

# Predicting bank customer churn

## Machine learning project report

6.1.2025

Havriil Pietukhin, 2mAIN

# 1 Introduction

Customer churn prediction is a critical task for financial institutions. When a bank can identify which customers are likely to leave, it can take proactive measures to retain them. The cost of acquiring new customers typically exceeds the cost of retention, making accurate churn prediction economically valuable.

This project addresses the binary classification problem of predicting whether a bank customer will leave ( $\text{churn} = 1$ ) or stay ( $\text{churn} = 0$ ) based on demographic and financial characteristics. The input to our algorithm consists of ten features describing each customer: credit score, country of residence, gender, age, tenure (years with the bank), account balance, number of products held, credit card ownership, active membership status, and estimated salary. We use multiple classification algorithms including Random Forest, AdaBoost, XGBoost, Support Vector Machine (SVM), and Logistic Regression to output a predicted churn probability and binary classification.

The primary objective is to compare ensemble methods (bagging vs boosting approaches) and evaluate which algorithm achieves the best performance across multiple metrics. We also examine the effect of feature engineering and analyze which customer characteristics are most predictive of churn behavior. You can find code of project here: [https://github.com/hpietukhin/churn/blob/master/churn\\_Pietukhin.ipynb](https://github.com/hpietukhin/churn/blob/master/churn_Pietukhin.ipynb)

## 2 Dataset and features

### 2.1 Data description

The dataset used in this project is the Bank Customer Churn Dataset from Kaggle<sup>1</sup>. It contains records for 10,000 customers of ABC Multistate Bank, where each row represents one customer.

The features available are:

- `credit_score`: customer's credit score (integer)
- `country`: country of residence (France, Germany, or Spain)
- `gender`: customer gender (Male or Female)
- `age`: customer age in years
- `tenure`: number of years as a bank customer
- `balance`: account balance
- `products_number`: number of bank products used by customer
- `credit_card`: whether customer has a credit card (0 or 1)
- `active_member`: whether customer is an active member (0 or 1)
- `estimated_salary`: estimated annual salary

---

<sup>1</sup><https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset>

- churn: target variable (1 = customer left, 0 = customer stayed)

The customer\_id column was removed as it carries no predictive information.

## 2.2 Class distribution

The target variable is moderately imbalanced: 7,963 customers (79.6%) stayed with the bank while 2,037 customers (20.4%) churned. This imbalance is not extreme but requires attention during model training. All models used class weighting or scale adjustment to handle this.

## 2.3 Data preprocessing

The dataset contained no missing values, so imputation was unnecessary. Categorical variables (country, gender) were one-hot encoded using dummy variables with one category dropped to avoid multicollinearity. Binary variables (credit\_card, active\_member) were kept as integers.

Numerical features were standardized using z-score normalization for models that require scaled input (SVM and Logistic Regression). Tree-based models (Random Forest, AdaBoost, XGBoost) were trained on unscaled data since they are invariant to monotonic transformations.

## 2.4 Feature engineering

One new feature was created: balance\_to\_salary\_ratio, computed as:

$$\text{balance\_to\_salary\_ratio} = \frac{\text{balance}}{\text{estimated\_salary} + 1}$$

The +1 in the denominator prevents division by zero. This feature captures how much of their income equivalent a customer keeps in the bank, which may correlate with churn behavior differently than balance or salary alone.

## 2.5 Data splitting

The data was split into three sets using stratified sampling to preserve class proportions:

Set	Samples	Churn=0	Churn=1	Churn rate
Training	7,000	5,574	1,426	20.4%
Validation	1,500	1,194	306	20.4%
Test	1,500	1,195	305	20.3%

Table 1: Data split summary

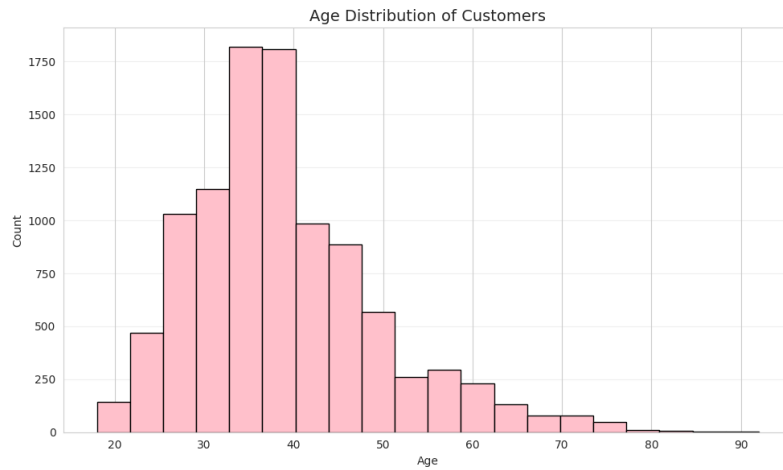


Figure 1: Distribution of customer age. Most customers are between 30 and 40 years old.

