Detecting Parkinson's Disease Through Audio Recordings

Jeremy Giese
School of Computer Science and Electrical Engineering
Marshall University
Huntington, West Virginia 25701
giese@live.marshall.edu

Alex Canfield
School of Computer Sciences and Electrical Engineering
Marshall University
Huntington, West Virginia 25701
canfield17@live.marshall.edu

***Abstract*—This is a project performed to determine the effectiveness of a random forest's prediction of Parkinson's Disease. A dataset from the UCI database is used. The results show that, with fine-tuning, a random forest can predict at 84% accuracy if a patient has Parkinson's.**

***Index Terms*—Machine Learning; Bioinformatics**

Introduction

Parkinson's disease is a neurodegenerative disorder known for causing severe tremors and deterioration of motor functions. According to the Parkinson's Foundation, about 1 million people in the United States live with Parkinson's disease, and about 60,000 people are diagnosed each year [1]. There is no cure for Parkinson's disease, but there is research being done on how to more effectively treat and diagnose this disease. J. Hardy and et al. states that genetic markers can be used to diagnose pre-symptomatic Parkinson's patients, while diagnosis testing is required for invasive DNA tests [2]. These invasive methods have been proved that they are not suitable for diagnosis. Recent bioinformatics research has focused on developing noninvasive methods of diagnosing Parkinson's patients who have not yet experienced symptoms. L. Naranjo and et al. presented a tool that relies on small changes in the voice to determine, at an early stage, a patient's diagnosis for Parkinson's. The tool takes recordings of a patient saying a sustained /a/ vowel sound and performs calculations on that recording to determine factors: fluctuation in pitch, frequency of the sound, and the period of the soundwave.

Our Project

Thanks to the UCI machine learning repository, we now have access to the dataset that these researchers used to develop their Expert System. [3] It contains 240 measurement samples from 80 patients, 40 with Parkinson's and 40 without. Using this dataset, our group would like to attempt a classification algorithm of our own. If possible, we would like to improve on the success rate of the Expert System developed in 2015 which saw success rates between about 70% and 85%. [3] Using this data, we will determine how data features are related and which features indicate Parkinson's disease. Using that information, we can then leverage R classification packages to build a diagnosis model. The package candidates for such a model include the randomForest and caret packages, but more research will be done on this. After our classification model has been constructed, we will evaluate its performance, comparing it to the Expert System. It would also be interesting to see what kinds of visualizations we can produce to show what factors impact our algorithm's decisions. The algorithm, results, and visualization would all be part of the final product for this project. While this project is not entirely original, an attempt to improve and learn from this project will certainly be worthwhile.

Hurdles

The main problem in this project is that, as the dataset notes point out, there are three recording measurements for each patient. Therefore, those three data entries cannot be assumed to be independent. Determining how to manage these dependent data entries will be a challenge for our project. Related Works

The following works provide the basis of knowledge for our Parkinson's Disease (PD) classification project. The most important sources from the list are the sources from the original PD classification project by Naranjo, Lizbeth, et al. [3], [4] The two papers surrounding this work give us background on how the metrics in this dataset were computed. These papers also provide discussion on how their classification algorithm was created. Perhaps most importantly, these papers give us a benchmark to compare our results against and a discussion of limitations of this previous project. Given that our dataset was created for the project described in Naranjo, Lizbeth, et al, these papers serve as the starting point for our project. [3], [4] Another important aspect for this project is that our team has an understanding of Parkinson's Disease. Why are we able to use voice recording measurements to detect PD? Why is diagnosing PD a problem worth working on? Our sources on PD answer questions such as these. They provide our formal definition of PD, statistics on PD, and a medical basis for why voice analysis is a viable diagnosis method. [1], [5] The next most important source for our project is the dataset itself taken from the UCI Machine Learning Repository. [6] This dataset contains a series of measurements taken from voice recordings of 80 people: 40 with PD, and 40 without. Each patient has 3 recordings in this dataset, meaning that these data points are dependent on one another. Working around this will be

