Tree Models

Jim Harner

1/12/2021

6.7 Other ML Algorithms

6.7.1 Basics

Spark 3 and `sparklyr` have many algorithms for classification. We have been exposed to logistic regression in Sections 6.4 and 6.5. We will compare 6 classification algorithms in this chapter.

- Logistic Regression: models the posterior conditional probabilities of the k classes by a linear functions in X, while ensuring that the probabilities sum to one and remain in [0,1]. The model is specified in terms of $k-1$ logit transformations with the last class as the denominator in the odds-ratios.
- Decision Tree (DT): use weak learners which output real values for splits. These outputs can be added together, allowing subsequent model outputs to be added and to correct the residuals in the predictions. Trees choose the best split points based on purity scores, e.g., Gini, or minimizing the loss. Regularization can be used to reduce overfitting. The output variable can be discrete (classification) or continuous (regression).
- Random Forests (RF): is an ensemble learner that combines bagging with random feature selection. Bagging (bootstrap aggregation) reduces the variance of a prediction model, e.g., a tree which tends to have high variance, but low bias.
- Gradient Boosted Trees (GBT): combines gradient-based optimization with boosting. Boosting additively collects an ensemble of weak learners to create a strong learner for prediction. Gradient boosting can be used for both regression and k-group classification.
- Naive Bayes (NB): are simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It is often used when the dimension p of the feature space is high.
- Neural Net (MLP): is a feedforward artificial neural network. An MLP has at least three layers of variables (nodes). Except for the input variables, each node uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training.

6.6.2 Model Comparison Wine Quality Example

We illustrate logistic regression modeling using the Wine Quality Data Set from the UCI Machine Learning Repository. The two datasets: the red and white variants of the Portuguese "Vinho Verde" wine, but we will restrict analysis to the red wine dataset.

The features (or predictors) are all numeric, whereas the labels (or the predictand) is a score ranging from 0 to 10 representing the wine quality. Technically, this is an ordered categorical model, but we will binarize the outcome variable at a threshold value of 5.0.

Read the `winequality-red.csv` file directory into a Spark DataFrame using `spark_red_sdf`.

```
wine_red_sdf <- spark_read_csv(sc, "wine_red_sdf",
                               path = "file:///home/rstudio/rspark-tutorial/data/wine/winequality-red.c
```

```r
                                delimiter = ";" )
wine_red_tbl <- sdf_register(wine_red_sdf, name = "wine_red_tbl")
```

We register the Spark DataFrame so that the Scala Spark DataFrame API is used directly rather than the `dplyr` interface. Registering forces the SQL to completion without using `collect`, which is necessary for the pipeline in the next chunk.

We split `wine_red_sdf` into a training and a test Spark DataFrame. First, we need to cast `quality` as numeric in order to binarize it with a threshold.

```r
wine_red_partition <- wine_red_tbl %>%
  mutate(quality = as.numeric(quality)) %>%
  ft_binarizer(input_col = "quality", output_col = "quality_bin",
               threshold = 5.0) %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 2)
# Create table references
wine_red_train_tbl <- wine_red_partition$training
wine_red_test_tbl <- wine_red_partition$test
```

Classifier Comparison At this point, we will compare logistics regression with other classification methods. The training data is used here. The 7-variable model from Section 6.4 is used.

```r
## Specify the model formula
ml_formula <- formula(quality_bin ~ fixed_acidity + volatile_acidity + citric_acid +
                        free_sulfur_dioxide + total_sulfur_dioxide + sulphates + alcohol)
## Logistic Regression
ml_log <- ml_logistic_regression(wine_red_train_tbl, ml_formula)
## Decision Tree
ml_dt <- ml_decision_tree(wine_red_train_tbl, ml_formula)

## Random Forest
ml_rf <- ml_random_forest(wine_red_train_tbl, ml_formula)

## Gradient Boosted Tree
ml_gbt <- ml_gradient_boosted_trees(wine_red_train_tbl, ml_formula)

## Naive Bayes
ml_nb <- ml_naive_bayes(wine_red_train_tbl, ml_formula)

## Neural Network
ml_nn <- ml_multilayer_perceptron(wine_red_train_tbl, ml_formula, layers = c(7, 5, 2))
```

```r
## Bundle the models results into a list structure
ml_models <- list(
  "Logistic" = ml_log,
  "Decision Tree" = ml_dt,
  "Random Forest" = ml_rf,
  "Gradient Boosted Trees" = ml_gbt,
  "Naive Bayes" = ml_nb,
  "Neural Net" = ml_nn
)

# Create a function for scoring
```

```r
score_test_data <- function(model, data = wine_red_test_tbl){
  ml_predict(model, data) %>%
  select(quality_bin, prediction)
}

# Score all the models

ml_score <- lapply(ml_models, score_test_data)
```

