HW5 - Tree-based Inference [MACS 30100]

Adarsh Mathew

2/24/2020

Cost functions for Classification Trees

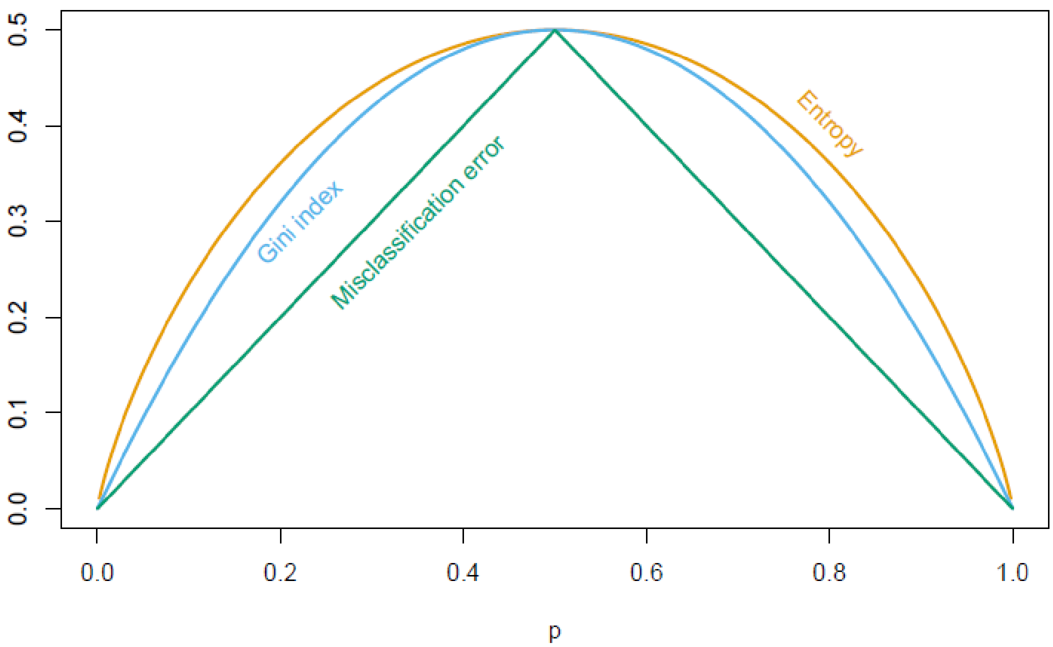


Figure 1: Node impurity measures for two-class classification, as a function of the proportion p in class 2.

While growing a tree, we're concerned with the purity of nodes – we ideally want trees which provide much cleaner (dominated by a single outcome class) partitions of the predictor space. Thus, we'd prefer a cost function which levies the greatest penalty on impure nodes. From the figure above[1], we can see that both the Gini index and cross-entropy are more sensitive to node impurity with their non-linear curves. We could choose from either of the two metrics to grow the tree.

For cost-complexity pruning, the objective is always minimization of error (while avoiding over-fitting). Thus, it makes sense to choose misclassification error, but the other two metrics can be used too[2].

[1] Source: Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, pg 309. Springer Science & Business Media, 2009.

[2] Hastie et al. explicity state that '…any of the three measures can be used, but typically it is the misclassification rate.'

Application Exercises

```
gss_train_df <- read_csv("../data/gss_train.csv",
                         col_types = cols(colrac = col_factor(levels = c("1", "0")))) %>%
   mutate(colrac = factor(colrac,
                          labels = make.names(levels(colrac))))

gss_test_df <- read_csv("../data/gss_test.csv",
                         col_types = cols(colrac = col_factor(levels = c("1", "0")))) %>%
   mutate(colrac = factor(colrac,
                          labels = make.names(levels(colrac))))
```

