ST443 : Machine Learning and Data Mining

CANDIDATE NUMBERS	: - 47516
	- 42021
	- 42765
	- 43049
MARKS ALLOCATION	: - 47516 (25%)
	- 42021 (25%)
	- 42765 (25%)
	- 43049 (25%)

Problem 1.1 – Real World Data

Description of Data

For the purpose of our analysis, we have chosen the “Indian Liver Patient Dataset” extracted from UCI Machine Learning Repository. This data set was donated in 2012 and it contains records of 416 liver patients and 167 non-liver patients, collected from the North-East of Andhra Pradesh, India. Altogether, there are 11 variables – binary outcome variable (is_patient) that indicates whether a person is a liver patient or not, and 10 predictors (9 continuous, 1 categorical). The categorical variable is gender of the patient (gender), and the rest of the predictors are: age of the patient (age), Total Bilirubin (tot_bilirubin), Direct Bilirubin (direct_bilirubin), Total Proteins (tot_proteins), Albumin (albumin), Albumin-Globulin Ratio (ag_ratio), Alamine Aminotransferase (sgpt), Aspartate Aminotransferase (sgot), and Alkaline Phosphotase (alkphos).

Aims of Analysis

1. Build a classification model that is able to classify an individual into either a liver patient or a non-liver patient.
2. Obtain probabilities of susceptibility to contracting a liver disease, since in real medical diagnosis, it is pertinent to not only perform binary classification, but also gauge an individual’s probability of contracting the disease.
3. Choose a classifier that minimizes False Negative Rate (high sensitivity) the most, while minimizing the False Positive Rate (acceptable specificity) since misclassifying a sick person as healthy is far more detrimental than misclassifying a healthy person as sick.

Exploratory Analysis

We began our analysis by cleaning the data – we removed 4 rows with missing values, as they only constituted 0.69% of the sample. After setting variables to their correct types and levels, we plotted several graphs to look at the data spread and detect potential dependence between variables. Amongst our key observations are – our outcome variable is mildly imbalanced (patients to non-patients ratio of 2.5)[1], there are linear correlations between several predictors, namely ‘tot_bilirubin’ and ‘direct_bilirubin’, ‘ag_ratio’ and ‘albumin’, ‘sgpt’ and ‘sgot’, and ‘alkphos’ and ‘sgot’[2], and that apart from ‘sgpt’ and ‘gender’, other variables seem to record some mean differences between patient group and non-patient group[1 & 3]. We then proceeded to the main part of the analysis, where we discovered the significance of these mean differences across the patient and non-patient groups.

Review of Machine Learning (ML) Approaches Used in Analysis

Before any ML methods were applied to the data, we split our data using 80:20 ratio into training and testing sets respectively; We chose the ratio to ensure that our training sample is not too small, while still having sufficient sample for testing purposes, since our total sample size after missing data removal is 579. We then used the same training and testing samples (random seed was set to 1) to perform the 8 different ML approaches, so that results obtained across the different methods are not dependent on the random sample division. Since our data is slightly imbalanced, we also ensured that the proportion of patients to non-patients (outcome variable) in both training and testing sets were similar to the proportion in the full data set. For every patient in the full / training / testing data sets, there were 2.51 / 2.56 / 2.31 non-patients respectively. The performances of each approach are summarized in Table 1.

1. *Logistic Regression (LR)*[4]

Using the glm, bestglm and glmnet packages in R, we fitted 5 different models – 1 full model and 4 models with feature selection using LASSO (penalized LR) and Best Subset Selection (BSS) with Bayesian Information Criterion (BIC), Akaike Information Criterion (AIC) and 10-fold Cross Validation (CV). The summary of the full model suggests that only 4 out of 10 variables are significant at 5% significance level, namely ‘age’, ‘direct bilirubin’, ‘albumin’ and ‘sgpt’. After performing variable selection using the 4 methods above, Figure 1 shows the suggested number of features to be included (ranging from 2 to 5)[4] – methods using CV and BIC had features selected only from the 4 significant variables, while the other 2 methods included some non-significant variables. For penalized LR using LASSO, 10-fold CV was used to select the best λ using the 1-standard error rule. Regardless of the variable selection method, LR produced consistently stable predictions across both training and testing data, i.e. similar precision, accuracy and specificity.

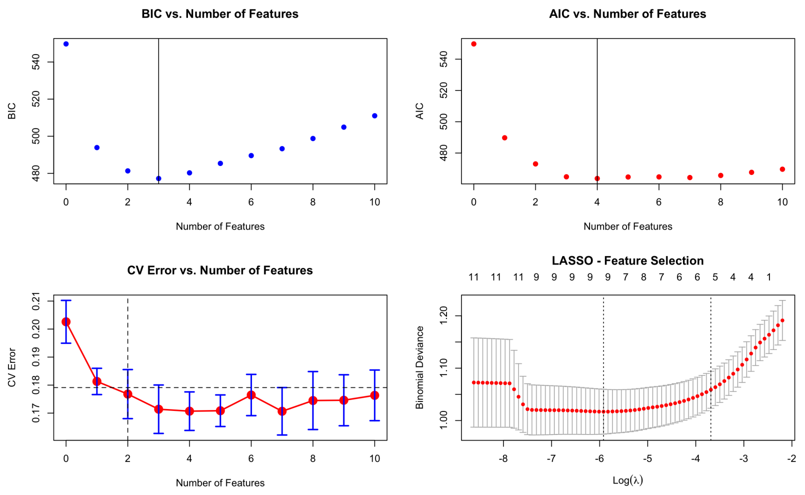


Figure 1: Feature Selection under Logistic Regression Umbrella

2. *Linear Discriminant Analysis (LDA) & Quadratic Discriminant Analysis (QDA)*[5]

The next parametric classification approach implemented was LDA (assuming covariance matrices are similar) and then QDA (assuming covariance matrices are different). The prior probabilities for both methods were 0.281 (for non-patients) and 0.719 (for patients). LDA is not the best method to use when the class distribution of the outcome variable is imbalanced because the prior probabilities are estimated from the observed counts. This resulted in most observations in the non-frequent class being classified wrongly[5] – low specificity as displayed in Table 1. QDA on the other hand was able to classify most of the minority class correctly, but recorded the worst sensitivity score amongst

[1] Refer to Appendix 1; [2] Refer to Appendix 2; [3] Refer to Appendix 3; [4] Refer to Appendix 4; [5] Refer to Appendix 5.

all approaches tried. Since our focus is more on maximizing sensitivity rather than specificity, QDA is not the best method. Changing the thresholds may improve the models, but will not improve their performance vs. other models in terms of Area Under ROC Curve (AUROC) – thus, they would still be the inferior methods, as displayed on Figure 3.

3. K-Nearest Neighbors (KNN)[6]

We applied one-hot-encoding to 'gender' (model.matrix function in R) before performing KNN on our training data set, since the KNN algorithm requires pre-processing of data in the presence of categorical variables. Then, using a for loop (while setting random seed to 1), we trained the KNN algorithm for k ranging from 1 to 100, to find the k that minimizes the misclassification error rate. This pointed to k=38[6] and so, that was the value chosen for parameter k that generated performance scores as in Table 1. KNN seems to outperform LDA and QDA, but its main drawback is that it does not tell us which predictors are important and so, our model is not sparse.

4. Random Forest (RF) and Bagging[7]

Random Forest and Bagging were both implemented with the number of predictors chosen as split candidates, 'm' set to 3 and 10 respectively. The Variable Importance Plots[7] for both approaches suggest that 'direct_bilirubin' and 'albumin' were amongst variables that registered high mean decrease of accuracies over all out-of-bag CV predictions when the variables were permuted after training, while 'age' and 'tot_proteins' were amongst the variables that when permuted, gave relatively large decreases in mean Gini-gain. Since these 2 approaches recorded really good performance as recorded in Table 1, we tried the RandomForest algorithm with all possible 'm' to see if there was any other 'm' that would outperform m=3 / m=10. From Figure 2, it is evident that indeed, RF was the best choice. RF is a strong contender against Logistic Regression, as seen in Figure 3 and Table 1. Although the model is not sparse, it still has a measure of variable importance, unlike KNN.

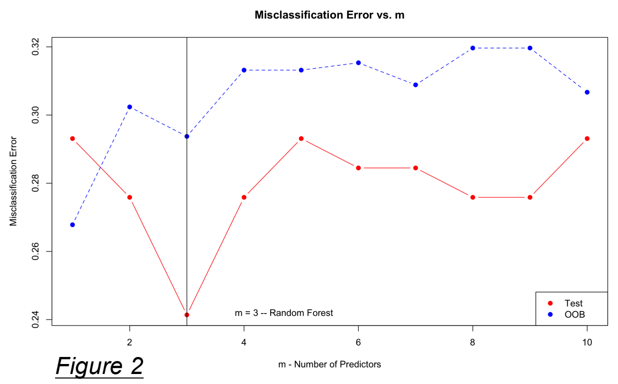


Figure 2

5. Boosting (Gradient Boosted Model)[8]

Using the 'caret' package in R, we fitted a gradient boosted model using 3 repeated 10-fold CVs to select the 3 tuning parameters, with random seed set to 1. The selected tuning parameters were: number of trees (B = 50), shrinkage parameter (λ = 0.1), and interaction depth (d = 1). This method recorded good performance scores as displayed in Table 1, but RF still outperformed this method overall. Like RF, this method does not generate a sparse model but it provides relative influence of the features[8] – the top 4 features having higher relative influence over the rest were the same as in the RF model. Although the boosting approach learnt slowly and consequently, resulted in much lower number of trees compared to RF (50 vs. 500 trees), it can be difficult to interpret the resulting model, just like RF.

