Classification techniques are logistic regression, decision tree, support vector machine

Classification Techniques:

1. **Logistic Regression:**
   - Logistic regression is a widely used classification algorithm that models the probability of a binary outcome based on one or more predictor variables.
   - It works by fitting a logistic curve to the data, which allows for predicting the probability of the outcome belonging to a certain class.
2. **Decision Trees:**
   - Decision trees are versatile classification algorithms that partition the feature space into regions, assigning each observation to a specific class based on the majority class within the region.
   - They are intuitive to interpret and can handle both numerical and categorical data.
3. **Support Vector Machines (SVM):**
   - SVM is a powerful classification algorithm that finds the hyperplane that best separates the classes in the feature space.
   - It works by maximizing the margin between the classes, making it robust to overfitting and effective in high-dimensional spaces.

Regression Techniques:

1. **Linear Regression:**
   - Linear regression is one of the simplest and most widely used regression techniques that models the relationship between a dependent variable and one or more independent variables.
   - It assumes a linear relationship between the independent and dependent variables.
2. **Random Forest Regression:**
   - Random forest regression is an ensemble learning method that combines multiple decision trees to improve prediction accuracy.
   - It works by averaging the predictions of individual decision trees, reducing overfitting and improving generalization.
3. **Gradient Boosting Regression:**
   - Gradient boosting regression is another ensemble learning technique that builds a series of weak learners sequentially, with each new learner correcting the errors made by the previous ones.
   - It achieves high accuracy by iteratively minimizing a loss function.

Clustering Techniques:

1. **K-Means Clustering:**
   - K-means clustering is a popular unsupervised learning algorithm that partitions data into 'k' clusters based on similarity.
   - It works by iteratively assigning data points to the nearest cluster centroid and updating the centroids based on the mean of the assigned points.
2. **Hierarchical Clustering:**
   - Hierarchical clustering is a method that builds a hierarchy of clusters by either agglomerative (bottom-up) or divisive (top-down) approaches.
   - It does not require the specification of the number of clusters beforehand and provides a dendrogram for visualizing cluster relationships.
3. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):**
   - DBSCAN is a density-based clustering algorithm that groups together points that are closely packed together, forming dense regions separated by sparser areas.
   - It can identify clusters of arbitrary shapes and is robust to noise and outliers.

Codes:

Logistic Regression for Classification:

Logistic Regression for Classification:

python

Copy code

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd
import numpy as np


# Load data
data = pd.read_csv("your_data.csv")
```

```
# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

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

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)

# Predictions
y_pred = logistic_regression.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

Random Forest Regression:

python

Copy code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import pandas as pd
from sklearn.model_selection import train_test_split

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

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

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

# Predictions
```

```
y_pred = random_forest.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Random Forest Mean Squared Error:", mse)
```

K-Means Clustering:

```
python
Copy code
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import pandas as pd
import numpy as np

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Features
X = data.drop('target_column', axis=1)

# Build model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Predictions (Clusters)
```

```
clusters = kmeans.predict(X)

# Evaluate Silhouette Score (higher is better)
silhouette_avg = silhouette_score(X, clusters)
print("K-Means Silhouette Score:", silhouette_avg)
```

Sure, here are the code examples for each of the specified techniques:

Decision Tree for Classification:

python

Copy code

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Split data into features and target
X = data.drop('target_column', axis=1)
y = data['target_column']
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

# Predictions
y_pred = decision_tree.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Decision Tree Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Support Vector Machine (SVM) for Classification:

python

Copy code

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import pandas as pd

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

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

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Predictions
y_pred = svm_classifier.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("SVM Accuracy:", accuracy)

# Classification report
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Linear Regression:

python

Copy code

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

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

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Build model
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)

# Predictions
y_pred = linear_regression.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Linear Regression Mean Squared Error:", mse)

# Calculate R^2 score
r2 = r2_score(y_test, y_pred)
print("Linear Regression R^2 Score:", r2)
```

Random Forest for Regression:

python

Copy code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import pandas as pd
from sklearn.model_selection import train_test_split

