Machine Learning Techniques

0. Load Dependencies

Import all necessary libraries <**May change as per requirement**>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.*  import # As per User needs
import joblib / import pickle
```

1. Choosing a Dataset:

Find a dataset that aligns with your interests and goals.

```
# Load dataset
your_dataset = pd.read_csv('your_dataset.csv')

# Display dataset information
print(your_dataset.info())
print(your_dataset.head())
print(your_dataset.describe())
```

2. Data Exploration

1. Data Exploratory (Drawing Graphs):

```
# Import visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Example: Draw pairplot for data exploration
sns.pairplot(your_dataset)
plt.show()
```

2. Correlation Matrix:

```
# Calculate correlation matrix
correlation_matrix = your_dataset.corr()

# Plot correlation matrix as a heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

3. Preprocessing:

Analyze dataset characteristics, handle missing values, outliers, and preprocess data (e.g., encoding categorical variables, scaling features).

1. Show Null Values:

```
# Show null values
print(your_dataset.isnull().sum())
```

2. Fill Null Values:

```
# Fill null values with mean, median, or mode based on column type
your_dataset['numeric_column'].fillna(your_dataset['numeric_column'].mean(),
inplace=True)
your_dataset['categorical_column'].fillna(your_dataset['categorical_column'].mode(
)[0], inplace=True)
```

3. Dropping Null Values:

```
# Drop null values from the dataset
your_dataset.dropna(inplace=True)

# Check if null values are removed
print(your_dataset.isnull().sum())
```

4. Managing Infinite Values

```
# Identify infinite values
your_dataset.replace([np.inf, -np.inf], np.nan, inplace=True)

# Handling infinite values
your_dataset.fillna(your_dataset.max(), inplace=True)  # Replace with maximum
finite value

# Handling null values in numeric columns
your_dataset.fillna(your_dataset.mean(), inplace=True)  # Replace with mean value

# Handling null values in categorical columns
your_dataset.fillna(your_dataset.mode().iloc[0], inplace=True)  # Replace with
mode value

# Drop rows with any null value
your_dataset.dropna(inplace=True)
```

5. Label Encoder:

```
# Import LabelEncoder
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder to categorical column
your_dataset['encoded_column'] =
label_encoder.fit_transform(your_dataset['categorical_column'])
```

6. One-Hot Encoder:

```
# Import OneHotEncoder
from sklearn.preprocessing import OneHotEncoder

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Apply OneHotEncoder to categorical column
encoded_features =
onehot_encoder.fit_transform(your_dataset[['categorical_column']])

# Concatenate encoded features DataFrame with the original DataFrame
your_dataset_encoded = pd.concat([your_dataset, encoded_features], axis=1)

# Drop the original categorical column if needed
your_dataset_encoded.drop(['categorical_column'], axis=1, inplace=True)
```

7. Apply Specific Function to the Column:

```
# Apply specific function to a column
your_dataset['transformed_column'] = your_dataset['numeric_column'].apply(lambda
x: x**2)
```

8. Applying StandardScaler and MinMaxScaler:

```
# Importing necessary libraries
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Assuming 'your_dataset' is your dataset containing numeric features

# Initialize StandardScaler
standard_scaler = StandardScaler()
```

```
# Apply StandardScaler to your dataset
your_dataset_scaled_standard =
pd.DataFrame(standard_scaler.fit_transform(your_dataset),
columns=your_dataset.columns)

# Initialize MinMaxScaler
minmax_scaler = MinMaxScaler()

# Apply MinMaxScaler to your dataset
your_dataset_scaled_minmax =
pd.DataFrame(minmax_scaler.fit_transform(your_dataset),
columns=your_dataset.columns)
```

9. Detecting and Removing Outliers using Z-Score Method:

- Calculate the z-score for each data point.
- Data points with a z-score beyond a certain threshold (typically 2 or 3) are considered outliers.

```
# Detect outliers using z-score
from scipy import stats

# Calculate z-scores for each data point
z_scores = np.abs(stats.zscore(your_dataset))

# Define threshold (e.g., 3)
threshold = 3

# Remove rows with z-scores exceeding threshold
your_dataset_no_outliers = your_dataset[(z_scores < threshold).all(axis=1)]
```

10. Apply PCA:

```
# Import PCA
from sklearn.decomposition import PCA

# Initialize PCA
pca = PCA(n_components=2)

# Apply PCA to dataset
pca_result = pca.fit_transform(your_dataset)
```

4. Choosing and Training Model:

Select an appropriate machine learning algorithm based on the problem type and dataset characteristics.