one limitation of our project. Many of the other sources deal with the construction of our classification model in R. Given that we are new to R, our project needed information about the classification packages available. Afroz presents several machine learning packages in R. [7] She provides a short discussion for each package and where that package is most effective. This article also provides links to the documentation of each package. Since we have never used any machine learning packages in R before, this article was extremely helpful in starting to understand which packages will be best for our project. From our research, we determined that one potential classification package we might use would be the Caret package in R. Pierobon presents a workflow using the Caret and other related packages to build a classification model in R. [8] More than just detailing how to use the Caret package, this article takes us through a project from start to finish. Especially helpful are the techniques presented for determining what data attributes most impact the classification result. This source also provides information on how to prepare data in R to be used with the required packages. While the sample project presented in this article is not applicable to our project, the techniques detailed will be very helpful. Another candidate package for this project is the R Random Forest package. This is a very popular classification algorithm right now, and part of our project will be looking at which classification algorithm performs best with PD diagnosing and our data. Maklin presents an article on how to correctly implement the Random Forest package in R. [9] Assuming we will at least attempt using this package, this article provides useful information on the algorithm and the package implementation. The second part of our project is evaluating how accurately our different models can classify patients with PD and patients without. Brownlee presents a detailed exploration of all the ways to test a classification algorithm. [10] His article also details how to ensure that you have chosen the correct model. This article provides a thorough method of determining model validity which our group will find useful for our project, especially given our unfamiliarity with R. Another useful article from Brownlee has to deal with the visualization side of our project. In his article, Brownlee details different ways to visualize data and relationships in R.[11] This provides instructions on different techniques to visualize data and results using various R packages. Visualization will be especially useful in our project in determining what data attributes are related to each other. It will also be helpful for displaying the results our model yields in an effective way. This article provides a good starting point for data visualization that we can use in our project. Chakure discusses how a random forest can create highly accurate classifiers for non-noisy datasets.[12] He also details how a random forest model can accept many input variables. Chakure defines two different approaches to classification: Boosting and Bagging. Boosting uses multiple learning algorithms to classification results. Bagging repeats the same model and uses the average across iterations to improve results. In our case we will consider bagging over boosting as this model will better fit our dataset. The article from Andrew Ng, details a method of reducing multiple, related attributes down into one, scalar value.[13] Ng discusses and demonstrates how to perform dimensionality reduction. He also provides instructions on how to use packages for data reduction using Principal Component Analysis.[14] Along with his demonstrations, Ng provides visualizations of these methods which we may find useful in our project. Hardy J. et al., researched how to diagnose Parkinson's before patients show symptoms.[2] However, the tests their research yielded were far too invasive to be done without reason. It was the results from this research that sparked the hunt for non-invasive diagnosis testing. Klein C. and Westenberger A. showed that there are different genetic markers that can show a person's likelihood for getting PD.[15] While we do not have any data on these genetic markers, adding such data would be an interesting addition to our project. The NICE Clinical Guidelines details information about Parkinson's and its criteria for diagnosis.[16] The details state that it can be difficult to confirm a patient needs treatment for Parkinson's even with diagnostic proof that they carry the disease. This is because even if they have active markers for the disease they may not suffer any symptoms from the disease. One limitation we may encounter is the size of our dataset. Given how small it is, our model may appear more accurate than it would be with a larger dataset. Another limitation will be dealing with the dependent entries in the dataset. One solution to this is to reduce the dimensionality, but that may not result in a correct classifier. Han S. and Kim H. have done research to determine the optimal size of the feature set used in a Random Forest.[17] They have information on what seemed to work the best for them and what worked the worst. Their research can help us determine how we can optimize a random forest model with our dataset. In "Random forest versus logistic regression: a large-scale benchmark experiment" they showed that logistic regression performs worse than random forest in three different benchmarks.[18] This article confirmed for us that we should use a random forest model instead of logistic regression. Downward E. defined the stages of PD in her article. [19] This article confirms that voice detection is one of the earliest methods of diagnosing PD before more serious symptoms. Finally, we examined the official criteria needed to be diagnosed with PD.[**?**]This article confirms that small voice changes, and other factors, can be strong indicators of PD in a patient. These articles on Parkinson's disease and how it is diagnosed give our work on this project a strong base to sit on medically. From that side, we know that what we are trying to detect and classify is a sign of Parkinson's. Now we have to build a model that will accurately classify the voice recordings we have.

Proposed Methods

This project will take the form of an R script. Using data from a .csv with voice recording information, our script will build a predictive model to determine whether a patient has Parkinson's Disease (PD) or not. Before our project begins, our R script will take care of any preprocessing that needs to occur.

The script will read in our .csv file, check that everything was read correctly, ensure that data is of the correct data type, and make any necessary corrections. The idea is that this script could be run given any .csv file with voice recording data. The final important feature from this preprocessing step will be to reduce our dataset. Our dataset contains three separate recordings per patient. Given that these are not independent of each other, we will have to combine them. This will take some trial and error, but our research shows that we can merge these three data entries using the mean or median values for each attribute. We will have to see which of these techniques works best. After preprocessing, the first aspect of this project will be an analysis of our dataset. We will use the corrplot package to create a visualization of our dataset's attributes and how they relate to each other. This plot will display the relationships between each attribute and its correlated value to a patient having PD. The charts from this package will not only help with creating the script but will provide the reader with some context for the relationships between attributes in this dataset. Another useful visualization might be a Correlation plot from the chart package, which would be used in the same way. Using these correlation values, we can then find p-values to test our hypothesis. From this point, it may be useful to reduce the dimensionality of our dataset using a PCA reduction. This would make our prediction a bit simpler and would combine many of the similar measurement values. Our dataset has a little less than 50 attributes for each record, so we will have to eliminate some of the attributes for our prediction engine to reasonably handle such a dataset. We can then use the caret package in R to create a model that will indicate which of our reduced attributes indicate the PD in a patient. This will help us determine which reduced features our model should focus on. Depending on the accuracy of our results, this data may come in handy for troubleshooting any issues with our predictions later on. From here, our data will be ready to use and we can split our data into test and training data. We will only have 80 data points because there are only 80 patients. The split between test and training data will have to be determined partially by trial and error. We want a large enough training set to create an accurate model but not so large that our test set is too small to give any useful predictions. We will split our data using the caTools package in R. We can then take our training data and use it to build a classification model using the Random Forest algorithm. This algorithm is easily available with the randomForest package in R. Once our classification model is created, we can then use our model to predict whether the patients in our test data have PD or not. Using the results from this predictive model on our test data, we can then evaluate how effective our model is at predicting PD. We can make this evaluation by looking at a confusion matrix of our predictions. The data from this matrix will allow us to determine how accurate our classification model is. We can then work on improving this accuracy by changing how we split our data, changing how we reduce dimensionality, changing how we combine patient records, and other aspects that go into our model. There will be some trial and error to this project, but we would like to see some level of improvement on our initial results by the end. We can then use packages like graphics, and lattice in R to help visualize the results we get. We can use the visreg, and forestFloor packages which target visualizing Random Forest's functions and learning specifically. These methods will help make our model more understandable to the reader. These visualizations may also indicate where our model can be made more accurate in the future. The specifics on result visualization will be partially determined by the results themselves, so that decision will be made later on.

