Examples with the `lime` R package applied to a Random Forest Model fit to the Iris Data

Katherine Goode

Last Updated: 2018/04/18

1 Overview

This document contains examples that apply the LIME algorithm to a random forest model fit to the iris data. These examples are both meant to better understand how the R package `lime` implements the LIME algorithm and to explore whether the explanations returned by `lime` are reasonable. The first example verifies how the ridge regression function is fit within the `lime` package. The next example considers plots of the perturbation created by `lime` in order to visually inspect if the default method in LIME does a good job of selecting the important local predictors. The last example compares the fit of the ridge regression model when different feature selection methods are used.

2 Set Up for Examples

2.1 Loading Packages, Sourcing Code, and Setting a Seed

The code below loads the R packages that will be used and sources the code from the `lime` R package and functions I have written. My functions include my version of the `lime` explain.data.frame, which exports the perturbations and numerified perturbations along with the explanation table. I have also written functions for plotting the perturbations. Additionally, a seed is set since some of the procedures in these examples involve randomization. This will allow for replication of the examples.

```
# Load libraries
library(R.utils)
library(caret)
library(tidyverse)
library(knitr)

# Source in code from the lime R package
sourceDirectory("../../../LIME R Package/R")

# Source in the functions I have written
sourceDirectory("../../my functions")

# Set a seed for any randomization that occurs in the example to maintain consistency
set.seed(174920)
```

2.2 Training and Testing Data

The code below splits the iris data into training and testing parts. To do this, two cases are selected randomly from within each of the three species of irises. These cases are put in the testing data, and the other cases are made into the training data. For both the training and testing data, the features are put in a data frame, and the response variables are put in separate vectors.

```r
# Randomly select two cases from within each of the three species of irises
selected <- c(sample(1:50, 2), sample(51:100, 2), sample(101:151, 2))

# Split up the features of the data into training and testing parts
iris_train <- iris[-selected, 1:4]
iris_test <- iris[selected, 1:4]

# Grab the response variables from both the training and testing data
iris_train_response <- iris[[5]][-selected]
iris_test_response <- iris[[5]][selected]
```

2.3 Random Forest Model

A random forest model is fit to the training data using the `caret` package with all of the default settings. All four features (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`) from the iris data are put in the random forest model. The model is then used to make predictions on the testing data. The predictions are compared to the actual observed species for the test cases. The table below contains the observed species and predicted species from the test data, and we see that the models gets all of the predictions correct. However, we will use `lime` to see which features were key in making these predictions to see if the predictions are based on features that make sense.

```r
# Random forest model run on the iris data
iris_model <- train(iris_train, iris_train_response, method = 'rf')

# Predictions made using the random forest model on the testing data
iris_model_predict <- predict(iris_model, iris_test)

# Matrix of observed and predicted values
iris_test_obs_pred <- matrix(c(iris_test_response, iris_model_predict),
                             ncol = 2, byrow = FALSE)

# Specify the column names
colnames(iris_test_obs_pred) <- c("Observed", "Predicted")

# Print the table
kable(iris_test_obs_pred, align = 'c')
```

Observed	Predicted
1	1
1	1
2	2
2	2
3	3
3	3

2.4 LIME Functions

The LIME algorithm is applied to the model now. First, the `lime` function from the R package is run on the iris training data and random forest model. All of the default settings are used, which are as follows.

- `bin_continuous = TRUE`: If set to 'TRUE', the continuous variables will be binned when making the explanations. If they are not binned, then perturbations will be obtained by simulating from a normal distribution with sample mean and standard deviation associated with the feature.
- `bins = 4`: The number of bins to divide the continuous variables into. The defualt is 4 bins.
- `quantile_bins = TRUE`: If set to 'TRUE', the bins will be be based on 'n_bins' quantiles. Otherwise, the bins will be spread evenly over the range. of the training data

Next, my version of the `explain` function for dataframes is applied to the testing data and the `lime` "explainer" object. The number of labels for the response variable is set to 3 (for the three species of irises), the number of features to include in the explanations is set to 2, and the remaining options are set to the default values. These default values are as follows.

- `n_permutations = 5000`: The number of perturbations generated for each feature.
- `feature_select = 'auto'`: This is the feature selection method for choosing the number of features specified. In the vignette for the `lime` package, Thomas describes the options as follows.
  - `none`: Use all features for the explanation. Not advised unless you have very few features.
  - `forward selection`: Features are added one by one based on their improvements to a ridge regression fit of the complex model outcome.
  - `highest weights`: The m features with highest absolute weight in a ridge regression fit of the complex model outcome are chosen.
  - `lasso`: The m features that are least prone to shrinkage based on the regularization path of a lasso fit of the complex model outcome is chosen.
  - `tree`: A tree is fitted with log2(m) splits, to use at max m features. It may possibly select less.
  - `auto`: Uses forward selection if m <= 6 and otherwise highest weights.

  The default is set to `auto`.
- `dist_fun = 'gower'`: The distance function to be used for determining how close a perturbation is to the test point. The default is set to 'gower'.
- `kernel_width = NULL`: If 'dist_fun' is not set to 'gower', then this is the kernel width that will be used in the distance function. If set to 'NULL', which is the default, the default method within `lime` is used.

The output from the `my_explain.data.frame` contains a list with three objects. The first contains the `lime` explanations, the second is the perturbations, and the thrid is the numerified perturbations. These three objects are separated and given the names of `iris_explanation`, `iris_perturb`, and `iris_perturb_numerified`, respectively.

```
# Run the lime function on the iris data and random forest model with 4 bins
iris_explainer <- lime.data.frame(iris_train, iris_model)

# Run the explain function for data frames on the iris data
iris_explanation_list <- my_explain.data.frame(iris_test, iris_explainer,
                                               n_labels = 3, n_features = 2)

# Create a separate data frame for the explanations
iris_explanation <- iris_explanation_list$explanations

# Create a separate data frame for the perturbations
iris_perturb <- iris_explanation_list$perturb
```

```r
# Create a separate data frame for the numerified perturbations
iris_perturb_numerified <- iris_explanation_list$perturb_numerified
```

3 Examples

3.1 Verifying the Ridge Regression Procedure

I wanted to verify that I understood how the ridge regression model was being fit by the `lime` package. To start, I selected the pertrubations and response probability of setosa associated with the first test case. Then I fit a ridge regression model with a response variable of the probability of the perturbation being setosa and predictor variables of the simulated perturbations for the two features selected by feature selection. I set $\lambda = 0.001$, which is what `lime` does. Then I extracted the coefficients from the ridge regression model and compared these to the coefficients from `lime`. These are included in the table below. I discovered that they did not agree, which told me that my initial understanding of how `lime` fit the ridge regression model was not correct.

```r
# Select only explanations associated with the first case and outcome of setosa
iris_explain_sub <- iris_explanation %>%
  filter(label == "setosa") %>%
  slice(1:2)

# Put perturbations for the selected features from the first test case into
# a data frame
iris_perturb_sub <- iris_perturb %>%
  filter(Test_Case == iris_explain_sub$case) %>%
  select(iris_explain_sub$feature, setosa, Weight)

# Fit the ridge regression model as I understand it...
rr1 <- glmnet(as.matrix(iris_perturb_sub[,1:2]), iris_perturb_sub$setosa,
              weights = iris_perturb_sub$Weight, alpha = 0, lambda = 0.001)

# Matrix with coefficients from my ridge regression model and from the lime
# ridge regression model
coefs1 <- matrix(c(rr1$a0, rr1$beta[1], rr1$beta[2],
                   iris_explain_sub$model_intercept[1],
                   iris_explain_sub$feature_weight),
                 nrow = 3, byrow = FALSE)

# Label the rows and columns of the matrix
rownames(coefs1) <- c("beta0", "beta1", "beta2")
colnames(coefs1) <- c("My Ridge Regression", "LIME Ridge Regression")

# Print the coefficients in a table
kable(coefs1, align = 'c')
```

	My Ridge Regression	LIME Ridge Regression
beta0	0.8758308	0.2446452
beta1	-0.0162788	-0.0050768
beta2	-0.1136287	0.4302248

