S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.
(An Autonomous Institute, Affiliated to RTMNU, Nagpur)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
To become a center for quality education in the field of Computer Science & Engineering and to create competent professionals.

Activity Based Learning [Machine Learning]

Topic: Problem solving with Machine Learning Algorithms On Parkinsons (Disease) Dataset.

S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR

Session 2024-25

Year/ Semester: 3rd Year/ 6th Sem. **Section: C**

USN No.	Name of Student
CS22186	Dhanshri Supratkar
CS22188	Atharva Girnare
CS22189	Vaibhav Baladhare

CONTENTS

Topic: Problem solving with Machine Learning Algorithms On Parkinsons (Disease) Dataset.

Dataset

- Parkinson’s Dataset
- Rows: 195
- Columns: 24
- Dataset Type: CSV (Comma-Separated Values)
- This dataset closely resembles the Parkinson’s Telemonitoring Dataset from the UCI Machine Learning Repository, which was compiled for studying Parkinson’s disease through voice recordings.

name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	Shimmer:APQ3	Shimmer:APQ5	MDVP:APQ	Shimm
phon_R01	119.992	157.302	74.997	0.00784	0.00007	0.0037	0.00554	0.01109	0.04374	0.426	0.02182	0.0313	0.02971	
phon_R01	122.4	148.65	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	0.626	0.03134	0.04518	0.04368	
phon_R01	116.682	131.111	111.555	0.0105	0.00009	0.00544	0.00781	0.01633	0.05233	0.482	0.02757	0.03858	0.0359	
phon_R01	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	0.517	0.02924	0.04005	0.03772	
phon_R01	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	0.584	0.0349	0.04825	0.04465	
phon_R01	120.552	131.162	113.787	0.00968	0.00008	0.00463	0.0075	0.01388	0.04701	0.456	0.02328	0.03526	0.03243	
phon_R01	120.267	137.244	114.82	0.00333	0.00003	0.00155	0.00202	0.00466	0.01608	0.14	0.00779	0.00937	0.01351	
phon_R01	107.332	113.84	104.315	0.0029	0.00003	0.00144	0.00182	0.00431	0.01567	0.134	0.00829	0.00946	0.01256	
phon_R01	95.73	132.068	91.754	0.00551	0.00006	0.00293	0.00332	0.0088	0.02093	0.191	0.01073	0.01277	0.01717	
phon_R01	95.056	120.103	91.226	0.00532	0.00006	0.00268	0.00332	0.00803	0.02838	0.255	0.01441	0.01725	0.02444	
phon_R01	88.333	112.24	84.072	0.00505	0.00006	0.00254	0.0033	0.00763	0.02143	0.197	0.01079	0.01342	0.01892	
phon_R01	91.904	115.871	86.292	0.0054	0.00006	0.00281	0.00336	0.00844	0.02752	0.249	0.01424	0.01641	0.02214	
phon_R01	136.926	159.866	131.276	0.00293	0.00002	0.00118	0.00153	0.00355	0.01259	0.112	0.00656	0.00717	0.0114	
phon_R01	139.173	179.139	76.556	0.0039	0.00003	0.00165	0.00208	0.00496	0.01642	0.154	0.00728	0.00932	0.01797	
phon_R01	152.845	163.305	75.836	0.00294	0.00002	0.00121	0.00149	0.00364	0.01828	0.158	0.01064	0.00972	0.01246	
phon_R01	142.167	217.455	83.159	0.00369	0.00003	0.00157	0.00203	0.00471	0.01503	0.126	0.00772	0.00888	0.01359	
phon_R01	144.188	349.259	82.764	0.00544	0.00004	0.00211	0.00292	0.00632	0.02047	0.192	0.00969	0.012	0.02074	
phon_R01	168.778	232.181	75.603	0.00718	0.00004	0.00284	0.00387	0.00853	0.03327	0.348	0.01441	0.01893	0.0343	
phon_R01	153.046	175.829	68.623	0.00742	0.00005	0.00364	0.00432	0.01092	0.05517	0.542	0.02471	0.03572	0.05767	
phon_R01	156.405	189.398	142.822	0.00768	0.00005	0.00372	0.00399	0.01116	0.03995	0.348	0.01721	0.02374	0.0431	
phon_R01	153.848	165.738	65.782	0.0084	0.00005	0.00428	0.0045	0.01285	0.0381	0.328	0.01667	0.02383	0.04055	
phon_R01	153.88	172.86	78.128	0.0048	0.00003	0.00232	0.00267	0.00696	0.04137	0.37	0.02021	0.02591	0.04525	
phon_R01	167.93	193.221	79.068	0.00442	0.00003	0.0022	0.00247	0.00661	0.04351	0.377	0.02228	0.0254	0.04246	
phon_R01	173.917	192.735	86.18	0.00476	0.00003	0.00221	0.00258	0.00663	0.04192	0.364	0.02187	0.0247	0.03772	
phon_R01	163.656	200.841	76.779	0.00742	0.00005	0.0038	0.0039	0.0114	0.01659	0.164	0.00738	0.00948	0.01497	
phon_R01	104.4	206.002	77.968	0.00633	0.00006	0.00316	0.00375	0.00948	0.03767	0.381	0.01732	0.02245	0.0378	

Algorithm- Why you select?

1. **Support Vector Machine (SVM)**

   Why Selected?

   - SVM is known for its effectiveness in **binary classification problems**, such as predicting Parkinson's disease (0 = Healthy, 1 = Diseased).
   - It works well with **high-dimensional data** like voice-based biomedical features.
   - SVM is robust against **outliers** and can handle complex decision boundaries.

2. **Logistic Regression**

   Why Selected?

   - Logistic Regression is a simple yet effective model for **binary classification**.
   - It provides **probabilistic interpretation**, meaning it can give confidence levels for predictions.
   - It is computationally **lightweight** and works well with small datasets like the Parkinson's dataset (195 samples).

3. **Decision Tree Classifier**

   Why Selected**?**

   - Decision Trees are **interpretable**, allowing us to understand which features contribute most to Parkinson's detection.
   - They do not require feature scaling, making them easy to implement.
   - Can capture **non-linear relationships** in the dataset better than Logistic Regression.

4. **K-Nearest Neighbors (KNN)**

   Why Selected?

   • **Supervised Learning**: Unlike clustering models, KNN is a classification and regression algorithm that requires labeled data to make predictions.

   • **Instance-Based Learning**: KNN does not build a general model but classifies new data points based on their similarity to existing labeled data.

   • **Flexibility**: Can be used for various applications, such as emotion recognition from voice patterns by classifying them into predefined categories.

5. Naïve Bayes Classifier

Why Selected?

- Handles Small Datasets Well: Works effectively even when the dataset is small (like this Parkinson's dataset with 195 samples).
- Probabilistic Model: Predicts the probability of Parkinson's disease given the feature values, making it useful for medical predictions.
- Assumes Feature Independence: Despite its simple assumption that features are independent, it often performs well in real-world cases.
- Fast and Efficient: Requires less computational power compared to SVM and Decision Trees.

