CEng499-HW3

Selim Tarık ARI

2467611

1 Part 1 - Decision Trees and Random Forests

1.1 Introduction

Decision trees are widely used machine learning models due to their interpretability and simplicity. The ID3 algorithm constructs a decision tree by recursively splitting the dataset based on a given criterion, such as information gain or gain ratio. This section describes the implementation of the ID3 algorithm and evaluates its performance on a classification task.

1.2 Implementation

The ID3 algorithm was implemented in Python, with the following structure:

- **TreeNode and TreeLeafNode Classes**: These classes represent non-leaf and leaf nodes, respectively, in the decision tree.
- **DecisionTree Class**: This class includes methods for calculating entropy, average entropy, information gain, gain ratio, and recursively constructing the tree using the ID3 algorithm. Key methods include:
  - `calculate_entropy_`: Computes the entropy of a dataset. This function takes the labels of the data instances, counts their occurrences, calculates the probabilities, and uses them to compute the entropy based on the formula: $H = -\sum p \log_2 p$, where p represents the probability of each class.
  - `calculate_average_entropy_`: Computes the average entropy for a specific attribute by partitioning the dataset based on the attribute's unique values, calculating entropy for each subset, and weighting it by the subset size.
  - `calculate_information_gain_`: Calculates the information gain for a specific attribute by subtracting the average entropy (calculated for the attribute) from the total entropy of the dataset.
  - `calculate_intrinsic_information_`: Computes the intrinsic information of an attribute using its value distribution, similar to entropy calculation.
  - `calculate_gain_ratio_`: Determines the gain ratio by dividing the information gain by the intrinsic information of an attribute, avoiding overfitting to attributes with many values.
  - `ID3_`: Recursively builds the decision tree by selecting the attribute with the highest score (information gain or gain ratio) and creating branches for each attribute value. The process stops when all data belongs to one class or all attributes are used.
  - `predict`: Predicts the class label for a given data instance by traversing the tree based on attribute values. For leaf nodes with multiple labels, the majority class is returned.
  - `train`: Initializes the recursive tree-building process by calling `ID3_` with the full dataset.

The code was executed with the following setup:

- **Dataset**: A predefined dataset representing animals, with 16 discrete features (e.g., hair, feathers, legs).
- **Labels**: Categories of animals (e.g., mammal, bird, reptile).
- **Criterion**: Splits were based on the `gain ratio` criterion.

1.3 Experimental Setup

The ID3 algorithm was trained and tested on the same dataset. The training process involved recursively partitioning the dataset and selecting attributes that maximized the gain ratio. Predictions were made using the constructed tree.

1.4 Results

The model achieved an accuracy of 100% on the provided dataset. The following output was obtained:

```
Splitting on attribute 'milk' using 'gain ratio' with score 1.0000
Splitting on attribute 'fins' using 'gain ratio' with score 1.0000
Splitting on attribute 'feathers' using 'gain ratio' with score 1.0000
Splitting on attribute 'backbone' using 'gain ratio' with score 1.0000
Splitting on attribute 'airborne' using 'gain ratio' with score 0.6074
Splitting on attribute 'predator' using 'gain ratio' with score 0.3449
Splitting on attribute 'legs' using 'gain ratio' with score 1.0000
Splitting on attribute 'tail' using 'gain ratio' with score 0.6074
Splitting on attribute 'aquatic' using 'gain ratio' with score 0.3449
Splitting on attribute 'eggs' using 'gain ratio' with score 1.0000

Training completed
Accuracy : 100.00
```

This result indicates that the decision tree perfectly classified the dataset, likely due to its deterministic nature and the structure of the data.

1.5 Conclusion

The implementation of the ID3 algorithm successfully constructed a decision tree capable of achieving perfect accuracy on the given dataset. This demonstrates the algorithm's effectiveness for this classification task, particularly when data is clean and well-structured.

2 Part 2 - Data Preprocessing, SVM, Kernel Functions

2.1 Dataset 1 - Support Vector Machine Decision Boundaries

2.1.1 Introduction

Support Vector Machines (SVM) are versatile supervised learning algorithms used for classification tasks. This subsection explores the impact of different hyperparameter configurations on the decision boundaries and accuracy of an SVM model trained on a synthetic dataset.