I remembered that `lime` creates numerified pertrubations, which assign a 0 or a 1 to the observation based on whether or not it falls into the same bin as the observed case. I wondered if `lime` used these values as the features. I fit a new ridge regression model using the numerified perturbations from the selected features as the predictor variables in the model. I compared the coefficients from my model to the one fit by `lime`. These are shown in the table below. They agree this time! Thus, `lime` is using the numerified perturbations when it fits the ridge regression model.

```
# Select the numerified perturbations associated with test case 1, and select
# only the chosen features
iris_perturb_numerified_sub <- iris_perturb_numerified %>%
  filter(Test_Case == iris_explain_sub$case) %>%
  select(iris_explain_sub$feature)

# Fit the ridge regression again with the numerified perturbations instead of
# the regular perturbations
rr2 <- glmnet(as.matrix(iris_perturb_numerified_sub), iris_perturb_sub$setosa,
              weights = iris_perturb_sub$Weight, alpha = 0, lambda = 0.001)

# Matrix with coefficients from my ridge regression model and from the lime
# ridge regression model
coefs2 <- matrix(c(rr2$a0, rr2$beta[1], rr2$beta[2],
                   iris_explain_sub$model_intercept[1],
                   iris_explain_sub$feature_weight),
                 nrow = 3, byrow = FALSE)

# Label the rows and columns of the matrix
rownames(coefs2) <- c("beta0", "beta1", "beta2")
colnames(coefs2) <- c("My Ridge Regression", "LIME Ridge Regression")

# Print the coefficients in a table
kable(coefs2, align = 'c')
```

	My Ridge Regression	LIME Ridge Regression
beta0	0.2446452	0.2446452
beta1	-0.0050768	-0.0050768
beta2	0.4302248	0.4302248

I do not understand why `lime` has chosen to use the numerified perturbations instead of the regular perturbations when fitting the ridge regression models. Someone did ask about this on the GitHub page for `lime`. The question was a follows.

> For continuous variables with bin_continuous=TRUE, it seems like the data set used for the regression (and for the computation of weights) consists of zeroes and ones only, where the value in row i for variable j is 1 if the bin for this variable in this row is equal to the bin for the same variable in the observation vector and 0 otherwise. Is this correct? If yes, doesn't one then discard a lot of information, since there obviously is a larger distance between e.g. bins 1 and 5 than there is between bins 1 and 2?

Thomas answered the question as follows.

> This is correct - the explanation looks into whether the bin significantly affects the model. I guess the reason that it is done this way is to ease interpretation at the cost of a potentially worse local

fit..

The link to where this disucssion can be found is https://github.com/thomasp85/lime/issues/68. This answer helps a bit in understanding why this was done, but I do not feel like it is satistfactory. I would still like to look more into this. I have two main questions that I would like to try to answer.

- How can the model coefficients be interpred since the numerified perturbations are used?
- How are the 'lime' explanations affected by using the numerified perturbations instead of the regular perturbations?

3.2 Plots of Perturbations with Default `lime` Settings

In order to better understand how `lime` works, I wanted to create some visualizations of the perturbations created by `lime`. I was interested in visualizing the distributions of the pertrubations and how they compared to the original data. I was also interested in looking at the relationship between the pertrubations and the probability that the perturbation fell in a specific species response category. This is the relationship that the ridge regression that `lime` fits is trying to capture and use to determine which features are most influnential in determining the predictions made by the model.

To access these visualizations, I created two functions. The function `plot_perturbs` plots the top two features chosen by the default settings in `lime` for both the perturbations and the original data. The perturbations are colored by the response that the model predicts for the perturbation. This plot allows one to visualize the regions of the feature space that the model predicts to be a certain response. I was hoping that this type of plot would help to identify whether the two chosen variables lead to a clear region in which the model predicts one way or the other. More details about what the plots shows will be described in Section @ref(plotperturb).

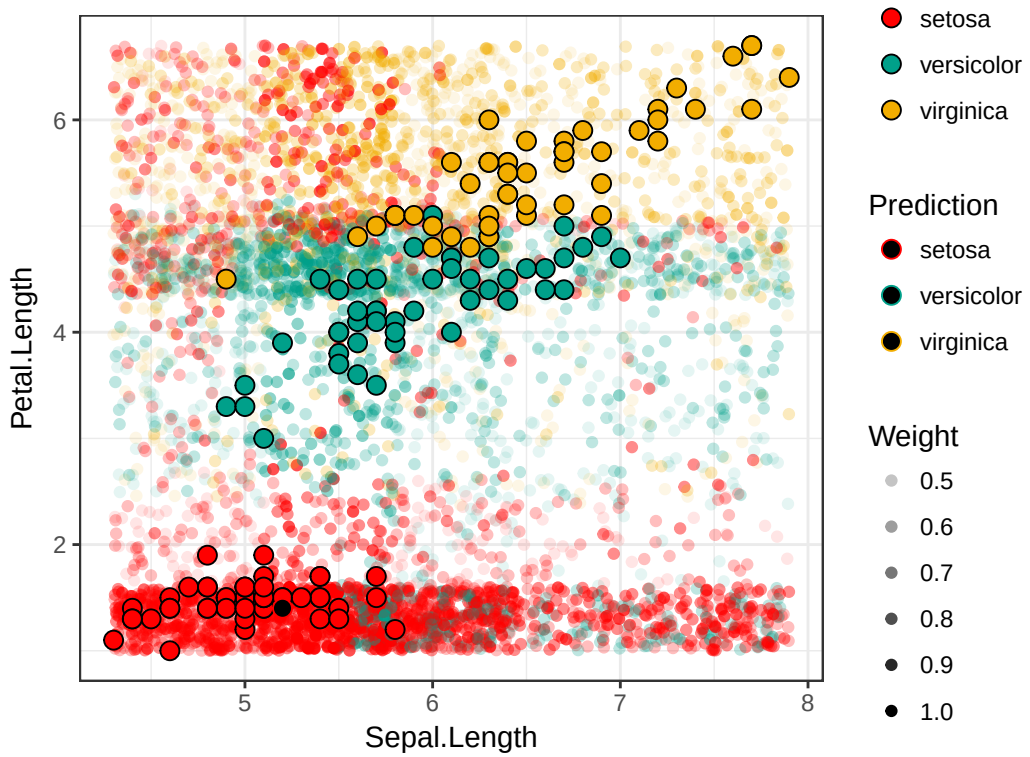
The other function, `plot_probs`, creates two plots corresponding to the top two features chosen by the default settings in `lime`. The plot shows the probability that a perturbation will be the associated response on the y axis and the value of the feature on the x-axis. These plots were meant to visualize whether there is a linear relationship between the probability, which is used as the response in the ridge regression, and the feature value. However, keep in mind that the ridge regression uses the numerified value of the feature instead of the raw value. More details about the plot will be included in Section @ref(plotprob)

