DATA MINING ASSIGNMENT

PARKINSON'S DISEASE PREDICTION

Manipal University
Jaipur

By-
Name: Jyoti Tuli
Reg No.: 179302068

Background of The Study

Data Mining is rapidly growing to occupy all the industries of the world today. This is just the beginning. The industries collect huge amounts of data containing hidden information useful for making effective decisions by providing appropriate results using data mining techniques.

Data mining knowledge is used to give a user-oriented approach to new and hidden patterns in the data which can be used by the medical experts for predicting Parkinson's disease which can improve the entire research and prevention process.

CONTENTS

Introduction

In this project, multiple modern machine learning and pattern recognition methods have been used in order to classify or predict the risk of Parkinson's disease based on the data. The methods discussed in this project consist of a number of classification methods (i.e. Naïve Bayes, K-NN, Decision Trees, Logistic Regression and SVM).

Libraries Used

Import libraries

```
import numpy as np                  #Large collection of high-level mathematical functions
import pandas as pd                 #Used for data manipulation and analysis
import matplotlib.pyplot as plt     #Collection of command style functions
import seaborn as sns               #Data visualization library based on matplotlib
```

```
#sklearn- software machine learning library
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix , classification_report , accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

Loading Dataset

Reading Dataset

```
df = pd.read_csv("data.csv")
```

```
#Display dataset
df.head()
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shir
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...	
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...	
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...	
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...	
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...	

5 rows × 24 columns

```
df.info()
```

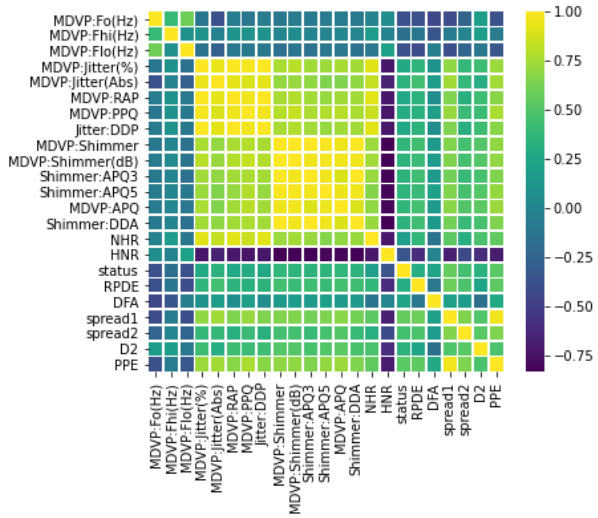
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              195 non-null    object
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

Data Visualization

Heatmap

```
corr = df.corr()
# plot the heatmap
fig = plt.figure(figsize=(6,5))
sns.heatmap(corr,linewidths=.75,cmap= 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ff7d2205b0>
```



Bar graph

```
plt.rcParams['figure.figsize'] = 6,4
plt.bar(df['status'].unique(), df['status'].value_counts(), color = ['lightseagreen', 'lightpink'])
plt.xticks([0, 1])
plt.xlabel('Target Classes')
plt.ylabel('Count')
plt.title('Count of each Target Class')
```

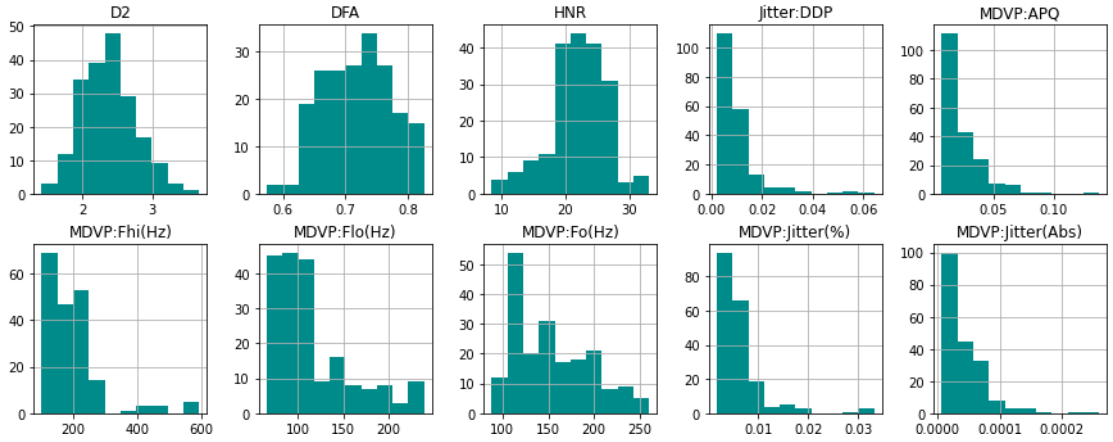
```
Text(0.5, 1.0, 'Count of each Target Class')
```

