**Task 2: Class Imbalance Handling and Custom Evaluation Metrics**

File 2: class\_imbalance\_task.py (Incomplete Version)

# Class Imbalance Handling and Custom Evaluation Metrics

# ------------------------------------------------------

# This task focuses on addressing class imbalance in the telco customer churn dataset

# and implementing custom metrics to better evaluate model performance from a business perspective.

# Import necessary libraries

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.neural\_network import MLPClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (classification\_report, confusion\_matrix, roc\_curve,

roc\_auc\_score, precision\_recall\_curve, auc)

# Create output directory for saving files

output\_dir = "output"

os.makedirs(output\_dir, exist\_ok=True)

# Load the dataset

print("Loading the dataset...")

dataset\_path = "TelcoCustomerChurn.csv"

# Read the CSV file

try:

telco\_data = pd.read\_csv(dataset\_path)

print(f"Dataset successfully loaded with shape: {telco\_data.shape}")

except Exception as e:

print(f"Error loading dataset: {e}")

raise

# Data Preprocessing

print("\nPreprocessing the data...")

# Convert 'TotalCharges' to numeric, replacing spaces with NaN

if telco\_data['TotalCharges'].dtype == 'object':

telco\_data['TotalCharges'] = pd.to\_numeric(telco\_data['TotalCharges'], errors='coerce')

telco\_data['TotalCharges'].fillna(telco\_data['TotalCharges'].mean(), inplace=True)

# Convert 'SeniorCitizen' from 0/1 to 'No'/'Yes' for consistent preprocessing

telco\_data['SeniorCitizen'] = telco\_data['SeniorCitizen'].map({0: 'No', 1: 'Yes'})

# Separate features and target variable

X = telco\_data.drop(['customerID', 'Churn'], axis=1)

y = telco\_data['Churn'].map({'Yes': 1, 'No': 0}) # Convert to binary

# Identify categorical and numerical columns

categorical\_cols = X.select\_dtypes(include=['object']).columns.tolist()

numerical\_cols = X.select\_dtypes(include=['int64', 'float64']).columns.tolist()

print(f"Categorical columns: {categorical\_cols}")

print(f"Numerical columns: {numerical\_cols}")

# Create preprocessors for both types of features

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), numerical\_cols),

('cat', OneHotEncoder(drop='first', sparse=False), categorical\_cols)

])

# TODO: IMPLEMENT CLASS IMBALANCE HANDLING

# --------------------------------------

# Your code should:

# 1. Check for class imbalance in the target variable

# 2. Implement at least one technique to address class imbalance (SMOTE, class weights, etc.)

# 3. Compare the distribution of classes before and after handling the imbalance

# TODO: IMPLEMENT CUSTOM EVALUATION FUNCTIONS

# ----------------------------------------

# Create functions to evaluate models from a business perspective

# Your code should define at least one custom metric relevant to churn prediction

# Example: Create a cost function that accounts for the business impact of

# false negatives vs. false positives

# TODO: FIND OPTIMAL CLASSIFICATION THRESHOLD

# ----------------------------------------

# Implement code to find the optimal classification threshold

# Your code should:

# 1. Calculate performance metrics at different thresholds

# 2. Identify the threshold that optimizes a specific business metric

# 3. Compare the results of the standard threshold (0.5) with the optimal threshold

# TODO: CREATE CUSTOM VISUALIZATIONS

# -------------------------------

# Create visualizations that help understand model performance from a business perspective

# Your code should create at least two plots that provide insights into model performance

# beyond standard accuracy and ROC curves

print("\nClass imbalance handling task completed!")

Sample Implementation Code for Task 2

# TODO: IMPLEMENT CLASS IMBALANCE HANDLING

# --------------------------------------

# Check for class imbalance

print("\nChecking for class imbalance...")

class\_counts = y.value\_counts()

print("Target class distribution:")

print(class\_counts)

print(f"Class imbalance ratio (majority:minority): {class\_counts[0]/class\_counts[1]:.2f}:1")

# Split the data into training and testing sets (before handling imbalance)

X\_train\_original, X\_test, y\_train\_original, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=y)

# Apply preprocessing to test data

X\_test\_processed = preprocessor.fit\_transform(X\_test)

