#################################################################
# compute test error
#################################################################
test.error1 <- stump %>%
  mutate(miss = map2_dbl(class.pred, actual, ~.x!=.y)) %>%
  summarize(error = round(sum(miss)/nrow(stump),3))
test.error2 <- J4 %>%
  mutate(miss = map2_dbl(class.pred, actual, ~.x!=.y)) %>%
  summarize(error = round(sum(miss)/nrow(J4),3))
#################################################################
# plot distribution of shannon entropy of the training set
#################################################################
shannon.df <- gbm.machine(test.df = raw_train, mymodel = gbm1, opt.M = best.iter1)

ggplot(shannon.df, aes(x = shannon)) +
  geom_histogram(aes(y = ..count../sum(..count..)),binwidth = 0.05) +
  labs(x = "Shannon entropy (in nat)",
       y = "frequency")
#################################################################
# compute relative importance dataframes for all 6 classes
#################################################################
rel.impt <- map2(gbm1, best.iter1, ~summary(.x, .y)) %>%
  map(., ~.x %>%
        filter(rel.inf >0.5))
#################################################################
# relative importance plots
#################################################################
rel.impt.plot <- map2(rel.impt, c("1: Psoriasis", "2 :Seborrheic Dermatitis ",
```

```
                                "3 :Lichen Planus", "4: Pityriasis Rosea",
                                "5: Chronic Dermatitis", "6: Pityriasis Rubra Pilaris"),
     ~ggplot(.x, aes(x = rel.inf, y = reorder(var, -rel.inf))) +
  geom_col(width = 0.5) +
  theme(panel.background  = element_blank(),
        axis.title.y = element_blank(),
        axis.title.x = element_text(size = 6),
        axis.text = element_text(size = 5),
        plot.title = element_text(size = 6)) +
  labs(x = "Relative importance",
       title = paste("Class", .y)))
grid.arrange(grobs = rel.impt.plot, nrow = 2)
################################################################
# Laimei's toy test
################################################################
test <- lrg.trees.5$lrg_tree[[1]]
obs.tab <- tibble(`obs nr` = seq(1, length(test$where)), node = test$where) %>%
  bind_cols(unnest(lrg.trees.5$train[[1]]))
rpart.plot(test)
residuals(test, type = "deviance")
################################################################
# dependence plots for class 1
################################################################
dep1 <- map(c("V20", "V22"), ~plot.gbm(gbm1[[1]], i.var = .x, n.trees = best.iter1[1],
                            type = "response",
                            return.grid = TRUE)) %>%
  map2(., c("V20", "V22"), ~ggplot(data = .x, aes(x = .x[,1], y = .x[,2])) +
        geom_point() +
        theme(panel.grid.major.y = element_blank(),
              axis.title.y = element_text(size = 9)) +
        labs(x = .y, y = "Class probability"))

grid.arrange(grobs = dep1, nrow = 1)
################################################################
# dependence plots for class 2
################################################################
dep2 <- map(c("V28", "V14", "V5"), ~plot(gbm1[[2]], i.var = .x, n.trees = best.iter1[2],
                                         type = "response",
                            return.grid = TRUE)) %>%
  map2(., c("V28", "V14", "V5"), ~ggplot(data = .x, aes(x = .x[,1], y = .x[,2])) +
        geom_point() +
        theme(panel.grid.major.y = element_blank(),
              axis.title.y = element_text(size = 9)) +
        labs(x = .y, y = "Class probability"))

grid.arrange(grobs = dep2, nrow = 1)
################################################################
# dependence plots for class 3
################################################################

dep3 <- map(c("V33", "V27", "V6"), ~plot(gbm1[[3]], i.var = .x, n.trees = best.iter1[3],
                                         type = "response",
                            return.grid = TRUE)) %>%
  map2(., c("V33", "V27", "V6"), ~ggplot(data = .x, aes(x = .x[,1], y = .x[,2])) +
        geom_point() +
        theme(panel.grid.major.y = element_blank(),
              axis.title.y = element_text(size = 9)) +
```

```
        labs(x = .y, y = "Class probability"))

grid.arrange(grobs = dep3, nrow = 1)
################################################################
# dependence plots for class 4
################################################################
dep4 <- map(c("V5", "V28", "V4"), ~plot(gbm1[[4]], i.var = .x, n.trees = best.iter1[4],
                                        type = "response",
                            return.grid = TRUE)) %>%
  map2(., c("V5", "V28", "V4"), ~ggplot(data = .x, aes(x = .x[,1], y = .x[,2])) +
        geom_point() +
        theme(panel.grid.major.y = element_blank(),
              axis.title.y = element_text(size = 9)) +
        labs(x = .y, y = "Class probability"))

grid.arrange(grobs = dep4, nrow = 1)
################################################################
# dependence plots for class 5
################################################################
dep5 <- map(c("V15", "V33"), ~plot(gbm1[[5]], i.var = .x, n.trees = best.iter1[5],
                            type = "response",
                            return.grid = TRUE)) %>%
  map2(., c("V15", "V33"), ~ggplot(data = .x, aes(x = .x[,1], y = .x[,2])) +
        geom_point() +
        theme(panel.grid.major.y = element_blank(),
              axis.title.y = element_text(size = 9)) +
        labs(x = .y, y = "Class probability"))
grid.arrange(grobs = dep5, nrow = 1)
################################################################
# dependence plots for class 6
################################################################
dep6 <- map(c("V31", "V7", "V34"), ~plot(gbm1[[6]], i.var = .x, n.trees = best.iter1[6],
                            type = "response",
                            return.grid = TRUE)) %>%
  map2(., c("V31", "V7", "V34"), ~ggplot(data = .x, aes(x = .x[,1], y = .x[,2])) +
        geom_point() +
        theme(panel.grid.major.y = element_blank(),
              axis.title.y = element_text(size = 9)) +
        labs(x = .y, y = "Class probability"))

grid.arrange(grobs = dep6, nrow = 1)
```