
How ACCURATE can I make it?

Predicting Default Risk in Loans

Anthony Tian



Problem

When credit card customers default...

- Banks lose revenue
- Fewer loans get approved
- Collections drain staff time & legal resources
- Mispriced credit raises rates for everyone
- Undermine banks' trust

Predicting who'll default can prevent losses!

Dataset

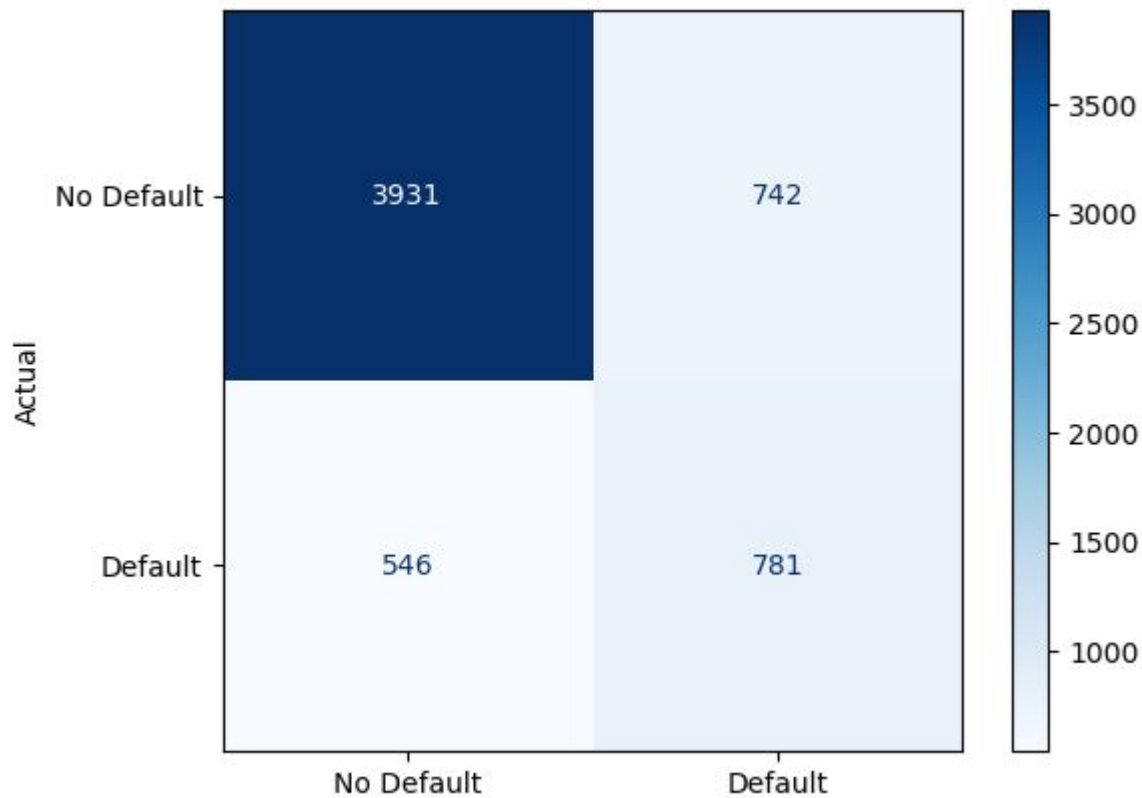
	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x15	x16	x17	x18	x19	x20	x21	x22	x23	y
0	1.0	20000.0	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	-1.0	...	0.0	0.0	0.0	0.0	689.0	0.0	0.0	0.0	0.0	1
1	2.0	120000.0	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	0.0	...	3272.0	3455.0	3261.0	0.0	1000.0	1000.0	1000.0	0.0	2000.0	1
2	3.0	90000.0	2.0	2.0	2.0	34.0	0.0	0.0	0.0	0.0	...	14331.0	14948.0	15549.0	1518.0	1500.0	1000.0	1000.0	1000.0	5000.0	0
3	4.0	50000.0	2.0	2.0	1.0	37.0	0.0	0.0	0.0	0.0	...	28314.0	28959.0	29547.0	2000.0	2019.0	1200.0	1100.0	1069.0	1000.0	0
4	5.0	50000.0	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	0.0	...	20940.0	19146.0	19131.0	2000.0	36681.0	10000.0	9000.0	689.0	679.0	0
...
29995	29996.0	220000.0	1.0	3.0	1.0	39.0	0.0	0.0	0.0	0.0	...	88004.0	31237.0	15980.0	8500.0	20000.0	5003.0	3047.0	5000.0	1000.0	0
29996	29997.0	150000.0	1.0	3.0	2.0	43.0	-1.0	-1.0	-1.0	-1.0	...	8979.0	5190.0	0.0	1837.0	3526.0	8998.0	129.0	0.0	0.0	0
29997	29998.0	30000.0	1.0	2.0	2.0	37.0	4.0	3.0	2.0	-1.0	...	20878.0	20582.0	19357.0	0.0	0.0	22000.0	4200.0	2000.0	3100.0	1
29998	29999.0	80000.0	1.0	3.0	1.0	41.0	1.0	-1.0	0.0	0.0	...	52774.0	11855.0	48944.0	85900.0	3409.0	1178.0	1926.0	52964.0	1804.0	1
29999	30000.0	50000.0	1.0	2.0	1.0	46.0	0.0	0.0	0.0	0.0	...	36535.0	32428.0	15313.0	2078.0	1800.0	1430.0	1000.0	1000.0	1000.0	1

