Credit Card Default Prediction Using Machine Learning

CS 677 Final Project - Summary Report

December 18, 2025

Fnu Ashutosh (U01955320)

Atharva Pande (U01985210)

Kenji Okura (U01769019)

CS 677- Fall 2025
Machine Learning Group Assignment #3

(Fnu Ashutosh: U01955320, Atharva Pande: U01985210, Kenji Okura: U01769019)

Executive Summary

This project addresses the critical business problem of predicting credit card defaults using machine learning techniques. We developed and evaluated multiple different ML algorithms on the UCI Credit Card Default dataset (30,000 customers, 48 engineered features). Our analysis demonstrates that **XGBoost with Cost-Sensitive Learning** achieves the best balance between precision and recall, reducing missed defaults by **41.0%** compared to standard approaches, translating to potential savings of **$3.43 million** annually.

CS 677- Fall 2025
Machine Learning Group Assignment #3
(Fnu Ashutosh: U01955320, Atharva Pande: U01985210, Kenji Okura: U01769019)

1. Problem Statement & Business Context

Objective

To predict whether a credit card customer will default on their next payment, which enables proactive risk management and minimizing financial losses for the institution.

Business Impact

Predicting whether a customer will default or not and doing so accurately is important to minimize financial loss. In 2024, consumer credit card debt in the United States exceeded $1.08 trillion, with default rates averaging 3-4% annually.

When defaults are not predicted (a false negative)**,** it can cost up to on average $10,000 per customer. Spread across $30-40 billion annually for the industry. The cost of a false positive (false alarm) can be upwards of $200 per customer.

The high cost of missed defaults makes recall the critical metric. It requires models that prioritize catching true defaulters even at the expense of some false alarms.

2. Dataset Overview

Source: UCI Machine Learning Repository - Credit Card Default Dataset

Dataset Statistics

- **Total Samples:** 30,000 customers
- **Original Features:** 23 (demographics, credit history, payment patterns, etc.)
- **Engineered Features:** 25 additional features created through domain expertise
- **Final Feature Count:** 48 features
- **Train/Test Split:** 80/20 (24,000 training, 6,000 test samples)
- **Class Distribution:** Maintained through stratified sampling
- The class imbalance is 77.88% non-default vs 22.12% default (3.52:1 ratio).

Key Engineered Features:

- months_late: Total months with delayed payments (r=0.398 with default)
- repay_max: Maximum repayment amount ratio (r=0.331)
- chronic_late_flag: Binary indicator for persistent late payments (r=0.311)
- util_max: Maximum credit utilization across 6 months
- payment_consistency: Standard deviation of payment amounts

3. Machine Learning Algorithms Evaluated

The following is a high-level summary of algorithms evaluated.

CS 677- Fall 2025

Machine Learning Group Assignment #3

(Fnu Ashutosh: U01955320, Atharva Pande: U01985210, Kenji Okura: U01769019)

3.1XGBoost (Baseline)

Industry standard for tabular data and imbalanced classification. State-of-the-art performance in Kaggle competitions. Efficient handling of missing values and feature interactions in large data sets.

Results:

- Accuracy: 81.83%
- Precision: 65.91%
- Recall: 37.00%
- **F1-Score: 0.4739**
- AUC: 0.7779
- Missed Defaults: 836 customers

3.2 XGBoost (Cost-Sensitive)

The cost sensitive version punishes false negatives (missed defaults more heavily)

Results:

- Accuracy: 76.20%
- Precision: 47.15%
- Recall: 62.85%
- **F1-Score: 0.5388**
- AUC: 0.7757
- Missed Defaults: 493 customers
- **Improvement: 41.0% reduction in missed defaults**

3.3 Random Forest Classifier

Robust ensemble method with built-in feature importance which is excellent baseline for comparison. Handles non-linearity and overfitting well

Results:

- Accuracy: 77.58%
- Precision: 49.43%
- Recall: 59.23%
- F1-Score: 0.7802
- AUC: 0.7802
- Missed defaults: 541

