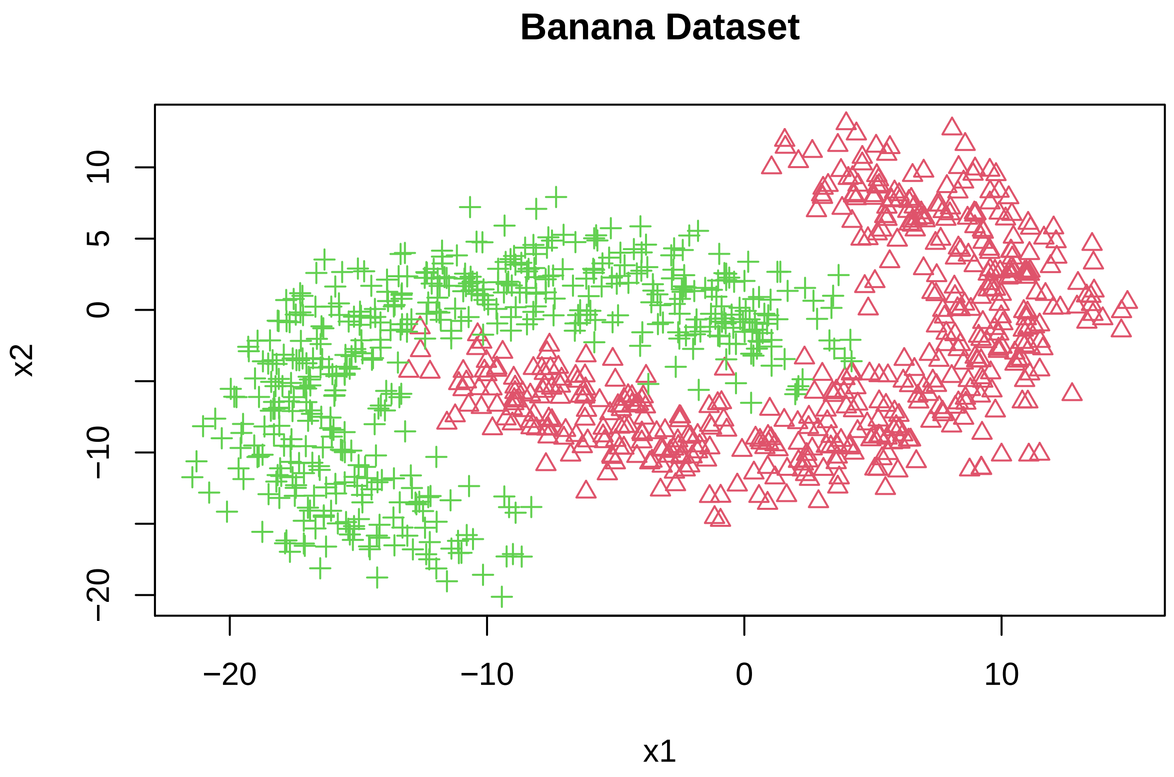
Banana Dataset Analysis

Michael Kogan

10/12/2020

Introduction

For my choose - your - own project I wanted to further explore the classification models available in Caret and see how well they performed on data with a complex decision boundary. During my research I came across an interesting academic dataset known as the "Banana Dataset". It contains 1000 entries with two numeric predictor variables x1 and x2, each with a range of about -25 to 20. Each entry has a corresponding binary response y that can take on the values of either 0 or 1, representing the two classes. Since this is an artificially created academic dataset, it is perfectly balanced with exactly 500 entries for each class. The name of the dataset comes from the shape of its plot, which resembles two curved opposite - facing adjacent bananas.



The green crosses correspond to y = 1 whereas the red triangles correspond to y = 0. Evidently, the resulting decision boundary is quite complex, and it is my hypothesis that not all machines would handle this type of problem well. I can say with high confidence that linear classifiers, such as basic logistic regression, LDA, or linear SVM, will likely result in lower accuracies than classifiers that are able to produce more complex decision boundaries. The goal of this project is to measure the performance of various models and determine which ones perform best on this kind of non - linear data. I would also like to examine if there are any emergent patterns in model performance. The models under consideration are as follows: Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Naive Bayes (NB), k Nearest Neighbour (kNN), Logistic Regression (LR), linear Support Vector Machine (SVM), Radial Basis Function (RBF) SVM, classification trees, Random Forest (RF), and XGBoost (XGB).

