462Final

Charles Gauthey

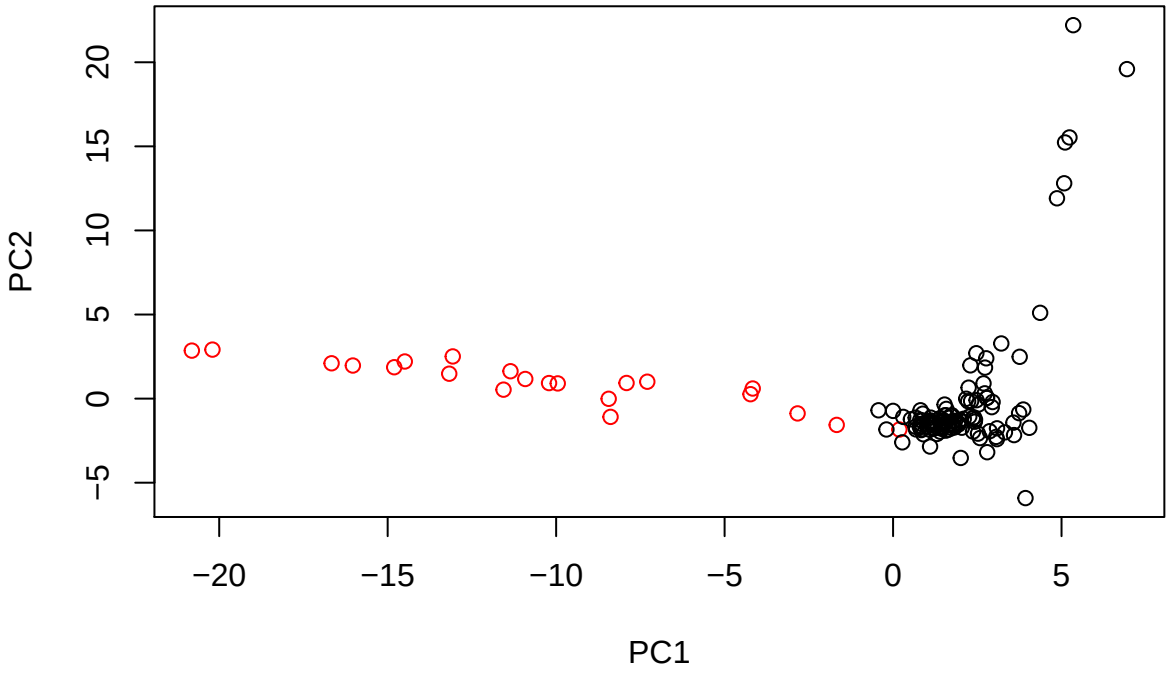
5/12/2018

##Introduction

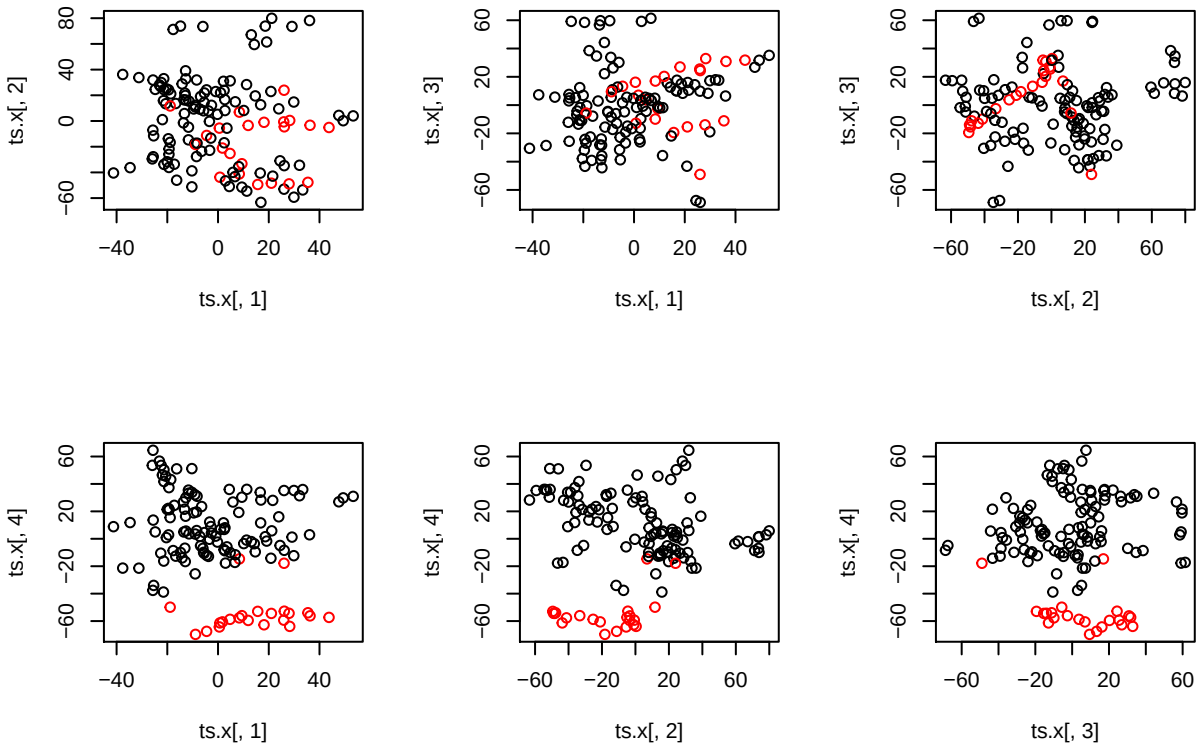
#Introduction

To discover if there was any structure in the data, we began with pca, and then moved on to a technique called tsne for visualizing the data.

After scaling the variables, we ran pca, and the first 2 principal components are shown below



We then moved on to another technique called t-distributed stochastic neighbor embedding, which is a technique for dimensionality reduction that tries to preserve the topology of the underlying data, and at the same time reduce the dimensions. For this we used the tsne library. Below are some visualiziations of the data after applying tsne to reduce to 4 dimensions.



As you can see, several of the plots show a separation between each of the 2 classes.