Model Estimation

```
# Helper function to feed into `map`

model_gen <- function(model_type){
  set.seed(02242020)
  tr_control <- trainControl(method = "cv", number = 10,
                             summaryFunction=twoClassSummary,
                             classProbs=T,
                             savePredictions = T)

  gss_model <- train(form = colrac ~ ., data = gss_train_df,
                     method = model_type, preProcess = c("center", "scale"),
                     metric = "ROC", trControl = tr_control, tuneLength = 5)

  return(gss_model)
}
```

```
plan(multiprocess)
model_df <- c("Logit" = "glm", "Naive Bayes" = "nb",
              "Elastic Net" = "glmnet", "CART_1SE" = "rpart1SE",
              "Bagged Trees" = "treebag", "Random Forest" = "rf",
              "xgBoosted Trees" = "xgbTree") %>%
  enframe(name = "model_name", value = "model_io") %>%
  mutate(model_obj = future_map(model_io, ~model_gen(.x)))
```

Model Evaluation

```
model_cv_metrics <- function(model_obj, model_io){
  pred_mat <- model_obj$pred
  model_auc <- prSummary(pred_mat, lev = levels(pred_mat$pred), model = model_io)["AUC"]
  model_acc <- postResample(pred = pred_mat$pred, obs = pred_mat$obs)["Accuracy"]

  # model_roc <- roc(predictor=pred_mat$X1, response=pred_mat$obs,
  #                  levels=levels(pred_mat$pred))
  # plot(model_roc)

  return(tibble(model_cv_accuracy = model_acc, model_cv_AUC = model_auc))

}
```

```r
model_df <- model_df %>%
  mutate(xx = map2(model_obj, model_io, model_cv_metrics)) %>%
  unnest(xx)

model_df %>%
  select(-model_obj) %>%
  arrange(desc(model_cv_AUC))
```

```
## # A tibble: 7 x 4
##   model_name      model_io model_cv_accuracy model_cv_AUC
##   <chr>           <chr>                <dbl>        <dbl>
## 1 Random Forest   rf                   0.803        0.875
## 2 Logit           glm                  0.799        0.874
## 3 Elastic Net     glmnet               0.797        0.865
## 4 xgBoosted Trees xgbTree              0.792        0.854
## 5 Naive Bayes     nb                   0.731        0.787
## 6 Bagged Trees    treebag              0.783        0.766
## 7 CART_1SE        rpart1SE             0.788        0.762
```

