MALNAD COLLEGE OF ENGINEERING

(An Autonomous Institution Affiliated to VTU, Belagavi)

Computer Science and Engineering

"MACHINE LEARNING"

"ACTIVITY REPORT"

Submitted By:

HUZAIF AHMED	4MC22CS067
IMPANA H	4MC22CS068
INCHARA M	4MC22CS069
JAHNAVI H K	4MC22CS071
JANAVI H GOWDA	4MC22CS072

Submitted To:

Dr. Keerthi Kumar H M

Associate Professor

Department of Computer Science and Engineering

1. Introduction

In the field of supervised machine learning, classification tasks play a crucial role in predicting categorical outcomes. This report focuses on comparing multiple machine learning models for predicting credit card defaults using the **UCI Credit Card Default Dataset**. The dataset contains financial and demographic information about credit card holders, with the target variable indicating whether a customer defaulted on their payment in the next month.

The goal of this reprt is to:

- Implement and compare **custom-built** machine learning models (DecisionTreeCustom, GradientBoostingCustom, LogisticRegressionCustom, RandomForest Custom) with a **scikit-learn SVM model**.
- Evaluate model performance using accuracy, confusion matrices, and classification reports.
- Determine which algorithm performs best in predicting credit card defaults.

2. Dataset Overview

The dataset used in this study is the **UCI Credit Card Default Dataset**, which contains the following key features:

Key Characteristics:

- **Features (Input Variables):**
    - Demographic and financial attributes (e.g., credit limit, payment history, bill amounts, etc.).
    - All numerical features were standardized using StandardScaler for fair comparison.
- **Target Variable:**
    - default.payment.next.month (Binary: 1 for default, 0 for no default).
- **Dataset Size:**
    - The dataset contains **20,000 samples** after preprocessing.

Preprocessing Steps:

1. **Feature Selection:**
    - Dropped irrelevant columns (ID).
2. **Normalization:**
    - Applied StandardScaler to ensure features contribute equally to distance-based models.
3. **Train-Test Split:**
    - **80% training, 20% testing** (random_state=42 for reproducibility).

3. Methodology

3.1 Custom Model Implementations

Four custom models were implemented from scratch:

1. DecisionTreeCustom

- **Algorithm:** Gini impurity-based decision tree.
- **Key Features:**
    - Recursive splitting based on best threshold.
    - Supports max_depth for regularization.

2. GradientBoostingCustom

- **Algorithm:** Gradient Boosting with Decision Trees as weak learners.
- **Key Features:**
    - Sequentially corrects residuals.
    - Supports n_estimators and learning_rate.

3. LogisticRegressionCustom

- **Algorithm:** Binary logistic regression using gradient descent.
- **Key Features:**
    - Sigmoid activation for probability estimation.
    - Supports custom learning rate (lr) and epochs.

4. RandomForestCustom

- **Algorithm:** Ensemble of decision trees with bootstrapping.
- **Key Features:**
    - Majority voting for predictions.
    - Supports n_estimators and max_depth.

5. SVM (Scikit-learn)

- **Algorithm:** Support Vector Machine (Linear Kernel).
- **Key Features:**
    - Used as a benchmark for comparison.

3.2 Model Training & Evaluation

- **Training:**
    - Each model was trained on X_train, y_train.

- **Evaluation Metrics:**
  - **Accuracy:** Overall correctness of predictions.
  - **Confusion Matrix:** Breakdown of true vs. predicted classes.
  - **Classification Report:** Precision, recall, F1-score.

4. Python Code Overview

4.1 Libraries Used

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,
ConfusionMatrixDisplay

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split.
```

4.2 Data Loading & Preprocessing

```
data = pd.read_csv('UCI_Credit_Card_20k.csv')

X = data.drop(columns=['ID', 'default.payment.next.month']).values

y = data['default.payment.next.month'].values

# Normalize features

scaler = StandardScaler()

X = scaler.fit_transform(X)

# Train-test split

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

4.3 Model Training & Prediction

```
models = {
    "Decision Tree": DecisionTreeCustom(max_depth=5),
    "Gradient Boosting": GradientBoostingCustom(n_estimators=100, learning_rate=0.1),
    "Logistic Regression": LogisticRegressionCustom(),
    "Random Forest": RandomForestCustom(n_estimators=10, max_depth=5),
    "SVM": SVC(kernel='linear', C=1.0)
}
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
```