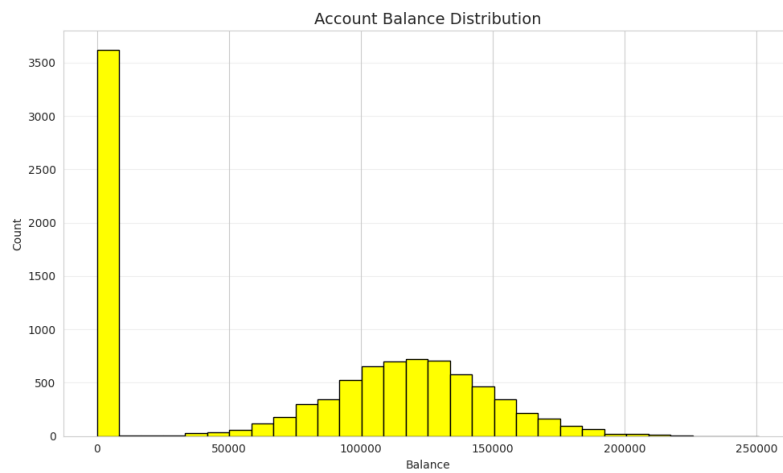


Figure 2: Distribution of account balance. A large proportion of customers have zero balance.

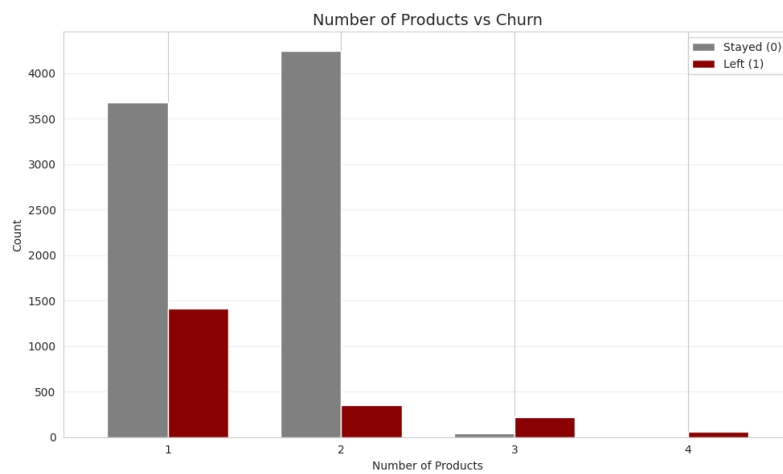


Figure 3: Number of products vs churn. Customers with 4 products show 100% churn rate.

## 2.6 Correlation analysis

A correlation matrix was computed for numerical features. The features least correlated with churn were tenure, credit\_card, and estimated\_salary. Age showed the strongest positive correlation with churn among numerical features.