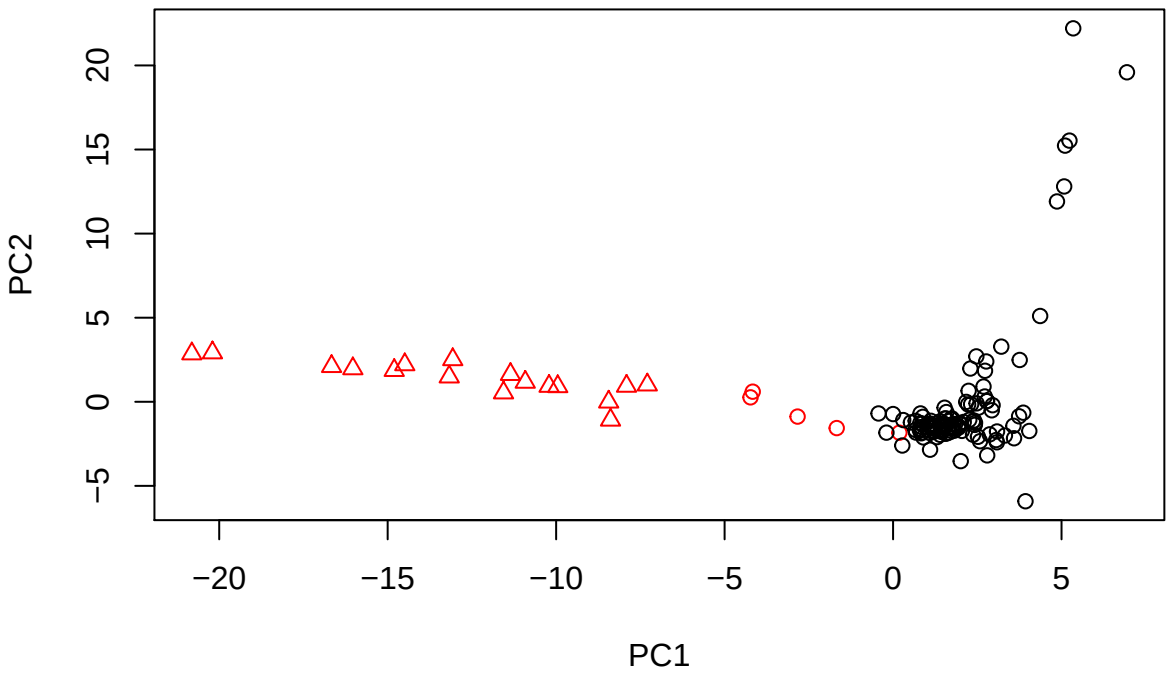
##Unsupervised learning

We looked at 2 ways to do the unsupervised learning.

We initially began by running PCA on the data set after scaling, and then used that to train a kmeans model with 4 data columns.

Below is a confusion matrix, and a graph of the first 2 columns of the PCA. The shape of the point corresponds to the predicted class label, while the color corresponds to the true class label

```
##           True_Labels
## Predicted   0   1
##         1 111   5
##         2   0  17
```




```
## [1] "Adjusted Rand Index: 0.807042757982835"
```

We like this technique because it is fairly simple and robust to changes in the size of the dataset.

Our second model, somewhat more creative, used unsupervised random forest based distances to train an average linkage based heirarchical cluster. Because this model got things correct most of the