Human Activity Recognition with Smartphones

Ensemble Learning Project. (By: - Shivesh (Ethan) Sahu)

Problem Definition

The goal of this project is to automatically classify **human activities** (e.g. walking, sitting, standing) based on smartphone sensor measurements, using classical machine learning and ensemble methods. The focus is not only on building a high-performing model, but also on:

- Comparing a range of linear, distance-based, and tree-based classifiers.
- Applying **hyperparameter tuning** and **ensemble methods** (voting, stacking, bagging, boosting).
- Understanding where the model still makes mistakes (especially between **SITTING** and **STANDING**).
- Packaging the final pipeline into a reusable artifact (best_har_model.joblib) with an easy inference helper.

Dataset Overview

- **Dataset**: Human Activity Recognition with Smartphones (Kaggle) – train.csv
- **Shape**: 7352 × 563
  - **561** numeric feature columns (engineered time/frequency domain features from accelerometer + gyroscope).
  - **1** subject identifier: subject (integer ID).
  - **1** target column: Activity (string label).
- **Target classes** and counts (in the full dataset):
  - LAYING – 1407 samples
  - STANDING – 1374 samples
  - SITTING – 1286 samples
  - WALKING – 1226 samples
  - WALKING_UPSTAIRS – 1073 samples
  - WALKING_DOWNSTAIRS – 986 samples
- **Data quality**:
  - All 561 feature columns are float64.
  - subject is int64.
  - Activity is object (categorical labels).
  - **No missing values**: total missing count = 0 across the entire dataset.

Exploratory Data Analysis (EDA)

3.1 Basic statistics

For the first 10 features (e.g. tBodyAcc-mean()-X, tBodyAcc-std()-X, …), summary statistics indicate:

- Features are already scaled into a range around -1, 1 with mean values typically between ~−0.7 and 0.3.
- Standard deviations vary but are moderate, confirming that features are numerically well-behaved.

3.2 Class distribution

A count plot of Activity shows:

- Slight class imbalance (e.g. more **LAYING** and **STANDING** than **WALKING_DOWNSTAIRS**), but all classes still have >900 samples.
- No extreme minority classes, so standard cross-entropy / macro-F1 evaluation is appropriate.

3.3 Subject distribution

- subject counts range from ~281 to 366 per user (first ten IDs inspected).
- This confirms that activities are captured across multiple individuals, which reduces the risk of the model overfitting to a single subject.

3.4 Feature behavior and correlations

For a small subset of features:

["tBodyAcc-mean()-X", "tBodyAcc-mean()-Y", "tBodyAcc-std()-X", "tBodyAcc-std()-Y"]

- **Histograms**:
    - Means (tBodyAcc-mean()) show fairly concentrated distributions, with some skew.
    - Standard deviations vary more widely, reflecting differences in movement intensity.
- **Boxplots**:

- Some potential outliers appear, but they are plausible sensor readings rather than obvious errors.
- **Correlation heatmap** (subset only):
    - High positive correlation between tBodyAcc-std()-X and tBodyAcc-std()-Y (~0.93).
    - Weak correlations between mean and std features across axes (near 0), indicating they provide complementary information.

Data Preparation

4.1 Feature/target split

- **Features (X)**: all columns except Activity and subject.
    - 561 numeric features kept.
    - subject is deliberately removed to avoid ID-leakage (we want a model that generalizes across unseen people).
- **Target (y)**: Activity.

4.2 Train–test split

- train_test_split with:
    - test_size=0.2
    - random_state=42
    - stratify=y to preserve class proportions.

Result:

- X_train: (5881, 561)
- X_test: (1471, 561)

4.3 Scaling & pipelines

Because many algorithms are sensitive to feature scaling, a **ColumnTransformer + Pipeline** pattern was used:

- scaler = ColumnTransformer([("num", StandardScaler(), num_features)], remainder="drop")

- Two helper constructors:
  - *def scaled_pipeline(estimator):*
  - *return Pipeline([("scaler", scaler), ("estimator", estimator)])*
  -
  - *def passthrough_pipeline(estimator):*
  - *return Pipeline([("clf", estimator)])*

- **Scaled** models:
  - linear and distance-based (Perceptron, RidgeClassifier, LogisticRegression, SVM, KNN).
- **Unscaled / passthrough**:
  - tree-based models and ensembles (DecisionTree, RandomForest, bagging/boosting).

Baseline Models

The following baseline models were trained on the training set and evaluated on the test set using **accuracy** and **macro-averaged F1**:

Model	Accuracy	F1 (macro)
SVM (RBF)	0.9898	0.9906
RandomForest	0.9878	0.9879
Logistic Regression	0.9857	0.9868
RidgeClassifier (Adaline-like)	0.9810	0.9822
Perceptron	0.9810	0.9818
KNN (k = 15, distance)	0.9565	0.9578
DecisionTree	0.9388	0.9380

Observations:

- All linear models perform surprisingly well once features are standardized, suggesting that the engineered HAR features are highly informative and almost linearly separable.