Histogram

```
fig = plt.figure(figsize = (15,15))
ax = fig.gca()

df.hist(ax=ax,color='darkcyan')
plt.show()
```

```
<ipython-input-14-3b63bfaaf83b>:4: UserWarning: To output multiple subplots, the figure containing the passed axes is being cle
ared
  df.hist(ax=ax,color='darkcyan')
```



Data Preprocessing

Data Preprocessing

```
correlation_values=df.corr()['status']
correlation_values.abs().sort_values(ascending=False)
```

```
status              1.000000
spread1             0.564838
PPE                 0.531039
spread2             0.454842
MDVP:Fo(Hz)         0.383535
MDVP:Flo(Hz)        0.380200
MDVP:Shimmer        0.367430
MDVP:APQ            0.364316
HNR                 0.361515
Shimmer:APQ5        0.351148
MDVP:Shimmer(dB)    0.350697
Shimmer:APQ3        0.347617
Shimmer:DDA         0.347608
D2                  0.340232
MDVP:Jitter(Abs)    0.338653
RPDE                0.308567
MDVP:PPQ            0.288698
MDVP:Jitter(%)      0.278220
MDVP:RAP            0.266668
Jitter:DDP          0.266646
DFA                 0.231739
NHR                 0.189429
MDVP:Fhi(Hz)        0.166136
Name: status, dtype: float64
```

```
# Checking null value sum
df.isna().sum()
```

```
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
```

```
y=df['status']
cols=['MDVP:RAP','Jitter:DDP','DFA','NHR','MDVP:Fhi(Hz)','name','status']
x=df.drop(cols,axis=1)
```

Data Splitting

Spliting of Data

```
train_size=0.80
test_size=0.20
seed=5

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=train_size,test_size=test_size,random_state=seed)
```

Naïve Bayes Algorithm

Naive Bayes Algorithm

```
#Create a Gaussian Classifier
nb = GaussianNB()

# Train the model using the training sets
nb.fit(x_train, y_train)

#Predict Output
nb_pred = nb.predict(x_test)
```

```
#Predicting the score
nb_score = accuracy_score(y_test, nb_pred)
nb_score
```

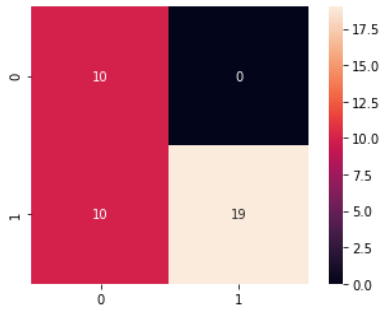
```
0.7435897435897436
```

```
cf_matrix=confusion_matrix(y_test , nb_pred)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , nb_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67        10
           1       1.00      0.66      0.79        29

    accuracy                           0.74        39
   macro avg       0.75      0.83      0.73        39
weighted avg       0.87      0.74      0.76        39
```



Support Vector Machine Algorithm

Support Vector Algorithm

```
#Create a SVM Classifier
svm_class = svm.SVC()

# Train the model using the training sets
svm_class.fit(x_train, y_train)

#Predict Output
svm_pred = svm_class.predict(x_test)
```

```
#Predicting the score
svm_score = accuracy_score(y_test, svm_pred)
svm_score
```

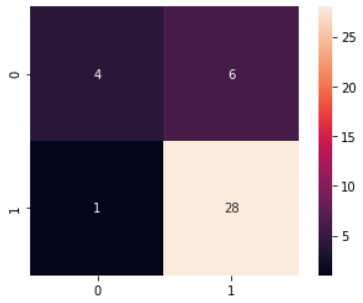
```
0.8205128205128205
```

```
cf_matrix=confusion_matrix(y_test , svm_pred)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , svm_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.40      0.53        10
           1       0.82      0.97      0.89        29

    accuracy                           0.82        39
   macro avg       0.81      0.68      0.71        39
weighted avg       0.82      0.82      0.80        39
```



K Nearest Neighbours

K-Nearest Neighbours

```
#Create a KNN Classifier
model = KNeighborsClassifier(n_neighbors=7)

# Train the model using the training sets
model.fit(x_train,y_train)

#Predict Output
y_predicted= model.predict(x_test)
```

```
knn_score = accuracy_score(y_test, y_predicted)
knn_score
```

```
0.8461538461538461
```

