Statistics 440 - Case Study 1: Predicting Crohn's Disease & Ulcerative Colitis Using Gene Expression Data

Barinder Thind
301193363
Simon Fraser University

Abstract

In this project, we used models on gene expression data in order to predict whether or not some particular individual has Crohn's disease, ulcerative colitis, or neither. Throughout the process, we fitted a multitude of these classifiers to varying degrees of success. In particular, a gradient boosting method built on the entirety of the data seemed to produce the best mixture between consistency & accuracy whereas a multinomial model was the least impressive (although an ensemble of dimensionally reduced data funneled through the multinomial model was much better). We also used k-means clustering which - more or less - partitioned patients with Crohn's disease from healthy individuals. In the report below, we highlight some of these models and the results associated with them with respect to this data.

Keywords: neural net, k-nn, PCA, random forest, xgboost, multinomial

I. Introduction

While gene expression data can be a bit mystifying to anyone not in the field of genetics, it is, in fact, not particularly important to have a deep understanding of what the data actually represents when it comes to prediction. This is because the data is, more or less, used as a means for an end and, in this case, that "end" was to try and solve the following questions:

1. Can data features (i.e., variables or probesets or genes) be used to cluster individuals into three biological groups (i.e., healthy individuals, CD patients, UC patients)?[1]

2. Can data features (i.e., variables, probesets or genes) predict the disease state of individuals from three biological groups (i.e., healthy individuals, CD patients, UC patients)?

In essence, we are trying to find commonalities within the data that allow us to then cluster individuals together and also trying to find models that will allow us to predict whether or not

[1] Where UC = ulcerative colitis and CD = Crohn's disease

any single person has a disease (given that we have information about their gene expressions). Our focus was more on the latter question and, in fact, it is the easier one to provide a solution for. This is because, while for the first question you could conceive some set of significant features with respect to classification, it is much more difficult to explain why these features behave this way and how they contribute to a particular disease[2].

In the following sections, we first provide information on the data that was used, and then a clustering algorithm to answer the first question above. We also provide small descriptions of the models fitted that were used to address the second question. After this, we present the results of each of the highlighted models along with some commentary as to why we got those particular results. Lastly, a concluding section to summarize the case study.

II. Methodology

Data Description

The data has 126 observations (individuals) and 313 variables. Of these 313 variables, 309 of them are information about the gene expressions of each of the 126 individuals. The other four variables are: *Group* (a variable that tells us whether the individual is a CD patient, UC patient, or healthy), *Age*, *Ethnicity*, and *Sex*. The *Ethnicity* variable was particularly troublesome because it had very few observations for some ethnicities (such as for Asian). This resulted in issues when the data was split between the training and test sets[3]. For this reason, adjustments were made and a couple of observations were dropped. Also, data-cleaning code was provided to us by Dr. Davis which made the start-up process much easier and so, an acknowledgement is made in the references.

Analyzing the Data

[2]Although, we acknowledge that this is not the goal of the case study, it is a question that immediately towers following *any* conclusion that we arrive at

[3]For example, different factor levels in the split sets

0.1. Clustering to Find Patterns

For this part of the project, we used k-means clustering to see if we could find groups of observations that are similar to one another. K-Means is an unsupervised machine learning model, popular for Big Data problems. The algorithm partitions data into k distinct groups,

$$C_1 \cup C_2 \cup \ldots C_k = 1, \ldots, n \text{ where } C_k \cap C'_k = \emptyset, \forall\ k\ != k'$$

A good cluster is one which the within-cluster variation is minimized. This minimization is done through squared Euclidean distance.

$$\min_{C_1, ..., C_k} [\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2]$$

0.2. Model: Random Forest

The first model that was fitted was the *random forest*. This algorithm is particularly useful for classification problems as the one we are presented with in this case study because it can handle high dimensional data exceptionally well. Tree ensembles do not expect linear features or features that interact linearly- which is a good thing because of how many features we have. This would not be the case for something like logistic regression (which is also a popular classifying method).

0.3. Model: Principal Component Analysis ⇒ Random Forest

Similar to the random forest model - despite the fact that a high number of dimensions is not necessarily a problem for tree ensembles, I decided to reduce the dimensions anyway and run it through a *random forest* model again. The results will be discussed in the Section III.

0.4. Model: Principal Component Analysis ⇒ Multinomial

A multinomial model is another classifier that is widely used to predict categories. Since we had three possible outcomes, it made more sense to use this model than something like logistic regression (which handles binomial cases).

0.5. Model: Neural Net

Neural nets are another way to make predictions. They fall under the umbrella of "machine learning" techniques. Neural nets "learn" through training examples and then use that learned information to make predictions. For example, you could use a neural net to recognize hand-writing. Initially, you might feed it some images that include hand written words and then, using this information, the neural net will began to create an internal network that, when prompted with new data, will use that old information to make predictions on the new data. You can improve its accuracy by feeding it more training data.

