IEOR 242: Applications in Data Analysis, Fall 2019

# Homework Assignment #3

October 4, 2019

**Problem 1:** (30 points)

Consider the algorithm for building a CART model in the case of regression. Following and expanding on the notation from class, suppose that our current tree, denoted by $T_{\text{old}}$, has $|T_{\text{old}}| = M$ terminal nodes/buckets. For each bucket $m = 1, \dots, M$, let:

1. $N_m$ denote the number of observations in bucket $m$ ,
2. $Q_m(T_{\text{old}})$ denote the value of the impurity function at bucket $m$ , and
3. $R_m$ denote the region in the feature space corresponding to bucket $m$ .

Also let $N$ be the overall total number of observations. Recall that, in the case of regression we have that:

$$Q_m(T_{\text{old}}) = \frac{1}{N_m} \sum_{i:x_i \in R_m} (y_i - \hat{y}_m)^2 \ ,$$

where $\hat{y}_m = \frac{1}{N_m} \sum_{i:x_i \in R_m} y_i$ is the mean response in bucket $m$.

Then the total impurity cost of the tree $T_{\text{old}}$ is defined as:

$$C_{\text{imp}}(T_{\text{old}}) = \sum_{m=1}^{M} N_m Q_m(T_{\text{old}}) \ .$$

Consider a potential split at the final bucket $M$ (we're using $M$ just for ease of notation), which results in a new tree $T_{\text{new}}$. This new tree has $|T_{\text{new}}| = M + 1$ terminal nodes/buckets, and for this new tree we let

1. $\tilde{N}_m$ denote the number of observations in bucket $m$ ,
2. $\tilde{Q}_m(T_{\text{new}})$ denote the value of the impurity function at bucket $m$ , and
3. $\tilde{R}_m$ denote the region in the feature space corresponding to bucket $m$ .

The total impurity cost of the tree $T_{\text{new}}$ is defined analogously as:

$$C_{\text{imp}}(T_{\text{new}}) = \sum_{m=1}^{M+1} \tilde{N}_m \tilde{Q}_m(T_{\text{new}}) \ .$$

Please answer the following:

*a*) (10 points) Let $\Delta = C_{\text{imp}}(T_{\text{old}}) - C_{\text{imp}}(T_{\text{new}})$ be the absolute decrease in total impurity resulting from the split. Derive a formula for $\Delta$ that can be computed locally at the bucket $M$, in other words it should only depend on the data points that fall in region $R_M$ in the original tree $T_{\text{old}}$. (Hint: we've discussed this concept in class, this question is asking for a more formal argument. You may assume that the two new buckets in $T_{\text{new}}$ resulting from the split are labeled as buckets $M$ and $M+1$ in $T_{\text{new}}$.)

*b*) (10 points) Show that $\Delta \geq 0$, hence splitting always reduces the total impurity cost. (Hint: you can use the fact that, given a sequence of real numbers $z_1, z_2, \ldots, z_n$, the mean $\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i$ is the minimizer of the function $\text{RSS}(z) = \sum_{i=1}^{n}(z_i - z)^2$)

*c*) (10 points) Let $R^2_{\text{old}}$ be the training set $R^2$ value for the model defined by $T_{\text{old}}$, and likewise let $R^2_{\text{new}}$ be the training set $R^2$ value for the model defined by $T_{\text{new}}$. Let $\text{SST} = \sum_{i=1}^{N}(y_i - \bar{y})^2$ be the total sum of squared errors, where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the overall mean. For a given value of the complexity parameter (cp) $\alpha \geq 0$, recall the modified cost function that is relevant in the pruning step:

$$C_\alpha(T) = C_{\text{imp}}(T) \ + \ \alpha \cdot \text{SST} \cdot |T|$$

Show that $C_\alpha(T_{\text{new}}) \leq C_\alpha(T_{\text{old}})$ if and only if $R^2_{\text{new}} - R^2_{\text{old}} \geq \alpha$. (Hence the choice of retaining a split if the increase in $R^2$ is at least $\alpha$ is equivalent to retaining a split if the modified cost function is smaller after the split.)