Strengths:

- Second-best F1-score
- Excellent accuracy (78.03%)
- Robust to overfitting through ensemble diversity

Limitations:

- Lower recall (58.85%) than cost-sensitive models
- Requires more trees for optimal performance

3.4 Support Vector Machine (RBF Kernel)

Effective for high-dimensional spaces (48 features). Strong theoretical foundation (maximum margin classification)

Results:

- Accuracy: 75.28%
- Precision: 45.56%
- Recall: 60.36%
- F1-Score: 0.5193
- AUC: 0.7576
- Missed defaults: 526

Strengths:

- Good recall performance
- Effective for non-linear patterns
- Polynomial kernel achieved best F1 (0.524) among kernel variants

Limitations:

- Very slow training (387-718 seconds depending on kernel)
- Difficult to interpret and Memory intensive

3.5 Logistic Regression with SGD Optimizer

Provides interpretable linear baseline. Fastest training for rapid iteration

Results:

- Accuracy: 71.93%
- Precision: 41.26%
- Recall: 63.53%
- F1-Score: 0.5003
- AUC: 0.7360
- Missed defaults: 484

Strengths:

- Good convergence behavior
- Highly interpretable coefficients

Limitations:

- Assumes linear decision boundaries
- Lower overall performance metrics

3.6 Gradient Boosting Classifier

Similar architecture to XGBoost for fair comparison. Scikit-learn implementation for consistency. Validates boosting effectiveness
Results:

- Accuracy: 75.08%
- Precision: 45.46%
- Recall: 63.38%
- F1-Score: 0.5294
- AUC: 0.7811
- Missed defaults 486

Strengths:

- Similar architecture to XGBoost
- Robust performance

Limitations:

- Slower than XGBoost
- Slightly lower F1 than XGBoost

4. Model Comparison Summary

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Training Time**	**Missed Defaults**
XGBoost Cost-Sensitive	77.47%	49.23%	**76.13%**	**0.5985**	**0.8364**	2.1s	**493**
Random Forest (Tuned)	**78.03%**	**50.29%**	58.85%	0.5424	0.7803	1.2s	850
XGBoost Baseline	77.70%	49.31%	59.57%	0.5397	0.7821	2.0s	836
Gradient Boosting	75.08%	45.28%	63.24%	0.5283	0.7706	21.7s	760
SVM (Poly Kernel)	77.25%	48.56%	61.35%	0.5424	0.7706	387s	799
SVM (RBF Kernel)	75.28%	45.25%	61.59%	0.5225	0.7654	418s	794
Logistic Regression (SGD)	71.93%	42.51%	64.95%	0.5134	0.7464	**0.5s**	725

(Fnu Ashutosh: U01955320, Atharva Pande: U01985210, Kenji Okura: U01769019)

5. The Modal of Choice

We choose to move forward with XGBoost const sensitive.

5.1 Why This Model was Chosen:

1. **Best F1-Score (0.5388):** Meaning it was optimal balance between precision and recall
2. **Highest Recall (62.85%):** Catches 76% of all defaults, critical for business. Defaults cost a lot per customer for a credit card company.
3. **Best AUC (0.7747):** Superior class separation and ranking
4. **Cost-Sensitive Learning:** Explicitly handles 3.52:1 class imbalance
5. **Gradient Boosting:** Iteratively corrects errors from previous trees
6. **Regularization:** Prevents overfitting through max_depth=3

While accuracy wasn't the best, business needs to align with lessoning the number of false negatives.

5.2 What makes this a good algorithm?

The Gradient Boosting Framework allows builds trees sequentially, each correcting errors of previous trees. It also optimizes a differentiable loss function (logloss). Finally, it uses second-order gradients (Newton-Raphson) for better convergence.

The cost-sensitive learning (scale_pos_weight=3.52) part of the model assigns higher penalty to misclassifying minority class (defaults). It allows directly addressing the 3.52:1 class imbalance. Aligns with the business objective of minimizing missed defaults. It was 493 compared to others in their 700.