0.6. Model: K-Nearest Neighbors

This is a rather simple technique that does a fairly remarkable job of classification. The idea is to define some neighborhood of size k and, for every observation in that neighborhood, you take the average[4] and make that your prediction. For example:

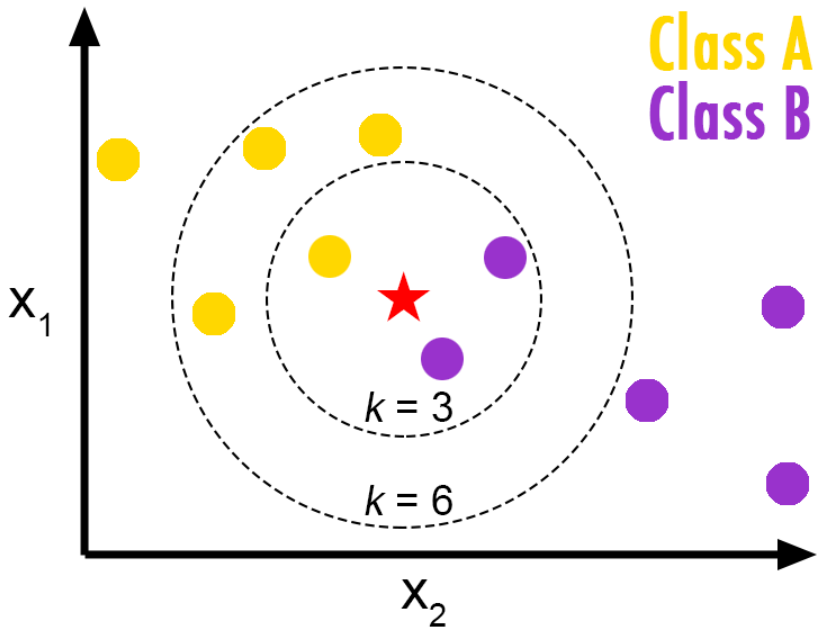

Figure 1: K-NN Example

If you were trying to predict the $\star$ in figure 1 with $k = 3$, you would predict that it would be

[4]In the case of a scalar prediction. For classification, your prediction becomes $\max[Class_1, Class_2, ..., Class_n]$ where the max refers to the number of appearances of each class in your neighborhood.

of Class B. However, if you were to predict with $k = 6$, it would be predicted to be Class A. While it can be difficult to find the optimal value of k, an empirical approach can often help - at least in the context of a specific data set.

0.7. Model: Gradient Boosting

Gradient boosting models are another machine learning technique used for classification for which "weak learners" are combined to form a strong learner. The particular technique that I used was xgboost (extreme gradient boosting) which is some optimization of gradient boosted decision trees. While we don't have an in-depth explanation about this model, it had the best results for this case study.

0.8. Measurement of Accuracy

In order to assess the adequacy of each model, the following, MCVE[5], was used:

$$\text{MCVE} = \sum_{j=1}^{k} \frac{[\frac{\sum_{i=1}^{n} u_i}{n} = x_j]}{k}$$

Where u_i takes on the value of 0 or 1 depending on whether or not the i^{th} prediction in the j^{th} cross-validation is the same as the actual result in the test value. This results in: $0 \leq \text{MCVE} \leq 1$.[6]

III. Results & Discussion

The error rates are displayed in the last line of code within the pictures[7].

0.1. Results: k-means Clustering

After running the k-means algorithm, we ended up with some interesting results. It seems that once $k = 6$, there is no point to adding any more clusters as none of the observations get partitioned anymore (at least up until $k = 50$ anyway). The following plot illustrates the results from the clustering:

[5]Mean Cross-Validation Error

[6]A quick note that for some of these models, the data was scaled.

[7]While, I know it wasn't required to submit any code, I put it in there just as kind of a semi-proof that I did indeed cross-validate and run all these models

Figure 2: K-Means Results

As you can see, it seems that people with Crohn's disease seem to be much more prevalent in Cluster 4, 5, and 6. There is some commonalities within the gene expression data that is similar for a lot of the patients with Crohn's disease. We can also notice that most of the "normal" patients are in the first two clusters. If I was collaborating with a geneticist, I would direct them to look at some of the specific genes that are causing these cluster results.

0.2. Results: Random Forest

After running this model, the MCVE was:

```
> success_rate_rf <- NULL
> for(i in 1:9){
+
+   # Segementing data by fold using the which() function
+   testIndexes <- which(folds == i, arr.ind=TRUE)
+   testData <- gene_data_f_s[testIndexes, ]
+   trainData <- gene_data_f_s[-testIndexes, ]
+
+   # Fitting model
+   rf_r_cv <- randomForest(trainData$Group ~ .,
+                           data = trainData, ntree = 10000)
+
+   # Making prediction on test set
+   success_rate_rf[i] <- sum(as.vector(predict(rf_r_cv, testData)) ==
+                               testData$Group)/nrow(testData)
+
+ }
> mean(success_rate_rf)
[1] 0.7428774929
```

