Table of Contents

Abstract

This project explores the supervised learning method of SVM(Support Vector Machine) Classification in the context of Voice Recognition. Voice Recognition refers to the ability to correctly identify the voice of an individual speaker from among many speakers' voice recordings and humans are generally adept at this. However, there are situations where voices need to be identified without the inherent biases that come from a human listener, for example in court cases when determining whether a potentially incriminating audio sample does or does not include the accused's voice. Currently there is a critical need for more accurate and objective voice recognition software and SVM Classing is one of the methods being investigated and used.

The project first makes use of the publicly available dataset voice.csv on kaggle.com as a basis for testing learning methods on spectrogram data of human voices to classify those voices by biological Sex. The project seeks to emulate the original experiment and results obtained by experimenter Kory Becker. After verifying the usefulness of those measures in predicting Sex using familiar Random Forest Classification, we test the unfamiliar SVM method on that data and achieve similar, even slightly improved results. Using her dataset as a template, we expand the experiment by bringing in new anonymous audio data from VoxForge and parsing its spectrogram measurements. The dataset is constructed this time with the intention of Voice Recognizing each case. Random Forest Classification and SVM Classification is performed on this new dataset and each attempts Voice Recognition.

The ultimate result of the project is confirmation that, while biological Sex is easily identified in the human voice using these methods, Voice Recognition is a thornier issue, and that although SVM performs better than Random Forest in the former experiment, it winds up being inferior in the latter, although reasons as to why that might be the case are, at this point, speculative.

Introduction to SVM

SVM Classification is a method of supervised learning that separates data into groups much like k-nearest neighbor. The training data is plotted in p-dimensional space where p is the number of variables and the algorithm then seeks a hyperplane (plane with dimension p-1) that divides the data points correctly into their classes while maintaining the largest margin of distance between itself and any of the points. This ensures a "maximum margin classifier", or a classifier that gives us the greatest chance of correctly classifying new test data based on our training data.

In essence, the points are plotted and the algorithm attempts to define a border halfway between the two groups, then new data is classified based on which side of the border it lands on (Figure 1).

The "Support Vector" part of the title refers to those observations that lie closest to the boundaries of the two groups, the points that have the most influence on the shape of the plane. If we expanded the training set by adding another point right on the line, that point would become a strong determining support vector of our line and would shift or even reshape the line. Points that fall well within a grouping do not influence the line as greatly and so are of less interest. The "Machine" part refers to machine learning.

For classification where the response variable is not binary, as in the case for Voice Classification among more than two individuals, several classifiers are trained in succession using a "one vs rest" approach such that the first classifier would be "Person A vs Not Person A", the second being "Person B vs Not Person B", etc. Think of it something along the same logical lines as nested If-Else statements.

For more complex data where the classifications are not always linearly seperable in p-dimensional space (Figure 2), the algorithm maps the whole set to a higher dimensional space and seeks a dividing hyperplane in that space (Figure 3). The hyperplane is then projected downward to the original dimensions in order to correctly class the data (Figure 4). This is known as the kernel trick.

`https://medium.com/machine-learning-101/chapter-2-svm-support-`

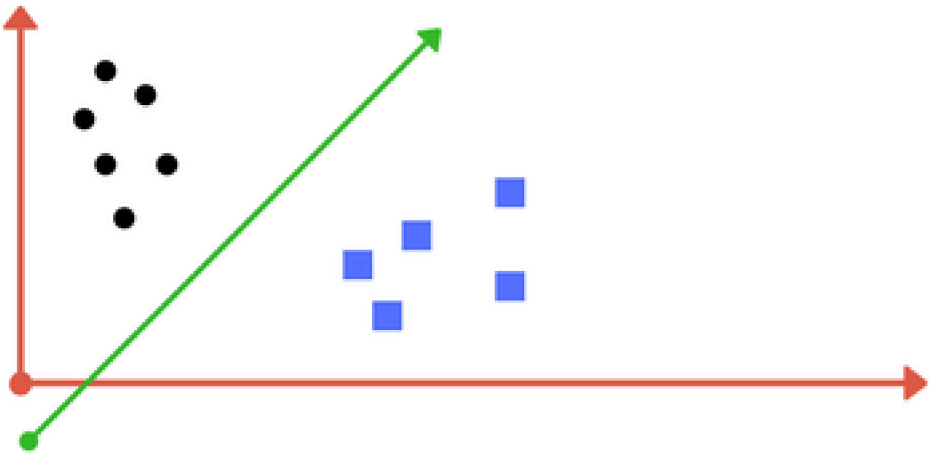

Figure 1

vector-machine-theory-f0812effc72 https://en.wikipedia.org/wiki/Support-vector_machine#History https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning https://www.hackerearth.com/blog/machine-learning/simple-tutorial-svm-parameter-tuning-python-r/

