BBM409: Introduction to Machine Learning Lab
Fall 2025
Instructor: Assoc. Prof. Dr. Hacer Yalım Keleş
TA: Sevginur İnce

Assignment 2: Classification with SVM and Ensemble Methods

Due date: Tuesday, 18-11-2025, 11:59 PM

In this assignment, you will explore classification algorithms for both binary and multi-class tasks. The assignment consists of two main parts: Support Vector Machines (SVM) for binary classification, and multi-class classification using Multinomial Logistic Regression (implemented from scratch), Decision Trees, and XGBoost. You will work with the Sonar Dataset for binary classification and the Dry Bean Dataset for multi-class classification.

The steps you will follow include:

1. **Implementation:** You will implement SVM models for binary classification, implement Multinomial Logistic Regression from scratch, and utilize sklearn for Decision Tree and XGBoost implementations.
2. **Hyperparameter Tuning:** You will use GridSearchCV with k-fold cross-validation to find optimal hyperparameters for all models.
3. **Evaluation:** You will evaluate models using accuracy, classification reports, and confusion matrices on test sets.
4. **Analysis:** You will compare model performances and discuss the advantages of different approaches.

1 Part 1: Binary Classification with SVM (30 points)

1.1 Dataset

You will use the Sonar Dataset from the UCI Machine Learning Repository. The dataset contains 208 samples with 60 numerical features representing sonar signal energy values. Your task is to classify sonar returns as either bouncing off a metal cylinder (Mine) or a rock (Rock).

1.2 Support Vector Machine (SVM) Algorithm

The Support Vector Machine (SVM) is a supervised learning algorithm for binary classification that finds the optimal hyperplane separating two classes with maximum margin.

1.2.1 How the Algorithm Works

1. Decision Function: For any input $\mathbf{x}$, the SVM computes:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

The SVM classifies $\mathbf{x}$ based on the sign of $f(\mathbf{x})$:

$$\text{Predicted class} = \begin{cases} +1, & \text{if } f(\mathbf{x}) \geq 0 \\ -1, & \text{if } f(\mathbf{x}) < 0 \end{cases}$$

2. Optimization Problem: SVM finds the optimal hyperplane by solving:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i$$

3. Soft Margin with Hinge Loss: For non-linearly separable data:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \max(0, 1 - y_i f(\mathbf{x}_i))$$

4. Kernel Trick: For non-linearly separable data, kernels transform inputs:

$$\begin{cases} \text{Linear: } K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j \\ \text{Polynomial: } K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j + c)^d \\ \text{RBF: } K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2) \end{cases}$$

1.3 Implementation Tasks

1.3.1 1.1. Linear Kernel SVM without Hyperparameter Tuning (5 points)

1. Load the Sonar Dataset from UCI Repository using `ucimlrepo`.
2. Encode target labels to numeric values.
3. Split data into 80% training and 20% test sets with stratification.
4. Create a pipeline with StandardScaler and Linear SVM (`kernel='linear'`).
5. Train the model on training data.
6. Calculate and report train and test accuracies.
7. Display classification report and confusion matrix on test set.

1.3.2 1.2. SVM with GridSearchCV and 5-Fold Cross-Validation (15 points)

1. Create a pipeline with StandardScaler and SVM.
2. Define parameter grid for different kernels:
   - Linear kernel: C values (e.g., [0.1, 1, 10, 100])
   - RBF kernel: C and gamma values (e.g., C=[0.1, 1, 10, 100], gamma=['scale', 'auto', 0.001, 0.01])

- Polynomial kernel: C, gamma, and degree values (e.g., degree=[2, 3, 4])

3. Use StratifiedKFold with 5 splits for cross-validation.
4. Run GridSearchCV on training data.
5. Report best parameters and best cross-validation score.
6. Evaluate the best model on test set.
7. Display top 5 parameter combinations with their CV scores.
8. Display classification report and confusion matrix for best model.

1.3.3 Comparison and Analysis (10 points)

Compare the performance of Linear SVM (without tuning) vs Tuned SVM (with GridSearchCV). Discuss:

- The performance impact of linear and nonlinear kernels
- Why the kernel trick is important
- Why k-fold cross-validation is more advantageous than a simple train-test split

2 Part 2: Multi-Class Classification (70 points)

2.1 Dataset

You will use the Dry Bean Dataset from the UCI Machine Learning Repository. The dataset contains 13,611 samples of seven different varieties of dry beans with 16 numerical features.

2.2 2.1. Multinomial Logistic Regression (20 points)

2.2.1 Algorithm Overview

Multinomial Logistic Regression extends binary logistic regression to multiple classes using the softmax function.

1. Softmax Function: For a sample $\mathbf{x}$ with K classes:

$$P(y = k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x} + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \mathbf{x} + b_j)}$$

2. Cross-Entropy Loss:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log P(y_i = k|\mathbf{x}_i)$$

3. Gradient Computation:

$$\frac{\partial L}{\partial \mathbf{w}_k} = -\frac{1}{N} \sum_{i=1}^{N} (y_{ik} - P(y_i = k|\mathbf{x}_i))\mathbf{x}_i$$

4. Weight Update:

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta \frac{\partial L}{\partial \mathbf{w}_k}$$

2.2.2 Implementation Requirements

IMPORTANT: You must implement Multinomial Logistic Regression from scratch without using sklearn's LogisticRegression.

