Van-AI-ML-Project02

February 27, 2021

1 Parkinson's Screening By Voiceprint Analysis

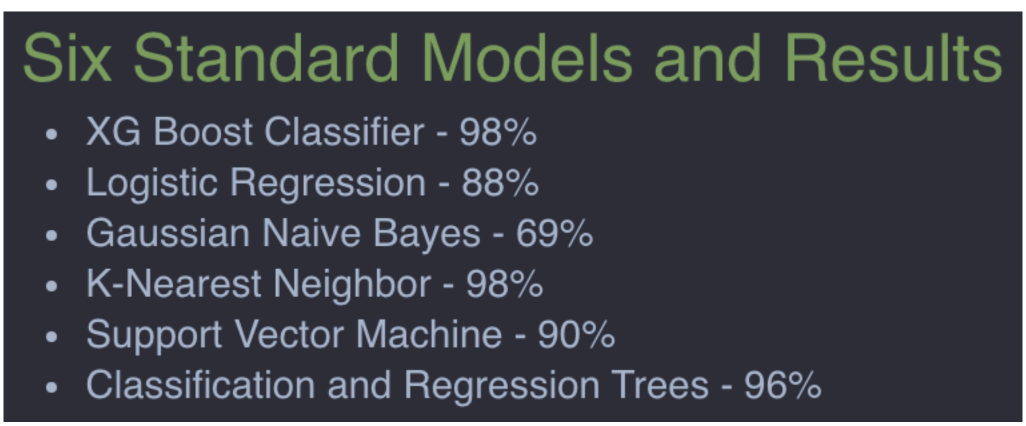
AI-5383
L. Van Warren
Assigned: Tuesday, February 23, 2021
Prof. Mariofanna Milanova
TA. Imran 'Md' Sarker
University of Arkansas, Little Rock

1.1 Introduction - Collection and Analysis of a Parkinson Dataset

This is a review of the paper whose subject is the title, and may be clicked to reference it. Unlike typical reviews this one is a living document, a Python Jupyter notebook, that allows its results and follow-on work to be executed by the reader. It can also be used as a reference template for similar machine learning investigations.

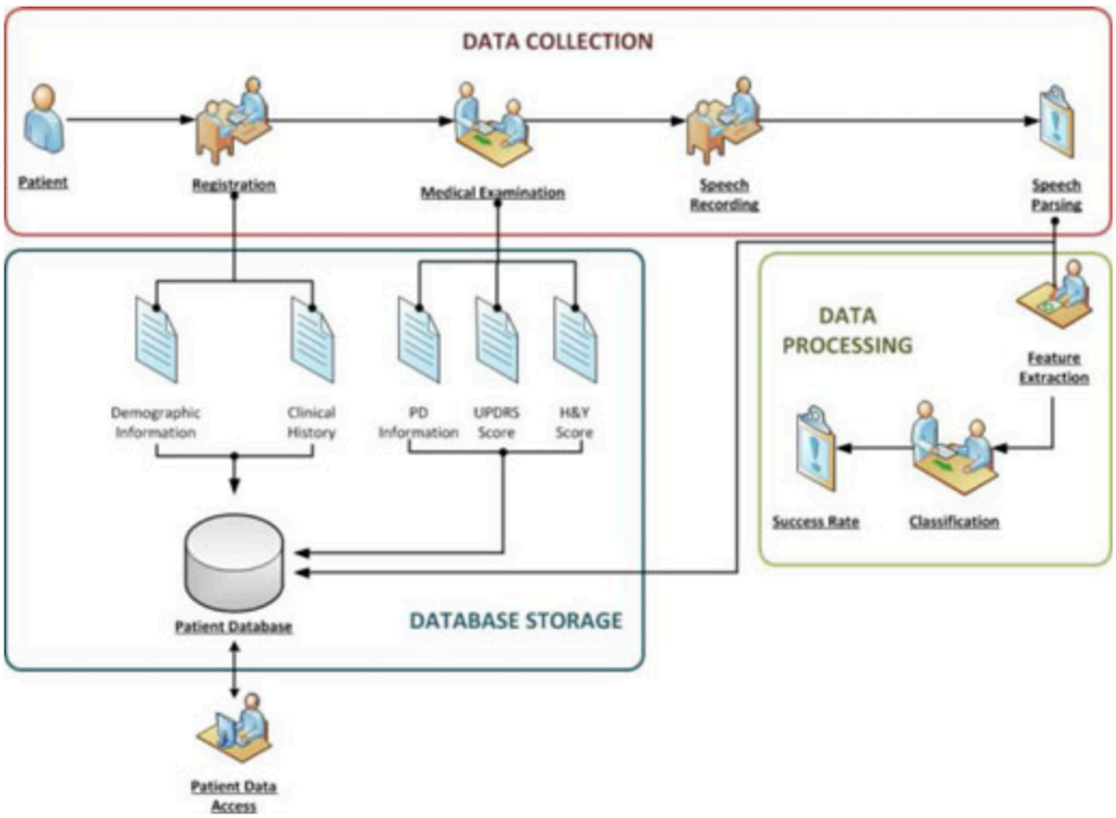
This was an interesting paper for two reasons. It introduces the revolution that a degenerative disease, like Parkinson's, that plagues millions, can be telediagnosed by voiceprint. With later work it developed into an excellent comparative platform for six machine learning algorithms we will run below.

Three of these algorithms had higher than high 95% accuracy we will see in the Results Section below.



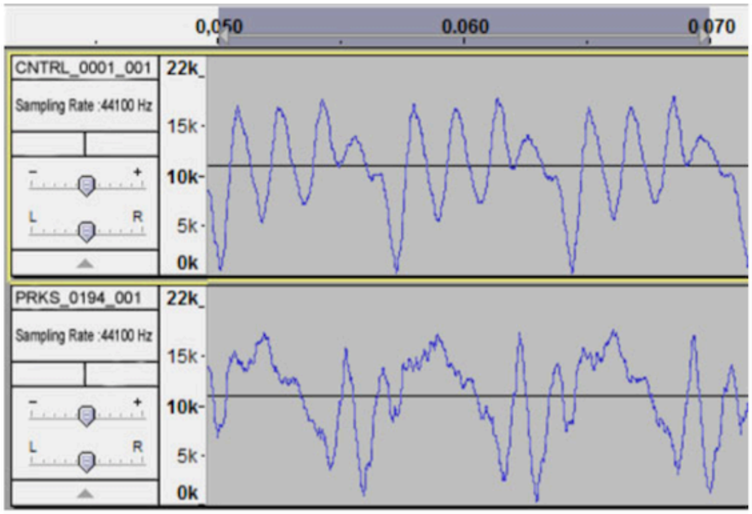
We will run all six of these below in a moment.

The original work, and paper published from it, only used Support Vector Machines and K-Nearest Neighbors. However, it did not detail or discuss a portion of method, process and technology that enabled it to be so effective. This cleverness took place in two steps, the Speech Parsing Section and the Feature Extraction Section of the Data Collection flow shown here in a diagram, Figure 2, from the paper:



IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, VOL. 17, NO. 4, JULY 2013

The speech parsing section was, a presumably manual, extraction of the sound waveforms representing specific utterances like the vowel ‘o’ and the word ‘four’ for each of the patients. The next stage feature extraction was much more significant. In order to produce and label the data provided it was necessary for them to transform the time-domain speech data into frequency-domain samples, and then label specific frequency bands in the data according that were most likely to correspond to patients who were negative and positive for the illness. The clue to this is in the waveform diagram, Figure 1 from the paper:



IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, VOL. 17, NO. 4, JULY 2013

According to the paper, 'the speech is parsed to be split into voice samples, and time-frequency based features are extracted from the voice samples using Praat acoustic analysis software.' It was this specific feature extraction that enables the algorithm to work, and it may be that that portion of the algorithm is being held back as proprietary, but the high frequencies appearing in the bottom waveform are a clue that there are a cluster of frequencies that are diagnostic for Parkinson's and this is the first place that someone trying to reproduce the full experiment would start. These speech aspects are listed in the paper as telltale features of Parkinson's: - dysphonia (defective use of the voice), - hypophonia (reduced volume), - monotone (reduced pitch range), - dysarthria (difficulty with articulation of sounds or syllables) Three of the four are frequency domain attributes and hypophonia is a time domain attribute that can also be acquired by the amplitude of the frequency domain bin contents. Here is a list of the features extracted in the paper, and this list is quickly examined by plotting a series of histograms from the supplied data to confirm plausible trends.