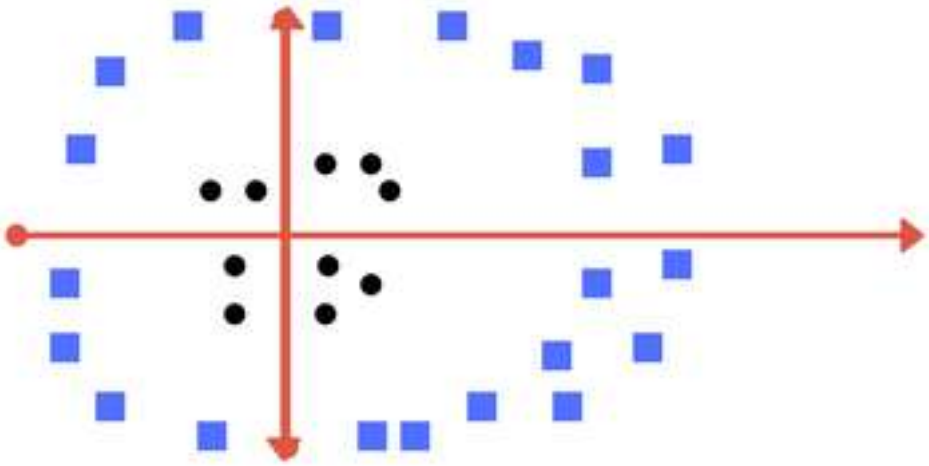

Figure 2

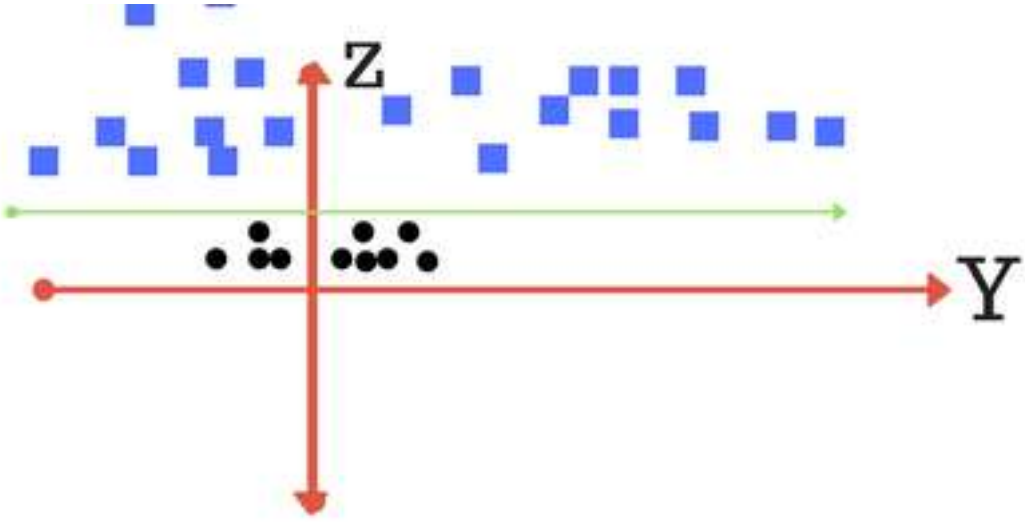

Figure 3

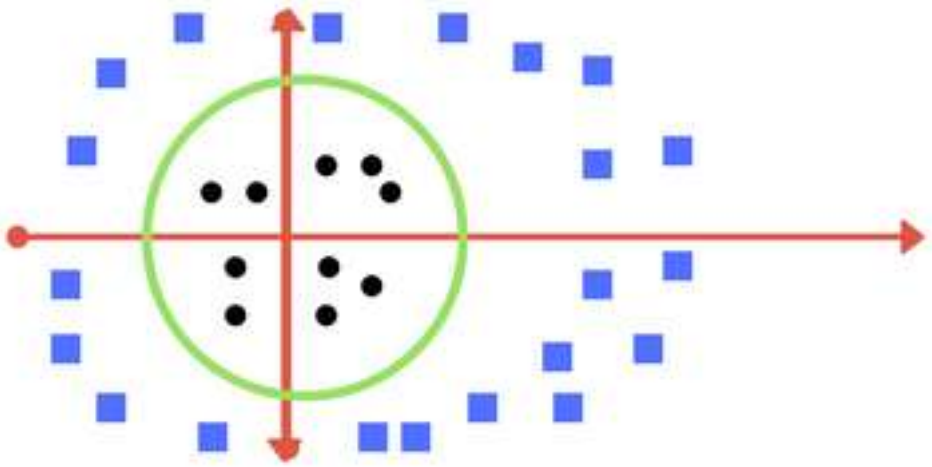

Figure 4

Introduction to Spectrograms

A spectrogram plots the frequency or amplitude of a sound wave over time, often with other information added in as well. Seewave's default spectrograms, shown here, represent the frequency and amplitude information over two graphs, but a full heat-map style of spectrogram might show frequency, time, amplitude, and intensity. If you have never seen a spectrogram before or would like a demonstration to experiment with, there's a free program in Google Chrome's Music Lab at `https://musiclab.chromeexperiments.com/Spectrogram/` . The concept makes a lot more sense if you see it in action.

Essentially, a spectrogram is a visual representation of a sound. Frequency is a measure of the pitch of a sound via the rapidity of its oscillations (in kHz) and amplitude is the volume or loudness of a sound (relative to a mean). These measures alone can be very useful in identifying sounds and even analyzing speech, since different phonemes (most basic unit sounds of a language) have different wave patterns when plotted. It is not a perfect representation of a sound, however. Tambre, the ill-defined notion of the "feel" of a sound or "that which differentiates two sounds of the same pitch and volume" (think how you'd recognize the same note played on a flute versus a violin) is less visible on these sorts of spectrograms, although it comes across moreso on heat-map style spectrograms like those in the Google demonstration, and it's not easy for people to reproduce a sound while looking at that sound's spectrogram. They do provide useful information for computerized analytics.

The measures we mined from the spectrogram data are listed and explained here.