time, but not all of the time, we replicated it 20 times and had them vote on the data points. This makes the model more robust, and illustrates the concept that imperfect learners working together can do a good job creating a better learner. This workes extremely well, and results in the best clustering that we were able to get. We set ntrees to 500 as it was as small a number as we could get without sacrificing the effectiveness of the model.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
predictions = replicate(20 ,{

  rf = randomForest(~ ., data = x, ntree = 500, proximity = TRUE, oob.prox = TRUE)
  hc = hclust(d = as.dist(1- rf$proximity), method = 'average')

  ##Makes it so that the clusters align with eachother to make voting smoother.
```

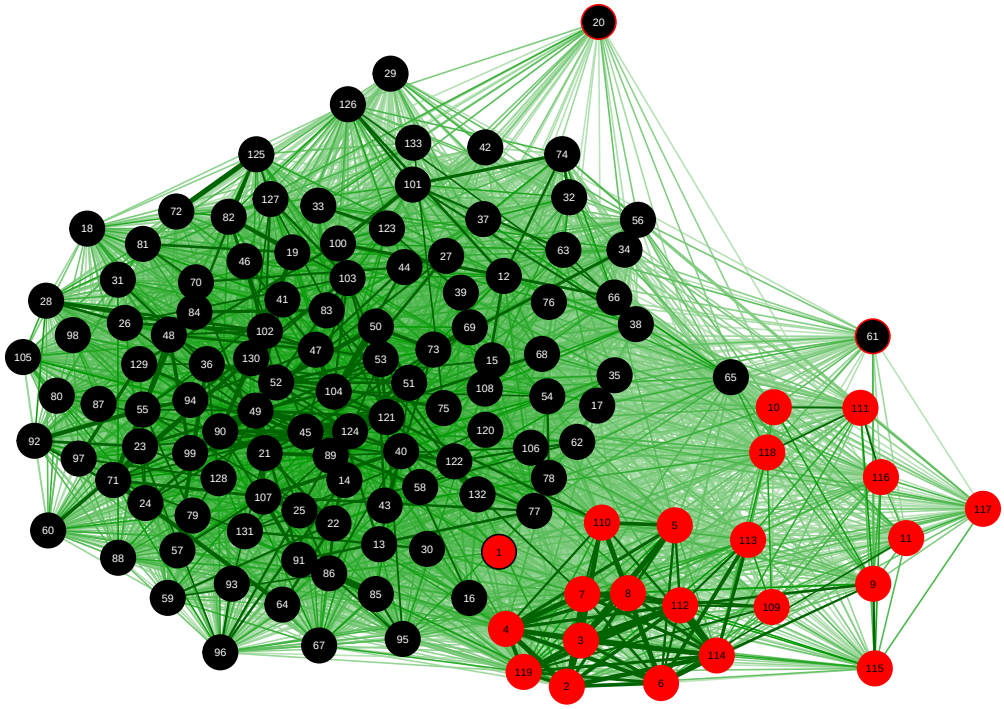
```
  if(mean(cutree(hc, 2)) > 1.5){
    -cutree(hc, 2) + 2
  }
  else{
    cutree(hc, 2) - 1
  }