Parkinson's Disease Detection

The goal of this project is to create a prediction model for detecting Parkinson's Disease (PD) by using voice recording metrics from a patient. The major aspects of our project include a correlation analysis, hypothesis testing, building a prediction model, and an analysis of that model. This methods paper will primarily focus on the first three of those tasks. There is a brief analysis at the end, but the analysis will be the focus of our next paper. The main goal of this paper is to detail the model creation process from reading in our dataset to obtaining results from our model.

First, we need to read in our dataset. This dataset contains metrics on 240 voice recordings of patients making a sustained /a/ vowel sound. There are three recordings per patient, 40 patients with PD and 40 without PD. We do not have access to the recordings themselves, as those are subject to HIPAA restrictions, but Naranjo developed a system to take these audio files and calculate metrics such as pitch, fluctuation, and jitter. Naranjo showed that these metrics can be used to detect PD in a patient. This is based on the fact that PD affects the vocal cords before becoming traditionally symptomatic with hand tremors and other symptoms. This project seeks to validate these results and, if possible, improve on them.
Here, we read in our dataset and import the libraries that this project will require. Most of these will be used towards the end when our model is created.

```
library(corrplot)
library(randomForest)
library(caret)
library(e1071)
require(caTools)

pd.data = read.csv("parkinsons.csv")
dim(pd.data)
```

```
## [1] 240  48
```

```
sapply(pd.data, class)
```

```
##          ID  Recording     Status
##    "factor"  "integer"  "integer"
## Gender Jitter_rel Jitter_abs Jitter_RAP
##  "integer"  "numeric"   "numeric"  "numeric"
```

```
## Jitter_PPQ    Shim_loc     Shim_dB
##  "numeric"  "numeric"  "numeric"
## Shim_APQ3 Shim_APQ5  Shi_APQ11
##  "numeric"  "numeric"  "numeric"
##  HNR05    HNR15      HNR25
##  "numeric"  "numeric"  "numeric"
##  HNR35      HNR38        RPDE
##  "numeric"  "numeric"  "numeric"
##  DFA        PPE        GNE
##  "numeric"  "numeric"  "numeric"
## MFCC0      MFCC1      MFCC2
## "numeric"  "numeric"  "numeric"
## MFCC3      MFCC4      MFCC5
##  "numeric"  "numeric"  "numeric"
##  MFCC6      MFCC7      MFCC8
##"numeric"  "numeric"  "numeric"
##    MFCC9     MFCC10     MFCC11
##  "numeric"  "numeric"  "numeric"
##   MFCC12  Delta0     Delta1
## "numeric" "numeric" "numeric"
##    Delta2     Delta3     Delta4
##  "numeric"  "numeric"  "numeric"
##   Delta5     Delta6  Delta7
## "numeric"  "numeric"  "numeric"
##  Delta8     Delta9    Delta10
##"numeric" "numeric" "numeric"
## Delta11    Delta12
##"numeric" "numeric"
```

Data Preprocessing: The first challenge we must correct is the problem of independence in our dataset. Our dataset has 240 recordings for 80 patients, so each patient has three recordings. For each patient then, those three recordings are not independent. Most prediction algorithms require independent data entries, so this is a problem we must correct before trying to predict PD in patients. Our solution to this problem is to combine the three data entries per patient into one data entry per patient. This will give us 80 independent data entries: 40 patients with PD, and 40 without PD.
Our dataset has 48 attributes. 44 of those attributes are numeric recording metrics. The next question is how to combine the values in each numeric column of the three dependent rows. We chose to take the mean of the three values. There are other possibilities for combining the values of each numeric column, but taking the mean is a simple and effective way of representing these three values.

```
#Create an empty data frame with
#the same columns as our dependent
#data frame
pd.data.ind = pd.data[FALSE,]

#Get the patient ID's for all our patients.
patientIDs = levels(pd.data$ID)

# Now we get the rows that are
#dependent, and merge them into one row.
for(i in 1:length(patientIDs)){
  dependent.rows = pd.data[
    grep(
      patientIDs[i],
      pd.data$ID
      ),
    ]
  newRow = list()

#We have our three dependent rows,
#we iterate over the columns and combine
#them into a single row.
  for(i in 1:ncol(dependent.rows)){
    #For the first few rows, we just need
    #the first value. These are static
    if(i <= 4){
      newRow[i] = dependent.rows[1,i]
    }
    else{
      column = dependent.rows[,i]
      newRow[i] = mean(column)
    }
  }

  #Here we add our new, combined row,
  #onto our independent dataset.
  names(newRow) = names(dependent.rows)
  pd.data.ind = rbind(pd.data.ind, newRow)
}

#Here we are changing our ID variable back
#into a factor and printing information
#about our dataset
pd.data.ind$ID = factor(
  pd.data.ind$ID,
  levels= pd.data.ind$ID,
  labels=c(patientIDs)
  )
dim(pd.data.ind)
```

```
## [1] 80 48
```

