Machine Learning – Tutorial

Gradient Boosting for Classification

Kokila Sivakumar – 24058234

Gradient Boosting

Introduction

Gradient Boosting is a powerful ensemble learning technique used for both classification and regression. It works by combining many weak learners most commonly shallow decision trees into a strong predictive model. Each new learner is trained to correct the mistakes of the previous ones, allowing the model to steadily improve its accuracy.

In this tutorial, we develop and evaluate a Gradient Boosting model using scikit-learn's GradientBoostingClassifier on the Breast Cancer dataset. We compare it with a baseline Decision Tree model, tune hyperparameters using GridSearchCV, and analyze the final performance.

Boosting

Boosting is a special type of Ensemble Learning technique that works by combining several weak learners (predictors with poor accuracy) into a strong learner (a model with strong accuracy). This works by each model paying attention to its predecessor's mistakes.

The two most popular boosting methods are:

- Adaptive Boosting
- Gradient Boosting

We will be discussing Gradient Boosting.

What is Gradient Boosting?

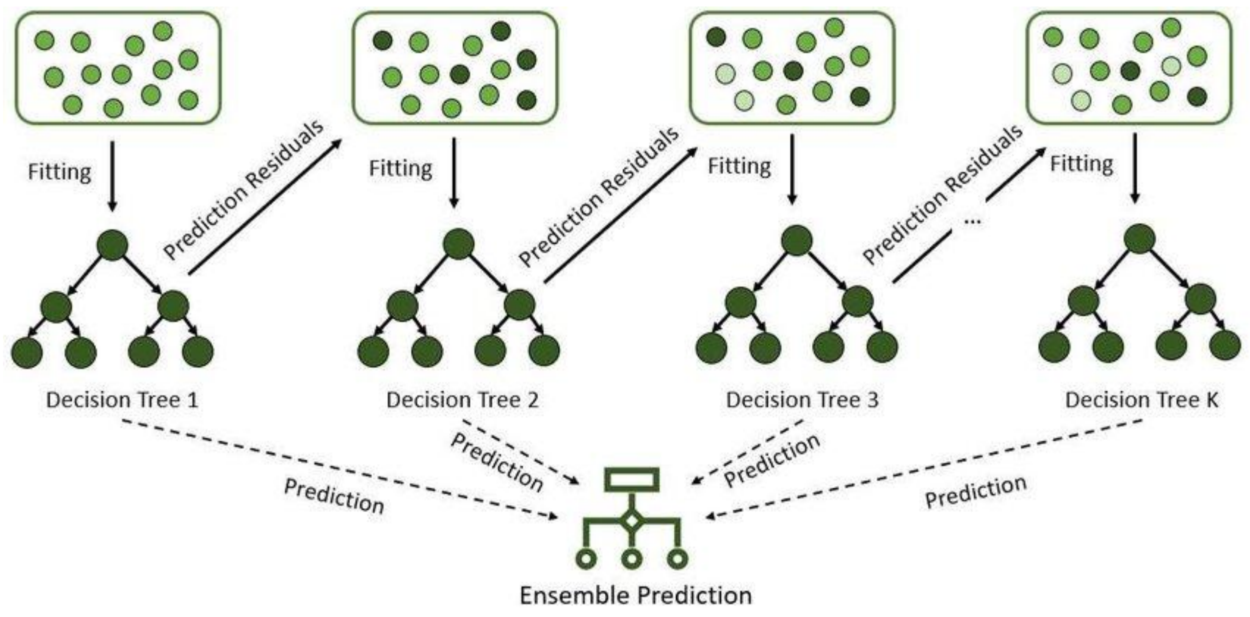
Gradient Boosting is a highly effective ensemble learning method that consistently performs well on a wide range of classification problems. It has gained popularity because it can handle complicated datasets, capture subtle patterns, and deliver highly accurate predictions. For anyone aiming to strengthen their machine learning skills, understanding how Gradient Boosting works is an essential step.

In this guide, we focus on building Gradient Boosting models using **scikit-learn's GradientBoostingClassifier**. We will look at the main ideas behind the algorithm, explore the key parameters that influence its behavior, and walk through a practical example to show how the model is trained and evaluated.