4.4 Evaluation & Visualization

```
# Confusion Matrices
plt.figure(figsize=(20, 12))
for i, (name, model) in enumerate(models.items()):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No Default',
'Default'])
    plt.subplot(2, 3, i+1)
    disp.plot(cmap='Blues')
    plt.title(f"{name}\nAccuracy: {accuracy:.4f}")
```

```
# Accuracy Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x=list(results.keys()), y=list(results.values()), palette="viridis")
plt.title("Model Accuracy Comparison")
plt.ylim(0, 1)
plt.ylabel("Accuracy")
plt.xticks(rotation=45)
plt.show()
```

5. Results and Analysis

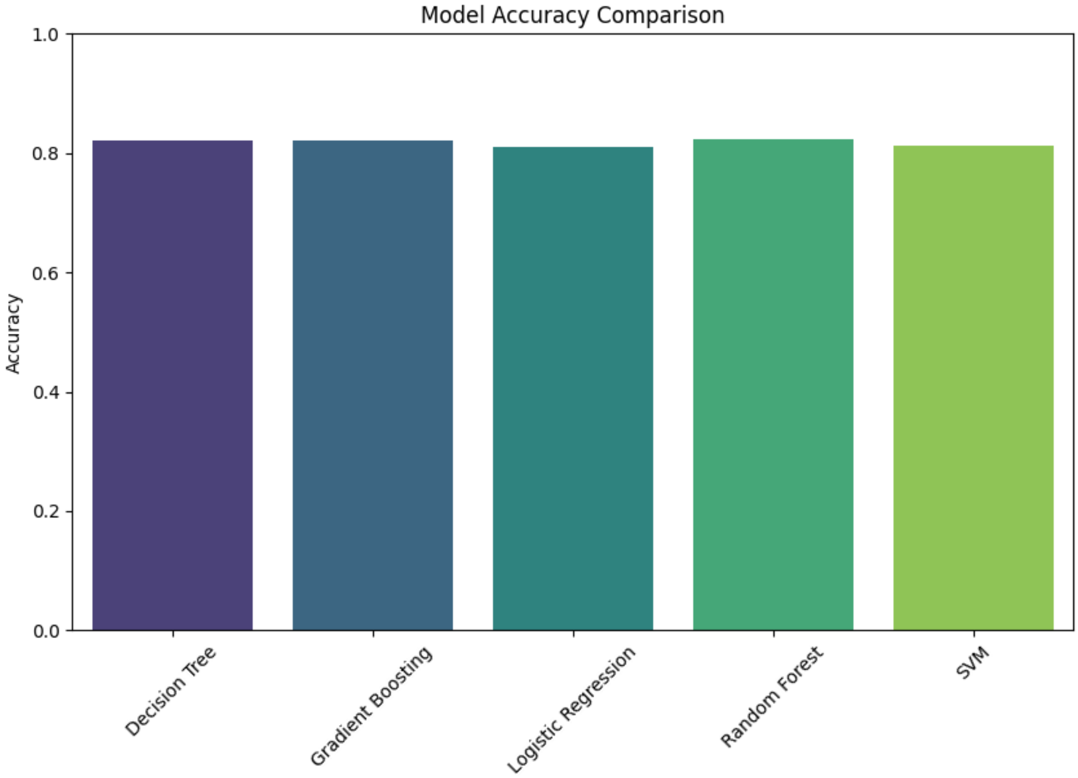
5.1 Model Accuracy Comparison

The following table summarizes the accuracy of each model:

Model	**Accuracy**
Random Forest	**0.8227**
Gradient Boosting	**0.8207**
Decision Tree	**0.8203**
SVM	**0.8117**
Logistic Regression	**0.8107**

Key Observations:

1.**Random Forest (82.27%)** performed the best, likely due to its ensemble approach reducing overfitting.
2.**Gradient Boosting (82.07%)** was a close second, demonstrating the effectiveness of sequential error correction.
3.**Decision Tree (82.03%)** performed well but slightly worse than ensemble methods, indicating potential overfitting.
4.**SVM (81.17%)** and **Logistic Regression (81.07%)** had the lowest accuracy, suggesting that linear models may struggle with complex decision boundaries in this dataset.