3.2.1 Plots of Perturbations and Original Data

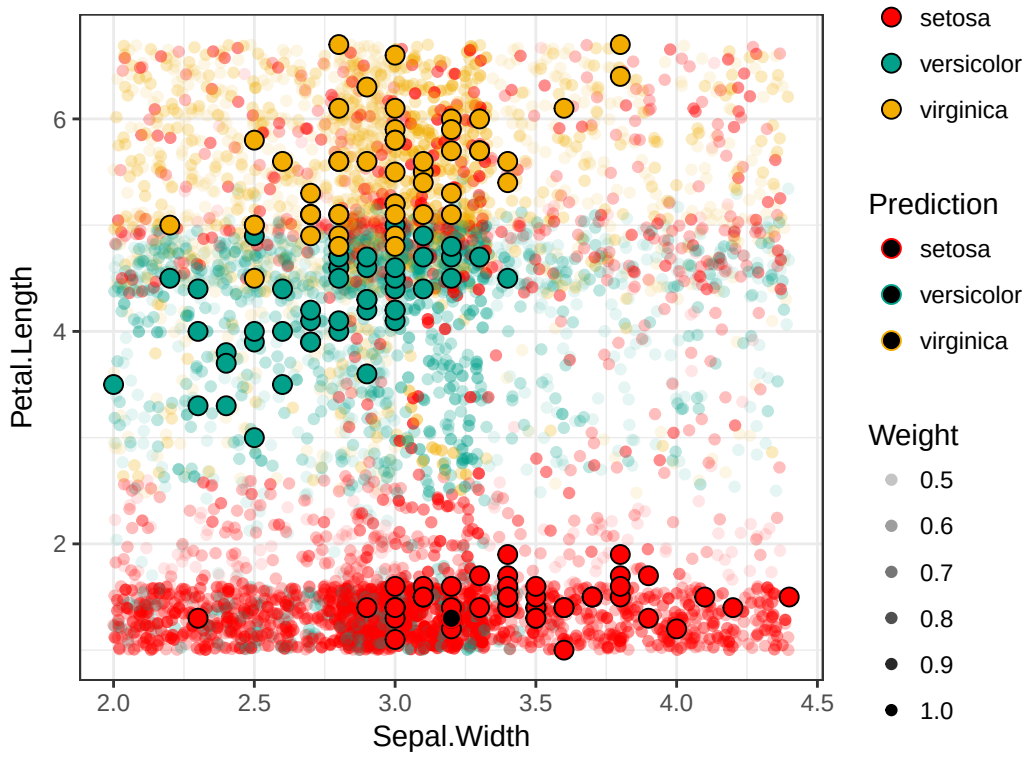
Each plot created using `plot_perturbs` corresponds to one observation (case) in the test data. This point is represented by a black dot in the plot. The ring of color around the black dot represents the species that the random forest predicts the test case to be. The larger dots represent the observed training data. The color of the point represents the true species associated with the observation. The smaller points in the plots represent the perturbations created by `lime`. The color of the small point represents the species that the random forest predicts the perturbation to be. The shade of the color of the small points is the weight assigned to the perturbation by `lime`, which signifies how "close" or how "similar" the perturbation is to the test case of interest. Darker shades indicate that the perturbation is "close" to the test case. The perturbations with higher weights will be more influential in the ridge regression model that is fit to determine which features are important. The variable on the x-axis is the feature selected by the default settings in `lime` as the most important feature in the predictive model. The variable on the y-axis is the second most important feature selected by the default settings in `lime`.

The plot shown below corresponds to the first observation in the `Iris` test data. This point falls in the cluster of training data points that belong to the species setosa. The model predicts that this test point has a species of setosa, and the model makes the right prediction.

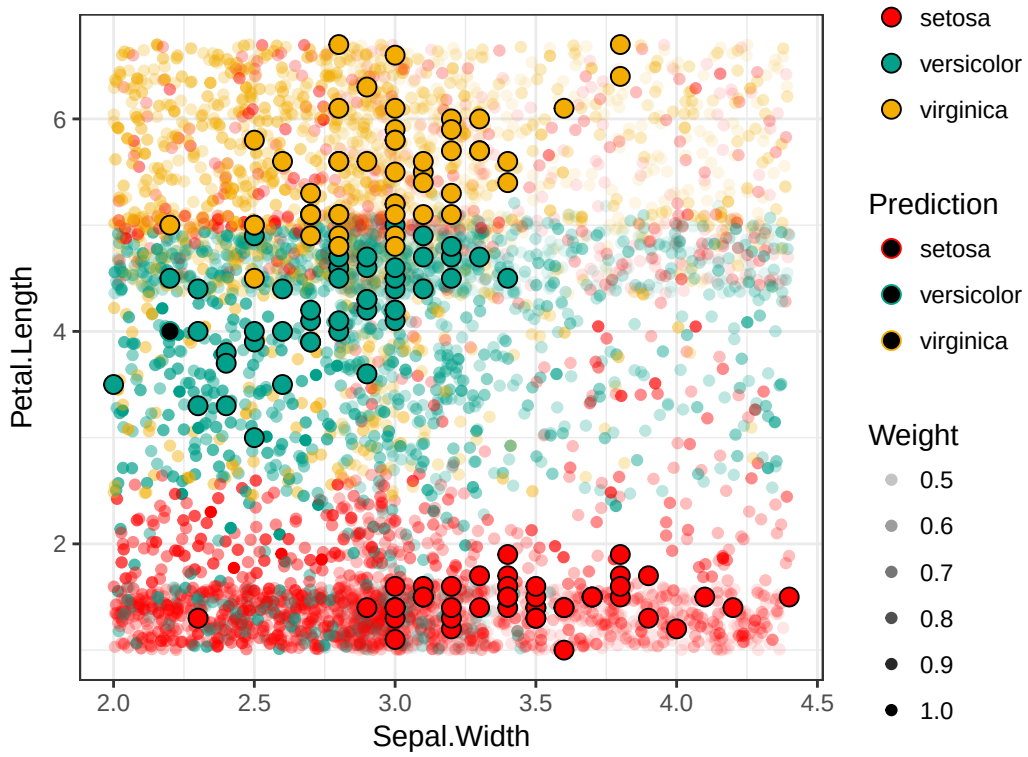
```
# Test Case 1: Plot of the selected features perturbations with predictions
# and observed training data
plot_perturbs(train = iris_train,
              response = iris_train_response,
              perturb = iris_perturb,
              explain = iris_explanation,
              test_case = unique(iris_explanation$case)[1],
              resp_category = 'setosa',
              separate = FALSE)
```



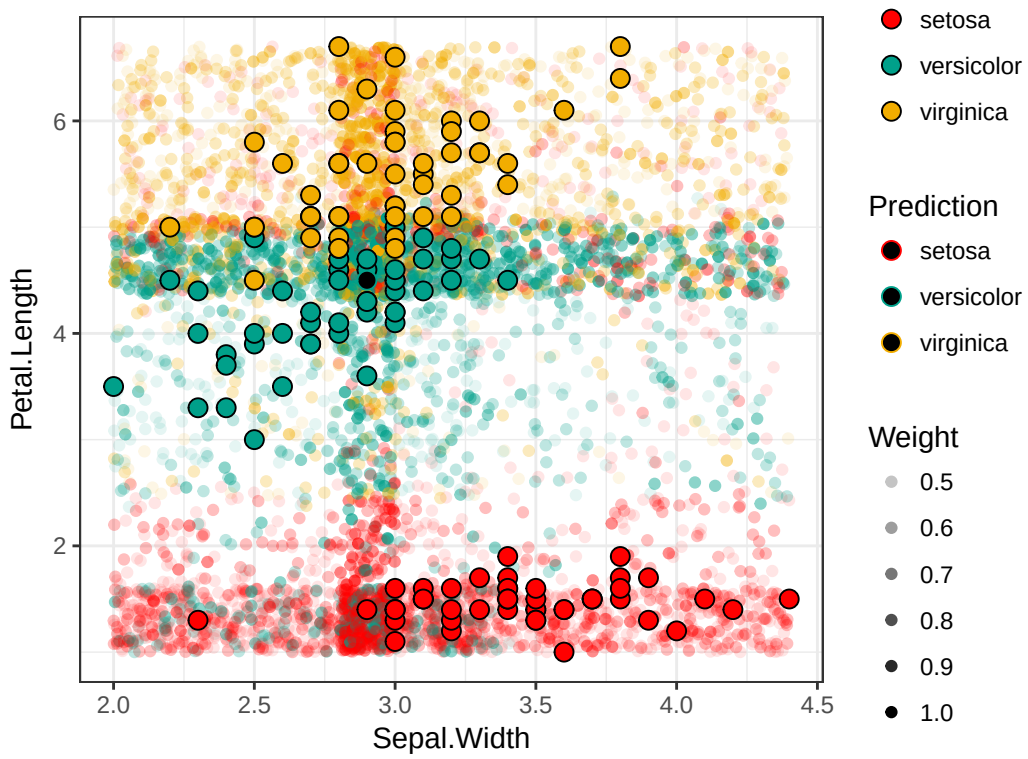

```
# Test Case 2: Plot of the selected features perturbations with predictions
# and observed training data
plot_perturbs(train = iris_train,
              response = iris_train_response,
              perturb = iris_perturb,
              explain = iris_explanation,
              test_case = unique(iris_explanation$case)[2],
              resp_category = 'setosa',
              separate = FALSE)
```