TABLE I
TIME-FREQUENCY-BASED FEATURES EXTRACTED FROM VOICE SAMPLES

Features	Group
Jitter (local) Jitter (local, absolute) Jitter (rap) Jitter (ppq5) Jitter (ddp)	**Frequency parameters**
Number of pulses Number of periods Mean period Standard dev. of period	**Pulse Parameters**
Shimmer (local) Shimmer (local, dB) Shimmer (apq3) Shimmer (apq5) Shimmer (apq11) Shimmer (dda)	**Amplitude parameters**
Fraction of locally unvoiced frames Number of voice breaks Degree of voice breaks	**Voicing Parameters**
Median pitch Mean pitch Standard deviation Minimum pitch Maximum pitch	**Pitch Parameters**
Autocorrelation Noise-to-Harmonic Harmonic -to-Noise	**Harmonicity Parameters**

IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, VOL. 17, NO. 4, JULY 2013

The data collected in the context of this study for the original study included:

- 20 PWP (6 female, 14 male) and
- 20 healthy individuals (10 female, 10 male)

who appealed at the Department of Neurology in Cerrahpaşa School of Medicine, Istanbul University Test group consists of patients who are suffering from PD for 0 to 6 years.

But inspection of the dataset showed that this number had grown from 40 subjects to 195. It is worth noting that in the larger dataset there are about 140 positive patients and 45 negative patients, so the data is unbalanced in that respect. This is noted in the code as well, and be computed exactly if desired. I did not have time to trace the pedigree of the additional data used in the analysis below, but it could be one reason that the accuracy went from 79% and 82% for k-NN and SVM with 40 subjects to 98% and 90% in the 195-subject case.

The Classification methods include:

- Classification with Leave-One-Subject-Out (LOSO)
- Classification with Summarized Leave-One-Out (s-LOO)

Evaluation Metrics: The evaluation metrics are repeated from the paper because they are the bread-and-butter formulas of data science and it never hurts to review them. It also gives the author a chance to practice inserting LaTeX math formulas in Jupyter notebooks and getting them formatted correcly. The figures of merit for classifiers are:

- Accuracy
- Sensitivity
- Specificity
- MCC

Accuracy is the ratio of correctly classified instances to all instances:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where: - TP true positives - TN true negatives - FP false positives - FN false negatives

Sensitivity and specificity are statistical measures of correctly classified positive and negative instances, respectively:

$$sensitivity = \frac{TP}{TP + FN}$$

$$specificity = \frac{TN}{TN + FP}$$

The Receiver Operating Characteristic (ROC) plots sensitivity vs. 1-specificity, but we will label the axes as TP vs. TN for brevity. These graphs are provided by real-time computation below.

Mathews Correlation Coefficient (MCC) is a measure that shows the quality of binary classification in machine learning. It is stable even if the class densities are considerably different. MCC is a correlation coefficient between the predicted and observed binary classifications and gets a value between -1 and $+1$. The formulation of MCC metric is given as follows:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt[2]{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

This coefficient gets the value of: - $+1$ when the classifier makes perfect predictions, - -1 when the predictions and actual values totally disagree, and - 0 when the classification is no better than a random prediction.

2 Libraries

[1]:
```
import math
import os
import sys
import pandas            as pd
import numpy             as np
import matplotlib.pyplot as plt
import seaborn           as sea
import xgboost           as xgb
from scipy                   import interp
from scipy                   import stats
from cycler                  import cycler
from itertools               import cycle
from xgboost                 import XGBClassifier
from sklearn                 import datasets
from sklearn                 import metrics
from sklearn.preprocessing   import MinMaxScaler
from sklearn.preprocessing   import label_binarize
from sklearn.model_selection import train_test_split
from sklearn.linear_model    import LogisticRegression
from sklearn.metrics         import roc_curve
from sklearn.metrics         import auc
from sklearn.metrics         import roc_auc_score
from sklearn.multiclass      import OneVsRestClassifier
from sklearn.naive_bayes     import GaussianNB
from sklearn.neighbors       import KNeighborsClassifier
from sklearn.svm             import SVC
from sklearn.tree            import DecisionTreeClassifier
%matplotlib inline
```

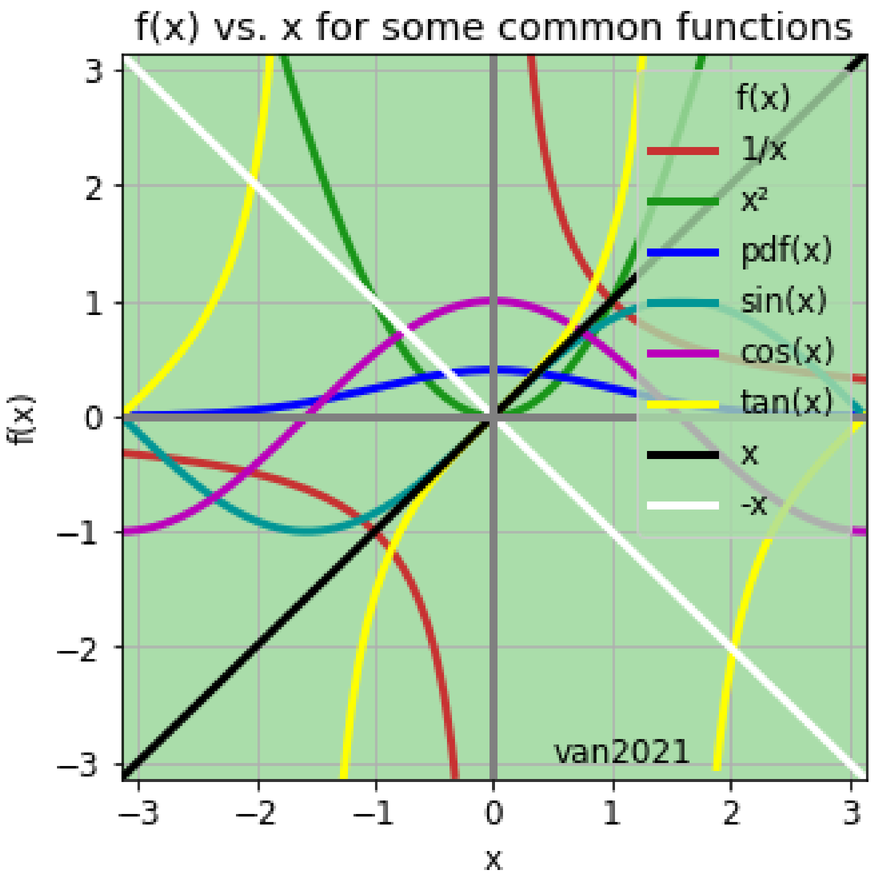
2.1 Van Graphics Lib

[2]:
```
# Extensions to matplotlib
# lvwarren@ualr.edu
# Tue Feb 23 17:16:12 CST 2021

import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from cycler import cycler

van_color_list = ['#C83232', '#199619', '#0000FF',
                  '#009696', '#BB00BB', '#FFFF00',
                  '#000000', '#FFFFFF', '#808080']
```

```
def van_defaults(plot, x_size=10, y_size=10):

    # # My canonical graphing style
    # plt.xkcd()
    plot.figure(figsize=(x_size,y_size))
#   plot.rcParams['figure.figsize'] = [10, 10]
    plot.rcParams['figure.facecolor'] = 'FFFFFF'
    plot.rcParams.update({'font.size': 12})
    plot.rcParams['axes.facecolor'] = '#AADDAA'
    plot.rcParams['lines.linewidth'] = 3
    plot.rcParams['lines.color'] = 'red'
    plot.rcParams['axes.prop_cycle'] = cycler('color', van_color_list)
    plot.grid(True)

def van_labels(plot, x_label, y_label, title):
    plot.xlabel(x_label)
    plot.ylabel(y_label)
    plot.title(title)

def van_limits(plot, mx, px, my, py):
        plot.xlim((mx, px))
        plot.ylim((my, py))

# hidden by underscore because of singularity removal
def van_OoX(x):
    y = 1/x
    y[y >  math.pi] =  np.inf # remove singularity
    y[y < -math.pi] = -np.inf
    return(y)

def van_TaN(x):
    y = np.tan(x)
    y[y >  math.pi] =  np.inf # remove singularity
    y[y < -math.pi] = -np.inf
    return(y)
```