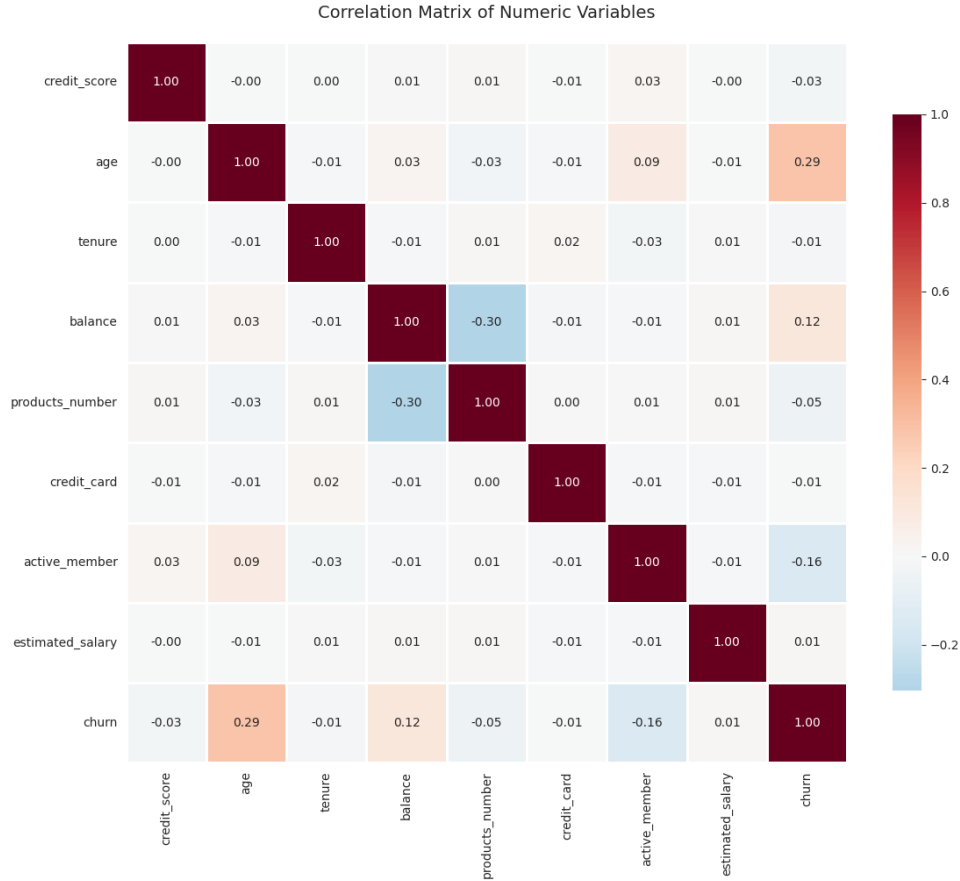


Figure 4: Correlation matrix of numerical variables

Variance Inflation Factor (VIF) analysis was performed to check for multicollinearity among predictors. All VIF values were less than 1.5, suggesting no concerning multicollinearity. VIF measures how much the variance of a regression coefficient is inflated due to linear dependence on other predictors. A VIF above 5-10 would suggest problematic multicollinearity.

## 3 Methods

Five classification algorithms were applied to this problem. All belong to well-established families of machine learning methods.

### 3.1 Random Forest

Random Forest is an ensemble method based on bagging (bootstrap aggregating). It constructs multiple decision trees, where each tree is trained on a bootstrap sample of the training data, and at each node split, only a random subset of features is considered. The final prediction combines all trees through majority voting.

The ensemble approach reduces variance compared to a single decision tree while maintaining low bias. The randomness in both sample selection and feature selection decorrelates the trees, making their average more stable than any individual tree.

For this project, Random Forest was regularized with `max_depth=10`, `min_samples_leaf=20`, and `min_samples_split=50` to prevent overfitting. Initial experiments without regularization achieved 100% training accuracy, indicating severe overfitting. The model used 100 trees and `class_weight='balanced'` to handle class imbalance.

### 3.2 AdaBoost

AdaBoost (Adaptive Boosting) is a boosting algorithm that trains weak learners sequentially. The key idea is that each subsequent learner focuses on the examples that previous learners misclassified.

The algorithm works as follows: initially, all training samples have equal weight. After training a weak learner (typically a shallow decision tree), samples that were misclassified receive higher weights, while correctly classified samples receive lower weights. The next learner is then trained on this reweighted distribution. The final prediction is a weighted vote of all learners, where each learner's weight depends on its accuracy.

Mathematically, for sample  $i$  after iteration  $t$ , the weight update is:

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp(\alpha_t \cdot \mathbf{1}[y_i \neq h_t(x_i)])$$

where  $\alpha_t$  is the learner weight based on its error rate.

The implementation used 100 estimators with default parameters.

### 3.3 XGBoost

XGBoost (Extreme Gradient Boosting) is a gradient boosting algorithm that offers several improvements over AdaBoost:

- Uses gradient descent on a differentiable loss function rather than sample weight adjustment
- Includes L1 and L2 regularization terms in the objective function to prevent overfitting
- Handles missing values natively
- Supports early stopping based on validation performance

The objective function at iteration  $t$  is:

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

where  $l$  is the loss function and  $\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \|w\|^2$  is the regularization term with  $T$  being the number of leaves and  $w$  being leaf weights.

For this project, XGBoost was configured with `max_depth=4`, `learning_rate=0.05`, and up to 500 estimators with early stopping patience of 30 rounds. The `scale_pos_weight` parameter was set to the ratio of negative to positive samples (approximately 3.9) to handle class imbalance. Early stopping triggered at iteration 154.

### 3.4 Support Vector Machine

SVM with RBF (Radial Basis Function) kernel finds a nonlinear decision boundary by implicitly mapping features to a higher-dimensional space where classes become linearly separable.