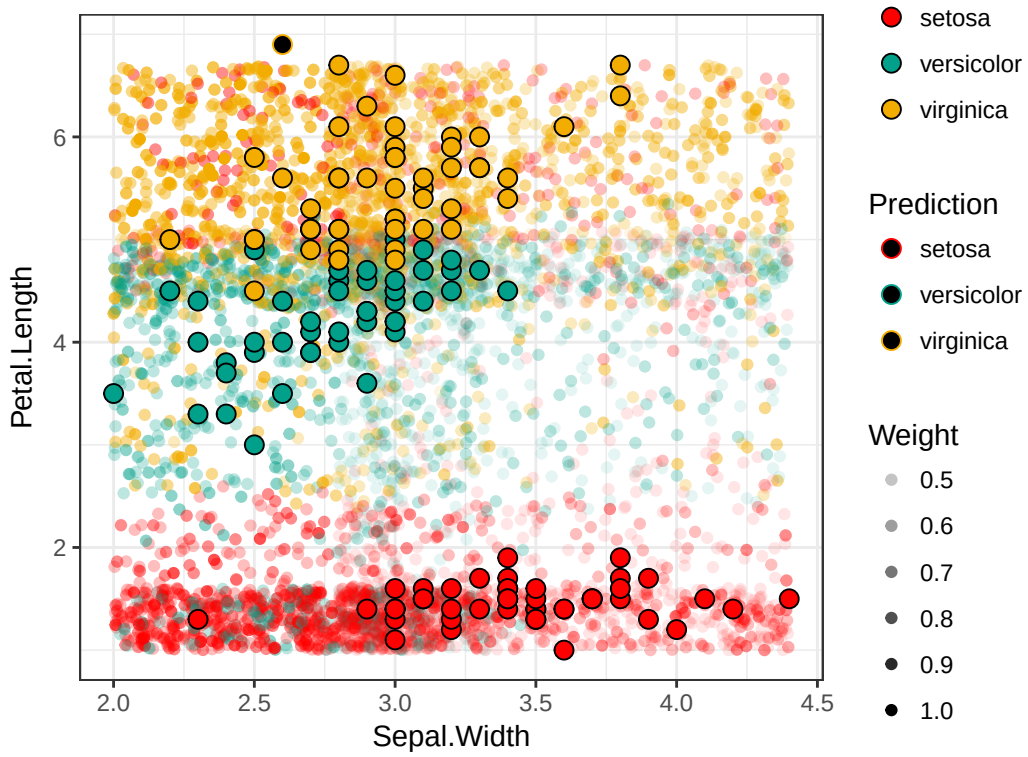

```
# Test Case 3: Plot of the selected features perturbations with predictions
# and observed training data
plot_perturbs(train = iris_train,
              response = iris_train_response,
              perturb = iris_perturb,
              explain = iris_explanation,
              test_case = unique(iris_explanation$case)[3],
              resp_category = 'versicolor',
              separate = FALSE)
```



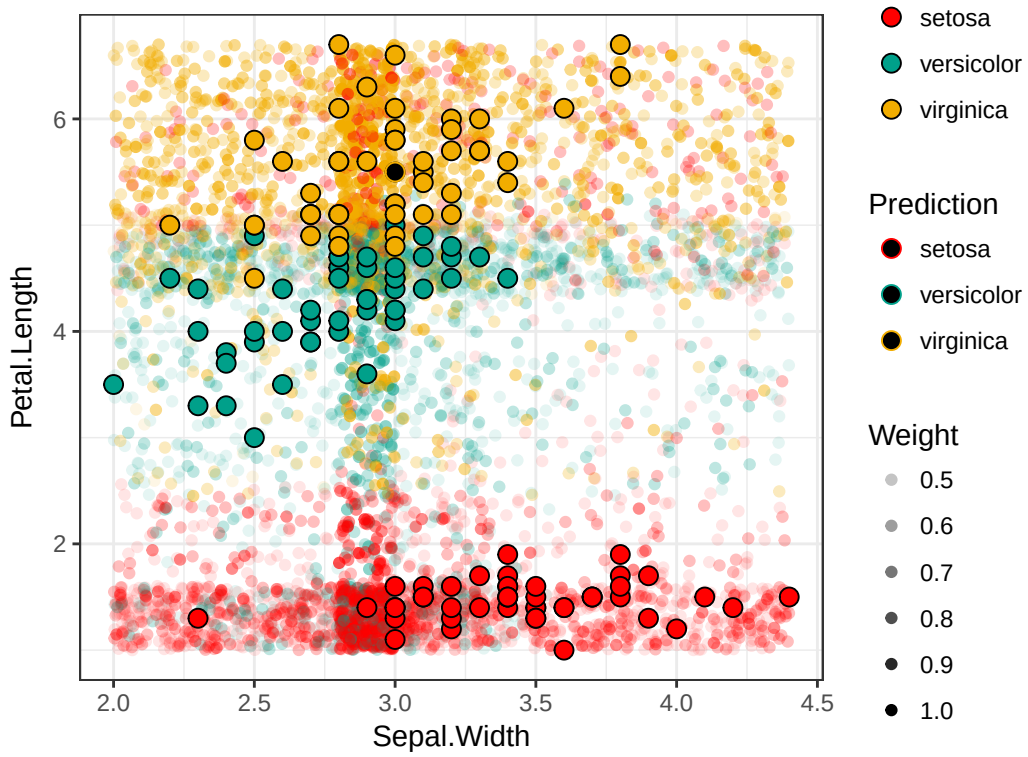

```
# Test Case 4: Plot of the selected features perturbations with predictions
# and observed training data
plot_perturbs(train = iris_train,
              response = iris_train_response,
              perturb = iris_perturb,
              explain = iris_explanation,
              test_case = unique(iris_explanation$case)[4],
              resp_category = 'versicolor',
              separate = FALSE)
```




```
# Test Case 5: Plot of the selected features perturbations with predictions
# and observed training data
plot_perturbs(train = iris_train,
              response = iris_train_response,
              perturb = iris_perturb,
              explain = iris_explanation,
              test_case = unique(iris_explanation$case)[5],
              resp_category = 'virginica',
              separate = FALSE)
```

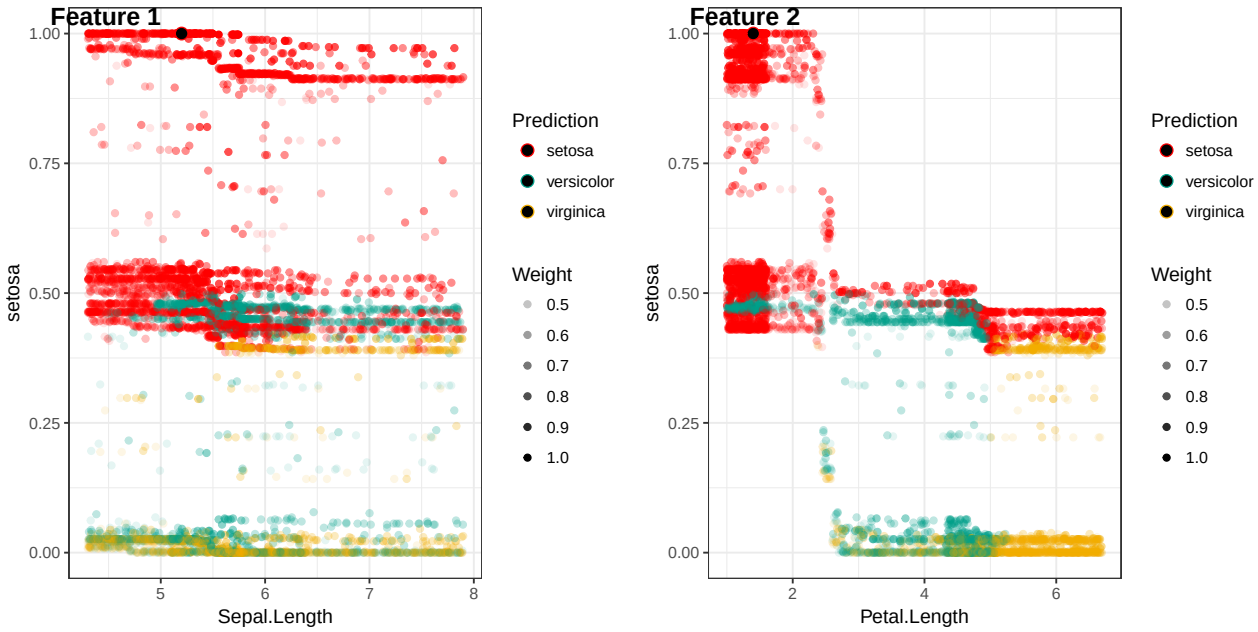



```
# Test Case 6: Plot of the selected features perturbations with predictions
# and observed training data
plot_perturbs(train = iris_train,
              response = iris_train_response,
              perturb = iris_perturb,
              explain = iris_explanation,
              test_case = unique(iris_explanation$case)[6],
              resp_category = 'virginica',
              separate = FALSE)
```