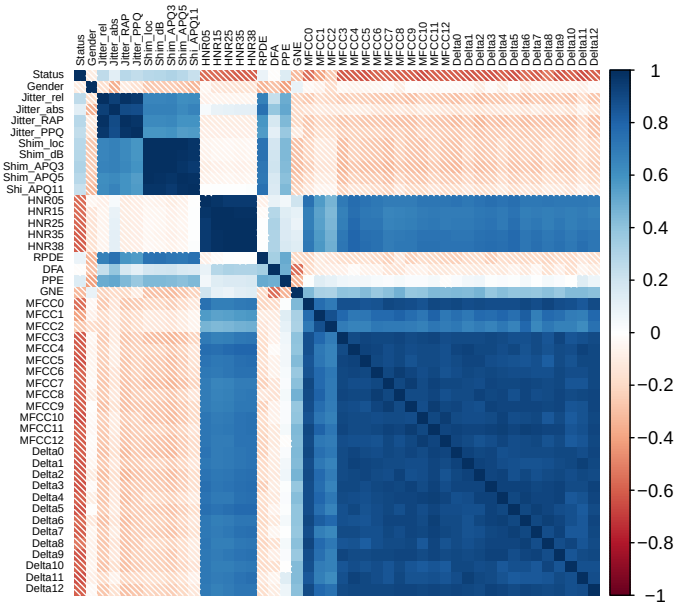
```
sapply(pd.data.ind, class)
```

```
##          ID  Recording     Status
##    "factor"  "integer"  "integer"
## Gender Jitter_rel Jitter_abs
##  "integer"  "numeric"  "numeric"
## Jitter_RAP Jitter_PPQ   Shim_loc
##   "numeric" "numeric"  "numeric"
##   Shim_dB  Shim_APQ3  Shim_APQ5
##  "numeric"  "numeric"  "numeric"
##      Shi_APQ11      HNR05    HNR15
##  "numeric"  "numeric"  "numeric"
##     HNR25      HNR35      HNR38
```

```
##  "numeric"  "numeric"  "numeric"
##      RPDE        DFA        PPE
##  "numeric"  "numeric"  "numeric"
##       GNE      MFCC0      MFCC1
##  "numeric"  "numeric"  "numeric"
## MFCC2      MFCC3      MFCC4
##  "numeric"  "numeric"  "numeric"
## MFCC5  MFCC6      MFCC7
##   "numeric"  "numeric"  "numeric"
## MFCC8      MFCC9     MFCC10
## "numeric"  "numeric"  "numeric"
## MFCC11     MFCC12  Delta0
## "numeric"  "numeric"  "numeric"
##     Delta1     Delta2     Delta3
## "numeric"  "numeric"  "numeric"
##     Delta4     Delta5     Delta6
## "numeric"  "numeric"  "numeric"
##    Delta7     Delta8     Delta9
## "numeric"  "numeric"  "numeric"
##    Delta10    Delta11    Delta12
## "numeric"  "numeric"  "numeric"
```

Correlation Analysis: You can see that we now have a data frame of 80 independent data recordings with all the same attributes as our dependent recordings. Now we can start developing a model of PD prediction. To start, we look at the correlation value of each attribute with PD presence (the 'Status' attribute). We will use Corrplot to get an initial look at these relationships. While these correlation values will not be used directly in our model, it will be helpful to see which values appear to have a linear relationship with a patient having PD or not. The most effective way to visualize this kind of relationship is with a heatmap.

```
corVals = cor(pd.data.ind[,-1:-2])
corrplot(
  corVals,
  method="shade",
  tl.cex=0.5,tl.col = "black"
  )
```



This graph here gives us an idea of which recording metrics are linearly related to PD in a patient. We are focused on the top row of this chart which tells us the attributes that have a linear relationship with PD (Status). This chart shows us that MCF and Delta attributes have a negative linear relationship with PD, but it is hard to see exactly how strong that relationship is. The next step is to see how strongly related our attributes are to PD in a patient. We don't necessarily need to know the correlation value for every attribute, but knowing the extremes will give us an idea of the extent of the linear relationship.

```
# Find the most correlated
#attributes to Status (PD).
corrVals = c()
for(i in 4:ncol(pd.data.ind)){
  corrVals[i-3] = cor(
    pd.data.ind[,i],
    pd.data.ind[,3]
    )
}

head(corrVals)
```

```
## [1] -0.1020621  0.2516473  0.1263721
0.2742858  0.2483064  0.2792990
```

From these results, we see that our strongest linear relationships are negative. This also confirms what our corrplot chart indicated. Most of the relationships with PD are negatively linear. We can tell this because the median of our correlation values is very close to the minimum correlation value. We now have some grounds to say that PD and many of our dataset attributes are somewhat linearly related. The next step for our project is to test the hypothesis that our dataset attributes indicate PD in a patient. The null hypothesis here is that these attributes give no indication of PD in a patient.

