Classifying Pre-Labeled Robot Wall-Following Commands from Ultrasonic Sensor Data Using Logistic Regression and Support Vector Machines

Marcus Greenan
MAE 146: Introduction to Machine Learning Algorithms with Python

1 Abstract

This project is an offline supervised classification study using the public UCI Wall-Following Robot Navigation Data Set. The goal is to test whether pre-labeled robot wall-following command labels can be classified from ultrasonic sensor snapshots, and whether a nonlinear SVM improves held-out classification performance over linear models. The main experiment uses `sensor_readings_24.data`, which contains 5,456 samples, 24 ultrasonic sensor features, and four dataset-provided command labels: Move-Forward, Slight-Right-Turn, Sharp-Right-Turn, and Slight-Left-Turn. I compare a dummy majority-class baseline, logistic regression, linear SVM, and RBF-kernel SVM. Hyperparameters are selected with stratified 5-fold GridSearchCV using macro F1, with scaling inside scikit-learn pipelines to avoid leakage. Final evaluation is performed on a held-out stratified 20% test set by comparing predicted command labels against true dataset command labels. The best class-report model was the RBF SVM, with test accuracy 0.929 and test macro F1 0.925. This project does not estimate robot position, measure physical standoff error, or deploy a controller on a physical robot.

2 Introduction and Research Question

This project evaluates a narrow classification problem: given one ultrasonic sensor snapshot from a pre-labeled dataset, predict the corresponding command label in that dataset. The research question is: Can pre-labeled robot wall-following commands be classified from ultrasonic sensor readings, and does a nonlinear SVM improve held-out classification performance over linear models?

This framing keeps the verification target within the available data. The model is not evaluated with physical coordinate measurements or robot-trial success. It is evaluated only on command-label agreement on held-out samples.

3 Dataset and Ground-Truth Verification

The dataset is the UCI Wall-Following Robot Navigation Data Set. The class report uses the official 24-sensor file, `sensor_readings_24.data`, collected from a SCITOS G5 robot. Each sample contains 24 numeric ultrasonic readings named `US_1` through `US_24` and one true command label. The loaded dataset has 5,456 samples, 24 features, and 0 missing values.

Table 1: Command-label distribution.

Command label	Samples	Percent
Move-Forward	2,205	40.4%
Sharp-Right-Turn	2,097	38.4%
Slight-Right-Turn	826	15.1%
Slight-Left-Turn	328	6.0%

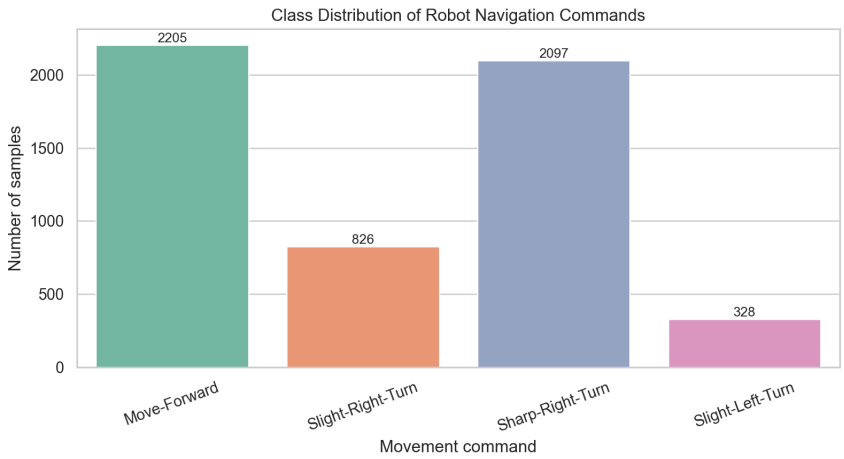

Figure 1: Class distribution of the dataset-provided command labels.

Ground-Truth Verification. Each dataset sample contains 24 ultrasonic sensor readings and a true command label. The model predicts one command label for each held-out test sample. The prediction is verified by comparing `predicted_command` to `true_command`. A prediction is counted correct when `predicted_command == true_command`. Accuracy is the fraction of held-out test samples classified correctly. Macro F1 is reported because the command classes are imbalanced. The confusion matrix directly cross-checks which commands were classified correctly and which command pairs were confused.

This project does not estimate robot position, measure physical standoff error, or deploy a controller on a physical robot; verification is performed by comparing predicted command labels against the dataset-provided command labels on a held-out test set.

Table 2: Dataset verification summary.

Verification item	Value
Dataset source	UCI Wall-Following Robot Navigation Data Set
Raw file	`sensor_readings_24.data`
Samples / features	5,456 / 24
Target column	`command`
Missing values	0
Train / test size	4,364 / 1,092
Split	Stratified 80/20, random state 42
Preprocessing	`StandardScaler` inside scikit-learn `Pipeline`

4 Methods

The dummy majority-class baseline predicts the most common command label. Logistic regression is a probability-based linear classifier with L2 regularization. The linear SVM is a margin-based linear classifier. The RBF SVM is a nonlinear kernel classifier that can represent curved decision boundaries in sensor-feature space. The class report focuses on logistic regression and SVMs because these are directly aligned with the course scope and the TA-facing verification target.