6. Support Vector Machines (SVM) using Radial Kernel

We began our analysis by implementing Support Vector Classifier on our training data with one-hot-encoding performed for 'gender', just like in KNN. However, even after performing CV to select the optimum cost parameter, the model still recorded 0% specificity. We then performed SVM with radial kernel – using the 'tune' function, the cost and gamma (γ) values selected were 1 and 4 respectively. The SVM model performed extremely well on the training data, recording 95.5% accuracy, alongside 99.7% sensitivity and 84.6% specificity; However, the model did not perform as well on the test data – refer to Table 1. Increasing γ may improve the model, but there is risk of over-fitting and thus, may underperform on the test data set again. With SVM, we are optimizing directly for the decision boundary rather than using probabilities, thus in our case where identifying probabilities of susceptibility to the disease is also as important as classifying individuals into patients / non-patients, this approach may not be the best. Another drawback is that it uses all the features, and does not easily select which features are important.

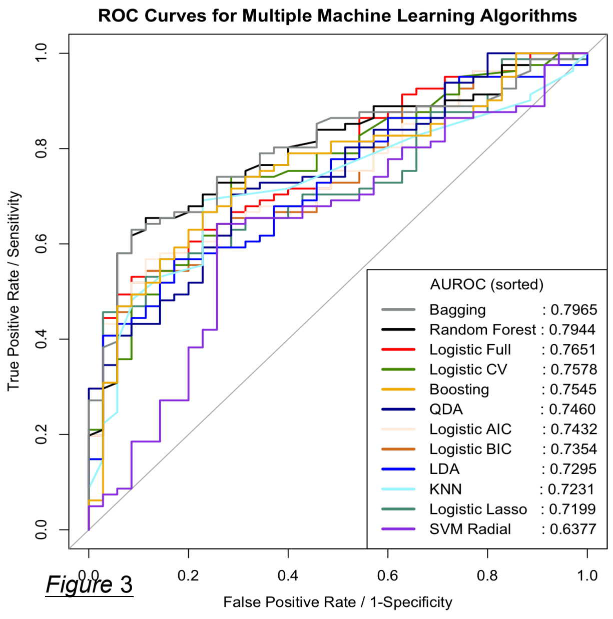


Figure 3

Final Approach / Justification / Results / Conclusion

With reference to the numerical performance metrics in Figure 3 and Table 1, apart from QDA and SVM Radial, other models seemed to record similar performances. However, since this study relates to disease detection, a large emphasis is placed on obtaining probabilities of contracting that disease (aim no. 3 of this study) – 2 individuals having probabilities of 0.51 and 0.99 of contracting a liver disease may be classified as 'patients', but the latter's condition is more severe and if resources are scarce, the latter should be treated first. All models except SVM are able to provide a measure of probability. It is also important to have an easily interpreted model, as the results from this study would be applied by medical practitioners who may not be well-versed in Statistics / Machine Learning Theory; This is where some of the best-performing methods such as Random Forest, Bagging and Boosting fell short.

[6] Refer to Appendix 6; [7] Refer to Appendix 7; [8] Refer to Appendix 8.

There are also monetary and time costs associated with carrying out medical tests and so, if a medical practitioner could diagnose a disease using only 2 tests and the results are not statistically different than performing 10 tests, sparsity in the feature space would result in large cost savings – this criterion served as the tiebreaker between Logistic Regression (with feature selection) and other methods. The correlation check at the beginning of the analysis also suggested potential multicollinearity issue, thus results from other models (with no feature selection) that assume independence between features may be affected.

Table 1 : Results from Methods Implemented and Their Performances – Darker Colour is Superior over Lighter Colour

Machine Learning Approach	**Additional Modification**	**Results and Ranks (Colour-coded) Based on Performance Measures**						
		Sensitivity (for test data set)	**Accuracy** (for test data set)	**AUROC** (for test data set)	**Specificity** (for test data set)	**No. of Features**	**Predicted Probability of Contracting Liver Disease**	**Ease of Model Interpretation** (for medicinal application)
Logistic Regression	Full Model	95.1%	73.3%	0.7651	22.9%	10	Present	Positive
	BSS (BIC)	96.3%	72.4%	0.7354	17.1%	3	Present	Positive
	BSS (AIC	96.3%	72.4%	0.7432	17.1%	4	Present	Positive
	BSS (CV)	100.0%	69.8%	0.7578	0.0%	2	Present	Positive
	Penalized - LASSO	100.0%	69.8%	0.7199	0.0%	5	Present	Positive
LDA	-	97.5%	69.0%	0.7295	2.9%	10	Present	Positive
QDA	-	38.3%	55.2%	0.7460	94.3%	10	Present	Positive
KNN	-	98.8%	73.3%	0.7231	14.3%	10	Present	Positive
Random Forest	-	90.1%	71.6%	0.7944	28.6%	10	Present	Negative
Bagging	-	88.9%	70.7%	0.7965	28.6%	10	Present	Negative
Boosting	-	90.1%	68.1%	0.7545	17.1%	10	Present	Negative
SVM Radial	-	97.5%	69.8%	0.6377	5.7%	10	Absent	Negative

These justifications point towards choosing Logistic Regression with feature selection. Amongst the 4 methods used as displayed in Figure 1, the model with BSS using CV not only had the least number of features (2 - 'direct_bilirubin', 'albumin'), but it also performed similarly well (and better in some metrics) compared to other feature selection methods. To avoid the risk of over fitting and for cost savings purposes, we decided on this model. However, this model recorded 0% specificity and 100% sensitivity using the typical 0.5 probabilistic threshold assigned during prediction – although we aim to maximize sensitivity scores to avoid patients from being falsely classified as non-patients, we do not wish to achieve this at 100% expense of specificity. There are no clear guidelines as to the acceptable False Negative Rate in liver disease detection, but based on our research[9], clinical error rates are still ranging at about 20%. Thus, based on Figure 4, we decided to move our threshold of classifying an individual as having a disease from >0.5 to >0.565, as this threshold enabled us to achieve an acceptable specificity level of 40% (previously 0%), while maintaining a sensitivity score (86.4%) that is well above typical clinical rates.[9] The equation given by this model is : **logit (Pr(is_patient=1|x)) = -0.244 + 0.806 x *direct_bilirubin* + 0.014 x *albumin***. When explaining to medical practitioners, we describe the model as – provided all else are equal, on average, the odds of being a liver disease patient increases by exp(0.806) when the reading of direct bilirubin increases by one unit / increases by exp(0.014) when albumin reading increases by one unit.

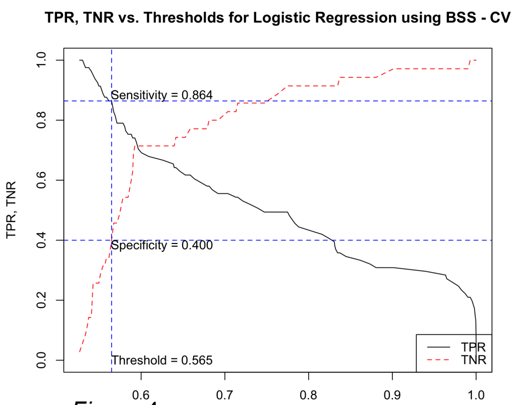


Figure 4

Upon further research, we discovered a further enhancement to our model: the Neyman-Pearson classifier (NPC)[10] - a general framework that enabled us to find a classifier with **population** type I errors (*we manipulated our data so that the type I error here is misclassifying a patient as non-patient*[10]) under α with high probability. In Figure 5, again our logistic regression model with 2 variables chosen by CV outperformed the next best option, i.e. Random Forest. The difference between the NPC model using Logistic Regression and the classical Logistic Regression is that, given the sensitivity score of 86.4% (α = 0.136), the former guarantees specificity score between 13.5% to 35% with a probability of at least 1 − 2δ **at population level**, where δ=0.05 is our tolerance rate to violation of α. Since it is crucial to have a model that works well at population level, our final model is the NPC with Logistic Regression (CV) using the 2 variables as above. The only shortfall of NPC is that it does not output any coefficients for the variables, but since it provides the predicted scores of having a disease (probability measure) and it guarantees similar performance measures at population level, it fulfills the aims of this study. NPC also works great on imbalanced data, and was partly designed to improve the generalization of results from medical studies to the population. To explain the model to medical practitioners, we may still use the classical LR model as an estimate to explain how changes in 'direct_bilirubin' and 'albumin' contribute to a person's risk of having liver disease. In conclusion, 'direct_bilirubin' and 'albumin' readings are sufficient to predict the risk of a person having liver disease and to classify them into patients / non-patients; In this regard, the NPC with Logistic Regression (CV) is the best technique to achieve the aims of this study.