Figure 3: Random Forest Error Rate

0.3. Results: Principal Component Analysis ⇒ Random Forest

After running this model, the MCVE was:

```
> for(i in 1:9){
+
+   # Segementing data by fold using the which() function
+   testIndexes <- which(folds == i, arr.ind=TRUE)
+   testData <- gene_data_pca[testIndexes, ]
+   trainData <- gene_data_pca[-testIndexes, ]
+
+   # Fitting model
+   rf_pca_cv <- randomForest(trainData$Group ~ .,
+                             data = trainData, ntree = 10000)
+
+   # Making prediction on test set
+   success_rate_rfpca[i] <- sum(as.vector(predict(rf_pca_cv, testData)) ==
+                          testData$Group)/nrow(testData)
+
+ }
> mean(success_rate_rfpca)
[1] 0.7613960114
```

Figure 4: PCA + Random Forest Error Rate

0.4. Results: Principal Component Analysis ⇒ Multinomial

After running this model, the MCVE was:

```
final  value 0.000002
converged
# weights:  114 (74 variable)
initial  value 92.283432
iter  10 value 33.796259
iter  20 value 29.342324
iter  30 value 28.817915
iter  40 value 28.294264
iter  50 value 28.281919
iter  60 value 25.493363
iter  70 value 14.991967
iter  80 value 0.027279
final  value 0.000020
converged
> mean(success_rate_mnom_pca)
[1] 0.6213333333
```

Figure 5: Multinomial on PCA Data Error Rate

For fun, we decided to run this model on the non-reduced data as well.

```
iter  40 value 12.798617
iter  50 value 12.340783
iter  60 value 12.062206
iter  70 value 11.886262
iter  80 value 11.793768
iter  90 value 11.760435
iter 100 value 11.632403
final  value 11.632403
stopped after 100 iterations
> mean(success_rate_mnom)
[1] 0.4946666667
```

Figure 6: Multinomial on Non-Reduced Data Error rate

0.5. Results: Neural Net

After running this model, the MCVE was:

```
iter  40 value 0.000521
final  value 0.000095
converged
> train_pred2 <- predict(gene
> sum(train_pred2 == data2_tr
[1] 84
> test_pred2 <- predict(gene_
> sum(test_pred2 == data2_tes
[1] 0.8333333333
```

Figure 7: Neural Net Error Rate

0.6. Results: K-Nearest Neighbors

After running this model, the MCVE was:

	Chron's Disease	Healthy	Ulcerative Colitis
Chron's Disease	43	0	7
Healthy	6	40	3
Ulcerative Colitis	10	1	16

Figure 8: K-NN Error Rate

This was done with $k = 1$. The MCVE $= \frac{99}{126} = 0.786$

0.7. Results: Gradient Boosting

After running this model, the MCVE was:

```
[96]    train-mlogloss:0.017879
[97]    train-mlogloss:0.017861
[98]    train-mlogloss:0.017842
[99]    train-mlogloss:0.017824
[100]   train-mlogloss:0.017805
> n
 [1] 1 1 1 1 1 1 1 1 1 1
```

Figure 9: XGBoost Error Rate

0.8. Discussion

It seems that the best model from these was the one that took advantage of boosted decision trees. However, it seems to be too good of a model as it achieved a 100% success rate through each and every iteration. Perhaps there were problems with the way we implemented the code and this will be explored in future analysis[8]. If there is an error, then the best model was the neural net. The worst model was the one involving multinomial methods on the regular data.This makes sense because the multinomial method is a regression techniquee that is obviously affected by a high number of features. The ensemble of principal component analysis and multinomial performed better.

Also, regarding the first question, it seems that there was some distinction between the three groups with respect to k-means clustering. There was double the amount of Crohn's disease patients in clusters 4, 5, and 6 when compared with clusters 1, 2, 3. For ulcerative colitis, it seemed a little more spread out and most of the "normal" observations were grouped into the bottom two clusters.

IV. Conclusion

The case study asked us two look at two questions regarding gene expression data and how it related with the prevalence of Crohn's disease and Ulcerative Colitis. Through our analysis, we found that there was some pattern to the way that the different groups were being clustered. This was shown on a plot that displayed results from a k-means clustering implementation. Next, we tested many models to see how well they performed in predicting categories. In the

[8]I provide some information about this in the appendix

end, it seemed that the *xgboost* model performed the best (however see the note on this) and the multinomial model on the regular data seemed to perform the worst. If we exclude the *xgboost* model then the next best one was the neural net. In the end, while initially it seemed very difficult to build decent models or to look for patterns in such (seemingly) ambiguous data, we seemed to have come up with some initial results that seem promising!

V. References

- Dr. Davis's Code (and his help)
- https://www.quora.com/What-are-the-advantages-of-different-classification-algorithms
- http://neuralnetworksanddeeplearning.com/chap1.html
- http://en.proft.me/2017/01/22/classification-using-k-nearest-neighbors-r/
- https://en.wikipedia.org/wiki/Gradient-boosting
- https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
- James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R. Springer, 2017.

VI. Appendix

Note about *xgboost*: We have narrowed down the problem to a data structure conversion of our data. It seemed initially that *xgboost* only took specific matrix types (densely sparse matrix?) however, we attempted to run it with a regular matrix and it compiled. After looking at these results, we noticed success rates between 80 to 90 percent. This seemed much more reasonable and this also tells us that there has to be an incorrect application of *xgboost* between our two.