Model Selection

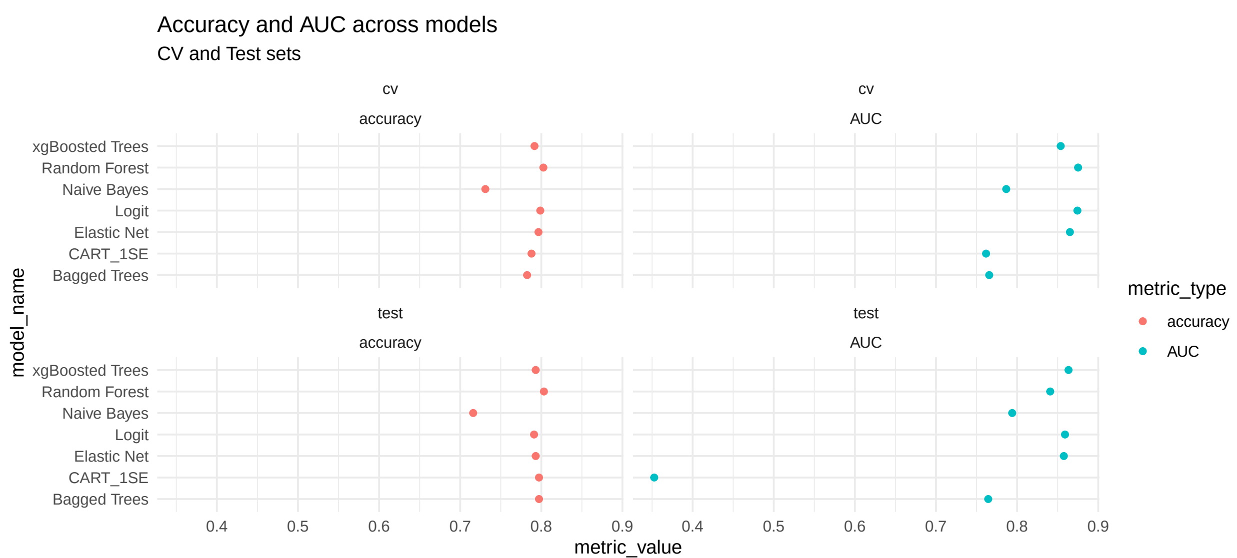
```r
model_test_metrics <- function(model_obj, model_io){
  pred_df <- gss_test_df %>%
    select(colrac) %>%
    mutate(obs = colrac) %>%
    bind_cols(pred = as.factor(predict(model_obj, newdata = gss_test_df, type = c("raw"))),
              predict(model_obj, newdata = gss_test_df, type = c("prob")))

  # model_roc <- roc(predictor=pred_df$X1, response=pred_df$obs,
  #                  levels=levels(pred_df$pred))
  # plot(model_roc)
  model_auc <- prSummary(pred_df, lev = levels(pred_df$pred), model = model_io)["AUC"]
  model_acc <- postResample(pred_df$pred, pred_df$colrac)["Accuracy"]

  return(tibble(model_test_accuracy = model_acc,
                #model_test_ROC = model_roc,
                model_test_AUC = model_auc))

}
```

```r
model_df <- model_df %>%
  mutate(xx = map2(model_obj, model_io, model_test_metrics)) %>%
  unnest(xx)

model_df %>%
  select(-model_obj) %>%
  arrange(desc(model_test_AUC))
```

```
## # A tibble: 7 x 6
##   model_name model_io model_cv_accura~ model_cv_AUC model_test_accu~
##   <chr>      <chr>               <dbl>        <dbl>            <dbl>
## 1 xgBoosted~ xgbTree             0.792        0.854            0.793
## 2 Logit      glm                 0.799        0.874            0.791
## 3 Elastic N~ glmnet              0.797        0.865            0.793
```

```
## 4 Random Fo~ rf                    0.803        0.875            0.801
## 5 Naive Bay~ nb                    0.731        0.787            0.716
## 6 Bagged Tr~ treebag               0.783        0.766            0.797
## 7 CART_1SE   rpart1SE              0.788        0.762            0.797
## # ... with 1 more variable: model_test_AUC <dbl>
```

```
model_df_stats <- model_df %>%
  pivot_longer(cols = c("model_test_accuracy", "model_cv_accuracy",
                         "model_test_AUC", "model_cv_AUC"),
               names_to = "metric_type", values_to = "metric_value") %>%
  select(-model_obj) %>%
  separate(col = "metric_type", into = c("model", "df_type", "metric_type"), sep = "_") %>%
  select(-model)

model_df_stats %>%
  ggplot(aes(x = model_name, y = metric_value, colour = metric_type)) +
  geom_point() +
  facet_wrap(vars(df_type, metric_type)) + coord_flip() +
  theme_minimal() +
  ggtitle("Accuracy and AUC across models", subtitle = "CV and Test sets")
```



On purely CV metrics, Logistic, Elastic Net, and Random Forest perform best. They have high AUC and accuracy metrics. Of these, I'd choose Logistic since it would have greater interpretability over Random Forest, and it has higher metrics than Elastic Net.

Looking at the test metrics though, we see Random Forest drop away, with xgBoost performing marginally better than Logistic and Elastic Net. I'd still pick Logistic, given the interpretability advantages and that it seems to generalize well here.