[3]:
```
van_defaults(plt, 5, 5)
van_labels  (plt, 'x', 'f(x)', 'f(x) vs. x for some common functions')
van_limits  (plt, -math.pi, math.pi, -math.pi, math.pi)

x = np.arange(-math.pi, math.pi, 0.01)

function_list = [van_OoX(x), x*x, stats.norm.pdf(x) , np.sin(x) , np.cos(x),␣
 ↪van_TaN(x) , x,  -x ]
function_name = [     '1/x','x²',          'pdf(x)' ,  'sin(x)',   'cos(x)',  ␣
 ↪'tan(x)','x','-x']
```

```
for y in function_list: plt.plot(x,y)
plt.axhline(0, color='#808080')
plt.axvline(0, color='#808080')
plt.text(0.5,-3,'van2021')
plt.legend(function_name, title='    f(x)', loc=1)
plt.show();
```



3 Data Review

3.1 Read Data

[4]:
```
data = pd.read_csv('data/parkinsons.csv')
predictors = data.drop(['name'], axis = 1)
predictors = predictors.drop(['status'], axis = 1).to_numpy()
target = data['status']
data.shape
```

[4]: (195, 24)

3.2 Textual Data Review

- Initially we take a look at the data to make sure the read has worked and that the numbers are plausible.
- We do a data.___head()___ call to look at the first 5 rows, and it also prints the data.___shape___.
- We inspect the *features* by echoing the contents of the variable **target** defined above.

[5]:
```
data.head()
```

[5]:
```
              name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
0  phon_R01_S01_1      119.992       157.302        74.997         0.00784
1  phon_R01_S01_2      122.400       148.650       113.819         0.00968
2  phon_R01_S01_3      116.682       131.111       111.555         0.01050
3  phon_R01_S01_4      116.676       137.871       111.366         0.00997
4  phon_R01_S01_5      116.014       141.781       110.655         0.01284

   MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  …  \
0           0.00007   0.00370   0.00554     0.01109       0.04374  …
1           0.00008   0.00465   0.00696     0.01394       0.06134  …
2           0.00009   0.00544   0.00781     0.01633       0.05233  …
3           0.00009   0.00502   0.00698     0.01505       0.05492  …
4           0.00011   0.00655   0.00908     0.01966       0.06425  …

   Shimmer:DDA      NHR     HNR  status      RPDE       DFA   spread1  \
0      0.06545  0.02211  21.033       1  0.414783  0.815285 -4.813031
1      0.09403  0.01929  19.085       1  0.458359  0.819521 -4.075192
2      0.08270  0.01309  20.651       1  0.429895  0.825288 -4.443179
3      0.08771  0.01353  20.644       1  0.434969  0.819235 -4.117501
4      0.10470  0.01767  19.649       1  0.417356  0.823484 -3.747787

    spread2        D2       PPE
0  0.266482  2.301442  0.284654
1  0.335590  2.486855  0.368674
2  0.311173  2.342259  0.332634
3  0.334147  2.405554  0.368975
4  0.234513  2.332180  0.410335

[5 rows x 24 columns]
```

[6]:
```
target # Presumably labels for 'has Parkinsons' vs. doesn't.
```

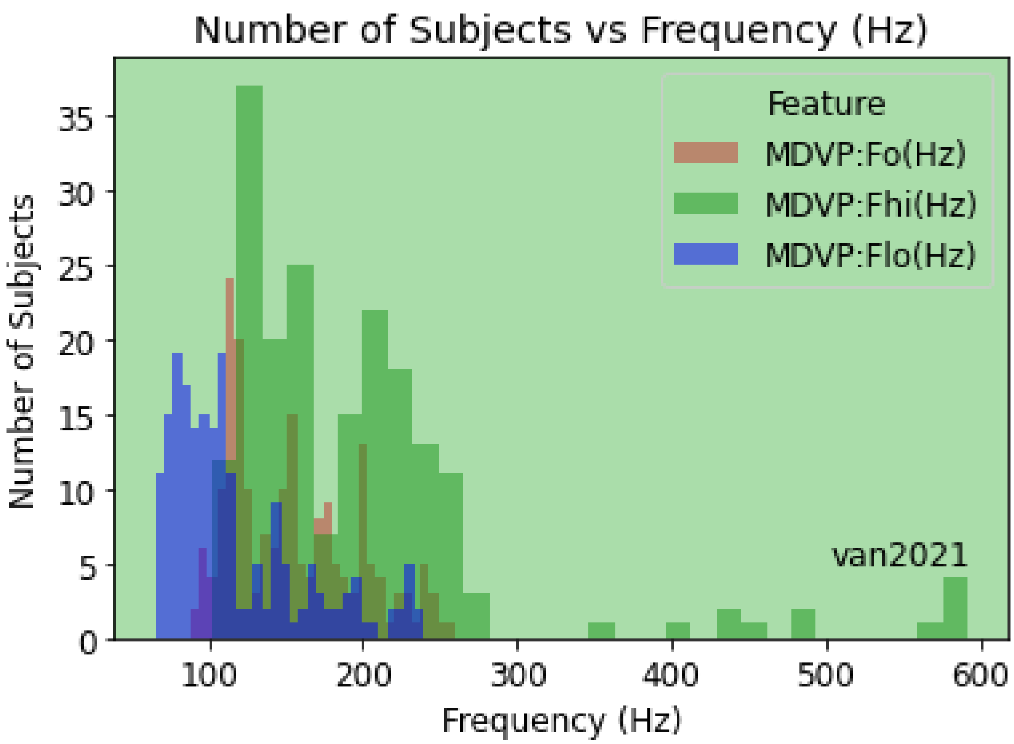
[6]:
```
0      1
1      1
2      1
3      1
4      1
      ..
```

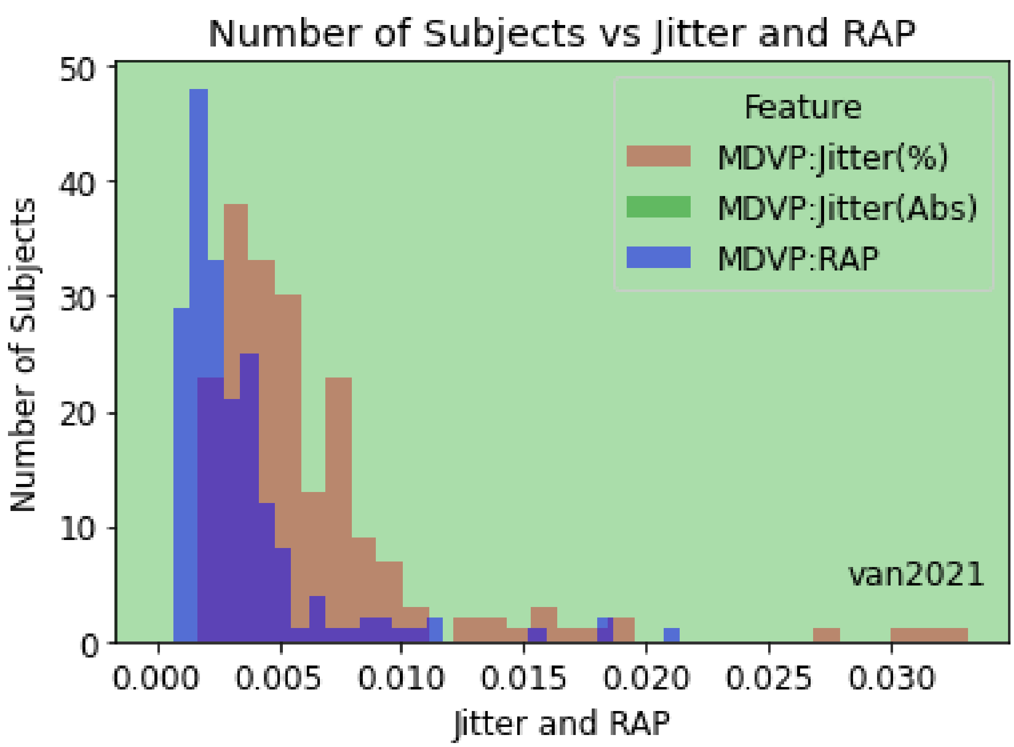
```
190    0
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64
```

