Foundations of Machine Learning

Sakshi Badole - CS24MTECH11008

November 2, 2024

ASSIGNMENT - 3

Questions: Theory

---

1 Non-Uniform Weights in Linear Regression

Weighted Least Squares Solution

Given a dataset with points (x_n, y_n), $n = 1, \cdots, N$, each associated with a weighting factor $\sigma_n > 0$, find the solution $\mathbf{w}^*$ that minimizes the error function:

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \sigma_n (y_n - \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_n))^2$$

where $\mathbf{\Phi}(\cdot)$ is any representation of the data.

Solution

To find the minimum, we take the derivative with respect to $\mathbf{w}$ and set it to zero:

$$\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}} = 0$$

Expanding the derivative:

$$\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{n=1}^{N} \sigma_n (y_n - \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_n)) \mathbf{\Phi}(\mathbf{x}_n)^T = 0$$

Rearranging terms:

$$\sum_{n=1}^{N} \sigma_n \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_n) \mathbf{\Phi}(\mathbf{x}_n)^T = \sum_{n=1}^{N} \sigma_n y_n \mathbf{\Phi}(\mathbf{x}_n)^T$$

Define the weighted covariance matrix:

$$\mathbf{\Sigma} = \sum_{n=1}^{N} \sigma_n \mathbf{\Phi}(\mathbf{x}_n) \mathbf{\Phi}(\mathbf{x}_n)^T$$

And the weighted target vector:

$$\mathbf{v} = \sum_{n=1}^{N} \sigma_n y_n \mathbf{\Phi}(\mathbf{x}_n)$$

Then our equation becomes:

$$\mathbf{w}^{*T}\mathbf{\Sigma} = \mathbf{v}^T$$

Therefore, the solution is:

$$\mathbf{w}^* = \mathbf{\Sigma}^{-1}\mathbf{v} = \left(\sum_{n=1}^{N} \sigma_n \mathbf{\Phi}(\mathbf{x}_n)\mathbf{\Phi}(\mathbf{x}_n)^T\right)^{-1} \left(\sum_{n=1}^{N} \sigma_n y_n \mathbf{\Phi}(\mathbf{x}_n)\right)$$

The solution is indeed a minimum because:

- The second derivative is positive definite (since $\sigma_n > 0$ for all n)
- Hence, the critical point found is a global minimum

The solution has several important properties:

1. It generalizes the standard least squares solution. When all $\sigma_n = 1$, it reduces to the standard least squares solution
2. Points with larger weights σ_n have more influence on the solution. The weights allow for different levels of confidence in different data points

---

2 Neural Network

Cross-Entropy Error Function Derivative Proof

Let's prove that $\frac{\partial E}{\partial a_k} = y_k - t_k$ for the cross-entropy error function with softmax activation.

First, we start with the cross-entropy error function:

$$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(x_n, w)$$

Since we're finding the derivative with respect to a particular a_k for a single input, we can drop the sum over n:

$$E = -\sum_{k} t_k \ln y_k$$

Using the chain rule:

$$\frac{\partial E}{\partial a_k} = -\sum_{i} t_i \cdot \frac{1}{y_i} \cdot \frac{\partial y_i}{\partial a_k}$$

The softmax function:

$$y_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}$$

Derivative components:

1. Direct component: $\frac{\partial y_i}{\partial a_k} = y_k(1 - y_k)$ when $i = k$
2. Cross component: $\frac{\partial y_i}{\partial a_k} = -y_i y_k$ when $i \neq k$

Substituting these back:

$$\frac{\partial E}{\partial a_k} = -\left[t_k \cdot \frac{1}{y_k} \cdot y_k(1 - y_k) + \sum_{i \neq k} t_i \cdot \frac{1}{y_i} \cdot (-y_i y_k) \right]$$

Simplifying:

$$\frac{\partial E}{\partial a_k} = -[t_k(1 - y_k) - y_k \sum_{i \neq k} t_i]$$

Since t is a one-hot vector:

$$\sum_{i \neq k} t_i = 1 - t_k$$

Therefore:

$$\begin{aligned} \frac{\partial E}{\partial a_k} &= -[t_k(1 - y_k) - y_k(1 - t_k)] \\ &= -[t_k - t_k y_k - y_k + y_k t_k] \\ &= -(t_k - y_k) \\ &= y_k - t_k \end{aligned}$$

Thus, we have proven that $\frac{\partial E}{\partial a_k} = y_k - t_k$.

