Report on Liver Cancer Predictions

Abigail Motley

Overview

The goal of this report is to generate a machine learning model to predict whether or not a patient has liver cancer given patient data. The patient data includes diverse factors such as age, gender, and different chemical compounds found in the body such as bilrubin, albumin, protiens, and alkaline phosphates.

We will use the `Indian Liver Patient Records` dataset collected from the North East of Andhra Pradesh, India by the University of California, School of Information and Computer Science. The dataset is published on kaggle.com and is from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml).

First, we'll clean the data and split it into training and final validation sets. Next, we'll do some analysis on the data and use that insight to build and test some intermediary machine learning models. Last, we'll use the best model to run to run our validation set on. Our final model is `loess` and has an accuracy of .729 on the validation set.

This report demonstrates how machine learning techniques can be applied to help solve impactful real world problems like cancer diagnosis. However, the models generated in this report are limited: the size of the dataset is small, the data was collected from only one region of the world, and there were only a few different types of data collected. In practice, you would want to use a much larger and more varied dataset to improve accuracy and have the ability to apply the model to more patients.

Analysis

Preparing the Data

The `Indian Liver Patient Records` dataset from `https://www.kaggle.com/uciml/indian-liver-patient-records` contains feature columns `Age`, `Gender`, `Total_Bilirubin`, `Direct_Bilirubin`, `Alkaline_Phosphate`, `Alamine_Aminotransferase`, `Aspartate_Aminotransferase`, `Total_Protiens`, `Albumin`, `Albumin_and_Globulin_Ratio`, and a column `DataSet` indicating whether a patient has liver cancer (1) or not (2).

We can read the .csv dataset as a data frame using the `read.csv` function in R. We use the 'stringsAsFactors' argument to ensure our string are not converted into factors. The type is a `data.frame`, and our columns have `int`, `num`, and `char` types.

```
str(liverCancer)
```

```
## 'data.frame':    583 obs. of  11 variables:
##  $ Age                       : int  65 62 62 58 72 46 26 29 17 55 ...
##  $ Gender                    : chr  "Female" "Male" "Male" "Male" ...
##  $ Total_Bilirubin           : num  0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
##  $ Direct_Bilirubin          : num  0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
##  $ Alkaline_Phosphotase      : int  187 699 490 182 195 208 154 202 202 290 ...
##  $ Alamine_Aminotransferase  : int  16 64 60 14 27 19 16 14 22 53 ...
##  $ Aspartate_Aminotransferase: int  18 100 68 20 59 14 12 11 19 58 ...
##  $ Total_Protiens            : num  6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
##  $ Albumin                   : num  3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
##  $ Albumin_and_Globulin_Ratio: num  0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
##  $ Dataset                   : int  1 1 1 1 1 1 1 1 2 1 ...
```

We will then split our data into a matrix of predictors 'X' and a vector of labels 'Y'.

```
liverCancer_x = liverCancer[,1:10]
liverCancer_y = factor(liverCancer[,11:11])
```

Next, we need to scale each feature column by the mean and standard deviation. So, we'll need to convert all columns of 'X' to numeric types.

```
liverCancer_x$Gender <- ifelse(liverCancer_x$Gender == 'Male', 1, 2)
liverCancer_x <- transform(liverCancer_x,
                           Age = as.numeric(Age),
                           Alkaline_Phosphotase = as.numeric(Alkaline_Phosphotase),
                           Alamine_Aminotransferase = as.numeric(Alamine_Aminotransferase),
                           Aspartate_Aminotransferase = as.numeric(Aspartate_Aminotransferase))
```

We also need get rid of all NAs and replace them with the column mean.

```
for(i in 1:ncol(liverCancer_x)){
  liverCancer_x[is.na(liverCancer_x[,i]), i] <- mean(liverCancer_x[,i], na.rm = TRUE)
}
```