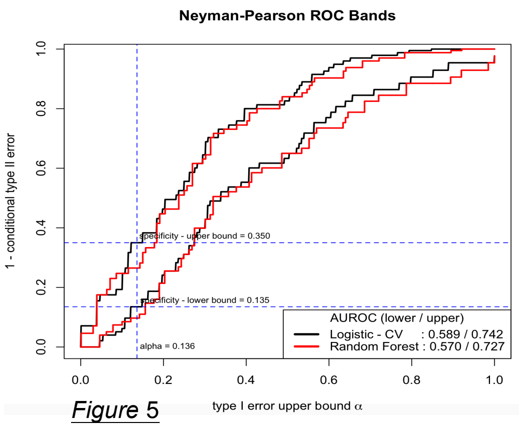
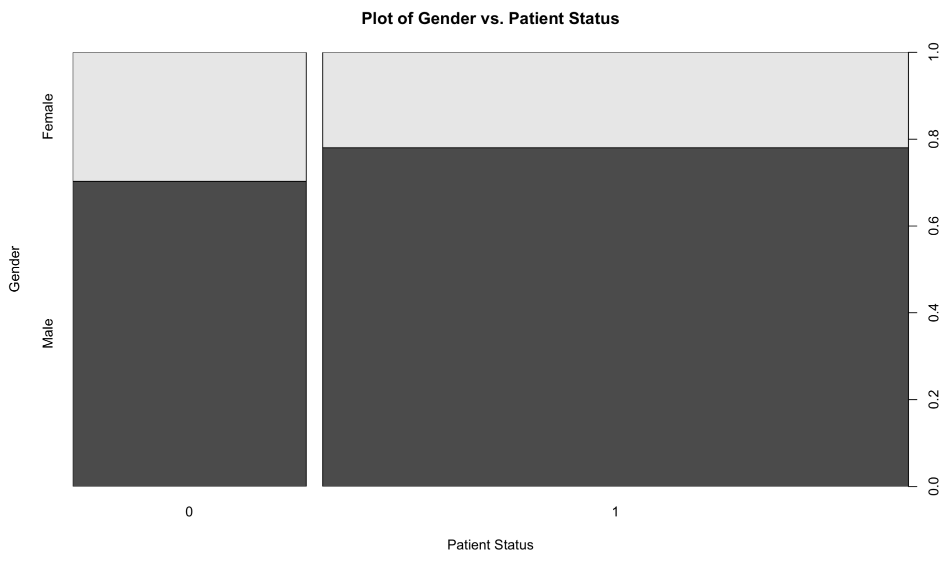


Figure 5

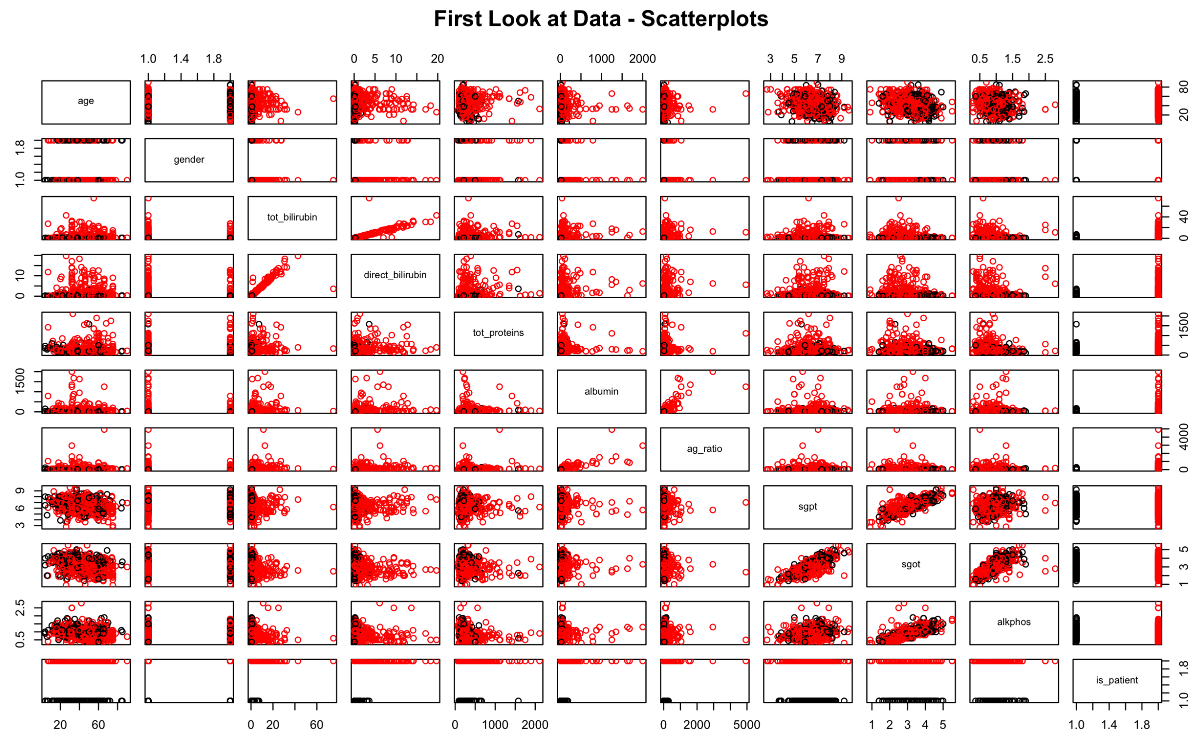
[9] Refer to Appendix 9; [10] Refer to Appendix 10.

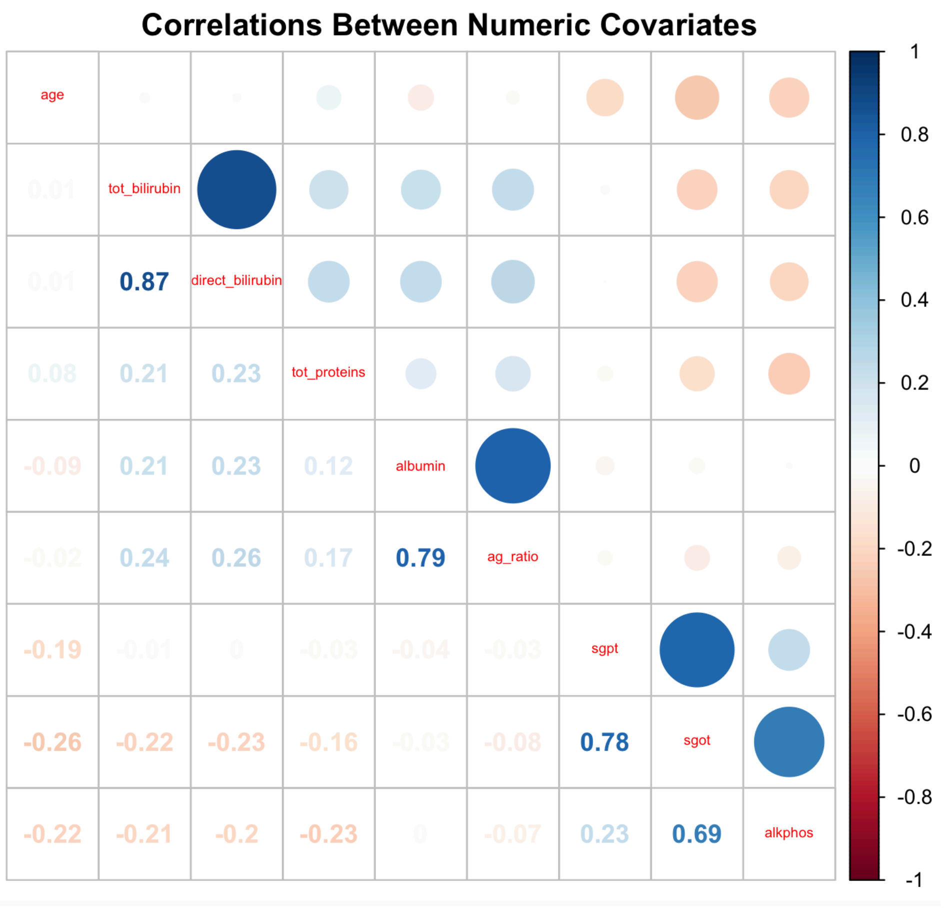
Appendix

1. In the plot below, we observe that the proportion of patients (status=1) is larger than the proportion of non-patients (status=0) – ratio as disclosed in the main part of the report. Spread of sample is also not balanced by gender – there are 441 male records and 142 female records, as plotted below. However, the spread of male and female individuals across the 2 patient statuses are still proportionate to each other, implying that gender may not be a significant predictor in this analysis.