}
)
print(paste0("Adjusted Rand Index: ", adj.rand.index(rowMeans(predictions) > .25, y)))
```

```
## [1] "Adjusted Rand Index: 0.890582547209612"
```

The below graph shows each of the data points, along with the similarity that the random forest has created. You'll notice that there are 2 neighborhoods of similarities, and they correspond pretty well to both the true class (The color of the circle), and the predicted cluster(the border of the circle).

```
## Warning: package 'qgraph' was built under R version 3.6.1
```

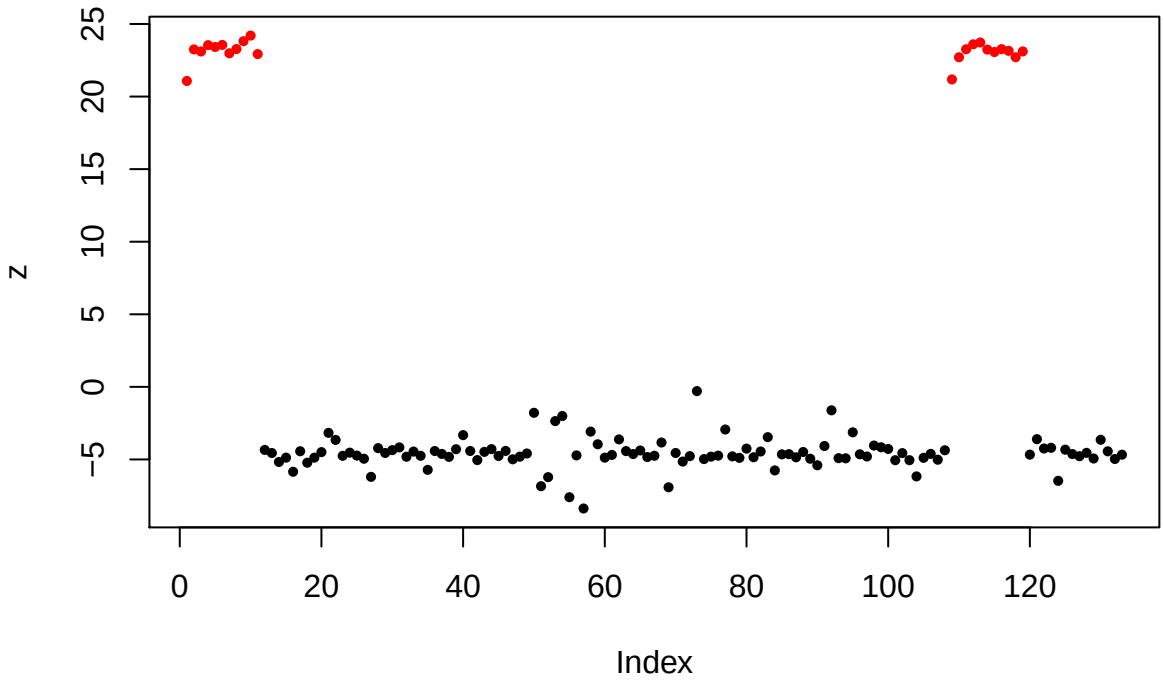
```
## Registered S3 methods overwritten by 'huge':
##   method    from
##   plot.sim  BDgraph
##   print.sim BDgraph
```



Supervised Learning

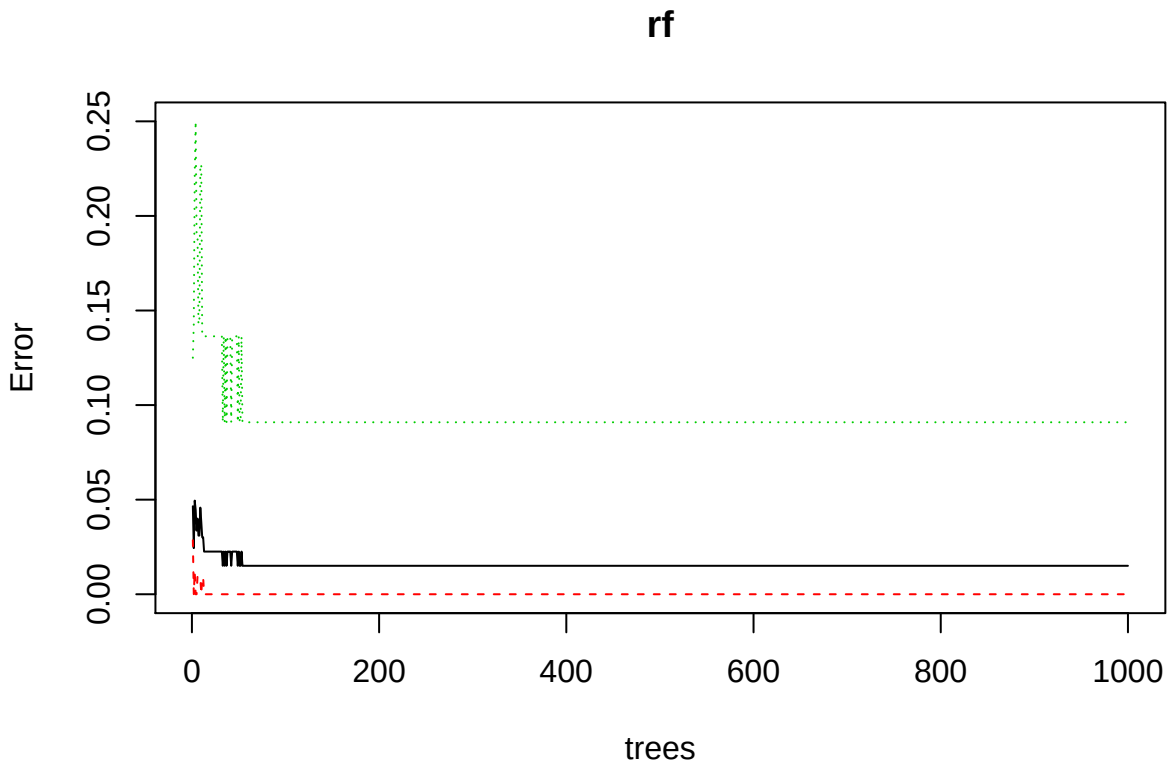