Hence, the gradient is simply the difference between the predicted probability (y_k) and the true label (t_k) for class k.

---

3 Ensemble Methods

Consider a convex function $f(x) = x^2$. Show that the average expected sum-of-squares error

$$E_{AV} = \frac{1}{M} \sum_{m=1}^{M} E_x[(y_m(x) - f(x))^2]$$

of the members of an ensemble model and the expected error

$$E_{ENS} = E_x[(\frac{1}{M} \sum_{m=1}^{M} y_m(x) - f(x))^2]$$

of the ensemble satisfy: $E_{ENS} \leq E_{AV}$

Proof

Jensen's inequality:

- For a convex function ϕ: $\phi(E[X]) \leq E[\phi(X)]$

- For finite sums: $\phi(\sum_i p_i x_i) \leq \sum_i p_i \phi(x_i)$ where $\sum_i p_i = 1$

For the sum-of-squares case:
The squared function is convex. Applying Jensen's inequality:

$$(\frac{1}{M}\sum_{i=1}^{M} y_i(x) - f(x))^2 \leq \frac{1}{M}\sum_{i=1}^{M}(y_i(x) - f(x))^2$$

Taking expectation of both sides:

$$E_x[(\frac{1}{M}\sum_{i=1}^{M} y_i(x) - f(x))^2] \leq E_x[\frac{1}{M}\sum_{i=1}^{M}(y_i(x) - f(x))^2]$$

The left side is E_{ENS}, the right side is E_{AV}, therefore:

$$E_{ENS} \leq E_{AV}$$

Extension to General Convex Error Functions

For any convex error function $E(y)$:

- Apply Jensen's inequality directly to E:

$$E(\frac{1}{M}\sum_{i=1}^{M} y_i(x)) \leq \frac{1}{M}\sum_{i=1}^{M} E(y_i(x))$$

Taking expectation:

$$E_x[E(\frac{1}{M}\sum_{i=1}^{M} y_i(x))] \leq E_x[\frac{1}{M}\sum_{i=1}^{M} E(y_i(x))]$$

$$E_x[E(\frac{1}{M}\sum_{i=1}^{M} y_i(x))] \leq \frac{1}{M}\sum_{i=1}^{M}[E_x(E(y_i(x)))]$$

Therefore:

$$E_{ENS} \leq E_{AV}$$

---

4 Regularizer

Solution:

Noisy data inputs: For each x_k, we have $\tilde{x}_k = x_k + \epsilon_k$ where $\epsilon_k \sim \mathcal{N}(0, \sigma^2)$
The expectation of the error over noisy inputs is:

$$E[\mathcal{E}(w)] = E[\frac{1}{2}\sum_{i=1}^{N}(y(\tilde{x}_i, w) - t_i)^2]$$

For noisy input $\tilde{x}$, our linear model becomes:

$$\begin{aligned} y(\tilde{x}, w) &= w_0 + \sum_{k=1}^{D} w_k(x_k + \epsilon_k) \\ &= w_0 + \sum_{k=1}^{D} w_k x_k + \sum_{k=1}^{D} w_k \epsilon_k \end{aligned}$$

Therefore:

$$\begin{aligned} y(\tilde{x}, w) - t_i &= (w_0 + \sum_{k=1}^{D} w_k x_k + \sum_{k=1}^{D} w_k \epsilon_k) - t_i \\ &= (y(x, w) - t_i) + \sum_{k=1}^{D} w_k \epsilon_k \end{aligned}$$

Expanding the squared term:

$$\begin{aligned} (y(\tilde{x}, w) - t_i)^2 &= ((y(x, w) - t_i) + \sum_{k=1}^{D} w_k \epsilon_k)^2 \\ &= (y(x, w) - t_i)^2 + (\sum_{k=1}^{D} w_k \epsilon_k)^2 + 2(y(x, w) - t_i)(\sum_{k=1}^{D} w_k \epsilon_k) \end{aligned}$$

Taking expectation over noise:

$$E[(y(\tilde{x}, w) - t_i)^2] = (y(x, w) - t_i)^2 + E[(\sum_{k=1}^{D} w_k \epsilon_k)^2] + 2(y(x, w) - t_i)E[\sum_{k=1}^{D} w_k \epsilon_k]$$