- SVM with RBF kernel already reaches **~0.99 accuracy**, indicating minimal remaining headroom.

A confusion matrix for the **baseline SVM** shows an almost perfect diagonal; the most noticeable residual confusion is between **SITTING** and **STANDING** (some SITTING predicted as STANDING and vice versa), which is intuitively reasonable given their similar motion patterns.

Hyperparameter Tuning

To push baseline models further, **RandomizedSearchCV** was applied with stratified K-fold CV:

- FAST_SEARCH = True → cv_splits = 3, modest n_iter per model (15).
- Scoring metric: f1_macro.
- Models tuned:
    - SVM (RBF)
    - RandomForest
    - Logistic Regression
    - KNN

Tuned parameter highlights

- **SVM (RBF)**
    - Best: C ≈ 31.43, gamma="auto"
    - CV macro-F1 ≈ 0.9830
    - Test: unchanged vs baseline (~0.9898 acc / 0.9906 F1), i.e., already near optimal.
- **RandomForest**
    - Best: ~341 trees, max_depth=30, max_features="log2", min_samples_split=3, min_samples_leaf=2.
    - CV macro-F1 ≈ 0.9757.
    - Test: ≈ 0.9850 acc / 0.9853 F1 (slightly below the untuned 300-tree RF).
- **Logistic Regression**
    - Best: C ≈ 0.53, penalty="l2", solver="lbfgs".
    - Test: ≈ 0.9857 acc / 0.9868 F1 (similar to baseline).
- **KNN**
    - Best: n_neighbors=16, weights="uniform", p=1.
    - Test: ≈ 0.9708 acc / 0.9721 F1 (noticeable improvement over the original k=15 but still below linear/SVM models).

Conclusion: tuning delivered small gains, but SVM (RBF) and RandomForest were already very strong.

Classical Ensembles: Voting & Stacking

Using the tuned (or baseline) pipelines as building blocks, three higher-level ensembles were built:

1. **Voting (hard)** – majority vote of:
   - Perceptron, RidgeClassifier, Logistic Regression (tuned), SVM (RBF, tuned), DecisionTree, RandomForest (tuned), KNN (tuned).
2. **Voting (soft)** – probability-average of:
   - Logistic Regression (tuned), SVM (RBF with probability=True), RandomForest (tuned), KNN (tuned).
3. **Stacking (LR meta)** – meta-learner:
   - Base estimators: Logistic Regression (tuned), SVM (probabilistic), RandomForest (tuned), KNN (tuned).
   - Final estimator: Logistic Regression with max_iter=5000.
   - stack_method="predict_proba", cross-validated meta-features with StratifiedKFold.

7.1 Ensemble performance

Ensemble	Accuracy	F1 (macro)
Voting (hard)	0.9905	0.9912
Voting (soft)	0.9918	0.9925
Stacking (LR meta)	**0.9946**	**0.9950**

The **Stacking** ensemble clearly outperforms all individual models and both voting schemes.

7.2 Confusion matrix & error analysis (Stacking)

The confusion matrix for **Stacking (LR meta)** is extremely close to a perfect diagonal:

- All **walking-type activities** (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS) are classified almost perfectly.
- **LAYING** has near-perfect precision/recall.
- Residual errors are almost entirely between **SITTING** and **STANDING** (e.g., a handful of SITTING misclassified as STANDING and vice versa).

Interpretation:

- SITTING vs STANDING are biomechanically similar when captured by a waist-mounted smartphone; differences in vertical acceleration and orientation are subtle.
- Additional sensor context (e.g., orientation sequences, seat pressure sensors) might be needed to fully disambiguate them; given the current feature set, an error rate of only a few samples is already extremely strong.

Bagging & Boosting Experiments

To keep tree-based ensembles clearly separated from the original model dictionary, a new bag_boost_models registry was built using passthrough_pipeline:

- **Bagging (DT)** – BaggingClassifier with DecisionTree base learners.
- **ExtraTrees** – Extremely randomized trees with 400 trees.
- **AdaBoost** – AdaBoostClassifier with shallow DecisionTree base estimators.
- **HistGradientBoosting** – Histogram-based gradient boosting classifier.

8.1 Baseline results

Model	Accuracy	F1 (macro)
HistGradientBoosting	0.9939	0.9941
ExtraTrees	0.9932	0.9937
AdaBoost	0.9864	0.9869
Bagging (DT)	0.9721	0.9722

These results are very competitive:

- **HistGradientBoosting** and **ExtraTrees** nearly match the Stacking ensemble, confirming that boosted/bagged trees are also an excellent fit for this problem.
- However, in this run the **Stacking (LR meta) model still achieves the highest macro-F1**, so it remains the global best.