The key steps of this project were as follows:

1) Data exploration: explore the data to understand the distribution of the variables and the shape of the "true" decision boundary.
2) Data cleaning: perform regular data cleaning operations, including the standardization of the predictor variables, elimination of incomplete entries, and balancing of the dataset.
3) Hyperparameter tuning: if required, use cross - validation to tune the hyperparameters of the models under consideration.
4) Performance measurement: use stochastic holdout to repeatedly split the data into a training and test set and evaluate the performance of the models under consideration. By doing this repeatedly we are able to obtain an estimate of the distribution of the errors of each classifier.
5) Analysis: analyze the results of the experiment described in the previous four steps and report the results, to include the most performant models as well as any patterns that may have emerged.

Methods and Analysis

This section will describe the processes and techniques used as part of this project. I will start by describing initial data exploration and cleaning. I will then talk about the various modeling approaches to include the benefits, drawbacks, and associated hyperparameters of each.

Data Exploration

Using a plot of the dataset and basic summary operations, I analyzed the distribution and data types of the predictors and the response. The results of this were already described in detail in the introduction section so I will not restate them. Other key operations included checking for missing values and calculating the correlation between predictor variables. As this is an academic dataset there were no missing values. There was minor correlation (around 0.4) between the predictor variables, as seen in the following table:

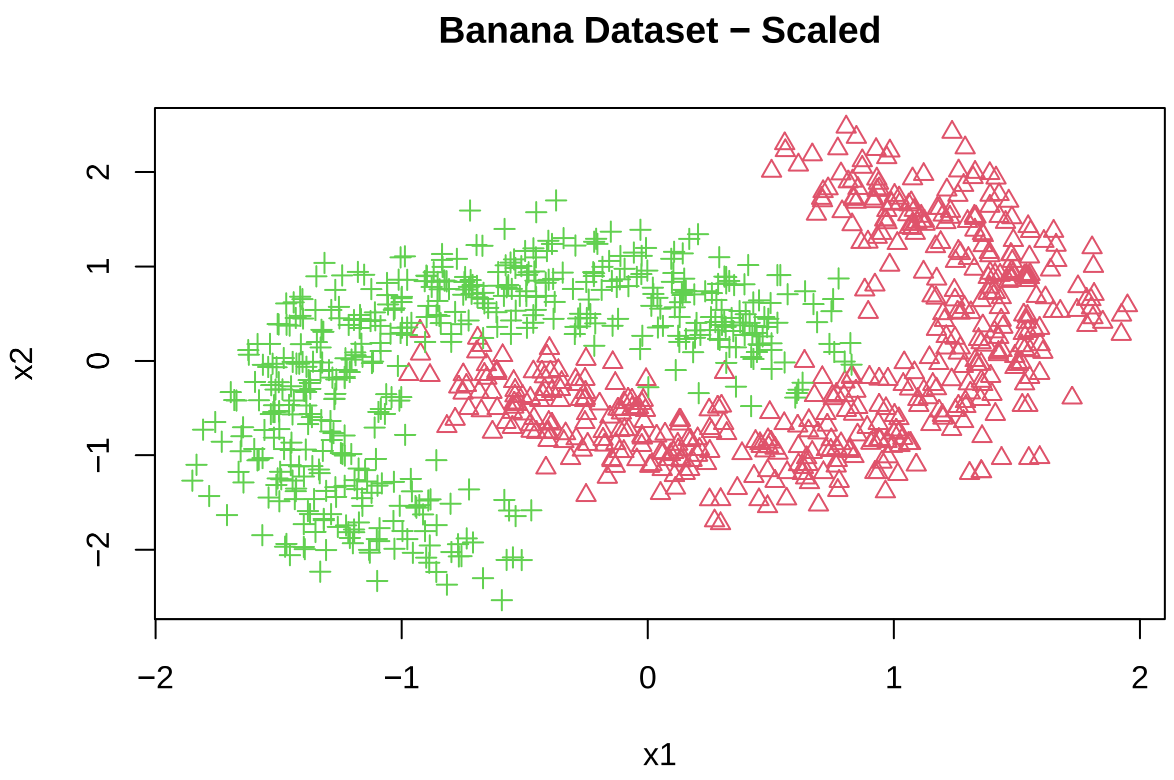
```
##           x1        x2
## x1 1.0000000 0.3786795
## x2 0.3786795 1.0000000
```

It is important to check for correlation between variables because multicollinearity can introduce problems in some modeling approaches such as logistic regression. Nevertheless, I did not consider a correlation of 0.4 to be significant enough to cause major impact.

Data Cleaning

Being an academic dataset there wasn't much cleaning to be done and the dataset came well - positioned for analysis. Nevertheless there were several minor operations and one major operation that had to be performed. The minor operations included defining the n and p, corresponding to the number of examples and number of predictors respectively, as well as splitting out the predictors and response into separate objects.