Since ϵ_k has zero mean, the last term becomes zero.
For the middle term:

$$E[(\sum_{k=1}^{D} w_k \epsilon_k)^2] = \sigma^2 \sum_{k=1}^{D} w_k^2$$

(because the noise terms are independent)
Therefore:

$$\begin{aligned} E[\mathcal{E}(w)] &= \frac{1}{2} \sum_{i=1}^{N} [(y(x_i, w) - t_i)^2 + \sigma^2 \sum_{k=1}^{D} w_k^2] \\ &= \frac{1}{2} \sum_{i=1}^{N} (y(x_i, w) - t_i)^2 + \frac{N\sigma^2}{2} \sum_{k=1}^{D} w_k^2 \end{aligned}$$

The final result:

$$E[\mathcal{E}(w)] = \mathcal{E}_{\text{noise-free}}(w) + \lambda \sum_{k=1}^{D} w_k^2$$

where $\lambda = \frac{N\sigma^2}{2}$ and $\mathcal{E}_{\text{noise-free}}(w)$ is the standard sum-of-squares error on noise-free data.
Conclusion: This shows that minimizing the expected error over noisy inputs is equivalent to minimizing the sum-of-squares error on noise-free data plus an L2 regularization term ($\lambda \sum_{k=1}^{D} w_k^2$) that excludes the bias term w_0. The regularization strength λ is proportional to both the number of data points N and the noise variance σ^2.

Programming Questions

5 Random Forests

Introduction

The analysis of a custom Random Forest Classifier applied to a spam dataset. We compare the custom implementation's performance with scikit-learn's implementation in terms of accuracy and computational efficiency. Additionally, we explore the classifier's sensitivity to the number of features (m) and analyze the Out-Of-Bag (OOB) and Test errors as a function of m.

Part (a) - Custom vs. Scikit-learn Random Forest Classifier

- **Custom Random Forest Classifier**
  - **Accuracy:** 93.99%
  - **Time Taken:** 183.86 seconds
- **Scikit-learn Random Forest Classifier**
  - **Accuracy:** 94.57%
  - **Time Taken:** 0.68 seconds

Comparison: The scikit-learn implementation achieved a slightly higher accuracy (94.57%) and was significantly faster (0.68 seconds) than the custom implementation (93.99%, 183.86 seconds). This difference in speed can be attributed to scikit-learn's optimization techniques, making it more efficient for larger datasets.

Part (b) - Sensitivity Analysis of Random Forests to the Number of Features (m)

The following table shows the accuracy and computation time for the custom Random Forest model evaluated at different values of m, the number of features considered at each split:

Number of Features (m)	Accuracy	Time Taken (seconds)
3 ($\frac{1}{2}\sqrt{m}$)	93.56%	61.50
7 ($\sqrt{m}$)	94.06%	135.11
15 ($2\sqrt{m}$)	94.50%	281.87

Table 1: Accuracy and Time Taken for Varying Values of m in the Custom Random Forest

```
Custom Random Forest - Accuracy: 0.9356 and Time taken: 68.03 seconds for m: 3
Custom Random Forest - Accuracy: 0.9406 and Time taken: 135.11 seconds for m: 7
Custom Random Forest - Accuracy: 0.9450 and Time taken: 281.87 seconds for m: 15
```

