

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import joblib
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from interpret.glassbox import ExplainableBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import (accuracy_score, classification_report,
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.model_selection import GridSearchCV, StratifiedKFold, c
# Keras (DNN)
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from collections import Counter

```

```

# โหลด train/test CSV ที่เราเซฟไว้ก่อนหน้านี้
train_path = '/Users/tanananyathongkum/Jenny/Visual Studio Code/Pro
test_path = '/Users/tanananyathongkum/Jenny/Visual Studio Code/Pro

# ถ้าไฟล์ชื่ออื่น ให้แก้ path ข้างบน

data_train = pd.read_csv(train_path)
data_test = pd.read_csv(test_path)

```

```

# Create directories for models and results
base_path = '/Users/tanananyathongkum/Jenny/Visual Studio Code/Proj
model_dir = os.path.join(base_path, 'model', 'imput')
os.makedirs(model_dir, exist_ok=True)
print(f'Created model directory: {model_dir}')

```

```

Created model directory: /Users/tanananyathongkum/Jenny/Visual Studi

```

```
# Settings
n_runs = 5
results = {
    'Logistic Regression': {'train': [], 'test': []},
    'Random Forest': {'train': [], 'test': []},
    'EBM': {'train': [], 'test': []},
    'SVM': {'train': [], 'test': []},
    'XGBoost': {'train': [], 'test': []},
    'DNN': {'train': [], 'test': []},
}
roc_data = {}
```

```
# ----- Helpers -----
def get_pos_proba_from_estimator(estimator, X):
    """คืน prob ของคลาส 1 ถ้ามี; ถ้าไม่มีจะลอง decision_function -> sigmoid
    if hasattr(estimator, "predict_proba"):
        p = estimator.predict_proba(X)
        return p[:, 1] if p.ndim == 2 and p.shape[1] >= 2 else None
    if hasattr(estimator, "decision_function"):
        s = estimator.decision_function(X)
        return 1.0 / (1.0 + np.exp(-s))
    return None

def save_run_metrics(model_name, run, metrics, *, split=None, params=None):
    row = {
        'model': model_name, 'run': run, 'split': split, 'is_final': True,
        'model_path': model_path, **((params or {}), **metrics)
    }
    detailed_metrics.append(row)
    return row

def save_model(model_name, model, is_dnn=False):
    if is_dnn:
        path = os.path.join(model_dir, f"{model_name.lower().replace(' ', '_')}")
        model.save(path)
    else:
        path = os.path.join(model_dir, f"{model_name.lower().replace(' ', '_')}")
        joblib.dump(model, path)
    print(f"Saved model: {path}")
    return path

def update_results_scalar(results, model_name, split, metrics_row, key):
    val = metrics_row.get(key)
    if val is not None:
        results[model_name][split].append(val)

def log_final_run_metrics(
```

```

model_name,
y_true,
y_pred,
y_prob,
roc_data_dict,
*,
split=None,      # 'train' | 'test'
run=None,        # run index
params=None      # dict ของ best_params_ หรือ config
):
    print(f"\n=== Final-Run Metrics: {model_name}" + (f" [{split}]"
    acc = accuracy_score(y_true, y_pred)
    err_rate = 1.0 - acc
    print(f"Accuracy: {acc:.4f}")
    print(f"Error rate (1-Acc): {err_rate:.4f}")

    print("Classification Report:")
    print(classification_report(y_true, y_pred, digits=4))

    cm = confusion_matrix(y_true, y_pred, labels=[0, 1])
    print("Confusion Matrix [rows=true, cols=pred]:")
    print(cm)

    tn, fp, fn, tp = cm.ravel()
    eps = 1e-12
    sensitivity    = tp / (tp + fn + eps)    # recall+
    specificity    = tn / (tn + fp + eps)
    precision_pos = tp / (tp + fp + eps)    # PPV
    npv           = tn / (tn + fn + eps)
    fprate       = fp / (fp + tn + eps)
    fnrate       = fn / (fn + tp + eps)
    f1           = f1_score(y_true, y_pred, zero_division=0)

    mcc          = matthews_corrcoef(y_true, y_pred)
    kappa        = cohen_kappa_score(y_true, y_pred)
    bal_acc      = balanced_accuracy_score(y_true, y_pred)
    h_loss       = hamming_loss(y_true, y_pred)
    z1_loss      = zero_one_loss(y_true, y_pred)

    support_pos = tp + fn
    support_neg = tn + fp
    prevalence_pos = support_pos / (support_pos + support_neg + eps)
    pred_pos_rate = (tp + fp) / (support_pos + support_neg + eps)
    youden_j = sensitivity + specificity - 1

    print(f"Sensitivity (Recall+): {sensitivity:.4f} | Specificity:
    print(f"Precision (PPV): {precision_pos:.4f} | NPV: {npv:.4f}")
    print(f"FPR: {fprate:.4f} | FNR: {fnrate:.4f} | F1: {f1:.4f}")

```

```

print(f'MCC: {mcc:.4f} | Cohen's kappa: {kappa:.4f} | Balanced A
print(f'Hamming loss: {h_loss:.4f} | Zero-One loss: {z1_loss:.4f
print(f'Prevalence+: {prevalence_pos:.4f} | Pred+ rate: {pred_po

# Prob-based
ll = brier = ap = roc_auc = None
if y_prob is not None:
    try:
        ll = float(log_loss(y_true, y_prob))
        brier = float(brier_score_loss(y_true, y_prob))
        prec, rec, _ = precision_recall_curve(y_true, y_prob)
        ap = float(average_precision_score(y_true, y_prob))
        fpr, tpr, _ = roc_curve(y_true, y_prob)
        roc_auc = float(auc(fpr, tpr))
        if roc_data_dict is not None:
            key = f"{model_name}" + (f"_{split}" if split else "")
            roc_data_dict[key] = (fpr, tpr, roc_auc)
        print(f'Log Loss: {ll:.4f} | Brier score: {brier:.4f} |
    except ValueError:
        print("ข้าม LogLoss/Brier/PR-AUC/ROC: โปรดตรวจว่า y_prob เป็น

return {
    'run': run, 'model': model_name, 'split': split, **(params o
    'tn': int(tn), 'fp': int(fp), 'fn': int(fn), 'tp': int(tp),
    'support_pos': int(support_pos), 'support_neg': int(support_
    'prevalence_pos': float(prevalence_pos),
    'pred_pos_rate': float(pred_pos_rate),
    'youden_j': float(youden_j),
    'accuracy': float(acc),
    'error_rate': float(err_rate),
    'sensitivity': float(sensitivity),
    'specificity': float(specificity),
    'precision_pos': float(precision_pos),
    'npv': float(npv),
    'fpr': float(fprate),
    'fnr': float(fnrate),
    'f1': float(f1),
    'mcc': float(mcc),
    'kappa': float(kappa),
    'balanced_accuracy': float(bal_acc),
    'hamming_loss': float(h_loss),
    'zero_one_loss': float(z1_loss),
    'log_loss': ll,
    'brier': brier,
    'avg_precision': ap,
    'roc_auc': roc_auc,
}

```

```
def plot_cm and save(y_true, y_pred, title, out_path, normalize=False)
```

```

def plot_cm_and_save(y_true, y_pred, title, out_path, normalize=True):
    cm = confusion_matrix(y_true, y_pred, labels=[0,1])
    if normalize:
        denom = cm.sum(axis=1, keepdims=True).clip(min=1)
        cm_disp = cm.astype(float) / denom
    else:
        cm_disp = cm

    fig, ax = plt.subplots(figsize=(4.8, 4.2))
    im = ax.imshow(cm_disp, interpolation='nearest', cmap='Blues')
    ax.set_title(title)
    ax.set_xlabel("Predicted"); ax.set_ylabel("Actual")
    ax.set_xticks([0,1]); ax.set_yticks([0,1])
    ax.set_xticklabels([0,1]); ax.set_yticklabels([0,1])
    fmt = ".2f" if normalize else "d"
    for (i, j), v in np.ndenumerate(cm_disp):
        ax.text(j, i, format(v, fmt), ha='center', va='center', font
    plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
    plt.tight_layout()
    os.makedirs(os.path.dirname(out_path), exist_ok=True)
    plt.savefig(out_path, dpi=150, bbox_inches='tight')
    plt.show()
    return cm

```

```
TARGET_COL = 'Heart Disease Status'
```

```

X_train = data_train.drop(columns=[TARGET_COL])
y_train = data_train[TARGET_COL].astype(int)

```

```

X_test  = data_test.drop(columns=[TARGET_COL])
y_test  = data_test[TARGET_COL].astype(int)

```

```
print('Class distribution (train) before SMOTE:', Counter(y_train))
```

```
Class distribution (train) before SMOTE: Counter({0: 6400, 1: 1600})
```

```

# ===== Logistic Regression + SMOTE (clean & working)
import os, numpy as np, pandas as pd, joblib
import matplotlib.pyplot as plt

```

```

from sklearn.model_selection import StratifiedKFold, GridSearchCV,
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, make_scorer, f1_score
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

```

```

MODEL_NAME = "Logistic Regression"

# --- require globals from your framework ---
for _n in ["X_train", "y_train", "X_test", "y_test", "results", "model_d
    "get_pos_proba_from_estimator", "log_final_run_metrics", "
    "save_model", "plot_cm_and_save"]:
    assert _n in globals(), f"Missing global: {_n}"
if 'detailed_metrics' not in globals():
    detailed_metrics = []
if MODEL_NAME not in results:
    results[MODEL_NAME] = {'train': [], 'test': []}

# ----- safe k for SMOTE (ไม่ให้ > จำนวน minority-1 ในแต่ละ fold) -----
counts = np.bincount(y_train.astype(int))
minor_count = int(counts.min())
major_count = int(counts.max())
base_ratio = minor_count / major_count
k_max = max(1, minor_count - 1)
k_set = sorted({1, min(3, k_max), min(5, k_max)})

# ----- multi-metric scoring (จะได้อ่าน mean_test_* ได้) -----
scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': make_scorer(f1_score, zero_division=0),
    'roc_auc': 'roc_auc'
}
REFIT = 'roc_auc' # ใช้ AUC เลือกโมเดล แต่ยังพิมพ์ metric อื่นจาก CV ได้

# ----- grid (ปรับ sampling_strategy ให้ไม่ต่ำกว่าอัตราเดิมเสมอ) -----
grid_strategies = sorted({round(max(base_ratio, s), 2) for s in [ba

for run in range(1, n_runs + 1):
    print("\n" + "="*75)
    print(f"[{MODEL_NAME}] Run {run}/{n_runs}")

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42

    pipe = ImbPipeline([
        ('smote', SMOTE(sampling_strategy=max(base_ratio, 0.5),
                        k_neighbors=min(5, k_max), random_state=42
        ('clf', LogisticRegression(max_iter=2000, solver='lbfgs', c
                                random_state=42 + run))
    ])

    param_grid = {
        'smote__sampling_strategy': [max(base_ratio, s) for s in [0.25,

```

```

'smote__k_neighbors': k_set,
'clf__C': [0.1, 1, 10],
'clf__solver': ['lbfgs', 'liblinear'],
'clf__penalty': ['l2'],
'clf__class_weight': [None],          # <-- ตรงไว้
'clf__max_iter': [2000],
}

grid = GridSearchCV(pipe, param_grid, cv=skf,
                    scoring=scoring, refit=REFIT, n_jobs=-1)

print("\nSearching best parameters...")
grid.fit(X_train, y_train)

best_est      = grid.best_estimator_
best_params   = dict(grid.best_params_)
bi            = grid.best_index_