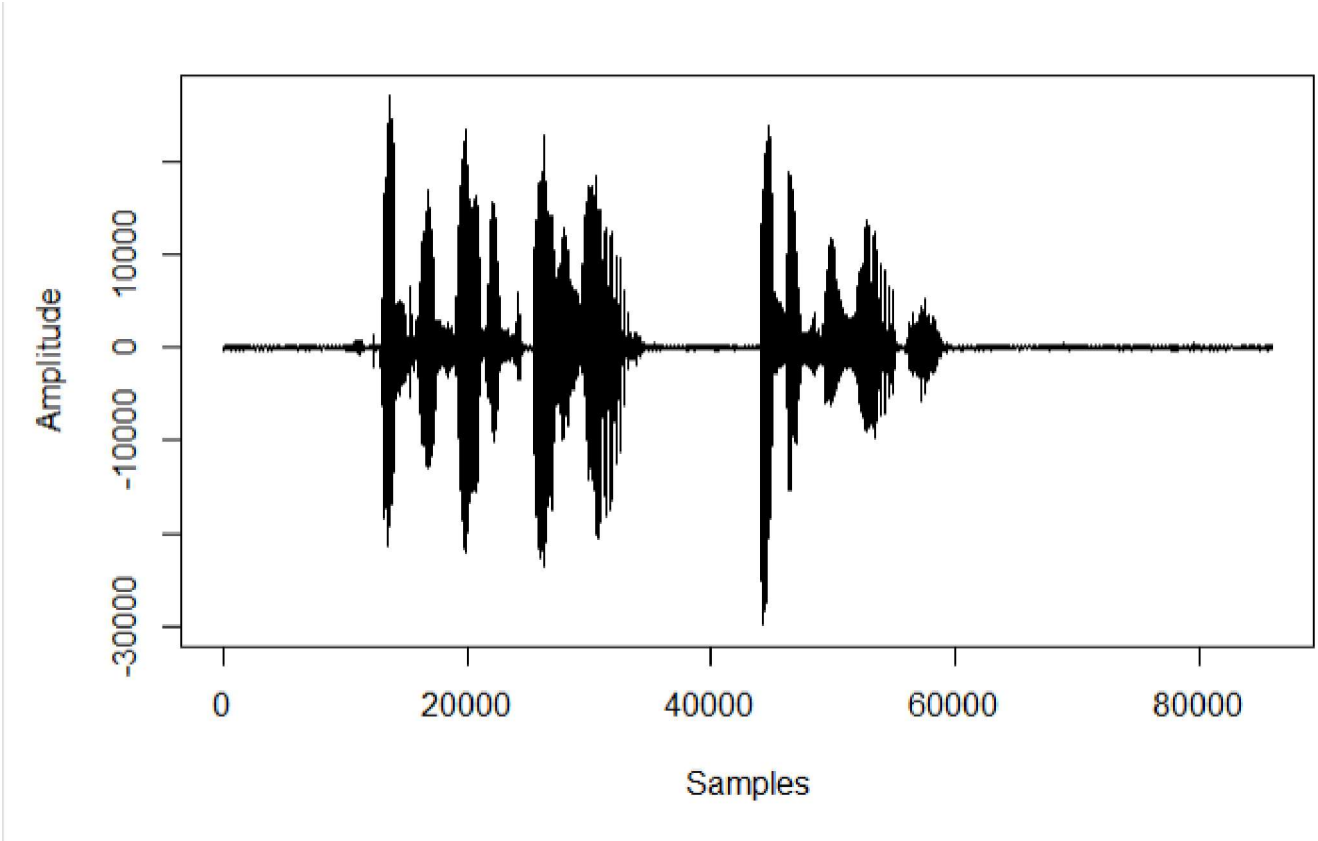
Amplitude
-30000
-10000
0
10000
0
20000
40000
60000
80000
Samples