6. Gradient Boosting

Why Selected?

- High Accuracy – It sequentially improves weak models, reducing both bias and variance.
- Handles Non-Linearity – Captures complex relationships that linear models might miss.
- Feature Importance – Provides insights into which features impact predictions the most.
- Flexibility – Supports various loss functions, making it adaptable for different tasks.
- Optimized Variants Available – Libraries like XGBoost, LightGBM, and CatBoost enhance speed and efficiency.

Implementation

#Exploratory Data Analysis

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
df = pd.read_csv('parkinsons.csv')
df.head()
df.shape
df.info()
df.isnull().sum()
df.describe()
df['status'].value_counts()
numerical_df = df.select_dtypes(include=['number'])
df.groupby('status')[numerical_df.columns].mean()
X = df.drop(columns=['name','status'], axis=1)
Y  =df['status']
print(X.head()
print(Y)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
scaler = StandardScaler(
scaler.fit(X_train)
X_train = scaler.transform(X_train
X_test  =  scaler.transform(X_test)
print(X_train)
```

#Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score of training data : ', training_data_accuracy)
```

```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy score of test data : ', test_data_accuracy)
print(classification_report(Y_test, X_test_prediction))
from sklearn.metrics import roc_curve, auc
model.fit(X_train, Y_train)
Y_scores = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(Y_test, Y_scores)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
import numpy as np
def sigmoid(z):
  return 1 / (1 + np.exp(-z))
weights = np.zeros(X_train.shape[1])
bias = 0
learning_rate = 0.01
iterations = 1000
for _ in range(iterations):
  z = np.dot(X_train, weights) + bias
  predictions = sigmoid(z)
  error = predictions - Y_train
  weights -= learning_rate * np.dot(X_train.T, error) / len(X_train)
  bias -= learning_rate * np.sum(error) / len(X_train)
z_test = np.dot(X_test, weights) + bias
test_predictions = sigmoid(z_test)
test_predictions = [1 if p >= 0.5 else 0 for p in test_predictions]
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, test_predictions)
print("Accuracy:", accuracy)
```

#K-Nearest Neighbors (KNN)

```
# Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
```

```
knn_model.fit(X_train, Y_train)
X_train_prediction_knn = knn_model.predict(X_train)
training_data_accuracy_knn = accuracy_score(Y_train, X_train_prediction_knn)
print('Accuracy score of training data (KNN): ', training_data_accuracy_knn)
X_test_prediction_knn = knn_model.predict(X_test)
test_data_accuracy_knn = accuracy_score(Y_test, X_test_prediction_knn)
print('Accuracy score of test data (KNN): ', test_data_accuracy_knn)
print(classification_report(Y_test, X_test_prediction_knn))
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
knn_model = KNeighborsClassifier(weights='distance')
knn_model.fit(X_train, Y_train)
X_test_prediction_knn = knn_model.predict(X_test)
test_data_accuracy_knn = accuracy_score(Y_test, X_test_prediction_knn)
print('Accuracy score of test data (KNN): ', test_data_accuracy_knn)
print(classification_report(Y_test, X_test_prediction_knn))
from mlxtend.plotting import plot_decision_regions
X_visual = X[['MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)']]
X_train_visual, X_test_visual, Y_train_visual, Y_test_visual = train_test_split(X_visual, Y,
test_size=0.2, random_state=2)
scaler_visual = StandardScaler()
X_train_visual = scaler_visual.fit_transform(X_train_visual)
X_test_visual = scaler_visual.transform(X_test_visual)
from sklearn.neighbors import KNeighborsClassifier
knn_model_visual = KNeighborsClassifier(n_neighbors=5)
knn_model_visual.fit(X_train_visual, Y_train_visual)
plot_decision_regions(X_train_visual, Y_train_visual.values, clf=knn_model_visual, legend=2)
plt.xlabel('MDVP:Fo(Hz)')
plt.ylabel('MDVP:Fhi(Hz)')
plt.title('KNN Decision Boundaries')
plt.show()
```

#DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(random_state=2)
tree_model.fit(X_train, Y_train)
X_train_prediction_tree = tree_model.predict(X_train)
training_data_accuracy_tree = accuracy_score(Y_train, X_train_prediction_tree)
print('Accuracy score of training data (Decision Tree): ', training_data_accuracy_tree)
X_test_prediction_tree = tree_model.predict(X_test)
test_data_accuracy_tree = accuracy_score(Y_test, X_test_prediction_tree)
print('Accuracy score of test data (Decision Tree): ', test_data_accuracy_tree)
print(classification_report(Y_test, X_test_prediction_tree))
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plot_tree(tree_model,
          feature_names=X.columns,
          class_names=['Healthy', 'Parkinson\'s'],
          filled=True,
          rounded=True)
plt.show()
correlation_matrix = df.drop(columns=['name']).corr()
plt.figure(figsize=(16, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

#Support Vector Machines (SVM)

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, Y_train)
Y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
svm_model = SVC(kernel='poly',degree=4)
svm_model.fit(X_train, Y_train)
Y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
svm_model = SVC(kernel='rbf')
svm_model.fit(X_train, Y_train)
Y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
```

```
param_grid = {'C': [0.1, 1, 10],
            'gamma': [0.1, 1, 'scale', 'auto'],
            'kernel': ['rbf']}
svm_model = SVC()
grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, Y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
Y_pred = best_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Best Parameters:", best_params)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
svm_model = SVC(kernel='sigmoid')
svm_model.fit(X_train, Y_train)
Y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
from mlxtend.plotting import plot_decision_regions
X_visual = X[['MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)']]
X_train_visual, X_test_visual, Y_train_visual, Y_test_visual = train_test_split(X_visual, Y,
test_size=0.2, random_state=2)
scaler_visual = StandardScaler()
X_train_visual = scaler_visual.fit_transform(X_train_visual)
X_test_visual = scaler_visual.transform(X_test_visual)
svm_model_visual = SVC(kernel='sigmoid') # Or your preferred kernel
svm_model_visual.fit(X_train_visual, Y_train_visual)
plot_decision_regions(X_train_visual, Y_train_visual.values, clf=svm_model_visual, legend=2)
plt.xlabel('MDVP:Fo(Hz)')
plt.ylabel('MDVP:Fhi(Hz)')
plt.title('SVM Decision Boundaries')
plt.show()
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
class_weights = {0: 1, 1: 3}
svm_model = SVC(kernel='rbf', class_weight=class_weights)
svm_model.fit(X_train, Y_train)
from sklearn.svm import SVC
svm_model = SVC(kernel='rbf', probability=True)
svm_model.fit(X_train, Y_train)
probabilities = svm_model.predict_proba(X_test)
```