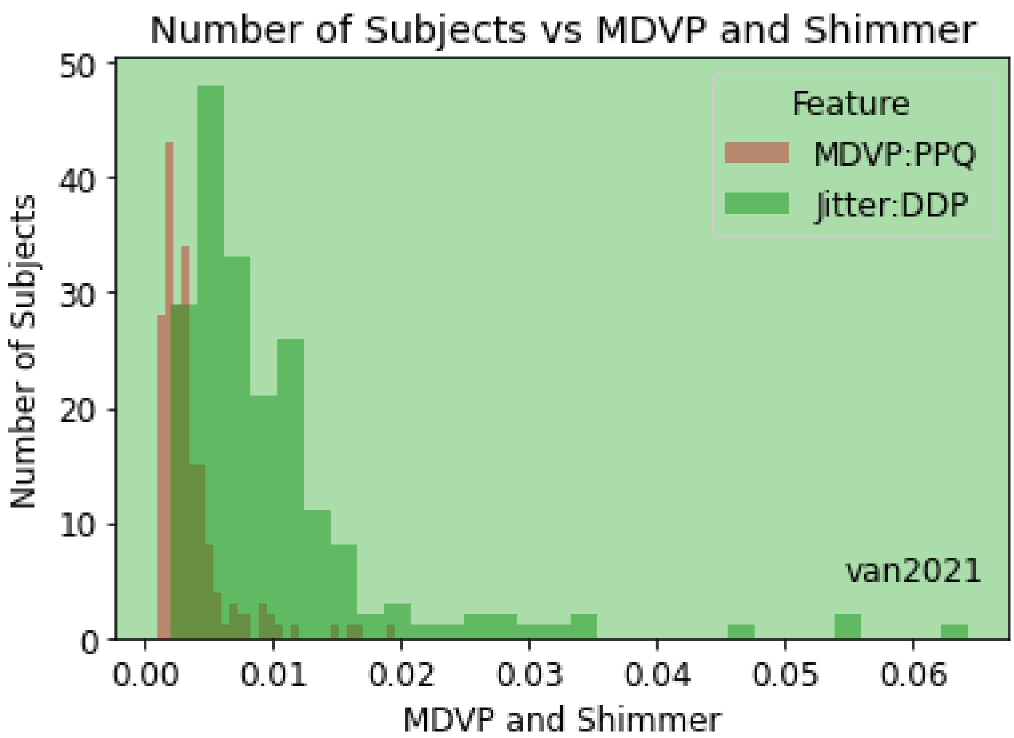
3.3 Graphical Data Review

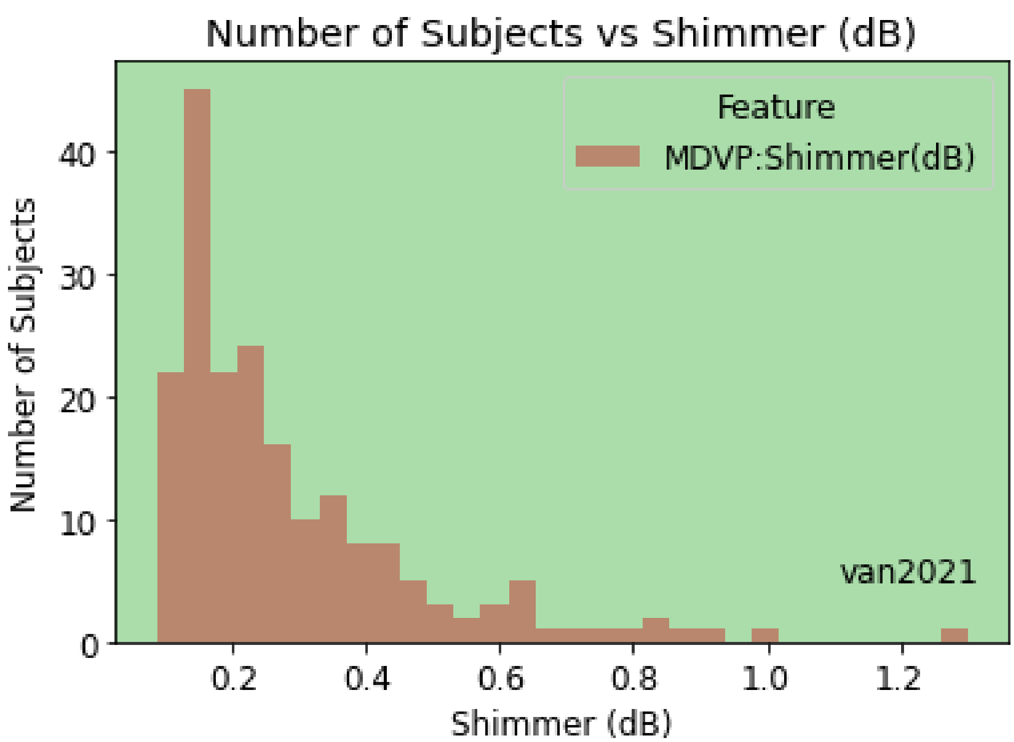
- We can spot distribution shapes and trends
- Note in the last graphic we determine that this data is unbalanced with respect to Parkinson's Status!
- We plot first 16 columns, those after status are derivative statistics

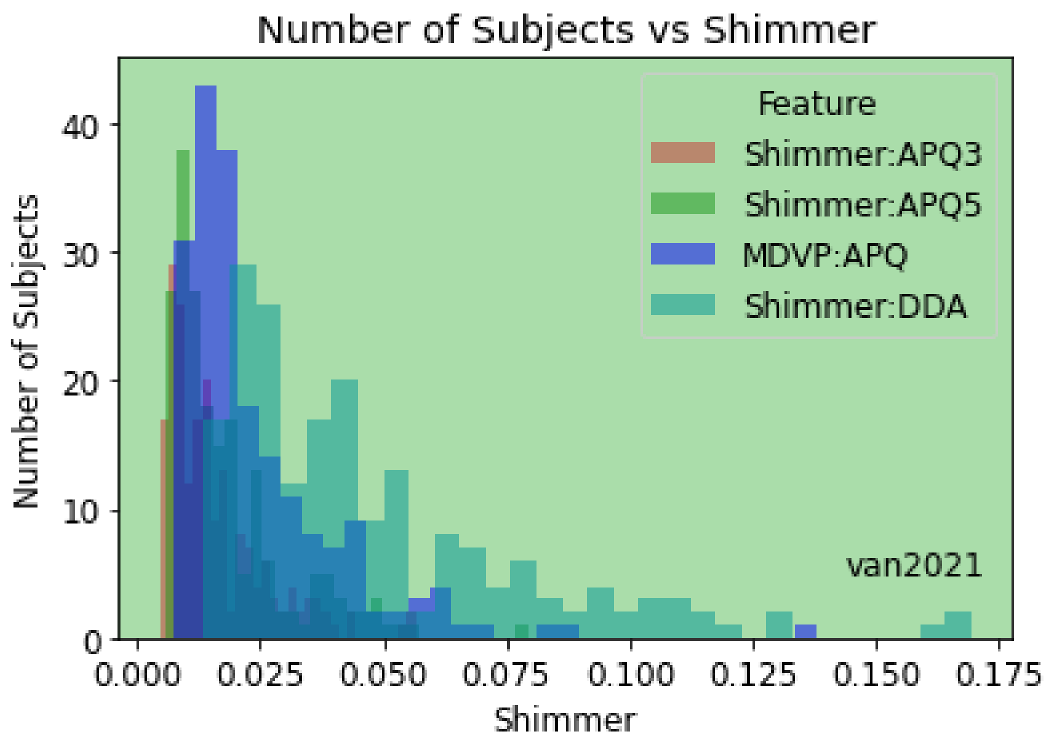
[7]:
```python
def park_plot(a, b, x_title=''):

    data_cols = range(a,b)
    y = [data.iloc[:,i+1] for i in data_cols]
    van_defaults(plt, x_size=6, y_size=4)
    van_labels(plt, x_title, 'Number of Subjects',
               'Number of Subjects vs ' + x_title)

    plt.text(0.85*np.max(y), 5,'van2021')

    [plt.hist(y[i], bins=30, alpha=0.50, label='foo') for i in range(0,b-a)]

    plt.legend(data.columns[np.add(data_cols, 1).tolist()],
               title='Feature', loc=1)
#   plt.axhline(0, color='#000000')
#   plt.axvline(0, color='#000000')
    plt.grid(False)
    plt.show();

park_plot(0,3, 'Frequency (Hz)')
park_plot(3,6, 'Jitter and RAP')
park_plot(6,8, 'MDVP and Shimmer')
park_plot(9,10, 'Shimmer (dB)')
park_plot(10,14, 'Shimmer')
park_plot(14,16, 'NHR and HNR')
park_plot(16,17, 'Parkinson\'s Status')
```