# เติมค่าที่ไม่ได้อยู่ในกริด (เพื่อการบันทึก/trace)
best_params.setdefault('smote__random_state', 42 + run)
best_params.setdefault('clf__random_state', 42 + run)

print("\nBest parameters:", best_params)
print("\nScores for best parameters (CV mean):")
print(f"ACC : {grid.cv_results_['mean_test_accuracy'][bi]:.4f}")
print(f"PREC: {grid.cv_results_['mean_test_precision'][bi]:.4f}")
print(f"REC : {grid.cv_results_['mean_test_recall'][bi]:.4f}")
print(f"F1  : {grid.cv_results_['mean_test_f1'][bi]:.4f}")
print(f"AUC : {grid.cv_results_['mean_test_roc_auc'][bi]:.4f}")

# ----- threshold tuning (Youden J บน TRAIN) -----
y_tr_prob_all = get_pos_proba_from_estimator(best_est, X_train)
fpr, tpr, thr = roc_curve(y_train, y_tr_prob_all)
j = tpr - fpr
best_thresh = float(thr[j.argmax()])
print(f"Best threshold by Youden J (train): {best_thresh:.4f}")

# ----- TRAIN metrics -----
y_tr_pred = (y_tr_prob_all >= best_thresh).astype(int)
tr_metrics = log_final_run_metrics(
    MODEL_NAME, y_train, y_tr_pred, y_tr_prob_all, roc_data,
    split="train", run=run, params={**best_params, 'threshold':
    })
row_tr = save_run_metrics(MODEL_NAME, run, tr_metrics, split="t
results[MODEL_NAME]['train'].append(tr_metrics.get('f1', np.nan

cm_train_png      = os.path.join(model_dir, f"cm_logreg_train_r
cm_train_norm_png = os.path.join(model_dir, f"cm_logreg_train_n
plot_cm_and_save(y_train, y_tr_pred, f"CM (TRAIN) - {MODEL_NAME

```

```

plot_cm_and_save(y_train, y_tr_pred, f"CM (TRAIN, normalized) -
row_tr["cm_png"] = cm_train_png; row_tr["cm_norm_png"] = cm_tra

# ----- TEST metrics (ใช้ threshold จาก TRAIN) -----
y_te_prob = get_pos_proba_from_estimator(best_est, X_test)
y_te_pred = (y_te_prob >= best_thresh).astype(int)

te_metrics = log_final_run_metrics(
    MODEL_NAME, y_test, y_te_pred, y_te_prob, roc_data,
    split="test", run=run, params={**best_params, 'threshold':
)
row_te = save_run_metrics(MODEL_NAME, run, te_metrics, split="t
results[MODEL_NAME]['test'].append(te_metrics.get('f1', np.nan)

cm_test_png = os.path.join(model_dir, f"cm_logreg_test_run
cm_test_norm_png = os.path.join(model_dir, f"cm_logreg_test_nor
plot_cm_and_save(y_test, y_te_pred, f"CM (TEST) - {MODEL_NAME}
plot_cm_and_save(y_test, y_te_pred, f"CM (TEST, normalized) - {
row_te["cm_png"] = cm_test_png; row_te["cm_norm_png"] = cm_test

# ----- Learning Curve (AUC) -----
sizes, tr_scores, cv_scores = learning_curve(
    best_est, X_train, y_train, cv=skf, n_jobs=-1, scoring='roc
    train_sizes=np.linspace(0.1, 1.0, 10)
)
plt.figure(figsize=(8,5))
plt.plot(sizes, tr_scores.mean(axis=1), marker='o', label='Trai
plt.plot(sizes, cv_scores.mean(axis=1), marker='s', label='CV A
plt.xlabel('Training examples'); plt.ylabel('ROC-AUC')
plt.title(f'Learning Curves - {MODEL_NAME} Run {run}')
plt.legend(loc='best'); plt.grid(True); plt.tight_layout()
plt.show()

# ----- Save pipeline -----
try:
    model_path = save_model(MODEL_NAME, best_est, is_dnn=False)
except Exception:
    os.makedirs("imput", exist_ok=True)
    model_path = os.path.join("imput", f"{MODEL_NAME.lower().re
    joblib.dump(best_est, model_path); print(f"[Fallback] Saved
# ป้องกัน index error ถ้า detailed_metrics ยังไม่ append
if len(detailed_metrics) > 0:
    detailed_metrics[-1]["model_path"] = model_path

# ----- Save tables -----
df_all = pd.DataFrame(detailed_metrics) if len(detailed_metrics)>0
if not df_all.empty:
    df_logreg = df_all[df_all.get("model","") == MODEL_NAME].copy()

```

```

    if not df_logreg.empty:
        df_logreg.to_csv(os.path.join(model_dir, "detailed_metrics_

def _summarize_f1_for_model(results_dict, model_name):
    rows = []
    for split in ["train", "test"]:
        arr = np.array(results_dict.get(model_name, {}).get(split,
            if arr.size:
                rows.append({
                    "model": model_name, "split": split, "n_runs": len(
                    "mean_f1": arr.mean(), "std_f1": arr.std(),
                    "min_f1": arr.min(), "max_f1": arr.max(),
                })
    return pd.DataFrame(rows).sort_values(["model","split"])

summary_f1_logreg = _summarize_f1_for_model(results, MODEL_NAME)
print("\nF1 summary (Logistic only):")
print(summary_f1_logreg.round(4) if not summary_f1_logreg.empty else
if not summary_f1_logreg.empty:
    summary_f1_logreg.to_csv(os.path.join(model_dir, "summary_f1_lc
# =====

```

```

=====
[Logistic Regression] Run 1/5

Searching best parameters...
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s

```

[illegible]

Page 11 of 101

Page 12 of 101

[illegible]

Page 14 of 101

Page 15 of 101

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

The score on these train-test partitions for these parameters will be low. If these failures are not expected, you can try to debug them by setting

Below are more details about the failures:

90 fits failed with the following error:

Traceback (most recent call last):

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
estimator.fit(X_train, y_train, **fit_params)
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return fit_method(estimator, *args, **kwargs)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
Xt, yt = self._fit(X, y, routed_params, raw_params=params)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
X, y, fitted_transformer = fit_resample_one_cached(
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return self.func(*args, **kwargs)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
X_res, y_res = sampler.fit_resample(X, y, **params.get("fit_resa
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return super().fit_resample(X, y, **params)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return fit_method(estimator, *args, **kwargs)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
self.sampling_strategy_ = check_sampling_strategy(
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
_sampling_strategy_float(sampling_strategy, y, sampling_type).it
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
raise ValueError(
```

ValueError: The specified ratio required to remove samples from the

```
warnings.warn(some_fits_failed_message, FitFailedWarning)
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
```

```
0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8
0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8
0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan
0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8
```

```
warnings.warn(
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
```

```
0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0.
0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0.
0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan
0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0.
```

```
warnings.warn(
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
```

```
0.50668115 0.50688818 0.50600146 0.50753857 nan 0.50668115
0.50413818 0.50420117 0.50419629 nan 0.50838135 0.51090283
0.51005654 0.506070 nan 0.50560156 0.5061050 0.50567570
```

```

0.51095654 0.506979 nan 0.50560156 0.5061956 0.50567578
0.50729004 nan 0.50612842 0.50369238 0.50395264 0.50404102
nan 0.50946777 0.51156494 0.51144775 0.50727539 nan
0.50679199 0.50697168 0.50605615 0.50758057 nan 0.50676514
0.50416797 0.50422217 0.50429883 nan 0.50930957 0.51149561
0.51136523 0.50719482 nan 0.50650635 0.50688916 0.50600391
0.5075249 nan 0.50671094 0.50408301 0.50420459 0.50425293
nan 0.50947461 0.51156543 0.51143164 0.50726172 nan
0.50669629 0.50697949 0.50607471 0.50759424 nan 0.50675781
0.50417578 0.50422754 0.50430225 nan 0.50943066 0.51156494
0.51141309 0.50724561 nan 0.50662842 0.50695557 0.50605029
0.5075542 nan 0.50671875 0.50412402 0.50421045 0.50425684]
warnings.warn(

```

Best parameters: {'clf__C': 0.1, 'clf__class_weight': None, 'clf__ma

Scores for best parameters (CV mean):

ACC : 0.8000

PREC: 0.0000

REC : 0.0000

F1 : 0.0000

AUC : 0.5116

Best threshold by Youden J (train): 0.2597

=== Final-Run Metrics: Logistic Regression [train] ===

Accuracy: 0.5337

Error rate (1-Acc): 0.4663

Classification Report:

	precision	recall	f1-score	support
0	0.8192	0.5353	0.6475	6400
1	0.2211	0.5275	0.3116	1600
accuracy			0.5337	8000
macro avg	0.5201	0.5314	0.4795	8000
weighted avg	0.6996	0.5337	0.5803	8000

Confusion Matrix [rows=true, cols=pred]:

```
[[3426 2974]
```

```
[ 756  844]]
```

Sensitivity (Recall+): 0.5275 | Specificity: 0.5353

Precision (PPV): 0.2211 | NPV: 0.8192

FPR: 0.4647 | FNR: 0.4725 | F1: 0.3116

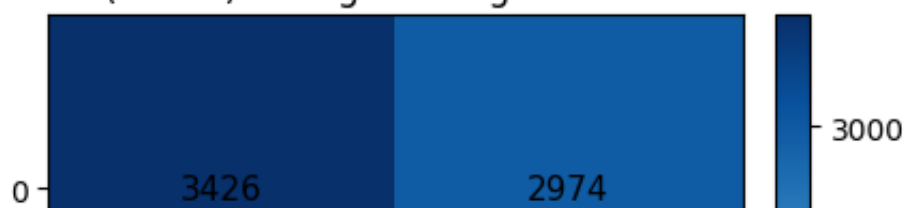
MCC: 0.0503 | Cohen's kappa: 0.0413 | Balanced Acc: 0.5314

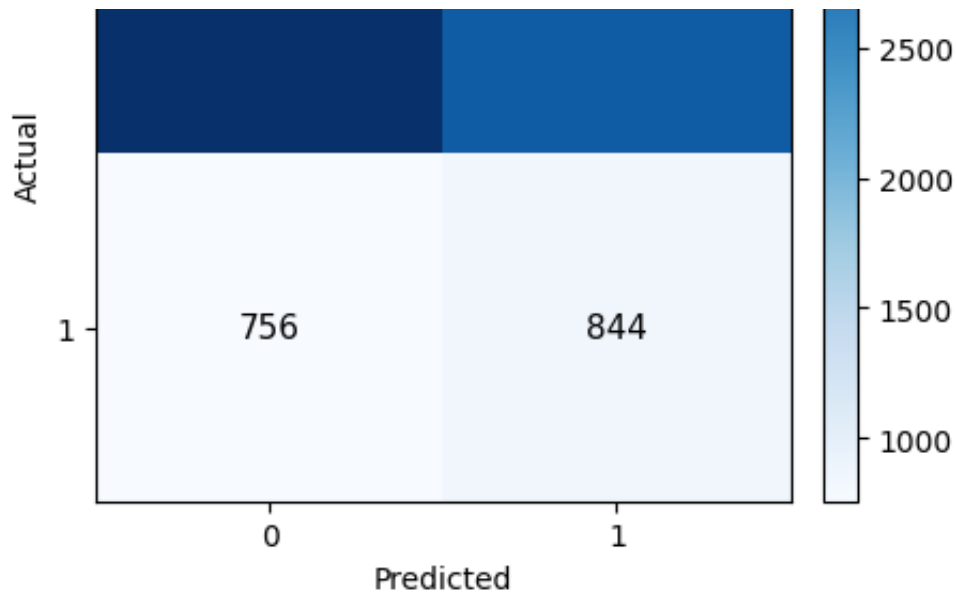
Hamming loss: 0.4662 | Zero-One loss: 0.4663