Apply the models to the validation data

```r
# Lift function
calculate_lift <- function(scored_data) {
  scored_data %>%
    mutate(bin = ntile(desc(prediction), 10)) %>%
    group_by(bin) %>%
    summarize(count = sum(quality_bin)) %>%
    mutate(prop = count / sum(count)) %>%
    arrange(bin) %>%
    mutate(prop = cumsum(prop)) %>%
    select(-count) %>%
    collect() %>%
    as.data.frame()
}

# Initialize results
ml_gains <- data.frame(bin = 1:10, prop = seq(0, 1, len = 10), model = "Base")

# Calculate lift
for(i in names(ml_score)){
  ml_gains <- ml_score[[i]] %>%
    calculate_lift %>%
    mutate(model = i) %>%
    rbind(ml_gains, .)
}
ml_gains
```

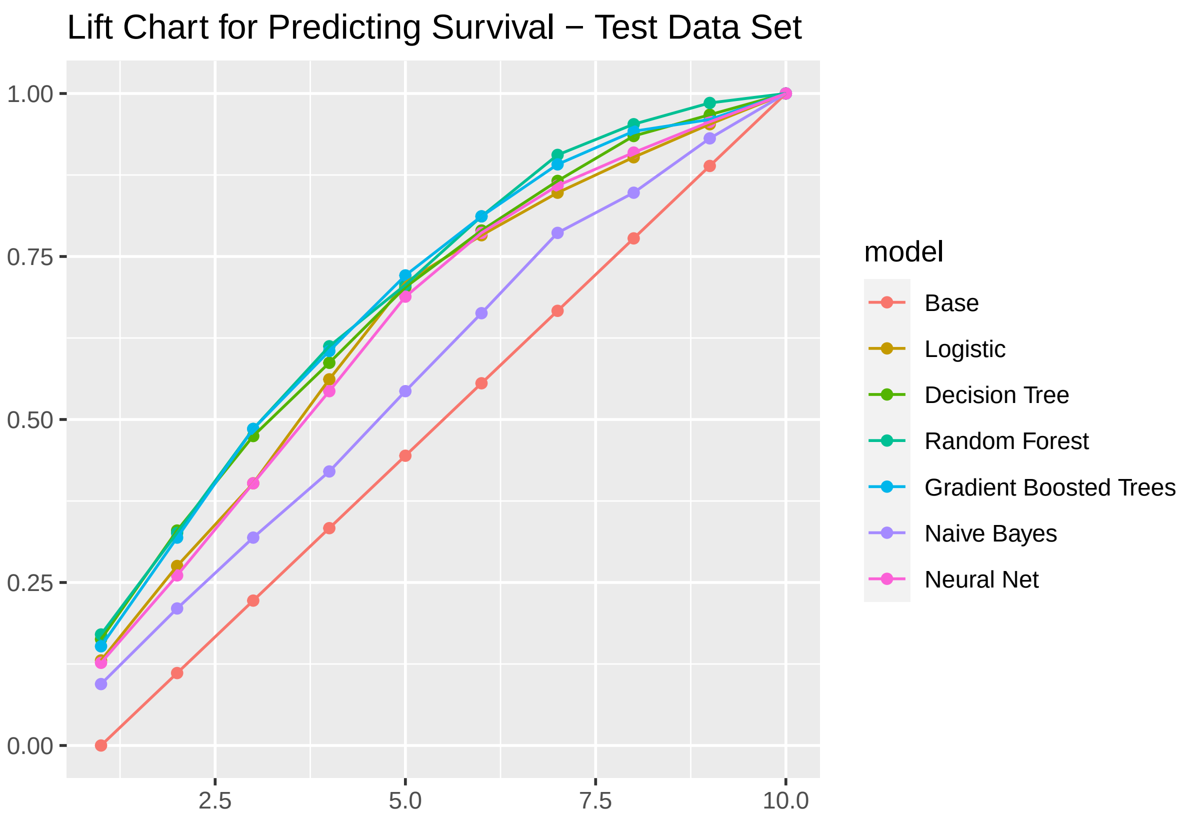
Calculate Lift

```
##    bin      prop                  model
## 1    1 0.0000000                   Base
## 2    2 0.1111111                   Base
## 3    3 0.2222222                   Base
## 4    4 0.3333333                   Base
## 5    5 0.4444444                   Base
## 6    6 0.5555556                   Base
## 7    7 0.6666667                   Base
## 8    8 0.7777778                   Base
## 9    9 0.8888889                   Base
## 10  10 1.0000000                   Base
## 11   1 0.1304348               Logistic
## 12   2 0.2753623               Logistic
## 13   3 0.4021739               Logistic
```

```
## 14   4 0.5615942               Logistic
## 15   5 0.7101449               Logistic
## 16   6 0.7826087               Logistic
## 17   7 0.8478261               Logistic
## 18   8 0.9021739               Logistic
## 19   9 0.9528986               Logistic
## 20  10 1.0000000               Logistic
## 21   1 0.1630435          Decision Tree
## 22   2 0.3297101          Decision Tree
## 23   3 0.4746377          Decision Tree
## 24   4 0.5869565          Decision Tree
## 25   5 0.7028986          Decision Tree
## 26   6 0.7898551          Decision Tree
## 27   7 0.8659420          Decision Tree
## 28   8 0.9347826          Decision Tree
## 29   9 0.9673913          