Number of Subjects vs Frequency (Hz)
Feature
MDVP:Fo(Hz)
MDVP:Fhi(Hz)
MDVP:Flo(Hz)
van2021
Number of Subjects
Frequency (Hz)

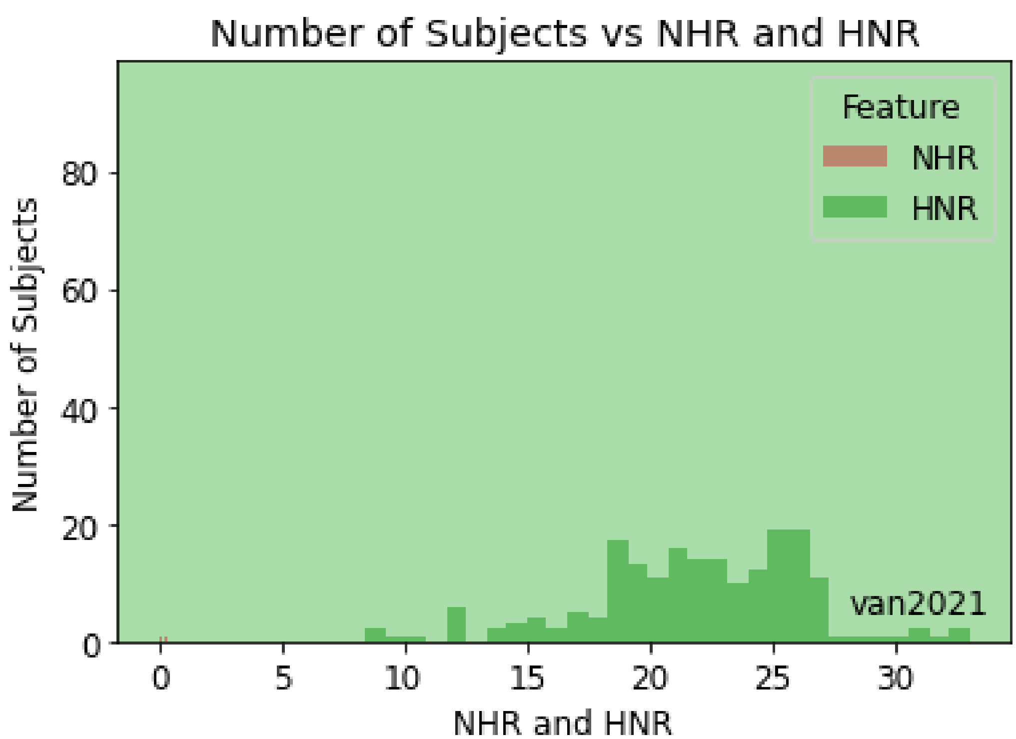

Number of Subjects vs Jitter and RAP
Feature
MDVP:Jitter(%)
MDVP:Jitter(Abs)
MDVP:RAP
van2021
Number of Subjects
Jitter and RAP


Number of Subjects vs MDVP and Shimmer
Number of Subjects
MDVP and Shimmer
Feature
MDVP:PPQ
Jitter:DDP
van2021
0
10
20
30
40
50
0.00
0.01
0.02
0.03
0.04
0.05
0.06

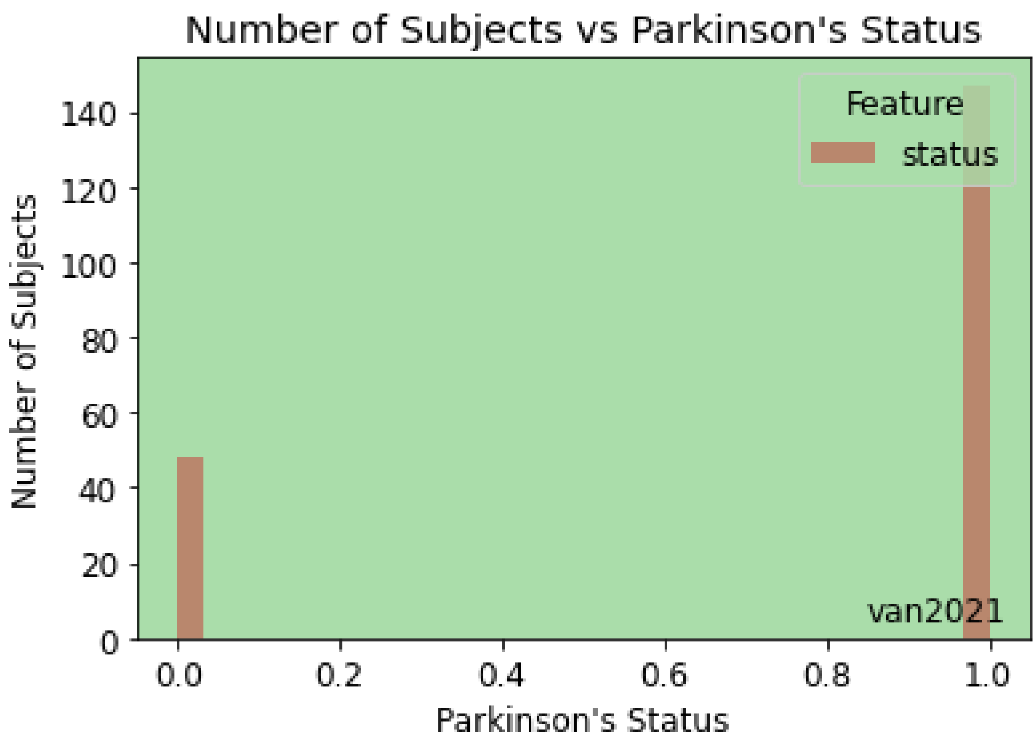

Number of Subjects vs Shimmer (dB)
Number of Subjects
Shimmer (dB)
Feature
MDVP:Shimmer(dB)
van2021
0
10
20
30
40
0.2
0.4
0.6
0.8
1.0
1.2



Number of Subjects vs Shimmer
Feature
Shimmer:APQ3
Shimmer:APQ5
MDVP:APQ
Shimmer:DDA
van2021
Number of Subjects
Shimmer
0
10
20
30
40
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175



Number of Subjects vs NHR and HNR
Feature
NHR
HNR
van2021
Number of Subjects
NHR and HNR
0
20
40
60
80
0
5
10
15
20
25
30



4 Analysis Using Six Different Models

4.1 Scale to Unit Interval and Test/Train Data Split

- Every modeling technique uses scaled version of the data
- Imports sklearn.preprocessing MinMaxScaler
- Remember you have both scaled and inverse transforms available for forward and inverse maps

[8]:
```
import warnings
warnings.filterwarnings('ignore') # ignore superfluous warning

scaler = MinMaxScaler((-1, 1))
X = scaler.fit_transform(predictors)
Y = target

X_train, X_test, Y_train, Y_test = \
    train_test_split(X, Y,test_size = .25, random_state = 7)
```

4.2 Four Utility Functions

- plot_confusion_matrix: a function's name that tells you what it does saves unnecessary comments!

- summarize_model_fit: calls plot_confusion_matrix()
- plot_ROC: plot the Receiver Operating Characteristic curve of True Positives vs. False Positives
- fit_predict_summarize: do the fit, make the predictions, call summarize_model_fit() to assess the result

[9]:
```
def plot_confusion_matrix(CM):
    group_names  = ['True Neg','False Pos','False Neg','True Pos']
    group_counts = ['{0:0.0f}'.format(value) \
                    for value in CM.flatten()]
    group_percents = ['{0:.2%}'.format(value) \
                      for value in CM.flatten()/np.sum(CM)]
    labels = [f'{v1}\n{v2}\n{v3}' \
              for v1, v2, v3 in zip(group_names,group_counts,group_percents)]
    labels = np.asarray(labels).reshape(2,2)
    # plt.figure()
    van_defaults(plt, x_size=4.5, y_size=4.5)
    sea.heatmap(CM, annot=labels, fmt='', cmap='Blues')
    plt.show()

def summarize_model_fit(title, test, pred):
    print(f'{title} Model Accuracy: \
    {round(metrics.accuracy_score(test, pred)*100,1)}%')
#   print(metrics.classification_report(test, pred))
    print('Confusion Matrix: ')
    plot_confusion_matrix(metrics.confusion_matrix(test, pred))

def plot_ROC(name, Y_test, y_pred):
    fpr, tpr, threshold = metrics.roc_curve(Y_test, y_pred)
    van_defaults(plt, x_size=3.75, y_size=3.75)
    van_labels(plt, 'False Positive Rate',
                    'True  Positive Rate',
                    'Receiver Operating Characteristic')
    plt.plot(fpr,tpr) # ROC curve
    ident = [0.0, 1.0]
    plt.plot(ident,ident)
    auc = np.trapz(tpr,fpr) # compute the area using trapezoidal rule
    plt.legend([round(auc,2)], title='AUC', loc=4)
    plt.text(0.9, -0.30,'van2021', fontsize=10)
    plt.text(0.1,  0.90 , name, fontsize=12)
    plt.show()

def fit_predict_summarize(model, name, X_train_, Y_train_, X_test_, Y_test_):
    model.fit(X_train_, Y_train_)
    y_pred_ = model.predict(X_test_)
    summarize_model_fit(name , Y_test_, y_pred_)
    plot_ROC(name, Y_test_, y_pred_)
```