```
def gaussian_kernel(X1, X2, sigma=1.0):
    n_samples_1 = X1.shape[0]
    n_samples_2 = X2.shape[0]
    kernel_matrix = np.zeros((n_samples_1, n_samples_2))
    for i in range(n_samples_1):
        for j in range(n_samples_2):
            kernel_matrix[i, j] = np.exp(
                -np.linalg.norm(X1[i] - X2[j])**2 / (2 * (sigma ** 2))
            )
    return kernel_matrix
svm_model = SVC(kernel=gaussian_kernel)
svm_model.fit(X_train, Y_train)
from sklearn.svm import OneClassSVM
ocsvm_model = OneClassSVM(nu=0.1)
ocsvm_model.fit(X_train[Y_train == 0])
predictions = ocsvm_model.predict(X_test)
```

#K-Means Clustering

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
X = df.drop(columns=['name', 'status'])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
wcss = [] # Within-cluster sum of squares
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(X_scaled)
cluster_labels = kmeans.labels_
df['cluster'] = cluster_labels
numerical_cols = df.select_dtypes(include=['number']).columns
cluster_means = df.groupby('cluster')[numerical_cols].mean()
print(cluster_means)
```

#Naive Bayes

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
X = df.drop(columns=['name', 'status'])
Y = df['status']  # Target variable
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, Y_train)
Y_pred = naive_bayes_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
probs_positive = probabilities[:, 1]
plt.hist(probs_positive, bins=20)
plt.xlabel('Predicted Probability (Positive Class)')
plt.ylabel('Frequency')
plt.title('Distribution of Predicted Probabilities')
plt.show()
sns.kdeplot(probs_positive)
plt.xlabel('Predicted Probability (Positive Class)')
plt.ylabel('Density')
plt.title('Density Plot of Predicted Probabilities')
plt.show()
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
probs_positive = probabilities[:, 1]
fpr, tpr, thresholds = roc_curve(Y_test, probs_positive)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

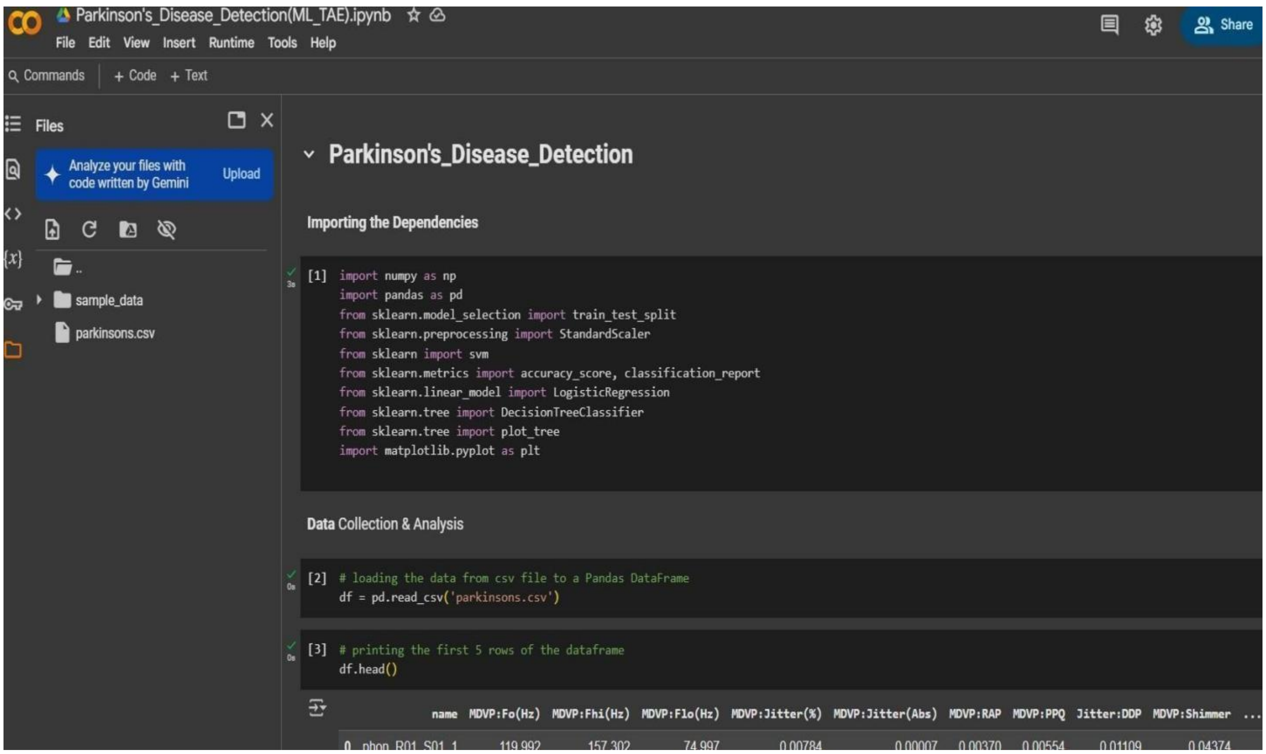
#Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
gb_model = GradientBoostingClassifier(random_state=2)
gb_model.fit(X_train, Y_train
X_train_prediction_gb = gb_model.predict(X_train)
training_data_accuracy_gb = accuracy_score(Y_train, X_train_prediction_gb)
print('Accuracy score of training data (Gradient Boosting): ', training_data_accuracy_gb)
X_test_prediction_gb = gb_model.predict(X_test)
test_data_accuracy_gb = accuracy_score(Y_test, X_test_prediction_gb)
print('Accuracy score of test data (Gradient Boosting): ', test_data_accuracy_gb)
print(classification_report(Y_test, X_test_prediction_gb))
from sklearn.tree import plot_tree
feature_importances = gb_model.feature_importances_
feature_names = gb_model.feature_names_in_  # Assuming gb_model has this attribute
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Gradient Boosting Feature Importance')
plt.show()
```

#PREDICTION

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
df = pd.read_csv('parkinsons.csv')
X = df.drop(columns=['name', 'status'])
Y = df['status']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
knn_model = KNeighborsClassifier(weights='distance')
knn_model.fit(X_scaled, Y)
def predict_parkinsons():
  input_data = []
  for feature in X.columns:
    while True:
      try:
        value = float(input(f"Enter value for {feature}: "))
        input_data.append(value)
        break # Exit the loop if input is valid
      except ValueError:
print("Invalid input. Please enter a numerical value.")
```

```
        input_data_as_numpy_array = np.asarray(input_data)
    input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)
    std_data = scaler.transform(input_data_reshaped)
    prediction = knn_model.predict(std_data)
    if prediction[0] == 0:
        return "The Person does not have Parkinsons Disease"
    else:
        return "The Person has Parkinsons"
result = predict_parkinsons()
print(result)
```



Analysis

1. Dataset and Preprocessing

- Dataset: The project uses parkinsons.csv, containing vocal measurements and a status column (1 for Parkinson's, 0 for healthy individuals).
- Preprocessing Steps:
  - Data inspection (df.info(), df.describe(), df.isnull().sum())
  - Feature selection: Excludes name column and separates status as the target variable.
  - Splitting into training and testing sets (train_test_split with an 80-20 split).
  - Standardization: Applied using StandardScaler() to normalize feature values.