Now we can do the scaling.

```
columnMeans = colMeans(liverCancer_x)
columnSDs = apply(liverCancer_x, 2, sd)
x_scaled = sweep(liverCancer_x, 2, columnMeans, "-")
x_scaled = sweep(liverCancer_x, 2, columnSDs, "/")
```

Next, we need to split our data into our test and final validation sets called `edx` and `validation`. The `edx` set will be used to explore our data and create intermediary models. The `validation` set will only be used at the end to test our best model. In this case, we create a partition where 10% of the data is used for the `validation` set.

```
dim(edx_x)
```

```
## [1] 524  10
```

```
dim(validation_x)
```

```
## [1] 59 10
```

Finally, we want to split the `edx` dataset into two separate `train` and `test` subsets that can be used for intermediate accuracy assessment. We use 10% of the `edx` data for the `train` and `test` sets.

```
dim(train_x)
```

```
## [1] 471  10
```

```
dim(test_x)
```

```
## [1] 53 10
```

Now that we've prepared our data, we will explore the properties of various predictors and decide how to use them to create our prediction model.

Exploring the Data

Let's take some time to explore our data.

```
numSamples = dim(liverCancer)[1]
numSamples
```

```
## [1] 583
```

```
numPredictors = dim(liverCancer)[2]
numPredictors
```

```
## [1] 11
```

We have only 583 samples to work with. Note this is not sufficient to train a highly accurate model. We have 11 predictors, which seems like a lot. These may not all be useful in our predictions.

```
mean(liverCancer_y == 1)
```

```
## [1] 0.7135506
```

We have a high proportion of our samples with liver cancer (.71).

```
which.max(columnSDs)
```

```
## Aspartate_Aminotransferase
##                          7
```

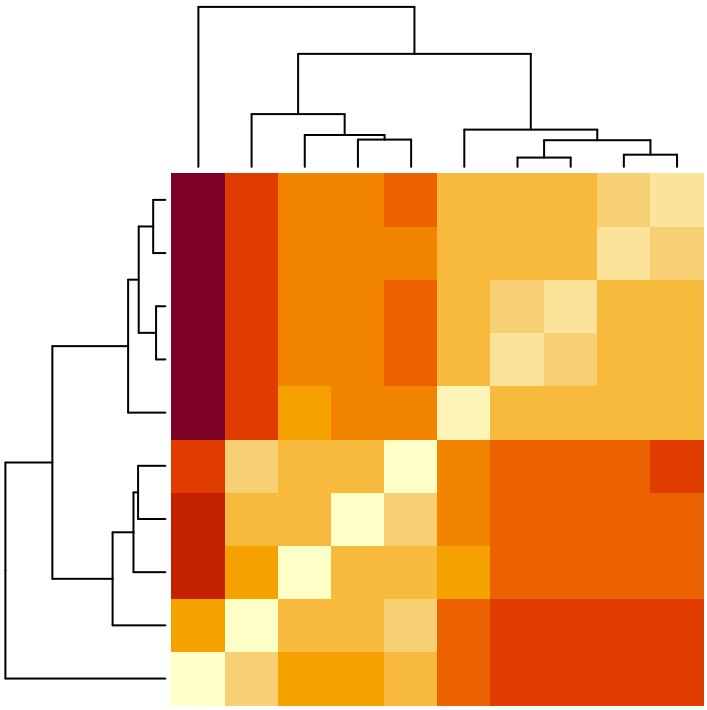
```
which.min(columnSDs)
```

```
## Albumin_and_Globulin_Ratio
##                         10
```

The predictor with the highest standard deviation, `Aspartate_Aminotransferase`, may prove to be the most useful in identifying the patients with and without cancer.