Figure 1: Random Forest Accuracy for different values of m

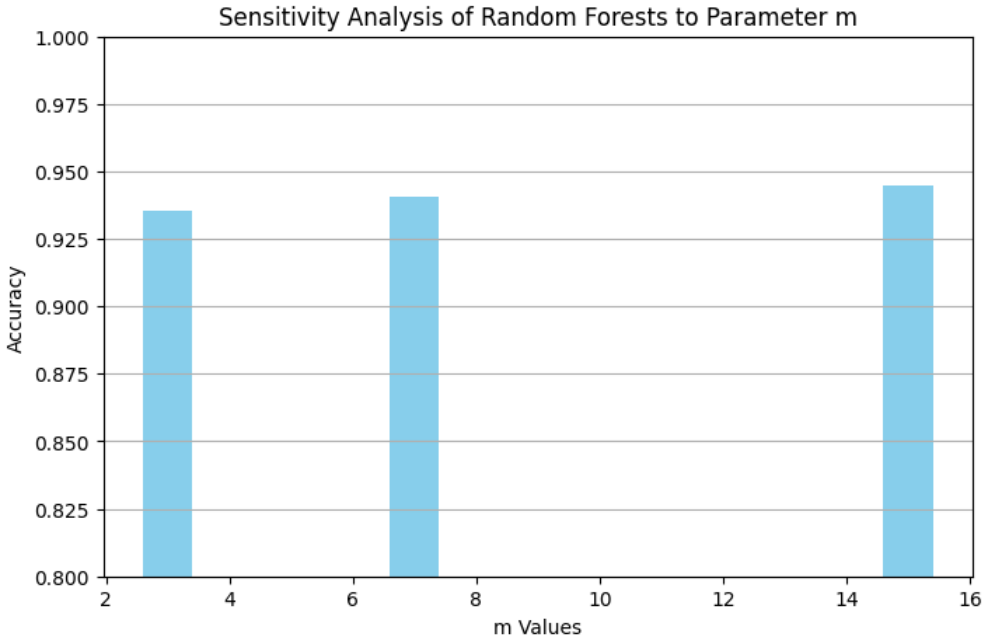

Figure 2: Accuracy of Random forest for different number of features

Interpretation: As m increases, the model's accuracy improves, peaking at 94.50% for $m = 15$. However, increasing m also leads to longer training times. The value $m = 7$ provides a good trade-off between high accuracy and computational efficiency as the increase after that is marginal.

Part (c) - OOB and Test Error vs. Number of Features (m)

The table below presents the Out-Of-Bag (OOB) and Test errors for different values of m:

Number of Features (m)	OOB Error	Test Error
3	0.05	0.06951
7	0.04	0.05793
15	0.03	0.05648

Table 2: OOB and Test Errors for Varying Values of m in the Custom Random Forest

```
Out-Of-Bag error and Test error values for m:3 : 0.05 and 0.06951
Out-Of-Bag error and Test error values for m:7 : 0.04 and 0.05793
Out-Of-Bag error and Test error values for m:15 : 0.03 and 0.05648
```

Figure 3: OOB error and Test errors

Figure 4: OOB Error and Test Error vs Number of Features

Interpretation: Both OOB and Test errors decrease as m increases, which indicates improved generalization with larger values of m. However, the marginal improvements beyond $m = 7$ suggest a diminishing return, making $m = 7$ an optimal choice for balancing error reduction and efficiency.

---

6 Gradient Boosting

Introduction

This covers the analysis and application of Gradient Boosting on the Lending Club dataset, focusing on predicting loan outcomes. The dataset includes labels indicating if a loan is "Fully Paid" (assigned as +1) or "Charged Off" (assigned as -1). Instances labeled "Current" were omitted as they are not relevant for this binary classification problem.

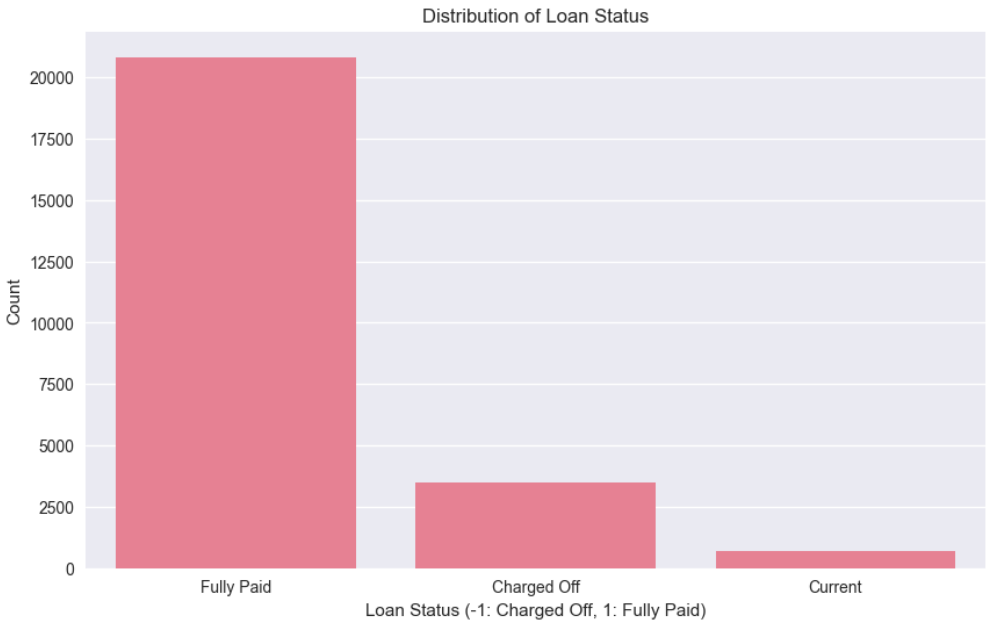

Figure 5: Distribution of Loan Status

The following tasks were performed:

- Data Preprocessing to prepare the dataset for machine learning.
- Training and tuning a Gradient Boosting model to predict loan outcomes.
- Evaluating model performance against a baseline Decision Tree classifier.

Part (a) - Data Preprocessing

To ensure effective model performance, the dataset underwent the following preprocessing steps:

1. **Handling Missing Values**: Records with missing values were addressed by imputing with mean values for numerical features and mode values for categorical features. While features with more than 60 percent Null values were dropped.

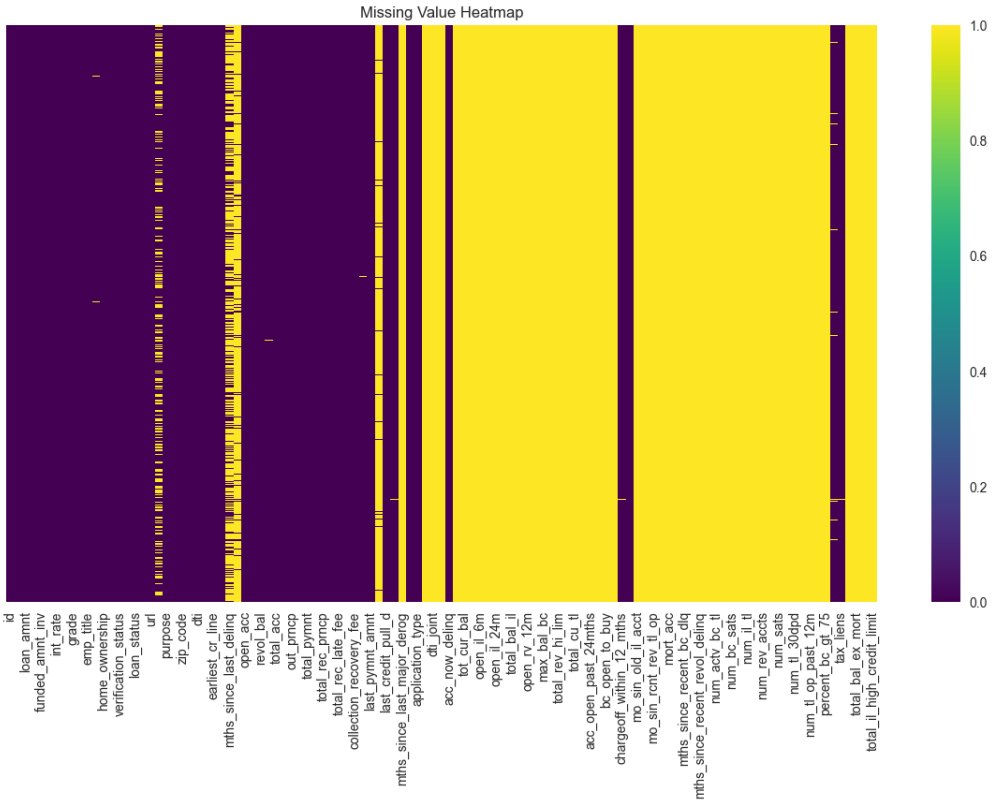

Figure 6: Missing Value Heatmap

2. **Feature Selection**: Irrelevant features, such as identifier columns(id, member_id, etc.), were removed based on correlation with the target label. The irrelevant columns/features with having only one unique value were also removed.

```
Correlations with loan_status:
                          correlation
open_acc                     0.000698
revol_bal                    0.006231
emp_length                   0.012040
total_rec_int                0.013844
delinq_2yrs                  0.013968
total_acc                    0.017341
earliest_cr_line_month       0.017854
last_pymnt_d_month           0.021381
installment                  0.026918
funded_amnt_inv              0.038255
annual_inc                   0.040116
member_id                    0.044758
dti                          0.046135
id                           0.048124
pub_rec_bankruptcies         0.049319
pub_rec                      0.050736
funded_amnt                  0.056209
loan_amnt                    0.058606
```