30000 rows x 25 columns

- n = 30k loans, 22% charge-off rate" (I. Yeh, Che-hui Lien. 2009)
- UCI Default of Credit Card Clients
 - [Original Source](#)
- Renamed columns
 - according to source guide
- Key features:
 - numeric: limit_balance, age, bill_amt1-6, pay_amt1-6
 - categorical: sex, education_level, marital_status, repayment_status (PAY_O-PAY_6), default_next_month (target)

Preview

Confusion Matrix - XGBoost v7





CATCH

**catches 6 of 10
defaulters**

Recall(0): 0.59

TRUE

**half of flags
come true**

Precision(0): 0.51



SAFE

**only 12%
false flags**

Precision(1): 0.88

here's how...

Journey

From one model to the next...

01

Log. Regression

Spaghetti Throwing– trying to
see what sticks

02

Random Forest

Could fit non-linear features
(limit_bal, late_trend, etc)

04

SVM

Kernels would handle
bill/payment splits

03

XGBoost

Handles complex pay/bill
interactions – best thus far

Journey

XGB turned out best... now how can I improve it?

05

Features (XGB)

Engineered five features
to increase accuracy

06

Tuned (XGB)

grid-searched to tighten fit

07

Balancing (XGB)

Pivoted to balancing,
oversampled minority class to
lift recall

08

Calibrating (XGB)

Align probability scores, set
F1-max cutoff

01 Logistic Regression

(80/20 split, x: all except default, y: default)

	precision	recall	f1-score	support
0	0.82	0.97	0.89	4673
1	0.69	0.24	0.36	1327
accuracy			0.81	6000
macro avg	0.75	0.61	0.62	6000
weighted avg	0.79	0.81	0.77	6000

Thought Process:

Logistic Regression:

- Handles binary classification directly
- Works well with numeric-heavy data
- First model: just throwing something out

Performance?

- Precision on defaulters: 0.51
- Recall: 0.59 F1: 0.55
- Still likely underfitting complex patterns
- No handling of class imbalance or interactions yet

02 Random Forest

(80/20 split, x: all except default, y: default)

	precision	recall	f1-score	support
0	0.83	0.95	0.89	4673
1	0.65	0.33	0.44	1327
accuracy			0.81	6000
macro avg	0.74	0.64	0.66	6000
weighted avg	0.80	0.81	0.79	6000

Thought Process:

Random Forest:

- Handles non-linear features better
- Uses ensemble power to reduce overfitting
- Balances class weight to address imbalance
- Goal: Catch more true defaulters than logistic reg

Performance?