Gradient Boosting constructs a strong predictive model by combining many weaker models—typically small decision trees. Unlike methods such as Random Forests, which generate multiple trees independently, Gradient Boosting builds trees one at a time, with each new tree specifically trained to correct the errors made by the previous ones. Over multiple iterations, this approach steadily improves the model's performance.

How Gradient Boosting Works

The fundamental concept behind Gradient Boosting is an iterative learning process where each new model focuses on the errors left unresolved by earlier models.



Here's how it works:

1. **Start with a simple baseline model**
   Usually this is just a constant value, such as the mean prediction for the dataset.
2. **Compute the residuals (errors)**
   These residuals represent the gap between the model's predictions and the true target values.
3. **Train a weak learner on the residuals**
   A small decision tree is fitted to explain these errors rather than the original target.
4. **Update the overall model**
   The predictions from this new tree are added to the existing model to reduce the remaining error.
5. **Repeat the process**
   More trees are added sequentially, each improving the model slightly until a stopping condition is reached (such as a fixed number of iterations or no further improvement).

This step-by-step correction mechanism allows Gradient Boosting to steadily reduce prediction errors and ultimately produce a powerful and accurate model.

Key Parameters in Gradient Boosting (Rewritten)

Scikit-learn, a popular machine learning library in Python, provides a robust implementation of Gradient Boosting through its GradientBoostingClassifier. This class allows you to easily apply this powerful algorithm to your classification problems.

To use Gradient Boosting effectively, it's important to understand the main parameters that shape how the algorithm learns.

1. **n_estimators**

This defines the number of boosting stages (trees) added to the model.

- More trees: potentially higher accuracy
- Anyway too many trees have risk of overfitting
- Common range: **100–300**

2. **learning_rate**

Controls how much influence each new tree has on the final model.

- Small learning rate: slower but more stable learning
- Large learning rate : faster training but may overfit
- Usually set between **0.01 and 0.1**

The learning rate and number of trees work together. A smaller learning rate often requires more trees.

3. **max_depth**

Determines the depth of each decision tree.

- Shallow trees (depth 1–3) are preferred
- They prevent overfitting and allow gradual learning

4. **subsample**

Fraction of the dataset used to train each tree.

- Value < 1 introduces randomness
- Helps prevent overfitting
- Example: **subsample = 0.8**

Together, these parameters allow you to fine-tune the trade-off between accuracy, speed, and generalization.

Step-by-Step: Fitting a GradientBoostingClassifier

1. Prepare the Data

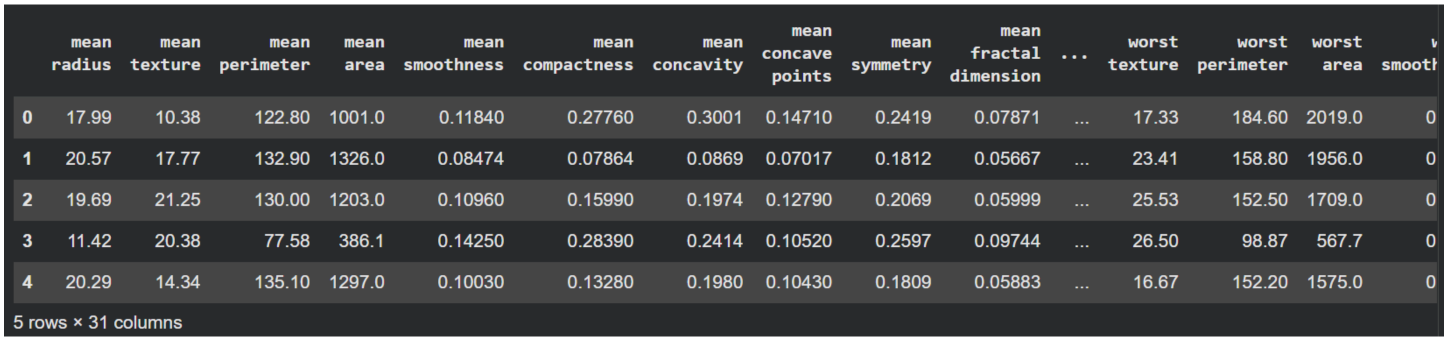
- Import Libraries and Dataset

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

- Load the Dataset

```
data = load_breast_cancer()
X = data.data
y = data.target
df = pd.DataFrame(X, columns=data.feature_names)
df['target'] = y
df.head()
```

```
Output:
```