**Problem 2: Letter Recognition (Adapted from Bertsimas 22.3)** (70 points)

One of the most widespread applications of machine learning is to do optical character/letter recognition, which is used in applications from physical sorting of mail at post offices, to reading license plates at toll booths, to processing check deposits at ATMs. Optical character recognition by now involves very well-tuned and sophisticated models. In this problem, you will build a simple model that uses attributes of images of four letters in the Roman alphabet – A, B, P, and R – to predict which letter a particular image corresponds to.

In this problem, we have four possible labels of each observation, namely whether the observation is the letter A, B, P, or R. Hence this is a *multi-class classification* problem.

The file `Letters.csv` contains 3116 observations, each of which corresponds to a certain image of one of the four letters A, B, P and R. The images came from 20 different fonts, which were then randomly distorted to produce the final images; each such distorted image is represented as a collection of pixels, and each pixel is either "on" or "off". For each such distorted image, we have available certain attributes of the image in terms of these pixels, as well as which of the four letters the image is. These features are described in Table 1. Note that **feature selection is not**

**required** for this problem.

**NOTE: For the questions in this problem, provide your numerical answer as well as any necessary code, calculations, or explanations needed to justify your answer.**

**REMINDER: Please submit your R code as part of your submission on Gradescope.**

*a*) (25 points) To warm up, we'll start by predicting whether or not the letter is "B". To do so, first create a new variable called `isB` in your dataset, which takes value "Yes" if the letter is B, and "No" if the letter is not B. Assuming that you named your data frame `Letters`, this can be done by running the following command in your R console:

```
> Letters$isB = as.factor(Letters$letter == "B")
```

Set the seed using the following command:

```
> set.seed(456)
```

Then randomly split your dataset into a training set and a test set, putting 65% of the data in the training set. Just do a standard random split (do not use stratified sampling) with the following commands:

```
> train.ids = sample(nrow(Letters), 0.65*nrow(Letters))
  Letters.train = Letters[train.ids,]
  Letters.test = Letters[-train.ids,]
```

*i*) Before building any models, first consider a baseline method that always predicts the most frequent outcome, which is "not B". What is the accuracy of this baseline method on the test set?

*ii*) Construct a logistic regression model to predict whether or not the letter is a B, using the training set to build your model. (Remember to not use the the variable `letter` as one of the independent variables in any of the models in this part, since it is not a valid feature.) What is the accuracy of your logistic regression model on the test set, using a threshold of $p = 0.5$?

*iii*) What is the AUC of your logistic regression model?

*iv*) Construct a CART tree to predict whether or not a letter is a B, using the training set to build your model. Select the `cp` value for the tree through cross-validation, and explain how you did the cross-validation and how you selected the `cp` value. What is the accuracy of your CART model on the test set?

*v*) Now construct a Random Forest model to predict whether or not the letter is a B. Just leave the Random Forest parameters at their default values (i.e., leave them out of the function call). What is the accuracy of this Random Forest model on the test set?

*vi*) Compare the accuracy of your logistic regression, CART, and Random Forest models. Which one performs best on the test set? For this application, which do you think is more important: interpretability or accuracy?

*b*) (45 points) Now let us move on to the original problem of interest, which is to predict whether or not a letter is one of the four letters A, B, P or R. The variable in the dataset which you will try to predict is `letter`.

*i*) In a multi-class classification problem, a simple baseline model is to predict (for every observation) the most frequent class over all of the classes. For this problem, what does the baseline method predict, and what is the baseline accuracy on the test set?

*ii*) Construct an LDA model to predict `letter`, using the training set to build your model. (Throughout remember not to use the variable `isB` in the model, as it is not a valid feature.) What is the test set accuracy of your LDA model?