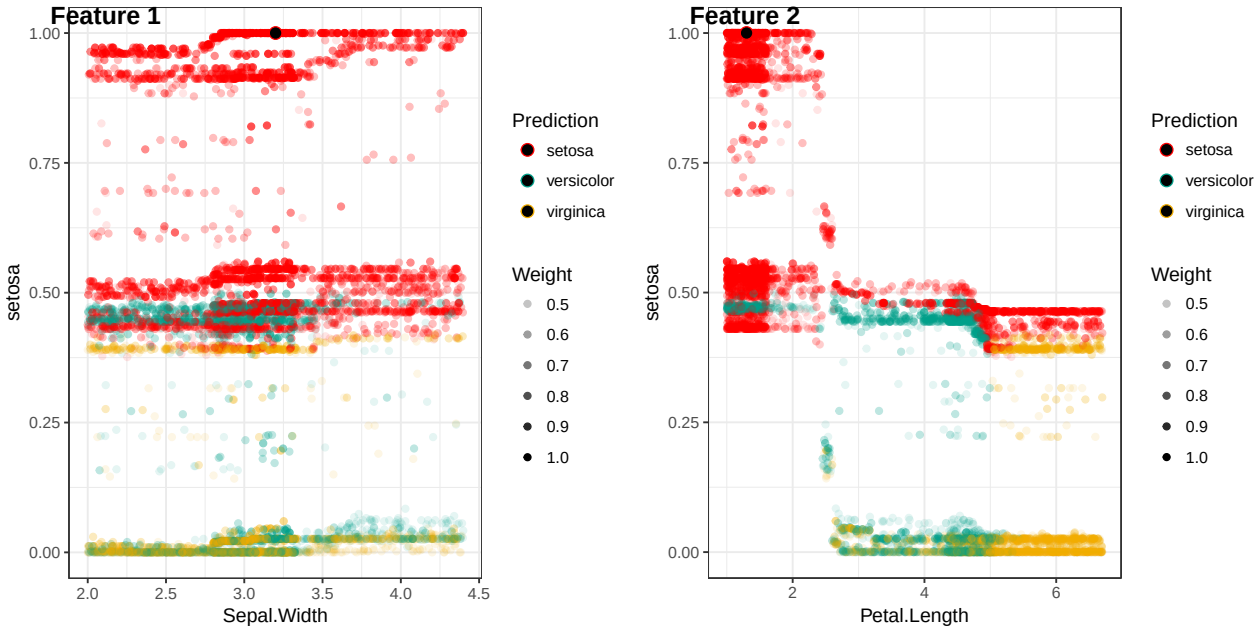


3.2.2 Plots of Probabilities and Perturbations

```
# Test Case 1: Plots of the probabilities versus the selected features of the
# perturbations with predictions and observed training data
plot_probs(train = iris_train,
           response = iris_train_response,
           perturb = iris_perturb,
           explain = iris_explanation,
           test_case = unique(iris_explanation$case)[1],
           resp_category = 'setosa')
```

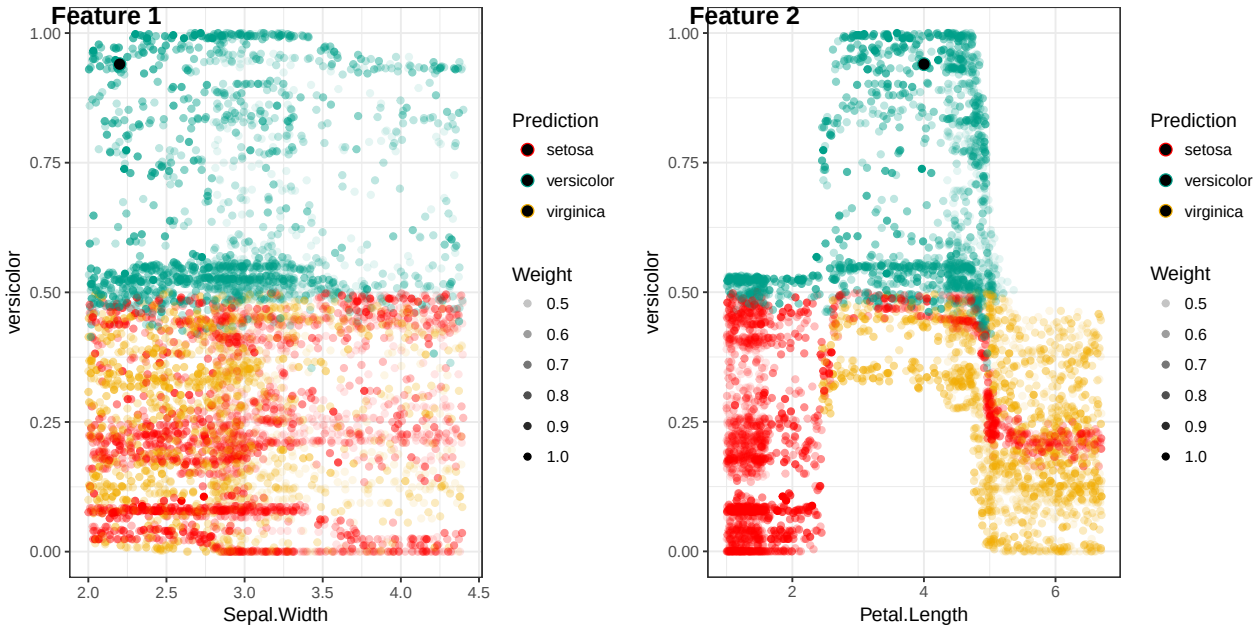



```
# Test Case 2: Plots of the probabilities versus the selected features of the
# perturbations with predictions and observed training data
plot_probs(train = iris_train,
           response = iris_train_response,
           perturb = iris_perturb,
           explain = iris_explanation,
           test_case = unique(iris_explanation$case)[2],
           resp_category = 'setosa')
```

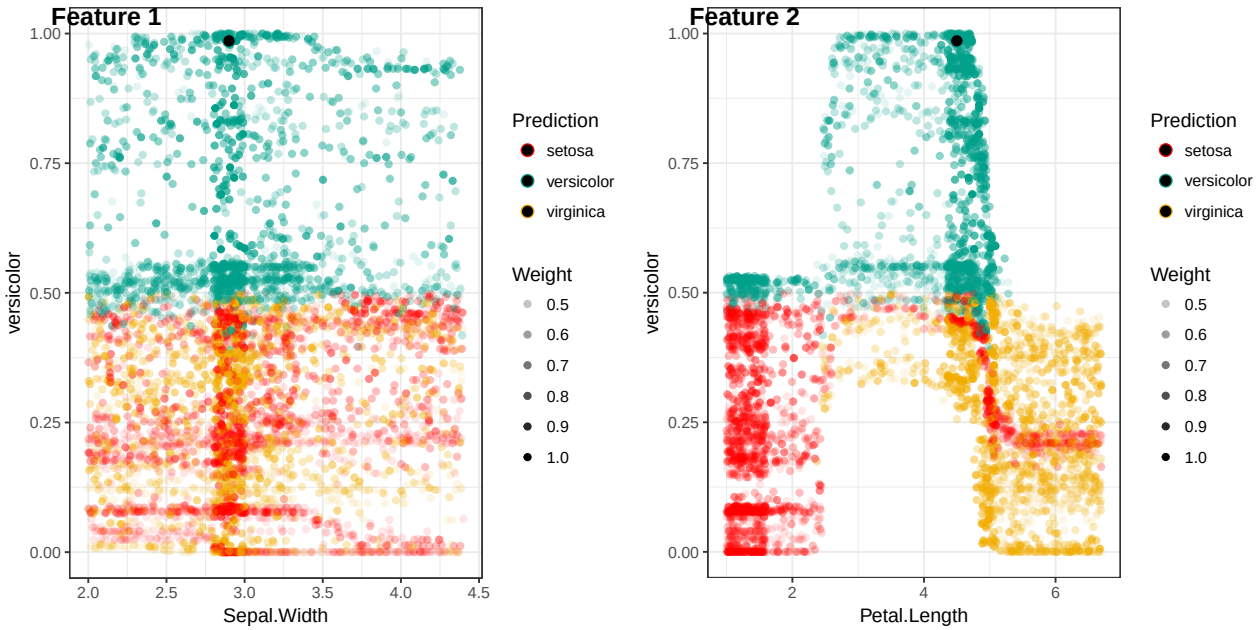



```
# Test Case 3: Plots of the probabilities versus the selected features of the
# perturbations with predictions and observed training data
plot_probs(train = iris_train,
           response = iris_train_response,
```