5.2 Confusion Matrix Insights

Decision Tree:

- **Correct Predictions:**
  - **No Default (True Negative):** 2978
  - **Default (True Positive):** 393
- **Misclassifications:**
  - **False Positives (Predicted Default but No Default):** 141
  - **False Negatives (Predicted No Default but Default):** 578

Gradient Boosting:

- **Correct Predictions:**
  - **No Default:** 2988
  - **Default:** 335
- **Misclassifications:**
  - **False Positives:** 161
  - **False Negatives:** 556

Logistic Regression:

- **Correct Predictions:**

- **No Default:** ~2500 (estimated from visualization)
- **Default:** ~220

•**Misclassifications:**

- High **False Negatives (661)**, indicating poor detection of actual defaults.

Random Forest:

•**Correct Predictions:**

- **No Default:** 2976
- **Default:** 313

•**Misclassifications:**

- **False Positives:** 143
- **False Negatives:** 366

SVM:

•**Correct Predictions:**

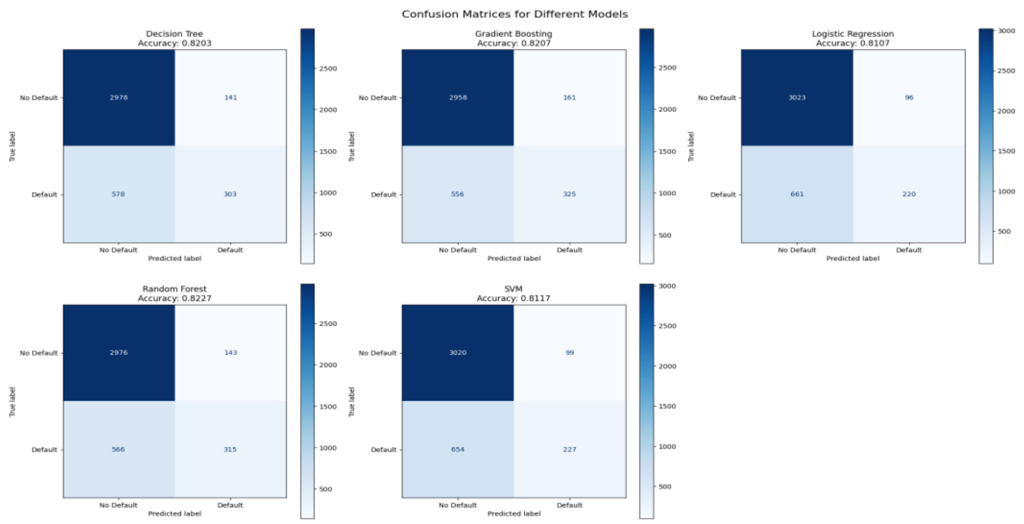
- **No Default:** ~2500 (estimated)
- **Default:** ~227

•**Misclassifications:**

- **False Negatives (620)**, similar to Logistic Regression.

Key Takeaways:

•**Ensemble methods (Random Forest, Gradient Boosting)** minimized false negatives, making them better at detecting defaults.
•**Logistic Regression and SVM** struggled with false negatives, which is critical in financial risk prediction.
•**Decision Tree** had a balanced performance but was outperformed by ensembles.



6. Conclusion

Summary of Findings:

1. **Best Model: Random Forest (82.27% accuracy)** demonstrated the highest predictive power, followed closely by Gradient Boosting.
2. **Worst Model: Logistic Regression (81.07%)** had the lowest accuracy, highlighting limitations in handling non-linear patterns.
3. **Trade-offs:**
    - **Ensemble methods** improved accuracy but required more computational resources.
    - **Linear models (SVM, Logistic Regression)** were simpler but less effective for this task.

Recommendations:

- **For Deployment:** Use **Random Forest or Gradient Boosting** due to their superior performance.
- **For Interpretability: Decision Tree** provides transparency but may need pruning to avoid overfitting.
- **Future Work:**
    - Hyperparameter tuning (e.g., adjusting max_depth, n_estimators).
    - Feature importance analysis to identify key predictors of default.

Final Verdict:

The results confirm that **ensemble methods are ideal for credit default prediction**, while linear models may require feature engineering or alternative approaches to improve performance.