	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	smoot
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0

5 rows × 31 columns

- Split the Dataset

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Training shape:", X_train.shape)
print("Testing shape:", X_test.shape)
```

OUTPUT:

```
Training data shape: (455, 30)

Testing data shape: (114, 30)
```

2. Initialize Decision Tree and Evaluate Performance

```
tree = DecisionTreeClassifier(max_depth=3, random_state=42)
tree.fit(X_train, y_train)

y_train_pred_tree = tree.predict(X_train)
y_test_pred_tree = tree.predict(X_test)

print("=== Decision Tree Performance ===")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred_tree))
print("Test Accuracy :", accuracy_score(y_test, y_test_pred_tree))
```

OUTPUT:

```
=== Decision Tree Performance ===

Train Accuracy: 0.9758241758241758

Test Accuracy : 0.9385964912280702
```

This Decision tree give 97 % accuracy for training and 93% accuracy for testing data set.

3. Gradient Boosting Classifier

```
gbClassifier = GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=3,
    random_state=42
)

gbClassifier.fit(X_train, y_train)

y_train_pred_gb = gbClassifier.predict(X_train)
y_test_pred_gb = gbClassifier.predict(X_test)

print("=== Gradient Boosting Performance ===")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred_gb))
print("Test Accuracy :", accuracy_score(y_test, y_test_pred_gb))
print("\nClassification Report:\n", classification_report(y_test,
y_test_pred_gb))
```

OUTPUT:

```
=== Gradient Boosting Performance ===
Train Accuracy: 1.0
Test Accuracy : 0.956140350877193

Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.90      0.94        42
           1       0.95      0.99      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

This Decision tree give 100% accuracy for training and 95% accuracy for testing data set.

Gradient Boosting delivered even higher accuracy and more balanced precision/recall metrics by sequentially correcting the errors of previous weak learners. Its ability to reduce bias and improve generalization resulted in a higher test accuracy (≈95.6%) compared to the Decision Tree (≈93.9%)

4. Hyper Parameter Tuning using GridSearchCV

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [2, 3, 4],
    'subsample': [0.8, 1.0]
}

gb = GradientBoostingClassifier(random_state=42)

grid = GridSearchCV(
    estimator=gb,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1
)

grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best CV Score:", grid.best_score_)
```

OUTPUT:

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 
300, 'subsample': 0.8}

Best Cross-Validation Accuracy: 0.9780219780219781
```

5. Final Model Evaluation

```
best_model = grid.best_estimator_
best_model.fit(X_train, y_train)

y_pred_best = best_model.predict(X_test)
```

```
print("=== Final Gradient Boosting Performance ===")
print("Train Accuracy:", best_model.score(X_train, y_train))
print("Test Accuracy :", accuracy_score(y_test, y_pred_best))
print("\nClassification Report:\n", classification_report(y_test,
y_pred_best))
```

OUTPUT:

```
=== Final Gradient Boosting Performance ===

Train Accuracy: 1.0

Test Accuracy : 0.956140350877193


Classification Report:

               precision    recall  f1-score   support


           0       0.97      0.90      0.94        42

           1       0.95      0.99      0.97        72


    accuracy                           0.96       114

   macro avg       0.96      0.95      0.95       114

weighted avg       0.96      0.96      0.96       114
```

