Overfitting Issues

Lisa Hall

University of the Cumberlands

Statistics for Data Science (MSDS-531-A01) - First Bi-term

Dr. Mark D. Mosley

February 18, 2025

**Overfitting Issues**

Data scientists need to be familiar with overfitting. Overfitting happens when noise is included in the data (Bruce & Bruce, 2019). This project explores some examples of overfitting, the risks of overfitting, and how to reduce overfitting potentially. Reducing overfitting is important for accurate models and interpreting new data.

**Understanding Overfitting**

Figure 1 illustrates a data set with two distributions. The “+” points are uniformly distributed, and the “o” points are typically distributed and clustered in specific areas. By utilizing this model, we can show the different types of model complexity and how to reduce the risk of overfitting. Figure 1 is a scatterplot that may not explicitly show overfitting but can represent where overfitting could occur. A too-complex model applied to this dataset could be at risk of overfitting; if a model is too intricate, it could struggle with differentiating noise. If a model were to fit every outlier in the dataset, it could be too specific for new data. There is a “Goldilocks” or just the right level of complexity we must endeavor to achieve when creating models (Matloff, n.d.).

**Figure 1 Binary Scatterplot**

A diagram of a training set

AI-generated content may be incorrect.

Figure 2 illustrates two decision trees, one with eleven leaf nodes and another with twenty-four leaf nodes. If we were to apply a model to these trees, we could encounter some obstacles that would need consideration. The goal of creating a model for decision trees is to predict a target variable by learning simple decision rules; these can be inferred from data features (Scikit-learn Developers, n.d.). Overly complex trees, like the twenty-four-node tree in Figure 2, are a good example of a tree that may benefit from pruning or removing unnecessary nodes or risk overfitting. A problem, however, with less complex decisions could be missing specific data points or misclassifying certain data points.

**Figure 2 Decision Trees**

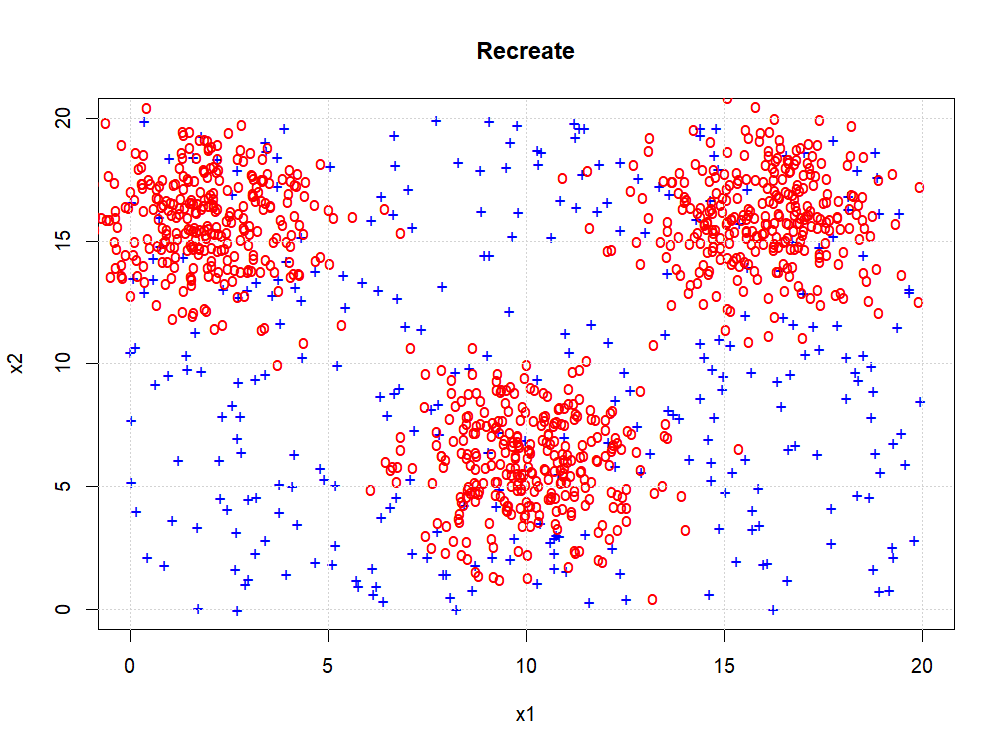
**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Data Simulation**

To further investigate, let us emulate the data in Figure 1 using R and compare it to another simulated dataset. Appendix A shows how we can generate a similar dataset. First, we set the cluster centers of the dataset at similar points; I chose (2,16), (16,16), and (10, 6). We can also create a list for the standard deviation; for the x coordinates, I chose 1.5, and for the y coordinates, I chose a standard deviation 2. Next, we used a runif() function to generate uniform data for the “+” points, using set.seed() for reproducibility (R Core Team, 2024). Then, we use the rnorm() function to generate normally distributed data for the “o” points, ensuring that we set the cluster centers and the standard deviation. Then, we combine the normally distributed data and plot the data. Observing the x and y-axis limits, we want to ensure it is also set to (0,20). Once the x and y axes are labeled, we can produce the scatterplot shown in **Figure 3.** It is similar to the scatterplot of Figure 1.