1. Load the Dry Bean Dataset from UCI Repository.
2. Encode categorical labels to numeric values.
3. Split data into 80% training and 20% test sets with stratification.
4. Standardize features using StandardScaler.
5. Implement a `MultinomialLogisticRegression` class with:
   - `__init__`: Initialize with learning_rate, epochs, reg_lambda
   - `_softmax`: Compute softmax probabilities
   - `_cross_entropy`: Compute loss with optional L2 regularization
   - `fit`: Train using gradient descent
   - `predict_proba`: Return class probabilities
   - `predict`: Return predicted classes
   - `score`: Return accuracy
   - `get_params` and `set_params`: For GridSearchCV compatibility
6. Define parameter grid (e.g., learning_rate=[0.01, 0.05, 0.1], epochs=[200, 300, 500], reg_lambda=[0.0, 0.01, 0.1]).
7. Use GridSearchCV with 5-fold cross-validation to find best parameters.
8. Report best parameters, best CV score, and test accuracy.
9. Display classification report and confusion matrix on test set.

2.3 2.2. Decision Tree (20 points)

2.3.1 Algorithm Overview

The Decision Tree Algorithm creates a tree structure where each internal node represents a feature test, each branch represents an outcome, and each leaf represents a class label.
Gini Impurity:

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2$$

Entropy:

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log_2 p_i$$

2.3.2 Implementation Requirements (15 points)

1. Use the same preprocessed Dry Bean Dataset (already split and scaled).
2. Define parameter grid for Decision Tree:
   - max_depth: [10, 15, 20, 25, None]
   - min_samples_split: [2, 5, 10]
   - min_samples_leaf: [1, 2, 4]
   - criterion: ['gini', 'entropy']
3. Use GridSearchCV with 5-fold cross-validation.
4. Report best parameters, best CV score, and test accuracy.
5. Display classification report and confusion matrix on test set.

2.3.3 Analysis (5 points)

Describe how a decision tree builds its decision structure.

2.4 2.3. XGBoost (20 points)

2.4.1 Algorithm Overview

XGBoost is an advanced gradient boosting implementation that builds an ensemble of decision trees sequentially.
Ensemble Learning:

$$\hat{y}_i = \sum_{t=1}^{T} f_t(\mathbf{x}_i)$$

Objective Function:

$$\text{Obj} = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t)$$

2.4.2 Implementation Requirements (10 points)

1. Use the same preprocessed Dry Bean Dataset.
2. Define parameter grid for XGBoost:
   - max_depth: [3, 5, 7, 10]
   - learning_rate: [0.01, 0.1, 0.3]
   - n_estimators: [100, 200, 300]
   - subsample: [0.8, 1.0]
   - colsample_bytree: [0.8, 1.0]
3. Use GridSearchCV with 5-fold cross-validation.
4. Set `eval_metric='mlogloss'` in XGBClassifier.

5. Report best parameters, best CV score, and test accuracy.
6. Display classification report and confusion matrix on test set.

2.4.3 Analysis (10 points)

1. What are the revolutionary features of XGBoost compared to other tree-based models?
2. Compare the classification results of Multinomial Logistic Regression, Decision Tree, and XGBoost on the Dry Beans dataset. Discuss:
   - Overall model performance
   - Risk of overfitting
   - Model complexity
   - Scenarios in which each model is most effective

3 Grading

- Part 1.1: Linear SVM without tuning (5 points)
- Part 1.2: SVM with GridSearchCV (15 points)
- Part 1: Comparison and Analysis (10 points)
- Part 2.1: Multinomial Logistic Regression (20 points)
- Part 2.2: Decision Tree (20 points)
- Part 2.3: XGBoost (20 points)
- Part 2.4: Comparison and analysis for multi-class classification experiments

Total: 100 points

4 Submission Instructions

Submit a ZIP file named `<studentid>_assignment2.zip` containing:

- Jupyter notebook file: `<studentid>_assignment2.ipynb`
- The notebook should include:
  - All code implementations
  - Markdown cells with answers to analysis questions
  - All outputs, including classification reports and confusion matrices
  - Comments explaining your code

Important Requirements:

- Multinomial Logistic Regression MUST be implemented from scratch
- SVM, Decision Tree, and XGBoost may use sklearn/xgboost libraries
- Use GridSearchCV with k-fold cross-validation for hyperparameter tuning
- Split data into 80% train, 20% test only once at the beginning
- Do not include datasets in submission
- Ensure notebook runs without errors from top to bottom

Submit via: submit.cs.hacettepe.edu.tr

5 Academic Integrity

All work must be done individually. Sharing code or using AI-generated code for required from-scratch implementations will result in severe penalties.