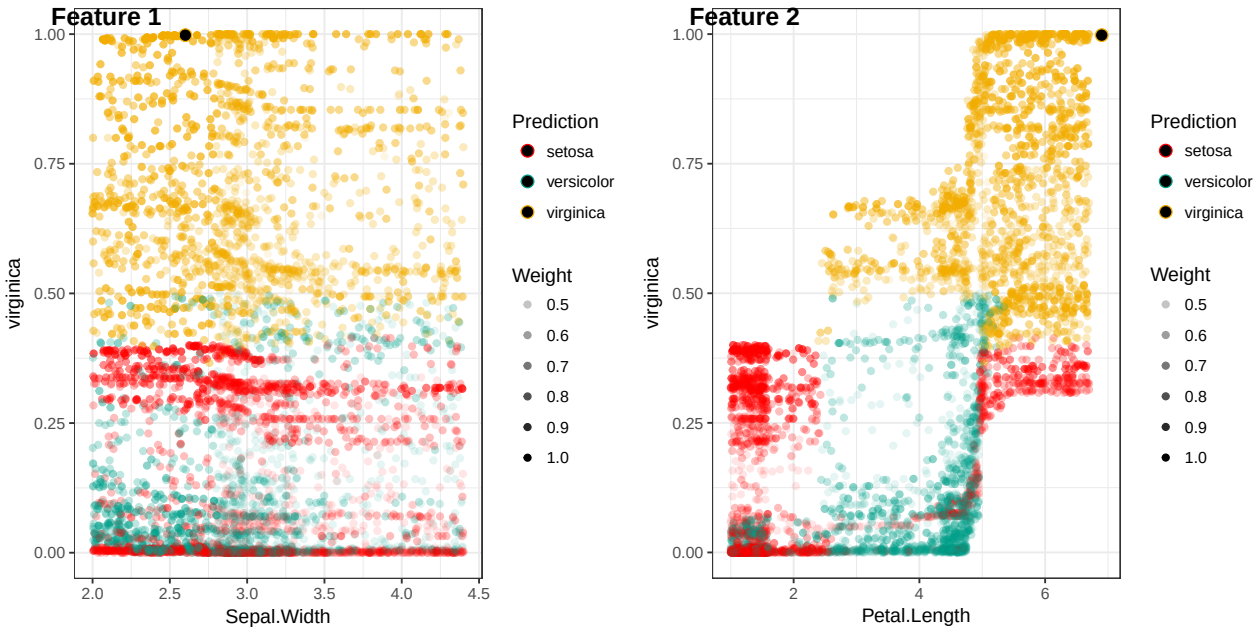
```
            perturb = iris_perturb,
            explain = iris_explanation,
            test_case = unique(iris_explanation$case)[3],
            resp_category = 'versicolor')
```




```
# Test Case 4: Plots of the probabilities versus the selected features of the
# perturbations with predictions and observed training data
plot_probs(train = iris_train,
            response = iris_train_response,
            perturb = iris_perturb,
            explain = iris_explanation,
            test_case = unique(iris_explanation$case)[4],
            resp_category = 'versicolor')
```

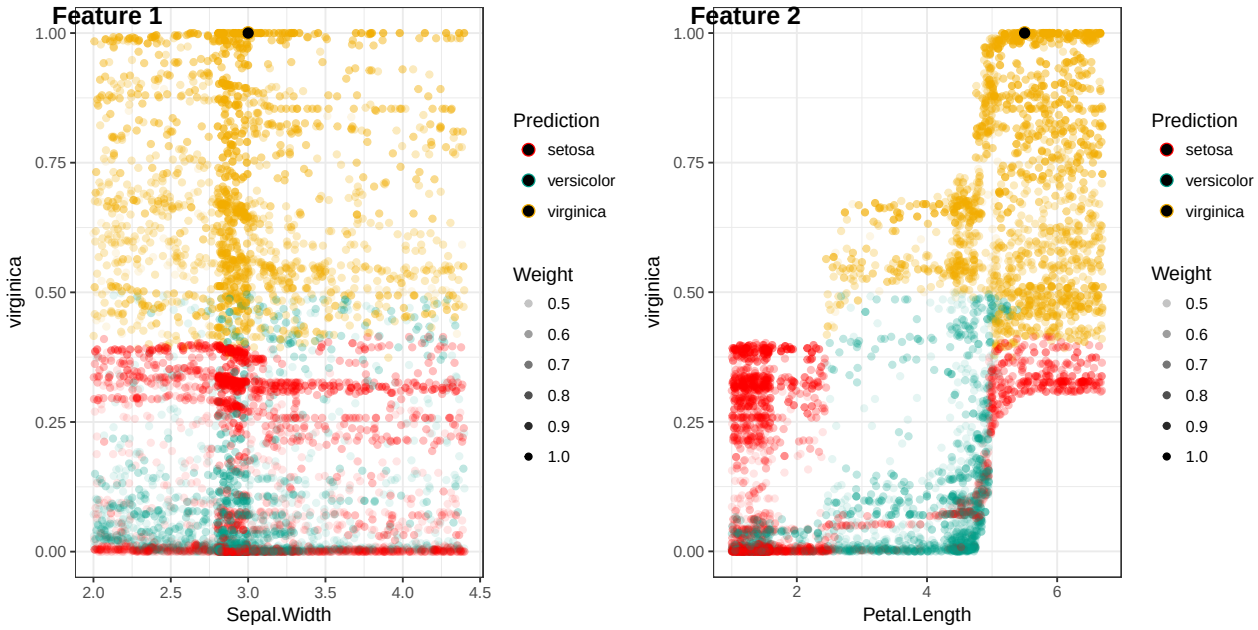



```
# Test Case 5: Plots of the probabilities versus the selected features of the
# perturbations with predictions and observed training data
plot_probs(train = iris_train,
           response = iris_train_response,
           perturb = iris_perturb,
           explain = iris_explanation,
           test_case = unique(iris_explanation$case)[5],
           resp_category = 'virginica')
```




```
# Test Case 6: Plots of the probabilities versus the selected features of the
# perturbations with predictions and observed training data
plot_probs(train = iris_train,
           response = iris_train_response,
```

```
            perturb = iris_perturb,
            explain = iris_explanation,
            test_case = unique(iris_explanation$case)[6],
            resp_category = 'virginica')
```



3.3 Comparing Feature Selection Methods in `lime`

Since the plots above led me to believe that LIME was not doing a great job of selecting the important features, I wanted to compare the feature selection methods. I noticed that the R^2 value seemed low for the ridge regressions that `lime` was fitting, and I was curious to know if a different feature selection method would lead to a higher R^2 value and more reasonable features chosen by `lime`. I also wanted to compare the R^2 values bewteen ridge regressions fit using the numerified perturbations (like `lime` does) and fit using the actual perturbations. The first section below considers the numerified perturbations, and the the following section considers the actual pertrubations.

