Machine Learning

Homework 3

Ensemble Learning

Supervisor:

Dr. Hashemi

Ali Mahmoodi

Atefe Rajabi

Spring 2024

Index

Problem Definition:

The project aims to implement and evaluate ensemble learning methods, specifically Bagging and Boosting, to address class imbalance in the Abalone dataset. These techniques will combine multiple models to enhance predictive performance and generalizability.

1. Class Imbalance: The dataset exhibits a significant class imbalance where one class (minority or positive class) contains much fewer samples than the other (majority or negative class). This imbalance can lead to models that are biased towards the majority class, resulting in poor predictive performance for the minority class.

2. Ensemble Learning Implementation:

- Bagging: Implement bagging techniques that involve constructing multiple models (also known as base learners) from different subsets of the training dataset.
- Boosting: Develop boosting methods where base learners are built sequentially by focusing on incorrectly predicted instances by the previous models.

3. Model Performance Metrics:

- Evaluate models using the Average Area Under the Receiver Operating Characteristic Curve (AUROC) and the Average Precision-Recall (AUPR). These metrics are crucial for assessing model performance in the context of imbalanced datasets.

Methodology:

1. Dynamic Local Neighborhood Calculation: Calculate the local neighborhood dynamics to adjust sampling strategies based on local densities.

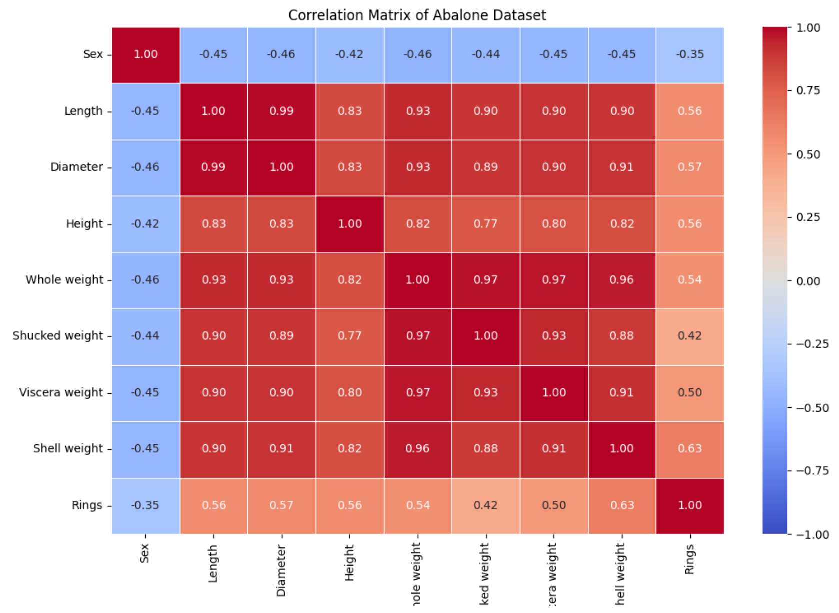
2. Risk Assessment for Synthetic Samples: Using techniques like SMOTE, evaluate risks associated with synthetic samples positioned in the majority class regions.

3. Sample Generation and Balancing: Generate new samples based on calculated probabilities to balance the dataset effectively before applying ensemble methods.

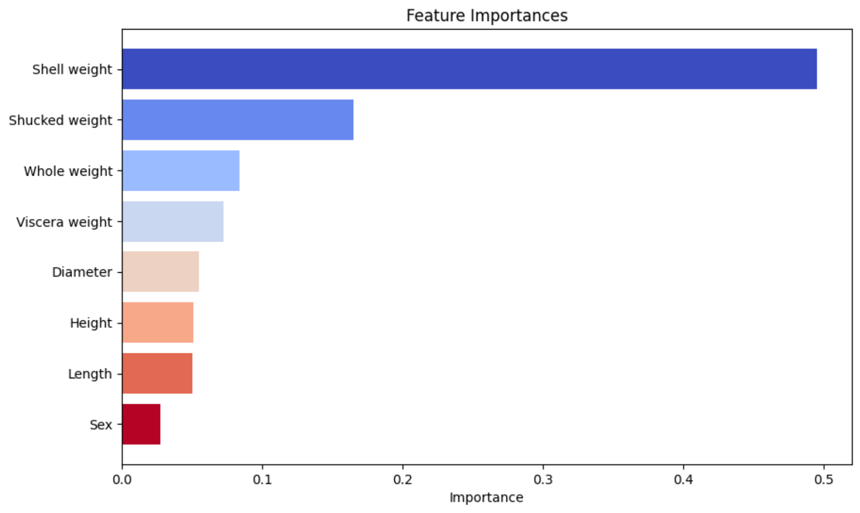
4. Weak Learner Updates: Update weak learners in boosting using a specified procedure that accounts for changes in data distribution and model focus.