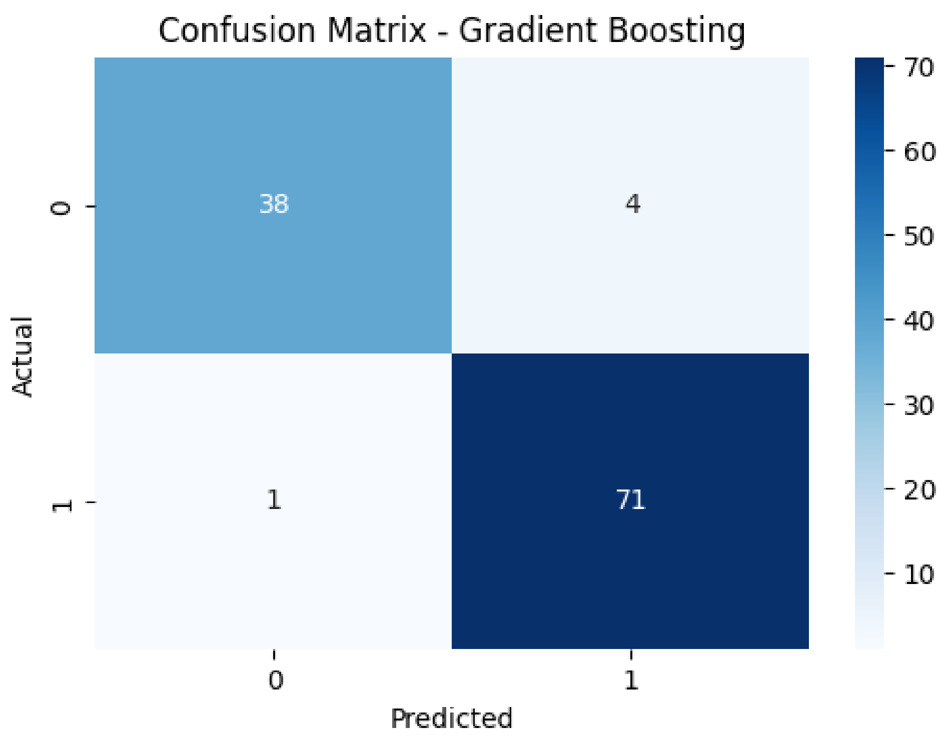
After applying hyperparameter tuning, the model achieved a cross-validation accuracy of 0.978, with final test accuracy remaining stable at 0.956, indicating excellent generalization. The classification report shows high precision, recall, and F1-score for both classes

6. Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred_best)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Gradient Boosting")
plt.show()
```

OUTPUT:



True Negatives (TN = 38) The model correctly identified 38 benign cases.

False Positives (FP = 4) Four benign cases were incorrectly classified as malignant.

False Negatives (FN = 1) one malignant case was incorrectly predicted as benign.

True Positives (TP = 71) The model correctly predicted 71 malignant cases.

Results Summary

The Gradient Boosting model delivered strong and reliable performance. The tuned model achieved a cross-validation accuracy of 0.978 and a final test accuracy of 95.6%. Only five mistakes were made across the test set:

• True Negatives (38)

• False Positives (4)

• False Negatives (1)

• True Positives (71)

These results show that Gradient Boosting is significantly more accurate than a single decision tree, thanks to its iterative error-correcting learning strategy.

Best Practices and Tips

To get the most out of your GradientBoostingClassifier models:

- **Feature Engineering:** The quality of your features heavily influences model performance. Invest time in creating meaningful features.
- **Cross-Validation:** Always use cross-validation during hyperparameter tuning to get a reliable estimate of your model's performance and prevent overfitting to the validation set.
- **Early Stopping:** GradientBoostingClassifier in scikit-learn doesn't directly support early stopping during fit() as some other libraries do (e.g., XGBoost). However, you can implement it manually by monitoring performance on a validation set and stopping training when performance no longer improves, or use parameters like n_iter_no_change if available in newer versions for specific loss functions.
- **Monitor Overfitting:** Keep an eye on the training vs. validation score. A large gap indicates overfitting. Adjust max_depth, n_estimators, learning_rate, and subsample to mitigate this.

‘

Conclusion

Gradient Boosting proved to be a highly effective classifier for this medical dataset, showing excellent accuracy and robustness. Its ability to learn from residuals and build incrementally stronger learners makes it one of the most powerful machine learning techniques available for structured data problems.

References

Jijo, Bahzad & Abdulazeez, Adnan. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. Journal of Applied Science and Technology Trends. 2. 20-28.

Patil, Dimple & Rane, Nitin & Desai, Pravin & Rane, Jayesh. (2024). Machine learning and deep learning: Methods, techniques, applications, challenges, and future research opportunities. 10.70593/978-81-981367-4-9_2.

P. Lavanya, Rimjhim Padam Singh, U. Kumaran, Priyanka Kumar, Gradient Boosting classifier performance evaluation using Generative Adversarial Networks, Procedia Computer Science, https://doi.org/10.1016/j.procs.2024.04.285.

https://www.geeksforgeeks.org/machine-learning/how-to-tune-hyperparameters-in-gradient-boosting-algorithm/

https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting

GitHub:

https://github.com/KokilaSivakumar06/Machine-Learning-Assignment---Individual