Machine Learning Shiny App

Vin Seixas, Samuel de Oliveira, Hunter Stopford

Github link: https://github.com/viniciusdel/MachineLearningPredictions

Abstract

In this project, we apply various Machine Learning (ML) models to quintessential problems in the field of computational statistics, and create a user-interactive Shiny app for demonstration purposes. A Random Forest (RF) classifier was used on a large dataset of handwritten numerical digits in order to train a model to recognize numbers rendered on a digital draw-pad. We then applied a Support Vector Machine (SVM) model to a medical cost dataset to create a regression model which could then be used in order to make predictions of insurance costs for the user, given relevant data like age, sex, BMI, region, etc. Lastly, we create a medical questionnaire which uses a RF classification model from the caret package to make a highly-predictive diabetes diagnosis for users. We discuss the differences between the datasets and the metrics used to evaluate the ML algorithms. Lastly, we explain the design of our Shiny application and how the user interface was integrated into the trained ML models.

Introduction

We tested our datasets (Digit Recognition notwithstanding) across a variety of ML algorithms, including Classification and Regression Trees (CART), Linear Discriminant Analysis (LDA), K-Nearest Neighbor (KNN), Neural Networks (NN), and Random Forest. The selection of model we would use in our shiny application was based on which produced the least error after training. The particular R packages and ML algorithms we used for each tab are as follows:

- Digit Recognition - randomForest package - Random Forest
- Insurance Cost - caret package - Support Vector Machine
- Diabetes Diagnosis - caret package - Random Forest

We chose for our datasets one regression example--insurance cost prediction, and two classification examples--digit recognition and diabetic diagnosis. Broadly speaking, supervised machine learning (more on supervised vs. unsupervised in the Discussion section) can be divided into 'regression' and 'classification' algorithms, where a distinguishing feature is whether the output variable type is of a continuous or discrete value. Regression algorithms try to make the best fit line between input variables and values which can be real and continuous, like the price of insurance. Classification algorithms try to make a decision boundary such that output value fits into a discrete value or class, like the binary logic of a diagnosis or the value of an integer.

Methods

Digit Recognition

For the digit recognition application, we trained an RF model with the MNIST database which is often used for testing and validating optical character recognition devices. The model is trained on a dataset of 42,000 known observations and then validated against another 28,000 observations. See table below:

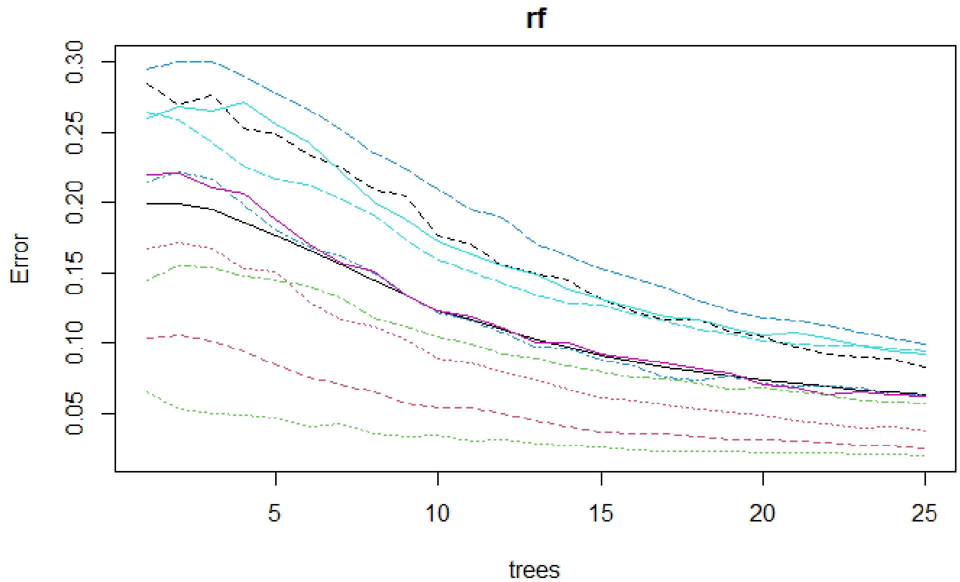
Number of Observations for Each Digit

0	1	2	3	4	5	6	7	8	9
4132	4684	4177	4351	4072	3795	4137	4401	4063	4188

The actual arguments fed into the algorithm are grayscale pixel values from 0 to 255, which form a 28x28 image, resulting in a total of 784 pixels.

One important parameter selected was the 'number of trees' which determines the number of decision trees the algorithm should build prior to evaluating the predictions of each tree, and determining which class was selected by the most number of trees. For our model, we chose 25 as the number of trees. A greater number of trees will also make the model training time slower. The training time given our parameters was around 209.9 seconds.