**Figure 3 Generated Binary Scatterplot**

****

Let us compare the scatterplot in Figure 3 to that in Figure 4. Figure 3 has the “o” plot points more tightly clustered; it is easily observed that the data is clustered into three distinct groups. A more apparent separation between the two classes indicates less risk of overfitting this model. If an overly complex model were applied to Figure 3, the model could be less accurate. Figure 4 shows that the binary variables are more dispersed, making classification more difficult; this data may benefit or even require a more complex model to have the best accuracy possible when training a learning model. Figure 4 may benefit a machine learning model, as it may be more likely to represent real-world data.

**Figure 4 Comparison Binary Scatterplot**

**A screen shot of a graph

AI-generated content may be incorrect.**

If the data from Figure 3 or Figure 4 were used to divide the tree, we would have some key considerations. We would need to maximize class separation, so we would need to determine how and where to split the data. The data would need to be split into smaller subsets based on features; this would help the machine learning model create a set of rules that could separate the “+” and the “o” points. Stopping criteria would also need to be set; as the tree grows, the rules become more complex and could result in tiny rules that contribute to noise (Bruce & Bruce, 2019). Once the rules were established, we could utilize pruning or removing unnecessary data to simplify the decision tree.

Simulating data is practical and valuable because different statistical methods can be developed under certain assumptions, which can sometimes be difficult to apply (Boulesteix et al., 2020). We can use data simulation to answer which model could be applied and which could be most appropriate in a particular setting. From healthcare, where data collection can introduce privacy risks, to retail sales performance, we can utilize simulated data to set performance markers and allow effective model testing.

**Conclusion**

Understanding overfitting and utilizing that understanding to evaluate model performance is important for statistical analysis. Overfitting occurs when models learn noise instead of important data points. Key considerations when applying models can be addressed by simulating data sets to evaluate models and their performance. Decision trees should be optimized through stop criteria and pruning before utilizing them on real-world data. Achieving balanced models with high accuracy is important for the best data evaluation outcomes.

**References**

Boulesteix, A.-L., Groenwold, R. H. H., Abrahamowicz, M., Binder, H., Briel, M., Hornung, R., Morris, T. P., Rahnenführer, J., & Sauerbrei, W. (2020). Introduction to statistical simulations in health research. *BMJ Open*, 10(12), e039921. https://doi.org/10.1136/bmjopen-2020-039921

Bruce, P., & Bruce, A. (2019). *Practical statistics for data scientists: 50+ essential concepts using R and Python* (2nd ed.). O'Reilly Media.

Matloff, N. (n.d.). *Overfitting*. The R Project for Statistical Computing. Retrieved February 18, 2025, from https://cran.r-project.org/web/packages/qeML/vignettes/Overfitting.html

R Core Team. (2024). *runif: Uniformly distributed random numbers*. The R Project for Statistical Computing. Retrieved February 18, 2025, from https://rdrr.io/r/base/Uniform.html

Scikit-learn Developers. (n.d.). *1.10. Decision trees*. Scikit-learn. Retrieved February 18, 2025, from https://scikit-learn.org/stable/modules/tree.html

**Appendix A**

**R Code for Simulating Scatterplot Data**

cluster\_centers <- list(c(2, 16), c(16, 16), c(10, 6))

std\_dev <- c(1.5, 1.5, 1.5, 2, 2, 2)

set.seed(42)

x\_plus <- runif(n\_points, min = 0, max = 20)

y\_plus <- runif(n\_points, min = 0, max = 20)

x\_cluster1 <- rnorm(n\_points, mean = cluster\_centers[[1]][1], sd = std\_dev[1])

y\_cluster1 <- rnorm(n\_points, mean = cluster\_centers[[1]][2], sd = std\_dev[4])

x\_cluster2 <- rnorm(n\_points, mean = cluster\_centers[[2]][1], sd = std\_dev[2])

y\_cluster2 <- rnorm(n\_points, mean = cluster\_centers[[2]][2], sd = std\_dev[5])

x\_cluster3 <- rnorm(n\_points, mean = cluster\_centers[[3]][1], sd = std\_dev[3])

y\_cluster3 <- rnorm(n\_points, mean = cluster\_centers[[3]][2], sd = std\_dev[6])

x\_o <- c(x\_cluster1, x\_cluster2, x\_cluster3)

y\_o <- c(y\_cluster1, y\_cluster2, y\_cluster3)

plot(x\_plus, y\_plus, pch='+', col="blue", xlim=c(0,20), ylim=c(0,20), main="Recreate", xlab="x1", ylab="x2")

points(x\_o, y\_o, pch='o', col="red")

grid()