4.3 Six Standard Models and Results

- XG Boost Classifier - 98%
- Logistic Regression - 88%
- Gaussian Naive Bayes - 69%
- K-Nearest Neighbor - 98%
- Support Vector Machine - 90%
- Classification and Regression Trees - 96%

Note that these analyses all take similar form of fit, predict and summarize.

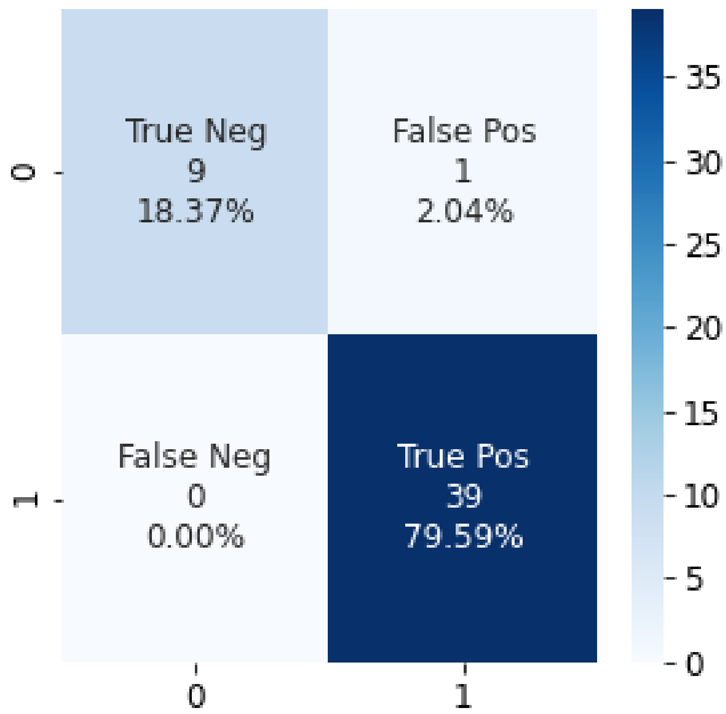
4.4 Use Function Dictionary to Invoke Models

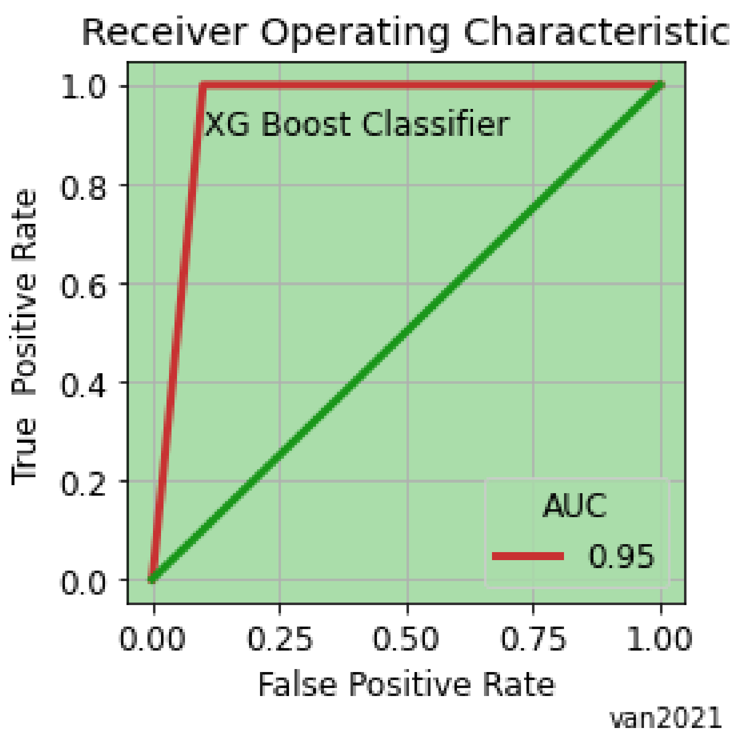
[10]:
```
# Note to Self: Use Dictionary as Function Dispatcher, so Pythonic!
model_dict = {
              'XG Boost Classifier'    : XGBClassifier(eval_metric='logloss'),
              'Logistic Regression'    : LogisticRegression(),
              'Gaussian Naive Bayes'   : GaussianNB(),
              'K-Nearest-Neigbhors'    : KNeighborsClassifier(),
              'Support Vector Machine' : SVC(),
              'Decision Trees'         : DecisionTreeClassifier()
             }
```

4.5 Detailed Results Confusion Matrices and ROC Plots

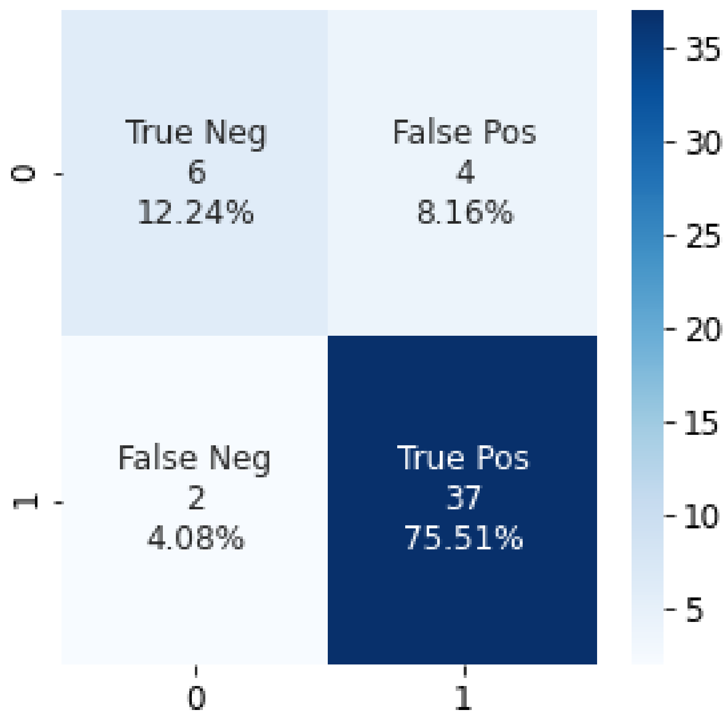
[11]:
```
for name, model in model_dict.items():
    fit_predict_summarize(model, name, X_train, Y_train, X_test, Y_test)
```

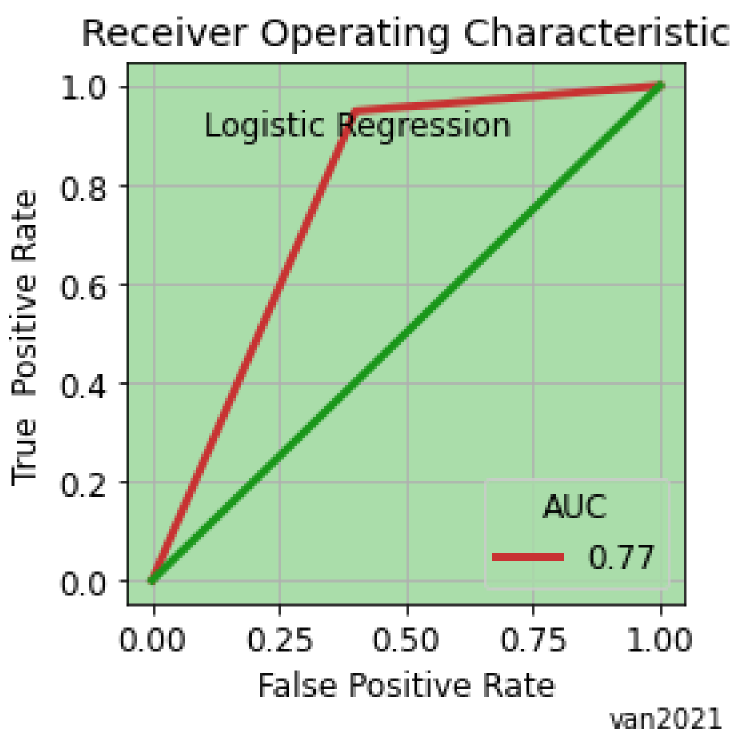
```
XG Boost Classifier Model Accuracy:     98.0%
Confusion Matrix:
```


