Parkinson's Disease Detection

July 20, 2022

1 Parkinson's Disease Detection Model

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

[2]:
```
pdf = pd.read_csv('C:/Users/ASV ARUN/Parkinsson disease.csv', header = 0)
pdf.head()
```

[2]:
```
            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]
```

[3]:

```
pdf.describe()
```

[3]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)
count	195.000000	195.000000	195.000000	195.000000
mean	154.228641	197.104918	116.324631	0.006220
std	41.390065	91.491548	43.521413	0.004848
min	88.333000	102.145000	65.476000	0.001680
25%	117.572000	134.862500	84.291000	0.003460
50%	148.790000	175.829000	104.315000	0.004940
75%	182.769000	224.205500	140.018500	0.007365
max	260.105000	592.030000	239.170000	0.033160

	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer
count	195.000000	195.000000	195.000000	195.000000	195.000000
mean	0.000044	0.003306	0.003446	0.009920	0.029709
std	0.000035	0.002968	0.002759	0.008903	0.018857
min	0.000007	0.000680	0.000920	0.002040	0.009540
25%	0.000020	0.001660	0.001860	0.004985	0.016505
50%	0.000030	0.002500	0.002690	0.007490	0.022970
75%	0.000060	0.003835	0.003955	0.011505	0.037885
max	0.000260	0.021440	0.019580	0.064330	0.119080

	MDVP:Shimmer(dB)	...	Shimmer:DDA	NHR	HNR	status
count	195.000000	...	195.000000	195.000000	195.000000	195.000000
mean	0.282251	...	0.046993	0.024847	21.885974	0.753846
std	0.194877	...	0.030459	0.040418	4.425764	0.431878
min	0.085000	...	0.013640	0.000650	8.441000	0.000000
25%	0.148500	...	0.024735	0.005925	19.198000	1.000000
50%	0.221000	...	0.038360	0.011660	22.085000	1.000000
75%	0.350000	...	0.060795	0.025640	25.075500	1.000000
max	1.302000	...	0.169420	0.314820	33.047000	1.000000

	RPDE	DFA	spread1	spread2	D2	PPE
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000
mean	0.498536	0.718099	-5.684397	0.226510	2.381826	0.206552
std	0.103942	0.055336	1.090208	0.083406	0.382799	0.090119
min	0.256570	0.574282	-7.964984	0.006274	1.423287	0.044539
25%	0.421306	0.674758	-6.450096	0.174351	2.099125	0.137451
50%	0.495954	0.722254	-5.720868	0.218885	2.361532	0.194052
75%	0.587562	0.761881	-5.046192	0.279234	2.636456	0.252980
max	0.685151	0.825288	-2.434031	0.450493	3.671155	0.527367

[8 rows x 23 columns]

Checking for Outliers in the Data : Outliers are points which are not present within the range of $\mu - 3\sigma$ and $\mu + 3\sigma$. That is if minimum and maximum of the data lies in this range $\mu - 3\sigma$ and $\mu + 3\sigma$ (or atleast within the neighborhood of the boundaries) then there will be no outliers for the data.

[4]:
```python
pdf.mean(axis=0)-3*pdf.std(axis=0)
```

```
C:\Users\ASV ARUN\AppData\Local\Temp\ipykernel_15152\1828174494.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError.  Select only valid columns before calling the reduction.
  pdf.mean(axis=0)-3*pdf.std(axis=0)
```

[4]:
```
MDVP:Fo(Hz)          30.058447
MDVP:Fhi(Hz)        -77.369725
MDVP:Flo(Hz)        -14.239609
MDVP:Jitter(%)       -0.008324
MDVP:Jitter(Abs)     -0.000061
MDVP:RAP             -0.005597
MDVP:PPQ             -0.004831
Jitter:DDP           -0.016790
MDVP:Shimmer         -0.026862
MDVP:Shimmer(dB)     -0.302381
Shimmer:APQ3         -0.014795
Shimmer:APQ5         -0.018193
MDVP:APQ             -0.026759
Shimmer:DDA          -0.044385
NHR                  -0.096408
HNR                   8.608682
status               -0.541788
RPDE                  0.186710
DFA                   0.552092
spread1              -8.955020
spread2              -0.023707
D2                    1.233429
PPE                  -0.063806
dtype: float64
```

[5]:
```python
pdf.mean(axis=0)+3*pdf.std(axis=0)
```

```
C:\Users\ASV ARUN\AppData\Local\Temp\ipykernel_15152\2041545410.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError.  Select only valid columns before calling the reduction.
  pdf.mean(axis=0)+3*pdf.std(axis=0)
```

[5]:
```
MDVP:Fo(Hz)          278.398835
MDVP:Fhi(Hz)         471.579561
MDVP:Flo(Hz)         246.888870
MDVP:Jitter(%)         0.020765
MDVP:Jitter(Abs)       0.000148
MDVP:RAP               0.012210
MDVP:PPQ               0.011723
```

```
Jitter:DDP            0.036630
MDVP:Shimmer          0.086280
MDVP:Shimmer(dB)      0.866883
Shimmer:APQ3          0.046124
Shimmer:APQ5          0.053949
MDVP:APQ              0.074922
Shimmer:DDA           0.138370
NHR                   0.146102
HNR                  35.163267
status                2.049480
RPDE                  0.810361
DFA                   0.884107
spread1              -2.413773
spread2               0.476728
D2                    3.530223
PPE                   0.476910
dtype: float64
```

Therfore there are no outliers in the data as minimum and maximum of the dataset always lie in the range of $\mu - 3\sigma$ and $\mu + 3\sigma$.