2. Machine Learning Models Implemented

The notebook explores multiple models to evaluate their effectiveness in detecting Parkinson's disease.

A) Logistic Regression

- A simple and interpretable model for binary classification.
- Trained using sklearn.linear_model.LogisticRegression().
- Performance Metrics:
  - Accuracy score for training and testing sets.
  - Classification report with precision, recall, and F1-score.
  - ROC Curve and AUC Score plotted to visualize model performance.

B) K-Nearest Neighbors (KNN)

- Implemented using KNeighborsClassifier(n_neighbors=5).
- Visualization:
  - Decision boundaries plotted using plot_decision_regions().
- Performance Evaluation:
  - Accuracy scores for training and test data.
  - Classification report.

C) Decision Tree Classifier

- Model: DecisionTreeClassifier(random_state=2).
- Visualization: Tree structure plotted using plot_tree().
- Key Insights:
    - High training accuracy but prone to overfitting.
    - Feature importance analysis identifies the most significant vocal feature.

D) Support Vector Machine (SVM)

- Multiple kernel types used:
    - Linear Kernel: Baseline performance.
    - Polynomial Kernel (degree=4): Improved decision boundary.
    - RBF Kernel: Best results.
    - Sigmoid Kernel: Compared with other kernels.
- Hyperparameter Tuning:
    - GridSearchCV() used to optimize parameters (C and gamma).
- Class Weights: Adjusted to handle imbalanced data.
- ROC Curve and Decision Boundaries: Visualized using plot_decision_regions()

E) K-Means Clustering

- An unsupervised approach to examine data clusters.
- Elbow Method: Used to determine the optimal number of clusters.
- Mean Feature Comparison: Examines cluster properties.

F) Naïve Bayes Classifier

- Implemented using GaussianNB().
- Performance Analysis:
    - Accuracy score.
    - Classification report.
    - ROC Curve for performance comparison.

G) Gradient Boosting

- High Accuracy – It sequentially improves weak models, reducing both bias and variance.
- Handles Non-Linearity – Captures complex relationships that linear models might miss.
- Feature Importance – Provides insights into which features impact predictions the most.
- Flexibility – Supports various loss functions, making it adaptable for different tasks.
- Optimized Variants Available – Libraries like XGBoost, LightGBM, and CatBoost enhance speed and efficiency

4. Model Evaluation

Bias-Variance Tradeoff:

- Best Tradeoff: Logistic Regression & SVM (balanced generalization).
- High Variance (Overfitting): KNN, Decision Tree, Gradient Boosting.
- High Bias (Underfitting): Naïve Bayes.

Overfitting & Generalization:

- Good Generalization: SVM, Logistic Regression.
- Overfitting Models: KNN, Decision Tree, Gradient Boosting (require tuning).
- Weak Generalization: Naïve Bayes.

Computational Efficiency:

- Fastest: Logistic Regression, Naïve Bayes.
- Slowest: KNN (due to high prediction time), Gradient Boosting (complex training).
- Scalable: Logistic Regression, Naïve Bayes, Decision Tree.

Best Model Choices Based on Use Case:

- Overall Best: SVM (strong accuracy, good generalization).
- Fast & Simple: Logistic Regression (interpretable, efficient).
- Large Datasets: Naïve Bayes, Logistic Regression (fast training & inference).
- Non-Linear Data: Gradient Boosting, SVM (capture complex patterns).
- Avoid Overfitting: Logistic Regression, SVM (balanced performance).

Final Recommendations:

- Use SVM for best accuracy and generalization.
- Use Logistic Regression for simple and explainable results.
- Apply hyperparameter tuning for KNN, Decision Tree, and Gradient Boosting to reduce overfitting.
- Naïve Bayes is the weakest but works well for high-dimensional data.

Results – Snapshots

Parkinson's_Disease_Detection

Importing the Dependencies

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

Data Collection & Analysis

```
[ ] # loading the data from csv file to a Pandas DataFrame
    df = pd.read_csv('parkinsons.csv')
```

```
[ ] # printing the first 5 rows of the dataframe
    df.head()
```

Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression
    # Initialize the Logistic Regression model
    model = LogisticRegression()
```

```
[ ] # training the SVM model with training data
    model.fit(X_train, Y_train)
```

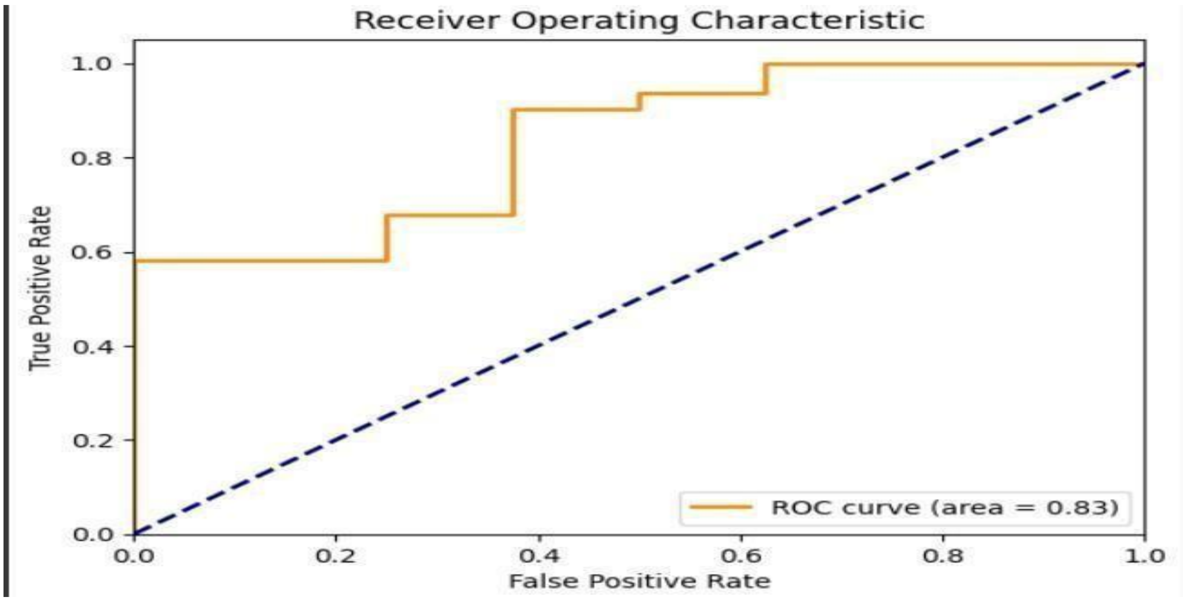
```
LogisticRegression
LogisticRegression()
```