8.2 Hyperparameter tuning for bagging/boosting

A second stage of RandomizedSearchCV was run with a **QUICK** mode (n_iter ≈ 12 for most models):

- **ExtraTrees** – tuned over number of trees, max depth, max features, and min samples split/leaf.
- **Bagging (DT)** – tuned over number of estimators, bootstrap vs. pasting, and inner tree depth / split settings.
- **AdaBoost** – tuned number of estimators, learning rate, and max depth of the base tree.
- **HistGradientBoosting** – tuned over learning rate, max iterations, max leaf nodes, max depth, min samples leaf, and L2 regularization.

The tuned versions improved macro-F1 further (see notebook output for exact values), but **none were promoted as the final production model**, primarily because:

- The stacking model was already saved and integrated into the deployment artifact.
- Differences in performance among the very top models are small enough that choosing the simpler, already-packaged model is reasonable.

Final Model Selection and Artifacts

9.1 Global leaderboard (non-bagging/boosting)

From the consolidated comparison of all non-bagging/boosting models plus ensembles:

1. **Stacking (LR meta)** – 0.9946 acc / 0.9950 macro-F1
2. Voting (soft)
3. Voting (hard)
4. SVM (RBF, tuned)

5. SVM (RBF, baseline)
6. RandomForest, etc.

Thus, **Stacking (LR meta)** is selected as the **final model** for deployment.

9.2 Saved artifacts

Two key artifacts were persisted:

1. **Model**
   - File: best_har_model.joblib
   - Content: full scikit-learn pipeline for **Stacking (LR meta)**, including all preprocessing steps (scaling) and base/meta learners.
2. **Feature list**
   - File: feature_names.txt
   - One feature name per line in the exact order used during training.
   - Ensures that incoming data for inference is correctly aligned with the model's expected columns.

Both files are stored alongside the notebook and can be version-controlled.

Inference Interface

To make real-world usage convenient, a helper function was implemented:

def predict_activities(new_data, model_path=MODEL_PATH, feature_list_path=FEATURE_LIST_PATH):

"""

new_data: pandas DataFrame (same feature space) OR path to CSV with same columns.

Returns: numpy array of predicted activity labels.

"""

from joblib import load

if isinstance(new_data, str):

```python
        new_df = pd.read_csv(new_data)
    else:
        new_df = new_data.copy()

    # Load feature list and check columns
    with open(feature_list_path, "r") as f:
        feats = [ln.strip() for ln in f if ln.strip()]

    missing = [c for c in feats if c not in new_df.columns]
    if missing:
        raise ValueError(
            f"Missing required columns: {missing[:10]}{'...' if len(missing) > 10 else ''}"
        )

    # Align column order
    new_df = new_df.reindex(columns=feats)

    # Load model and predict
    model = load(model_path)
    return model.predict(new_df)
```

Key properties:

- Accepts either a **DataFrame** or a path to a **CSV file**.
- Validates that all required features are present; fails fast with a helpful message if not.
- Reorders columns to match training order before prediction.
- Returns predicted **activity labels** as a NumPy array.

This function can be imported into a separate script or API service to perform inference on new sensor data batches.

Limitations and Future Work

1. **SITTING vs STANDING confusion**
   - Even the best model occasionally confuses these two classes, reflecting the inherent similarity from accelerometer/gyroscope data alone.
   - Future work could explore:
     - Additional engineered features capturing postural orientation more explicitly.
     - Sequence models (e.g. LSTMs or HMMs) that consider temporal dynamics rather than single windows.
2. **Subject leakage / generalization**
   - The model is trained and tested on the same subject pool (though stratified).
   - A stricter evaluation would use **subject-wise cross-validation**: train on some subjects, test on entirely unseen subjects.
3. **Model size and latency**
   - Stacking combines several heavy base learners; for deployment on a resource-constrained device, a lighter model such as HistGradientBoosting or a single tuned SVM might be preferable.
4. **Hyperparameter search budget**
   - All searches used a "FAST_SEARCH" / "QUICK" configuration; deeper search (more n_iter, 5-fold CV) could extract slightly more performance at higher compute cost.

Summary

- Using the Kaggle Human Activity Recognition dataset with 561 engineered features, we built and compared a rich set of classical ML models and ensembles.
- After careful preprocessing, EDA, baseline comparison, and hyperparameter tuning, the **StackingClassifier with Logistic Regression meta-learner** emerged as the best performer with **~0.995 macro-F1** on a held-out test set.
- Bagging/boosting methods (ExtraTrees, HistGradientBoosting, AdaBoost, Bagging DT) proved highly competitive and almost matched stacking performance, further validating the predictability of the problem.
- The project concludes with a **production-ready pipeline** (best_har_model.joblib), a **feature specification file**, and a **robust inference helper**, making the solution easy to integrate into downstream systems or real-time HAR applications.