**Loan Policy Exercise: Choosing a Probability of Default (PD) Cutoff**

Logistic regression python code for exercise:

#########################################

#  Code: Train a PD Model and Make Approval Decisions

# Python (scikit-learn) - Logistic Regression for Credit Risk (PD)

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_auc\_score, average\_precision\_score, confusion\_matrix, classification\_report

from sklearn.utils.class\_weight import compute\_class\_weight

df = pd.read\_csv("/content/loan\_applications.csv")

target\_col = "default\_12m"

categorical = ["purpose", "home\_ownership", "channel", "region", "loan\_term\_months"]

numeric = ["age", "annual\_income", "employment\_length", "credit\_score", "debt\_to\_income",

          "num\_open\_accounts", "delinquencies\_2y", "inquiries\_6m", "loan\_amount", "interest\_rate"]

X = df[categorical + numeric]

y = df[target\_col].astype(int)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42, stratify=y)

# Class weights for imbalance

classes = np.array([0,1])

cw = compute\_class\_weight("balanced", classes=classes, y=y\_train)

cw\_dict = {cls:w for cls,w in zip(classes, cw)}

preprocess = ColumnTransformer([

   ("num", StandardScaler(), numeric),

   ("cat", OneHotEncoder(handle\_unknown="ignore"), categorical),

])

lr = LogisticRegression(max\_iter=2000, class\_weight=cw\_dict, solver="lbfgs")

pipe = Pipeline([("prep", preprocess), ("model", lr)])

pipe.fit(X\_train, y\_train)

# Evaluate

y\_proba = pipe.predict\_proba(X\_test)[:,1]

print("ROC-AUC:", roc\_auc\_score(y\_test, y\_proba))

print("PR-AUC :", average\_precision\_score(y\_test, y\_proba))

# Policy: approve if PD < 0.05

pd\_cutoff = 0.05

y\_pred\_policy = (y\_proba >= pd\_cutoff).astype(int)  # 1 = predict default (reject)

print("Confusion Matrix (reject=1 at cutoff 0.05):\n", confusion\_matrix(y\_test, y\_pred\_policy))

print(classification\_report(y\_test, y\_pred\_policy, digits=3))

**Exercise:**

Below are confusion-matrix summaries at three policy cutoffs (5%, 8%, 12%). A lower cutoff is stricter (more rejections). Use these to evaluate trade-offs and compute business costs using the Excel worksheet. 5% is prefilled for you.

Rerun the line of code above changing the pd\_cutoff to 0.08 and 0.12.

Fill in the empty cells and assess the precision and accuracy of each model.

**Confusion Matrix Summary**

| **PD Cutoff** | **TP** | **FP** | **TN** | **FN** | **Precision (rejects)** | **Recall (defaults caught)** | **Approval Rate** | **Rejection Rate** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5.00% | 483 | 2016.0 | 1.0 | 0.0 | 0.193 | 1.0 | 0.000 | 1.000 |
| 8.00% |  |  |  |  |  |  |  |  |
| 12.00% |  |  |  |  |  |  |  |  |

**Task**

1. Download the Excel sheet *pd\_threshold\_costs.xlsx* and inspect the **Cutoffs** tab for the counts above.
2. Open the **Costs\_5pc**, **Costs\_8pc**, and **Costs\_12pc** tabs. Adjust *Cost per FP* and *Cost per FN* to reflect realistic values for your bank.
3. Compare **Total Cost** and **Cost per Application** across the three policies and recommend a cutoff. Explain your choice.

Tip: You can also explore PD cutoffs at 7% or 10% by re-running the notebook and adding rows to the worksheet.

Appendix:

Here’s a full breakdown of your **cutoff = 0.05 results**, explained step by step:

**1. ROC-AUC: 0.653**

* **What it measures:** Ability of the model to distinguish between good borrowers and defaulters across *all thresholds*.
* **Interpretation:** 0.653 > 0.5, so it’s better than random, but still modest discriminatory power.

**2. PR-AUC: 0.309**

* **What it measures:** Trade-off between precision and recall, useful in imbalanced data.
* **Interpretation:** 0.309 shows the model can’t maintain high precision and recall together — it tends to sacrifice one for the other.

**3. Confusion Matrix (cutoff = 0.05)**

[[ 1 2016]

[ 0 483]]

* Format:
* [[TN FP]
* [FN TP]]
  + **TN = 1** → 1 good borrower correctly approved.
  + **FP = 2016** → 2016 good borrowers wrongly rejected.
  + **FN = 0** → No defaulters wrongly approved.
  + **TP = 483** → All 483 defaulters correctly rejected.

**Interpretation:** The model rejects nearly everyone. It catches *all defaults*, but also wrongly rejects nearly every good customer.

**4. Classification Report**

**Class 0 = Good borrowers**

* **Precision = 1.000** → Of those *predicted good*, 100% really were good (but there was only 1 case!).
* **Recall = 0.000** → It almost never approves good customers (caught 1 out of 2017).
* **F1 = 0.001** → Essentially useless for good borrowers.
* **Support = 2017** → Number of good borrowers.

**Class 1 = Defaulters**

* **Precision = 0.193** → Only 19.3% of rejected applicants were truly defaulters; the rest were good borrowers wrongly rejected.
* **Recall = 1.000** → Every default was caught.
* **F1 = 0.324** → Better than class 0, but still low.
* **Support = 483** → Number of default borrowers.

**5. Accuracy = 0.194**

* Only ~19% of predictions are correct overall.
* This happens because the model *rejects almost everyone*, so it only gets defaults right, not the majority good customers.

**6. Macro Average**

* **Precision = 0.597, Recall = 0.500, F1 = 0.162**
* Average across both classes (treats them equally).
* Shows imbalance: defaults are well-detected, good borrowers are not.

**7. Weighted Average**

* **Precision = 0.844, Recall = 0.194, F1 = 0.063**
* Weighted by class size (good borrowers dominate).
* Looks high for precision because almost all rejections are predicted as default, but recall is terrible (most good customers lost).
* F1 is very low — confirming the model is not useful in practice at this threshold.

✅ **Overall takeaway at cutoff 0.05:**

* The model is **overly strict** → rejects almost everyone.
* It **catches 100% of defaulters** but at the cost of **rejecting 99.95% of good customers**.
* Business-wise: disastrous — customers leave, revenue collapses, even though defaults are avoided.