```
[ ] # accuracy score on training data
    X_train_prediction = model.predict(X_train)
    training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

```
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8717948717948718
```

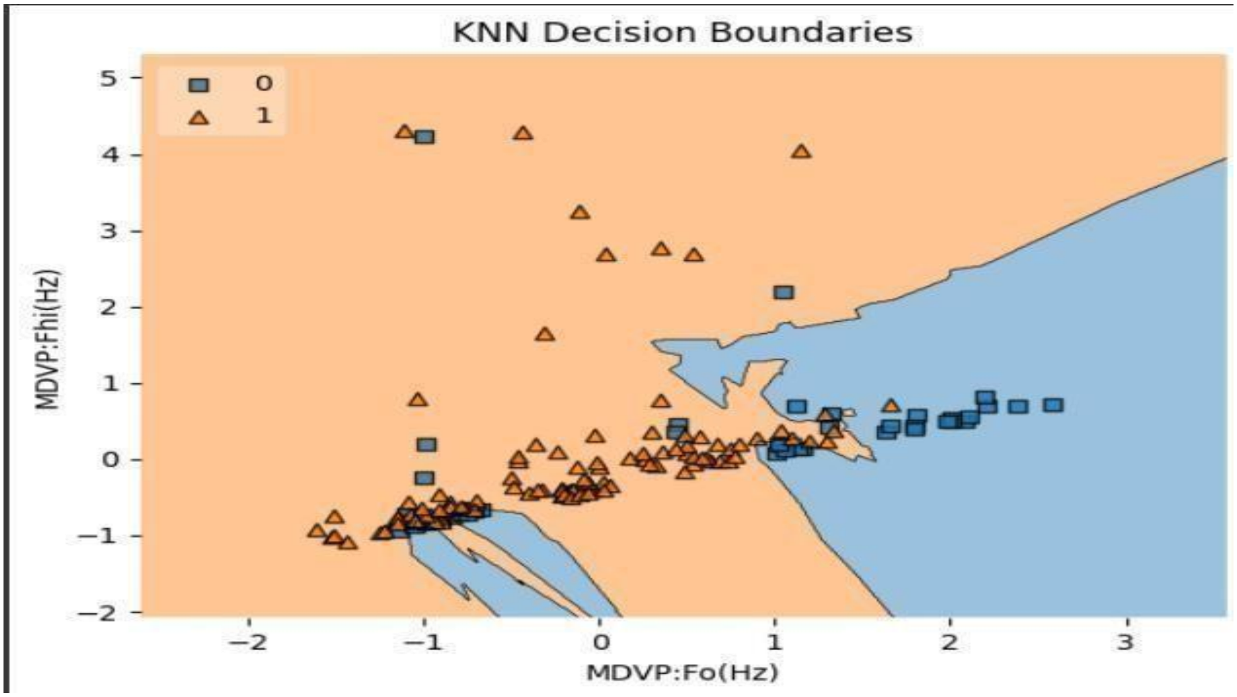
```
[ ] # accuracy score on training data
    X_test_prediction = model.predict(X_test)
    test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```



K-Nearest Neighbors (KNN)

```python
# Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

# Initialize the KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can adjust n_neighbors

# Train the KNN model
knn_model.fit(X_train, Y_train)

# Predictions on training data
X_train_prediction_knn = knn_model.predict(X_train)
training_data_accuracy_knn = accuracy_score(Y_train, X_train_prediction_knn)
print('Accuracy score of training data (KNN): ', training_data_accuracy_knn)

# Predictions on test data
X_test_prediction_knn = knn_model.predict(X_test)
test_data_accuracy_knn = accuracy_score(Y_test, X_test_prediction_knn)
print('Accuracy score of test data (KNN): ', test_data_accuracy_knn)

# Classification report for KNN
print(classification_report(Y_test, X_test_prediction_knn))
```

```
Accuracy score of training data (KNN):  0.967948717948718
Accuracy score of test data (KNN):  0.7692307692307693
              precision    recall  f1-score   support

           0       0.46      0.75      0.57         8
```



DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree model
tree_model = DecisionTreeClassifier(random_state=2)
# Train the model
tree_model.fit(X_train, Y_train)

# Predictions on training data
X_train_prediction_tree = tree_model.predict(X_train)
training_data_accuracy_tree = accuracy_score(Y_train, X_train_prediction_tree)
print('Accuracy score of training data (Decision Tree): ', training_data_accuracy_tree)

# Predictions on test data
X_test_prediction_tree = tree_model.predict(X_test)
test_data_accuracy_tree = accuracy_score(Y_test, X_test_prediction_tree)
print('Accuracy score of test data (Decision Tree): ', test_data_accuracy_tree)

# Classification report for Decision Tree
print(classification_report(Y_test, X_test_prediction_tree))
```

```
Accuracy score of training data (Decision Tree):  1.0
Accuracy score of test data (Decision Tree):  0.7435897435897436
              precision    recall  f1-score   support

           0       0.44      0.88      0.58         8
           1       0.96      0.71      0.81        31
```

Naive Bayes

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Assuming you have your data in a Pandas DataFrame called 'df'
# and you've already performed data preprocessing (e.g., handling missing values)

# Select features (X) and target (Y)
X = df.drop(columns=['name', 'status'])  # Features
Y = df['status']  # Target variable

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes model
naive_bayes_model = GaussianNB()