The major operation was scaling the data to normalize each predictor to approximately the same range. Scaling results in each predictor having zero mean and unit variance. This is important for modeling approaches (such as SVM) that measure the distance between inputs to ensure that the scale of a given predictor variable does not impact it's importance. Scaling was done using the "scale" function in R and the results are evident in the following plot of the scaled data:



The plot retains the same shape as the original while the range of the predictor variables has be normalized to the [-2, 2]. The scale function takes each element in a given vector, subtracts the mean of the vector, and divides by the standard deviation of the vector.

Insights Gained

There were several key insights gleamed from the exploration phase of this project:

1) The dataset is perfectly balanced. This is excellent because it means that upsampling of the under-represented class is not required.
2) The decision boundary is non - linear and complex. As such, this dataset meets the goal of this project in that it allows us to test how various modeling approaches handle this kind of complex boundary.
3) This is a low - dimensional dataset with the number of examples (n) much greater than the number of predictors (p). As such, there is no curse of dimensionality which can cause significant issues with many modeling approaches.

4) The relatively low number of examples (1000) makes this dataset unsuitable for some of the most complex modeling approaches with a large number of parameters such as neural networks, as they require a very large number of training examples to properly train.

5) Some multicollinearity was present but this was not found to be a concern as it was relatively minor.

Modeling Approaches

This section will describe the ten modeling approaches attempted in this project, as well as the corresponding hyperparameters. Some approaches will be grouped into a single section due to their inherent similarity.

LDA and QDA Linear Discriminant Analysis (LDA) and Quadratic Discrimant Analysis (QDA) are two similar approaches that the conditional distribution of each class, $p(x|y = c)$, follows the multivariate normal distribution. The difference between the approaches is the LDA assumes all conditional class distributions have the same covariance matrix with different mean vectors, whereas QDA assumes that each class condition distribution has a unique mean vector and covariance matrix. LDA is only capable of producing linear decision boundaries, whereas QDA is capable of producing non - linear boundaries and is likely better suited for this dataset. The tradeoff is that QDA normally requires more training examples because we have to estimate a convariance matrix for each class and not a single covariance matrix. There are no hyperparameters that require tuning and these approaches can be used out of the box. That being said, the distribution of these classes is obviously not normal, they form banana - shaped curves and not the ovals we would expect to see if they were indeed normal. However, it is still worth trying them out to see what kind of performance they generate.

Naive Bayes Naive Bayes is related to LDa and QDA, but assumes that predictors are independent of each other. It estimates the class conditional distributions for each predictor variable, and then calculates the overall class conditional distributions as the product of the conditional distributions of the individual predictors. It makes no assumption about the shape of the class conditional distribution and is therefor somewhat more flexible that LDA and QDA, as long as the independence assumption proves true. We know for our dataset that the predictor variables are not fully independent as there is some multicollinearity, but this approach may prove successful since the multicollinearity is not severe. As with LDA and QDA, there are not hyperparameters to tune.

k - Nearest Neighbor k - Nearest Neighbor (kNN) is an approaches that determines the class of a new input by comparing it's predictors to all other examples in the training set, and choosing the class corresponding to the majority class of it's k closest neighbors. It is a very simple machine but is very effective and can generate very complex decision boundaries, especially for low values of k. As k goes up, the decision boundary becomes smoother and less complex. This approach works well for many different problem sets, and I expect it to perform well on this dataset as well. The value of k needs to be determined by us. To determine k, as well as all subsequent hyperparameters, I used 10 - fold cross validation through the Caret package. My search grid for k was the values 1 to 25 inclusive.

Logistic Regression Logistic Regression (LR) uses a logistic function to model a binary dependent variable, and is therefor well suited to our binary classification problem. It is an extension of linear regression and is usually trained by maximum likelihood estimation. As it is an extension of linear regression, in it's basic form it is only able to generate linear decision boundaries and I don't expect it to perform well on this dataset. Nevertheless, it is worth trying for comparative purposes. There are no hyperparameters to tune, so it can be used out of the box.

Support Vector Machines and Kernel Extensions Support Vector Machines (sVMs) and their kernel extensions are famous modeling approaches that attempt to find the optimal separating hyperplane between two or more classes by maximizing the margin, or distance, between classes. SVMs are normally formulated and solved as quadratic programming problems and are very flexible in that they work well even on short and wide datasets where p > n. SVMs work by measuring the distance between inputs and is one of the modeling approaches for which scaling the input data is very important.

Since we are not always able to perfectly separate the classes, it is important to account for inevitable misclassification by introducing a slack variable. The extent of allowable error is represented by the "C" hyperparameter. The larger the value of C, the harsher misclassified examples will be penalized. This will generally result in a smaller margin as the algorithm will do it's best to avoid misclassification. This hyperparameter can be tuned using cross - validation.