Data Exploration:

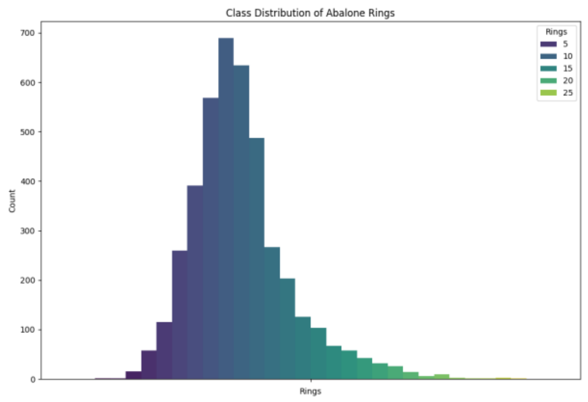
Feature Correlation:



Feature Importance:



Data Balance:



Class Imbalance:

While there is a broad range of ring counts, the distribution is not heavily imbalanced. There are fewer abalones with very low or very high ring counts.

The dataset contains sufficient examples of abalones across different age ranges to potentially build robust predictive models.

Implementation Details:

‘NearestNeighbors’

This class is an implementation of the nearest neighbors algorithm:

- ‘fit(data)’: Stores the data upon which distances will be calculated later.

- ‘get_neighbors(points, n_neighbors=1)’: Calculates the Euclidean distance from each point in a provided set to all points in the fitted dataset, returning the distances and indices of the ‘n_neighbors’ closest points.

‘SMOTE’ (Synthetic Minority Over-Sampling Technique)

This class is used for handling imbalanced datasets by creating synthetic samples:

- ‘fit(data, labels)’: Fits the model to the data and labels, identifying the minority and majority classes.

- ‘r1_risk(data, labels)’: Computes the ratio of same-class neighbors for each data point.

- ‘_intra_extra(data, labels)’: Calculates a measure combining intra-class and extra-class distances for risk assessment.

- ‘r2_risk(data, labels)’: Computes a normalized risk score based on intra-class and extra-class distances.

- ‘risk_score(data, labels)’: Adjusts the risk scores based on a threshold.

- ‘cls_score(data, labels, weights, iteration, max_iter)’: Computes a score for each sample based on class similarity and iteration-based weights.

- ‘sample_prob(data, labels, weights, iteration, max_iter)’: Computes probabilities for sampling based on class scores and risk scores.

- ‘resample()’: Generates new samples from the minority class to balance the dataset.

‘Bagging’

This class implements the bagging ensemble technique:

- ‘fit(data, labels)’: Fits multiple instances of a base model to random subsets of the data.

- ‘predict(data)’: Aggregates predictions from all the fitted models using the mode (most frequent label).

- ‘predict_prob(data)’: Calculates the average probability across all models.

- ‘score(data, labels)’: Evaluates the accuracy of the ensemble model.

‘Boosting’

This class implements a boosting ensemble method:

- ‘fit(data, labels)’: Trains models sequentially, each time fitting to a modified version of the data that focuses on previously misclassified examples by adjusting their weights. SMOTE is used for handling class imbalances in each iteration.

- ‘predict(data)’: Computes final predictions by aggregating weighted votes from all models.

- ‘predict_prob(data)’: Calculates probabilities by weighting predictions from all models.

- ‘score(data, labels)’: Calculates the accuracy of the ensemble model based on final predictions.

Bagging Implementation:

Bagging with Decision Trees: A Bagging ensemble is set up using Decision Trees as the base model. The ensemble's method could be set to either "CART" or "C4.5", indicating the use of different decision tree algorithms.

Model Training: The Bagging ensemble is trained on the resampled and split data.