The RBF kernel is defined as:

$$K(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$$

The optimization problem finds support vectors that define the maximum-margin hyperplane in the transformed space. SVM requires scaled features because it is sensitive to the magnitude of input variables. The implementation used `C=1.0`, `gamma='scale'`, and `class_weight='balanced'`.

### 3.5 Logistic Regression

Logistic Regression is a linear model that serves as a baseline. It models the log-odds of the positive class as a linear combination of features:

$$\log \frac{P(y = 1|x)}{1 - P(y = 1|x)} = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$$

Despite its simplicity, Logistic Regression provides interpretable coefficients and often performs reasonably well when the decision boundary is approximately linear. The implementation used L2 regularization and `class_weight='balanced'`.

## 4 Experiments and results

### 4.1 Evaluation metrics

All metrics are computed from the confusion matrix, which contains four values: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

Accuracy measures overall correctness:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity (also called recall) measures the proportion of actual churners correctly identified:

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

Specificity measures the proportion of actual non-churners correctly identified:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Precision measures the proportion of predicted churners who actually churned:

$$\text{Precision} = \frac{TP}{TP + FP}$$

F1-score is the harmonic mean of precision and sensitivity:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

AUC (Area Under the ROC Curve) measures discrimination ability across all classification thresholds. The ROC curve plots true positive rate (sensitivity) against false positive rate (1 - specificity) at various threshold settings. AUC = 1 indicates perfect discrimination; AUC = 0.5 indicates random guessing.

Gini coefficient is a linear transformation of AUC commonly used in credit scoring:

$$\text{Gini} = 2 \cdot \text{AUC} - 1$$

Gini ranges from 0 (random) to 1 (perfect). A model with AUC = 0.85 has Gini = 0.70. The Gini coefficient can be interpreted as the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example, scaled to [0,1]. In banking applications, Gini above 0.4 is typically considered acceptable, and above 0.6 is considered good.

## 4.2 Confusion matrices

The confusion matrices for all models on the test set are:

Random Forest:

actual / predicted	0	1
0	989	206
1	92	213

AdaBoost:

actual / predicted	0	1
0	1132	63
1	172	133

XGBoost:

actual / predicted	0	1
0	989	206
1	68	237

SVM:

actual / predicted	0	1
0	954	241
1	79	226

Logistic Regression:

actual / predicted	0	1
0	864	331
1	102	203



### 4.3 Performance comparison

Metric	Random Forest	AdaBoost	XGBoost	SVM	Logistic Reg
Accuracy	0.8013	0.8433	0.8173	0.7867	0.7113
Sensitivity	0.6984	0.4361	0.7770	0.7410	0.6656
Specificity	0.8276	0.9473	0.8276	0.7983	0.7230
Precision	0.5084	0.6786	0.5350	0.4839	0.3801
F1-score	0.5884	0.5309	0.6337	0.5855	0.4839
AUC	0.8487	0.8521	0.8743	0.8545	0.7510
Gini	0.6975	0.7042	0.7486	0.7091	0.5021

Table 2: Model performance on test set

The results show clear patterns:

XGBoost achieved the best overall performance with the highest AUC (0.8743), highest sensitivity (0.7770), and highest F1-score (0.6337). Its Gini coefficient of 0.7486 indicates strong discriminative ability.

AdaBoost achieved the highest accuracy (0.8433) and specificity (0.9473), meaning it was most conservative in predicting churn. However, its sensitivity was the lowest among ensemble methods (0.4361), indicating it missed many actual churners.

Random Forest and SVM showed similar performance across most metrics, with AUC around 0.85.

Logistic Regression performed worst overall, suggesting the decision boundary is not linear in the original feature space.