Figure 7: Correlation-1

```
int_rate                    0.210081
collection_recovery_fee     0.216374
last_pymnt_amnt             0.221733
total_pymnt_inv             0.230976
total_pymnt                 0.233951
total_rec_prncp             0.334436
recoveries                  0.340974
loan_status                 1.000000
out_prncp                        NaN
out_prncp_inv                    NaN
collections_12_mths_ex_med       NaN
policy_code                      NaN
acc_now_delinq                   NaN
chargeoff_within_12_mths         NaN
delinq_amnt                      NaN
tax_liens                        NaN
last_pymnt_d_year                NaN
earliest_cr_line_year            NaN
```

Figure 8: Correlation-2

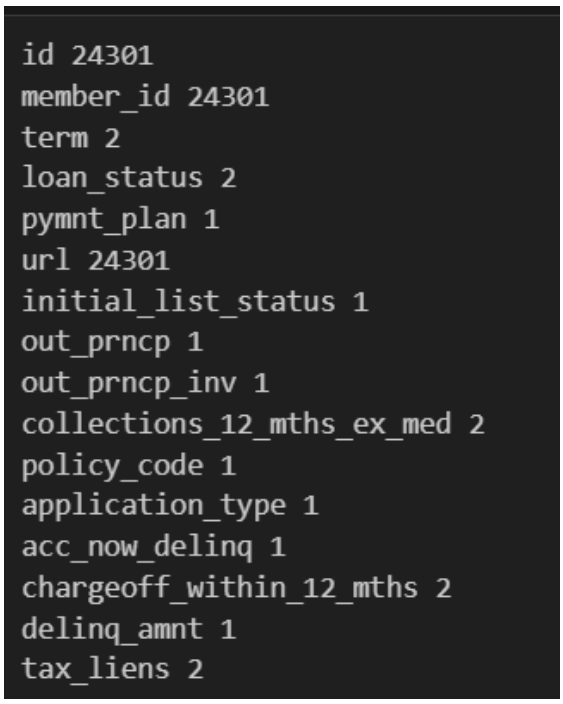

Figure 9: Uniques Values in each feature

3. **Encoding Categorical Features**: All categorical features were transformed using one-hot encoding.
4. **Processing of different features into proper datatype**:
   - Date columns were processed into proper format using datetime
   - Term feature : the term value is extracted from term column
   - Interest rate value extracted from the format $x\%$ to x for further use.
   - Revol_util feature value transformed from $revol\%$ to $revol$.
   - Employment Length: transformed from less than 1 years,...10+ years to numerical values in range 0 to 10 (where 10 denotes 10+years and 0 denotes less than 1 year employment).
5. **Data Scaling**: Standardization was applied to numerical features to normalize the scale (using log transform) , which is beneficial for Gradient Boosting.

Part (b) - Gradient Boosting Model

The Gradient Boosting model was implemented using the `GradientBoostingClassifier` from `sklearn.ensemble`, and the hyperparameters were tuned to achieve optimal performance. The best hyperparameters and results are summarized below.

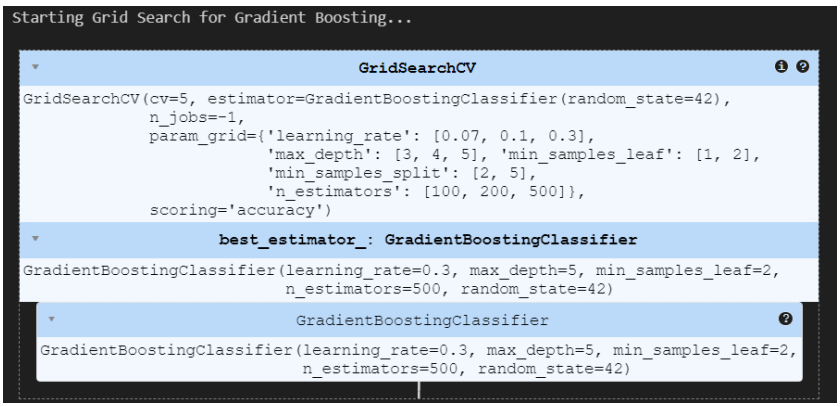

Figure 10: Gradient Boosting Grid Search

Optimal Hyperparameters

The model was optimized using a grid search, yielding the following best-performing parameters:

```
Best Parameters:
    {'learning_rate': 0.3, 'max_depth': 5, 'min_samples_leaf': 2,
     'min_samples_split': 2, 'n_estimators': 500}
```

Performance Metrics

The table below summarizes the performance metrics for the Gradient Boosting model, indicating high precision and recall.