- Precision on defaulters: 0.65

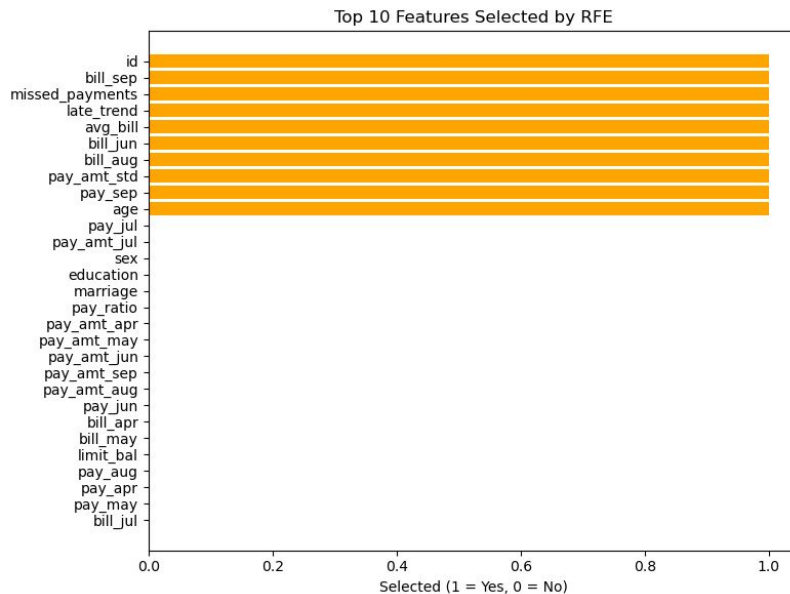
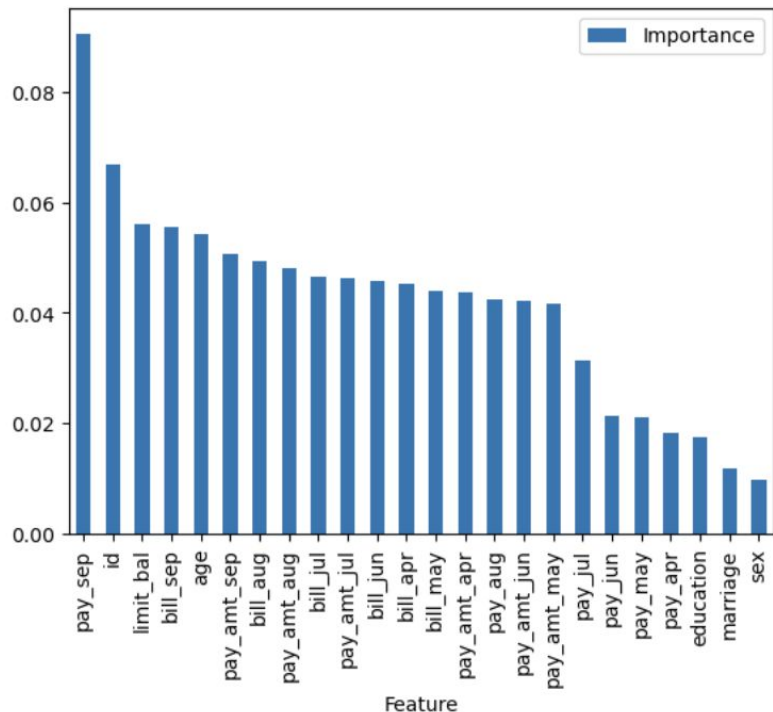
Recall: 0.33 F1: 0.44

- Better once it does flag a defaulter
- Still misses ~67% of them
- Small step forward, but still cautious