True Neg
9
18.37%
False Pos
1
2.04%
False Neg
0
0.00%
True Pos
39
79.59%
0
1
0
1
35
30
25
20
15
10
5
0

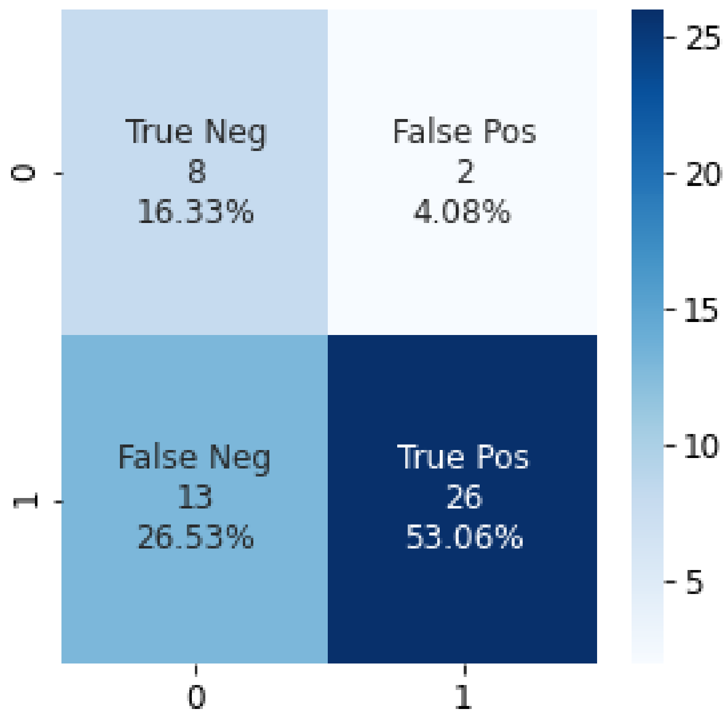

Receiver Operating Characteristic
XG Boost Classifier
True Positive Rate
False Positive Rate
AUC
0.95
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
van2021

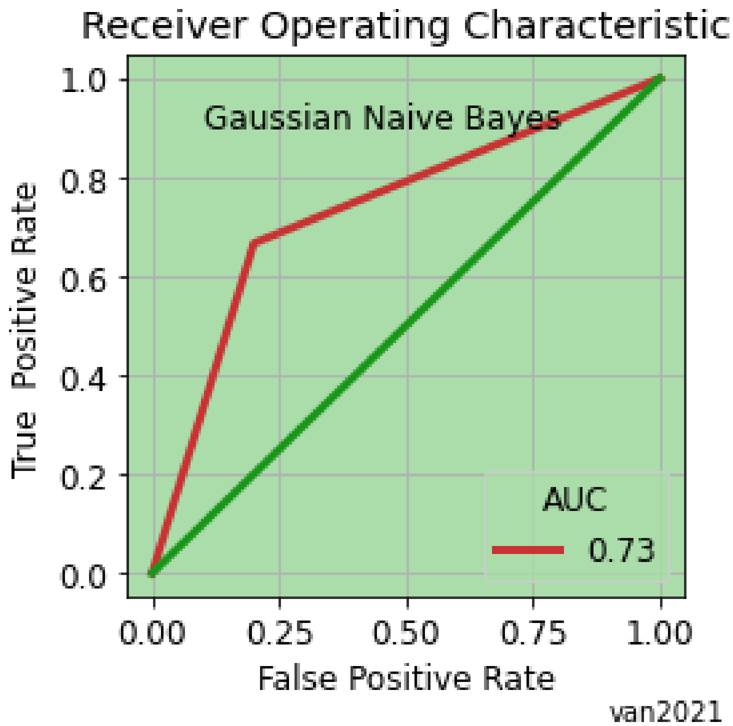
```
Logistic Regression Model Accuracy:      87.8%
Confusion Matrix:
```



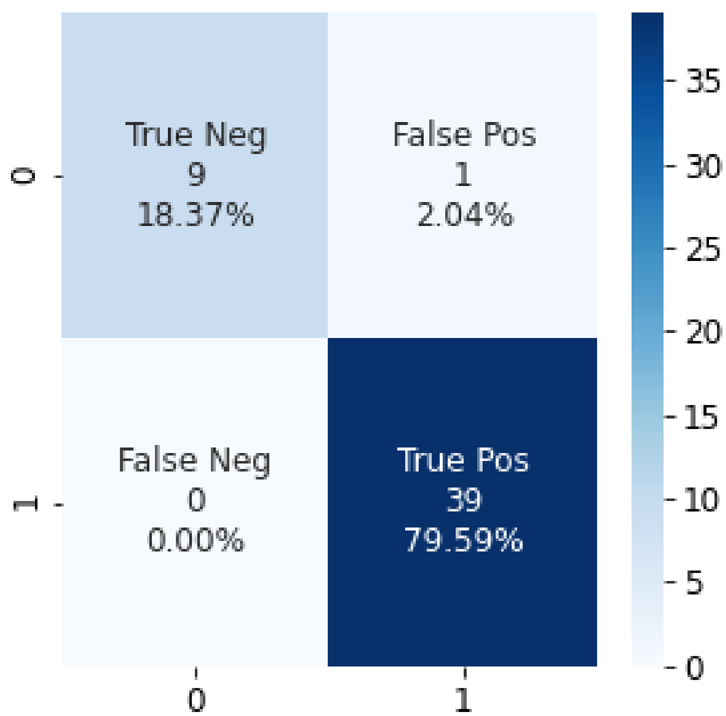


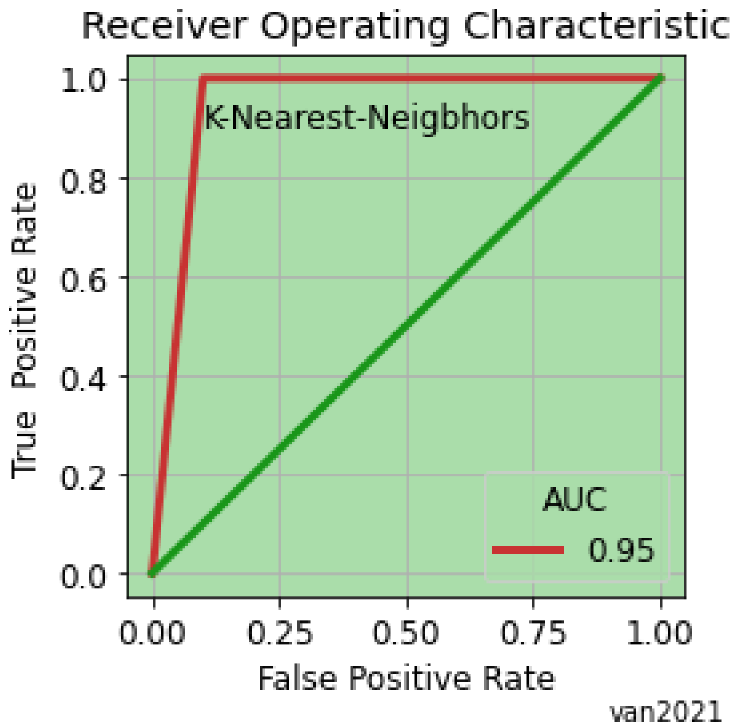
```
Gaussian Naive Bayes Model Accuracy:     69.4%
Confusion Matrix:
```