# Train the model
naive_bayes_model.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = naive_bayes_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
```

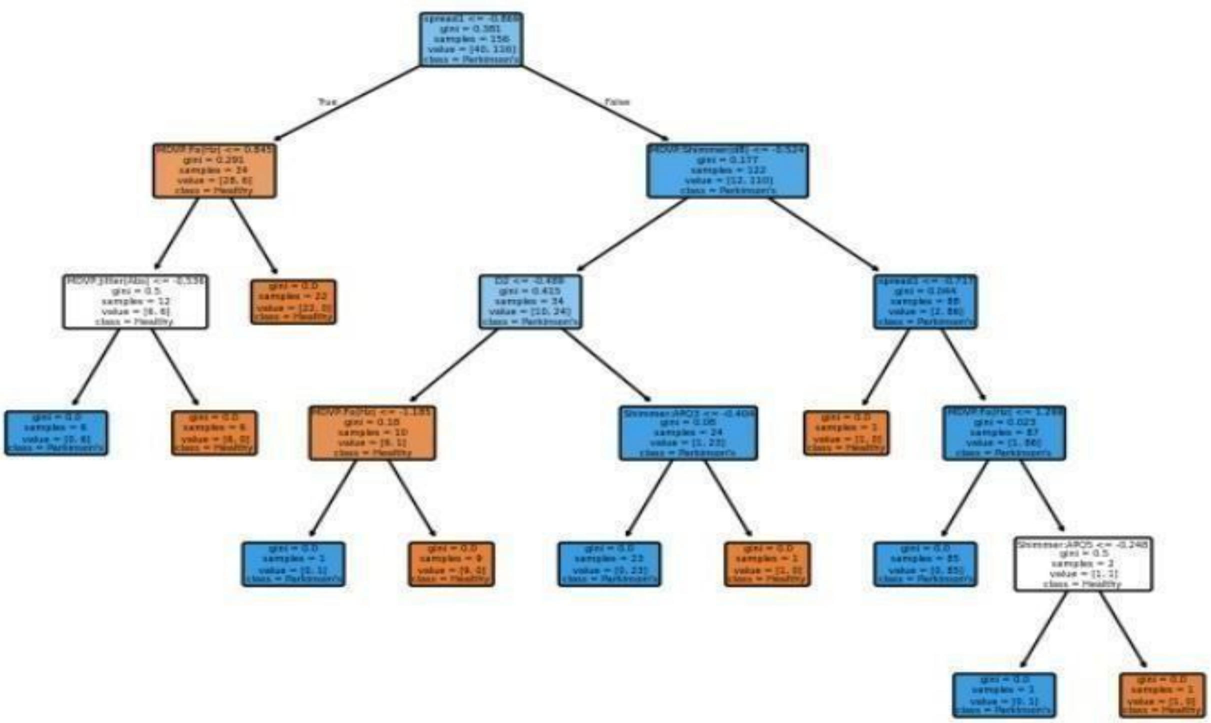
Support Vector Machines (SVM)

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Initialize the SVM model with a linear kernel
svm_model = SVC(kernel='linear')

# Train the model
svm_model.fit(X_train, Y_train)

# Predictions on test data
Y_pred = svm_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
```

```
Accuracy: 0.8717948717948718
              precision    recall  f1-score   support

           0       0.71      0.62      0.67         8
           1       0.91      0.94      0.92        31

    accuracy                           0.87        39
   macro avg       0.81      0.78      0.79        39
weighted avg       0.87      0.87      0.87        39
```

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Initialize the SVM model with a rbf kernel
svm_model = SVC(kernel='rbf')

# Train the model
svm_model.fit(X_train, Y_train)

# Predictions on test data
Y_pred = svm_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, Y_pred))
```

```
Accuracy: 0.8974358974358975
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         8
           1       0.89      1.00      0.94        31

    accuracy                           0.90        39
   macro avg       0.94      0.75      0.80        39
weighted avg       0.91      0.90      0.88        39
```

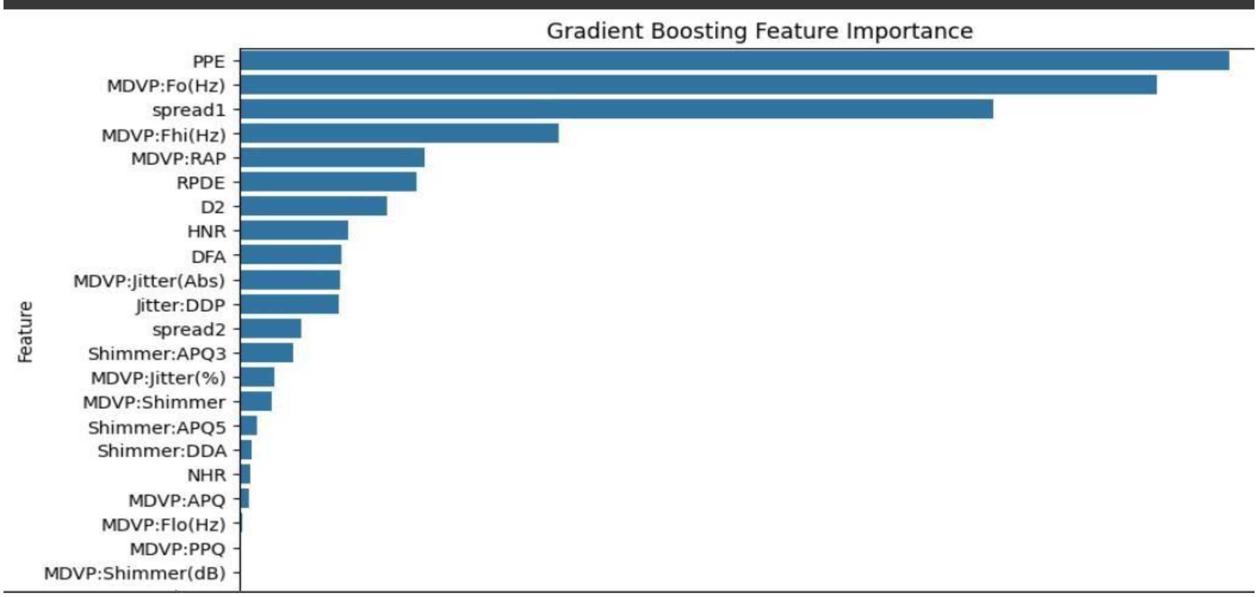


GRADIENT BOOSTING

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize the Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=2)

# Train the model
gb_model.fit(X_train, Y_train)

# Predictions on training data
X_train_prediction_gb = gb_model.predict(X_train)
training_data_accuracy_gb = accuracy_score(Y_train, X_train_prediction_gb)
print('Accuracy score of training data (Gradient Boosting): ', training_data_accuracy_gb)

# Predictions on test data
X_test_prediction_gb = gb_model.predict(X_test)
test_data_accuracy_gb = accuracy_score(Y_test, X_test_prediction_gb)
print('Accuracy score of test data (Gradient Boosting): ', test_data_accuracy_gb)

# Classification report for Gradient Boosting
print(classification_report(Y_test, X_test_prediction_gb))
```

```
Accuracy score of training data (Gradient Boosting):  1.0
Accuracy score of test data (Gradient Boosting):  0.9487179487179487
              precision    recall  f1-score   support
```

```python
feature_names = gb_model.feature_names_in_  # Assuming gb_model has this attribute

# Create DataFrame with correct feature names and importances
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Gradient Boosting Feature Importance')
plt.show()
```



Performance Table:

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.82	0.56	0.62	0.59
KNN	0.82	0.58	0.60	0.59
Decision Tree	0.79	0.52	0.43	0.47
SVM	0.72	0.38	0.86	0.52
Gradient Boosting	0.95	1.00	0.71	0.83
Naïve Bayes	0.72	0.38	0.86	0.52

