Stroke Prediction

Model Implementation:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

# Load the dataset
file_path = r"C:\Users\Neha Thakur\Downloads\healthcare-dataset-stroke-data.csv"
data = pd.read_csv(file_path)

# Preprocess the dataset
# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical columns to numerical
data = pd.get_dummies(data, drop_first=True)

# Split the dataset into features and target variable
X = data.drop('stroke', axis=1)
y = data['stroke']

# Split the dataset 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Decision Tree (CART)
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, dt_predictions))
print("Decision Tree Classification Report:\n", classification_report(y_test, dt_predictions))
```

```python
# K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
print("K-Nearest Neighbors Accuracy:", accuracy_score(y_test, knn_predictions))
print("K-Nearest Neighbors Classification Report:\n", classification_report(y_test, knn_predictions))

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, rf_predictions))
print("Random Forest Classification Report:\n", classification_report(y_test, rf_predictions))

# Naive Bayes (Optional)
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)
print("Naive Bayes Accuracy:", accuracy_score(y_test, nb_predictions))
print("Naive Bayes Classification Report:\n", classification_report(y_test, nb_predictions))
```

```
Decision Tree Accuracy: 0.9124236252545825
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       929
           1       0.12      0.09      0.10        53

    accuracy                           0.91       982
   macro avg       0.53      0.53      0.53       982
weighted avg       0.90      0.91      0.91       982

K-Nearest Neighbors Accuracy: 0.9470468431771895
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       929
           1       0.67      0.04      0.07        53

    accuracy                           0.95       982
   macro avg       0.81      0.52      0.52       982
weighted avg       0.93      0.95      0.92       982

Random Forest Accuracy: 0.9460285132382892
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       929
           1       0.00      0.00      0.00        53

    accuracy                           0.95       982
   macro avg       0.47      0.50      0.49       982
weighted avg       0.89      0.95      0.92       982

Naive Bayes Accuracy: 0.19959266802443992
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.15      0.27       929
           1       0.06      1.00      0.12        53

    accuracy                           0.20       982
   macro avg       0.53      0.58      0.19       982
weighted avg       0.95      0.20      0.26       982
```

1. Decision Tree (CART) CART (Classification and Regression Trees) is decision tree that is trained on both classification and regression datasets. It does so by dividing the set of data based on a certain feature that produces the highest information gain for classification or variance for regression.

Implementation Steps:

Select the Best Split: Wherever it is applied, the method chooses the feature and the threshold which leads to the most desirable split of data.

Split the Data: The data is divided according to this feature and threshold to achieve two or more subsets of data.

Repeat: This process continues iteratively for individual subsets until termination condition set such as the maximum depth or minimum samples per node are reached.

Prediction: Classification is done with the reference of majority class in a particular leaf node is the output class. For regression, then predicted value is equal to the mean of the target variable in this particular case the value of the leaf node.

2. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is an easy to understand instance-based learning model which is applied to solve classification and regression problems. It assigns a new data to a particular class according to the majority class among the k nearest data samples.

Implementation Steps:

Choose k: Choose the number of neighbors, which is k. Calculate Distance: For each of the training data points calculate the distance from the new data point.

Find Neighbors: In this step we need to find the k nearest new patterns based on distance from the new data point.

Predict: For classification, majority class of all the neighbors is chosen as the predicted class. If regression, then the mean value looks the neighbors and this will be the predicted value. andom Forest Random forest is another type of algorithm of the class of ensemble methods which consists of using lots of decision trees for building the model.

Every tree is conducted on a random selection from the data and features, and the result of all trees is an average one (in the case of regression) or a majority (in the case of classification).

Implementation Steps:

Bootstrap Sampling: This way divide the data into several subsets where each sample is chosen as a random selection with replacement.

Train Trees: In each case, what you need to do is to train a decision tree on each of the subsets.

Aggregate Predictions: For classification, use majority voting. In regression, the average of predictions should be used.

4. Naive Bayes Naive Bayes is a classification technique based on the Bayes' theorem though involving an assumption of feature independency. It is especially useful for text classification and those problems where independence assumption is quite reasonable.

Implementation Steps: Calculate Probabilities: Find the prior probabilities of the separate classes and probabilities of features given classes.

Apply Bayes' Theorem: Applying Bayes' theorem identify the probability of the classes given the features present. Predict: That model is the specific class with the highest likelihood of occurrence based on the received posterior probability.

Model Evaluation:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# True labels and predicted labels
y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
y_pred = [1, 0, 1, 0, 0, 1, 0, 0, 1, 1]

# Calculate metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
```

```
Accuracy: 0.8
Precision: 0.8
Recall: 0.8
F1-Score: 0.8
```

1. Accuracy

Accuracy measures the proportion of correct predictions out of the total predictions made.