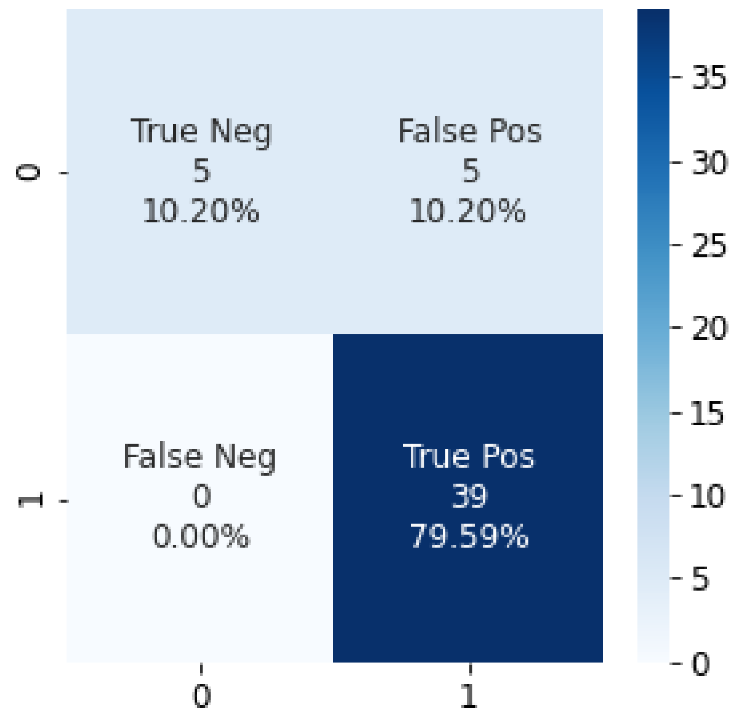


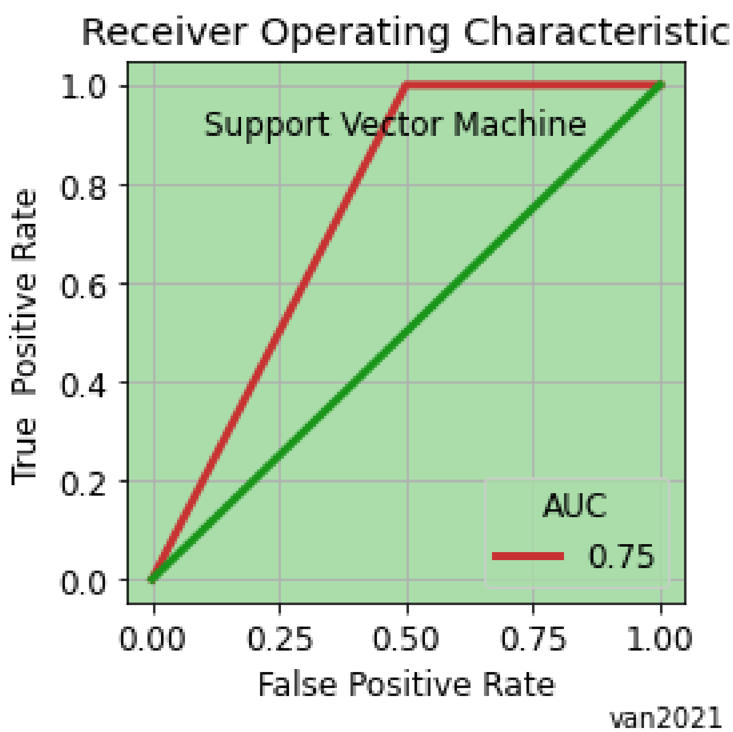

```
K-Nearest-Neigbhors Model Accuracy:     98.0%
Confusion Matrix:
```



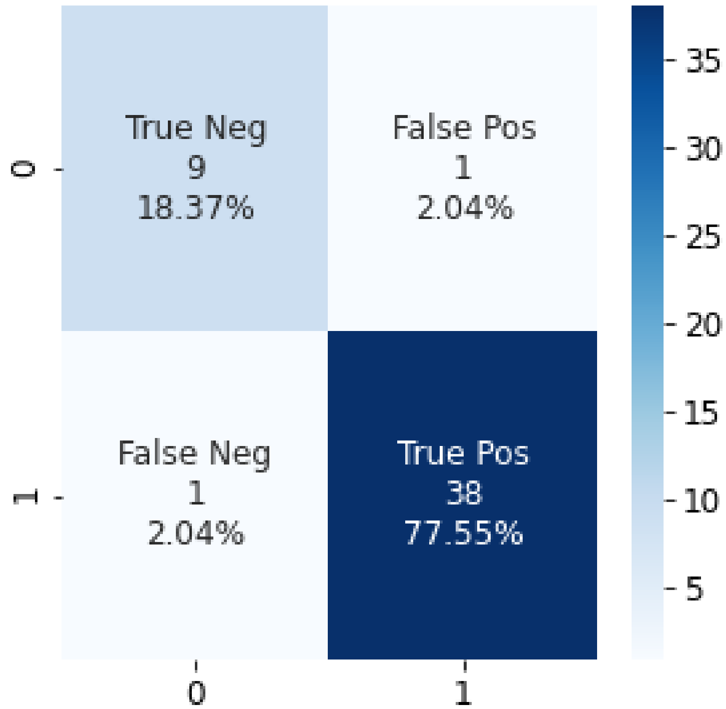


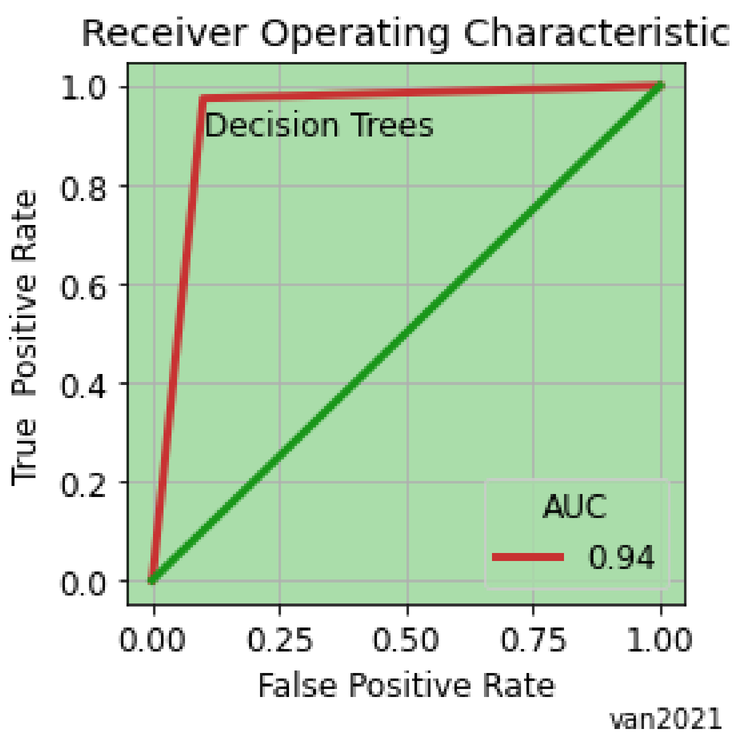

```
Support Vector Machine Model Accuracy:     89.8%
Confusion Matrix:
```


True Neg
5
10.20%
False Pos
5
10.20%
False Neg
0
0.00%
True Pos
39
79.59%
0
1
0
1
35
30
25
20
15
10
5
0


Receiver Operating Characteristic
Support Vector Machine
True Positive Rate
False Positive Rate
AUC
0.75
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
van2021

```
Decision Trees Model Accuracy:     95.9%
Confusion Matrix:
```





4.5.1 References

- Original Parkinson Speech Paper
- Parkinson Data
- Parkinson Data Python
- Parkinson Voice Data
- Python Dictionary Comprehension
- Storing Function in Python Dictionary
- Hide Warnings in Python
- ROC Plots
- Numpy max for list of lists
- Adorn Histogram Text
- Add Two Lists Element-Wise
- Mac pbcopy trick
- Panda Dataframe Column Names
- Python Lists
- More Python Lists
- Pandas Dataframe Row and Column Selection
- Pandas Dataframe First Column
- Dataframe as_matrix Deprecation
- XGBoost Introduction
- K Nearest Neighbors Algorithm
- Logistic Regression
- Logistic Regression Sklearn
- K Nearst Neighbors Classifier
- Print Key Value Dictionary Pairs

4.5.2 Acknowledgements

This project extensively modifies code from a University of Maryland Research Project by: Shlok Khandelwal and Elcin Ergin, Shu Hayakawa, and Timardeep Kaur
It utilizes a confusion matrix pretty printer by Dennis T
Code Repository by Shlok Khandelwal on Github