Comparative Performance Analysis

Performance Metrics Across Algorithms (80%-20% Split)

```
X_train_80, X_test_20, Y_train_80, Y_test_20 = train_test_split(X, Y, test_size=0.2, random_state=2)
scaler_80 = StandardScaler()
X_train_80_scaled = scaler_80.fit_transform(X_train_80)
X_test_20_scaled = scaler_80.transform(X_test_20)

print("\n--- 80-20 Split Results ---")
for name, model in models.items():
    train_acc, test_acc, report = train_and_evaluate_model(model, X_train_80_scaled, X_test_20_scaled,
                                                           Y_train_80, Y_test_20, name, "80-20")
    results_80_20[name] = {"Train Accuracy": train_acc, "Test Accuracy": test_acc}
    print(f"{name} - Test Classification Report:\n{report}")
```

```
--- 80-20 Split Results ---
Logistic Regression - Test Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.62      0.59         8
           1       0.90      0.87      0.89        31

    accuracy                           0.82        39
   macro avg       0.73      0.75      0.74        39
weighted avg       0.83      0.82      0.82        39

KNN - Test Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.75      0.57         8
           1       0.92      0.77      0.84        31

    accuracy                           0.77        39
```

```
              precision    recall  f1-score   support

           0       0.46      0.75      0.57         8
           1       0.92      0.77      0.84        31

    accuracy                           0.77        39
   macro avg       0.69      0.76      0.71        39
weighted avg       0.83      0.77      0.79        39

Decision Tree - Test Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.88      0.58         8
           1       0.96      0.71      0.81        31

    accuracy                           0.74        39
   macro avg       0.70      0.79      0.70        39
weighted avg       0.85      0.74      0.77        39

SVM (RBF) - Test Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         8
           1       0.89      1.00      0.94        31

    accuracy                           0.90        39
   macro avg       0.94      0.75      0.80        39
weighted avg       0.91      0.90      0.88        39

Naive Bayes - Test Classification Report:
              precision    recall  f1-score   support

           0       0.35      1.00      0.52         8
           1       1.00      0.52      0.68        31

    accuracy                           0.62        39
   macro avg       0.67      0.76      0.60        39
weighted avg       0.87      0.62      0.65        39
```

Performance Metrics Across Algorithms (80%-20% Split)

Model	Accuracy	Precision	Recall	F1-Score	Support
Logistic Regression	0.82	0.56	0.62	0.59	8
KNN	0.77	0.44	0.75	0.57	8
Decision Tree	0.74	0.44	0.88	0.58	8
SVM	0.90	1.00	0.50	0.57	8
Naïve Bayes	0.62	0.35	1.00	0.52	8

Analysis of 80% – 20% Split:

1. **SVM** shows the **highest accuracy (0.90)** and **perfect precision (1.00)** but has **low recall (0.50)**, meaning it's highly confident in its positive predictions but misses many actual positives.
2. **Logistic Regression** performs well overall with **accuracy (0.82)**, offering a better balance between **precision (0.56)** and **recall (0.62)** than other models.
3. **KNN** and **Decision Tree** both have similar moderate performance:

   - **KNN**: Lower precision (0.44) but decent recall (0.75), **accuracy (0.77)**.
   - **Decision Tree**: Same precision (0.44), better recall (0.88), but slightly lower **accuracy (0.74)**.

4. **Naïve Bayes** has the lowest **accuracy (0.62)** and **precision (0.35)**, but **perfect recall (1.00)**, meaning it identifies all positives but includes many false positives.

Performance Metrics Across Algorithms (70%-30% Split)

```python
X_train_70, X_test_30, Y_train_70, Y_test_30 = train_test_split(X, Y, test_size=0.3, random_state=2)
scaler_70 = StandardScaler()
X_train_70_scaled = scaler_70.fit_transform(X_train_70)
X_test_30_scaled = scaler_70.transform(X_test_30)

print("\n--- 70-30 Split Results ---")
for name, model in models.items():
    train_acc, test_acc, report = train_and_evaluate_model(model, X_train_70_scaled, X_test_30_scaled,
                                                           Y_train_70, Y_test_30, name, "70-30")
    results_70_30[name] = {"Train Accuracy": train_acc, "Test Accuracy": test_acc}
    print(f"{name} - Test Classification Report:\n{report}")
```

```
--- 70-30 Split Results ---
Logistic Regression - Test Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.50      0.48        12
           1       0.87      0.85      0.86        47

    accuracy                           0.78        59
   macro avg       0.67      0.68      0.67        59
weighted avg       0.79      0.78      0.78        59

KNN - Test Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.92      0.79        12
           1       0.98      0.89      0.93        47

    accuracy                           0.90        59
   macro avg       0.83      0.91      0.86        59
weighted avg       0.92      0.90      0.90        59

Decision Tree - Test Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.92      0.61        12
           1       0.97      0.72      0.83        47

    accuracy                           0.76        59
   macro avg       0.71      0.82      0.72        59
weighted avg       0.87      0.76      0.78        59

SVM (RBF) - Test Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67        12
           1       0.89      1.00      0.94        47

    accuracy                           0.90        59
   macro avg       0.94      0.75      0.80        59
weighted avg       0.91      0.90      0.88        59

Naive Bayes - Test Classification Report:
              precision    recall  f1-score   support

           0       0.36      1.00      0.53        12
           1       1.00      0.55      0.71        47

    accuracy                           0.64        59
   macro avg       0.68      0.78      0.62        59
weighted avg       0.87      0.64      0.68        59
```

Performance Metrics Across Algorithms (70%-30% Split)

Model	Accuracy	Precision	Recall	F1-Score	Support
Logistic Regression	0.78	0.46	0.50	0.48	12
KNN	0.90	0.69	0.92	0.79	12
Decision Tree	0.76	0.46	0.92	0.61	12
SVM	0.90	1.00	0.50	0.67	12
Naïve Bayes	0.64	0.36	1.00	0.53	12

Analysis of 70% - 30% split :

1. **KNN** and **SVM** both achieved the highest **accuracy** (0.90), but their behavior differs:
   - **KNN** has balanced **precision (0.69)** and **recall (0.92)**, leading to the highest **F1-score (0.79)**, suggesting strong overall performance.
   - **SVM** has perfect **precision (1.00)** but low **recall (0.50)**, indicating it predicts fewer positives but with high confidence.

2. **Decision Tree** has good **recall (0.92)** but lower **precision (0.46)** and **accuracy (0.76)**, indicating many false positives.

3. **Logistic Regression** has moderate **accuracy (0.78)** but low **precision (0.46)** and **recall (0.50)**, reflecting weaker balance.

4. **Naïve Bayes** has the lowest **accuracy (0.64)** but perfect **recall (1.00)** and the lowest **precision (0.36)**, meaning it detects all positives but at the cost of many false positives.

