Nonlinear Methods: Classification

Signal Data Science

Previously, we considered nonlinear *regression* techniques. Next, we'll move to briefly covering nonlinear *classification* techniques, allowing us to improve upon logistic regression via more sophisticated and general methods. The theory of some of the classification techniques used in this assignment is discussed in *Notes on Classification Theory*; feel free to back to those notes as you work through the exercises below.

The field of nonlinear techniques is *very* vast and wide. As with the nonlinear regression assignment, you should be constantly referring to textbooks and online resources to clarify details and to improve your own understanding and intuition. The lesson below is designed to *guide* you in the correct directions, and the goal is not to speedily finish the exercises but rather to use them as prompts for your curiosity and as sources of further directions of intellectual exploration.

Getting started

We will use a combination of simulated and real data to explore classification techniques in R.

First, we can write some utility functions to generate our simulated data. For simplicity, we will restrict attention to the 2-dimensional unit square ($x \in [0,1]$, $y \in [0,1]$).

- Write a function `lin_pair(m, b, label)` that takes in integers `m`, `b`, and `label` and returns two numerics `c(x, y)` satisfying the following criteria:
  - Both `x` and `y` should be between 0 and 1.
  - If `label` is set to 1, then `y` must be greater than `m*x + b`. Conversely, if `label` is set to -1, then `y` must be less than `m*x + b`.

You can think of the function `lin_pair(m, b, label)` as picking a point in the unit square *uniformly randomly* from the region falling above or below the line $y = mx + b$.

- Plot the points you get from running `lin_pair()` multiple times with `m=2` and `b=-0.3`. Verify that the resulting plot resembles the image below.

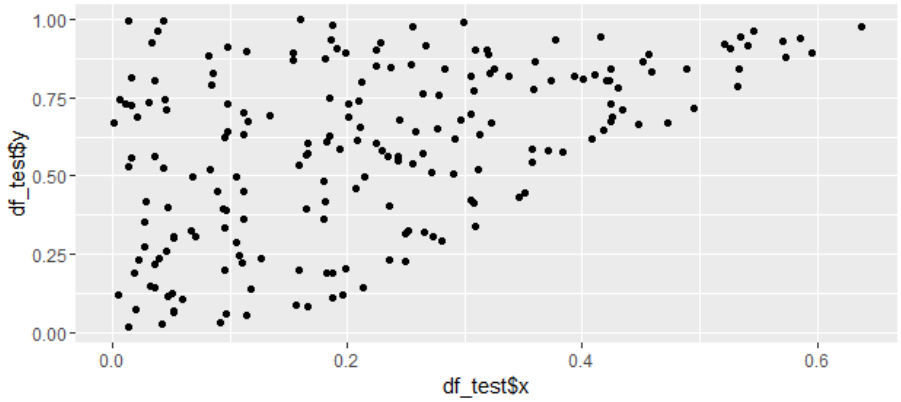

Figure 1: The results of plotting 200 randomly generated points in the unit square falling above the line $y = 2x - 0.3$.

We can also extend the same methodology to generate data falling above or below a *quadratic* curve (which will be more difficult to classify).

- Write a function `quad_pair(a, b, c, label)` which does the same thing for the quadratic parabola $y = a(x - b)^2 + c$. Verify that it works as desired by plotting the results.

We will also use the *Iris* flower data set, which was introduced by the statistician Ronald Fisher expressly for the purpose of demonstrating the usage of linear discriminant analysis for classification of different *Iris* species (*Iris setosa*, *Iris virginica*, and *Iris versicolor*).

Figure 2: A specimen of *Iris versicolor*.

The *Iris* dataset is a default variable in base R.

- Set `df_iris = iris` to copy the *Iris* dataset to a different variable. Call `?iris` and read the *Description* section of the documentation.
- There are 4 different numeric variables in the *Iris* dataset, yielding $\binom{4}{2} = 6$ different pairs of these variables. Plot each pair of variables on a scatterplot with the points colored according to their species. For example, the code for plotting sepal width against sepal length should look like this:

  ```
  ggplot(df_iris,
    aes(Sepal.Length, Sepal.Width, color=Species))
    + geom_point()
  ```

  Which pairs of species are linearly separable (*i.e.*, in the 4-dimensional space of the 4 numeric variables, which pairs of species can be perfectly separated from each other with a 3-dimensional hyperplane)? Which pairs of species are not?

Other methods

k-Nearest Neighbors and decision tree methods previously discussed (random forests and gradient boosted trees)

k-Nearest Neighbors

k-Nearest Neighbors classification works in precisely the way one would expect it to work: for any given point to classify, the algorithm looks at the nearest *k* points in the training set and finds the most common class among those *k* points. That class is the prediction result.

Although *k*-NN works well when there is enough data to capture, at a very fine-grained level, all of the detailed structure in the data, prediction is slow (because it depends on searching through the *entire* training set for every single prediction) and *k*-NN has difficulty with higher-dimensional data (because distance metrics stop working well at very high dimensions).

Decision tree methods