Figure 5

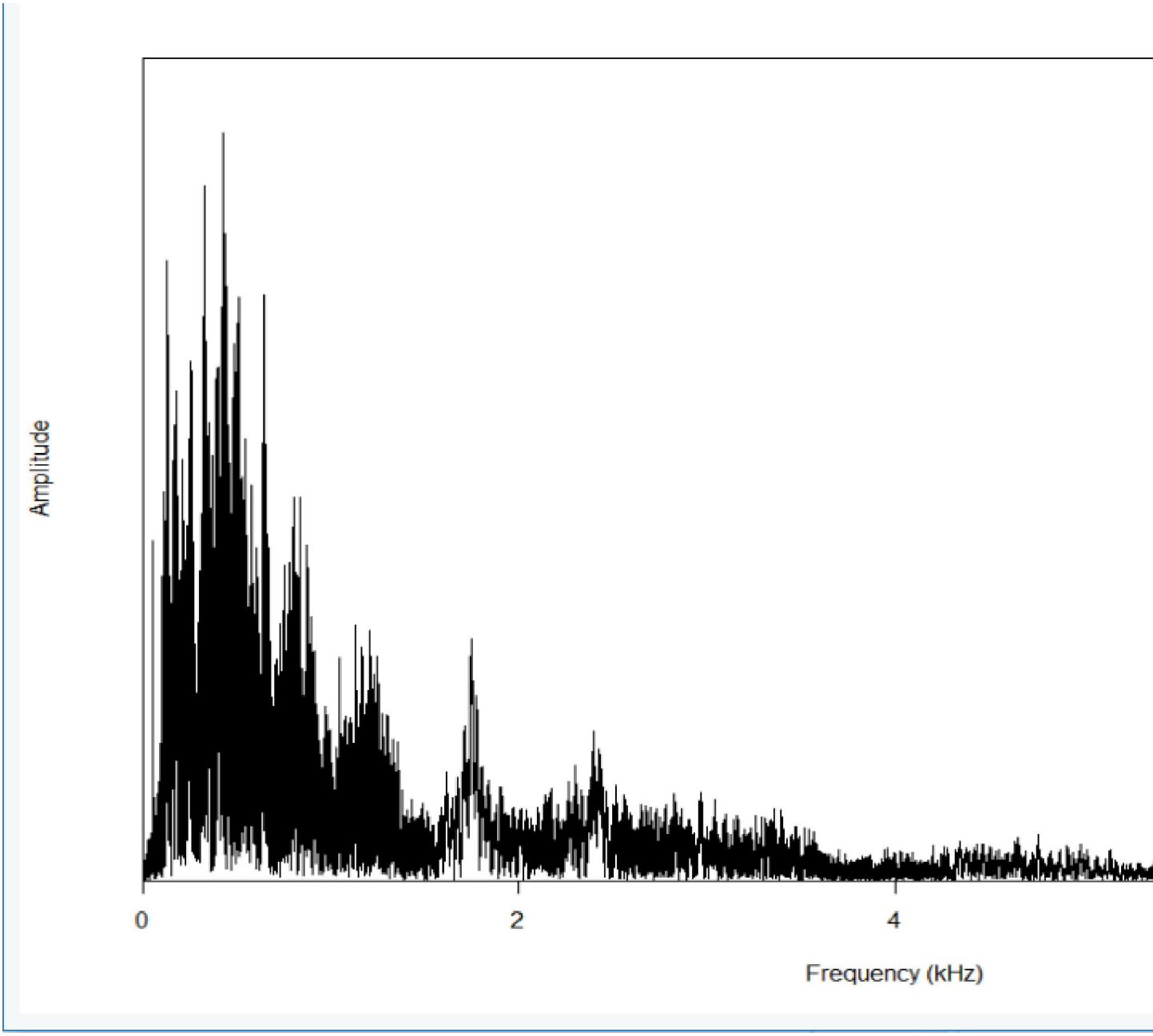

Figure 6

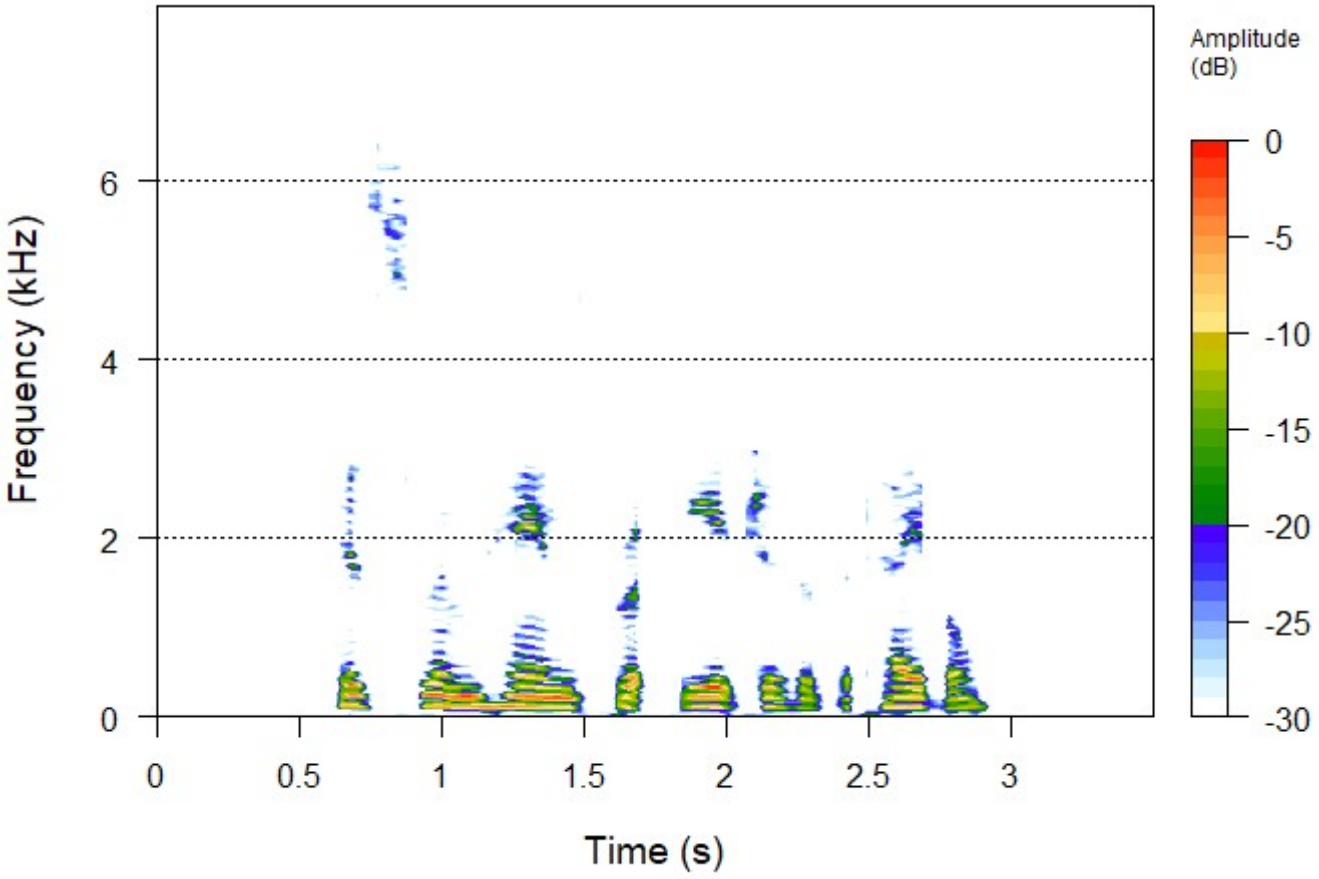
Frequency (kHz)
6
4
2
0
0
0.5
1
1.5
2
2.5
3
Time (s)
Amplitude
(dB)
0
-5
-10
-15
-20
-25
-30

Figure 7

meanfreq: mean frequency of the wave(in kHz)

sd: standard deviation of frequency of the wave

median: median frequency (in kHz)

Q25: first quantile for frequency (in kHz)

Q75: third quantile for frequency (in kHz)

IQR: interquantile range for frequency (in kHz)

skew: skewness of the spectrogram's frequency, computed as $S = \sum_{i=1}^{N}(freq_i - meanfreq)^3 \times \frac{1}{sd^3}$. S<0 indicates left skew, S>0 indicates right skew, S=0 indicates perfect symmetry

kurt: kurtosis of the spectrogram's frequency, computed according to $K = \sum_{i=1}^{N}(freq_i - meanfreq)^4 \times \frac{1}{sd^4}$, measures the spectrogram relative to the normal curve. K<3 indicates fewer items at center and tails than expected from normal but more in the shoulders, K>3 indicates more items at the center and tails than expected but fewer at the shoulders, and K=3 indicates a perfect normal curve.

sp.ent: spectral entropy. Describes the complexity of a sound wave, ie how much information is being conveyed. A pure synthetic tone has low entropy, a recording from a crowded diner has high entropy. Roughly speaking it indicates how "noise-like" a sound is compared to how "tonelike". Ranges between 0 and 1.

sfm: spectral flatness. White noise produces a spectrogram that looks nearly flat, with only minor rising and falling around its central tone. This measure indicates how steady and near to white noise a spectrogram is. Sfm closer to 1.0 indicates a very monotonic sound and human voices typically score much closer to 0.0.

mode: mode frequency

centroid: frequency centroid. Computed as $C = \sum_{i=1}^{N}(freq_i - meanfreq)^2 \times \frac{1}{sd^2}$

peakf: peak frequency (frequency with highest energy)

meanfun: average of fundamental frequency measured across acoustic signal. Fundamental frequency is the lowest frequency produced by oscillation of the object. In terms of hearing, it is the lowest pitch or tone that you hear with harmonics rising above it where the the frequency of the sound waves exactly double that frequency. The mean fundamental frequency thus gives an approximation of a person's most comfortable "natural" pitch, though it can be forced higher or lower by modulation of the voice.

minfun: minimum fundamental frequency measured across acoustic signal

maxfun: maximum fundamental frequency measured across acoustic signal

meandom: average of dominant frequency measured across acoustic signal. Dominant frequencies are local maximums of the frequencies, the apexes in the wave.

mindom: minimum of dominant frequency measured across acoustic signal

maxdom: maximum of dominant frequency measured across acoustic signal

dfrange: range of dominant frequency measured across acoustic signal

modindx: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range.