Performance Evaluation: It uses cross-validation to evaluate the performance of the ensemble across different metrics like accuracy, precision, and recall.

Boosting Implementation:

Boosting with Decision Trees: Similarly, to Bagging, Boosting is used with Decision Trees. Boosting focuses on converting weak learners into strong ones through a weighted process where subsequent models focus more on previously misclassified examples.

Model Training and Prediction: After training, predictions are made on the validation data. The document also demonstrates calculating the ROC AUC score, which helps in evaluating the classifier's performance across all classification thresholds.

Cross-validation:

Cross-validation is a robust statistical method used primarily to evaluate the generalization capabilities of a model on unseen data. It's especially critical in scenarios where it's crucial to ensure that the model performs well across different subsets of data and isn't just fitted to a particular set of data points. Here are the main reasons why cross-validation is used:

1. Model Validation:

- Avoid Overfitting: Cross-validation helps in verifying that a model does not overfit to the training data. Overfitting occurs when a model learns the details and noise in the training data to an extent that it negatively impacts the performance of the model on new data.

- Model Stability: It assesses the stability of the model's predictions across different data samples. This is crucial for ensuring that the model can handle variability in data effectively.

2. Model Selection:

- Tuning Hyper-parameters: Cross-validation is used to compare the performances of different models or the same model with different configurations. This is essential for fine-tuning model parameters and selecting the best-performing model or configuration.

- Choosing Features: It can also help in determining which features or variables are contributing most to the predictive power of the model, allowing for optimization of the input features.

3. Performance Estimation:

- Unbiased Estimation: It provides a less biased or more accurate estimate of the model effectiveness compared to a simple train/test split. This is due to the use of multiple train-test splits, ensuring that the model's performance is tested across all available data.

- Robust Metrics: By aggregating the outcomes of multiple train-test splits, it delivers robust performance metrics, which are crucial for assessing how well the model is likely to perform in real-world tasks.

4. Efficient Use of Data:

- Maximize Data Usage: In scenarios where the amount of data is limited, cross-validation maximizes the use of available data. This is done by repeatedly using different subsets of the data for training and validation.

- Reduce Bias: By rotating the validation set across different parts of the dataset, cross-validation reduces the risk of model bias towards any specific portion of the data.

5. Statistical Significance:

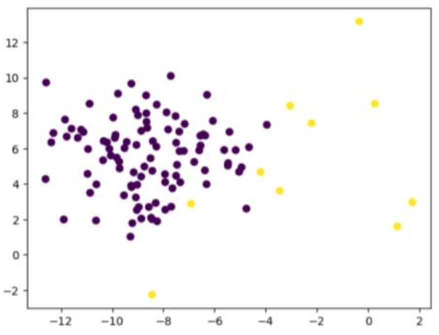
- Statistical Analysis: The results from multiple folds of cross-validation can be used to perform statistical tests to compare different models. This helps in understanding if differences in performance are statistically significant.

In ensemble learning, as described in our project, cross-validation is particularly important because it ensures that the ensemble model, which combines predictions from multiple models, is robust and performs consistently across different subsets of the dataset. This is crucial when handling tasks like classifying imbalanced data, where every data point's correct classification is vital.

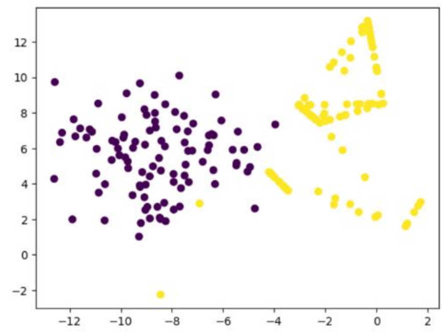
Result:

Resample test:

before smote resampling



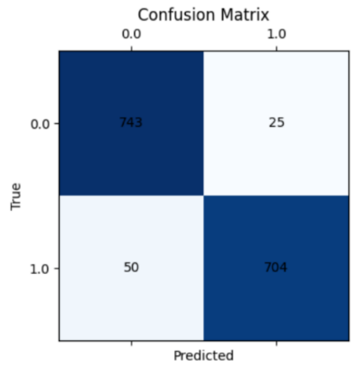
after smote resampling

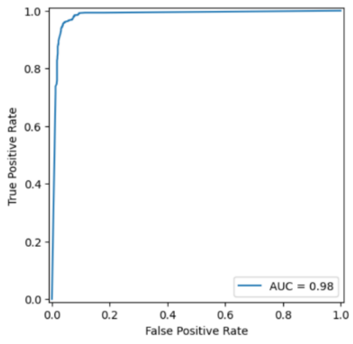


Bagging:

CART

```
Accuracy: 0.9507
Precision: 0.9506
Recall: 0.9513
```





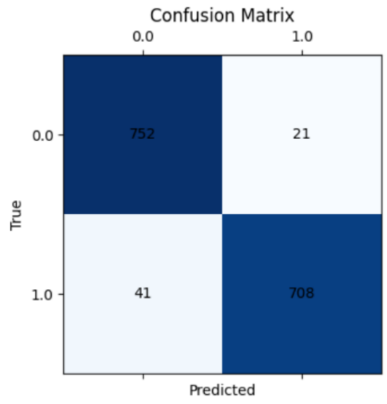
AUPR: 0.13738250566061713

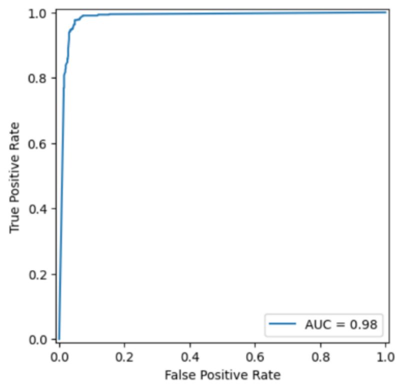
Cross validation

{'fit_time': array([107.55995584, 92.57110763, 95.7877667 , 107.84571528, 95.42621183]),
'score_time': array([0.09782887, 0.09474659, 0.09275198, 0.14860153, 0.10771203]),
'test_accuracy': array([0.95069844, 0.95398521, 0.95234182, 0.94823336, 0.94325658]),
'test_precision_macro': array([0.95128431, 0.95437422, 0.95267909, 0.94851026, 0.94430144]),
'test_recall_macro': array([0.95042846, 0.95410423, 0.95180889, 0.9487527 , 0.94282853])}

C4.5

```
Accuracy: 0.9593
Precision: 0.9590
Recall: 0.9597
```



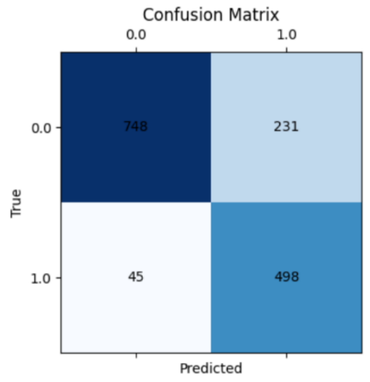


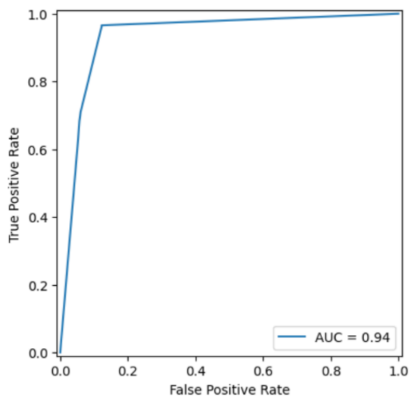
AUPR: 0.12440021964676642

Boosting:

CART

```
Accuracy: 0.8187
Precision: 0.8406
Recall: 0.8132
```





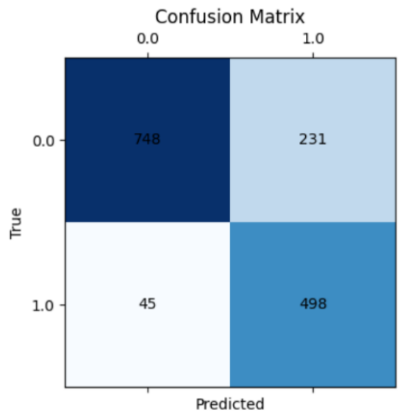
AUPR: 0.08202112305551795

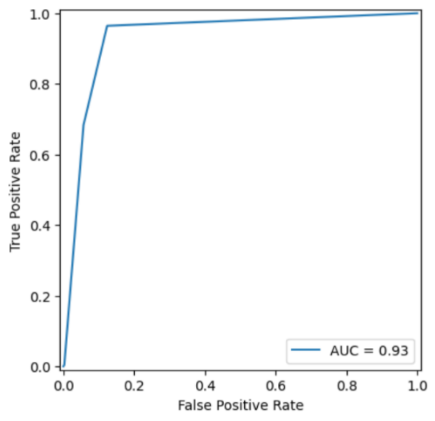
Cross validation

```
{'fit_time': array([28.86337137, 25.53080964, 26.34652448, 25.5741055 , 26.26956868]),
 'score_time': array([0.00598431, 0.00598383, 0.01396322, 0.00598407, 0.00598407]),
 'test_accuracy': array([0.8085456 , 0.79046836, 0.76910435, 0.74691865, 0.79276316]),
 'test_precision_macro': array([0.8347205 , 0.82332929, 0.80471668, 0.79239579, 0.82409033]),
 'test_recall_macro': array([0.81094224, 0.78928406, 0.77740008, 0.74142568, 0.79579996])}
```

C4.5

```
Accuracy: 0.8187
Precision: 0.8406
Recall: 0.8132
```



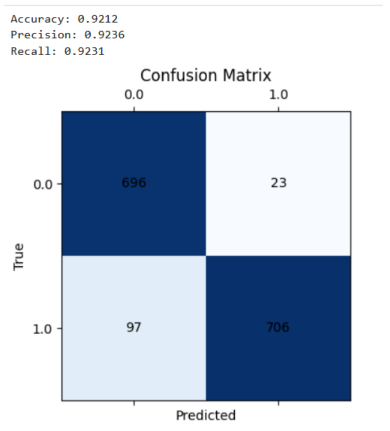


AUPR: 0.02855813404015506

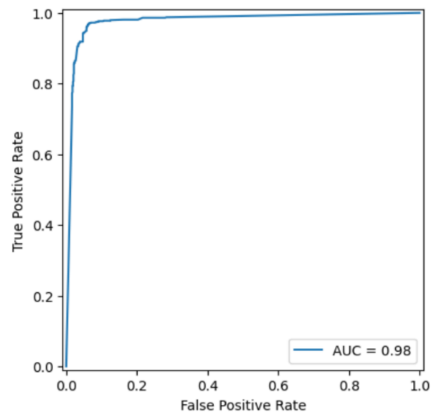
Noise Generating:

Bagging – CART

AUPR: `0.455599637728501`

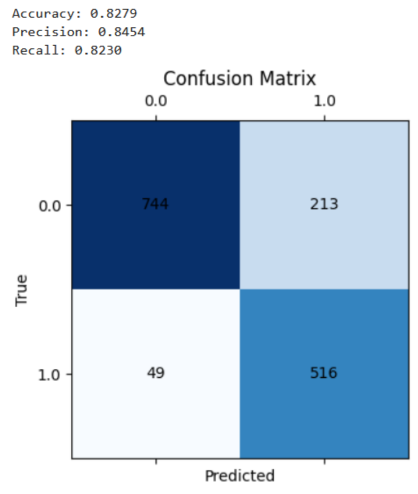


```
Accuracy: 0.9212
Precision: 0.9236
Recall: 0.9231
```

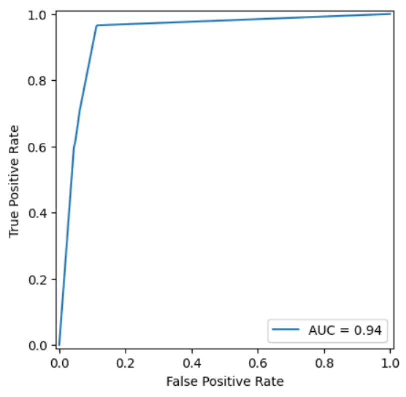


Boosting – CART

AUPR: `0.07764266573332632`



```
Accuracy: 0.8279
Precision: 0.8454
Recall: 0.8230
```