Hypothesis Testing: Our explanatory variable in this hypothesis test is PD in a patient, and our response variables would be the voice recording metrics. Given that our explanatory variable is two-tiered and our response variables are quantitative, we can use a t-test to test this hypothesis for each of our dataset's attributes. Using the p-value from these tests, we can get an idea of how many of our attributes indicate PD in a patient.

While this data will not be used directly in our prediction model, it is an important step in confirming that the data we will use in our model truly does indicate PD in a patient. Without this test, we risk building a model with data that does not actually indicate PD. The hypothesis testing here is a necessary step in ensuring that our model is based on legitimate relationships in the data.

```
#this takes the index of the
#column you want and returns
#a p-value with the status attribute.
extractPVal = function(column){
  formula = paste(
    names(
      pd.data.ind)[column],
    " ~ Status"
    )
  return(
    t.test(
      as.formula(formula),
      data = pd.data.ind
      )$p.value
    )
}

#We can then calculate the p-values
#between Status and the other
#attributes with Status as our
#explanatory variable and the
#recording attribute as our
#response variable.
pValues = rep(NA, ncol(pd.data.ind)-3)
pValues = sapply(
  4:ncol(pd.data.ind),
  extractPVal
  )
pValues = p.adjust(pValues, method="BH")

#This will print out the
#number of adjusted p-values
#that reject the null hypothesis.
print(
  paste("Number of significant
        adjusted p-values (<= 0.05):",
        sum(pValues <= 0.05)
        )
  )
```

```
## [1] "Number of significant
         adjusted p-values (<= 0.05): 40"
```

```
print(
  paste("Number of non-significant
        adjusted p-values (> 0.05):",
        sum(pValues > 0.05)
        )
  )
```

```
## [1] "Number of non-significant
        adjusted p-values (> 0.05): 5"
```

```
print(
  paste(
    "Number of significant
    adjusted p-values (<= 0.01):",
    sum(pValues <= 0.01)
    )
  )
```

```
## [1] "Number of significant
    adjusted p-values (<= 0.01): 32"
```

```
print(
  paste("Number of non-significant
        adjusted p-values (> 0.01):",
        sum(pValues > 0.01)
        )
  )
```

```
## [1] "Number of non-significant
        adjusted p-values (> 0.01): 13"
```

To test our hypotheses, we performed multiple hypothesis testing with each attribute of our dataset against the presence of PD in a patient. We then adjusted the p-values from this process using the Benjamini-Hochberg procedure and set our significance value at 0.01. Our results indicate that we can reject the null hypothesis for 32/45 of our tests. With a significance of 0.05, we can reject the null hypothesis for 40/45 of our tests. This gives us strong reason to believe that most of our attributes indicate PD in a patient.

Given that we are using adjusted p-values, we also have an approximated false-positive rate of 1% or 5% for our hypothesis tests, both of which are very low. Again, while these data will not be used in our prediction model, it establishes that the data we want to use really does indicate PD in a patient with a reasonable level of confidence.

Based on the results from our hypothesis tests, it is now reasonable for us to try and predict PD using the data from this dataset. For our project, we have decided to do so using the RandomForest model. RandomForest is a popular, strong prediction model that is fairly simple to implement in R. We will use this model and our independent dataset to predict PD in patients.

We considered running a PCA on our dataset to reduce the

number of attributes, but we only have 80 data entries. Even with 48 attributes in our dataset, the compute time should be relatively quick since we have so few data entries. For this reason, we are going to build our prediction model on our independent dataset, unmodified.

Prediction Model: Setting up our Random Forest model requires some processing of our dataset. The first thing is to change some of the datatypes of our attributes. These were fine as other datatypes in our previous analyses, but the random forest package has certain requirements for datatypes to function. We start by making those necessary changes, turning some attributes into factors. We have already imported the necessary packages at the beginning of our code. The RandomForest package is used to build our model. The caTools package is used to split our data into testing and training data. The caret package is used for our confusion matrix in evaluating the model. We also reset the random seed before building our model. Setting our random seed will allow us to replicate the same results every time we run this model. With these changes made, we are now ready to build our model.

The first step is to separate our dataset into test data and training data. We will explore later what the "optimal" split is, but for now, we chose an arbitrary split of 60% training and 40% testing data. Our random forest model will be built with the training data, and its predictions will be made using the testing data.

```
# we need to make these
#variables factors for our prediction model
pd.data.ind$Status =
  as.factor(pd.data.ind$Status)
pd.data.ind$Gender =
  as.factor(pd.data.ind$Gender)

# In the pd.data.ind the Status variable
#must be a Label to use Classification.
#Otherwise the forest uses regression.

set.seed(42)
#it seems that the random forest
#uses the random function
#so setting the seed keeps
#the results consistent

#We need to split our independent
#records into training and testing data
sample = sample.split(
  pd.data.ind$Status,
  SplitRatio = 0.6
  )

#Creating the training/testing
#sets along our split.
train = subset(
  pd.data.ind,
  sample==TRUE
  )
test = subset(
  pd.data.ind,
  sample==FALSE
  )

#Shows the dimensions of our training set
#vs. our testing set.
dim(train)
```

```
## [1] 48 48
```

```
dim(test)
```

```
## [1] 32 48
```