Chapter 1: Spec Measures and Biological Sex

The following is an example of a classification forest model built upon measurements from spectrogram data. The work is done on the dataset voice.csv compiled by Kory Becker and available on Kaggle. This dataset was created using functions from the libraries seewave and tuneR in R's Cran repository to create and analyze spectrograms from a set of voice recordings. Each entry (row) in the data is a voice recording and each variable (column) is a measurement taken off of the spectrogram (excepting the variable "label" which is the biological sex of the entry, reported directly from Becker's data). This dataset was compiled with the express intention of predicting biological sex. My intent is to construct a model that groups entries by the individual who speaks them, however this seemed a fair way to test my procedures and this sort of measurement data. It is a simpler problem than Voice Recognition, previous work has shown biological sex to be one of the foremost predictable features of a voice. Emulating this experiment lets us test our understanding of the new SVM method against known data and a known method of classification, Random Forest.

We began by splitting the data into training and testing sets.

```
voice = read.csv("/home/class19/gyoung19/StatComps/voice.csv")
```

```
set.seed(36) #for reproducability
train <- voice %>%
  sample_frac(0.80) %>%
  mutate(label = as.factor(label))

test <- voice %>%
  setdiff(train) %>%
  mutate(label = as.factor(label))
```

```
x_train <- model.matrix(label ~ ., train)[, -1]
x_test <- model.matrix(label ~ ., test)[, -1]

y_train <- train %>%
  dplyr::select(label)

y_test <- test %>%
  dplyr::select(label)
```

We create a random forest on the training set, using default params of 500 trees considering 4 variables at each step with no maximum tree length for the forest and the default options for the svm.

```
set.seed(2250) #reproducability
rf <- randomForest(label ~ . , data = train, importance = TRUE) #produce random forest
svm <- svm(label ~ ., data = train)
```

```
estimate1 <- predict(rf, newdata = test) #assess accuracy of model on "new" data
misclass1 <- ifelse(estimate1 != test$label,1,0) #create vector indicating where miscla
mean(misclass1) #get percentage of misclassifications on test set
```

```
[1] 0.009463722
```

Our random forest model seems to be able to differentiate between the biological sex of the speakers with a high degree of accuracy, misclassifying only 0.94% of the time. We would also like to check which of these many variables were most informative to our model.

```
importance(rf) #list variables and importance
```

```
            female      male MeanDecreaseAccuracy MeanDecreaseGini
meanfreq 10.223607 11.481962             14.99099        26.770622
sd       16.851184 15.945382             21.81419        97.635865
median    9.696393 14.214570             16.44871        18.682786
Q25      18.925172 23.620311             28.38364       159.398463
Q75      10.617579 12.139675             15.66817        13.699193
```

```
IQR      20.394572 35.714711          38.58051       244.137512
skew     10.372807  8.376263          12.99566        13.602288
kurt     10.526120  8.171386          12.32354        10.229845
sp.ent   14.865760  9.826768          17.64559        57.016189
sfm      16.982912 12.861330          20.18108        41.084571
mode     10.639303 11.233951          13.37967        19.905143
centroid  9.520364 12.591789          16.01013        22.352289
meanfun  42.814560 65.873813          71.07732       473.254813
minfun   11.732828 12.073388          16.17091        12.338545
maxfun    9.386261  7.286574          11.16917         6.341982
meandom  12.123758  8.913113          13.98629         9.560521
mindom    5.866225 10.972492          11.43918         8.878321
maxdom   13.384097  9.478714          15.51142        12.034714
dfrange  15.439257  9.604017          17.23617        11.119503
modindx  14.207147  7.524171          14.92663         8.412018
```

It seems that the mean fundamental frequency leads by quite a large margin and probably merits further consideration in my continuing work. This is not at all unexpected, fundamental frequency has been useful in differentiating biological sex of a speaker before. Reports on the actual numbers vary, but generally state that the average range of male fundamental frequency is somewhere between 85 to 180 Hz and the average female range is between 165 and 255 Hz.

Now let's see how the svm performed.

```
estimate2 <- predict(svm,newdata = test)
misclass2 <- ifelse(estimate2 != test$label,1,0)
mean(misclass2)
```

```
[1] 0.006309148
```

It performed even better than the Random Forests, misclassifying a scant 0.63% of the time. Let's examine the weights to get an idea of which variables were most important.

```
#code from https://www.researchgate.net/post/How_can_I_find_the_w_coefficients_o
W <- t(svm$coefs) %*% svm$SV
W
```

```
     meanfreq        sd   median      Q25       Q75       IQR      skew
[1,]  5.15502 -2.862442 7.208953 13.2442 -1.074973 -15.65304 1.295321
          kurt    sp.ent      sfm     mode centroid  meanfun   minfun
[1,] 0.7507584 -5.611745 10.72697 10.22149  5.15502 74.00405 20.95893
       maxfun  meandom    mindom    maxdom   dfrange   modindx
[1,] 14.30399 4.155208 -3.703357 -7.158458 -7.095421 -3.072451
```

Again, this seems to verify that meanfun leads by a wide margin and that the IQR of the frequency is also of great influence. This example has served to confirm the legitimacy of these measures and our chosen methods.

ScrapingVoxForge

Before SVM could be performed, we needed data that was suitable for Voice Recognition. This was the process by which I compiled my own data set after the fashion of Becker's.