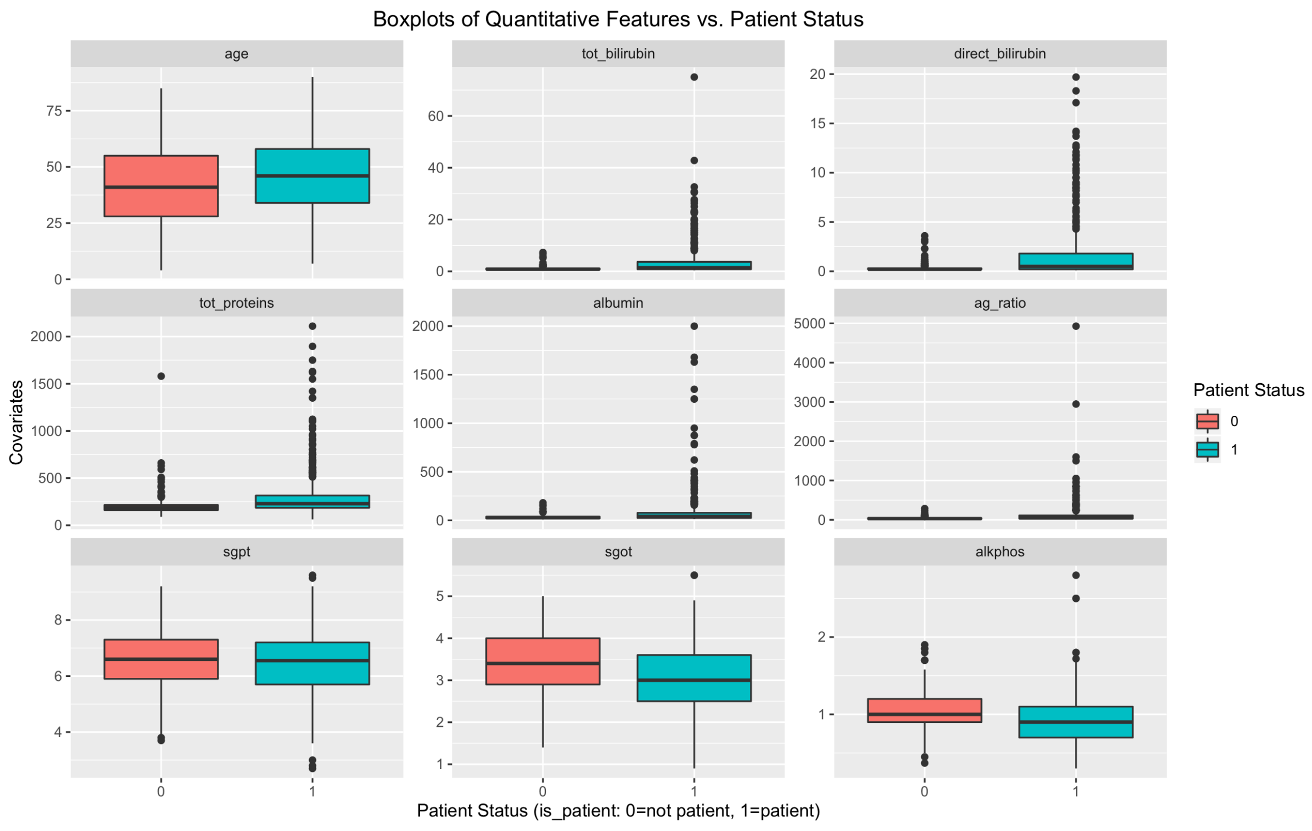


2. Below are a matrix scatterplot and a correlation plot respectively. From the first plot, we observe linear relationships as mentioned in the main part of the report. We then confirmed the correlations using the correlation plot, i.e. the next plot below.





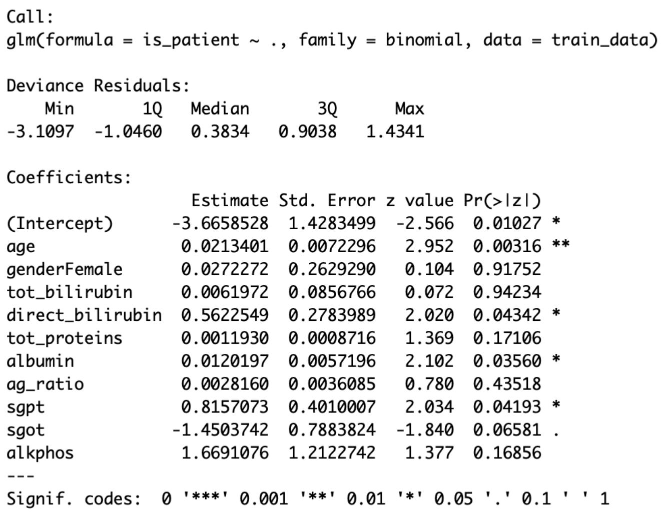
3. Below are boxplots of continuous variables vs. the outcome variable, generated using ggplot.



4. Image below shows the coefficients of variables when regressed against 'is_patient' under the full LR model. As observed below, 'age', 'direct_bilirubin', 'albumin' and 'sgpt' are the only significant variables. Below is a list of the 4 methods used to perform feature selection and the variables that they include:

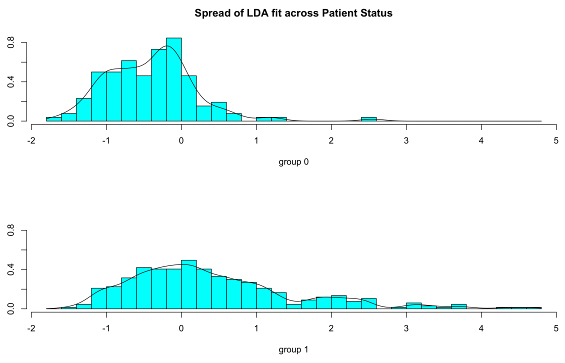
    a. Best Subset Selection using BIC: 'age', 'direct_bilirubin', 'albumin'
    b. Best Subset Selection using AIC: 'age', 'direct_bilirubin', 'tot_proteins', 'albumin'
    c. Best Subset Selection using CV: 'age', 'direct_bilirubin'
    d. Penalized Logistic Regression using LASSO: 'age', 'direct_bilirubin', 'tot_proteins', 'ag_ratio', 'alkphos'

    From above, we see that BSS using AIC and Penalized LR using LASSO both include variables that are statistically not significant, even at 10% significance level.