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)
```

```
# Split data into features and target
X = data.drop('target_column', axis=1)
y = data['target_column']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

# Predictions
y_pred = random_forest.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Random Forest Mean Squared Error:", mse)
```

Hierarchical Clustering:

python

Copy code

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import pandas as pd

# Load data
data = pd.read_csv("your_data.csv")
```

```
# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Features
X = data.drop('target_column', axis=1)

# Build model
hierarchical_clustering = AgglomerativeClustering(n_clusters=3)
clusters = hierarchical_clustering.fit_predict(X)

# Evaluate Silhouette Score (higher is better)
silhouette_avg = silhouette_score(X, clusters)
print("Hierarchical Clustering Silhouette Score:", silhouette_avg)
```

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

```
python
Copy code
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import pandas as pd

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)
```

```
# Features
X = data.drop('target_column', axis=1)

# Build model
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(X)

# Evaluate Silhouette Score (higher is better)
silhouette_avg = silhouette_score(X, clusters)
print("DBSCAN Silhouette Score:", silhouette_avg)
```

Make sure to replace "your_data.csv" with the path to your actual dataset file and "target_column" with the name of your target column. These codes perform data preprocessing, model building, and evaluation for Decision Tree, SVM, Linear Regression, Random Forest, Hierarchical Clustering, and DBSCAN clustering techniques individually. Adjust parameters and preprocessing steps according to your specific dataset and requirements.

ENSEMBLING MODEL:

```
from sklearn.ensemble import RandomForestRegressor, VotingRegressor, VotingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.model_selection import train_test_split
import pandas as pd

# Load data for Random Forest Regression
data_rf = pd.read_csv("your_data_for_rf.csv")
```

```
# Preprocessing for Random Forest Regression
data_rf.fillna(data_rf.mean(), inplace=True)
X_rf = data_rf.drop('target_column_rf', axis=1)
y_rf = data_rf['target_column_rf']
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_rf, y_rf, test_size=0.2, random_state=42)

# Build Random Forest model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest.fit(X_train_rf, y_train_rf)

# Predictions for Random Forest
y_pred_rf = random_forest.predict(X_test_rf)
mse_rf = mean_squared_error(y_test_rf, y_pred_rf)
print("Random Forest Mean Squared Error:", mse_rf)

# Load data for SVM Classification
data_svm = pd.read_csv("your_data_for_svm.csv")

# Preprocessing for SVM Classification
data_svm.fillna(data_svm.mean(), inplace=True)
X_svm = data_svm.drop('target_column_svm', axis=1)
y_svm = data_svm['target_column_svm']
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(X_svm, y_svm, test_size=0.2, random_state=42)

# Build SVM model
```

```
svm_classifier = SVC(kernel='linear', probability=True)
svm_classifier.fit(X_train_svm, y_train_svm)

# Predictions for SVM
y_pred_svm = svm_classifier.predict(X_test_svm)
accuracy_svm = accuracy_score(y_test_svm, y_pred_svm)
print("SVM Accuracy:", accuracy_svm)

# Ensemble VotingRegressor for Random Forest Regression and VotingClassifier for SVM Classification
ensemble_rf = VotingRegressor(estimators=[('rf', random_forest)])
ensemble_svm = VotingClassifier(estimators=[('svm', svm_classifier)], voting='soft')

# Fit ensemble models
ensemble_rf.fit(X_train_rf, y_train_rf)
ensemble_svm.fit(X_train_svm, y_train_svm)

# Predictions for ensemble models
y_pred_ensemble_rf = ensemble_rf.predict(X_test_rf)
y_pred_ensemble_svm = ensemble_svm.predict(X_test_svm)

# Evaluate ensemble models
mse_ensemble_rf = mean_squared_error(y_test_rf, y_pred_ensemble_rf)
accuracy_ensemble_svm = accuracy_score(y_test_svm, y_pred_ensemble_svm)

print("Ensemble Random Forest Mean Squared Error:", mse_ensemble_rf)
print("Ensemble SVM Accuracy:", accuracy_ensemble_svm)
```