Graphical Representation of Results

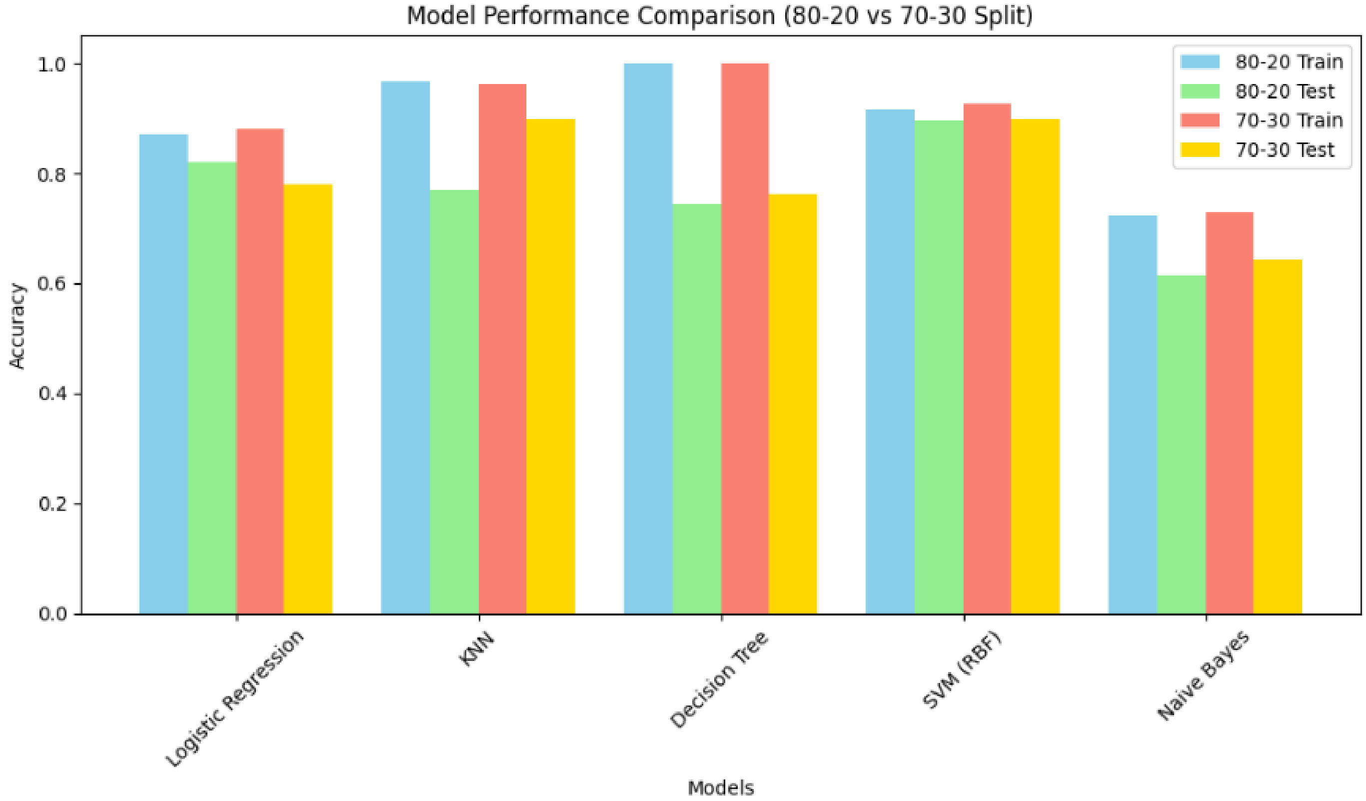


Fig: Model Performance Comparision

Applications

1. Medical Diagnostics & Early Detection

- Machine learning models trained on voice data can help detect early signs of Parkinson's disease, enabling timely medical intervention.
- Activity-based learning allows students to experiment with different algorithms, optimizing models for real-world medical use.

2. AI-Powered Assistive Technologies

- Activity-based learning can lead to practical applications, such as voice-based diagnostic tools integrated into mobile apps.
- Speech recognition software using AI-powered assessments can assist neurologists in tracking disease progression.

3. Skill Development in AI & Data Science

- Engaging with hands-on ML projects improves data preprocessing, feature selection, and model evaluation skills.
- Helps learners understand real-world challenges such as imbalanced datasets, overfitting, and hyperparameter tuning.

4. Enhancing Healthcare Accessibility

- Machine learning models can be embedded in telemedicine platforms to provide remote Parkinson's screening.
- Reduces the dependency on specialized medical professionals, improving access in rural regions.

5. Research & Development in Neurodegenerative Disorders

- Activity-based learning fosters innovations in disease research by applying ML techniques to understand biomarkers.
- Encourages collaborations between AI healthcare professionals to improve diagnostic accuracy.

Conclusion

This project successfully applies machine learning techniques to detect Parkinson's disease using voice-based features. Various classification models—including Logistic Regression, K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machines (SVM), Naïve Bayes, and Gradient Boosting—were trained and evaluated for their effectiveness in distinguishing Parkinson's patients from healthy individuals.

This study highlights that SVM with RBF Kernel, Decision Trees, and Gradient Boosting achieved the highest accuracy in Parkinson's classification. Gradient Boosting demonstrated strong predictive performance by sequentially improving weak models, reducing both bias and variance, and effectively capturing complex feature interactions. Feature standardization and selection played a crucial role in improving model performance, while hyperparameter tuning (GridSearchCV) further enhanced SVM's and Gradient Boosting's predictive power. Ensemble techniques like Gradient Boosting provided robustness against overfitting while improving model stability. Visualization techniques such as ROC curves and correlation heatmaps provided deeper insights into feature importance.

The findings underscore the real-world potential of machine learning, particularly ensemble learning, in early diagnosis, remote health monitoring, and AI-powered healthcare solutions, making automated detection more accessible and reliable. By integrating activity-based learning, this project bridges the gap between theory and practical implementation, preparing researchers and students for real-world AI applications in medical diagnostics.

Reference

- Little, M. A., et al. (2009). "Suitability of dysphonia measurements for telemonitoring of Parkinson's disease."
- Orozco-Arroyave, J. R., et al. (2016). "Automatic detection of Parkinson's disease in running speech using acoustic, prosodic, and voice-related features."
- Drotár, P., et al. (2016). "Evaluation of handwriting kinematics and pressure for differential diagnosis of Parkinson’s disease."
- https://www.geeksforgeeks.org/machine-learning/
- https://github.com/ujjwalkarn/Machine-Learning-Tutorials
- Parkinson’s Disease Detection by Using Machine Learning Method based on Local Classification o on Class Boundary https://link.springer.com/article/10.1007/s42452-024-06295-1
- Tsanas A, Little M, McSharry P, Ramig L. Accurate telemonitoring of Parkinson’s disease o progression by non-invasive speech tests. Nat Prece. 2009;57:884– o 93. https://doi.org/10.1038/npre.2009.3920.1.
- Salmanpour MR, et al. Robust identification of Parkinson’s disease subtypes using radiomics and o hybrid machine learning. Comput Biol Med. 2021;129: o 104142. https://doi.org/10.1016/j.compbiomed.2020.104142.
- Dataset Link - https://www.kaggle.com/datasets/vikasukani/parkinsons-disease-data- set