[6]:
```
pdf.isna().sum()
```

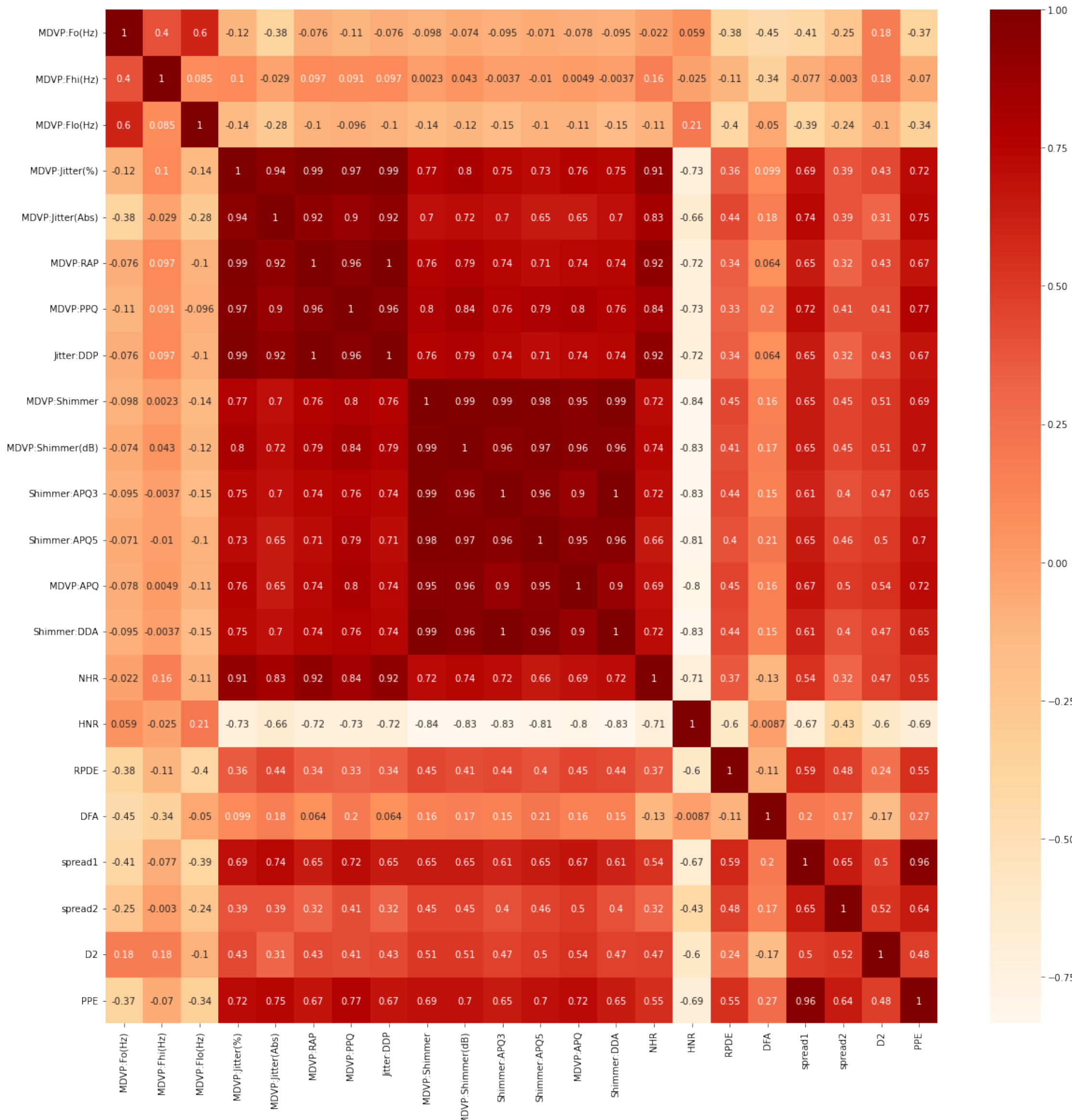
[6]:
```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

So therefore there are no missing values in the data.

[7]:
```
del pdf['name']
```

[8]:
```
x = pdf.loc[:, pdf.columns != 'status']
y = pdf['status']
```

[9]:
```
import matplotlib.pyplot as plt
correl=x.corr()
plt.figure(figsize=(20,20))
sns.heatmap(correl,annot=True,cmap='OrRd')
plt.show()
```



[10]:
```
x=x.drop(['MDVP:Jitter(Abs)','MDVP:RAP','MDVP:PPQ','Jitter:DDP','MDVP:
↪Shimmer(dB)','Shimmer:APQ3','Shimmer:APQ5','MDVP:APQ','Shimmer:
↪DDA','NHR'],axis=1)
```

Therefore now our variables aren't correlated

1.0.1 Test Train Split Correlation Matrix

[11]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,␣
↪random_state = 9)
```