Metric	Accuracy	Precision	Recall
Score	99.67%	99.68%	99.93%

Table 3: Gradient Boosting Model Performance

Detailed Classification Report

The following table shows the classification metrics for each class:

	precision	recall	f1-score	support
-1	1.00	0.98	0.99	2153
1	1.00	1.00	1.00	12123
accuracy			1.00	14276
macro avg	1.00	0.99	0.99	14276
weighted avg	1.00	1.00	1.00	14276

Effect of the Number of Trees

To assess the impact of tree count on accuracy, the following accuracies were recorded for varying numbers of trees:

Number of Trees	Accuracy
10	97.22%
50	98.93%
100	99.43%
200	99.55%
500	99.66%

Table 4: Accuracy vs. Number of Trees in Gradient Boosting Model

Figure 11: Effect of the Number of Trees

Part (c) - Comparison with Decision Tree Classifier

For comparison, a simple Decision Tree classifier was built using `sklearn`'s `DecisionTreeClassifier`. The following table presents the performance comparison:

Model	Accuracy	Precision	Recall
Gradient Boosting	99.67%	99.68%	99.93%
Decision Tree	99.36%	99.58%	99.66%

Table 5: Performance Comparison of Gradient Boosting and Decision Tree Models

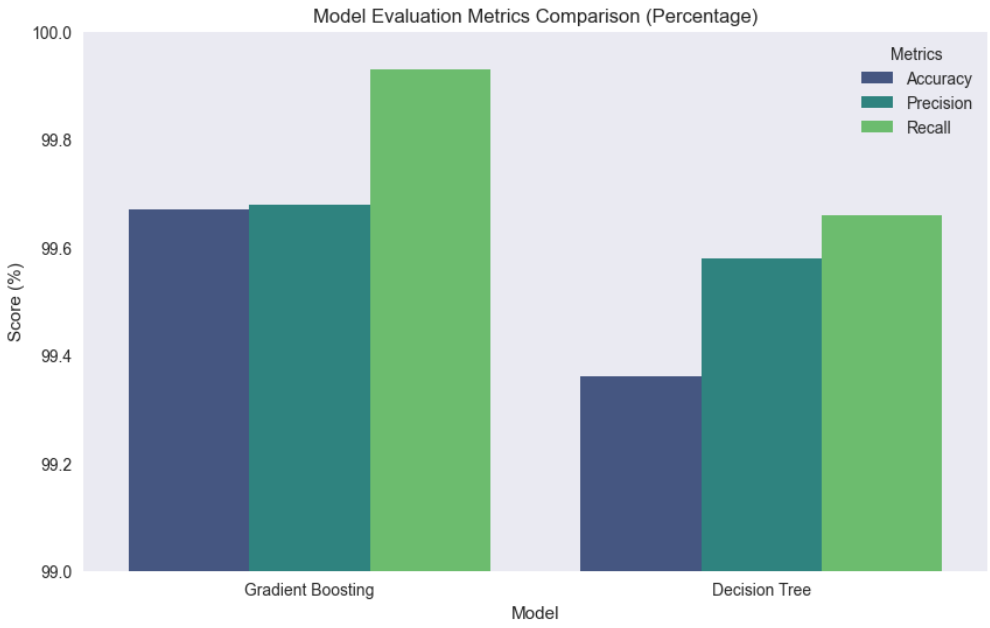

Figure 12: Comparison between Gradient Boost and Decision Tree Classifier

Conclusion

The Gradient Boosting model achieved superior performance, with the optimal parameters resulting in 99.67% accuracy. Increasing the number of trees enhanced the model's accuracy, with the best results obtained at 500 trees. When compared to the Decision Tree classifier, Gradient Boosting demonstrated better precision and recall, proving its effectiveness for loan outcome prediction.