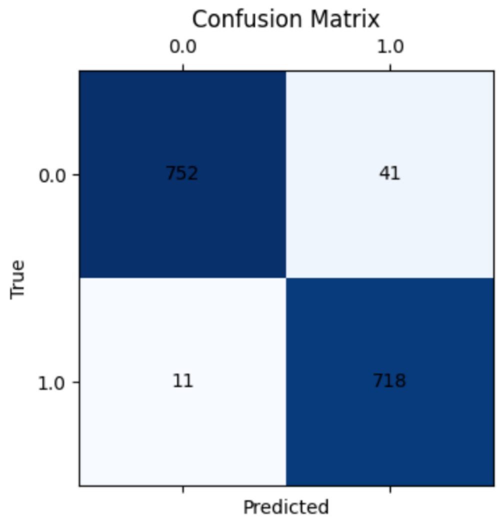


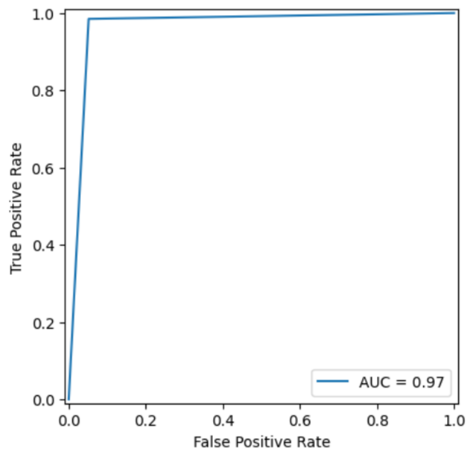
Random Forest

Scikit learn classifier

```
AUPR: 0.0
```

```
Accuracy: 0.9658
Precision: 0.9666
Recall: 0.9658
```





Discussion:

Bagging Results

- Models Used: Decision Trees (CART and C4.5).

- Performance Metrics: AUPR values of 0.137 and 0.124 for CART and C4.5, respectively.

- Cross-validation Results: For CART, accuracy scores ranged between 0.943 and 0.954, indicating strong performance. Precision and recall metrics were similarly high.

Boosting Results

- Models Used: Decision Trees (CART and C4.5).

- Performance Metrics: AUPR values significantly lower for boosting (0.082 for CART and 0.029 for C4.5) compared to bagging.

- Cross-validation Results: Boosting with CART showed lower performance with accuracy scores between 0.746 and 0.808. Precision and recall also showed variability but were generally lower than bagging results.

1. Model Effectiveness:

   - Bagging proved to be more effective with higher AUPR and better cross-validation results, suggesting it managed the class imbalance more effectively.

   - Boosting, while aiming to convert weak learners into strong ones, showed lower performance metrics, possibly indicating issues like overfitting or insufficient parameter tuning.

2. Model Characteristics:

   - Bagging helps in reducing variance and avoiding overfitting, making it suitable for complex datasets like the one used.

   - Boosting is sensitive to noise and can overfit, particularly with complex models such as C4.5.

3. Technical Insights:

   - The choice of base models (CART vs. C4.5) influenced the results. CART generally showed better performance in both bagging and boosting, possibly due to its simpler structure, which is more manageable in an ensemble setup.

- The lower performance of boosting in this study might be due to the inherent noise in the dataset or suboptimal tuning of the boosting parameters.

4. Speed:

In bagging, each base model (e.g., a decision tree in this case) is trained independently on a different bootstrap sample of the data. If these base models are decision trees allowed to grow without constraints (fully grown trees), each tree could potentially become very complex and deep, especially if the data has many features and complex patterns. This can lead to longer training times because each tree in the ensemble might be executing extensive computations to make as many splits as needed to perfectly classify its bootstrap sample.

5. Sensitivity to Model Parameters:

Bagging:

Moderately Sensitive: While the individual model complexity can affect performance, bagging is generally less sensitive to the parameters of its base models compared to boosting. As long as the base models are not too weak, bagging tends to perform well.

Boosting:

Highly Sensitive: The performance of boosting is heavily dependent on the settings of parameters such as the number of boosting rounds and the learning rate. The choice of loss function and the depth of the trees (or complexity of whatever base model is used) are also crucial. Poor parameter choices can lead to underfitting or overfitting.

In conclusion, bagging was more successful in this implementation, providing higher reliability and robustness against the dataset's complexities and class imbalances. Boosting's lower performance indicates a potential for overfitting and sensitivity to noise, suggesting that parameter optimization and model selection are critical for its success.