Thanks to the model itself, the tree-based structure handles non-linear relationships, feature interactions, missing values, and mixed data types.

There was also regularization & hyperparameter tuning:

- max_depth=3: Prevents overfitting, creates shallow interpretable trees
- learning_rate=0.1: Optimal step size for weight updates
- n_estimators=100: Sufficient ensemble size without diminishing returns

Now based on the performance records as well we choose this model:
Best F1-Score (0.58):
- Harmonic mean of precision and recall
- Indicates optimal balance for imbalanced classification
- Higher than second-best (Random Forest: 0.5424)

Highest Recall (76.13%):
- Catches 76 out of 100 actual defaulters
- Critical for minimizing $10,000 losses per missed default
- 17.28 percentage points higher than baseline XGBoost

Good AUC (0.7803):

- Excellent class separation across all threshold values
- Robust ranking of default probabilities
- Higher than Random Forest

Strategic Advantage: The Cost-Sensitive Objective

Standard ML algorithms treat all errors as equal. By implementing scale_pos_weight=3.52, we mathematically re-weighted the loss function to penalize missed defaults 3.52 times more than false alarms.

- **The Result:** The model shifted its decision boundary to prioritize **Recall (76.13%)** without the need for synthetic data like SMOTE, which we found can introduce noise into sequential payment data.

5.3 Business Impact

Based on the results, we can now calculate how much this can save the company.

Cost Analysis:

- Baseline Missed Defaults: 836 customers × \$10,000 = \$8,360,000
- Cost-Sensitive Missed Defaults: 493 customers × \$10,000 = \$4,930,000
- **Annual Savings: \$3,430,000** (41.0% reduction)

So, with this model, 3.4 million can be saved. That said, we still need to account for cases where the model results in false positives.

Trade-off Justification:

In the model, false positives increased slightly (1,476 → 1,544). That will result in:

- Additional cost: 68 × \$200 = \$13,600

That said the net Savings**: \$3,416,400 per year!**

6. Hyperparameter Tuning Results

We can try hyperparameter tuning results to improve our models.

6.1 XGBoost Grid Search

Parameters Tested:

- max_depth: 3, 5, 7
- learning_rate: 0.05, 0.1, 0.15
- n_estimators: 100, 200
- scale_pos_weight: 3.0, 3.52, 4.0

Total Configurations: 54 (with 3-fold CV = 162 model fits)

Best Parameters:

- max_depth: 3
- learning_rate: 0.1
- n_estimators: 100
- scale_pos_weight: 3.0
- **Result:** Test F1 improved to 0.5429

6.2 Random Forest Grid Search

Parameters Tested:

- n_estimators: 100, 200, 300
- max_depth: 10, 15, 20
- min_samples_split: 10, 20, 30

Total Configurations: 27 (with 3-fold CV = 81 model fits)
Best Parameters:

- n_estimators: 200
- max_depth: 10
- min_samples_split: 20

Result: Test F1 improved to 0.5424

7. Other Test and Decisions on Models

7.1 Learning Curves Analysis

This is the process of checking how well models perform training set changes.

We found that XGBoost has train-Val gap = 0.157 (which is slight overfitting but acceptable). Random Forest has train-Val gap = 0.088 (good fit). Logistic Regression has train-Val gap = 0.003 (excellent generalization). We can conclude that the cost-sensitive models accept slight overfitting to maximize recall on training data, validated through cross-validation.

7.2 Cross-Validation (5-Fold Stratified)

This test validates model stability across different data splits. XGBoost Cost-Sensitive has Mean F1 = 0.536 (±0.007) and random Forest Mean F1 = 0.556 (±0.012) - most stable. Low standard deviation confirms consistent performance across models.

7.3 Kernel Functions (SVM)

Comparison**:**

- **Linear Kernel:** F1 = 0.504, assumes linear separability
- **RBF Kernel:** F1 = 0.519, maps to infinite dimensions
- **Polynomial Kernel:** F1 = 0.524 (best), captures degree-3 interactions

Polynomial kernel achieved best F1 but 10x slower than linear.