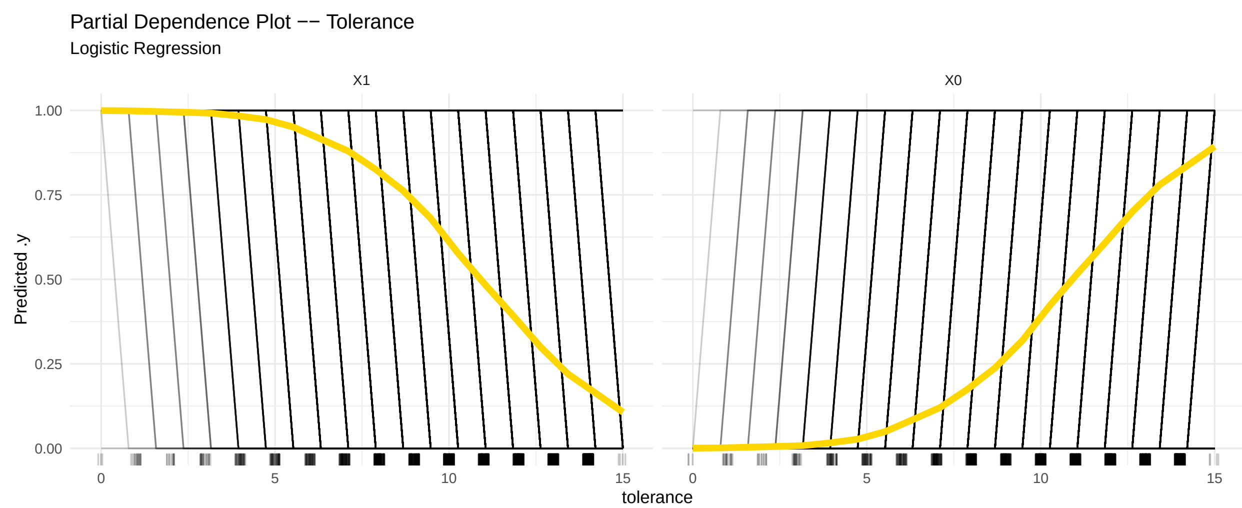
Partial Dependence plots

```
model_df_im <- model_df %>%
  filter(model_io == "glm") %>%
  mutate(predictor_obj = map(model_obj,
                             ~Predictor$new(model = .x,
                                            data = gss_train_df %>%
                                              select(-colrac),
```

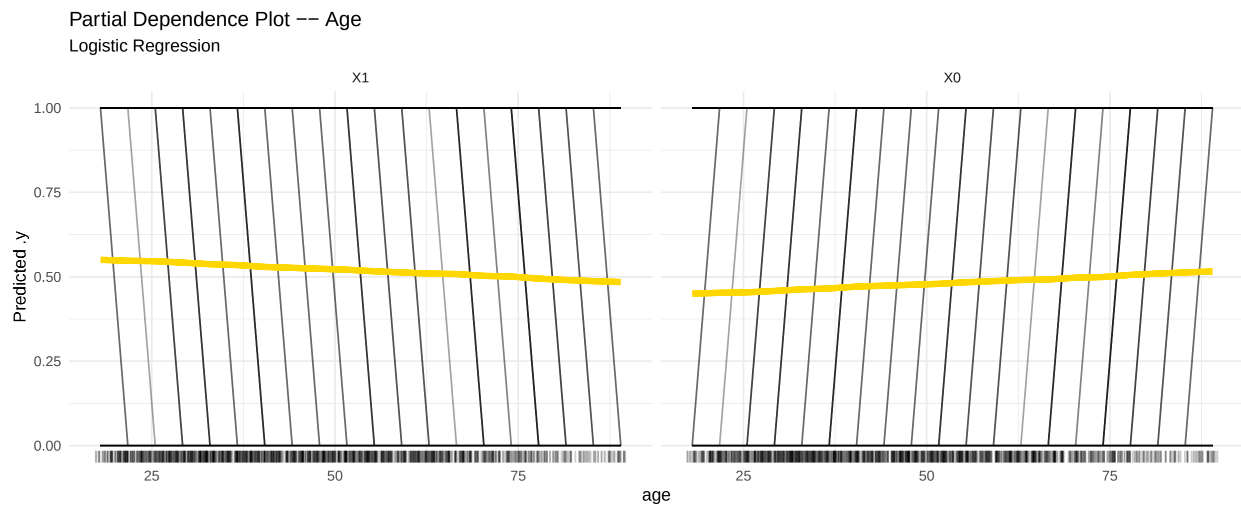
```
                                                   y = gss_train_df$colrac,
                                                   predict.fun = predict(.x, newdata = gss_test_df))),
         pdp_tol_obj = map(predictor_obj,
                           ~Partial$new(.x, "tolerance")),
         pdp_age_obj = map(predictor_obj,
                           ~Partial$new(.x, "age")),
         pdp_agetol_obj = map(predictor_obj,
                           ~Partial$new(.x, c("age","tolerance"))),
         ice_tol_obj = map(predictor_obj,
                           ~Partial$new(.x, "tolerance", ice = TRUE)),
         ice_age_obj = map(predictor_obj,
                           ~Partial$new(.x, "age", ice = TRUE)),
         ice_agetol_obj = map(predictor_obj,
                           ~Partial$new(.x, c("age","tolerance"), ice = TRUE)))
```

```
## Warning: The FeatureEffect class replaces the Partial class. Partial will be
## removed in future versions.

## Warning: The FeatureEffect class replaces the Partial class. Partial will be
## removed in future versions.

## Warning: The FeatureEffect class replaces the Partial class. Partial will be
## removed in future versions.

## Warning: The FeatureEffect class replaces the Partial class. Partial will be
## removed in future versions.

## Warning: The FeatureEffect class replaces the Partial class. Partial will be
## removed in future versions.

## Warning: The FeatureEffect class replaces the Partial class. Partial will be
## removed in future versions.
```