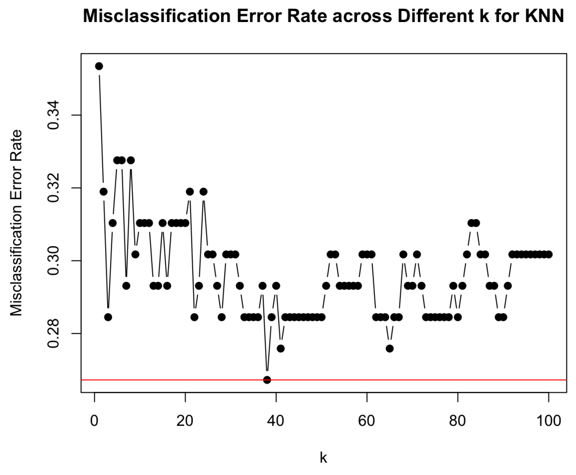


```
Call:
glm(formula = is_patient ~ ., family = binomial, data = train_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.1097  -1.0460   0.3834   0.9038   1.4341

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)     -3.6658528  1.4283499  -2.566  0.01027 *
age              0.0213401  0.0072296   2.952  0.00316 **
genderFemale     0.0272272  0.2629290   0.104  0.91752
tot_bilirubin    0.0061972  0.0856766   0.072  0.94234
direct_bilirubin 0.5622549  0.2783989   2.020  0.04342 *
tot_proteins     0.0011930  0.0008716   1.369  0.17106
albumin          0.0120197  0.0057196   2.102  0.03560 *
ag_ratio         0.0028160  0.0036085   0.780  0.43518
sgpt             0.8157073  0.4010007   2.034  0.04193 *
sgot            -1.4503742  0.7883824  -1.840  0.06581 .
alkphos          1.6691076  1.2122742   1.377  0.16856
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

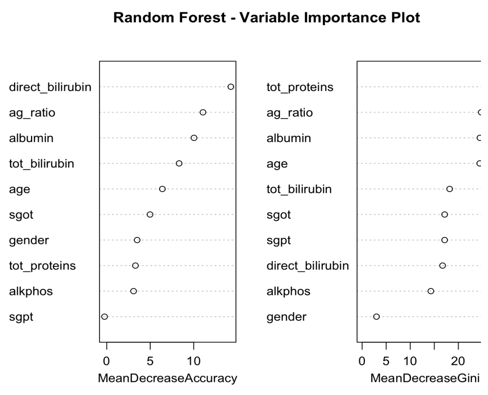
5. Image on the right shows how the LDA fit predictions across the 2 groups overlapped with each other – this coincided with the very low specificity recorded for the LDA model.

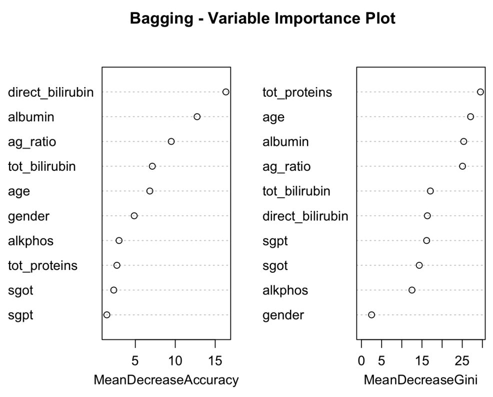


6. Image on the right shows the Misclassification Error Rate across different values of k ranging from 1 to 100, when the random seed is set to 1. According to this, k=38 gives the lowest possible error rate, thus this was the value chosen for parameter k when implementing the KNN algorithm on our training data set.



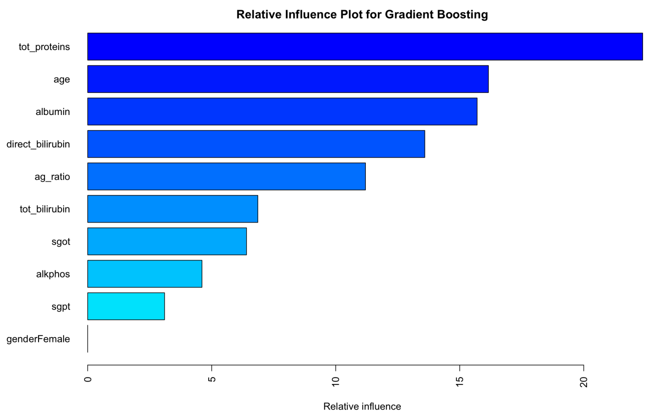
7. The 2 images on the right display the Variable Importance Plots for both Random Forest and Bagging approaches. The number of trees used in both were 500, and so, the numbers on the graphs represent the total amount that the accuracy and Gini index are decreased by





splits over a given predictor, averaged over the 500 trees. The numbers of trees were left at the default of 500 because after checking with different number of trees using CV, we did not notice any major difference to the test / OOB errors. Since having a large number of trees will not cause overfitting and it helps to reduce variance (up to a certain point, after which it plateaus), we decided to keep the number of trees at 500.

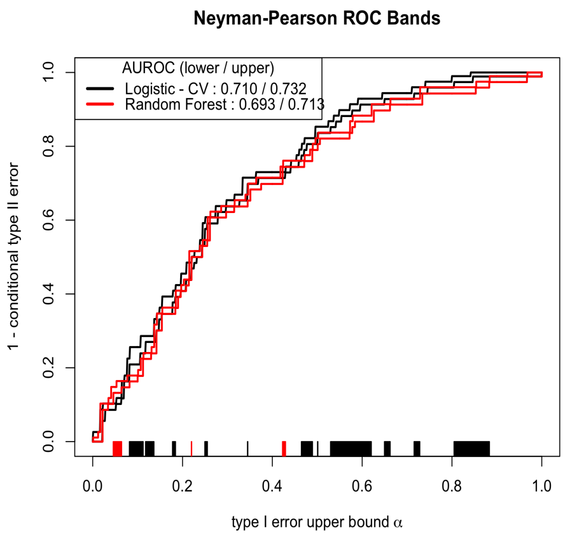
8. The image on the right shows the relative influence amongst the features used to predict 'is_patient'.



9. The 2 links below served as guidelines in choosing the maximum False Negative Rate / the upper bound for type I error i.e. α, as well as our tolerance to type II error for our model. There are no centralized guidelines on the ideal False Negative Rate / False Positive Rate for liver disease detection, but these articles suggest that in clinical practice, false negative errors made are still above 20% in general, and so in our model, we decided to set α at a level of not more than 0.15, i.e. still achieve at least 85% sensitivity / True Positive Rate.
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4731367/
- http://clinchem.aaccjnls.org/content/clinchem/23/11/2034.full.pdf

10. Below are some points on the Neyman-Pearson Classifier, a paper authored by Xin Tong (Uni of Southern California), Yang Feng (Columbia University) and Jingyi Jessica Li (Uni of California) released in 2018:

- The argument provided was that the existing ad hoc use of classification algorithms are not able to control type I errors under α with high probability. The Neyman-Pearson paradigm can address this issue, but still unable to implement it with different classification algorithms.
- This NPC method uses NP-ROC bands that enable us to choose α in a data-adaptive way and also compare different NP classifiers. It serves as a new visualization tool for classification methods under the NP paradigm.
- To use the NPC, the important class has to be coded as 0, and since patients are coded as 1 in our earlier analysis, we had to swap the labels of the 2 classes – refer to our R codes in Appendix 11.
- Within the NP-ROC space, the horizontal axis is defined as the type I error upper bound (with high probability), and the vertical axis represents (1 − conditional type II error), where we define the conditional type II error of a classifier as its type II error conditioned on training data.
- This NP-ROC band has the interpretation such that every type I error upper bound α corresponds to a vertical line segment and the achievable (1− conditional type II error) is sandwiched between the lower and upper ends of the line segment with a probability of at least 1 − 2δ.
- As can be seen on the figure to the right, on the horizontal axis, the black marks represent the α values for which the lower curve of method 1 is higher than the upper curve of method 2. Likewise, the α values for which the lower curve of method 2 is higher than the upper curve of method 1 are marked in red. (The image on the right corresponds to δ=0.5. In our main part of the report, we chose δ to be 0.05, to minimize our tolerance to changes in α. By doing that, there was no specific part of the bands where one method dominated the other one – that is why there were no marks on the horizontal axis for Figure 5 in our main report.)



- We cannot simply determine a classifier from an empirical ROC curve to control the type I error under α with high probability, whereas the NP-ROC bands provide direct information for us to make such a decision.

11. Below are the R codes used in this study:

```
##### READING AND CLEANING DATA #####
data = read.csv("IndianLiverPatient.csv", header = TRUE)
attach(data)
dim(data)
#583 observations with 11 variables
head(data)
#2 categorical variables - yvar: "is_patient" and xvar: "gender"
summary(data)
#4 NA's observed in alkphos
#check again total number of missing values
sum(is.na(data))
#4 in total
sum(is.na(data)) / dim(data)[1] *100
#since the NA's only make up 0.69% of the total sample size, we will remove the 4 rows containing missing values
data = na.omit(data)
sum(is.na(data))
#0 now


##### CHECKING AND CORRECTING TYPES OF VARIABLES #####
str(data)
#age             : int
#gender          : factor
#tot_bilirubin    : num
#direct_bilirubin : num
#tot_proteins     : int
#albumin         : int
#ag_ratio        : int
#sgpt            : num
#sgot            : num
#alkphos         : num
#is_patient      : int

#want to change the type of is_patient to factor
#relevel gender so that its baseline is the level with the most observations, i.e. Male
#change types of other variables to numeric
data$is_patient = ifelse(data$is_patient==2,0,1)
data$is_patient = as.factor(data$is_patient)
data$gender = relevel(as.factor(data$gender), "Male") #setting Male as baseline as it is the most commonly occurring
level
for(i in c(1,5:7)){
  data[,i]<-as.numeric(as.character(data[,i]))
}
str(data)
#age             : num
#gender          : factor
#tot_bilirubin    : num
#direct_bilirubin : num
#tot_proteins     : num
#albumin         : num
#ag_ratio        : num
#sgpt            : num
#sgot            : num
#alkphos         : num
#is_patient      : factor


##### INITIAL PLOTS #####
pairs(data, col=data$is_patient, main="First Look at Data - Scatterplots")
#from the plots, noticed some linear correlations between some pairs of covariates, check as below
plot(data$is_patient, data$gender, xlab="Patient Status", ylab="Gender", main="Plot of Gender vs. Patient Status")
#imbalance sample for both, is_patient and gender
```

```
library(reshape2)
library(ggplot2)
data.melt = melt(data[,-c(2)], id.var = "is_patient")
ggplot_box = ggplot(data=data.melt, aes(x=is_patient, y=value, fill=is_patient))
ggplot_box = ggplot_box + geom_boxplot() + facet_wrap(~variable, scales="free_y", ncol = 3)
ggplot_box = ggplot_box + xlab("Patient Status (is_patient: 0=not patient, 1=patient)")
ggplot_box = ggplot_box + ylab("Covariates") + ggtitle("Boxplots of Quantitative Features vs. Patient Status")
ggplot_box = ggplot_box + guides(fill=guide_legend(title="Patient Status"))
ggplot_box = ggplot_box + theme(plot.title = element_text(hjust=0.5))
ggplot_box


##### CORRELATION ANALYSIS #####
library(corrplot)
data_corr = cor(data[ ,-c(2,11)]) #create an object of the features
dev.off()
corrplot.mixed(data_corr, tl.cex=0.55)
title("Correlations Between Numeric Covariates", line = 2.5)

#high correlations between:
#tot_bilirubin, direct bilirubin -- expected, as both are just manipulations of the same thing
#ag_ratio, albumin -- expected, albumin is part of ag_ratio
#sgpt, sgot -- expected, as both have one element that's the same
#alkphos, sgot -- don't know why
#to be wary of these variables, if they are in the final model