Found 10 most predictive features using RF importances

- wanted to understand what signals model relied on
 - hoped to reduce noise

=> top 10 = ['id', 'bill_aug', 'pay_amt_sep', 'bill_apr', 'bill_may', 'bill_jun', 'bill_jul', 'bill_sep', 'pay_sep', 'age']



02 Random Forest v2

(80/20 split, x: top 10 (from prev. page), y: default)

	precision	recall	f1-score	support
0	0.83	0.95	0.89	4673
1	0.64	0.34	0.44	1327
accuracy			0.81	6000
macro avg	0.74	0.64	0.67	6000
weighted avg	0.79	0.81	0.79	6000

ROC AUC: 0.76

Thought Process:

Random Forest v2:

- Cut model complexity by using just top 10 features
- Chose based on RF importances + RFE
- Hoped to retain signal while reducing noise + boosting clarity

Performance?

– Precision on defaulters: 0.64

Recall: 0.34 F1: 0.44

- Nearly identical to full-feature RF
- Inch of improvement recall: 0.33 → 0.34
- Simplified model without major loss
- Should be able to use these features for future models

03 SVM

(80/20 split, x: top 10 , y: default)

Thought Process:

SVM:

- Tried a boundary-based model (SVM) to flag risk patterns more decisively
- Used same top 10 features for fairness
- Hoped to push recall much higher, even at cost of precision

	precision	recall	f1-score	support
0	0.88	0.30	0.45	4673
1	0.26	0.86	0.40	1327
accuracy			0.42	6000
macro avg	0.57	0.58	0.42	6000
weighted avg	0.74	0.42	0.43	6000

Performance?

- Precision on defaulters: 0.26 Recall: 0.86 F1: 0.40
- Huge recall jump (↑ from 0.34), barely misses any defaulters
- But 74% of defaulter flags are false alarms
- Overall accuracy: 42%
- Likely overfit to risky cases — tries too hard to catch them all
- Possibly:
 - RBF kernel overreacts to patterns in small feature space
 - Class_weight = 'balanced' + nonlinear boundary = extreme shifts
 - Lack of calibration makes it trigger-happy

Massive trade-off: SVM aggressively flags defaulters (high recall), but sacrifices overall accuracy and precision.

04 XGBoost

(80/20 split, x: top 10, y: default)

	precision	recall	f1-score	support
0	0.87	0.83	0.85	4673
1	0.49	0.58	0.53	1327
accuracy			0.77	6000
macro avg	0.68	0.70	0.69	6000
weighted avg	0.79	0.77	0.78	6000

Thought Process:


- Tried XGBoost to better capture nonlinear patterns
- Set scale_pos_weight to help with imbalance
- Wanted a model that could balance recall + precision with more nuance than SVM or RF

Performance?

- Precision on defaulters: 0.49
Recall: 0.58 F1: 0.53
- Most balanced model so far
- Catches a solid % of defaulters (↑ recall vs RF)
- Accuracy = 77% despite fewer features
- Likely benefiting from boosting: iteratively learning where others fail
- May underperform if more interaction terms or deeper features are needed

04 XGBoost v2

+ tuning (randomized)



	precision	recall	f1-score	support
0	0.87	0.83	0.85	4673
1	0.49	0.57	0.53	1327
accuracy			0.77	6000
macro avg	0.68	0.70	0.69	6000
weighted avg	0.79	0.77	0.78	6000

Thought Process:

Hypertuning might help:

- Tried RandomizedSearchCV and Optuna to push performance further
- Targeted f1-score and class balance using scale_pos_weight
- Hoped to reduce false positives without sacrificing recall

Attempt 1:

Parameters tuned:

- n_estimators: [100, 200, 300, 400]
- max_depth: [3, 4, 5, 6, 7]
- learning_rate: [0.01, 0.05, 0.1, 0.2]
- subsample: [0.6, 0.8, 1.0]
- colsample_bytree: [0.6, 0.8, 1.0]
- gamma: [0, 1, 5]

Why / Hope:

- Start with basic knobs
- Aim: catch more defaulters while reducing false positives
- Expected a bump in F1 by balancing fit depth and learning rate

04 XGBoost v3

+ tuning w/ broader search



	precision	recall	f1-score	support
0	0.87	0.84	0.85	4673
1	0.49	0.55	0.52	1327
accuracy			0.78	6000
macro avg	0.68	0.69	0.69	6000
weighted avg	0.79	0.78	0.78	6000

Thought Process:

Broadened search (50 trials)

Attempt 2:

Extra parameters added:

- min_child_weight: [1, 5, 10]
- reg_alpha: [0, 0.1, 1]
- reg_lambda: [0.1, 1, 5]
- Slightly broader ranges for existing params
- learning_rate: [0.001, 0.01, 0.05, 0.1, 0.2]
- n_estimators: [100, 200, 300, 500]
- max_depth: [3, 5, 7, 9]

Why / Hope:

- Try regularization to prevent overfitting
- Add flexibility in tree shape and depth
- See if interaction of new knobs would uncover better balance

04 XGBoost v4

+ tighter tuning

	precision	recall	f1-score	support
0	0.87	0.84	0.86	4673
1	0.50	0.57	0.53	1327
accuracy			0.78	6000
macro avg	0.69	0.71	0.69	6000
weighted avg	0.79	0.78	0.78	6000

Performance?

After three tuning attempts:

- Precision: 0.50 Recall: 0.57 F1: 0.53 (unchanged)
- Nearly identical to untuned XGB; accuracy plateaued at 78%

=> Takeaway: Tuning's juice has been squeezed— time to move on

Thought Process:

Attempt 3:

(All previous params with tighter float sampling)

- learning_rate: 0.0096
- gamma: 4.15
- min_child_weight: 8
- reg_alpha: 0.40
- reg_lambda: 1.04
- n_estimators: 435
- colsample_bytree: 0.80, subsample: 0.80

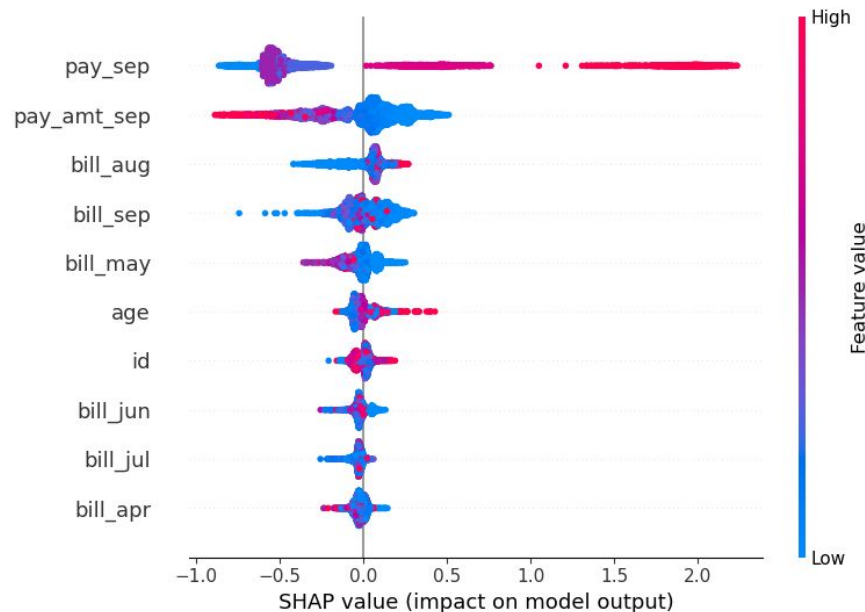
Why / Hope:

- Use smarter search (Optuna) to escape local optima
- Finer tuning to push class 1 performance up
- Hoping for marginal lift from smaller false positive pool

What to try next?