3.3.1 Feature Selection with Numerified Perturbations

I ran my version of the `explain` function with all six options for feature selection. I fit the ridge regression models using the original code from `lime` that uses the numerified perturbations as the predictor variables in the model.

```
## Auto Selection
iris_auto <- my_explain.data.frame(iris_test, iris_explainer,
                                   n_labels = 3, n_features = 2,
                                   feature_select = 'auto')
iris_explain_auto <- iris_auto$explanations

## Forward Selection
iris_forward <- my_explain.data.frame(iris_test, iris_explainer,
                                      n_labels = 3, n_features = 2,
```

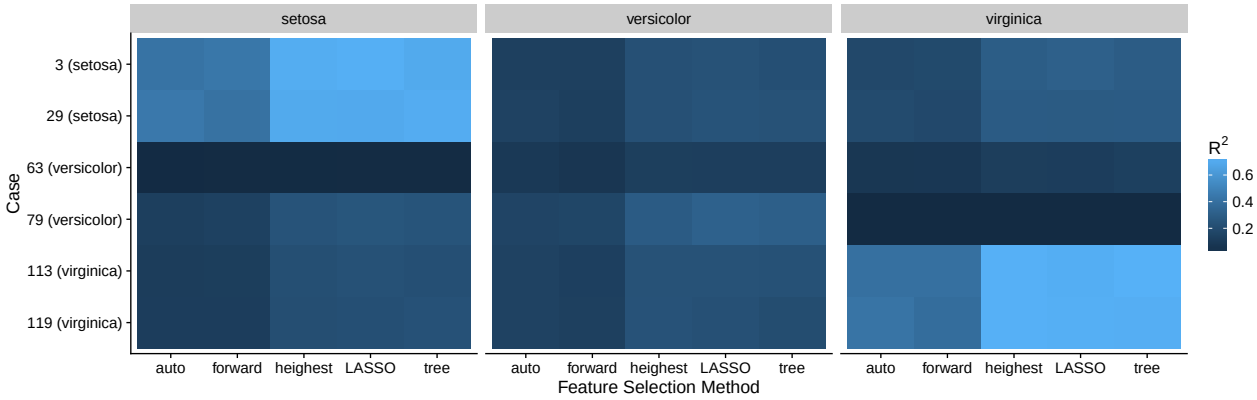
```
                                        feature_select = 'forward_selection')
iris_explain_forward <- iris_forward$explanations

## Heighest Weights
iris_heighest <- my_explain.data.frame(iris_test, iris_explainer,
                                       n_labels = 3, n_features = 2,
                                       feature_select = 'highest_weights')
iris_explain_heighest <- iris_heighest$explanations

## LASSO
iris_lasso <- my_explain.data.frame(iris_test, iris_explainer,
                                    n_labels = 3, n_features = 2,
                                    feature_select = 'lasso')
iris_explain_lasso <- iris_lasso$explanations

## Tree
iris_tree <- my_explain.data.frame(iris_test, iris_explainer,
                                   n_labels = 3, n_features = 2,
                                  feature_select = 'tree')
iris_explain_tree <- iris_tree$explanations
```

I extracted the R^2 values from all of the ridge regression models. This includes models for all 6 cases and 3 species labels. Thus, there were a total 18 R^2 values per feature selection method.

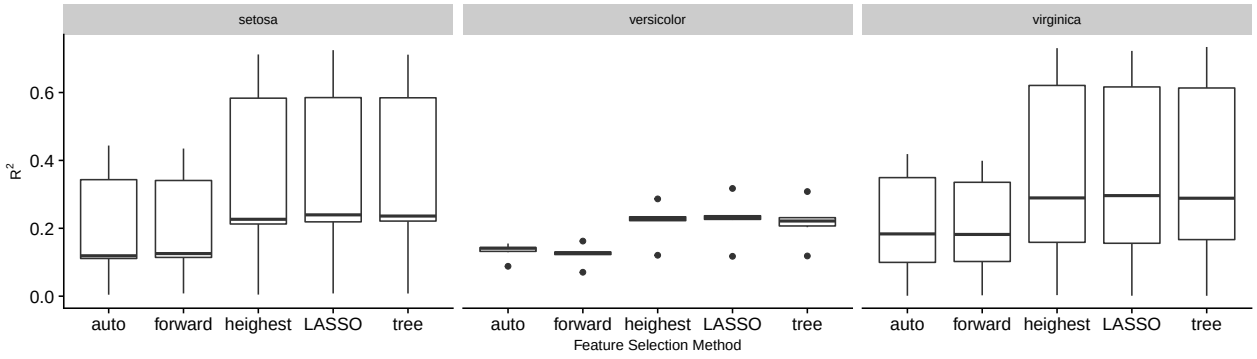
```
# Create a data frame with all of the R2 values from the different feature selection
# methods
r2 <- data.frame(case = iris_explain_auto$case,
                 label = iris_explain_auto$label,
                 label_prob = iris_explain_auto$label_prob,
                 auto = iris_explain_auto$model_r2,
                 forward = iris_explain_forward$model_r2,
                 heighest = iris_explain_heighest$model_r2,
                 LASSO = iris_explain_lasso$model_r2,
                 tree = iris_explain_tree$model_r2)

# Only keep the unique rows
r2 <- unique(r2)

# Gather the dataset
r2_gathered <- r2 %>% gather(method, r2, 4:8)

# Reorder the levels of the case variable
r2_gathered$case <- factor(r2_gathered$case,
                           levels(r2_gathered$case)[c(2, 1, 6, 5, 3, 4)])

# Rename the levels of the case variable
levels(r2_gathered$case) <- c("119 (virginica)", "113 (virginica)",
                              "79 (versicolor)", "63 (versicolor)",
                              "29 (setosa)", "3 (setosa)")
```

The plot below shows the R^2 values for each method associated with a case and facetted by a species label.

Within each case and within each species label, the R^2 values are the highest for the feature selection methods of highest, LASSO, and tree. This suggests that these three feature selection methods lead to the best linear fit with the iris perturbations. Within the species facets, the cases which have a true species corresponding to the label have the highest R^2 values of the cases for both setosa and virginica. This suggests that the cases with a model fit with a response variable corresponding to the true species of the case leads to a higher R^2 for both setosa and virginica. It appears that it is hard to find a linear relationship for any feature selection method with the species of versicolor.



The boxplots below show the distribution of R^2 values within a feature selection method for each of the species. This further shows that the R^2 values tend to be higher for the feature selection methods of highest, LASSO, and tree for all three methods. These plots also highlight how the R^2 values are much more variable for the species of setosa and virginica than the species of versicolor. All of the R^2 values for versicolor are low.

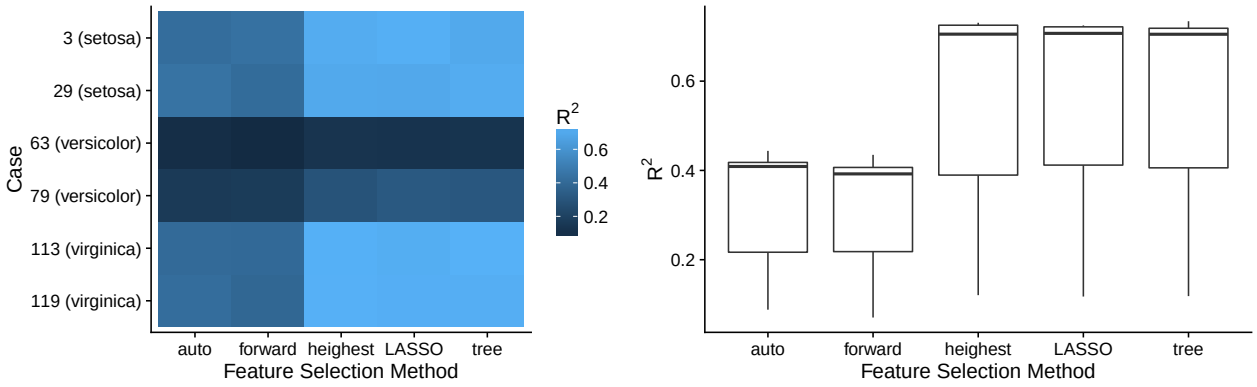


I also wanted to see how the R^2 values compared for just the true species labels. I subsetted the data to only contain the R^2 values from models fit with a response of the probability from the true species.

```
# Create a data frame with only the true species labels
r2_true <- r2 %>%
  group_by(case) %>%
  filter(label_prob == max(label_prob))

# Gather the dataset
r2_true_gathered <- r2_true %>% gather(method, r2, 4:8)
```

```
# Reorder the levels of the case variable
r2_true_gathered$case <- factor(r2_true_gathered$case,
                                levels(r2_true_gathered$case)[c(2, 1, 6, 5, 3, 4)])

# Rename the levels of the case variable
levels(r2_true_gathered$case) <- c("119 (virginica)", "113 (virginica)",
                                   "79 (versicolor)", "63 (versicolor)",
                                   "29 (setosa)", "3 (setosa)")
```

Again, a tile plot shows the R^2 value for each feature selection method within a case. The feature selection methods of heighest, LASSO, and tree have the highest R^2 values, especially for the cases with a true species of setosa and virginica. The boxplots show a similar message.