You can see that our training data uses 48 rows, and the testing data uses 32 rows from our split. After splitting our data, we are ready to build our random forest model. Doing so is fairly simple with the RandomForest package in R. You may notice that our formula is slightly different from the one we used in our t-tests. This is to accommodate the RandomForest package and to indicate that, for our model, we want to predict Status (PD) and not any of our other attributes. We also indicate here that we want our model to be built off the training data. From this training data, we tell our model to ignore the first two columns: ID and Recording. These attributes have nothing to do with predicting PD, so we do not want them to skew our results. With this section of code, we have built our random forest model.

```
#Now we create our RandomForest model
#using the data we just split off
# Our formula is different from our
#t-tests because here we are placing
#Status as the item to be predicted
rf = randomForest(
  Status ~ .,
  #We ignore the ID and Recording
  #attributes in our model.
  data = train[,-1:-2]
)

#prints out the model information
rf
```

```
##
## Call:
##  randomForest(
## formula = Status ~ ., data = train[, -1:-2]
##   )
##     Type of random forest:
##           classification
##
##          Number of trees: 500
## No. of variables tried
## at each split: 6
```

```
##
##          OOB estimate of
##          error rate: 20.83%
## Confusion matrix:
##    0  1 class.error
## 0 19  5   0.2083333
## 1  5 19   0.2083333
```

From our random forest model, we can see that the estimated error rate is about 20%, and we also see a confusion matrix from the building of our model. Keep in mind that this model was built with a 60/40 split in our data, so a 20% estimated error rate is not so bad. After an initial evaluation, we will look to see how other splits in our data affect the model.

For now, we can use this model to predict the presence of PD in a patient, and we can look at the confusion matrix from this prediction to evaluate the effectiveness of our model. Predicting from our model is easy in R, we just call the predict method from the RandomForest package. We indicate in this prediction call the model we want to use, and the data to be used for this prediction. To see the results from our prediction, we use the caret confusionMatrix method. This method gives us a confusion matrix for our prediction as well as some statistics on our confusion matrix. A discussion of our prediction results is below.

Results and Evaluation

```
#Use our random forest
#model to predict Status.
#This prediction ignores
#Status and Recording in
#its prediction
pred = predict(
  rf,
  newdata = test[-1:-3]
  )

#Print out a confusion matrix
#and evaluation information
#for our model.
confusionMatrix(pred, test[,3])
```

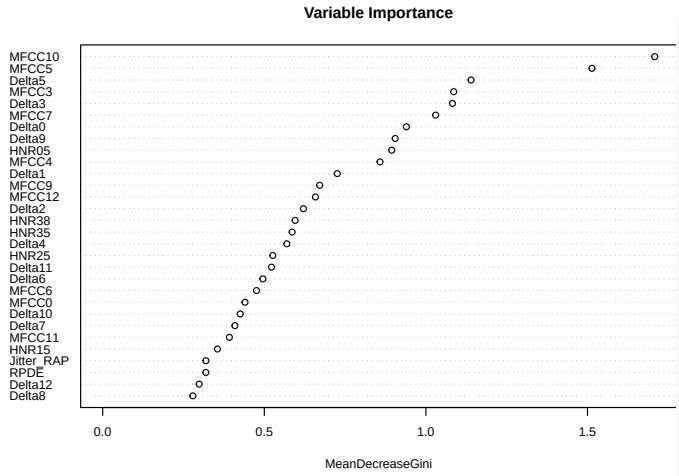
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 12  1
##          1  4 15
##
##     Accuracy : 0.8438
##       95% CI : (0.6721, 0.9472)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 5.654e-05
##
##           Kappa : 0.6875
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.7500
##             Specificity : 0.9375
##          Pos Pred Value : 0.9231
##          Neg Pred Value : 0.7895
##              Prevalence : 0.5000
##          Detection Rate : 0.3750
##    Detection Prevalence : 0.4062
##       Balanced Accuracy : 0.8438
##
##        'Positive' Class : 0
##
```

From the confusion matrix of our prediction, our model has 84% accuracy and a 0.69 kappa value. The P-value from this matrix is also significant. We can see from our matrix that we only had four false positives and one false negative out of 32 predictions. These are promising results, and it shows that we can predict PD in patients from these voice metrics with a fair measure of accuracy.

There are still some questions around our model that we would like to address. For one, what variables played the biggest role in these predictions? How would different test/train splits affect our results? After all, we want the size of our training data to be as small as possible without impacting accuracy. Right now, we have a 60/40 split between training and testing data. How small can we make the training data before we compromise our results? These are the questions that we will address in the final part of our project.

First, the question of examining which variables affect the PD prediction the most. We can do this fairly easily using the varImp caret function. This allows us to see the importance factor of every attribute in our dataset concerning our Random Forest model.