A heatmap and hierarchical clustering of the features show there are some relationships among different features.

```
d_features <- dist(t(x_scaled))
heatmap(as.matrix(d_features), labRow = NA, labCol = NA)
```



```
h <- hclust(d_features)
groups <- cutree(h, k = 5)
split(names(groups), groups)
```

```
## $`1`
## [1] "Age"    "Gender"
##
## $`2`
## [1] "Total_Bilirubin"            "Direct_Bilirubin"
## [3] "Alamine_Aminotransferase"   "Aspartate_Aminotransferase"
##
## $`3`
## [1] "Alkaline_Phosphotase"
##
## $`4`
## [1] "Total_Protiens"
##
## $`5`
## [1] "Albumin"                    "Albumin_and_Globulin_Ratio"
```

Let's run principal component analysis.

```
pca <- prcomp(x_scaled)
summary(pca)
```

```
## Importance of components:
##                            PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.6664 1.4235 1.1697 1.0284 0.95930 0.89575 0.81444
## Proportion of Variance 0.2777 0.2026 0.1368 0.1057 0.09203 0.08024 0.06633
## Cumulative Proportion  0.2777 0.4803 0.6171 0.7229 0.81493 0.89517 0.96150
##                            PC8     PC9    PC10
## Standard deviation     0.45093 0.35435 0.23688
## Proportion of Variance 0.02033 0.01256 0.00561
## Cumulative Proportion  0.98183 0.99439 1.00000
```

We can see the proportion of variance explained by the first principal component analysis is only .278. We need at least 7 principal components to explain at least 90% of the variance. Based on this, we will leverage all features to build our model predictions. While a few may not be as useful, most will be.

Let's look at the average distance between the first sample, which is a patient with liver cancer, and other samples where the patient has cancer.

```
d_samples <- dist(x_scaled)
dist_Cancer <- as.matrix(d_samples)[1, liverCancer_y == 1]
mean(dist_Cancer[2:length(dist_Cancer)])
```

```
## [1] 3.770527
```

Next, let's find out the average distance between the first sample and other samples where the patient does not have cancer.

```
dist_Not_Cancer <- as.matrix(d_samples)[1, liverCancer_y == 2]
mean(dist_Not_Cancer)
```

```
## [1] 3.100319
```

Strangely, we see that the patient with cancer is actually closer in distance to other patients without cancer. This may indicate our features are not going to be very helpful predictors, but we will still try the different machine learning models and see.

Let's move on to creating and evaluating different machine learning models.

Model Evaluations

In this section, we'll train several different machine learning models based on our `train` data, then use the `test` data to evaluate the accuracy. Notice we will not use the `validation` set until the final accuracy analysis at the end.

R makes analyzing our different models fairly straight-forward using the `train` function from the `caret` package, so we will not go into many details. Notice in each case we pass in a different `method` parameter into the `train` method. We will use the mean of the results predicted by the model and the actual results of the `test` set to determine the accuracy of each model.

Logistic Regression

```
train_glm = train(train_x, train_y, method = "glm")
y_hat_glm = predict(train_glm, test_x)
mean(y_hat_glm == test_y)
```

```
## [1] 0.7358491
```

LDA

```
train_lda = train(train_x, train_y, method = "lda")
y_hat_lda = predict(train_lda, test_x)
mean(y_hat_lda == test_y)
```

```
## [1] 0.7358491
```

QDA

```
train_qda = train(train_x, train_y, method = "qda")
y_hat_qda = predict(train_qda, test_x)
mean(y_hat_qda == test_y)
```

```
## [1] 0.5471698
```

Loess

```
set.seed(5, sample.kind="Rounding")
train_loess = train(train_x, train_y, method = "gamLoess")
y_hat_loess = predict(train_loess, test_x)
mean(y_hat_loess == test_y)
```

```
## [1] 0.7735849
```

kNN. Notice that k-nearest neighbors uses an `tuneGrid` parameter. This is to give a range of k's which will be sampled, after which the best parameter to use to train the final model.

```
set.seed(7, sample.kind="Rounding")
tuning <- data.frame(k = seq(3, 21, 2))
train_knn <- train(train_x, train_y,
                   method = "knn",
                   tuneGrid = tuning)
train_knn$bestTune
```

```
##     k
## 10 21
```

```
y_knn <- predict(train_knn, test_x)
mean(y_knn == test_y)
```

```
## [1] 0.754717
```