*iii*) Now construct a CART model to predict `letter` using the training data. Again, select the `cp` value for the tree through cross-validation. Again, explain how you did the cross-validation and how you selected the `cp` value. What is the test set accuracy of your CART model?

*iv*) Next build a model for predicting `letter` using "vanilla" bagging of CART models. This can be achieved, for example, by setting `mtry` equal to the total number of features (i.e., set $m = p$) in the `randomForest` package in R. What is the test set accuracy of your bagging model?

*v*) Now build a Random Forest model, and this time use cross-validation to select the `mtry` value for the Random Forests method. Explain how you did the cross validation and how you selected the `mtry` value. What is the test set accuracy for your Random Forest model?

*vi*) Let us now apply boosting to this problem. To apply boosting to a multi-class classification problem, one must specify the distribution as "multinomial" in the `gbm` function call (for binary classification, the distribution should be set to "adaboost" or "bernoulli"). Train a "gradient boosting machine (gbm)" model to predict `letter` using the training data. Set the interaction depth to 10, run the method for 3300 iterations, and leave all other parameters at their default values. (The values for the interaction depth and the number of iterations were obtained via a 5-fold cross validation procedure.) What is the test set accuracy of your gradient boosting model?

HINT: After training your gradient boosting model, set the `type` parameter equal to `"response"` when you call the `predict` function. This returns a matrix with rows corresponding to test set data points and with 4 columns, where each column represents the probability estimate (as predicted by the gradient boosting model) for each of the 4 classes. Suppose that we used the variable name `blah` for this matrix. To convert this matrix to a vector of class predictions, run the following code:

```
> pred = apply(blah, 1, which.max)
  pred = factor(pred, levels = c(1,2,3,4), labels = c("A", "B", "P", "R"))
```

The first line computes the column number that corresponds to the largest probability in each row, and the second line converts these numbers back to the values “A”, “B”, “P”, and “R”.

(Note: If you are interested in more details about boosting, I highly recommend perusing Chapter 10 of “The Elements of Statistical Learning” by Hastie, Tibshirani, and Friedman.)

*vii*) Compare the accuracy of your LDA, CART, bagging, Random Forest, and boosting models for this problem. Which one would you recommend for this problem? Is your choice different from the model you recommended in part (*a*)? Why or why not?

Table 1: Variables in the dataset `Letters`.

| **Variable** | **Description** |
|---|---|
| `letter` | The letter that the image corresponds to (A, B, P or R). |
| `xbox` | The horizontal position of where the smallest box enclosing the letter shape begins. |
| `ybox` | The vertical position of where the smallest box enclosing the letter shape begins. |
| `width` | The width of this smallest box. |
| `height` | The height of this smallest box. |
| `onpix` | The total number of "on" pixels in the character image. |
| `xbar` | The mean horizontal position of all of the "on" pixels. |
| `ybar` | The mean vertical position of all of the "on" pixels. |
| `x2bar` | The mean squared horizontal position of all of the "on" pixels in the image. |
| `y2bar` | The mean squared vertical position of all of the "on" pixels in the image. |
| `xybar` | The mean of the product of the horizontal and vertical position of all of the "on" pixels in the image. |
| `x2ybar` | The mean of the product of the squared horizontal position and the vertical position of all of the "on" pixels. |
| `xy2bar` | The mean of the product of the horizontal position and the squared vertical position of all of the "on" pixels. |
| `xedge` | The mean number of edges (the number of times an "off" pixel is followed by an "on" pixel, or the image boundary is hit) as the image is scanned from left to right, along the whole vertical length of the image. |
| `xedgeycor` | The mean of the product of the number of horizontal edges at each vertical position and the vertical position. |
| `yedge` | The mean number of edges as the images is scanned from top to bottom, along the whole horizontal length of the image. |
| `yedgexcor` | The mean of the product of the number of vertical edges at each horizontal position and the horizontal position. |