2.1.2 Experimental Setup

The dataset was split into training (70%) and testing (30%) subsets. The following configurations were tested:

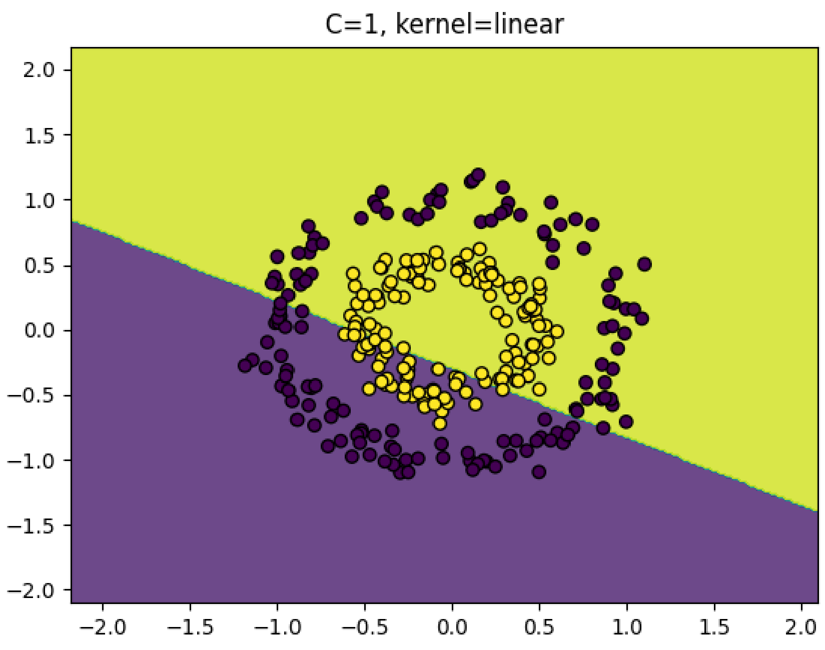
- Linear kernel with $C = 1$ and $C = 10$
- RBF kernel with $C = 1$ and $C = 10$

The decision boundaries were plotted for the training data, and the models were evaluated on the test data.

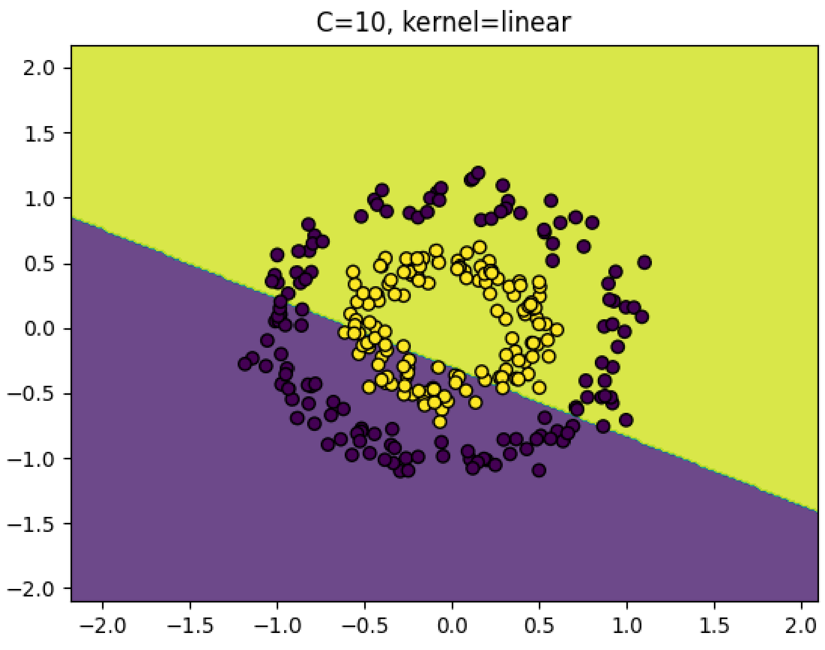
2.1.3 Results

The following results and decision boundary visualizations were obtained:

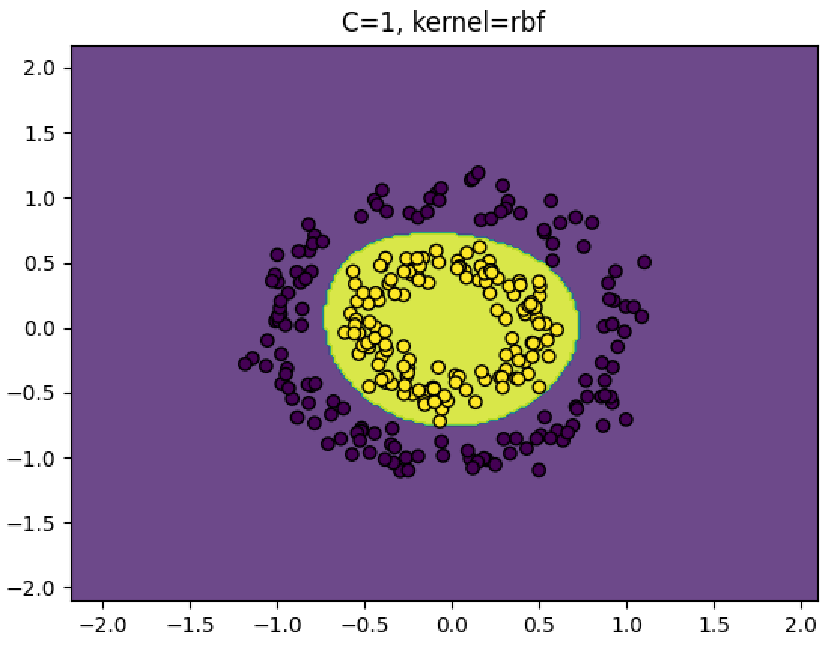
- **Configuration:** $C = 1$, **kernel = linear**
  Accuracy: 0.47



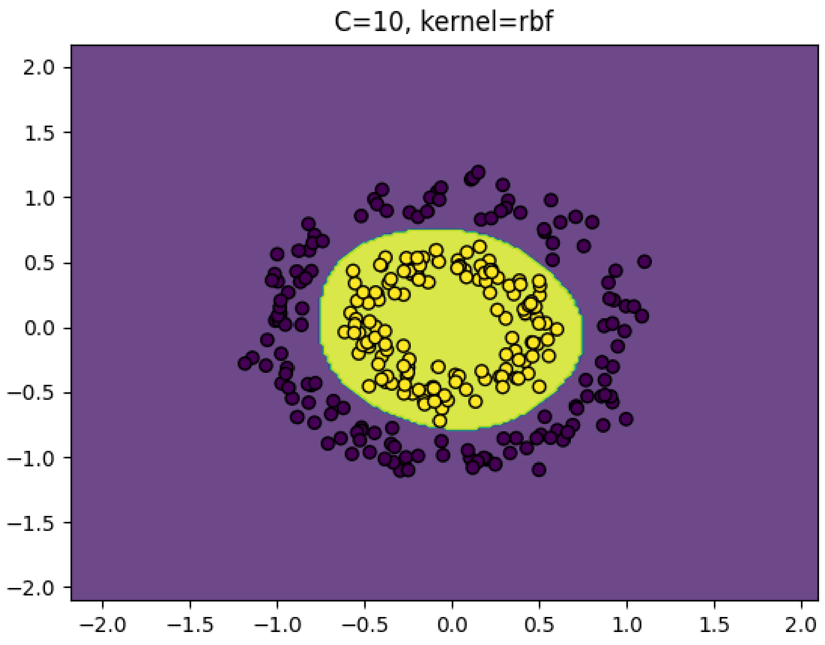
- **Configuration:** $C = 10$, **kernel = linear**
  Accuracy: 0.47



- **Configuration:** $C = 1$, **kernel = RBF**
  Accuracy: 1.00



- **Configuration:** $C = 10$**, kernel = RBF**
  Accuracy: 1.00



2.1.4 Conclusion

The SVM model achieved perfect accuracy (1.00) with the RBF kernel for both $C = 1$ and $C = 10$, demonstrating the kernel's ability to handle non-linear decision surfaces. The linear kernel configurations struggled, achieving only 47% accuracy, reflecting the dataset's non-linear separability.

2.2 Dataset 2 - Hyperparameter Tuning and Confidence Intervals

2.2.1 Introduction

This subsection evaluates an SVM model on a second dataset, focusing on hyperparameter optimization through cross-validation and estimating confidence intervals for performance metrics.

2.2.2 Experimental Setup

The dataset was preprocessed using the StandardScaler for feature standardization. Hyperparameter tuning was conducted using GridSearchCV with the following configurations:

- C: [0.1, 1, 10]
- Kernel: [linear, rbf]

The process was repeated for 5 iterations with 10-fold StratifiedKFold cross-validation, ensuring robust evaluation.