Reference: Model Selection

1. Test-Train Split:

```
from sklearn.model_selection import train_test_split

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=69)
```

2. Time Series Based Split:

```
from sklearn.model_selection import TimeSeriesSplit

# Initialize TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)  # Adjust number of splits as needed

# Perform time series cross-validation
for train_index, test_index in tscv.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train and evaluate your model on the current fold
    # Example:
    # model.fit(X_train, y_train)
    # score = model.score(X_test, y_test)

    # Print fold number and evaluation score
    print(f"Fold {tscv.n_splits_ - tscv.n_iter_}/{tscv.n_splits_}: Score =
{score}")
```

5. Model Evaluation:

1.Prediction:

```
# Predict on test data
y_pred = your_model.predict(X_test)
```

Evaluate the trained model's performance using appropriate metrics on the testing dataset.

Classification Metrics:

1. **Accuracy**: Measures the proportion of correctly classified instances.
2. **Precision**: Measures the proportion of true positive predictions among all positive predictions.
3. **Recall (Sensitivity)**: Measures the proportion of true positive predictions among all actual positive instances.
4. **F1 Score**: Harmonic mean of precision and recall, balancing between precision and recall.
5. **ROC AUC Score**: Area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate.
6. **Confusion Matrix**: Tabulates the true positive, false positive, true negative, and false negative predictions.
7. **Precision-Recall Curve**: Plots the precision-recall trade-off curve.
8. **F-beta Score**: Generalized form of the F1 score that allows adjustment of the emphasis on precision versus recall (controlled by the beta parameter).

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix, precision_recall_curve, fbeta_score

# Assuming 'y_true' is the true labels and 'y_pred' is the predicted labels

# Accuracy
accuracy = accuracy_score(y_true, y_pred)

# Precision
precision = precision_score(y_true, y_pred)

# Recall
recall = recall_score(y_true, y_pred)

# F1 Score
f1 = f1_score(y_true, y_pred)

# ROC AUC Score (requires probability estimates)
roc_auc = roc_auc_score(y_true, y_pred_proba)

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
```

```
# Precision-Recall Curve (requires probability estimates)
precision, recall, thresholds = precision_recall_curve(y_true, y_pred_proba)

# F-beta Score
fbeta = fbeta_score(y_true, y_pred, beta=2)  # Adjust beta as needed
```

Regression Metrics:

1. **Mean Absolute Error (MAE)**: Average of the absolute differences between predictions and actual values.
2. **Mean Squared Error (MSE)**: Average of the squared differences between predictions and actual values.
3. **Root Mean Squared Error (RMSE)**: Square root of the MSE, providing the same unit of measurement as the target variable.
4. **R-squared (R2 Score)**: Proportion of the variance in the target variable explained by the model.
5. **Mean Absolute Percentage Error (MAPE)**: Average of the absolute percentage differences between predictions and actual values.
6. **Median Absolute Percentage Error (MdAPE)**: Median of the absolute percentage differences between predictions and actual values.
7. **Coefficient of Determination (Adjusted R2)**: Adjusted version of R-squared that penalizes complexity.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,
mean_absolute_percentage_error, median_absolute_error

# Assuming 'y_true' is the true labels and 'y_pred' is the predicted labels

# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_true, y_pred)

# Mean Squared Error (MSE)
mse = mean_squared_error(y_true, y_pred)

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# R-squared (R2 Score)
r2 = r2_score(y_true, y_pred)

# Mean Absolute Percentage Error (MAPE)
mape = mean_absolute_percentage_error(y_true, y_pred)

# Median Absolute Percentage Error (MdAPE)
mdape = median_absolute_error(y_true, y_pred)

# Adjusted R-squared
n = len(y_true)
p = X.shape[1]  # Number of features
adj_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1))
```

Clustering Metrics:

1. **Silhouette Score**: Measures how similar an object is to its own cluster compared to other clusters.
2. **Davies-Bouldin Index**: Computes the average similarity between each cluster and its most similar one, while also considering the cluster's size.
3. **Calinski-Harabasz Index**: Computes the ratio of between-cluster dispersion to within-cluster dispersion.

```
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score

# Assuming 'X' is your feature matrix and 'labels' are the cluster labels

# Silhouette Score
silhouette = silhouette_score(X, labels)

# Davies-Bouldin Index
davies_bouldin = davies_bouldin_score(X, labels)

# Calinski-Harabasz Index
calinski_harabasz = calinski_harabasz_score(X, labels)
```

6. Hyperparameter Tuning: <**Not Actual Implementation**>

Here I have listed some possible hyperparameters tunning methods, But for model accurate tuning search Google or Read Documentation

Fine-tune model hyperparameters to optimize performance using techniques like grid search or random search.