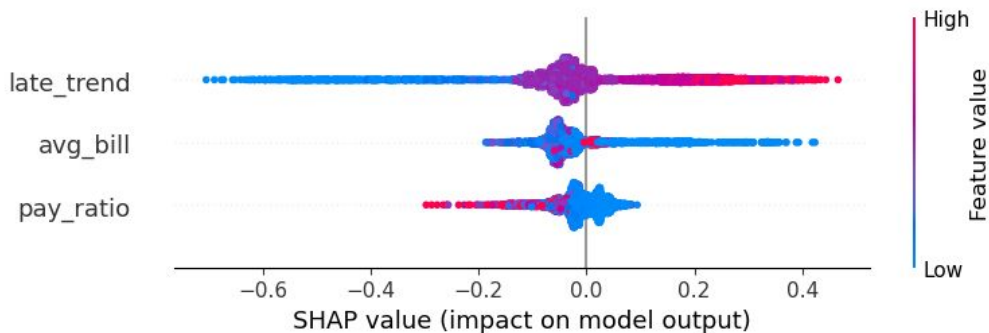


- **Feature Importance (SHAP)**
 - how XGBoost is deciding
 - prioritize those features
- **Findings:**
 - pay_sep dominates → late payments spike default risk
 - Big bills (bill_aug, bill_sep, bill_may) push risk ↑
 - Higher payments (pay_amt_sep) pull risk ↓
 - Age tilts slightly (younger = riskier)
- **Hope: less noise & more focus => better model**



- **Feature Engineering**

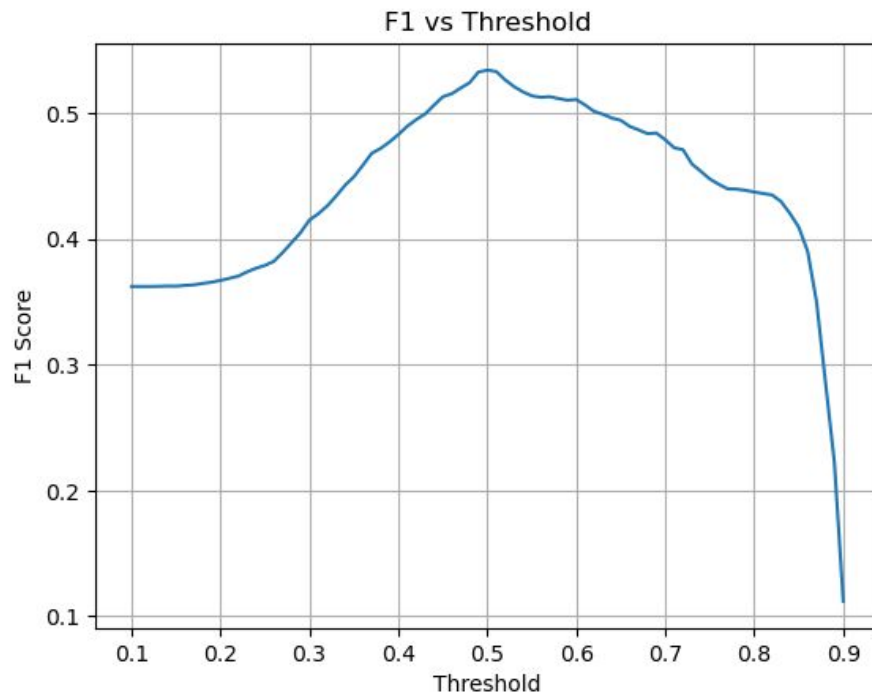
- Tuning & threshold both plateaued → model could use new signals



- **Goal: create relevant columns**

- combining pay + bill history so trees can spot patterns
- **pay_ratio:**
(pay_amt_sep / bill_sep)
- **avg_bill:**
mean(bills Apr-Sep)
- **late_trend:**
(Sep bill - Aug bill)
- **missed_payments:**
of billing cycles paid late
- **pay_amt_std**
stdev of payment amounts
(how inconsistent they are)

- **Threshold Tuning**
 - find potentially more accurate F1 cut-off
 - default 0.50 cut-off might not fit skewed data
- **Findings**
 - F1 peaks ≈ 0.53 at threshold
 - turns out 0.50 was okay
 - Still worth trying
- Still worth trying



04 XGBoost v5

+ threshold tuning & new features

	precision	recall	f1-score	support
0	0.84	0.95	0.89	4673
1	0.67	0.35	0.46	1327
accuracy			0.82	6000
macro avg	0.76	0.65	0.68	6000
weighted avg	0.80	0.82	0.80	6000

Performance?