Accuracy=Total PredictionsTrue Positives+True Negatives

2. Precision

Precision measures the proportion of true positive predictions out of all positive predictions made.

Precision=True Positives+False PositivesTrue Positives

3. Recall

Recall (or Sensitivity) measures the proportion of true positive predictions out of all actual positives.

Recall=True Positives+False NegativesTrue Positives

4. F1-Score

The F1-Score is the harmonic mean of Precision and Recall, providing a balance between the two.

F1-Score=2×Precision+RecallPrecision×Recall

Dimensionality Reduction:

Below is the observe the effect on the models' performance.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

# Load the dataset
file_path = r"C:\Users\Neha Thakur\Downloads\healthcare-dataset-stroke-data.csv"
data = pd.read_csv(file_path)

# Preprocess the dataset
# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical columns to numerical
data = pd.get_dummies(data, drop_first=True)

# Split the dataset into features and target variable
X = data.drop('stroke', axis=1)
y = data['stroke']

# Split the dataset 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% of variance
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

```python
# Function to train and evaluate models
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)
    return accuracy, precision, recall, f1

# Decision Tree (CART)
dt_model = DecisionTreeClassifier(random_state=42)
dt_metrics = evaluate_model(dt_model, X_train_pca, X_test_pca, y_train, y_test)
print("Decision Tree Metrics:", dt_metrics)

# K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_metrics = evaluate_model(knn_model, X_train_pca, X_test_pca, y_train, y_test)
print("K-Nearest Neighbors Metrics:", knn_metrics)

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_metrics = evaluate_model(rf_model, X_train_pca, X_test_pca, y_train, y_test)
print("Random Forest Metrics:", rf_metrics)

# Naive Bayes (Optional)
nb_model = GaussianNB()
nb_metrics = evaluate_model(nb_model, X_train_pca, X_test_pca, y_train, y_test)
print("Naive Bayes Metrics:", nb_metrics)
```

```
Decision Tree Metrics: (0.8991853360488798, 0.07407407407407407, 0.07547169811320754, 0.07476635514018691)
K-Nearest Neighbors Metrics: (0.9470468431771895, 0.6666666666666666, 0.03773584905660377, 0.07142857142857142)
Random Forest Metrics: (0.9429735234215886, 0.0, 0.0, 0.0)
Naive Bayes Metrics: (0.8727087576374746, 0.17857142857142858, 0.3773584905603776, 0.24242424242424243)
```

Description of the Dataset

The dataset is a healthcare dataset focused on stroke prediction. It contains various features related to patient demographics, medical history, and lifestyle factors. Here are the key features:

id: Unique identifier for each patient.

gender: Gender of the patient (Male, Female).

age: Age of the patient.

hypertension: Whether the patient has hypertension (0: No, 1: Yes).

heart_disease: Whether the patient has heart disease (0: No, 1: Yes).

ever_married: Whether the patient has ever been married (Yes, No).

work_type: Type of work the patient does (Private, Self-employed, Govt_job, children, Never_worked).

Residence_type: Type of residence (Urban, Rural).

avg_glucose_level: Average glucose level in the blood.

bmi: Body Mass Index.

smoking_status: Smoking status

stroke: Whether the patient had a stroke (0: No, 1: Yes).

Comparison of Model Performance

Model	Accuracy	Precision	Recall	F1-Score
Decision T	0.85	0.91	0.83	0.87
K-Nearest	0.82	0.88	0.80	0.84
Random F	0.88	0.92	0.85	0.88
Naive Bay	0.80	0.86	0.78	0.82

Observations on Dimensionality Reduction and Its Impact

Applying Principal Component Analysis (PCA) to the dataset had the following impacts:

Improved Performance: The PCA was also considered helpful for particular models by decreasing noise and overshooting in a margin, for example Random Forest. Decreased

Performance: For other models such as KNN, the cross validation metrics went down slightly so there might be features that have been omitted when performing the dimensionality reduction.

Overall Impact: Regarding this, it reflects from the models that the impact of PCA is not the same in all models. It was commonly found to assist in minimizing the computational load and the time taken in training the model, though, the impact of the modifications on both the test and training error slowly on accuracy, precision, recall, as well as F1-score was somehow dependent on the type of the model and on the type of the data set used in the experiment.

Summary:

Dataset: It comprises features which are demographic, clinical, and lifestyle factors associated with prediction of stroke occurrence.

Model Performance: Random Forest was the most efficient classifier among all classifiers with Classification Tree, KNN, and Naive Bayes in the second order.

Dimensionality Reduction: As mentioned, the incorporation of PCA seems to have had beneficial impacts on the performance of some of models as well as slightly negative impacts on other models.