##### LOGISTIC REGRESSION MODEL #####
## SPLIT INTO TRAINING AND TESTING DATA ##
library(MASS)
set.seed(1)
train = sample(x = 579, size = 463, replace=FALSE)
#using 80:20 as training:testing ratio
train_data = data[train,]
test_data = data[-train,]
#to ensure a balanced spread of outcome across train_data and test_data
table(data$is_patient)
#split ratio is 165:414 = 0.399
table(train_data$is_patient)
#split ratio is 130:333 = 0.390
table(test_data$is_patient)
#split ratio is 35:81 = 0.432
#acceptable split ratios of outcomes in dataset

## FULL MODEL ##
full.model = glm(is_patient~., data=train_data, family=binomial)
summary(full.model)
#4 predictors are significant at 5% significance level
#4 predictors are: age, direct_bilirubin, albumin, sgpt
confint(full.model)
#the 4 significant predictors have confidence intervals that do not cross zero
#produce odds ratios to enable model interpretability
exp(coef(full.model))
#to interpret -- change in the outcome odds resulting from a unit change in the feature
#all the features will decrease the log odds except sgot
#now to investigate further the potential multicollinearity issue with having a full model
library(car)
vif(full.model)
#vif for all variables are below 2, except sgpt, sgot and alkphos, all exceeding the VIF rule of thumb statistic of 5
#vif for sgpt = 15.81
#vif for sgot = 30.12
#vif for alkphos = 8.97

#test for model fit on train_data
full.model.train.probs=predict(full.model, type="response")
```

```
#inspect the first 5 predicted probabilities
full.model.train.probs[1:5]
contrasts(train_data$is_patient)
#probabilities closer to 0 would be classified as 0("not a patient")
#probabilities closer to 1 would be classified as 1("is a patient")
full.model.train.pred = ifelse(full.model.train.probs>0.5, 1,0)
table(pred=full.model.train.pred, true=train_data$is_patient)
mean(full.model.train.pred==train_data$is_patient)
#accuracy: 0.7365
#sensitivity: 0.919
#specificity: 0.269

#now, test performance on test_data
full.model.test.probs=predict(full.model, newdata=test_data, type="response")
full.model.test.pred = ifelse(full.model.test.probs>0.5, 1,0)
table(pred=full.model.test.pred, true=test_data$is_patient)
mean(full.model.test.pred==test_data$is_patient)
#accuracy: 0.7328
#sensitivity: 0.951
#specificity: 0.229

## BEST SUBSET SELECTION -- FOR MORE PARSIMONIOUS MODEL ##
library(bestglm)

#best subsets using BIC
best.subset.bic = bestglm(Xy = train_data, family=binomial, IC = "BIC", method = "exhaustive")
best.subset.bic
#only includes 3 variables, i.e. age, direct_bilirubin and albumin -- 3 of the 4 significant predictors in full model
names(best.subset.bic)
bic_for_plot <- best.subset.bic$Subsets
#need to refit in order to predict probabilities
best.subset.fit.bic = glm(is_patient~age+direct_bilirubin+albumin, data=train_data, family=binomial)
#test for model fit on train_data
best.subset.train.bic.probs = predict(best.subset.fit.bic, type="response")
best.subset.train.bic.pred=ifelse(best.subset.train.bic.probs>0.5, 1, 0)
table(best.subset.train.bic.pred, train_data$is_patient)
mean(best.subset.train.bic.pred==train_data$is_patient)
#accuracy: 0.728
#sensitivity: 0.931
#specificity: 0.208
#test performance on test_data
best.subset.test.bic.probs = predict(best.subset.fit.bic, newdata=test_data, type="response")
best.subset.test.bic.pred=ifelse(best.subset.test.bic.probs>0.5, 1, 0)
table(best.subset.test.bic.pred, test_data$is_patient)
mean(best.subset.test.bic.pred==test_data$is_patient)
#accuracy: 0.724
#sensitivity: 0.963
#specificity: 0.171

#best subsets using AIC
best.subset.aic = bestglm(Xy = train_data, family=binomial, IC = "AIC", method = "exhaustive")
best.subset.aic
#includes 4 variables, i.e. age, direct_bilirubin, tot_proteins(which was not significant in the full model) and albumin
aic_for_plot <- best.subset.aic$Subsets
#need to refit in order to predict probabilities
best.subset.fit.aic = glm(is_patient~age+direct_bilirubin+tot_proteins+albumin, data=train_data, family=binomial)
#test for model fit on train_data
best.subset.train.aic.probs = predict(best.subset.fit.aic, type="response")
best.subset.train.aic.pred=ifelse(best.subset.train.aic.probs>0.5, 1, 0)
table(best.subset.train.aic.pred, train_data$is_patient)
mean(best.subset.train.aic.pred==train_data$is_patient)
#accuracy: 0.737
#sensitivity: 0.931
#specificity: 0.238
#test performance on test_data
```

```
best.subset.test.aic.probs = predict(best.subset.fit.aic, newdata=test_data, type="response")
best.subset.test.aic.pred=ifelse(best.subset.test.aic.probs>0.5, 1, 0)
table(best.subset.test.aic.pred, test_data$is_patient)
mean(best.subset.test.aic.pred==test_data$is_patient)
#accuracy: 0.7241
#sensitivity: 0.963
#specificity: 0.171

#best subsets using CV, K=10
#need outcome to be called y, so add variable y, later can remove
train_data$y = rep(0, 463)
train_data$y[train_data$is_patient=="1"]=1
head(train_data[ ,12])
#remove the original is_patient column as now we're replacing it with y
train_data.cv = train_data[ ,-11]
head(train_data.cv)
#conduct best subset selection using CV
set.seed(2)
best.subset.cv = bestglm(Xy = train_data.cv, family=binomial, IC = "CV", CVArgs = list(Method="HTF", K=10, REP=1))
best.subset.cv
##using 1 std. error rule, 2 is the number of variables
#need to refit in order to predict probabilities
train_data = train_data[, -12]
best.subset.fit.cv = glm(is_patient~direct_bilirubin+albumin, data=train_data, family=binomial)
#test for model fit on train_data
best.subset.train.cv.probs = predict(best.subset.fit.cv, type="response")
best.subset.train.cv.pred=ifelse(best.subset.train.cv.probs>0.5, 1, 0)
table(best.subset.train.cv.pred, train_data$is_patient)
mean(best.subset.train.cv.pred==train_data$is_patient)
#accuracy: 0.721
#sensitivity: 1.000
#specificity: 0.008
#test performance on test_data
best.subset.test.cv.probs = predict(best.subset.fit.cv, newdata=test_data, type="response")
best.subset.test.cv.pred=ifelse(best.subset.test.cv.probs>0.5, 1, 0)
table(best.subset.test.cv.pred, test_data$is_patient)
mean(best.subset.test.cv.pred==test_data$is_patient)
#accuracy: 0.698
#sensitivity: 1.000
#specificity: 0.000

#penalized logistic regression using LASSO
library(glmnet)
#preparing data
x.lasso.train <-model.matrix(is_patient~.-1, data=train_data)
y.lasso.train <- train_data$is_patient

#generating model
#use CV to select the best lambda, and then find the model with the best lambda
set.seed(1)
cv.logit_lasso <-cv.glmnet(x=x.lasso.train, y=as.factor(y.lasso.train), alpha=1, family="binomial", nfolds=10)
best_lambda <- cv.logit_lasso$lambda.1se
#0.0251
final_logit_lasso <- glmnet(x=x.lasso.train, y=as.factor(y.lasso.train), lambda=best_lambda, family = "binomial")
coef(final_logit_lasso)
#5 variables chosen

#make predictions on the test data
x.lasso.test <- model.matrix(is_patient~.-1, data=test_data)
final_logit_lasso.probs <- predict(final_logit_lasso, newx = x.lasso.test, type="response")
final_logit_lasso.pred <- ifelse(final_logit_lasso.probs>0.5, 1, 0)
table(final_logit_lasso.pred, test_data$is_patient)
mean(final_logit_lasso.pred==test_data$is_patient)
#accuracy: 0.698
#sensitivity: 1.000
```

```
#specificity: 0.000

#plot all 4 methods in one screen
par(mfrow=c(2,2))
##BIC
plot(x=0:10, y=bic_for_plot$BIC, xlab="Number of Features", ylab="BIC", main="BIC vs. Number of Features", pch=19, col="blue")
abline(v=3)
##AIC
plot(x=0:10, y=aic_for_plot$AIC, xlab="Number of Features", ylab="AIC", main="AIC vs. Number of Features", pch=19, col="red")
abline(v=4)
##CV
cverrs <- best.subset.cv$Subsets[,"CV"]
sdCV<- best.subset.cv$Subsets[,"sdCV"]
CVLo <- cverrs - sdCV
CVHi<- cverrs + sdCV
ymax <- max(CVHi)
ymin <- min(CVLo)
k <- 0:(length(cverrs)-1)
plot(k, cverrs, xlab="Number of Features", ylab="CV Error", ylim=c(ymin,ymax), type="n", main="CV Error vs. Number of Features")
points(k, cverrs, cex=2, col="red", pch=16)
lines(k, cverrs, col="red", lwd=2)
axis(2, yaxp=c(0.6,1.8,6))
segments(k, CVLo, k, CVHi, col="blue", lwd=2)
eps <- 0.15
segments(k-eps, CVLo, k+eps, CVLo, col="blue", lwd=2)
segments(k-eps, CVHi, k+eps, CVHi, col="blue", lwd=2)
#cf. oneSDRule
indBest <- oneSDRule(best.subset.cv$Subsets[,c("CV", "sdCV")])
abline(v=indBest-1, lty=2)
indMin <- which.min(cverrs)
fmin <- sdCV[indMin]
cutOff <- fmin + cverrs[indMin]
abline(h=cutOff, lty=2)
indMin <- which.min(cverrs)
fmin <- sdCV[indMin]
cutOff <- fmin + cverrs[indMin]
##LASSO
plot(cv.logit_lasso)
title("LASSO - Feature Selection", line = 2.5)