Prevalence+: 0.2000 | Pred+ rate: 0.4772 | Youden J: 0.0628

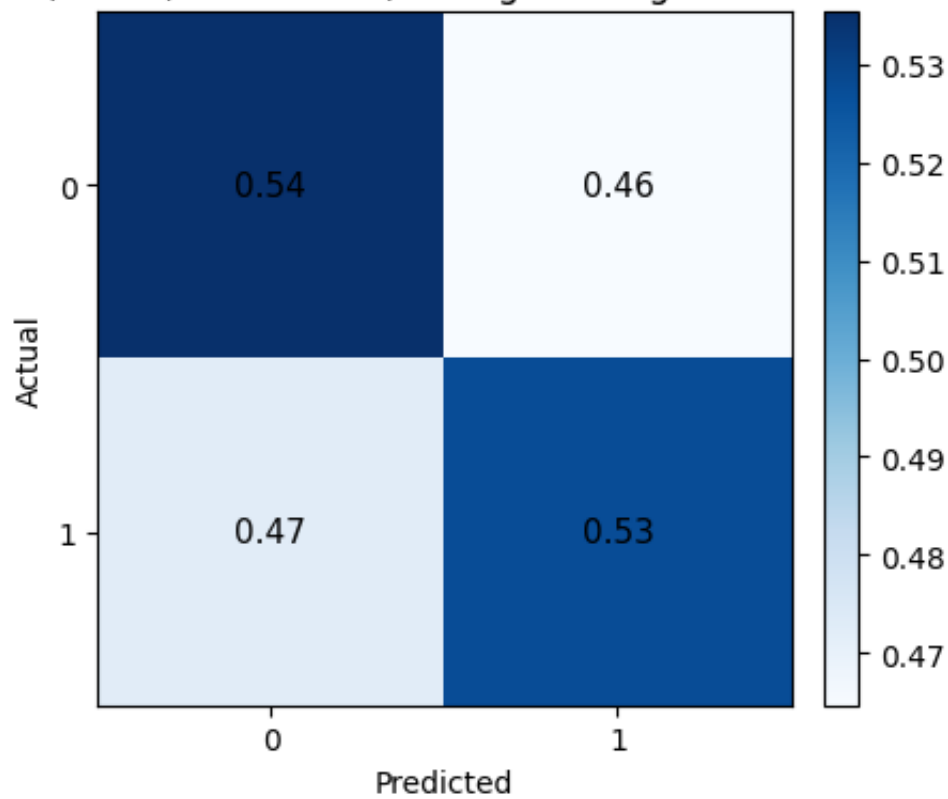
Log Loss: 0.5088 | Brier score: 0.1631 | Average Precision (PR-AUC):

CM (TRAIN) — Logistic Regression Run 1





CM (TRAIN, normalized) — Logistic Regression Run 1



=== Final-Run Metrics: Logistic Regression [test] ===

Accuracy: 0.5110

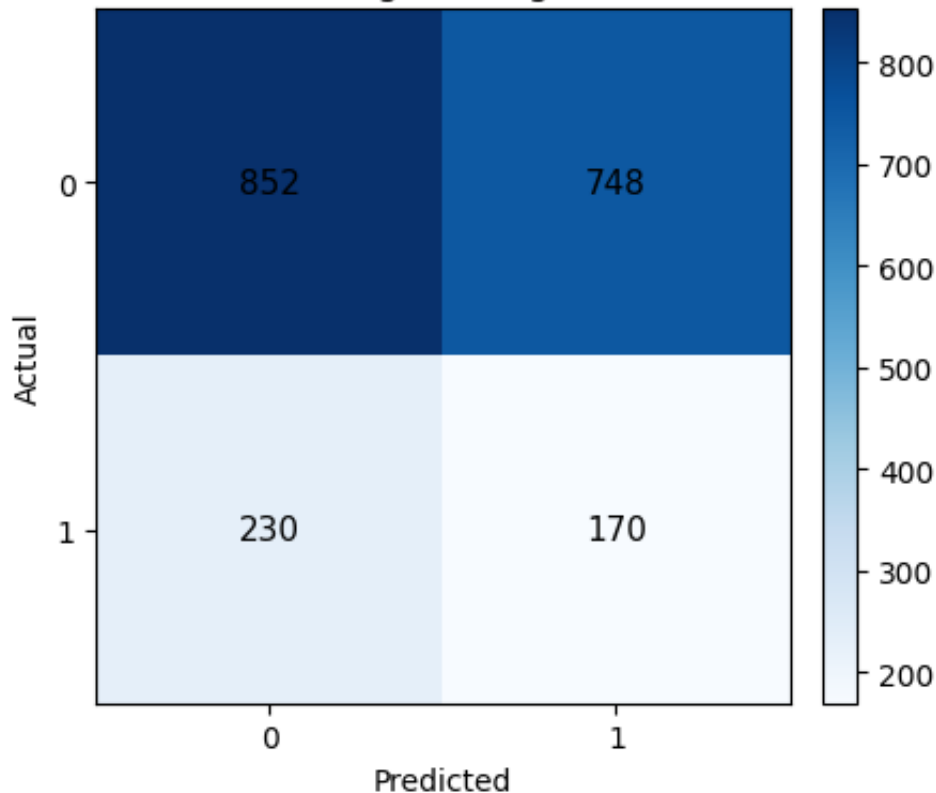
Error rate (1-Acc): 0.4890

Classification Report:

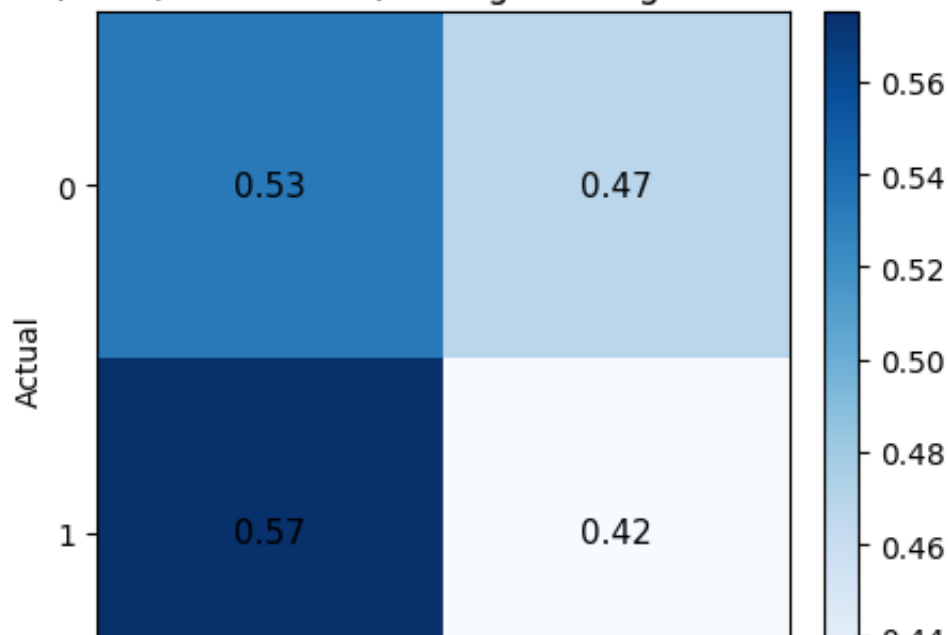
	precision	recall	f1-score	support
0	0.7874	0.5325	0.6353	1600
1	0.1852	0.4250	0.2580	400
accuracy			0.5110	2000
macro avg	0.4863	0.4788	0.4467	2000
weighted avg	0.6670	0.5110	0.5599	2000

```
Confusion Matrix [rows=true, cols=pred]:  
[[852 748]  
 [230 170]]  
Sensitivity (Recall+): 0.4250 | Specificity: 0.5325  
Precision (PPV): 0.1852 | NPV: 0.7874  
FPR: 0.4675 | FNR: 0.5750 | F1: 0.2580  
MCC: -0.0341 | Cohen's kappa: -0.0286 | Balanced Acc: 0.4788  
Hamming loss: 0.4890 | Zero-One loss: 0.4890  
Prevalence+: 0.2000 | Pred+ rate: 0.4590 | Youden J: -0.0425  
Log Loss: 0.5144 | Brier score: 0.1653 | Average Precision (PR-AUC):
```

CM (TEST) — Logistic Regression Run 1



CM (TEST, normalized) — Logistic Regression Run 1



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
```

[illegible]

```
warn printf(average modifier f"%metric capitalize/%11s" result s
```

warn printf(average, modifier, f"Metric {capitalize(metric)} is", result)

var prof/average modifier f"(matrix capitalizing(A) is" require a

Page 40 of 40

Below are more details about the failures:

Traceback (most recent call last):

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    estimator.fit(X_train, y_train, **fit_params)
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    return fit_method(estimator, *args, **kwargs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    Xt, yt = self._fit(X, y, routed_params, raw_params=params)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    X, y, fitted_transformer = fit_resample_one_cached(
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    return self.func(*args, **kwargs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    X_res, y_res = sampler.fit_resample(X, y, **params.get("fit_resa
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    return super().fit_resample(X, y, **params)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    return fit_method(estimator, *args, **kwargs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    self.sampling_strategy_ = check_sampling_strategy(
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    _sampling_strategy_float(sampling_strategy, y, sampling_type).it
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
    raise ValueError(
```

```
ValueError: The specified ratio required to remove samples from the

warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8
0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8
0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan
0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8
warnings.warn(
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0.
0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0.
0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan
0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0.
warnings.warn(
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0.50650928 0.5088291 0.5056416 0.5059043 nan 0.49825244
```

```

0.4993335 0.4970127 0.49924951 nan 0.49980615 0.4989292
0.50047656 0.50034375 nan 0.50527881 0.50806152 0.50487012
0.50596338 nan 0.49711914 0.49822998 0.49603027 0.49877539
nan 0.50054297 0.49988281 0.50119531 0.50078857 nan
0.50653809 0.50885498 0.50570996 0.50589844 nan 0.49840723
0.49945117 0.49708008 0.49933447 nan 0.50051221 0.49982666
0.50117773 0.50067236 nan 0.50638965 0.50874854 0.5056626
0.50583447 nan 0.498146 0.49925781 0.49687939 0.4992251
nan 0.50053955 0.49991162 0.5011709 0.50078809 nan
0.50651416 0.50885938 0.50571094 0.50590967 nan 0.49841406
0.49944873 0.49709961 0.49932227 nan 0.50050879 0.49984863
0.50119482 0.50074219 nan 0.50653076 0.50883594 0.50570459
0.50588916 nan 0.49831396 0.49936768 0.49704053 0.49927637]
warnings.warn(

```

Best parameters: {'clf__C': 10, 'clf__class_weight': None, 'clf__max

Scores for best parameters (CV mean):

ACC : 0.8000

PREC: 0.0000

REC : 0.0000

F1 : 0.0000

AUC : 0.5089

Best threshold by Youden J (train): 0.2717

=== Final-Run Metrics: Logistic Regression [train] ===

Accuracy: 0.6358

Error rate (1-Acc): 0.3642

Classification Report:

	precision	recall	f1-score	support
0	0.8125	0.7081	0.7567	6400
1	0.2287	0.3463	0.2755	1600
accuracy			0.6358	8000
macro avg	0.5206	0.5272	0.5161	8000
weighted avg	0.6957	0.6358	0.6605	8000

Confusion Matrix [rows=true, cols=pred]:

```

[[4532 1868]
 [1046  554]]

```

Sensitivity (Recall+): 0.3462 | Specificity: 0.7081

Precision (PPV): 0.2287 | NPV: 0.8125

FPR: 0.2919 | FNR: 0.6537 | F1: 0.2755

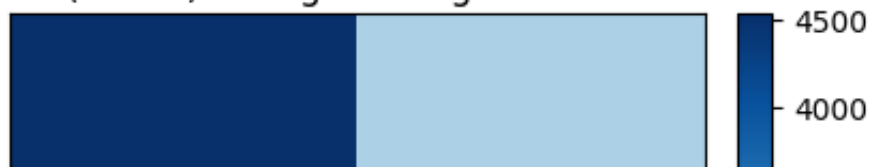
MCC: 0.0473 | Cohen's kappa: 0.0456 | Balanced Acc: 0.5272