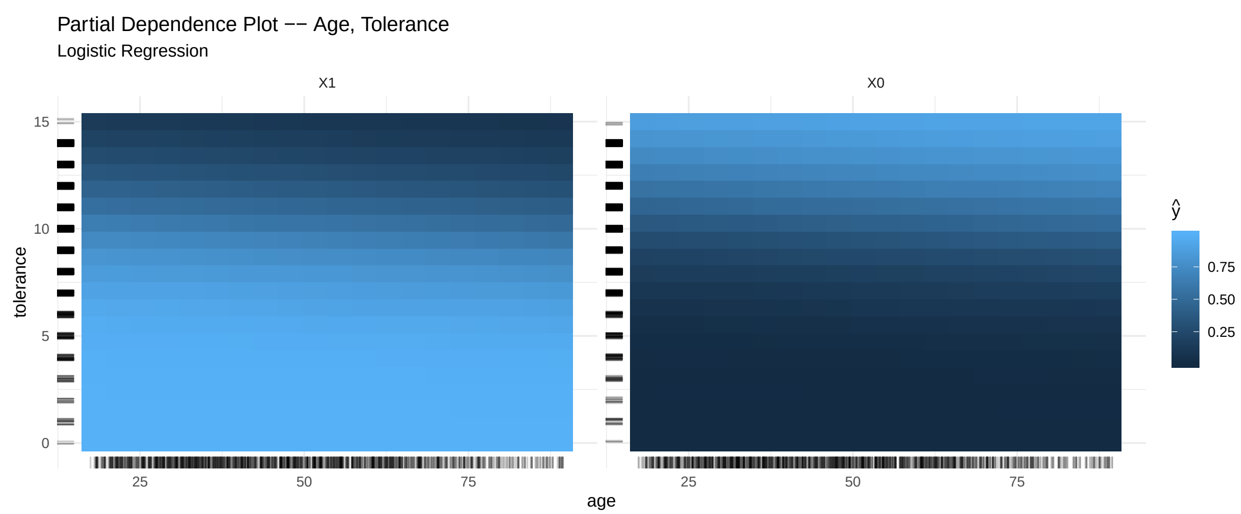
```
pdp_tolerance <- model_df_im$pdp_tol_obj[[1]]
pdp_tolerance %>% plot() + theme_minimal() +
  ggtitle("Partial Dependence Plot -- Tolerance",
          "Logistic Regression")
```



```
pdp_age <- model_df_im$pdp_age_obj[[1]]
pdp_age %>% plot() + theme_minimal() +
  ggtitle("Partial Dependence Plot -- Age",
          "Logistic Regression")
```




```
pdp_age_tolerance <- model_df_im$pdp_agetol_obj[[1]]
pdp_age_tolerance %>% plot() + theme_minimal() +
  ggtitle("Partial Dependence Plot -- Age, Tolerance",
          "Logistic Regression")
```



We see that the effect of `age` is marginal at best – the curve moves from 0.55 to 0.48 over the range of the variable. `tolerance`, on the other hand, seems to have a much greater effect on `colrac`, with low values of the `tolerance` variable (upto 11) associated with greater lenience towards racist professors.

The dominant effect of `tolerance` is evident in the two-variable PDP plot too – the change in $\hat{y}$ values seem to be largely along the `tolerance` axis.