##### LINEAR DISCRIMINANT ANALYSIS #####
##full model
lda.fit.full = lda(is_patient~., data=train_data)
lda.fit.full
#prior probabilities of groups 0 and 1 are 28.1% and 71.9% respectively
#group means show the average of each feature by their class
#coefficients of linear discriminants are the standardized linear combination of the features that are used to determine an observation’s discriminant score
#the higher the score, the more likely that the classification is 1, i.e. a patient
dev.new()
plot(lda.fit.full, type = "both")
title("Spread of LDA fit across Patient Status")
#a lot of overlap between the 2 groups, indicating that there will be many incorrectly classified observations
#test performance on train_data
lda.full.train.predict = predict(lda.fit.full)
names(lda.full.train.predict)
#use "class" to predict
lda.full.train.pred = lda.full.train.predict$class
table(lda.full.train.pred, train_data$is_patient)
mean(lda.full.train.pred==train_data$is_patient)
#accuracy: 0.728
```

```
#sensitivity: 0.982
#specificity: 0.077
#test performance on test_data
lda.full.test.predict = predict(lda.fit.full, newdata = test_data)
lda.full.test.pred = lda.full.test.predict$class
table(lda.full.test.pred, test_data$is_patient)
mean(lda.full.test.pred==test_data$is_patient)
#accuracy: 0.690
#sensitivity: 0.975
#specificity: 0.029


##### QUADRATIC DISCRIMINANT ANALYSIS #####
##full model
qda.fit.full = qda(is_patient~., data=train_data)
qda.fit.full
#prior probabilities of groups 0 and 1 are 28.1% and 71.9% respectively
#group means show the average of each feature by their class
#however, qda does not have the coefficients as it is a quadratic function
#test performance on train_data
qda.full.train.predict = predict(qda.fit.full)
qda.full.train.pred = qda.full.train.predict$class
table(qda.full.train.pred, train_data$is_patient)
mean(qda.full.train.pred==train_data$is_patient)
#accuracy: 0.562
#sensitivity: 0.420
#specificity: 0.923
#test performance on test_data
qda.full.test.predict = predict(qda.fit.full, newdata = test_data)
qda.full.test.pred = qda.full.test.predict$class
table(qda.full.test.pred, test_data$is_patient)
mean(qda.full.test.pred==test_data$is_patient)
#accuracy: 0.552
#sensitivity: 0.383
#specificity: 0.943


##### KNN #####
library(class)
#prepare data
x.knn.train <-model.matrix(is_patient~.-1, data=train_data)
x.knn.test <- model.matrix(is_patient~.-1, data=test_data)
y.knn.train <- train_data$is_patient

#perform knn on different values of k to find k that minimizes test error
k = seq(1:100)
misclass_error = rep(0, 100)
for(k in 1:100){
  set.seed(1)
  knn_pred = knn(x.knn.train, x.knn.test, y.knn.train, k = k)
  misclass_error[k] = mean(knn_pred != test_data$is_patient)
}
which(misclass_error==min(misclass_error))
#when k=38, misclass_error is minimized
plot(1:k, misclass_error, pch=19, type="b", ylab="Misclassification Error Rate", xlab = "k")
abline(h=min(misclass_error),col="red")
title("Misclassification Error Rate across Different k for KNN")

#fit KNN with k=38
#get the classification predictions
set.seed(1)
knn_pred = knn(x.knn.train, x.knn.test, y.knn.train, k = 38)
table(knn_pred, test_data$is_patient)
mean(knn_pred == test_data$is_patient)
#accuracy: 0.733
```

```
#sensitivity: 0.988
#specificity: 0.143

#get the predicted probabilities
knn_pred_prob = knn(x.knn.train, x.knn.test, y.knn.train, k = 38, prob = T)
knn_pred_prob = attributes(knn_pred_prob)$prob


##### RANDOM FOREST / BAGGING #####
library(randomForest)
#perform random forest with m=3 (done automatically, no need to indicate mtry), trees=500 set automatically
set.seed(1)
rf.fit <- randomForest(is_patient~., data=train_data, importance=TRUE)
rf.fit
#test accuracy on test_data
pred_rf_test_prob = predict(rf.fit, test_data, type = "prob")
pred_rf_test=predict(rf.fit, test_data, type="class")
table(pred_rf_test, test_data$is_patient)
mean(pred_rf_test == test_data$is_patient)
#accuracy: 0.716
#sensitivity: 0.901
#specificity: 0.286
importance(rf.fit)
varImpPlot(rf.fit)
title("Random Forest - Variable Importance Plot", line = 0.8)

#perform bagging with m=p=10
set.seed(1)
bagging.fit <- randomForest(is_patient~., data=train_data, mtry=10, importance=TRUE)
bagging.fit
#test accuracy on test_data
pred_bagging_test_prob=predict(bagging.fit, test_data, type="prob")
pred_bagging_test=predict(bagging.fit, test_data, type="class")
table(pred_bagging_test, test_data$is_patient)
mean(pred_bagging_test == test_data$is_patient)
#accuracy: 0.707
#sensitivity: 0.889
#specificity: 0.286
importance(bagging.fit)
varImpPlot(bagging.fit)
title("Bagging - Variable Importance Plot", line = 0.8)

## Since p=10 here, could try all 10 possible values of `mtry`. Will do so, record the results, and make a plot.
set.seed(1)
oob.err = double(10) #set up variable to record the errors
test.err=double(10)  #set up variable to record the errors
for(mtry in 1:10){
  fit=randomForest(is_patient~. ,data=train_data, mtry=mtry, ntree=500, importance=TRUE) #there is no good reason
to choose 400 trees, but chosen here because it is sufficient
  oob.err[mtry]=fit$err.rate[500]
  pred_rf=predict(fit, test_data, type="class")
  test.err[mtry]=mean(pred_rf != test_data$is_patient)
  cat(mtry, " ")
}
matplot(1:mtry,cbind(test.err,oob.err), pch=19, col=c("red","blue"), type="b", ylab="Misclassification Error", xlab="m -
Number of Predictors", main="Misclassification Error vs. m")
legend("bottomright", legend=c("Test","OOB"), pch=19, col=c("red","blue"))
abline(v=3)
text(4.7, 0.242, "m = 3 -- Random Forest")
#random forest provides the best predictions


##### BOOSTING #####
library(caret)
set.seed(1)
```

```
boost_fit <- caret::train(is_patient ~ .,
                         data = train_data,
                         method = "gbm",
                         preProcess = c("scale", "center"),
                         trControl = trainControl(method = "repeatedcv",
                                          number = 10,
                                          repeats = 3,
                                          verboseIter = FALSE),
                         verbose = 0)
boost_fit
#plot relative influence
par(mar = c(5, 8, 1, 1))
summary(boost_fit, cBars=10, las = 2)
title("Relative Influence Plot for Gradient Boosting")

#n.trees = 50, interaction.depth = 1, shrinkage = 0.1, n.minobsinnode = 10
caret::confusionMatrix(
  data = predict(boost_fit, test_data),
  reference = test_data$is_patient,
  positive = "1"
)
pred_boosting_test_prob=predict(boost_fit, test_data, type="prob")
#accuracy: 0.681
#sensitivity: 0.901
#specificity: 0.171


##### SVM #####
library(e1071)
#preparing data
x.svm.train <- model.matrix(is_patient~.-1, data=train_data)
x.svm.test <- model.matrix(is_patient~.-1, data=test_data)
train.dat = data.frame(x = x.svm.train, y = train_data$is_patient)
test.dat = data.frame(x = x.svm.test, y = test_data$is_patient)

#fitting SVC
svc = svm(y~., data = train.dat, kernel = "linear", cost = 10, scale = TRUE)
summary(svc)
table(fitted = svc$fitted, true = train.dat$y)
#predicts all non-patients as patients even on training data
#try to vary cost to see if results can improve
set.seed(1)
tune.svc = tune(svm, y~., data = train.dat, kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune.svc)
#errors are the same across all cost values -- linear kernel may not be a good fit to the data

#fitting SVM with radial kernel
set.seed(1)
tune.svm = tune(svm, y~., data = train.dat, kernel = "radial", ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma =
c(0.5,1,2,3,4)))
summary(tune.svm)
best.svm <- tune.svm$best.model
tune.svm$best.parameters
#performance on training dataset
table(fitted = best.svm$fitted, true = train.dat$y)
mean(best.svm$fitted==train.dat$y)
#accuracy: 0.955
#sensitivity: 0.997
#specificity: 0.846