I downloaded data from links off of the VoxForge Home site to local files. The code here makes spectrograms for each of the waves in the files and then pulls our desired spectrogram stats from those sepctrograms.

http://samcarcagno.altervista.org/blog/basic-sound-processing-r/?doing_wp_cron=1548891484.3526279926300048828125

https://marce10.github.io/2017/02/17/Choosing_the_right_method_for_measuring_acoustic_signal_structure.html

```
path <- 'C:/Users/Gabriel Young/Documents/StatComps/VoxForge/anonymous-20110310-it
spectro1 <- readWave(path)
```

```
#Code for pulling stats from a single wav file

#isolate the signal
signal <- spectro1@left

#obtain signal duration
dur = length(signal)/spectro1@samp.rate

#get the sampling rate of the signal
fs = spectro1@samp.rate

# center to remove DC offset
signal = signal - mean(signal)
```

```
#obtain the spec object
s <- spec(spectro1,fs)

#get frequency properties we want using specprop
props <- as.data.frame(specprop(s,fs))

#get fundamental frequencies, eliminate NaNs
f <- as.data.frame(fund(spectro1,fs))
f2 <- f[complete.cases(f),]

#get summary stats of fundamental frequency measures
meanfun <- mean(f2$y)
minfun <- min(f2$y)
maxfun <- max(f2$y)
props <- cbind(props,meanfun, minfun, maxfun)

#get dominant frequencies, eliminate NaNs
d <- as.data.frame(dfreq(spectro1,fs))
d2 <- d[complete.cases(d),]

#get summary stats of dominant frequencies
meandom <- mean(d2$y)
mindom <- min(d2$y)
maxdom <- max(d2$y)

#add to data
props <- cbind(props,meandom,mindom,maxdom)
```

Having developed code for mining a single observation, we then mined the rest of our data.

```
path <- 'C:/Users/Gabriel Young/Documents/StatComps/VoxForge/'
folders <- as.data.frame(list.files(path))
folders <- folders[4:16,]
folders <- as.vector(folders)
```

```
#initialize storage dataframe
desiredstats <- data.frame()

#apply spec processes to all files to obtain descriptive statistics that we want
for(folder in folders)
{
  files <- as.data.frame(list.files(paste(path, folder, "/wav", sep = "")))
  files <- files %>% mutate(names = as.character(`list.files(paste(path, folder, '
  realfiles = files$names
  for(f in realfiles)
  {
    spectro1 <- readWave(filename = paste(path,folder,"/wav/",f,sep=""))
    signal <- spectro1@left

    dur = length(signal)/spectro1@samp.rate

    fs = spectro1@samp.rate

    # demean to remove DC offset
    signal = signal - mean(signal)

    s <- spec(spectro1,fs)
    props <- as.data.frame(specprop(s,fs))
    props <- props %>% mutate(Files = f, Folders = folder)

    f <- as.data.frame(fund(spectro1,fs))
    f2 <- f[complete.cases(f),]
    meanfun <- mean(f2$y)
    minfun <- min(f2$y)
    maxfun <- max(f2$y)

    d <- as.data.frame(dfreq(spectro1,fs))
    d2 <- d[complete.cases(d),]
    meandom <- mean(d2$y)
    mindom <- min(d2$y)
```

```
    maxdom <- max(d2$y)

    props <- cbind(props,meanfun, minfun, maxfun)
    props <- cbind(props,meandom,mindom,maxdom)

    desiredstats <- rbind(desiredstats, props)
  }
}
```

```
write.csv(desiredstats, file = "VoiceRecogData.csv")
```

This furnished us with a 130 observation dataset with labels of the person speaking for each observation, catering to the Voice Recognition experiment.

Voice Recognition

Having created a suitable dataset for ourselves, we again run the same procedure as with the preliminary analysis. In this data, our response variable is Speaker. We start by splitting the data into training and testing sets.

```
voice <- read.csv("/home/class19/gyoung19/StatComps/GYoung/VoiceRecogData.csv")
voice <- voice %>% mutate(Speaker = Folders) %>% dplyr::select(-X,-Folders,-Files
```

```
set.seed(36) #for reproducability
train <- voice %>%
  sample_frac(0.80) %>%
  mutate(Speaker = as.factor(Speaker))

test <- voice %>%
  setdiff(train) %>%
  mutate(Speaker = as.factor(Speaker))

x_train <- model.matrix(Speaker ~ ., train)[, -1]
x_test <- model.matrix(Speaker ~ ., test)[, -1]

y_train <- train %>%
  dplyr::select(Speaker)

y_test <- test %>%
  dplyr::select(Speaker)
```

We create a random forest on the training set, using default params of 500 trees considering 4 variables at each step with no maximum tree length for the forest and the default options for the svm.

```
set.seed(2250) #reproducability
rf <- randomForest(Speaker ~ . , data = train, importance = TRUE) #produce random fores
svm <- svm(Speaker ~ ., data = train)
```

```
estimate1 <- predict(rf, newdata = test) #assess accuracy of model on "new" data
misclass1 <- ifelse(estimate1 != test$Speaker,1,0) #create vector indicating where misc
mean(misclass1) #get percentage of misclassifications on test set
```

```
[1] 0.4615385
```

We can see here that we are doing much worse than with the previous data set aimed at biological sex, our model is only classifying 64% of the data correctly. Still that is an improvement over random guesswork, which would have a classification rate of around 8%. Let's check again and see which of these many variables were most informative to our model.

```
importance(rf) #list variables and importance
```

With so many response categories, it has become much more difficult to discern the importance of of any single variable.