1. Grid Search:

```
from sklearn.model_selection import GridSearchCV

# Assuming 'model' is your machine learning model and 'param_grid' is the
dictionary of hyperparameters to search over

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
scoring='accuracy')

# Perform grid search
grid_search.fit(X_train, y_train)

# Get best parameters and best model
best_params_grid = grid_search.best_params_
best_model_grid = grid_search.best_estimator_
```

2. Random Search:

```
from sklearn.model_selection import RandomizedSearchCV

# Assuming 'model' is your machine learning model and 'param_distributions' is the
dictionary of hyperparameters and their distributions

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_distributions, n_iter=100, cv=5, scoring='accuracy',
random_state=42)

# Perform random search
random_search.fit(X_train, y_train)

# Get best parameters and best model
best_params_random = random_search.best_params_
best_model_random = random_search.best_estimator_
```

3. Bayesian Optimization (using Optuna):

```python
import optuna
from sklearn.model_selection import cross_val_score

# Define objective function for optimization
def objective(trial):
    # Sample hyperparameters from the search space
    param_sample = {
        'param1': trial.suggest_float('param1', 0.1, 1.0),
        'param2': trial.suggest_int('param2', 10, 100),
        # Add more hyperparameters as needed
    }

    # Initialize model with sampled hyperparameters
    model = YourModel(**param_sample)

    # Perform cross-validation and return mean accuracy
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    return scores.mean()

# Perform Bayesian optimization
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

# Get best parameters and best model
best_params_bayesian = study.best_params
best_model_bayesian = YourModel(**study.best_params)
```

7. Cross-Validation:

Description:

Use techniques like k-fold cross-validation to estimate model performance and reduce overfitting.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score

# Perform k-fold cross-validation
cv_scores = cross_val_score(your_model, X_train, y_train, cv=5)

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))

# Plot the cross-validation scores
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(cv_scores) + 1), cv_scores, color='blue')
plt.axhline(np.mean(cv_scores), color='red', linestyle='--', label='Mean CV
Score')
plt.xlabel('Fold')
plt.ylabel('Cross-Validation Score')
plt.title('Cross-Validation Scores')
plt.legend()
plt.show()
```

8. Possible Graphs: <**As Per Model Used**>

1. Feature Importance:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'model' is your trained model and 'X_train' is your training data

# Get feature importances
feature_importances = model.feature_importances_  # For tree-based models

# Create DataFrame for better visualization
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance':
feature_importances})

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

2. Learning Curve:

```
from sklearn.model_selection import learning_curve

# Define learning curve function
train_sizes, train_scores, val_scores = learning_curve(model, X_train, y_train,
cv=5, train_sizes=np.linspace(0.1, 1.0, 10))

# Calculate mean and standard deviation of train and validation scores
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
val_scores_mean = np.mean(val_scores, axis=1)
val_scores_std = np.std(val_scores, axis=1)

# Plot learning curve
plt.figure(figsize=(10, 6))
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, val_scores_mean - val_scores_std, val_scores_mean +
val_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, val_scores_mean, 'o-', color="g", label="Cross-validation
score")
plt.title("Learning Curve")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.legend(loc="best")
plt.show()
```

3. Actual vs Predicted Curve:

```
# Assuming 'y_true' are the true labels and 'y_pred' are the predicted labels

# Plot actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_true, y_pred, color='blue')
plt.plot([min(y_true), max(y_true)], [min(y_true), max(y_true)], 'k--', lw=2)  #
Diagonal line
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```

4. ROC Curve:

```
from sklearn.metrics import roc_curve, roc_auc_score

# Assuming 'y_true' are the true labels and 'y_pred_proba' are the predicted
probabilities

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_true, y_pred_proba)

# Calculate ROC AUC score
roc_auc = roc_auc_score(y_true, y_pred_proba)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

9. Save Model

```python
# Example: Save the model for deployment
import joblib
joblib.dump(your_model, 'your_model.pkl')

# Later, load the model for predictions
loaded_model = joblib.load('your_model.pkl')
```