2.2.3 Results

The mean accuracy, standard deviation, and 95% confidence intervals for each configuration are as follows:

C	**Kernel**	**Mean Accuracy**	**Std. Dev.**	**Lower Bound**	**Upper Bound**
0.1	Linear	0.743	0.0041	0.7394	0.7466
0.1	RBF	0.700	0.0000	0.7000	0.7000
1	Linear	0.7462	0.0041	0.7426	0.7498
1	RBF	0.7618	0.0033	0.7589	0.7647
10	Linear	0.7452	0.0043	0.7414	0.7490
10	RBF	0.7524	0.0072	0.7461	0.7587

Table 1: SVM Hyperparameter Tuning Results and Confidence Intervals

2.2.4 Conclusion

The RBF kernel with $C = 1$ achieved the highest mean accuracy of 76.18%, demonstrating its capability to model complex decision boundaries. The linear kernel configurations showed slightly lower performance, highlighting the importance of kernel choice for this dataset. The confidence intervals provide insights into the robustness of each configuration's performance.

3 Part 3 - Method Comparison

3.1 Introduction

This section compares six machine learning algorithms on an ECG dataset to classify heart conditions into five categories. The comparison was performed using the nested cross-validation technique to ensure statistically reliable results. Metrics such as F1 score, standard deviation, and 95% confidence intervals were calculated for each model.

3.2 Experimental Setup

The dataset consisted of ECG measurements with 187 numeric features, labeled with five distinct heart conditions. The data was preprocessed using min-max normalization. Nested cross-validation was employed as follows:

- **Outer loop:** RepeatedStratifiedKFold with 3 splits and 5 repetitions.
- **Inner loop:** RepeatedStratifiedKFold with 5 splits and 5 repetitions.

The following algorithms and hyperparameter grids were evaluated:

- **KNN:** `n_neighbors` = {3, 5}.
- **SVM:** `C` = {0.1, 1}, `kernel` = {`linear`, `rbf`}.
- **Decision Tree:** `max_depth` = {5, 10}.
- **Random Forest:** `n_estimators` = {50, 100}.
- **MLP:** `hidden_layer_sizes` = {(50,), (100,)}.
- **Gradient Boosting:** `n_estimators` = {50, 100}.

Each randomized algorithm was executed five times per test-train split to account for variability.

3.3 Results

The results of the nested cross-validation procedure are summarized in Table 2.

Algorithm	**Mean F1 Score**	**Std. Dev.**	**95% Confidence Interval**
KNN	0.8441	0.0093	(0.8394, 0.8488)
SVM	0.8265	0.0092	(0.8219, 0.8312)
Decision Tree	0.7971	0.0083	(0.7929, 0.8013)
Random Forest	0.8746	0.0094	(0.8698, 0.8793)
MLP	0.8210	0.0125	(0.8147, 0.8274)
Gradient Boosting	0.8437	0.0107	(0.8383, 0.8491)

Table 2: Nested cross-validation results for Part 3.

3.4 Discussion

The performance of each algorithm highlights its strengths and weaknesses for the ECG dataset:

- **Random Forest:** Achieved the highest mean F1 score (0.8746). Its ensemble approach effectively reduces overfitting by averaging predictions across multiple trees, capturing complex patterns in the data.
- **KNN:** Performed well (0.8441 F1 score) due to its simplicity and ability to classify based on local neighborhood relationships. However, it may struggle with high-dimensional data like this dataset if feature scaling is inconsistent.
- **Gradient Boosting:** Achieved similar performance to KNN (0.8437). Gradient Boosting's iterative learning process allows it to focus on misclassified examples, but it can be sensitive to hyperparameter settings.
- **SVM:** While effective (0.8265 F1 score), SVM's performance depends heavily on kernel choice and hyperparameters. The RBF kernel helped capture non-linear relationships in the dataset.
- **MLP:** The neural network achieved an F1 score of 0.8210, showcasing its ability to learn complex representations. However, it is sensitive to initialization and hyperparameters, which may explain its variability.
- **Decision Tree:** Scored the lowest (0.7971) due to its simplicity and tendency to overfit small datasets. Unlike Random Forest, it lacks ensemble techniques to improve robustness.

3.5 Conclusion

The results indicate that ensemble methods like Random Forest and Gradient Boosting excel in handling this dataset, likely due to their ability to aggregate decisions from multiple models. KNN's performance also highlights its effectiveness when data is appropriately normalized. The variability in MLP and Gradient Boosting results suggests potential sensitivity to hyperparameter tuning and data partitioning. Future work could explore additional algorithms or more extensive hyperparameter tuning to further improve classification performance.