Let's see how the svm performed.

```
estimate2 <- predict(svm,newdata = test)
misclass2 <- ifelse(estimate2 != test$Speaker,1,0)
mean(misclass2)
```

Contrary to our original test, it actually performed worse here than the Random Forests, misclassifying a full half of the time. When we look at the weights table

```
#code from https://www.researchgate.net/post/How_can_I_find_the_w_coefficients_of_SVM
W <- t(svm$coefs) %*% svm$SV
W
```

```
           mean          sd      median         sem       mode        Q25
[1,] -3.7273657 -0.03504357 -5.58251486 -2.7585724  8.0089323  2.1029127
[2,]  4.9149467  5.97236428  1.57934857 -1.7984637 16.2977456  9.0568985
[3,]  2.4430041  3.15959784  2.18914618  3.9075201  9.3424773  6.0155046
```

```
 [4,] -4.3363175  0.70669410 -7.27112686  0.4687597  4.1212862 -1.2130570
 [5,]  2.9657714  4.54441037 -0.10707175  7.4846005 -0.8047437  3.6522111
 [6,] -4.6442970  0.06849099 -8.00870090 -4.7513667  0.3669068  0.7318693
 [7,]  1.5837223  3.93134435  0.09710231 -0.2548240  2.3967546  4.3386253
 [8,] -1.0258777 -1.71589904  0.03869577  2.0004297  3.1859179  1.5586233
 [9,] -0.8900622 -6.79781252  3.03238813 -3.9605366  1.3089928  3.8510332
[10,]  2.9620493 -6.32363338  7.07651874  3.2701599 -4.9970677  9.8997442
[11,] 10.0425287  3.92044436  9.15200402  7.4967070 -4.7372618 12.5528749
[12,] -1.8759387 -3.31372558 -2.77781483  1.6148697 -8.5517959  1.2833512
            Q75        IQR       cent   skewness   kurtosis       sfm
 [1,] -5.6690218 -6.4968414 -3.7273657  1.2677866 -1.1073605  1.601158
 [2,]  1.8989262  0.3017076  4.9149467 -0.1524015 -0.1428662 10.098620
 [3,] -1.9373725 -3.2374552  2.4430041  2.0406163  0.3960797  9.133933
 [4,] -5.7790335 -5.9782995 -4.3363175  2.8122456 -2.2144499  2.820106
 [5,]  2.8846962  2.3990916  2.9657714  0.8386053 -3.3524253  8.806495
 [6,] -5.6218079 -6.1828084 -4.6442970 10.3796360  5.3147227  1.460055
 [7,] -1.7271329 -2.6894726  1.5837223  6.8664843  3.5752615  3.458104
 [8,] -1.5472363 -1.9622643 -1.0258777  6.8632694  2.0727582 -1.670629
 [9,] -0.8455694 -1.6483421 -0.8900622  0.9900556 -0.3946107 -7.983400
[10,]  1.9933572  0.2413454  2.9620493 -0.3296966  0.7555857 -7.651431
[11,]  7.9197072  6.1014349 10.0425287 -4.5676899 -1.5692786  0.340268
[12,] -0.5544463 -0.8423616 -1.8759387 -3.0333163 -1.9858746 -8.692907
             sh         prec     meanfun     minfun    meandom     mindom
 [1,] -3.4254504 -5.589857209   3.2526870  -1.171805  0.1486528 13.3012140
 [2,]  4.3300590 -9.598518839   9.9010218   4.132047 10.5008830 17.4033954
 [3,]  1.9281122  1.456739908   4.7199488  -4.061788  0.4078934 14.9565552
 [4,] -6.1158154  0.001394542   6.5301834  -2.431810 -2.8417556 12.0046692
 [5,]  2.7445453  6.730958610  -0.4557016 -12.617306 -4.3497170  8.1417027
 [6,] -4.1950520 -7.159249282   4.8806369 -13.325389 -5.2246208  8.2967867
 [7,]  0.4207524 -4.807737997 -11.5276607  -5.673415  0.7048216  6.4339940
 [8,] -5.6058899  4.690492208  -9.0788104  -2.196705 -4.6599762  2.1922283
 [9,] -4.5281344  1.964586231  -7.7697213  -9.807704  0.8704886  0.3802329
[10,] -0.7237214 12.558625913  -6.5326919  -5.319211 -2.4049458 -1.5195758
[11,]  8.4233051  7.222046510  -6.6567882 -10.805694  6.2766950 -1.9826413
[12,] -3.1345126  7.112813949  -2.5208326  -8.678459 -5.1222461 -4.7456389
          maxdom
```

```
 [1,]  2.3137295
 [2,] 10.6591428
 [3,] -3.0028746
 [4,] -3.3971310
 [5,] -4.9415422
 [6,] -2.9269086
 [7,] -0.6771455
 [8,] -6.1297493
 [9,] -2.3974406
[10,] -7.6074678
[11,]  4.3375244
[12,] -4.4410566
```

We can see that it doesn't tell us a whole lot more than the importance table did for random forests.

Is there a possibility that this data is overfit or that we have too much confusing information? Let's reduce the dataset to the components that were most informative in our previous build.

```
train2 <- train %>% dplyr::select(IQR,Speaker,meanfun,meandom)
test2 <- test %>% dplyr::select(IQR,Speaker,meanfun,meandom)
```

```
set.seed(2250) #reproducability
rf <- randomForest(Speaker ~ . , data = train2, importance = TRUE) #produce random fore
svm <- svm(Speaker ~ ., data = train2)
```

```
estimate1 <- predict(rf, newdata = test2) #assess accuracy of model on "new" data
misclass1 <- ifelse(estimate1 != test2$Speaker,1,0) #create vector indicating where mis
mean(misclass1) #get percentage of misclassifications on test set
```

```
[1] 0.7692308
```

This seems to be a step in the wrong direction for the Random Forests. How about the svm?

```
estimate2 <- predict(svm,newdata = test2)
misclass2 <- ifelse(estimate2 != test2$Speaker,1,0)
mean(misclass2)
```

```
[1] 0.6538462
```

While the svm is now performing better than the Random Forest, neither can be said to be performing particularly well, certainly not at the level of the previous experiment. There are many possible explanations, but a likely one is that there are too many classes. Let's see how things go in a "one vs rest" scenario, since svm is supposed to operate best under the condition of a binary response variable anyway.

```
train3 <- train %>% mutate(Speaker = ifelse(Speaker == "anonymous-20110312-wbu","a
test3 <- test %>% mutate(Speaker = ifelse(Speaker == "anonymous-20110312-wbu","ano
```

```
set.seed(2250) #reproducability
rf <- randomForest(as.factor(Speaker) ~ . , data = train3, importance = TRUE) #pr
svm <- svm(as.factor(Speaker) ~ ., data = train3)
```

```
estimate1 <- predict(rf, newdata = test3) #assess accuracy of model on "new" data
misclass1 <- ifelse(estimate1 != test3$Speaker,1,0) #create vector indicating whe
mean(misclass1) #get percentage of misclassifications on test set
```

```
[1] 0.03846154
```