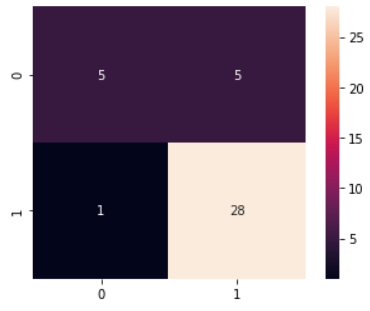
```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.83      0.50      0.62        10
           1       0.85      0.97      0.90        29

    accuracy                           0.85        39
   macro avg       0.84      0.73      0.76        39
weighted avg       0.84      0.85      0.83        39
```



Decision Tree

Decision Tree

```
#Create a KNN Classifier
model = DecisionTreeClassifier()
# Train the model using the training sets
model.fit(x_train,y_train)

#Predict Output
y_predicted= model.predict(x_test)
```

```
dt_score = accuracy_score(y_test, y_predicted)
dt_score
```

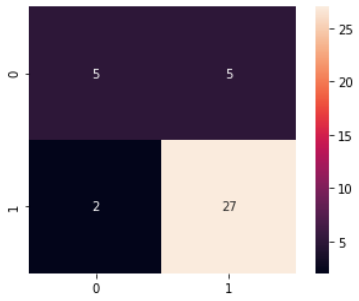
```
0.8974358974358975
```

```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.71      0.50      0.59        10
           1       0.84      0.93      0.89        29

    accuracy                           0.82        39
   macro avg       0.78      0.72      0.74        39
weighted avg       0.81      0.82      0.81        39
```



Logistic Regression

Logistic Regression

```
#Create a KNN Classifier
model = LogisticRegression()
# Train the model using the training sets
model.fit(x_train,y_train)

#Predict Output
y_predicted= model.predict(x_test)
```

```
lr_score = accuracy_score(y_test, y_predicted)
lr_score
```

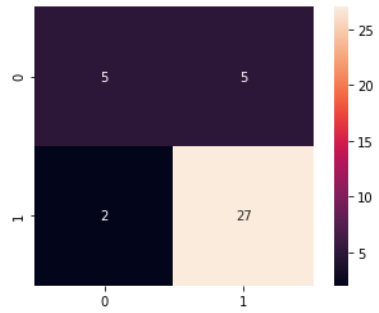
```
0.8205128205128205
```

```
cf_matrix=confusion_matrix(y_test , y_predicted)
sns.heatmap(cf_matrix, annot=True,square=True)
print(classification_report(y_test , y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.71      0.50      0.59        10
           1       0.84      0.93      0.89        29

    accuracy                           0.82        39
   macro avg       0.78      0.72      0.74        39
weighted avg       0.81      0.82      0.81        39
```

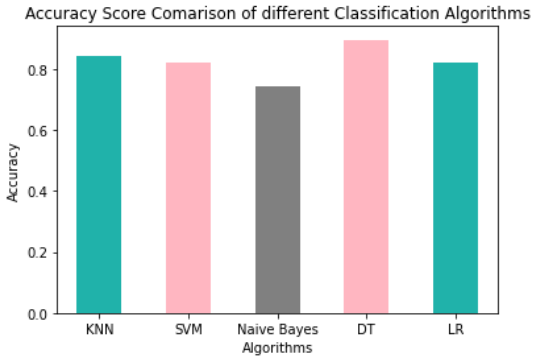


Comparison between Algorithms

Comparison

```
algos = ["KNN", "SVM", "Naive Bayes","DT","LR"]
scores = [knn_score, svm_score, nb_score,dt_score,lr_score]
```

```
plt.bar(algos, scores, width=0.5,color = ['lightseagreen', 'lightpink','grey','lightpink','lightseagreen'])
plt.title("Accuracy Score Comarison of different Classification Algorithms")
plt.xlabel('Algorithms')
plt.ylabel('Accuracy')
plt.show()
```



Conclusion

Proposed project is user-friendly, scalable, reliable and an expandable analysis which can also help in reducing treatment costs by providing initial diagnostics in time. The model can also serve the purpose of training tool for medical students and will be a soft diagnostic tool.

There are many possible improvements that could be explored to improve the scalability and accuracy of this prediction system. As we have developed a generalized system, in future we can use this system for the analysis of different data sets.

The performance of the diagnosis can be improved significantly by handling numerous class labels in the prediction process, and it can be another positive direction of research.