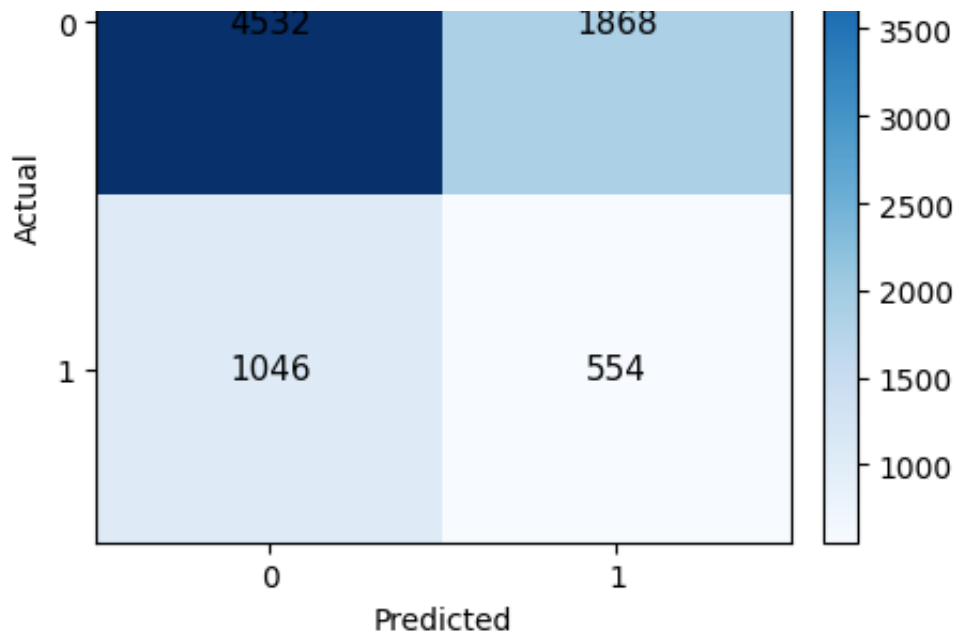
Hamming loss: 0.3643 | Zero-One loss: 0.3642

Prevalence+: 0.2000 | Pred+ rate: 0.3027 | Youden J: 0.0544

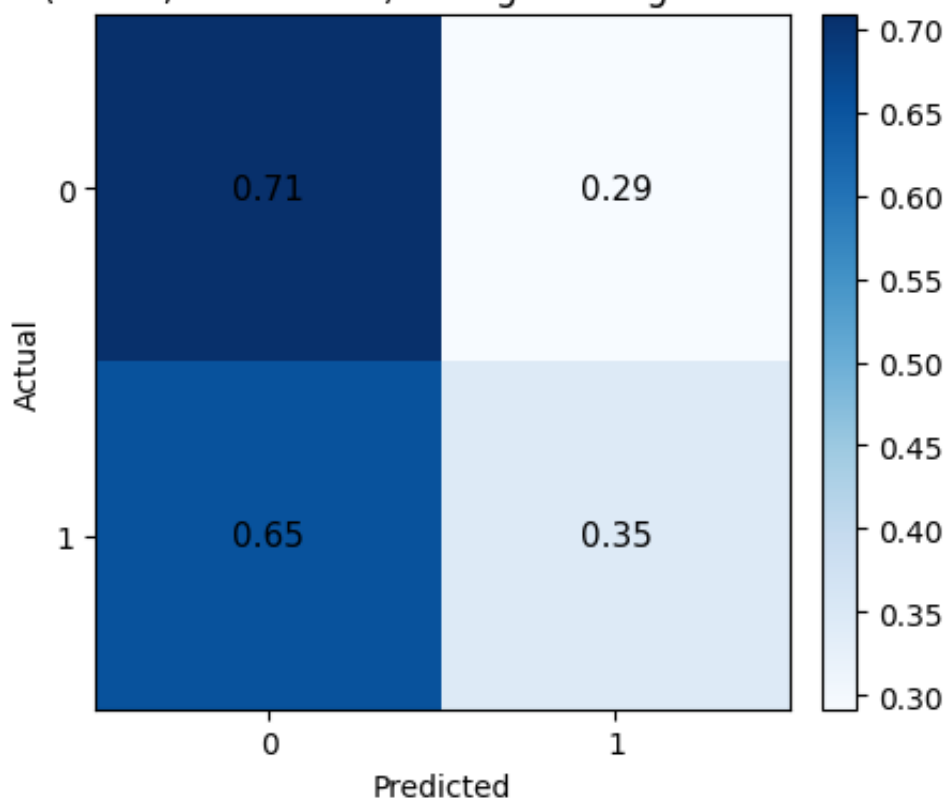
Log Loss: 0.5088 | Brier score: 0.1631 | Average Precision (PR-AUC):

CM (TRAIN) — Logistic Regression Run 2





CM (TRAIN, normalized) — Logistic Regression Run 2



=== Final-Run Metrics: Logistic Regression [test] ===

Accuracy: 0.6165

Error rate (1-Acc): 0.3835

Classification Report:

	precision	recall	f1-score	support
0	0.7927	0.7050	0.7463	1600
1	0.1820	0.2625	0.2149	400
accuracy			0.6165	2000
macro avg	0.4873	0.4838	0.4806	2000

weighted avg 0.6705 0.6165 0.6400 2000

Confusion Matrix [rows=true, cols=pred]:

```
[[1128  472]
 [ 295  105]]
```

Sensitivity (Recall+): 0.2625 | Specificity: 0.7050

Precision (PPV): 0.1820 | NPV: 0.7927

FPR: 0.2950 | FNR: 0.7375 | F1: 0.2149

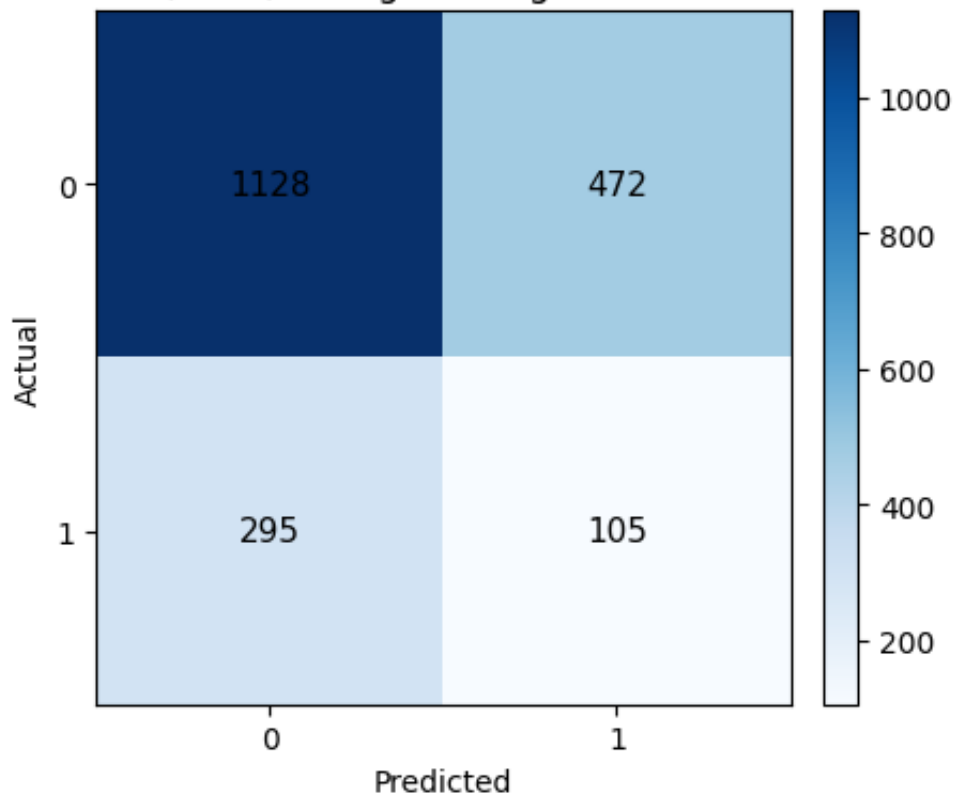
MCC: -0.0287 | Cohen's kappa: -0.0279 | Balanced Acc: 0.4838

Hamming loss: 0.3835 | Zero-One loss: 0.3835

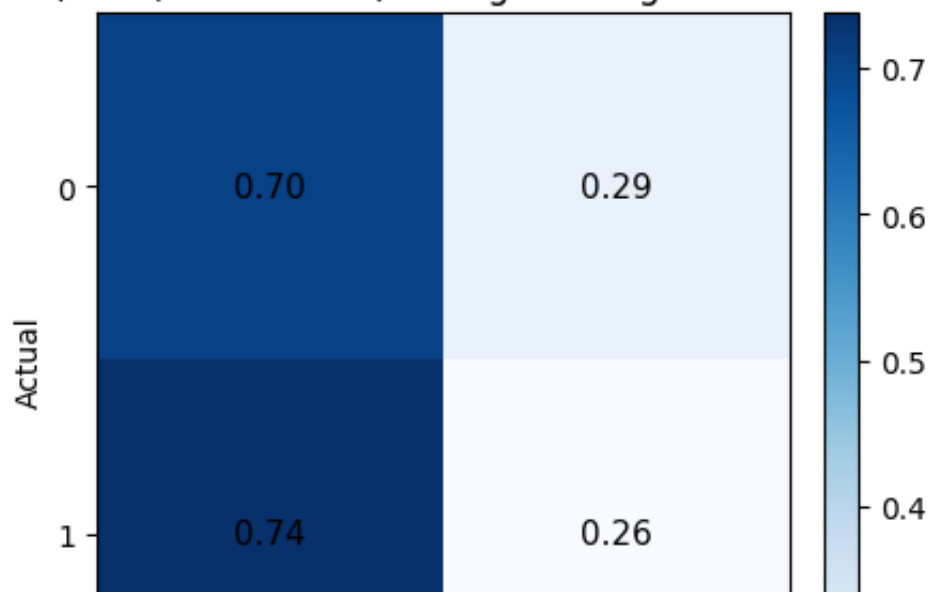
Prevalence+: 0.2000 | Pred+ rate: 0.2885 | Youden J: -0.0325

Log Loss: 0.5140 | Brier score: 0.1651 | Average Precision (PR-AUC):

CM (TEST) — Logistic Regression Run 2



CM (TEST, normalized) — Logistic Regression Run 2





Searching best parameters...