- precision on defaulters: 0.67 recall: 0.35
F1: 0.46
- recall ↑ from 0.34 → 0.35, F1 ↑ from 0.44
→ 0.46
- accuracy: 82%
- modest lift shows new signals add value
- still not quite where I want it though

Thought Process:

- new frequency + volatility signals might catch more risk patterns:
- `pay_ratio` – September payment ÷ September bill size → “how much of the bill did they actually cover?”
- `avg_bill` – mean bill size from April-September → typical burden level
- `late_trend` – Sep payment minus Aug payment → catches worsening payment behaviour
- `missed_payments` – count of months with any late payment → frequency of being behind
- `pay_amt_std` – stdev of payments Apr-Sep → variability / inconsistency in what they pay

04 XGBoost v6

+ balancing classes

	precision	recall	f1-score	support
0	0.88	0.85	0.86	4673
1	0.52	0.58	0.55	1327
accuracy			0.79	6000
macro avg	0.70	0.71	0.70	6000
weighted avg	0.80	0.79	0.79	6000

Thought Process:

- dataset's been imbalanced (most people pay their debts! => class 0 > class 1)
- likely why recall's been low
- balanced it w/ XGB's built-in class weighting
(scale_pos_weight = #neg/#pos)
- gave extra weight to defaulters so model focuses on true positives
- found F1-maximizing cutoff again (~0.55)

Performance?

- recall jumped from 0.38 → 0.58, F1 from 0.48 → 0.55
- precision dipped from 0.65 → 0.52
- accuracy ~80% held steady
- class weighting + threshold tuning outperformed oversampling for feature set

04 XGBoost v7

+ isotonic calibration

	precision	recall	f1-score	support
0	0.88	0.84	0.86	4673
1	0.51	0.59	0.55	1327
accuracy			0.79	6000
macro avg	0.70	0.71	0.70	6000
weighted avg	0.80	0.79	0.79	6000

Thought Process:

- raw XGB scores weren't true probabilities (e.g., 0.60 didn't equal 60 % default risk)
- used CalibratedClassifierCV (method = 'isotonic', cv = 5); adds small “stretch/squeeze” curve to match model's raw scores w/ reality
- => best isotonic threshold: 0.26

Performance?

- precision 0.51, recall 0.59, F1 0.55 (same F1 as before)
- now probabilities are trustworthy: a 0.40 score \approx 40 % observed default rate
- calibration gives better decision flexibility without hurting headline metrics

Final Model (at least for now)

Class 1 (defaulters):

- precision 0.51 → half the flagged borrowers do default

recall 0.59 → we catch ~6 out of 10 real defaulters

F1 0.55 → our best balance so far

Class 0 (non-defaulters):

-precision 0.88

→ mostly true positives

-recall 0.84

→ only 16 % false positives

Overall Metrics

- accuracy 0.79 (skewed data, so less meaningful)

- macro-avg F1 0.70

→ solid, given 22 % positives

	precision	recall	f1-score	support
0	0.88	0.84	0.86	4673
1	0.51	0.59	0.55	1327
accuracy			0.79	6000
macro avg	0.70	0.71	0.70	6000
weighted avg	0.80	0.79	0.79	6000

Thanks!

Questions?

tian.ruy@northeastern.edu

showhq2@gmail.com

+1 (626) 677-0875

site

CREDITS: This presentation template was created by [**Slidesgo**](#), and includes icons by [**Flaticon**](#), and infographics & images by [**Freepik**](#)