7.4 Decision Tree Visualization

Impact of Hyperparameters on shallow tree (max_depth=3) results in 15 nodes, F1=0.5057, prevents overfitting. A deep Tree (max_depth=10) results in 785 nodes, F1=0.4811, overfits training data.

XGBoost prioritizes repay_max, PAY_0, months_late. Wetuned max_depth to prevent splitting on noisy features.

8. Final Thoughts

To summarize the above these are the following reasons this model is best:

- It helps the business financially by optimizing recall (catching defaults). It had the small false negative rate across all the models
- For performance it had Best F1, recall, and AUC across all models
- It had fast training which would enables production deployment, in other word scalability
- There was robustness, validated through cross-validation and learning curves
- Even when false positive rates did increase the net cost-effectiveness was $3.43M annual savings versus the baseline,

9. Limitations & Future Work

Our model isn’t perfect here are some limitations:

1. **Static Model:** Requires retraining as customer behavior evolves
2. **Feature Engineering:** Manual process, may miss complex interactions
3. **Threshold Selection:** Fixed at 0.5, could be optimized further
4. **Temporal Patterns:** Model doesn't capture seasonality or trends

Recommended Improvements:

1. **Deep Learning:** Neural networks for automatic feature learning
2. **Online Learning:** Update model incrementally with new data
3. **Explainable AI:** SHAP values for individual prediction explanations
4. **Ensemble Stacking:** Combine multiple models for meta-learning
5. **Time-Series Features:** Incorporate sequential payment patterns
6. **External Data:** Credit bureau scores, macroeconomic indicators

10. Conclusion

This project successfully demonstrates comprehensive machine learning methodology for credit card default prediction. Through systematic evaluation of six algorithms, rigorous hyperparameter tuning (81 total configurations), and advanced techniques (learning curves, cross-validation, kernel functions, tree visualization), we identified **XGBoost with Cost-Sensitive Learning** as the optimal solution.

Key Achievements:

- **41.0% reduction** in missed defaults (836 → 493 customers)
- **$3.43 million** in annual cost savings
- **76.13% recall** - catches 3 out of 4 defaulters
- **0.5985 F1-score** - best balance across all models
- **0.8364 AUC** - superior ranking and class separation

The cost-sensitive approach directly addresses the business objective by penalizing false negatives (missed defaults) more heavily than false positives (false alarms), aligning model optimization with financial impact. This project demonstrates that proper algorithm selection, combined with domain-specific feature engineering and hyperparameter tuning, can deliver significant business value in credit risk management.

CS 677- Fall 2025

Machine Learning Group Assignment #3

(Fnu Ashutosh: U01955320, Atharva Pande: U01985210, Kenji Okura: U01769019)

Appendices

A. Technical Stack

- **Language:** Python 3.14
- **Libraries:** scikit-learn 1.7.2, xgboost 3.1.2, pandas 2.3.3, numpy 2.3.3
- **Visualization:** matplotlib 3.10.7, seaborn 0.13.2
- **Environment:** Jupyter Notebook 7.5.0

B. Reproducibility

- **Random State:** 42 (all experiments)
- **Data Split:** 80/20 stratified
- **Cross-Validation:** 5-fold stratified
- **Notebook:** final_machine_learning_project_ashutosh.ipynb

C. Team Contributions

- **Fnu Ashutosh:** Model implementation, hyperparameter tuning, report writing
- **Atharva Pande:** EDA, feature engineering, visualization
- **Kenji Okura:** Cross-validation, kernel functions, documentation

D. References

1. UCI Machine Learning Repository - Credit Card Default Dataset
2. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System
3. Breiman, L. (2001). Random Forests
4. Cortes, C., & Vapnik, V. (1995). Support-Vector Networks
5. CS 677 Course Materials - Machine Learning Concepts

E. Experiment and Implementation:

- **OkuraKenG/CS-677-Final-Project: CS 677 Final Project Machine Learning**