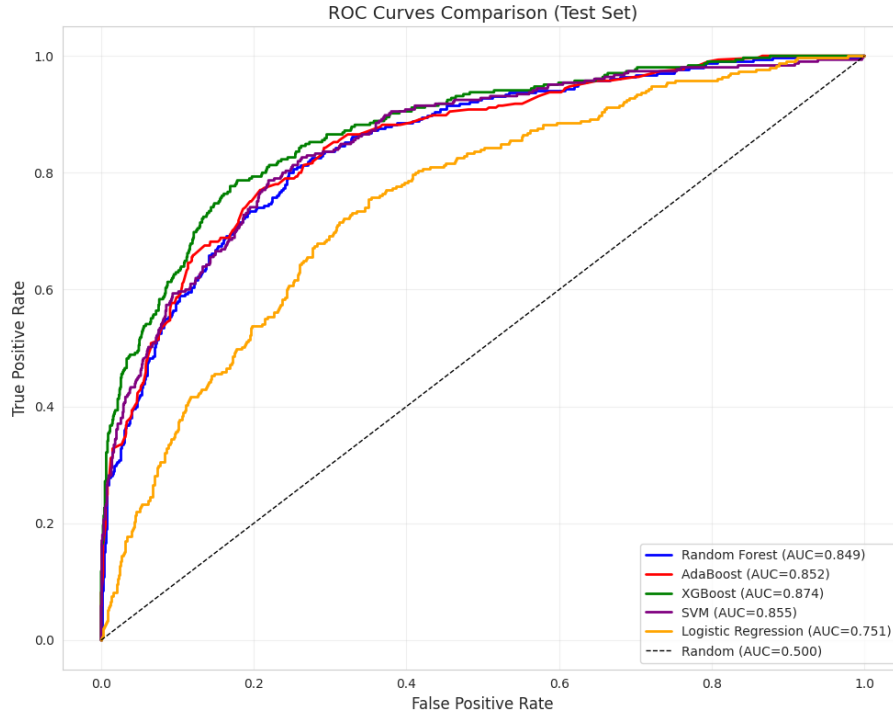


Figure 5: ROC curves for all models on test set. XGBoost achieves the highest AUC.

## 4.4 Overfitting analysis

Learning curves were generated to diagnose bias-variance tradeoff. XGBoost showed a healthy gap of approximately 0.03 between training and validation AUC, indicating good generalization without significant overfitting. The early stopping mechanism prevented the model from overtraining.

Random Forest without regularization showed 100% training accuracy with lower test accuracy, indicating overfitting. After adding depth and leaf constraints, the training-test gap became reasonable.

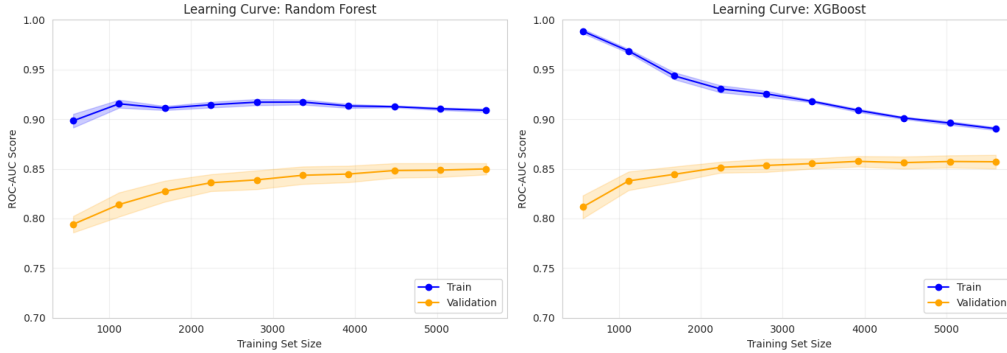


Figure 6: Learning curves for Random Forest and XGBoost showing AUC vs training set size

## 4.5 Feature importance

Tree-based models provide feature importance scores. For both Random Forest and XGBoost, the most important features were age, number of products, and balance. The engineered feature `balance_to_salary_ratio` appeared in the top features for XGBoost, suggesting it captures useful information.

## 5 Conclusion and future work

This project compared five classification algorithms for predicting bank customer churn. XGBoost achieved the best overall performance with  $AUC = 0.8743$  and  $Gini = 0.7486$ , followed closely by SVM ( $AUC = 0.8545$ ) and AdaBoost ( $AUC = 0.8521$ ).

The success of XGBoost can be attributed to several factors: its built-in regularization prevents overfitting, early stopping finds the optimal number of iterations automatically, and gradient boosting is generally more sample-efficient than bagging methods like Random Forest.

AdaBoost showed an interesting tradeoff, achieving the highest accuracy and specificity at the cost of low sensitivity. This makes it suitable for applications where false positives are costly but missing actual churners is acceptable. In contrast, XGBoost's higher sensitivity makes it better suited for proactive retention campaigns where identifying potential churners is the priority.

The linear Logistic Regression baseline performed substantially worse than all nonlinear methods, confirming that the churn decision boundary in this dataset is nonlinear.

For future work, several directions could improve results. Hyperparameter optimization using grid search or Bayesian optimization could fine-tune model parameters. Additional feature engineering based on domain knowledge, such as interaction terms or behavioral patterns over time, might capture more complex relationships. If longitudinal data were available, recurrent neural networks or temporal models could exploit sequential patterns in customer behavior. Finally, ensemble methods combining predictions from multiple models could further improve robustness.