Final Project Report: Comparative Analysis of Machine Learning Models

Overview

This project investigates the performance of several supervised learning models on a real-world dataset by implementing them from scratch and comparing their behavior across different conditions. The goal was to understand the nuances of how various models perform when applied to the same data, especially with respect to feature engineering and hyperparameter tuning. All implementations were built using Python without relying on high-level machine learning libraries such as scikit-learn or PyTorch, allowing for deeper engagement with the underlying algorithmic principles.

The implemented models include Support Vector Machine (SVM), Logistic Regression, AdaBoost, Decision Tree, and Perceptron. Each model was trained and evaluated using multiple feature preprocessing strategies and optimized using cross-validation techniques.

1. Important Ideas Explored

Feature Transformation

A major component of the project involved understanding the effect of different feature transformations on model performance. Initially, features that were zero in more than 98% of examples were removed, under the assumption that they did not contribute meaningful information. However, through experimentation, it became clear that such aggressive pruning could discard rare but significant signals. Instead, the strategy was refined to only eliminate features that were zero in **all** training examples, ensuring no informative features were lost.

In addition to zero-value pruning, **variance thresholding** was applied to eliminate features with extremely low variability, as these typically do not contribute significantly to model performance.

Hyperparameter Tuning

Each model was extended to allow tuning of its key hyperparameters:

- **SVM and Logistic Regression:** Learning rate (γ_0), regularization trade-off (C), and training epochs.
- **Decision Tree and AdaBoost:** Maximum depth, minimum samples per split, and splitting criterion (entropy vs. gini).
- **Perceptron:** Learning rate, margin (mu), and number of epochs.

These parameters were systematically explored using grid search and cross-validation to determine the most effective configuration for each model.

Cross-Validation and Evaluation

Five-fold cross-validation was implemented manually. This approach enabled a robust comparison of different model settings and prevented overfitting to a specific partition of the data. Accuracy metrics were computed for training, testing, and evaluation subsets to gain insights into generalization performance.

2. Ideas from Class Used

All models were built from first principles, reflecting core concepts introduced in the course:

- **Perceptron:** A linear classifier that updates weights based on a margin condition. The margin-based variant was implemented to allow better generalization and noise tolerance.
- **SVM:** Trained using stochastic gradient descent to minimize the hinge loss, incorporating L2 regularization for stability.
- **Logistic Regression:** Optimized using cross-entropy loss and gradient descent. A sigmoid function was used to map outputs to probabilities.
- **Decision Tree:** Built via recursive binary splits using entropy or gini impurity to maximize information gain at each node.
- **AdaBoost:** An ensemble learning method that combines weak learners (decision stumps) by reweighting misclassified samples over multiple rounds.

These models provided a concrete way to understand the theory from lectures and apply it to a real-world dataset in a controlled experimental environment.

3. Key Learnings

Several insights emerged over the course of this project:

- **Feature Transformation is Model-Specific:** Linear models (SVM, logistic regression) benefit greatly from normalization and filtering, whereas tree-based models (decision tree, AdaBoost) are robust to raw input features.
- **Aggressive feature removal is risky:** Initially removing features with high zero frequency hurt model performance. Instead, removing only fully zero features retained important signal-bearing columns.
- **Tooling matters:** Reimplementing models by hand was instructive, but using libraries like scikit-learn or PyTorch yields dramatically improved performance and flexibility. Exploring these frameworks will be a key part of future learning.
- **No one-size-fits-all model:** Each model had strengths depending on the data structure. Decision trees overfit easily but performed well on training data, while logistic regression and SVM offered more balanced results.

4. Results and Observations

Perceptron

Best hyperparameters: lr = 1.0, margin = 10.0
Train = 0.506, Test = 0.506, Eval = 1.000
Design decisions: Implemented a margin-based variant for better generalization. Training involved epoch-based updates with shuffled data. Hyperparameters were selected using cross-validation. The model showed robustness to minor preprocessing but struggled with complex, non-linear boundaries.

AdaBoost

Best hyperparameters: max_depth = 1, min_samples_split = 2, criterion = gini
Train = 0.707, Test = 0.702, Eval = 0.642
Design decisions: Constructed as an ensemble of shallow decision stumps. Each stump was reweighted according to misclassification error, enhancing focus on harder examples. This method worked well on the structured input due to its ability to focus on minority signals.

Decision Tree

Best hyperparameters: max_depth = 15, min_samples_split = 2
Train = 0.973, Test = 0.836, Eval = 0.524
Design decisions: Implemented recursive splitting with entropy as the splitting criterion. Limited depth and sample split to avoid overfitting. Good fit for categorical or mixed data, but struggled with generalization due to model complexity.

SVM

Best hyperparameters: lr0 = 0.0001, reg_tradeoff = 0.001
Train = 0.612, Test = 0.636, Eval = 0.616
Design decisions: Used stochastic gradient descent to minimize hinge loss. Learning rate decay was introduced for convergence stability. Regularization handled through a trade-off term balancing margin width and error. Well-suited for sparse feature sets.

Logistic Regression

Best hyperparameters: lr0 = 0.0001, reg_tradeoff = 1000.0
Train = 0.626, Test = 0.632, Eval = 0.649
Design decisions: Trained using SGD with logistic loss and sigmoid output. Learning rate decay was introduced, and regularization improved stability. Required careful preprocessing and showed sensitivity to hyperparameter tuning.

Logistic Regression (scikit-learn)

Best hyperparameters: solver = 'liblinear', C = 1.0, max_iter = 100
Train = 0.755, Test = 0.756, Eval = 0.572
Design decisions: Used `LogisticRegression` from scikit-learn with basic hyperparameters tuned through validation. Regularization parameter C was set to 1.0 and the 'liblinear' solver was chosen for stability on smaller datasets. The implementation showed improved performance even without explicit feature transformation, demonstrating the robustness of scikit-learn's built-in preprocessing and optimization. Higher F1 scores on both test and evaluation sets indicated better class balance handling.

5. Future Directions

With additional time and resources, several enhancements would be explored:

- **Model-Specific Preprocessing:** Tailoring preprocessing pipelines to suit the inductive biases of each model.
- **Library Comparison:** Benchmarking these custom models against scikit-learn or PyTorch equivalents to isolate where optimizations yield benefits.
- **Ensemble Techniques:** Combining multiple models through voting or stacking could harness complementary strengths.
- **Automated Feature Selection:** Tools like recursive feature elimination or mutual information filtering could enhance model input relevance.
- **Explainability:** Implementing SHAP or LIME to understand why each model makes certain predictions.