[illegible]

Page 46 of 101

Page 47 of 101

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

—	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	4
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	---

The score on these train-test partitions for these parameters will b
If these failures are not expected, you can try to debug them by set

Below are more details about the failures:

90 fits failed with the following error:

Traceback (most recent call last):

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
estimator.fit(X_train, y_train, **fit_params)
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return fit_method(estimator, *args, **kwargs)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
Xt, yt = self._fit(X, y, routed_params, raw_params=params)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
X, y, fitted_transformer = fit_resample_one_cached(
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return self.func(*args, **kwargs)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
X_res, y_res = sampler.fit_resample(X, y, **params.get("fit_resa
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return super().fit_resample(X, y, **params)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
return fit_method(estimator, *args, **kwargs)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
self.sampling_strategy_ = check_sampling_strategy(
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
_sampling_strategy_float(sampling_strategy, y, sampling_type).it
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
File "/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE
raise ValueError(
```

ValueError: The specified ratio required to remove samples from the

```
warnings.warn(some_fits_failed_message, FitFailedWarning)
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8
0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8
0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan
0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8 0.8 nan 0.8 0.8 0.8
```

```
warnings.warn(
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0.
0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0.
0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan
0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0.
```

```
warnings.warn(
```

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
```

```

0.50600293 0.50433203 0.5055752 0.50811572 nan 0.50349365
0.50456738 0.50737012 0.50615967 nan 0.50391553 0.5032998
0.50186523 0.50796191 nan 0.50531055 0.50388867 0.50525
0.50781201 nan 0.5031543 0.50420996 0.50725146 0.50595801
nan 0.50457715 0.50338721 0.50179395 0.50793555 nan
0.50600195 0.50437061 0.50568457 0.5081543 nan 0.50352832
0.5046377 0.50744678 0.5062915 nan 0.50454687 0.50333691
0.50177002 0.50792822 nan 0.50590479 0.5043374 0.50563232
0.50810449 nan 0.50351123 0.50459668 0.50741016 0.50629687
nan 0.50459033 0.503396 0.50180225 0.50792627 nan
0.50601025 0.50436035 0.5056875 0.50815869 nan 0.50354541
0.5046377 0.50745361 0.50629639 nan 0.50455518 0.50334277
0.50178613 0.50790381 nan 0.50599902 0.50431641 0.50564795
0.50815137 nan 0.50355371 0.50460938 0.50744922 0.50628467]
warnings.warn(

```

Best parameters: {'clf__C': 10, 'clf__class_weight': None, 'clf__max

Scores for best parameters (CV mean):

ACC : 0.8000

PREC: 0.0000

REC : 0.0000

F1 : 0.0000

AUC : 0.5082

Best threshold by Youden J (train): 0.3500

=== Final-Run Metrics: Logistic Regression [train] ===

Accuracy: 0.6330

Error rate (1-Acc): 0.3670

Classification Report:

	precision	recall	f1-score	support
0	0.8130	0.7030	0.7540	6400
1	0.2291	0.3531	0.2779	1600
accuracy			0.6330	8000
macro avg	0.5210	0.5280	0.5159	8000
weighted avg	0.6962	0.6330	0.6588	8000

Confusion Matrix [rows=true, cols=pred]:

[[4499 1901]

[1035 565]]

Sensitivity (Recall+): 0.3531 | Specificity: 0.7030

Precision (PPV): 0.2291 | NPV: 0.8130

FPR: 0.2970 | FNR: 0.6469 | F1: 0.2779

MCC: 0.0486 | Cohen's kappa: 0.0466 | Balanced Acc: 0.5280

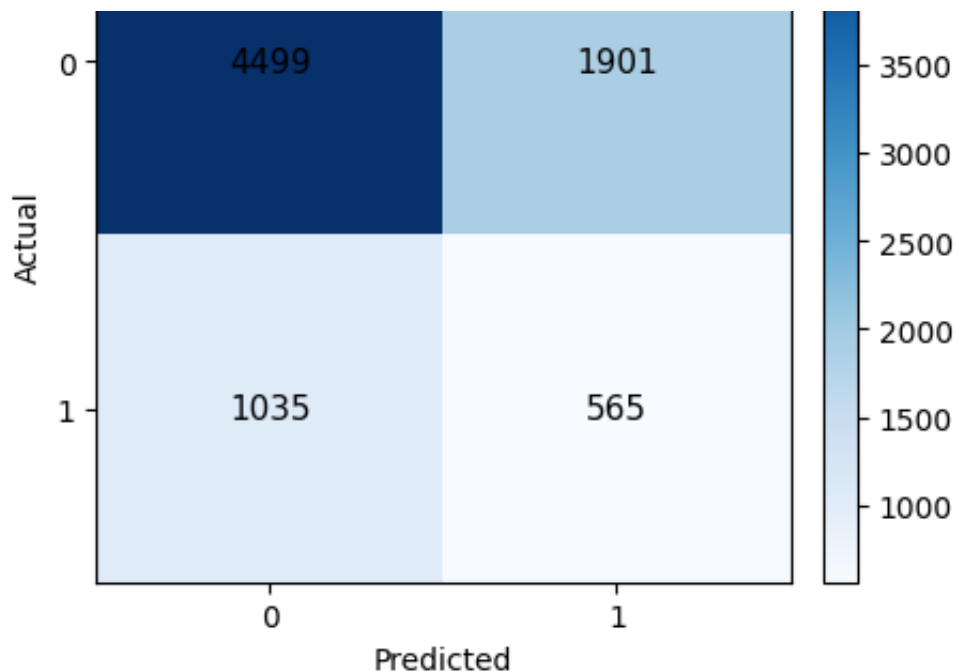
Hamming loss: 0.3670 | Zero-One loss: 0.3670

Prevalence+: 0.2000 | Pred+ rate: 0.3082 | Youden J: 0.0561

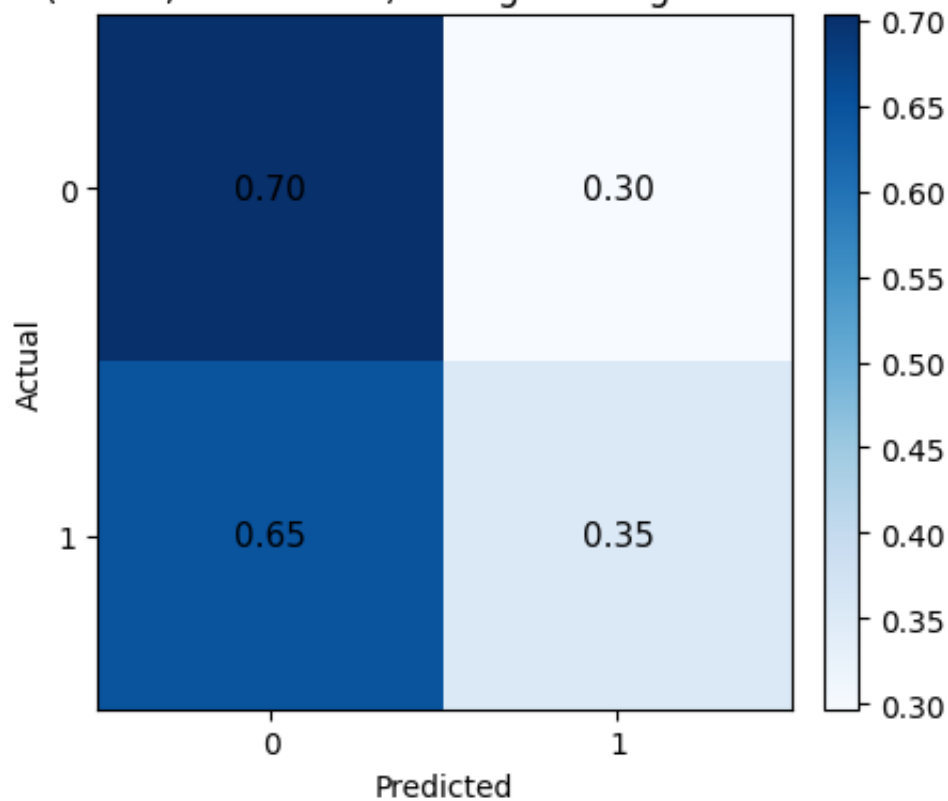
Log Loss: 0.5427 | Brier score: 0.1774 | Average Precision (PR-AUC):

CM (TRAIN) — Logistic Regression Run 3





CM (TRAIN, normalized) — Logistic Regression Run 3



=== Final-Run Metrics: Logistic Regression [test] ===

Accuracy: 0.6055

Error rate (1-Acc): 0.3945

Classification Report:

	precision	recall	f1-score	support
0	0.7874	0.6944	0.7380	1600
1	0.1698	0.2500	0.2022	400
accuracy			0.6055	2000

macro avg	0.4786	0.4722	0.4701	2000
weighted avg	0.6639	0.6055	0.6308	2000

Confusion Matrix [rows=true, cols=pred]:

```
[[1111  489]
 [ 300  100]]
```

Sensitivity (Recall+): 0.2500 | Specificity: 0.6944

Precision (PPV): 0.1698 | NPV: 0.7874

FPR: 0.3056 | FNR: 0.7500 | F1: 0.2022

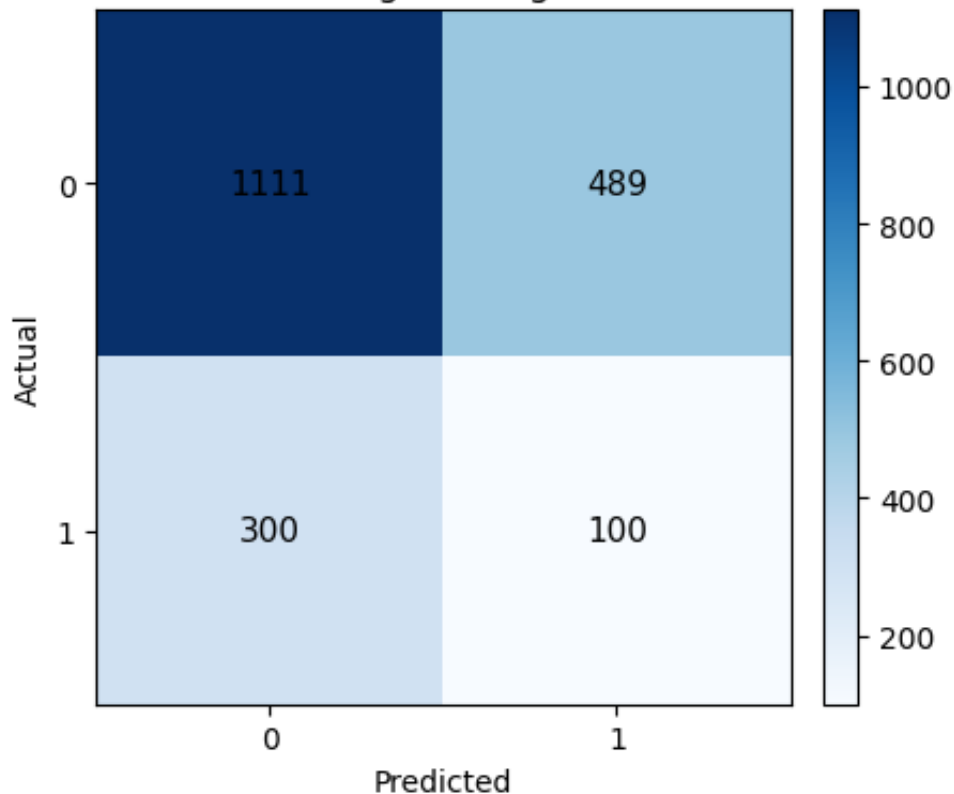
MCC: -0.0488 | Cohen's kappa: -0.0473 | Balanced Acc: 0.4722

Hamming loss: 0.3945 | Zero-One loss: 0.3945

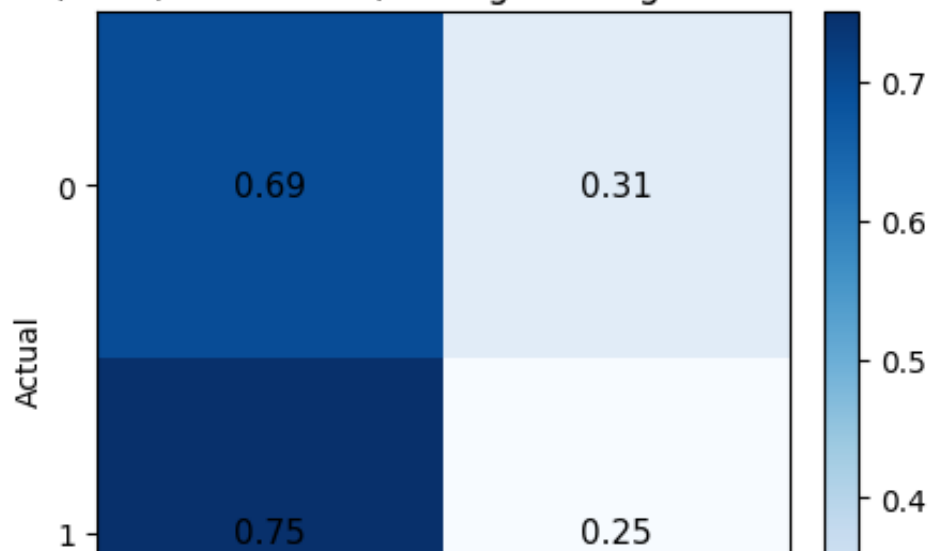
Prevalence+: 0.2000 | Pred+ rate: 0.2945 | Youden J: -0.0556

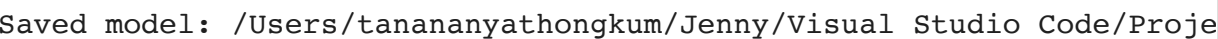
Log Loss: 0.5475 | Brier score: 0.1795 | Average Precision (PR-AUC):

CM (TEST) — Logistic Regression Run 3



CM (TEST, normalized) — Logistic Regression Run 3





Searching best parameters...