```
#Prints out a plot of
#variable importance for
#our random forest model.
varImpPlot(
  rf,
  main="Variable Importance",
  cex=0.7
  )
```



This chart gives us an idea of which variables in our model have the biggest weight in determining PD in a patient. We can see that MFCC10 and MFCC5 carry much of the weight in our model. This chart provides some insight into what our Random Forest model is considering for its predictions. The next interesting question to consider is the optimal test/train split for our data. We arbitrarily chose a 60/40 split in our data, but how small can we make our training dataset before it starts to impact our results? We can make a chart to visualize the impact of different test/train sizes on our model.

```
# Create a graph showing
#different test/train sizes
#and accuracy.
#Reset our seed so
#we get the same result
#every time.
set.seed(42)
trainSizes = seq(
  from=0.2,
  to=0.85,
  by=0.05
  )
accuracyVals = rep(
  NA,
  length(trainSizes)
  )

#Function to get the
#Accuracy value from a
#given training set size.
getAccuracyValue = function(trainSize){
  sample = sample.split(
    pd.data.ind$Status,
    SplitRatio = trainSize
    )

  #Creating the
  #training/testing
  #sets along our split.
  train = subset(
    pd.data.ind,
    sample==TRUE
    )
  test = subset(
    pd.data.ind,
    sample==FALSE
    )

  #creat random forest model
  rf = randomForest(
    Status ~ .,
    data = train[,-1:-2]
  )

  #make prediction and get
  #accuracy from that prediction.
  #Return accuracy value
  pred = predict(
    rf,
    newdata = test[-1:-3]
    )
  accuracy = confusionMatrix(
    pred, test[,3])$overall[['Accuracy']]

  return(accuracy)
}

# Get an accuracy value for
#every training size we want to consider.
accuracyVals = sapply(
  trainSizes,
  getAccuracyValue
  )

#Plot out our datapoints.
#Accuracy vs ratio
plot(x = trainSizes,
     y = accuracyVals,
     xlab = "Training Split Ratio",
     ylab = "Prediction Accuracy %",
     main = "Training Split Ratio
     vs. Prediction Accuracy",
     ylim = c(0,1),
     type = 'h')
```


```
# See how much our
#accuracy varies based on
#training set sizes.
summary(accuracyVals)
```

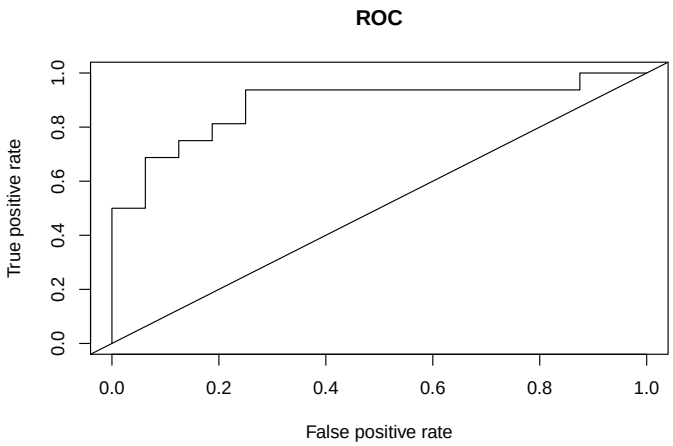
```
##     Min. 1st Qu.  Median    Mean 3rd Qu.
##   0.7083  0.7557  0.8063  0.8062  0.8442
##    Max.
##   0.9167
```

With this plot, we get an idea of how our training/testing split affects the accuracy of our prediction. This data would indicate that the ratio of test/train actually doesn't impact our predictions as much as one might think. In a range from 0.2-0.85 training ratio split, the accuracy of our predictions varies between 70-90% with a mean of 80%. Our chart indicates that even with very small training sets down to 20% of our data, the accuracy of our prediction remains about 80%. This would indicate that our data gives a good indication of PD regardless of sample size. Given how variable human voices are, detecting minute fluctuations in those voice patterns is a challenging task. An accuracy rate between 70-90% is good given that we only have 80 data entries to work with. Having more recording data to work with would likely make our predictions more accurate. Regardless, we have shown through these steps that we can successfully create a prediction model for PD in patients from metrics taken off voice recordings. While the accuracy of our predictions is not perfect, this method provides a quick, non-invasive PD test for asymptomatic patients.

We are trying to make predictions, so the question is how well is our model making those predictions? We have already seen some measures of this. The first being the confusion matrix. We saw from that matrix that our model's accuracy was about 85%. We then evaluated how our training/testing data split affected our model accuracy and found that the accuracy of our model centered around 80% regardless of how much data we used for the training dataset. This indicates that our model is strong given that we can still achieve a high level of accuracy with a small amount of training data. However, we want to evaluate our model using something other than accuracy. Accuracy can be misleading, so it might not tell the whole story in terms of model quality. We need another metric with which to evaluate our model.

One metric that we can use to evaluate our model is an ROC curve. This curve indicates our model's true positive rate against it's false positive rate. The perfect model follows the left and top side of the graph, so we want our model to be as close to that as possible. We use this ROC curve to see how well our model makes predictions. This plot will give us a model evaluation that does not rely on our accuracy measure. We can graph the ROC curve for our model using the ROCR package.