1.1 Logistic Regression

[12]:
```
from sklearn.linear_model import LogisticRegression
model_logi = LogisticRegression()
model_logi.fit(x_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[12]:
```
LogisticRegression()
```

[13]:
```
logi_predict_train = model_logi.predict(x_train)
logi_predict_test = model_logi.predict(x_test)
```

[14]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
confusion_matrix(y_train, logi_predict_train)
```

[14]:
```
array([[ 20,  17],
       [  5, 114]], dtype=int64)
```

[15]:
```
confusion_matrix(y_test, logi_predict_test)
```

[15]:
```
array([[ 6,  5],
       [ 2, 26]], dtype=int64)
```

[16]:
```
accuracy_logi_test = accuracy_score(y_test, logi_predict_test)
accuracy_logi_train = accuracy_score(y_train, logi_predict_train)
```

```
accuracy_logi_train, accuracy_logi_test
```

[16]:
```
(0.8589743589743589, 0.8205128205128205)
```

[17]:
```
from sklearn.metrics import precision_score, recall_score, roc_auc_score
```

[18]:
```
precision_score(y_test, logi_predict_test)
```

[18]:
```
0.8387096774193549
```

[19]:
```
recall_score(y_test, logi_predict_test)
```

[19]:
```
0.9285714285714286
```

[20]:
```
roc_auc_score(y_test, logi_predict_test)
```

[20]:
```
0.737012987012987
```

Therefore our Logistic Regression Model is accurate as the roc_auc_score() for our model is fairly close to 1.

1.2 Linear Discriminant Analysis

[21]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

[22]:
```
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(x_train,y_train)
```

[22]:
```
LinearDiscriminantAnalysis()
```

[23]:
```
ytrain_pred_lda = clf_lda.predict(x_train)
ytest_pred_lda = clf_lda.predict(x_test)
```

[24]:
```
confusion_matrix(y_train,ytrain_pred_lda)
```

[24]:
```
array([[ 21,  16],
       [  4, 115]], dtype=int64)
```

[25]:
```
confusion_matrix(y_test,ytest_pred_lda)
```

[25]:
```
array([[ 6,  5],
       [ 1, 27]], dtype=int64)
```

[26]:
```
accuracy_score(y_train,ytrain_pred_lda)
```

[26]:
```
0.8717948717948718
```

[27]:
```
accuracy_score(y_test,ytest_pred_lda)
```

[27]: 0.8461538461538461

Therefore our model is moderately accuracate.

1.3 K-Nearest Neighbors using Grid Search

[28]:
```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(x_train)
x_train_s = scaler.transform(x_train)
```

[29]:
```
scaler = preprocessing.StandardScaler().fit(x_test)
x_test_s = scaler.transform(x_test)
```

[30]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

[31]:
```
pars = {'n_neighbors' :[1,2,3,4,5,6,7,8,9]}
grid_search_cv = GridSearchCV(KNeighborsClassifier(),pars)
grid_search_cv.fit(x_train_s,y_train)
```

[31]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9]})
```

[32]:
```
grid_search_cv.best_params_
```

[32]:
```
{'n_neighbors': 1}
```

[33]:
```
optimised_KNN=grid_search_cv.best_estimator_
```

[34]:
```
y_test_pred = optimised_KNN.predict(x_test_s)
```

[35]:
```
confusion_matrix(y_test,y_test_pred)
```

[35]:
```
array([[10,  1],
       [ 0, 28]], dtype=int64)
```

[36]:
```
accuracy_score(y_test,y_test_pred)
```

[36]: 0.9743589743589743

Even the KNN model is also accurate.

1.4 Classification Tree

[37]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
clf_tree = DecisionTreeClassifier(max_depth = 5)
clf_tree=clf_tree.fit(x_train, y_train)
```

```
[38]: y_predict_train = clf_tree.predict(x_train)
      y_predict_test = clf_tree.predict(x_test)
```

```
[39]: confusion_matrix(y_test,y_predict_test)
```

```
[39]: array([[ 7,  4],
             [ 2, 26]], dtype=int64)
```

```
[40]: accuracy_score(y_test,y_predict_test)
```

```
[40]: 0.8461538461538461
```

1.4.1 Plotting Decision Tree

```
[41]: plt.figure(figsize=(40,20))
      plot_tree(clf_tree, feature_names = x_train.columns ,filled=True)
      plt.title("Decision tree training for training dataset")
      plt.show()
```