The number of trees was decided based on flattening trend-line in the error reduction as the number approached 25. We can see the error decreases as we add more trees, but anything more than 25 would not significantly increase accuracy:



The data frame also returns a confusion matrix generated after testing. The confusion matrix shows the number of times a digit is correctly identified, and how often it is confused for each of the other classes. We can see the results of the trained model in the following confusion matrix:

```
               Type of random forest: classification
                     Number of trees: 25
No. of variables tried at each split: 28

        OOB estimate of  error rate: 6.3%
Confusion matrix:
     0    1    2    3    4    5    6    7    8    9 class.error
0 4027    1   11    8    4   15   27    4   24   11  0.02541142
1    1 4588   18   21    9   10    6    6   14   11  0.02049530
2   20   13 3913   42   33   17   30   45   49   15  0.06320326
3    8    9  101 3940    7  118   13   44   76   35  0.09446104
4   11   11   20    9 3816   13   28    6   21  137  0.06286837
5   20   12   17  124   15 3479   40   12   45   31  0.08326746
6   33   10   19    3   27   43 3982    1   18    1  0.03746676
7    7   17   68   20   34    7    1 4150   16   80  0.05681818
8   18   34   53   79   36   75   31   15 3658   64  0.09968004
9   29    9   19   48  129   28    8   66   50 3802  0.09216810
```

We were able to train the model to predict numbers with only a 6.3% error.

Insurance Cost

In contrast to the previous algorithm, predicting the cost of insurance premiums would require a regression algorithm to generate a continuous value output. Our dataset was composed of 1,338 observations with 7 factors each, 80% of which was used for training and 20% for testing. A regression fit was attempted using CART, LDA, KNN, SVM, RF, and NN. LDA and NN failed to converge entirely, which may be due to formatting issues in the data-set. The metric used to evaluate the suitability of any particular matrix was the Mean Absolute Error and the Root Mean Squared Error. Although, RF and SVM had similar results, with some tweaking SVM produced slightly better results. See table and plot below:

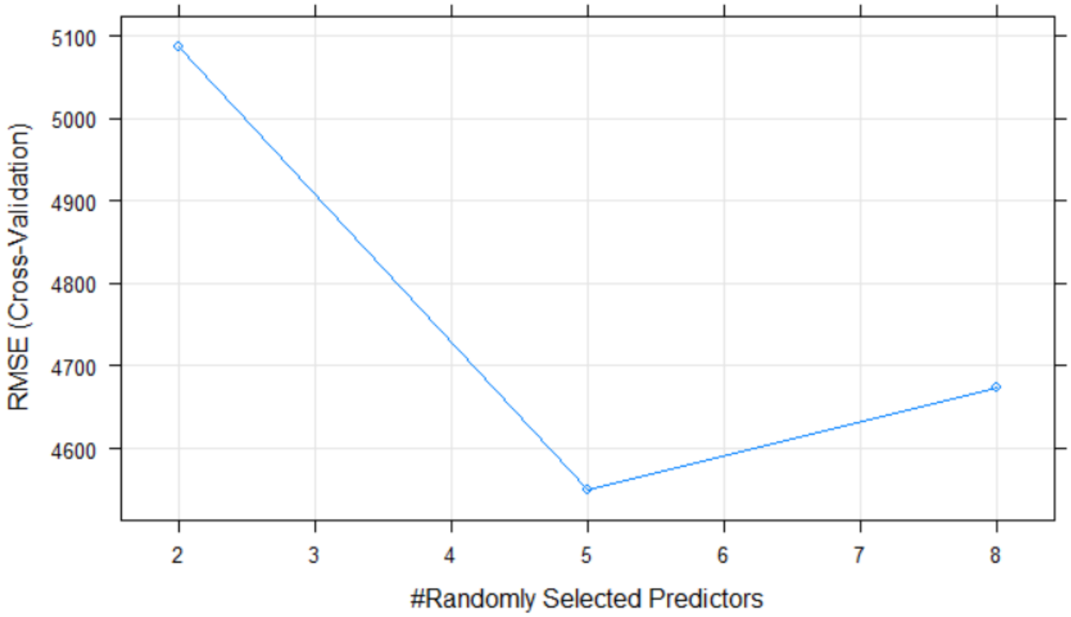
```
Call:
summary.resamples(object = results)

Models: cart, knn, svm, rf, nnet
Number of resamples: 10

MAE
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
cart  2681.995  3218.822  3472.598  3706.852  4143.310  4980.824    0
knn   7146.108  7485.428  7799.340  7882.858  8356.537  8605.010    0
svm   1836.417  2446.088  2556.530  2564.770  2782.064  3047.852    0
rf    1972.843  2396.621  2641.320  2573.146  2758.312  3066.249    0
nnet 12632.847 12934.084 13215.645 13187.160 13453.747 13703.147    0

RMSE
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
cart  3721.808  4720.339  5237.334  5308.145  5847.030  6950.018    0
knn   9724.608 10543.973 10808.022 10906.570 11079.246 12483.971    0
svm   3104.436  4315.990  4641.125  4652.820  5127.560  5804.294    0
rf    2937.678  4184.731  4708.914  4550.633  4983.398  5648.545    0
nnet 16900.679 17113.729 17718.930 17767.779 18147.124 19276.253    0

Rsquared
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
cart 0.61445255 0.7737296 0.8234133 0.7968216 0.8357828 0.9138882    0
knn  0.09233985 0.1441629 0.1630478 0.1746007 0.2089587 0.2731238    0
svm  0.78568037 0.8111291 0.8536403 0.8486732 0.8725954 0.9410740    0
rf   0.78948683 0.8257163 0.8504202 0.8531205 0.8737969 0.9542256    0
nnet         NA        NA        NA       NaN        NA        NA   10
```