Decision Tree
## 30  10 1.0000000          Decision Tree
## 31   1 0.1702899          Random Forest
## 32   2 0.3260870          Random Forest
## 33   3 0.4855072          Random Forest
## 34   4 0.6123188          Random Forest
## 35   5 0.7065217          Random Forest
## 36   6 0.8115942          Random Forest
## 37   7 0.9057971          Random Forest
## 38   8 0.9528986          Random Forest
## 39   9 0.9855072          Random Forest
## 40  10 1.0000000          Random Forest
## 41   1 0.1521739 Gradient Boosted Trees
## 42   2 0.3188406 Gradient Boosted Trees
## 43   3 0.4855072 Gradient Boosted Trees
## 44   4 0.6050725 Gradient Boosted Trees
## 45   5 0.7210145 Gradient Boosted Trees
## 46   6 0.8115942 Gradient Boosted Trees
## 47   7 0.8913043 Gradient Boosted Trees
## 48   8 0.9420290 Gradient Boosted Trees
## 49   9 0.9601449 Gradient Boosted Trees
## 50  10 1.0000000 Gradient Boosted Trees
## 51   1 0.0942029            Naive Bayes
## 52   2 0.2101449            Naive Bayes
## 53   3 0.3188406            Naive Bayes
## 54   4 0.4202899            Naive Bayes
## 55   5 0.5434783            Naive Bayes
## 56   6 0.6630435            Naive Bayes
## 57   7 0.7862319            Naive Bayes
## 58   8 0.8478261            Naive Bayes
## 59   9 0.9311594            Naive Bayes
## 60  10 1.0000000            Naive Bayes
## 61   1 0.1268116             Neural Net
## 62   2 0.2608696             Neural Net
## 63   3 0.4021739             Neural Net
## 64   4 0.5434783             Neural Net
## 65   5 0.6884058             Neural Net
## 66   6 0.7862319             Neural Net
## 67   7 0.8586957             Neural Net
```

```
## 68   8 0.9094203              Neural Net
## 69   9 0.9565217              Neural Net
## 70  10 1.0000000              Neural Net
```

```r
# Plot results
ggplot(ml_gains, aes(x = bin, y = prop, colour = model)) +
  geom_point() + geom_line() +
  ggtitle("Lift Chart for Predicting Survival - Test Data Set") +
  xlab("") + ylab("")
```