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
warn prf(average, modifier, f"{metric.capitalize()} is", result.s
```

[illegible]

[illegible]

[illegible]

[illegible]

```
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
```

[illegible]

```

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.s

```

warn printf(average, modifier, f"Metric capitalized/\\ is" result, c)

var prof/average modifier f"(matrix capitalizing(A) is" require a

[illegible]

Page 75 of 101

Page 76 of 101

Below are more details about the failures:

Page 77 of 101

```

/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/1
0.5051665 0.50560742 0.50568018 0.50416211 nan 0.50816064
0.50536182 0.50616309 0.50574951 nan 0.51153564 0.51012793
0.50792871 0.50870605 nan 0.50563721 0.50556592 0.50558789
0.50421533 nan 0.5083418 0.50501074 0.50592383 0.50586035
nan 0.51142432 0.50981494 0.50730273 0.50846777 nan
0.50517139 0.50560254 0.5057417 0.50418506 nan 0.50820508
0.50532568 0.50624463 0.50573096 nan 0.51151367 0.50977393
0.50728174 0.50845801 nan 0.50525488 0.50565039 0.50566699
0.5040874 nan 0.50821143 0.50537988 0.50611475 0.50575439
nan 0.51141992 0.50982178 0.50732324 0.50848291 nan
0.5051958 0.50562402 0.50572656 0.50420752 nan 0.5082207
0.50534082 0.50623438 0.50573584 nan 0.51144629 0.50978906
0.50728857 0.50844434 nan 0.50521191 0.50561133 0.50565771
0.50409619 nan 0.50818896 0.50536523 0.50612061 0.50569482]
warnings.warn(

```

Best parameters: {'clf__C': 0.1, 'clf__class_weight': None, 'clf__ma

Scores for best parameters (CV mean):

ACC : 0.8000

PREC: 0.0000

REC : 0.0000

F1 : 0.0000

AUC : 0.5115

Best threshold by Youden J (train): 0.2382

=== Final-Run Metrics: Logistic Regression [train] ===

Accuracy: 0.5845

Error rate (1-Acc): 0.4155

Classification Report:

	precision	recall	f1-score	support
0	0.8152	0.6216	0.7053	6400
1	0.2237	0.4363	0.2958	1600
accuracy			0.5845	8000
macro avg	0.5194	0.5289	0.5005	8000
weighted avg	0.6969	0.5845	0.6234	8000

Confusion Matrix [rows=true, cols=pred]:

[[3978 2422]

[902 698]]

Sensitivity (Recall+): 0.4362 | Specificity: 0.6216

Precision (PPV): 0.2237 | NPV: 0.8152

FPR: 0.3784 | FNR: 0.5637 | F1: 0.2958

MCC: 0.0474 | Cohen's kappa: 0.0426 | Balanced Acc: 0.5289

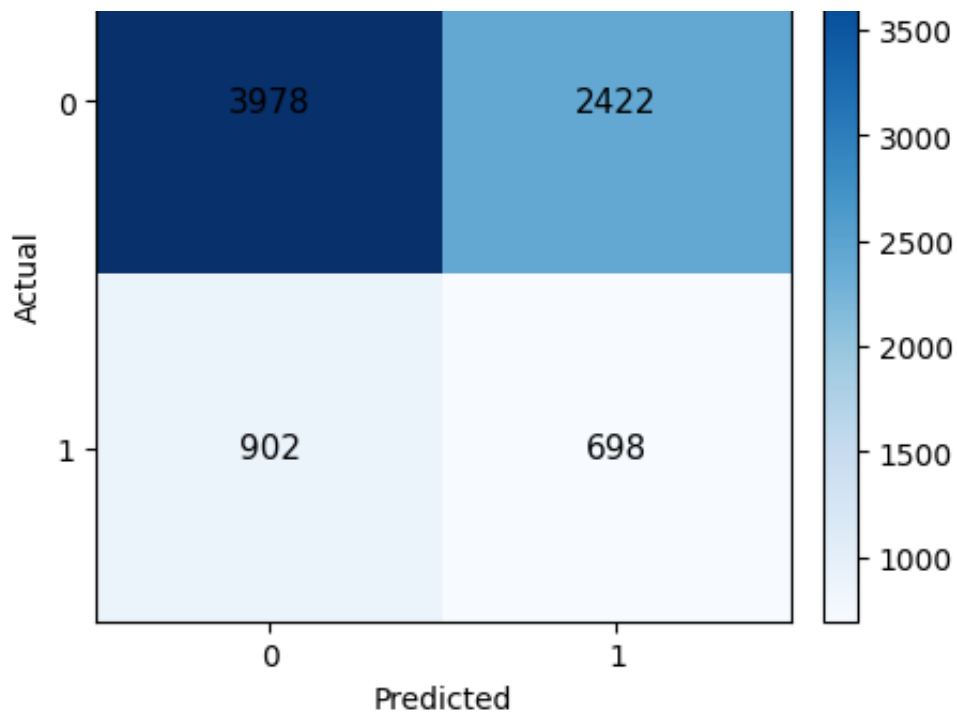
Hamming loss: 0.4155 | Zero-One loss: 0.4155

Prevalence+: 0.2000 | Pred+ rate: 0.3900 | Youden J: 0.0578

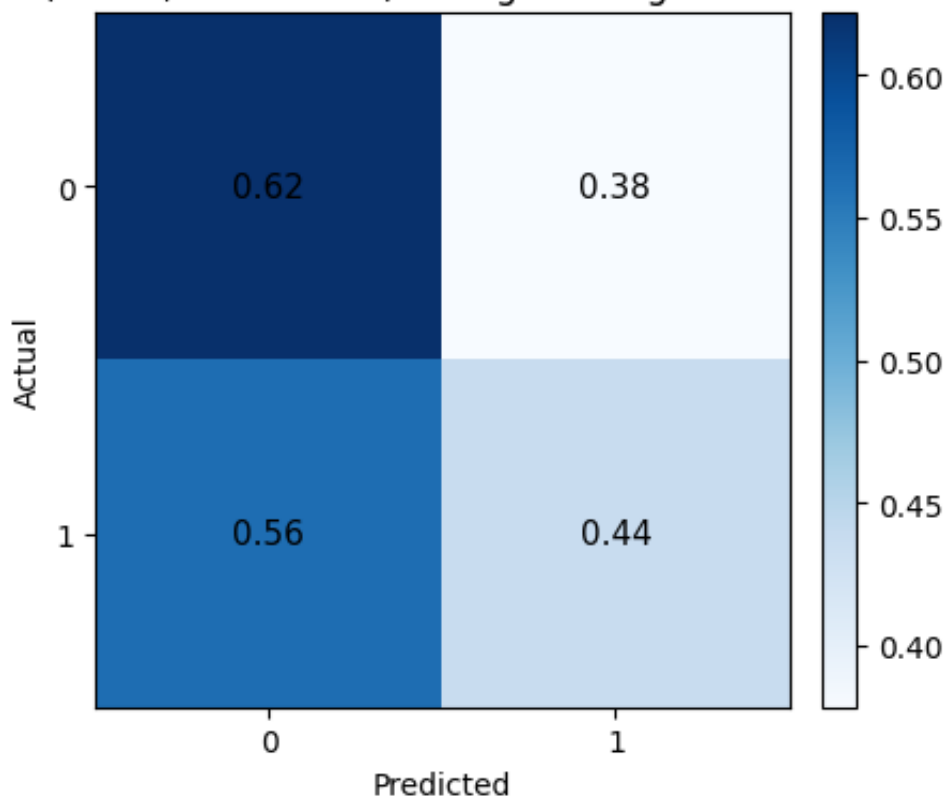
Log Loss: 0.5020 | Brier score: 0.1606 | Average Precision (PR-AUC):

CM (TRAIN) — Logistic Regression Run 4





CM (TRAIN, normalized) — Logistic Regression Run 4



=== Final-Run Metrics: Logistic Regression [test] ===

Accuracy: 0.5580

Error rate (1-Acc): 0.4420

Classification Report:

	precision	recall	f1-score	support
0	0.7896	0.6100	0.6883	1600
1	0.1832	0.3500	0.2405	400

accuracy			0.5580	2000
macro avg	0.4864	0.4800	0.4644	2000
weighted avg	0.6684	0.5580	0.5987	2000

Confusion Matrix [rows=true, cols=pred]:

```
[[976 624]
 [260 140]]
```

Sensitivity (Recall+): 0.3500 | Specificity: 0.6100

Precision (PPV): 0.1832 | NPV: 0.7896

FPR: 0.3900 | FNR: 0.6500 | F1: 0.2405

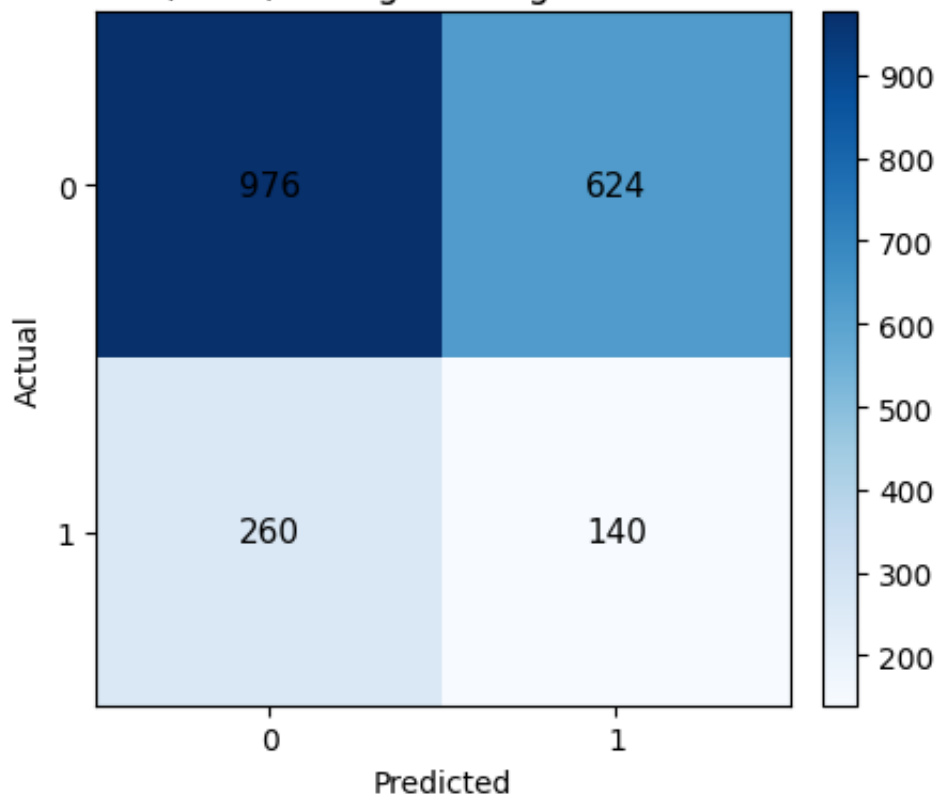
MCC: -0.0329 | Cohen's kappa: -0.0298 | Balanced Acc: 0.4800

Hamming loss: 0.4420 | Zero-One loss: 0.4420

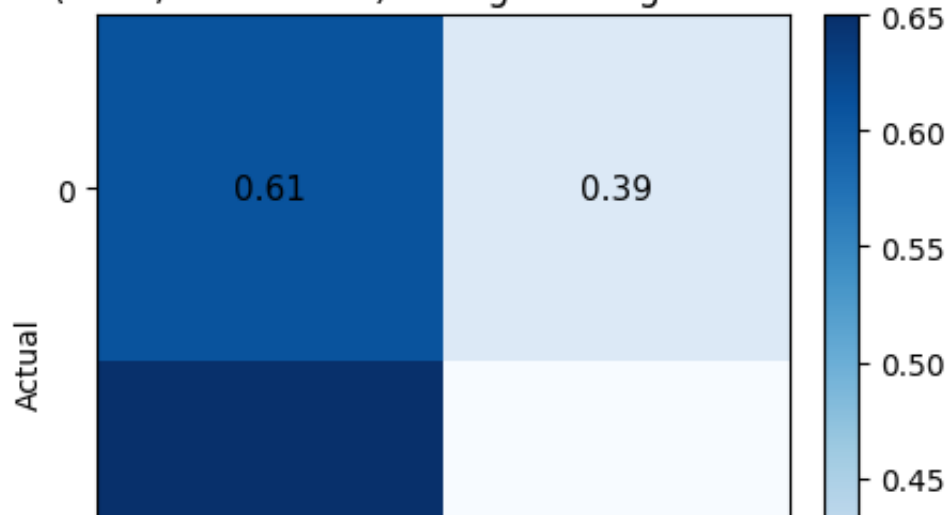
Prevalence+: 0.2000 | Pred+ rate: 0.3820 | Youden J: -0.0400

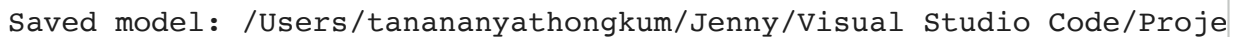
Log Loss: 0.5075 | Brier score: 0.1625 | Average Precision (PR-AUC):

CM (TEST) — Logistic Regression Run 4



CM (TEST, normalized) — Logistic Regression Run 4