Diabetes

In the diabetes prediction application, we went a different route and used the *caret* package and its Random Forest classifier to train a model on patient data of people that suffer from diabetes. 520 patients were analyzed, with fields ranging from Age and Gender to symptoms like irritability. 200 of those patients were negative for diabetes, while the rest were positive.

We analyzed different models like CART, LDA, K nearest neighbors, random forests and neural networks, but you can see that random forests had the highest accuracy mean, so that's what we picked.

	Overall
	<dbl>
PoluryaYes	100.000000
PolydipsiaYes	83.294493
Age	27.624015
GenderMale	26.684665
IrritabilityYes	9.408118
AlopeciaYes	8.487711
DelayedHealingYes	8.416027
suddenWeightLossYes	8.095202
PartialParesisYes	7.082277
ItchingYes	4.231519

```
Accuracy 
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
cart 0.7619048 0.8302846 0.8554007 0.8461672 0.8750000 0.9024390    0
lda  0.8048780 0.8571429 0.8571429 0.8724739 0.8963415 0.9523810    0
knn  0.7142857 0.7804878 0.8214286 0.8220093 0.8750000 0.9268293    0
rf   0.9268293 0.9523810 0.9759001 0.9734611 1.0000000 1.0000000    0
nnet 0.9024390 0.9515099 0.9642857 0.9638211 0.9761905 1.0000000    0

Kappa 
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
cart 0.5183486 0.6592883 0.6889002 0.6780094 0.7250288 0.7902813    0
lda  0.6076555 0.7133300 0.7174888 0.7432124 0.7841767 0.8990385    0
knn  0.4349776 0.5724218 0.6435685 0.6456811 0.7423176 0.8512696    0
rf   0.8479604 0.8996310 0.9485360 0.9442979 1.0000000 1.0000000    0
nnet 0.7950000 0.8992177 0.9249371 0.9240457 0.9501188 1.0000000    0
```

An interesting analysis we also did was to see which fields had the highest relevance to the algorithm's prediction. We performed a variable importance analysis that shows that some features are more important than others namely Polyuria and Polydipsia.

Shiny Application

On the application side, the packages used were Shiny, Shinythemes for changing the UI appearance, Magick for image processing and manipulation, Gtools, and Imager for additional image processing functionalities. The Shiny package divides the structure of the application into a 'user interface' and a 'server' side. On the UI side, we organized each of our trained models into separate tabs, along with the user functions for those models. The syntax for the UI design is something like a mix between HTML and R. Elements of a page can be generated with a function call like in R, and each function will have its own function parameters, but specific elements can be placed within one another by separation with a comma. Since the UI has to interact with the server side, variables which must be passed to the server side are identified with 'inputID.' On the server side, those variables are then just referenced using the syntax 'input$[insert input ID].' We include our already trained models in a subdirectory of the main file. Those can be referenced and used using the ‘readRDS’ function.

Conclusion & Future Work

One commonality shared by all the data we used was that it was labeled data, suitable for supervised machine learning. With supervised machine learning, it’s necessary to train a model with observations wherein all relevant variables are known. The model is then validated with test data to determine how well it makes a regression fit or classification. Unsupervised machine learning uses unlabeled data, and the algorithm is then left to make associations amongst the variables. This has the added benefit of finding hidden correlations, but also may not offer good results and is more difficult to validate. For future work, we would like to try some unsupervised algorithms on larger, less understood datasets.

One of the challenges we faced in this project was generating and processing the plot in such a way as to make it usable for the digit recognition model. The model seems to be highly sensitive to the order in which we carried out the image processing functions. Certains numbers also reliably produced errors beyond what we saw from the training data, even though the pixel image being fed into the model greatly resembled what we saw in the training data. For future work, we would invest more time into the image processing to reduce the amount of error the draw-pad function produced.

References

M M Faniful Islam, Rahatara Ferdousi, Sadikur Rahman, Yasmin Bushra. UC Irvine. Early Stage Diabetes Risk Prediction dataset. https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.

Yann LeCun, Corinna Cortes, Christopher J.C. Burges. NIST. MNIST database of handwritten digits.http://yann.lecun.com/exdb/mnist/index.html

Marc Agenis. Plotting function on Number Recognition app. https://stackoverflow.com/posts/48442522/timeline