We tried three different supervised learning methods, Lda, GBM and Random Forest. We began with GBM and Random Forest because the models tend to perform well on large data sets, and because they are tree based, can generally handle many factors well. The lda performed noticable worse than the other 2 methods. We tuned the GBM and Random Forest using a grid search and cross validation. We used LDA from mass, and it had an output of a single linear discriminant. When visualized, it actually separated the data out well, but when cross validated, it did not perform as well as hoped.

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

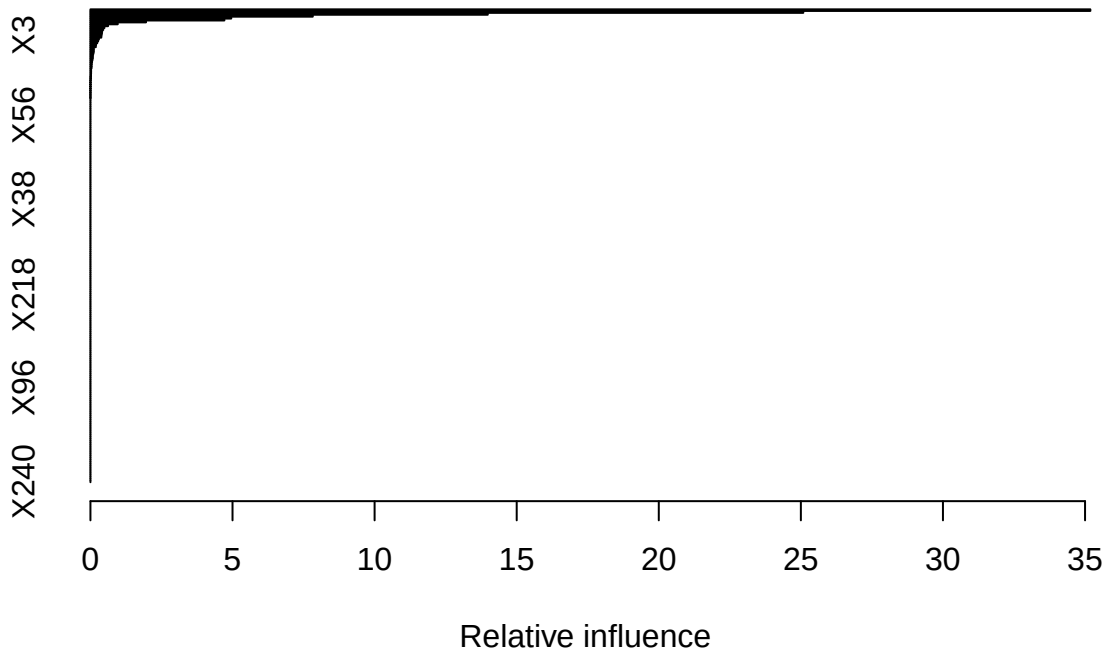


For the random forest we ran it on all of the data, without modifying or transforming, as random forest is robust to scale. Below is a plot of the expected error including the oob error.