The misclass rate has dropped dramatically for the Random Forest. How does the svm fare?

```
estimate2 <- predict(svm,newdata = test3)
misclass2 <- ifelse(estimate2 != test3$Speaker,1,0)
mean(misclass2)
```

```
[1] 0.07692308
```

The misclass rate is twice as high as the one for Random Forest. However, the fact that it is exactly twice as high clues us into a very simple error that has occurred.

```
tally(test3$Speaker)
```

```
X
    anonymous-20110312-wbu Not anonymous-20110312-wbu
                         2                         24
```

```
2/26
```

```
[1] 0.07692308
```

Under these conditions, with so few points in the test and train set belonging to our target group and so many falling into "other", both programs are learning primarily to just guess the "other" group. This illustrates the basic fact that it becomes increasingly difficult to isolate a single voice out of those you know for every new voice that you have to compare it to.

PreLim Code

```
library(mosaic)
library(dplyr)
library(randomForest)
library(glmnet)
library(e1071)
```

```
voice = read.csv("/home/class19/gyoung19/StatComps/voice.csv")
```

```
set.seed(36) #for reproducability
train <- voice %>%
  sample_frac(0.80) %>%
  mutate(label = as.factor(label))

test <- voice %>%
  setdiff(train) %>%
  mutate(label = as.factor(label))

x_train <- model.matrix(label ~ ., train)[, -1]
x_test <- model.matrix(label ~ ., test)[, -1]

y_train <- train %>%
  dplyr::select(label)

y_test <- test %>%
  dplyr::select(label)
```

```r
set.seed(2250) #reproducability
rf <- randomForest(label ~ . , data = train, importance = TRUE) #produce random forest
svm <- svm(label ~ ., data = train)

estimate1 <- predict(rf, newdata = test) #assess accuracy of model on "new" data
misclass1 <- ifelse(estimate1 != test$label,1,0) #create vector indicating where miscla
mean(misclass1) #get percentage of misclassifications on test set

importance(rf) #list variables and importance

estimate2 <- predict(svm,newdata = test)
misclass2 <- ifelse(estimate2 != test$label,1,0)
mean(misclass2)

#code from https://www.researchgate.net/post/How_can_I_find_the_w_coefficients_of_SVM
W <- t(svm$coefs) %*% svm$SV
W
```

ScrapingVoxForge Code

```
knitr::opts_chunk$set(echo = TRUE)
#install.packages("e1071")
#install.packages("caret")
library(mosaic)
library(e1071)
library(seewave)
```

```
path <- 'C:/Users/Gabriel Young/Documents/StatComps/VoxForge/anonymous-20110310-it
spectro1 <- readWave(path)
```

```
#Code for pulling stats from a single wav file

#isolate the signal
signal <- spectro1@left

#obtain signal duration
dur = length(signal)/spectro1@samp.rate

#get the sampling rate of the signal
fs = spectro1@samp.rate

# center to remove DC offset
signal = signal - mean(signal)

#obtain the spec object
s <- spec(spectro1,fs)
```

```
#get frequency properties we want using specprop
props <- as.data.frame(specprop(s,fs))

#get fundamental frequencies, eliminate NaNs
f <- as.data.frame(fund(spectro1,fs))
f2 <- f[complete.cases(f),]

#get summary stats of fundamental frequency measures
meanfun <- mean(f2$y)
minfun <- min(f2$y)
maxfun <- max(f2$y)
props <- cbind(props,meanfun, minfun, maxfun)

#get dominant frequencies, eliminate NaNs
d <- as.data.frame(dfreq(spectro1,fs))
d2 <- d[complete.cases(d),]

#get summary stats of dominant frequencies
meandom <- mean(d2$y)
mindom <- min(d2$y)
maxdom <- max(d2$y)

#add to data
props <- cbind(props,meandom,mindom,maxdom)
```

```
path <- 'C:/Users/Gabriel Young/Documents/StatComps/VoxForge/'
folders <- as.data.frame(list.files(path))
folders <- folders[4:16,]
folders <- as.vector(folders)



#initialize storage dataframe
desiredstats <- data.frame()

#apply spec processes to all files to obtain descriptive statistics that we want
for(folder in folders)
```

```
{
  files <- as.data.frame(list.files(paste(path, folder, "/wav", sep = "")))
  files <- files %>% mutate(names = as.character(`list.files(paste(path, folder,
  realfiles = files$names
  for(f in realfiles)
  {
    spectro1 <- readWave(filename = paste(path,folder,"/wav/",f,sep=""))
    signal <- spectro1@left

    dur = length(signal)/spectro1@samp.rate

    fs = spectro1@samp.rate

    # demean to remove DC offset
    signal = signal - mean(signal)

    s <- spec(spectro1,fs)
    props <- as.data.frame(specprop(s,fs))
    props <- props %>% mutate(Files = f, Folders = folder)

    f <- as.data.frame(fund(spectro1,fs))
    f2 <- f[complete.cases(f),]
    meanfun <- mean(f2$y)
    minfun <- min(f2$y)
    maxfun <- max(f2$y)

    d <- as.data.frame(dfreq(spectro1,fs))
    d2 <- d[complete.cases(d),]
    meandom <- mean(d2$y)
    mindom <- min(d2$y)
    maxdom <- max(d2$y)

    props <- cbind(props,meanfun, minfun, maxfun)
    props <- cbind(props,meandom,mindom,maxdom)

    desiredstats <- rbind(desiredstats, props)
```

```
  }
}
```

```
write.csv(desiredstats, file = "VoiceRecogData.csv")
```

Voice Recognition Code

```
library(mosaic)
library(dplyr)
library(randomForest)
library(glmnet)
library(e1071)
```

```
# from https://www.r-bloggers.com/machine-learning-using-support-vector-machines
mod_svm <- svm(as.factor(Folders)~ ., desiredstats2)
```