3.3.2 Feature Selection with Actual Perturbations

Next, I ran my version of the `explain` function with all six options for feature selection. This time, I added an option in my `explain` function that allowed the ridge regression to be fit using the actual pertrubations as the predictor variables instead of the numerified perturbations.

```
## Auto Selection (with actual perturbations)
iris_auto_v2 <- my_explain.data.frame(iris_test, iris_explainer,
                                      n_labels = 3, n_features = 2,
                                      feature_select = 'auto',
                                      rr_predictors = "perturb")
iris_explain_auto_v2 <- iris_auto_v2$explanations

## Forward Selection (with actual perturbations)
iris_forward_v2 <- my_explain.data.frame(iris_test, iris_explainer,
                                         n_labels = 3, n_features = 2,
                                         feature_select = 'forward_selection',
                                         rr_predictors = "perturb")
iris_explain_forward_v2 <- iris_forward_v2$explanations

## Heighest Weights (with actual perturbations)
iris_heighest_v2 <- my_explain.data.frame(iris_test, iris_explainer,
                                          n_labels = 3, n_features = 2,
```

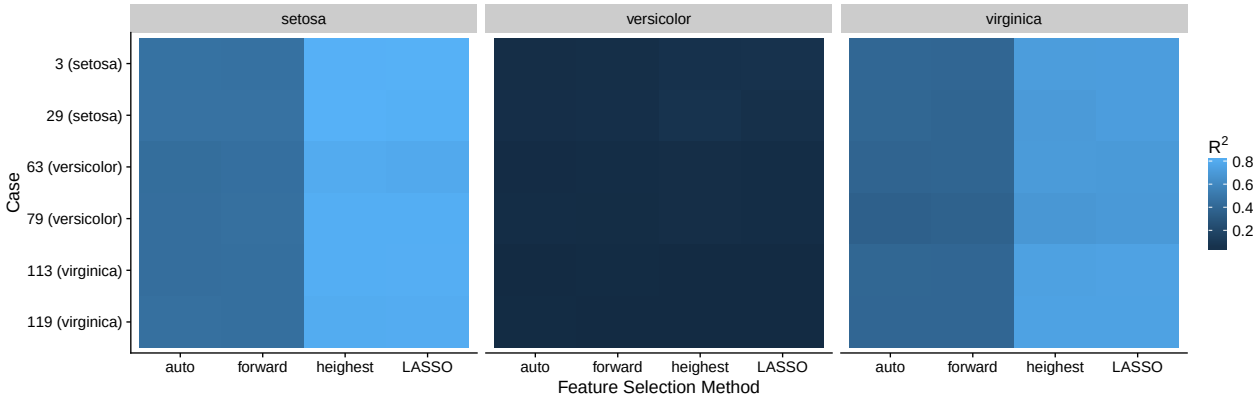
```
                                          feature_select = 'highest_weights',
                                          rr_predictors = "perturb")
iris_explain_heighest_v2 <- iris_heighest_v2$explanations

## LASSO (with actual perturbations)
iris_lasso_v2 <- my_explain.data.frame(iris_test, iris_explainer,
                                       n_labels = 3, n_features = 2,
                                       feature_select = 'lasso',
                                       rr_predictors = "perturb")
iris_explain_lasso_v2 <- iris_lasso_v2$explanations

## Tree (with actual perturbations)
iris_tree_v2 <- my_explain.data.frame(iris_test, iris_explainer,
                                      n_labels = 3, n_features = 2,
                                      feature_select = 'tree',
                                      rr_predictors = "perturb")
iris_explain_tree_v2 <- iris_tree_v2$explanations
```

I extracted the R^2 values from the ridge regressions fit with a response of one of the species and the pertrubations as the predictors for each case and each of the feature selection methods. For some reason, the tree method is not creating a model for the same number of cases as the other methods. I am not sure why this is. For now, I have exculded the tree feature selection method, and I will try to fix this later.

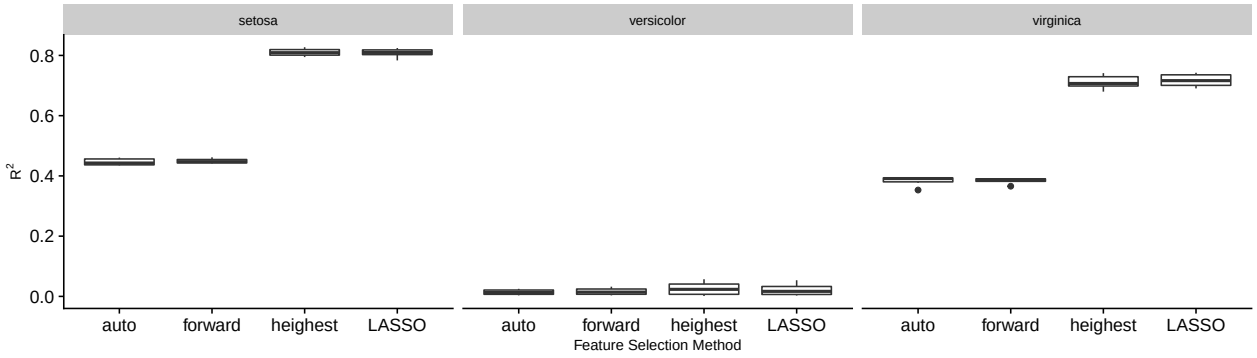
```
# Create a data frame with all of the R2 values from the different feature selection
# methods
r2_v2 <- data.frame(case = iris_explain_auto_v2$case,
                    label = iris_explain_auto_v2$label,
                    label_prob = iris_explain_auto_v2$label_prob,
                    auto = iris_explain_auto_v2$model_r2,
                    forward = iris_explain_forward_v2$model_r2,
                    heighest = iris_explain_heighest_v2$model_r2,
                    LASSO = iris_explain_lasso_v2$model_r2)
                    #tree = iris_explain_tree_v2$model_r2)

# Only keep the unique rows
r2_v2 <- unique(r2_v2)

# Gather the dataset
r2_v2_gathered <- r2_v2 %>% gather(method, r2, 4:7)

# Reorder the levels of the case variable
r2_v2_gathered$case <- factor(r2_v2_gathered$case,
                           levels(r2_v2_gathered$case)[c(2, 1, 6, 5, 3, 4)])

# Rename the levels of the case variable
levels(r2_v2_gathered$case) <- c("119 (virginica)", "113 (virginica)",
                              "79 (versicolor)", "63 (versicolor)",
                              "29 (setosa)", "3 (setosa)")
```

The tile plots below depict the R^2 values from the ridge regression models for each case and feature selection method within each of the three species responses. With the species of setosa and virginica, the R^2 values

are higher for the methods of highest and LASSO. For the species of versicolor, all of the R^2 values are very low and similar. It appears to be hard to find a linear relationship for versicolor. Unlike the R^2 values from the ridge regressions fit with the numerified perturbations, the R^2 values do no vary much between cases within a species and feature selection method. I am not sure why this is.



The box plots below make it clear that there is much less variation in R^2 values within a species and feature selection method. Thess plots also highlight that when the ridge regression models are fit to the perturbations, there is a clear difference in R^2 values between the methods of auto and forward and the methods of highest and LASSO for the species of setosa and virginica. The boxplots from the numerified pertrubations showed overlap between the feature selection methods.



I also decided to look at the R^2 values only from models fit to a case with the true species. The code below extracts these cases from the dataframe with all of the R^2 values.

```
# Create a data frame with only the true species labels
r2_v2_true <- r2_v2 %>%
  group_by(case) %>%
  filter(label_prob == max(label_prob))

# Gather the dataset
r2_v2_true_gathered <- r2_v2_true %>% gather(method, r2, 4:7)

# Reorder the levels of the case variable
r2_v2_true_gathered$case <- factor(r2_v2_true_gathered$case,
```

```
                          levels(r2_v2_true_gathered$case)[c(2, 1, 6, 5, 3, 4)])

# Rename the levels of the case variable
levels(r2_v2_true_gathered$case) <- c("119 (virginica)", "113 (virginica)",
                                      "79 (versicolor)", "63 (versicolor)",
                                      "29 (setosa)", "3 (setosa)")
```

A tile plot and box plots of the R^2 values are shown below. The tile plot highlights the difference between the methods of auto and forward and the methods of highest and LASSO for the species of setosa and virginica. There is little difference between the method within the species of versicolor. The boxplots show more variation when the true cases from the three species are combined.