```
Searching best parameters...
```

[illegible]

[illegible]

Page 83 of 101

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

I_{max}	I_{min}	ΔI	δI	I_{max}	I_{min}	ΔI	δI	I_{max}	I_{min}	ΔI	δI	I_{max}	I_{min}	ΔI	δI
------------------	------------------	------------	------------	------------------	------------------	------------	------------	------------------	------------------	------------	------------	------------------	------------------	------------	------------

[illegible]

[illegible]

90 fits failed out of a total of 450.

Below are more details about the failures:

Traceback (most recent call last):

```
File "/Users/tanahanyathongkum/Jenny/Visual Studio Code/Project_CE
raise ValueError(
```

```
warnings.warn(some fits failed message, FitFailedWarning)
```

0.8	0.8		nan	0.8	0.8	0.8	0.8	
0.8	0.8	0.8		0.8		nan	0.8	0.8
0.8		nan	0.8	0.8	0.8	0.8		nan
0.8	0.8	0.8		nan	0.8	0.8	0.8	0.8
	nan	0.8	0.8	0.8	0.8		nan	0.8
0.8	0.8		nan	0.8	0.8	0.8	0.8	
0.8	0.8	0.8		0.8		nan	0.8	0.8
0.8		nan	0.8	0.8	0.8	0.8		nan
0.8	0.8		0.799875		nan	0.8	0.8	0.8
	nan	0.8	0.8	0.8	0.8		nan	0.8
0.8	0.8	1						

```
warnings.warn(
```

```

/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0.
0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0.
0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan
0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0. 0. nan 0. 0. 0.
warnings.warn(
/Users/tanananyathongkum/Jenny/Visual Studio Code/Project_CE/.venv/l
0.50909424 0.50797754 0.51052979 0.51356445 nan 0.51261572
0.51280713 0.51224365 0.5132915 nan 0.51137402 0.51120898
0.51080518 0.51117822 nan 0.50790332 0.50717041 0.5100542
0.51332617 nan 0.5117832 0.51201904 0.51185449 0.51292676
nan 0.51209814 0.51171045 0.51119287 0.51140869 nan
0.50912109 0.50795166 0.5105918 0.51349121 nan 0.51252734
0.51278564 0.51214844 0.5133125 nan 0.51197998 0.51158691
0.51111377 0.51131299 nan 0.50897363 0.5078291 0.51039746
0.51345166 nan 0.51248486 0.51272217 0.51207471 0.51327832
nan 0.51208252 0.51169189 0.51119434 0.51139746 nan
0.50911523 0.50795313 0.51060254 0.5134624 nan 0.51251953
0.51279004 0.51213965 0.51329834 nan 0.51201807 0.51161621
0.51111768 0.51134814 nan 0.50908105 0.50785303 0.51047852
0.51345166 nan 0.51247852 0.51270801 0.51208936 0.51328662]
warnings.warn(

```

Best parameters: {'clf__C': 0.1, 'clf__class_weight': None, 'clf__ma

Scores for best parameters (CV mean):

ACC : 0.8000

PREC: 0.0000

REC : 0.0000

F1 : 0.0000

AUC : 0.5136

Best threshold by Youden J (train): 0.3090

=== Final-Run Metrics: Logistic Regression [train] ===

Accuracy: 0.3684

Error rate (1-Acc): 0.6316

Classification Report:

	precision	recall	f1-score	support
0	0.8346	0.2625	0.3994	6400
1	0.2116	0.7919	0.3340	1600
accuracy			0.3684	8000
macro avg	0.5231	0.5272	0.3667	8000
weighted avg	0.7100	0.3684	0.3863	8000

Confusion Matrix [rows=true, cols=pred]:

```

[[1680 4720]
 [ 333 1267]]

```

Sensitivity (Recall+): 0.7919 | Specificity: 0.2625

Precision (PPV): 0.2116 | NPV: 0.8346

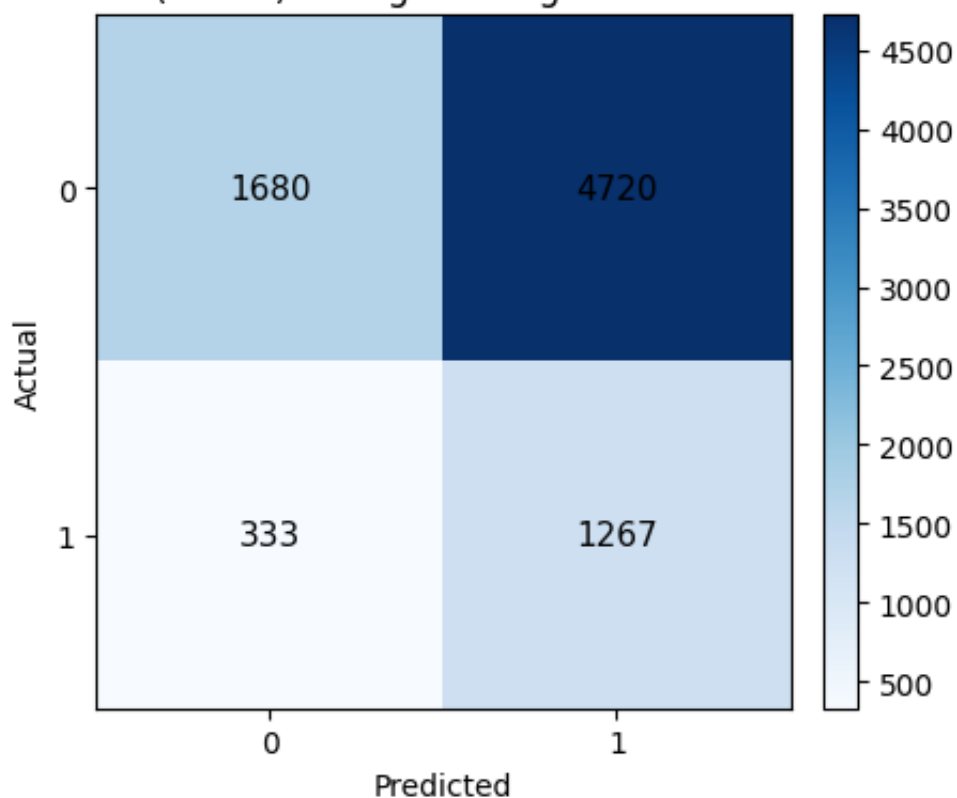
FPR: 0.7375 | FNR: 0.2081 | F1: 0.3340

MCC: 0.0501 | Cohen's kappa: 0.0268 | Balanced Acc: 0.5272

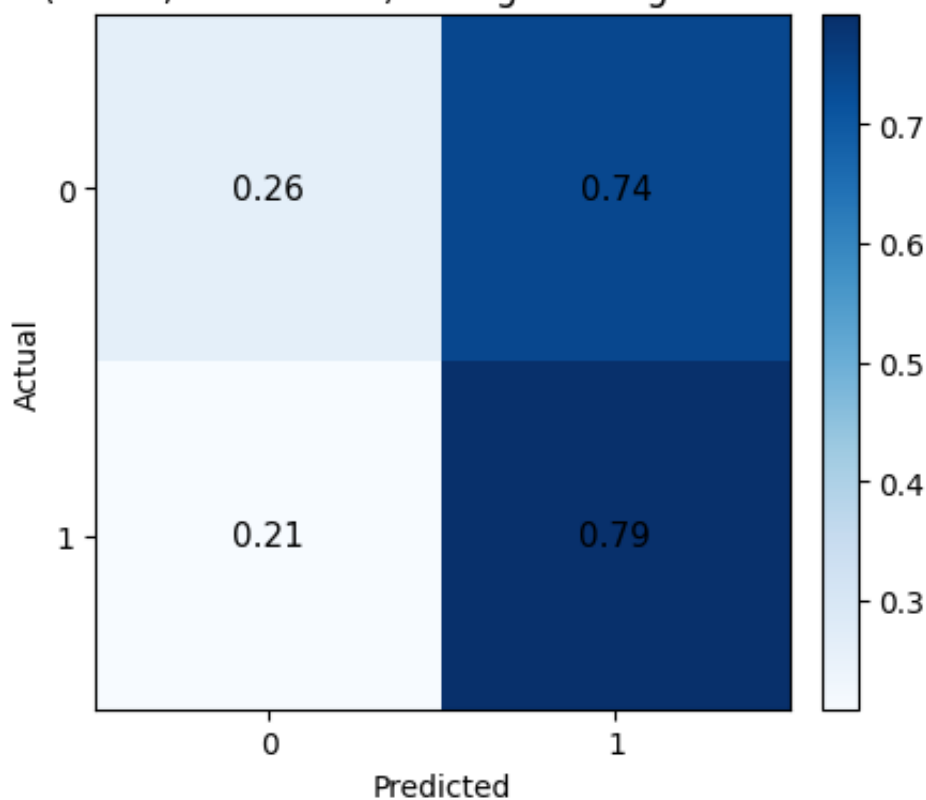
Hamming loss: 0.6316 | Zero-One loss: 0.6316

Prevalence+: 0.2000 | Pred+ rate: 0.7484 | Youden J: 0.0544
Log Loss: 0.5427 | Brier score: 0.1774 | Average Precision (PR-AUC):

CM (TRAIN) — Logistic Regression Run 5



CM (TRAIN, normalized) — Logistic Regression Run 5



=== Final-Run Metrics: Logistic Regression [test] ===
Accuracy: 0.3415
Error rate (1-Acc): 0.6585

Classification Report:

	precision	recall	f1-score	support
0	0.7695	0.2525	0.3802	1600
1	0.1892	0.6975	0.2976	400
accuracy			0.3415	2000
macro avg	0.4793	0.4750	0.3389	2000
weighted avg	0.6534	0.3415	0.3637	2000

Confusion Matrix [rows=true, cols=pred]:

```
[[ 404 1196]
 [ 121  279]]
```

Sensitivity (Recall+): 0.6975 | Specificity: 0.2525

Precision (PPV): 0.1892 | NPV: 0.7695

FPR: 0.7475 | FNR: 0.3025 | F1: 0.2976

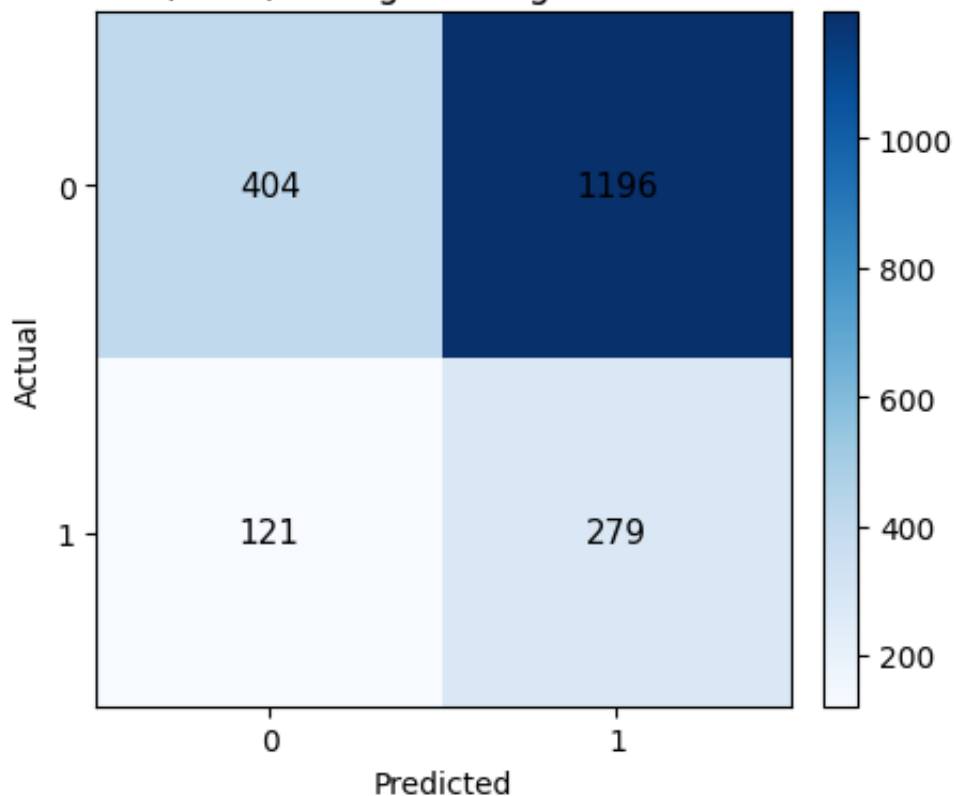
MCC: -0.0455 | Cohen's kappa: -0.0249 | Balanced Acc: 0.4750

Hamming loss: 0.6585 | Zero-One loss: 0.6585

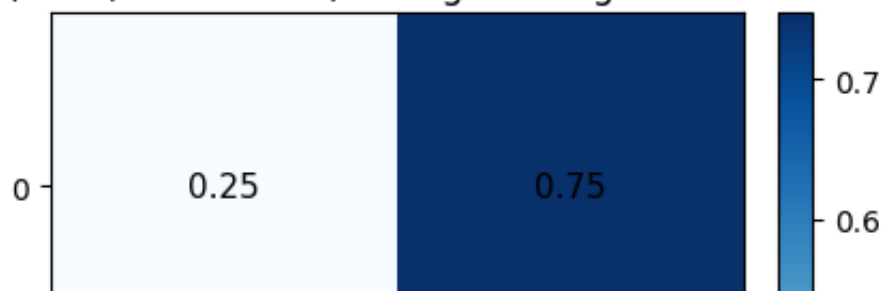
Prevalence+: 0.2000 | Pred+ rate: 0.7375 | Youden J: -0.0500

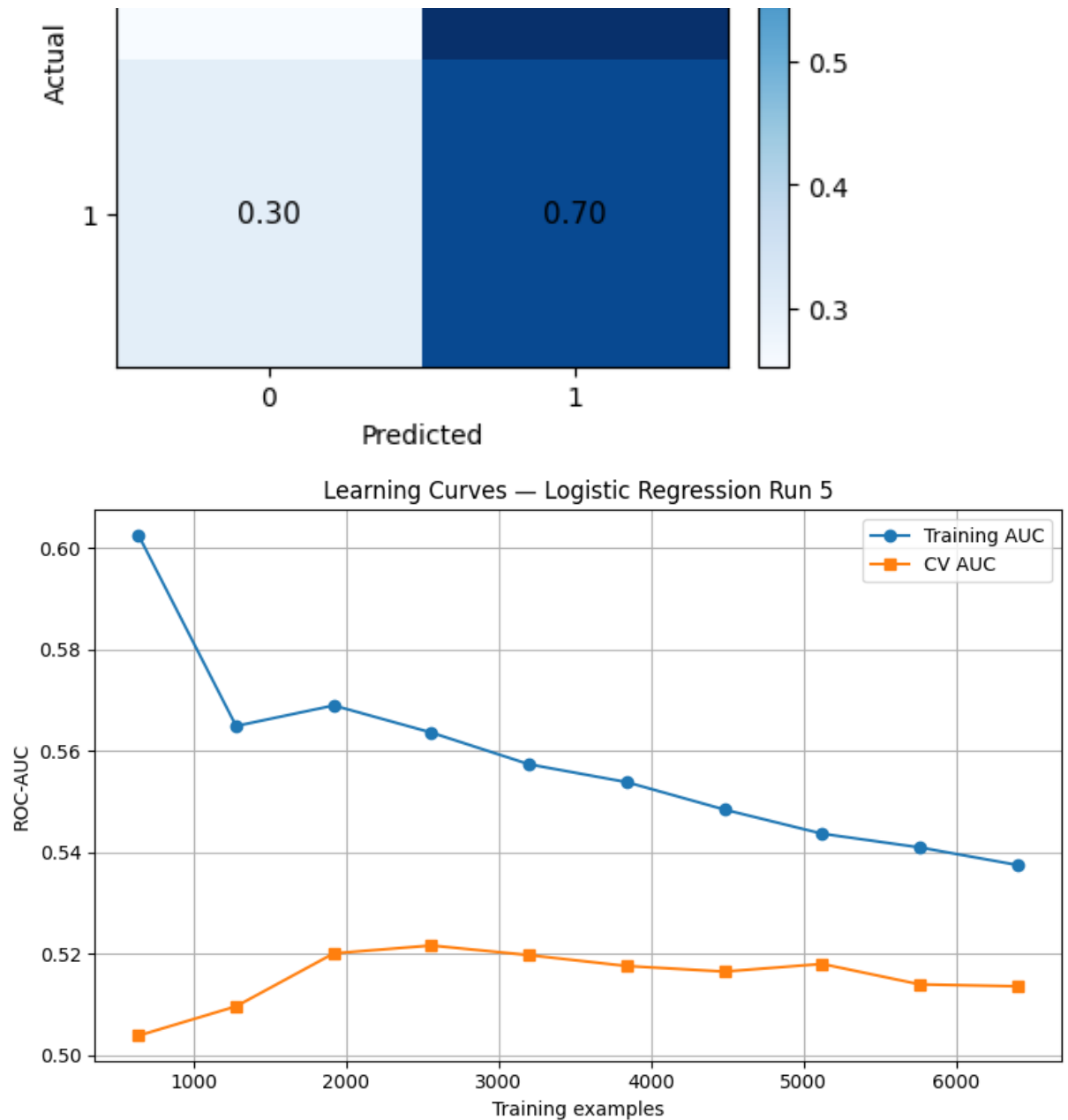
Log Loss: 0.5487 | Brier score: 0.1800 | Average Precision (PR-AUC):

CM (TEST) — Logistic Regression Run 5



CM (TEST, normalized) — Logistic Regression Run 5

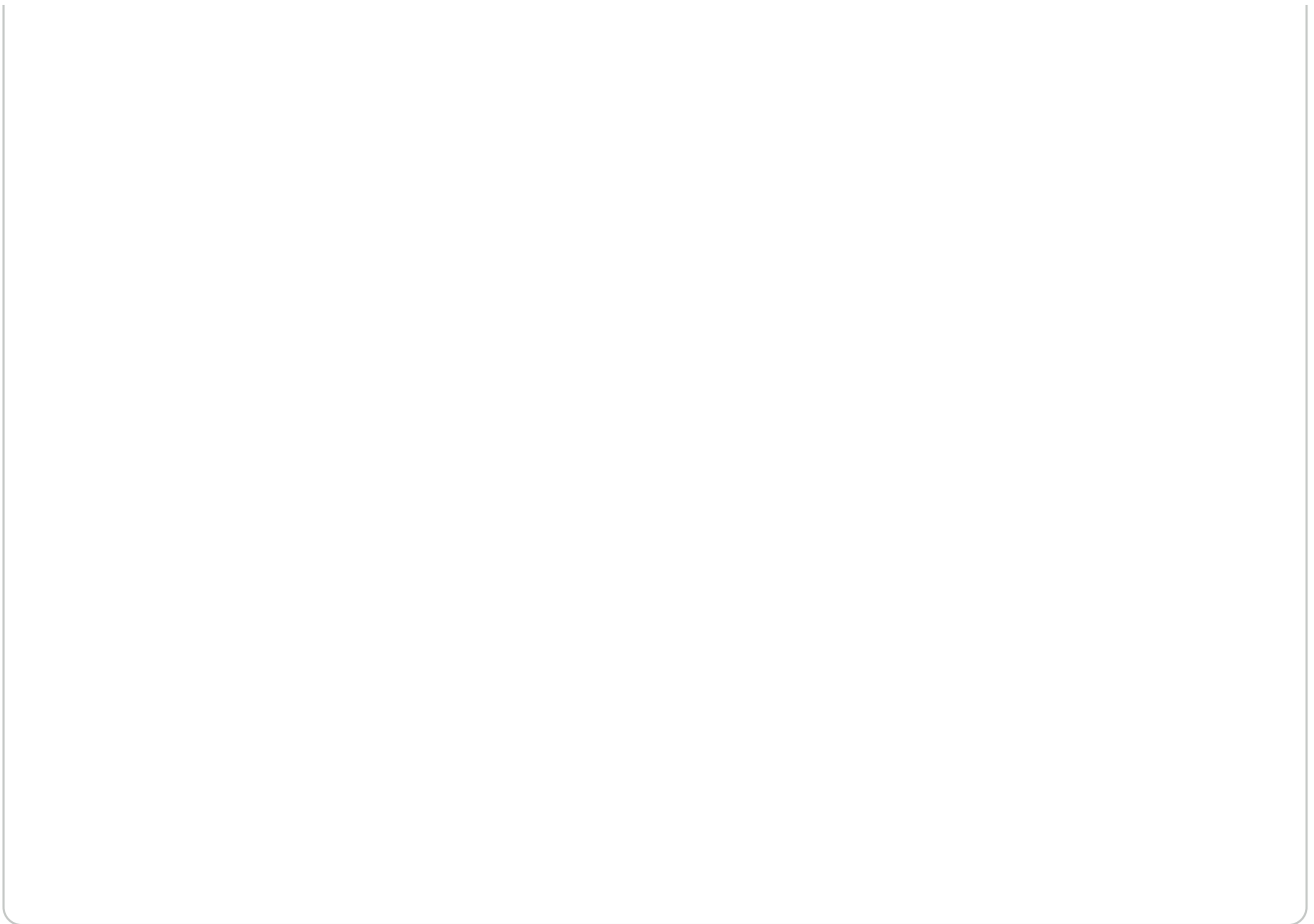




Saved model: /Users/tanananyathongkum/Jenny/Visual Studio Code/Proje

F1 summary (Logistic only):

	model	split	n_runs	mean_f1	std_f1	min_f1	max_
1	Logistic Regression	test	20	0.2747	0.0423	0.2022	0.33
0	Logistic Regression	train	20	0.3145	0.0195	0.2755	0.33



Start coding or generate with AI.