Random Forest

```
set.seed(9, sample.kind = "Rounding")
train_rf = train(train_x, train_y, ntree = 100, method = "rf", importance=TRUE, tuneGrid = data.frame(m
train_rf$bestTune
```

```
##   mtry
## 1    3
```

```
y_hat_rf = predict(train_rf, test_x)
mean(y_hat_rf == test_y)
```

```
## [1] 0.7358491
```

```
varImp(train_rf$finalModel)
```

```
##                                1         2
## Age                        2.0223821 2.0223821
## Gender                     2.0511832 2.0511832
## Total_Bilirubin            3.3376194 3.3376194
## Direct_Bilirubin           3.1321657 3.1321657
## Alkaline_Phosphotase       2.0757521 2.0757521
## Alamine_Aminotransferase   3.0820760 3.0820760
## Aspartate_Aminotransferase 2.4545408 2.4545408
## Total_Protiens             0.8303932 0.8303932
## Albumin                    2.0617012 2.0617012
## Albumin_and_Globulin_Ratio 1.5607390 1.5607390
```

You can see the most important variable from the random forest model is `Aspartate_Aminotransferase`. Interestingly, this matches our earlier data analysis where we speculated that the column with the highest SD may be the most useful factor.

Next we'll try an ensemble method using the previous models. To create our ensemble method, we'll use majority voting over all the previous models to determine the prediction.

```
ensemble <- cbind(glm = y_hat_glm == 1,
                  lda = y_hat_lda == 1,
                  qda = y_hat_qda == 1,
                  loess = y_hat_loess == 1,
                  rf = y_hat_rf == 1,
                  knn = y_knn == 1)

y_hat_ensemble <- ifelse(rowMeans(ensemble) > 0.5, 1, 2)
mean(y_hat_ensemble == test_y)
```

```
## [1] 0.754717
```

Results

This chart shows the intermediate accuracy of different models.

```r
data.frame(Model = models, Accuracy = accuracy)
```

```
##                  Model  Accuracy
## 1 Logistic regression 0.7358491
## 2                  LDA 0.7358491
## 3                  QDA 0.5471698
## 4               Loess 0.7735849
## 5 K nearest neighbors 0.7547170
## 6       Random forest 0.7358491
## 7            Ensemble 0.7547170
```

You can see the `loess` model is the most accurate. We'll choose this method as our final selection and use the `validation` set to calculate final accuracy.

```r
y_hat_final = predict(train_loess, validation_x)
mean(y_hat_final == validation_y)
```

```
## [1] 0.7288136
```

The accuracy of the final model on our `validation` set is .729.

Let's look at the proportion of patients with liver cancer correctly classified.

```r
sum(validation_y == 1 & y_hat_final == 1)/sum(validation_y == 1)
```

```
## [1] 0.9047619
```

We were able to correctly identify 90% of the patients with liver cancer.

Let's see the proportion of patients without liver cancer that were predicted as having cancer.

```r
sum(validation_y == 2 & y_hat_final == 1)/sum(validation_y == 2)
```

```
## [1] 0.7058824
```

We can see we've incorrectly classified 70% of patients without cancer as having cancer.

Conclusion

To summarize, we've shown how to build a machine learning model to predict whether or not a patient has liver cancer using the `Indian Liver Patient Records` dataset.

This process included data cleaning, data analysis, and evaluating and selecting the best model. In the end, we were able to build a `loess` regression model that performs with a accuracy of .729.

We saw that even though we correctly identified a high percentage of patients with cancer as having cancer, we also incorrectly identified patients without cancer as having cancer. The low accuracy can be explained in part by considering the limitations of this dataset. Recall that our data was unbalanced, containing 71% patients with cancer. The dataset contained a small sample size of 583. Finally, the analysis on the distance between patients indicated the predictors may not be particluarly useful.

Despite the limitations, this report demonstrates how machine learning might be used to solve important problems in the world using only data.