```
library(ROCR)
set.seed(42)
roc.predict = predict(
  rf,
  type="prob",
  newdata = test[-1:-3]
  )
forestpred = prediction(
  roc.predict[,2],
  test$Status
  )
forestperf = performance(
  forestpred,
  "tpr",
  "fpr"
  )
plot(
  forestperf,
  main="ROC"
  )
abline(
  a=0,
  b= 1
  )
```



The ROC curve for our graph looks pretty good. It isn't perfect, but it is closer to the upper left corner than it is to the diagonal line. This would indicate that our model reliably makes correct predictions about PD. We would like to have some number that points to what our graph shows. For an ROC

curve, the number that indicates how well your model makes predictions is the Area Under the Curve (AUC). The greater the AUC value, the better your model is. We can calculate our AUC value using the same ROCR package.

```
auc.perf = performance(
  forestpred,
  measure = "auc"
  )
auc.perf@y.values
```

```
## [[1]]
## [1] 0.8828125
```

The perfect AUC value is 1, so we want to be as close to that value as possible. We can see from our calculation above that our AUC value is about 0.88 which is good. This is just a numeric representation of what our ROC curve is telling us. Our model isn't perfect, but it reasonably predicts PD in a patient with minimal errors. With this, we have evaluated our model on several fronts using both accuracy and an ROC curve. These are commonly used metrics to evaluate a model, and both indicate that our model provides reasonably correct predictions. We could spend years fine-tuning our model, but time limits the scale of this project. For now, we are accepting 85% accuracy and an 0.88 ROC AUC value as these indicate that our model is strong as it stands.

Conclusions and Future Work

With this work, we have shown that we can correctly predict Parkinsons Disease in a patient using data calculated from patient voice recordings. We first showed that the voice data was indeed related to Parkinson's Disease using correlation values and multiple hypothesis testing. After showing that the data was related, we then build a random forest model that predicted PD in a patient. We showed that the Accuracy of our model varied only slightly with different training/testing data splits. We then used metrics like Accuracy, Kappa Value, and an ROC curve to show that our model was reliably making the correct prediction of PD in a patient. With this, we have constructed a reasonable body of knowledge surrounding PD predictions that confirms what Naranjo proposed in her research paper. In terms of future work, we would like to be able to make these voice recording calculations using our own method. The actual voice recordings were not provided with this dataset, so we have no way of knowing how these recording metrics were calculated. It would be interesting to see if we can develop our own method of analyzing voice recording files. It would also be worth trying to improve the quality of our prediction model. Accuracy of 85% is good, but there are certainly methods that we can explore to improve on those results. For one, we could explore reducing the dimensionality of our dataset in an attempt to improve our results. Finally, the quality of this project would be greatly improved by having a larger dataset to study with. Having only 80 data entries really limits the results of any model we build. For this project to grow, it would be necessary to have additional patients give voice data that we could include in our model.

Acknowledgment

The authors would like to thank Dr. Lee, Dr.

References

[1] *What Is Parkinson's?* Parkinson's Foundation.

[2] J. Hardy, H. Cai, M. R. Cookson, K. Gwinn-Hardy, and A. Singleton, "Genetics of parkinson's disease and parkinsonism," *Annals of Neurology*, vol. 60, no. 4, pp. 389–398, 2006.

[3] L. Naranjo, C. J. Pérez, Y. Campos-Roca, and J. Martín, "Addressing voice recording replications for parkinson's disease detection," *Expert Systems with Applications*, vol. 46, pp. 286 – 292, 2016.

[4] L. Naranjo, C. J. Pérez, J. Martín, and Y. Campos-Roca, "A two-stage variable selection and classification approach for parkinson's disease detection by using voice recording replications," *Computer Methods and Programs in Biomedicine*, vol. 142, pp. 147 – 156, 2017.

[5] "Closing the gap for voice impairment in parkinson's," Mar 2020.

[6] U. M. L. Repository.

[7] M. Hasan, "The 20 best r machine learning packages in 2020," 2020.

[8] G. Pierobon, "A comprehensive machine learning workflow with multiple modelling using caret and caretensemble in...," Oct 2018.

[9] C. Maklin, "Random forest in r," Jul 2019.

[10] J. Brownlee, "How to evaluate machine learning algorithms with r," Dec 2019.

[11] J. Brownlee, "Better understand your data in r using visualization (10 recipes you can use today)," Aug 2019.

[12] A. Chakure, "Random forest and its implementation," Apr 2020.

[13] "14: Dimensionality reduction (pca)."

[14] "Pca analysis in r."

[15] C. Klein and A. Westenberger, "Genetics of parkinson's disease," *Cold Spring Harbor Perspectives in Medicine*, vol. 2, no. 1, 2012.

[16] N. C. C. for Chronic Conditions (UK), *Diagnosing Parkinson's disease*. U.S. National Library of Medicine, Jan 1970.

[17] S. Han and H. Kim, "On the optimal size of candidate feature set in random forest," *Applied Sciences*, vol. 9, p. 898, Mar 2019.

[18] R. Couronné, P. Probst, and A.-L. Boulesteix, "Random forest versus logistic regression: a large-scale benchmark experiment," *BMC Bioinformatics*, vol. 19, p. 270, Dec 2018.

[19] "What are the stages of parkinson's disease?."