📊 Support Vector Machine (SVM) for Human Activity Recognition

Project Objective

This project uses a **Support Vector Machine (SVM)** classifier to predict human activities based on sensor signals. The workflow includes:

- Data preprocessing and splitting
- Training an SVM model
- Model evaluation using metrics and a confusion matrix
- Hyperparameter optimization with Grid Search
- Inference on new test data
- Saving the trained model for future use

📁 Dataset Overview

The dataset originates from the UCI Machine Learning Repository and is available on Kaggle: Human Activity Recognition with Smartphones.

- **Participants**: 30 individuals aged between 19 and 48
- **Activities**: Each participant performed six activities while carrying a waist-mounted Samsung Galaxy S II smartphone:
    - WALKING
    - WALKING_UPSTAIRS
    - WALKING_DOWNSTAIRS
    - SITTING
    - STANDING
    - LAYING
- **Sensors**: The smartphone's embedded accelerometer and gyroscope captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz.
- **Features**: The dataset comprises 561 features extracted from the raw sensor signals. These features are derived from time and frequency domain variables and include statistical measures such as mean, standard deviation, and signal magnitude area.
- **Data Partitioning**: The dataset is partitioned into training and test sets, with data from 70% of the participants used for training and the remaining 30% for testing.

---

1. 🔧 Importing Required Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

Essential libraries for data handling, model training, evaluation, and optimization.

---

2. 📂 Data Loading

```
train_df = pd.read_csv("C:/Users/abedi/OneDrive/Desktop/SVM-HumanActivity/train.csv")
```

---

3. 📊 Initial Data Exploration

```
train_df.info()
train_df.describe()
train_df.columns
train_df.head()
```

Provides basic information about the dataset's structure, column names, statistics, and sample data.

---

4. 📈 Class Distribution Check

```
train_df['Activity'].value_counts()
```

Displays the number of samples for each activity class to check for imbalance.

---

5. 🏷️ Label Encoding

```
class_names = {
  "LAYING": 0,
  "STANDING": 1,
  "SITTING": 2,
  "WALKING": 3,
  "WALKING_UPSTAIRS": 4,
  "WALKING_DOWNSTAIRS": 5
```

}

Converts activity labels to numeric format required for machine learning models.

---

6. 🖌️ Feature and Target Separation

```
X = train_df.drop('Activity', axis=1)
y = train_df['Activity'].map(class_names)
```

Separates input features (X) and output labels (y).

---

7. 🪆 Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
```

Splits the dataset into:

- 80% training data
- 20% testing data
  for model validation.

---

8. 🧠 Training the Initial SVM Classifier

```
svm_model = SVC()
svm_model.fit(X_train, y_train)
```

🔮 What is SVM (Support Vector Machine)?

- SVM is a **supervised classification algorithm**.
- It works by finding the **optimal hyperplane** that separates data points of different classes with the **maximum margin**.
- It is effective for **high-dimensional** spaces and **non-linear** problems using kernels (like RBF).
- In this step, the SVM is trained with **default hyperparameters** (kernel=RBF, C=1.0, gamma='scale').

---

9. 🧪 Predictions and Evaluation (Initial Model)

```
y_pred = svm_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print("------------------")
print(classification_report(y_test, y_pred))
```

📊 Evaluation Metrics

```
[[275   0   0   0   0   0]
 [  0 231  31   0   0   0]
 [  4  43 223   0   0   0]
 [  0   0   0 224   2   0]
 [  0   0   0   4 217   4]
 [  0   0   0   2   5 206]]
------------------
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       275
           1       0.84      0.88      0.86       262
           2       0.88      0.83      0.85       270
           3       0.97      0.99      0.98       226
           4       0.97      0.96      0.97       225
           5       0.98      0.97      0.97       213

    accuracy                           0.94      1471
   macro avg       0.94      0.94      0.94      1471
weighted avg       0.94      0.94      0.94      1471
```

The classification_report shows the following for each class:

- **Precision**: How many predicted activities were correct?
  Precision = TP / (TP + FP)
- **Recall**: How many actual activities were correctly predicted?
  Recall = TP / (TP + FN)
- **F1-score**: Harmonic mean of precision and recall
  F1 = 2 * (Precision * Recall) / (Precision + Recall)
- **Support**: Number of true samples in each class

📉 Confusion Matrix

The **confusion matrix** compares actual vs predicted values:

- Rows = Actual class
- Columns = Predicted class
- Diagonal values = Correct predictions
- Off-diagonal values = Misclassifications

It helps identify which classes are being confused.

---

10. 🔁 Hyperparameter Tuning with Grid Search

```
param_grid = {
  'C': [0.1, 1, 10, 100, 1000],
  'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
}
grid_model = GridSearchCV(SVC(), param_grid, verbose=3)
grid_model.fit(X_train, y_train)
```

Explanation:

- GridSearchCV searches for the best combination of:
  - **C**: Regularization strength
  - **gamma**: Influence of a single training example
- Evaluates each combination using **cross-validation**
- Finds the model with the highest validation accuracy

---

11. 📌 Best Model Selection

```
grid_model.best_score_
grid_model.best_params_
grid_model.best_estimator_
```

Reports the best model, its parameters, and cross-validation score.

---

12. ✅ Evaluation of the Tuned Model

```
grid_pred = grid_model.predict(X_test)
print(confusion_matrix(y_test, grid_pred))
print("------------------")
print(classification_report(y_test, grid_pred))
```

```
[[275   0   0   0   0   0]
 [  0 254   8   0   0   0]
 [  0   5 265   0   0   0]
 [  0   0   0 225   1   0]
 [  0   0   0   0 225   0]
 [  0   0   0   0   0 213]]
------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       275
           1       0.98      0.97      0.98       262
           2       0.97      0.98      0.98       270
           3       1.00      1.00      1.00       226
           4       1.00      1.00      1.00       225
           5       1.00      1.00      1.00       213

    accuracy                           0.99      1471
   macro avg       0.99      0.99      0.99      1471
weighted avg       0.99      0.99      0.99      1471
```

Evaluates the model again using updated hyperparameters. Usually improves accuracy and class-wise metrics.

---

13. 💾 Saving the Best Model

```
import pickle


with open("grid_model.pkl", "wb") as file:
  pickle.dump(grid_model, file)
```

Serializes the trained model to disk using pickle for later reuse.

---

14. 💉 Inference on New Test Data

```
test_df = pd.read_csv(".../test.csv")

y_test_inference = test_df.drop('Activity', axis=1)

predictions = grid_model.predict(y_test_inference)
```

This line uses the best-trained SVM model (selected by GridSearchCV) to **predict the activity labels** based on the **test feature data**. Although the variable is named y_test_inference, it actually contains the input features (not the true labels). The output predictions will be the model's guesses for which activity each sample in the test set represents.

---

15. 🔍 Inspecting a Single Prediction

```
print(y_test_inference.iloc[0])
```

```
predictions[0]
```

```
test_df['Activity'].iloc[0]
```

Examines the sensor readings for a single instance, its predicted label, and actual label.

✅ Summary

This project:

- Implemented an SVM model for human activity classification
- Preprocessed and split sensor data
- Evaluated model performance using precision, recall, F1, and confusion matrix
- Tuned hyperparameters for performance improvement
- Saved and tested the final model on new data