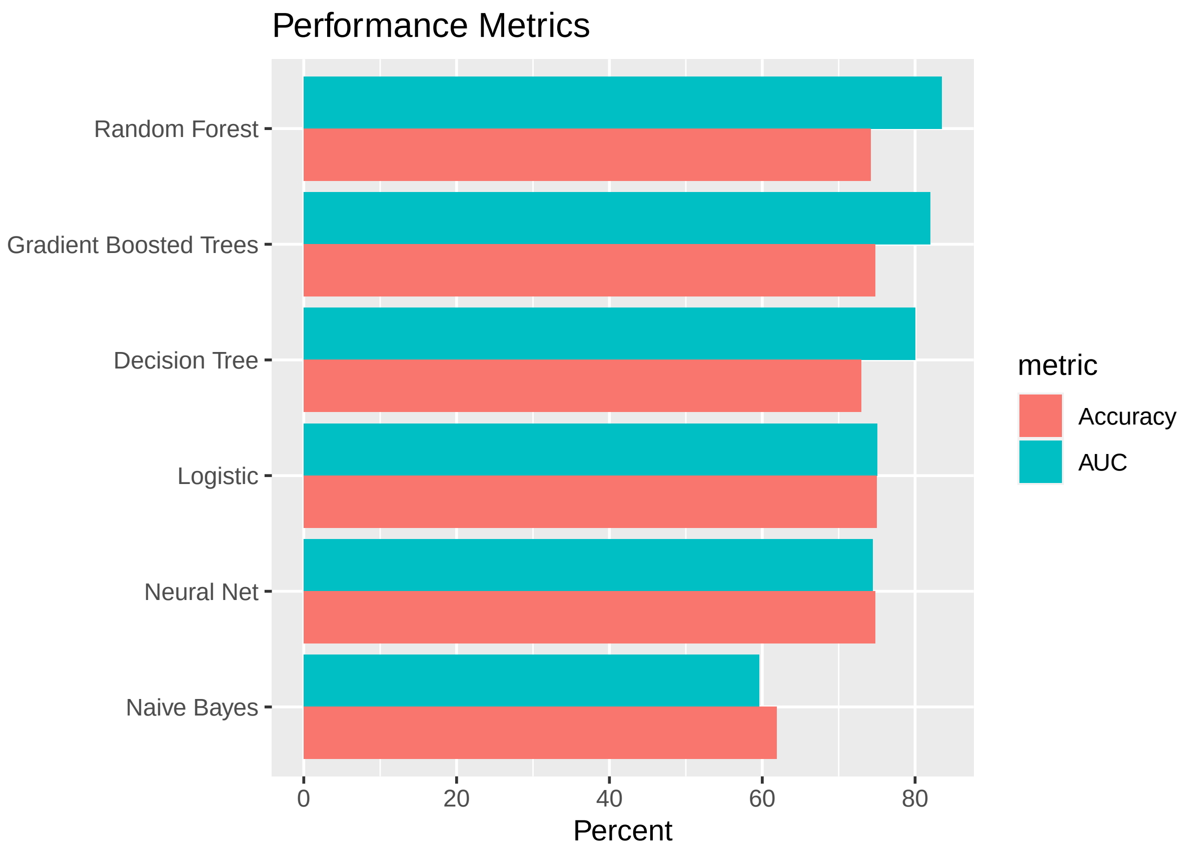
AUC and accuracy Spark does not provide output for computing the ROC curve, but the Area under the ROC curve (AUC) is provided. The higher the AUC the better.

```r
# Function for calculating accuracy
calc_accuracy <- function(data, cutpoint = 0.5){
  data %>%
    mutate(prediction = if_else(prediction > cutpoint, 1.0, 0.0)) %>%
    ml_classification_eval("prediction", "quality_bin", "accuracy")
}

# Calculate AUC and accuracy
perf_metrics <- data.frame(
  model = names(ml_score),
  AUC = 100 * sapply(ml_score, ml_binary_classification_eval, "quality_bin", "prediction"),
  Accuracy = 100 * sapply(ml_score, calc_accuracy),
  row.names = NULL, stringsAsFactors = FALSE)

# Plot results
gather(perf_metrics, metric, value, AUC, Accuracy) %>%
```

```r
  ggplot(aes(reorder(model, value), value, fill = metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  xlab("") +
  ylab("Percent") +
  ggtitle("Performance Metrics")
```



The tree-based methods perform the best followed by logistic regression, at least by using accuracy and AUC as measures. Naive Bayes in not competitive.