```
##
## Call:
##  randomForest(formula = factor(Y) ~ ., data = df, mtry = 50, strata = Y,      importance = TRUE, ntr
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 50
##
##         OOB estimate of  error rate: 1.5%
## Confusion matrix:
##     0  1 class.error
## 0 111  0  0.00000000
## 1   2 20  0.09090909
```



For the gbm we ran it on all of the data, without modifying or transforming, as boosted models are robust to scale. We did weight the importance of the 2 classes, with each Y = 1 observation, being weighted as 5x higher than the Y = 0 observations, to ensure that the unbalanced data did not result in a bad model. With this done, looking at the relative influence, we can see that some variables have much stronger predictive power than others.




```
##       var    rel.inf
## X33   X33 35.1871454
## X206 X206 25.0841866
## X19   X19 13.9808079
## X142 X142  7.8232334
## X187 X187  4.9548471
## X20   X20  4.7124350
## X139 X139  1.9554296
## X220 X220  0.9603428
## X32   X32  0.6191468
## X133 X133  0.4910416
## X117 X117  0.4498342
## X26   X26  0.4134638
## X196 X196  0.4042896
## X3     X3  0.3900306
## X112 X112  0.3822556
## X27   X27  0.2980679
## X226 X226  0.2704488
## X144 X144  0.2378251
## X109 X109  0.1949124
## X149 X149  0.1882929
```

Overall, we chose the parameters for the Random forest and GBM through a grid search. For the Random forest, we modified Mtry and ntree, the number of variables looked at for each tree and the number of trees respectively. We found that changing both of these changed very little, and we settled on 50 for mtry and

1000 for ntry. With the boosted model, we went with a shrinkage parameter of .001 and a ntrees of 1000. Overall, both of these had pretty similar error rates during cross validation.