The basic soft - margin SVM described above is only able to generate linear decision boundaries, and would therefor not work well on our dataset. Luckily, there are Kernel extensions, which project the input space into a higher dimension where the classes are linearly separable. This enables SVM to generate very complex, non - linear decision boundaries. The most commonly - user Kernel extension is the Radial Basis Function (RBF) kernel, which is the one I decided to attempt as part of this project.

The RBF SVM approach introduces a second hyperparameter, sigma, which controls the reach of each training example. When the value of sigma is high, only the points close to the decision boundary will have an effect on it. Conversely, when sigma is low, the majority of the points will impact the decision boundary. This hyperparameter can similarly be estimated using cross - validation. The grid of values I tried for C and sigma were taken from a paper, "A Practical Guide to Support Vector Classification", which can be found here: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

Classification Trees Classification trees are a commonly - used modeling approach that generates a set of rules by starting with the entire dataset and splitting it on one predictor variable at a time in order to maximize or minimize some sort of metric that measures the quality of the split. Common metrics include the Gini impurity measure, information gain, and variance reduction. This generated set of rules, or splits, can then be used to classify new inputs. The rpart approach in Caret uses the Gini measure by default, and that is the one I chose to use. Decision trees can handle awkward decision boundaries and I expect them to perform relatively well on this dataset. There is a single hyperparameter, the maximum tree depth, which determines how deep the tree can be, and is the same as setting a cap on the number of rules (or splits). As always, I used cross validation to tune the maximum depth hyperparameter, trying values from 1 to 12.

Random Forest Random Forest (RF) is a ensemble learning model that applies the random subspace learning approach to trees. It works by training many classification trees, each on a random subset of the predictor variables, and then taking the most common class amongst the generated trees as the predicted output. There is a hyperparameter, mtry, which determines the size of the subset, or the number of predictor variables available for training each individual tree. Naturally for us this can only be one or two, since there are only two predictor variables.

Boosted Trees Boosted treed are another ensemble learning model that applies boosting to trees. It works by training many trees sequentially, where each tree is trained in a way to focus on the mistakes of the previous. This is usually done by emphasizing, or putting a higher weight, on the mistakes of the previous tree. Each tree is assigned a weight, and the output is the sign of the weighted average of the predictions. Naturally, for this to work, we are required to label examples from the first class as "-1" and examples from the second class as "+1". As a side note, this is a requirement in other approaches, such as SVM. The specific implementation I chose to try is Xtreme Gradient Boosting, or XGBoost (XGB). This is a famous approach and has been known to yield excellent results on many types of problems.

There are several hyperparameters that require tuning, but the three main ones are maximum depth (same as in the tree approach), the learning rate (eta), and the number of generated trees. The first and the last hyperparameters are self explanatory. The learning rate is similar to the learning rate in other gradient boosting algorithms, in that it determines how much of the update we incorporate in each step. Generally the lower the learning rate, the more trees we will need in order to attain good performance. However, a learning rate that is too high will likely result in overfitting. The range of values I used for the tree depth is 1 to 6, for the number of trees is {50, 100, 200, 400}, and for the learning rate is {0.2, 0.3, 0.5, 0.8}.