#But we have to check performance on test dataset
pred.svm = predict(best.svm, test.dat)
table(pred = pred.svm, truth=test.dat$y)
mean(pred.svm==test.dat$y)
#accuracy: 0.698
```

```
#sensitivity: 0.975
#specificity: 0.057

#for the purpose of plotting ROC curve:
svmfit=svm(y~., data=train.dat, kernel="radial", gamma=4, cost=1, decision.values=T)
svm.radial.decisionvalues <- attributes(predict(svmfit, test.dat, decision.values=T))$decision.values


##### MODEL EVALUATION #####
library(pROC)
#to calculate auc later
roc(predictor = full.model.test.probs, response = test_data$is_patient, positive = 1)$auc
#0.7651
roc(predictor = best.subset.test.bic.probs, response = test_data$is_patient, positive = 1)$auc
#0.7354
roc(predictor = best.subset.test.aic.probs, response = test_data$is_patient, positive = 1)$auc
#0.7432
roc(predictor = best.subset.test.cv.probs, response = test_data$is_patient, positive = 1)$auc
#0.7578
roc(predictor = final_logit_lasso.probs, response = test_data$is_patient, positive = 1)$auc
#0.7199
roc(predictor = lda.full.test.predict$posterior[,1], response = test_data$is_patient, positive = 1)$auc
#0.7295
roc(predictor = qda.full.test.predict$posterior[,1], response = test_data$is_patient, positive = 1)$auc
#0.7460
roc(predictor = knn_pred_prob, response = test_data$is_patient, positive = 1)$auc
#0.7231
roc(predictor = pred_rf_test_prob[,1], response = test_data$is_patient, positive = 1)$auc
#0.7944
roc(predictor = pred_bagging_test_prob[,1], response = test_data$is_patient, positive = 1)$auc
#0.7965
roc(predictor = pred_boosting_test_prob[,1], response = test_data$is_patient, positive = 1)$auc
#0.7545
roc(predictor = svm.radial.decisionvalues, response = test_data$is_patient, positive = 1)$auc
#0.6377

#plot ROC curves
par(pty = "s")
#full logistic model
roc(predictor = full.model.test.probs, response = test_data$is_patient, positive = 1, plot = TRUE, legacy.axes = TRUE,
xlab="False Positive Rate / 1-Specificity", ylab="True Positive Rate / Sensitivity", col="red", main="ROC Curves for
Multiple Machine Learning Algorithms")
#BIC logistic model
plot.roc(test_data$is_patient, predictor = best.subset.test.bic.probs, positive = 1, add=TRUE, col="chocolate")
#AIC logistic model
plot.roc(test_data$is_patient, predictor = best.subset.test.aic.probs, positive = 1, add=TRUE, col="antiquewhite1")
#CV logistic model
plot.roc(test_data$is_patient, predictor = best.subset.test.cv.probs, positive = 1, add=TRUE, col="chartreuse4")
#LASSO logistic model
plot.roc(test_data$is_patient, predictor = final_logit_lasso.probs, positive = 1, add=TRUE, col="aquamarine4")
#LDA
plot.roc(test_data$is_patient, predictor = lda.full.test.predict$posterior[,1], positive = 1, add=TRUE, col="blue")
#QDA
plot.roc(test_data$is_patient, predictor = qda.full.test.predict$posterior[,1], positive = 1, add=TRUE, col="blue4")
#KNN
plot.roc(test_data$is_patient, predictor = knn_pred_prob, positive = 1, add=TRUE, col="cadetblue1")
#Random Forest
plot.roc(test_data$is_patient, predictor = pred_rf_test_prob[,1], positive = 1, add=TRUE, col="black")
#Bagging
plot.roc(test_data$is_patient, predictor = pred_bagging_test_prob[,1], positive = 1, add=TRUE, col="azure4")
#Boosting
plot.roc(test_data$is_patient, predictor = pred_boosting_test_prob[,1], positive = 1, add=TRUE, col="darkgoldenrod2")
#SVM Radial
plot.roc(test_data$is_patient, predictor = svm.radial.decisionvalues, positive = 1, add=TRUE, col="blueviolet")
```

```
legend("bottomright", legend = c("Bagging              : 0.7965", "Random Forest : 0.7944", "Logistic Full        : 0.7651",
"Logistic CV        : 0.7578",
                          "Boosting            : 0.7545", "QDA                  : 0.7460", "Logistic AIC       : 0.7432", "Logistic BIC
: 0.7354",
                          "LDA                 : 0.7295", "KNN                 : 0.7231", "Logistic Lasso   : 0.7199",
                          "SVM            Radial                                                                    :              0.6377"),
col=c("azure4","black","red","chartreuse4","darkgoldenrod2","blue4","antiquewhite1","chocolate","blue","cadetblue1","a
quamarine4","blueviolet"), lwd = 3, title="AUROC (sorted)")


##### CHOOSING FINAL MODEL (BUT BEFORE FURTHER IMPROVEMENT) #####
#logistic using CV
roc_logistic_cv <- roc(predictor = best.subset.test.cv.probs, response = test_data$is_patient, positive = 1)
plot(roc_logistic_cv, legacy.axes=TRUE)
roc_logistic_cv$auc
#0.7578
# look at TPR and TNR distribution over threshold
matplot(data.frame(roc_logistic_cv$sensitivities, roc_logistic_cv$specificities), x = roc_logistic_cv$thresholds, type='l',
xlab = 'threshold', ylab='TPR, TNR')
legend('bottomright', legend=c('TPR', 'TNR'), lty=1:2, col=1:2)
title("TPR, TNR vs. Thresholds for Logistic Regression using BSS - CV", cex=0.4)
abline(v=0.5647181, col="blue", lty=2)
text(0.625, 0.0, "Threshold = 0.565")
abline(h=0.40000000, col="blue", lty=2)
text(0.625, 0.38, "Specificity = 0.400")
abline(h=0.86419753, col="blue", lty=2)
text(0.625, 0.88, "Sensitivity = 0.864")


##### FURTHER IMPROVEMENTS #####
## USE NEYMAN-PEARSON CLASSIFIER WITH LOGISTIC REGRESSION AND RANDOM FOREST ##
library(nproc)
#prepare data -- for NP, important class has to be labelled 0, and non-important class is 1
#here, create y that inverts the assignments for is_patient
#training data
train_data.NP <- train_data
train_data.NP$y = rep(2,463)
train_data.NP$y[train_data.NP$is_patient=="1"]=0
train_data.NP$y[train_data.NP$y=="2"]=1
train_data.NP$y <- as.factor(train_data.NP$y)
train_data.NP <- train_data.NP[,-11]
x.train_data.NP <- model.matrix(y~.-1, data = train_data.NP)
#testing data
test_data.NP <- test_data
test_data.NP$y = rep(2,116)
test_data.NP$y[test_data.NP$is_patient=="1"]=0
test_data.NP$y[test_data.NP$y=="2"]=1
test_data.NP$y <- as.factor(test_data.NP$y)
test_data.NP <- test_data.NP[,-11]
str(test_data.NP)
x.test_data.NP <- model.matrix(y~.-1, data = test_data.NP)

#nproc for random forest
nproc_randomforest = nproc(x=x.train_data.NP, y=train_data.NP$y, method = "randomforest", randSeed = 1, delta =
0.05)
#nproc for logistic using variables selected by CV
nproc_logistic_cv <- nproc(x=x.train_data.NP[,c(5,7)], y=train_data.NP$y, method = "logistic", randSeed = 1, delta =
0.05)
nproc_logistic_cv$auc.l
nproc_logistic_cv$auc.u
nproc_randomforest$auc.l
nproc_randomforest$auc.u
compare(nproc_logistic_cv, nproc_randomforest, plot = TRUE)
legend("bottomright",  legend  =  c("Logistic  -  CV          :  0.589  /  0.742",  "Random  Forest  :  0.570  /  0.727"),
col=c("black","red"), lwd = 3, title="AUROC (lower / upper)")
```

```
title("Neyman-Pearson ROC Bands")
abline(v=0.136, col="blue", lty=2)
abline(h=0.135, col="blue", lty=2)
abline(h=0.35, col="blue", lty=2)
text(0.21, 0.0, "alpha = 0.136", cex=0.7)
text(0.3, 0.155, "specificity - lower bound = 0.135", cex=0.7)
text(0.3, 0.37, "specificity - upper bound = 0.350", cex=0.7)

#use the alpha to get the final model and its predictions
npc.logistic.cv <- npc(x=x.train_data.NP[,c(5,7)], y=train_data.NP$y, method="logistic", alpha = 0.136, delta=0.05)
npc.logistic.cv.pred = predict(npc.logistic.cv, x.test_data.NP[,c(5,7)])
mean(npc.logistic.cv.pred$pred.label == test_data.NP$y)
#accuracy: 0.733
table(npc.logistic.cv.pred$pred.label, test_data.NP$y)
head(npc.logistic.cv.pred$pred.score)
head(npc.logistic.cv.pred$pred.label)
```