Because the target has four command labels rather than a binary $+1/-1$ label, the implementation uses scikit-learn's multiclass classification support. `LogisticRegression` with the `lbfgs` solver handles the four-class problem directly, and `SVC` handles multiclass SVM classification internally using one-vs-one decision functions. All reported precision, recall, and F1 values use macro averaging across the four command labels.

5 Cross-Validation and Model Selection

The data were split into stratified training and held-out test sets. Hyperparameters were selected using stratified 5-fold GridSearchCV on the training set only. Macro F1 was used as the primary cross-validation score because the command labels are imbalanced. Scaling was placed inside each model pipeline so preprocessing was fit only within training folds. The held-out test set was used only after model selection.

Table 3: Model selection summary.

Model	Tuned parameters	Best parameters	CV macro F1
Dummy Majority	strategy	most_frequent	0.144
Logistic Regression	C	C=100	0.671
Linear SVM	C	C=100	0.710
RBF SVM	C, gamma	C=100, gamma=scale	0.919

6 Results

The RBF SVM was the best class-report model by held-out test macro F1 among logistic regression, linear SVM, and RBF SVM. It reached test accuracy 0.929 and test macro F1 0.925. The nonlinear SVM substantially outperformed the linear SVM and logistic regression on the held-out test set.

Table 4: Held-out command-label classification results.

Model	Train Acc.	Train F1	Test Acc.	Test Prec.	Test Recall	Test F1
RBF SVM	0.991	0.992	0.929	0.924	0.927	0.925
Linear SVM	0.742	0.723	0.737	0.739	0.678	0.701
Logistic Regression	0.713	0.677	0.701	0.668	0.650	0.657
Dummy Majority	0.404	0.144	0.404	0.101	0.250	0.144

Figure 2: Macro F1 for class-report models.

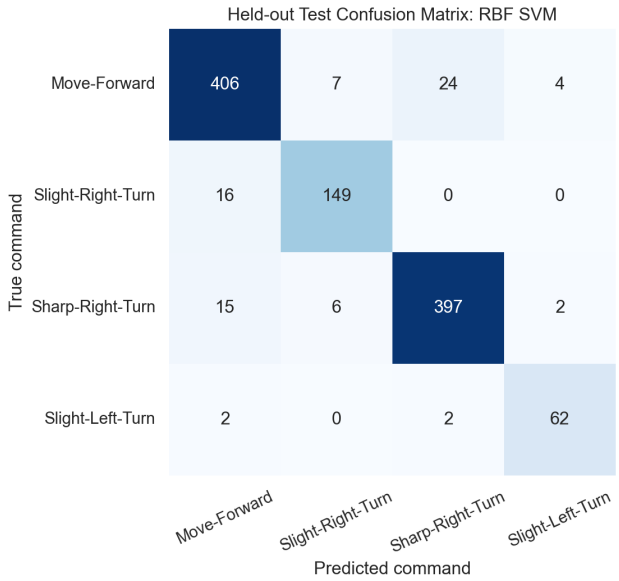

Figure 3: RBF SVM confusion matrix.

The largest off-diagonal confusion for the RBF SVM was Move-Forward predicted as Sharp-Right-Turn for 24 held-out test samples. This is a command-label classification error, not a measured physical path error.

7 Discussion, Limitations, Conclusion, Sources, and AI Disclosure

The results support the research question: the nonlinear RBF SVM improved held-out classification performance over the linear models in this split. Logistic regression reached test macro F1 0.657, the linear SVM reached 0.701, and the RBF SVM reached 0.925. The gap suggests that the relationship between ultrasonic sensor snapshots and command labels is not well captured by a single linear boundary.

The main limitation is the verification target. This project verifies command-label agreement only. It does not use physical ground-truth measurements, is not a reinforcement-learning method, and was not tested as software on a physical robot. The dataset comes from one robot/platform/environment, and each input is a static snapshot rather than a temporal sequence. Therefore, the results should be interpreted as held-out classification performance on pre-labeled data, not as evidence of physical robot performance.

In conclusion, the project satisfies the MAE 146 classification objective with a public dataset, a clear supervised learning question, multiple model types, cross-validation for parameter selection, quantitative held-out metrics, and a confusion-matrix check against dataset-provided labels. The safest supported claim is that an RBF SVM classified pre-labeled wall-following command labels from 24 ultrasonic readings with test accuracy 0.929 and macro F1 0.925 on the held-out split. The code appendix includes the wrapper code in `main.py`, the supporting `src/` modules, and the commands needed to reproduce the analysis.

Sources: UCI Wall-Following Robot Navigation Data Set; scikit-learn; NumPy; pandas; Matplotlib; seaborn; joblib. OpenAI Codex/ChatGPT was used to assist with code organization, debugging, report drafting, and editing. All code outputs, numerical results, and claims were reviewed against generated project files.