Comparison Approach

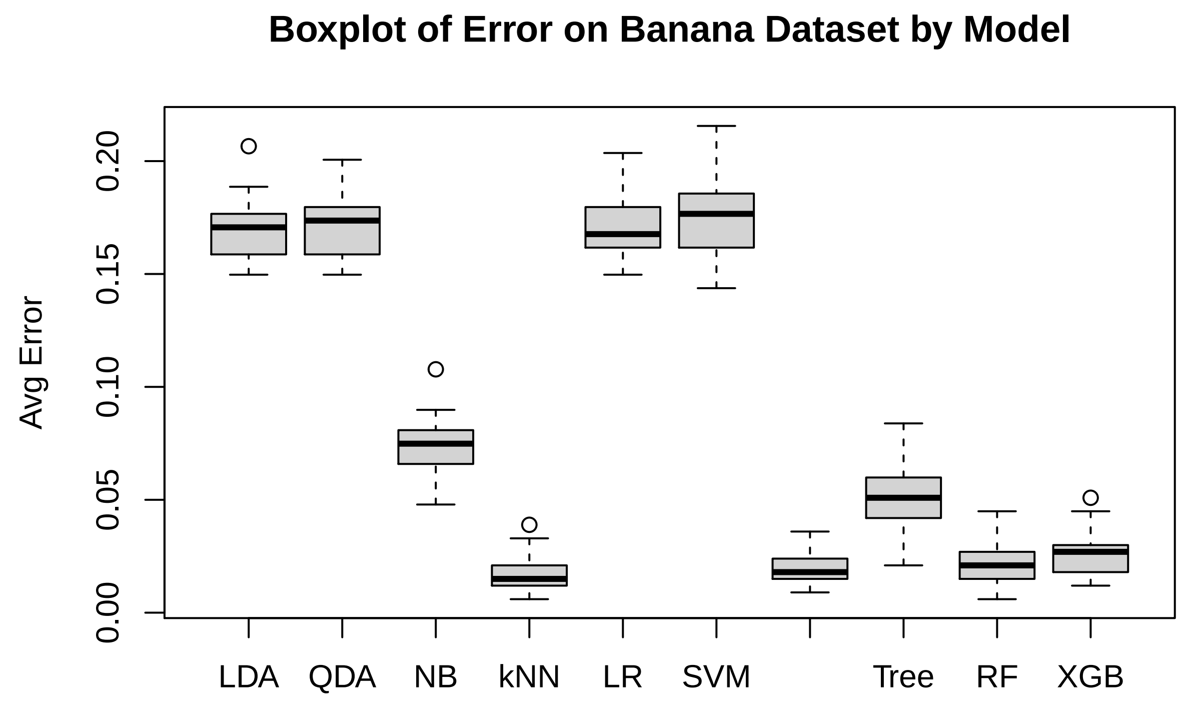
In order to compare the various approaches I first determined the optimal hyperparameters for each model that requires them. I then randomly split the dataset into a training and test set 25 times, training each of the models on the training set, and evaluating each model on the test set. This was done to reduce the variability in performance that can happen due to a specific training/test split, generating 25 test set error samples for each modeling approach. I was then able to compare mean errors as well and create boxplots of errors by model in order to visualize the median error rate as well as the distribution of the error rates for each model.

Results

The mean error rate of each model is summarized in the table below:

```
##      Model Mean.Error
## 1      LDA 0.16994012
## 2      QDA 0.17029940
## 3       NB 0.07377246
## 4      kNN 0.01856287
## 5       LR 0.17041916
## 6      SVM 0.17365269
## 7  RBF SVM 0.01868263
## 8     Tree 0.05173653
## 9       RF 0.02251497
## 10     XGB 0.02730539
```

As we can see, the models can be split into three catagories: those achieving a mean error rate of 17%, those achieving a mean error rate of around 2%, and those that are hovering around the 5 - 7% range. The boxplots of the errors tell a similar story:



As we can see, the models that attain an error rate of around 17% are the ones that generate linear decision boundaries (LDA, LR, and SVM), and QDA, which as already discussed is unsuitable when the class conditional probabilities do not follow the normal distribution. Naive Bayes and Trees are able to attain an error rate of around 5 - 7%, and are generally quite performant but do not achieve the same results as the remaining models. Perhaps if there was less multicollinearity Naive Bayes would perform better.

The models that performed best are kNN, RBF SVM, RF, and XGB. I initially expected RBF SVM to perform best, but to my surprise kNN actually had a lower mean error rate. That being said, in my experimentation the median error rate of RBF SVM was lower. RF and XGB were able to attain better performance than their base learner (the tree), which is unsurprising because the goal of Ensemble methods is to improve base learned performance.

Conclusion

During this project I explored and learned about many different approaches to classification. I tested these approaches using repeated evaluation against the Banana dataset, which corresopnds to a binary classification problem with a complex non - linear decision boundary. As expected, models that are able to handle such boundaries performed substantially better than those that aren't. At worst, models generating linear decision boundaries or those unsuited to this task attained an average error rate of about 17%. Of the remaining models, none attained an average error rate greater than 7.5%, and kNN managed to attain an average error rate of less than 2%.

The impact of this project on the wider community is limited as this was an exploratory project which I undertook to widen my understanding of various modeling approaches and test them against a relatively complex but artificial dataset. One key takeway that I think many members of the data science community would benefit from is to always remember the No Free Lunch Theorem. Of all modeling approaches, kNN, arguably the simplest approach, attained the best performance. We never know which model will perform best on a given dataset and it is important to not discard any approaches without at least considering them.

The main limitation of this report is that the list of models that were tested is somewhat limited as many famous approaches, such as Neural Networks, Relevance Vector Machines, and Generalized Linear Models were not considered. Future work would entail testing out these other approaches, and potentially doing so on a variety of datasets. Other future work could entail running similar experiments for regression, or trying to redo this project with a non - academic dataset.