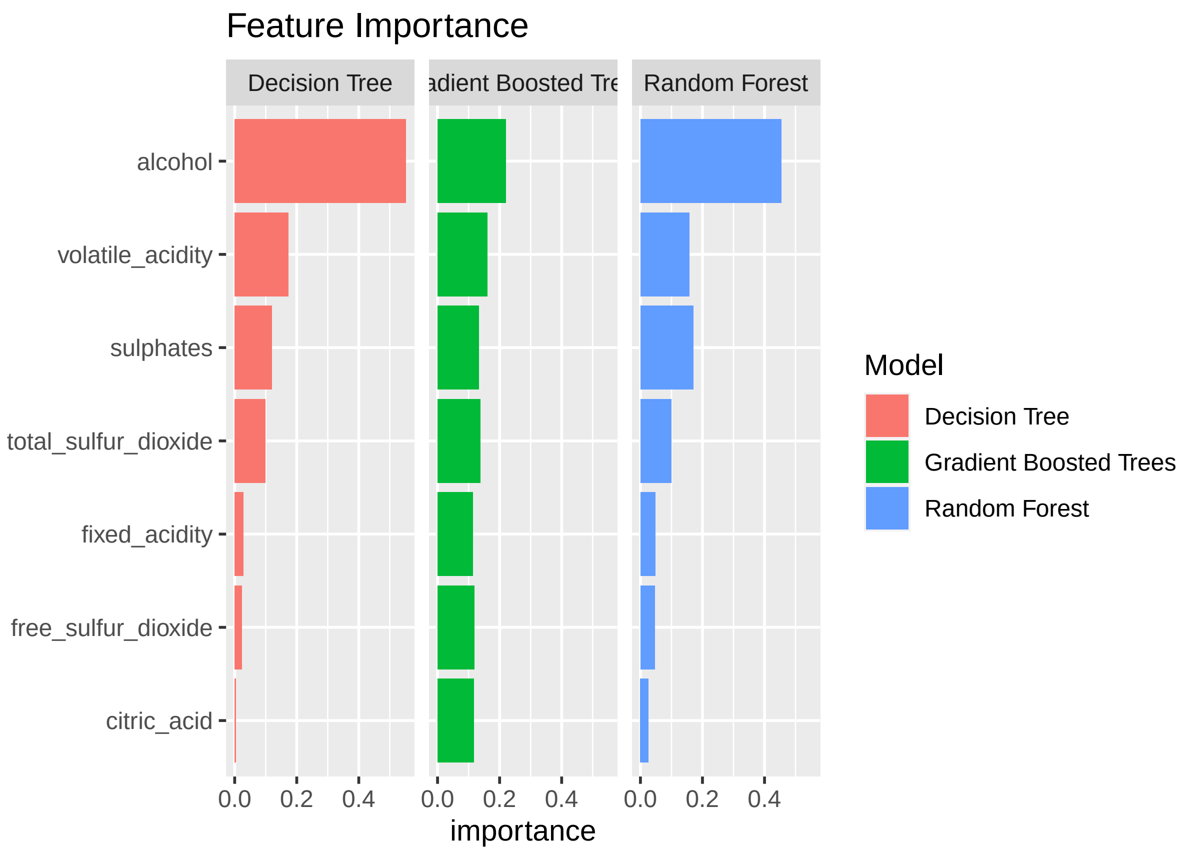
Feature importance We now compare the features that were identified by each model as being important predictors for red wine quality. The logistic regression and tree models implement feature importance metrics, but only the tree models will be used here.

```r
# Initialize results
feature_importance <- data.frame()

# Calculate feature importance
for(i in c("Decision Tree", "Random Forest", "Gradient Boosted Trees")){
  feature_importance <- ml_tree_feature_importance(ml_models[[i]]) %>%
    mutate(Model = i) %>%
#   mutate(importance = as.numeric(levels(importance))[importance])
    mutate(feature = as.character(feature)) %>%
    rbind(feature_importance, .)
}

# Plot feature importance
feature_importance %>%
```

```r
  ggplot(aes(reorder(feature, importance), importance, fill = Model)) +
  facet_wrap(~ Model) +
  geom_bar(stat = "identity") +
  coord_flip() +
  xlab("") +
  ggtitle("Feature Importance")
```



`alcohol` is the most important feature with `volatile acidity` and `sulphates` as the the second and third most important features with the order depending on the type of model. Other features have varying degrees of predictive importance. The features in Gradient Boosted Trees are more even in terms of importance.

```r
spark_disconnect(sc)
```