# Method 1: Use SMOTE for oversampling the minority class

print("\nImplementing SMOTE oversampling...")

from imblearn.over\_sampling import SMOTE

# Apply preprocessing to training data

X\_train\_processed\_original = preprocessor.transform(X\_train\_original)

# Apply SMOTE to the processed training data

smote = SMOTE(random\_state=42)

X\_train\_processed\_smote, y\_train\_smote = smote.fit\_resample(

X\_train\_processed\_original, y\_train\_original)

print("Class distribution after SMOTE:")

print(pd.Series(y\_train\_smote).value\_counts())

# Method 2: Use class weights to handle imbalance

print("\nImplementing class weights...")

# Calculate class weights inversely proportional to class frequencies

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

class\_weights = compute\_class\_weight(

class\_weight='balanced', classes=np.unique(y\_train\_original), y=y\_train\_original)

class\_weight\_dict = {i: weight for i, weight in enumerate(class\_weights)}

print(f"Class weights: {class\_weight\_dict}")

# Train models with different imbalance handling techniques

# 1. Baseline model (no imbalance handling)

baseline\_model = RandomForestClassifier(random\_state=42)

baseline\_model.fit(X\_train\_processed\_original, y\_train\_original)

# 2. SMOTE model

smote\_model = RandomForestClassifier(random\_state=42)

smote\_model.fit(X\_train\_processed\_smote, y\_train\_smote)

# 3. Class weights model

weighted\_model = RandomForestClassifier(

random\_state=42, class\_weight=class\_weight\_dict)

weighted\_model.fit(X\_train\_processed\_original, y\_train\_original)

# TODO: IMPLEMENT CUSTOM EVALUATION FUNCTIONS

# ----------------------------------------

print("\nImplementing custom evaluation metrics...")

def calculate\_business\_metrics(y\_true, y\_pred, y\_prob=None, fn\_cost=5, fp\_cost=1):

"""

Calculate business-oriented metrics for model evaluation.

Parameters:

-----------

y\_true : array-like

True class labels

y\_pred : array-like

Predicted class labels

y\_prob : array-like, optional

Predicted probabilities for the positive class

fn\_cost : float, optional

Cost of a false negative (missing a churner)

fp\_cost : float, optional

Cost of a false positive (incorrectly predicting churn)

Returns:

--------

dict

Dictionary of business metrics

"""

# Calculate confusion matrix elements

tn, fp, fn, tp = confusion\_matrix(y\_true, y\_pred).ravel()

# Calculate standard metrics

accuracy = (tp + tn) / (tp + tn + fp + fn)

precision = tp / (tp + fp) if (tp + fp) > 0 else 0

recall = tp / (tp + fn) if (tp + fn) > 0 else 0

f1 = 2 \* precision \* recall / (precision + recall) if (precision + recall) > 0 else 0

# Calculate business costs

total\_cost = (fn \* fn\_cost) + (fp \* fp\_cost)

cost\_per\_customer = total\_cost / len(y\_true)

# Calculate customer retention metrics

retention\_rate = tn / (tn + fn) if (tn + fn) > 0 else 0

intervention\_efficiency = tp / (tp + fp) if (tp + fp) > 0 else 0

# Calculate profitability metrics (assuming average values)

avg\_customer\_value = 1000 # Hypothetical average customer lifetime value

avg\_intervention\_cost = 100 # Hypothetical cost of retention intervention

# Potential savings from interventions

potential\_savings = tp \* avg\_customer\_value - (tp + fp) \* avg\_intervention\_cost

# ROI of the churn prevention program

roi = (potential\_savings / ((tp + fp) \* avg\_intervention\_cost)) if (tp + fp) > 0 else 0

# Return all metrics in a dictionary

metrics = {

'accuracy': accuracy,

'precision': precision,

'recall': recall,

'f1\_score': f1,

'false\_negatives': fn,

'false\_positives': fp,

'total\_business\_cost': total\_cost,

'cost\_per\_customer': cost\_per\_customer,

'retention\_rate': retention\_rate,

'intervention\_efficiency': intervention\_efficiency,

'potential\_savings': potential\_savings,

'roi': roi

}

return metrics

# TODO: FIND OPTIMAL CLASSIFICATION THRESHOLD

# ----------------------------------------

print("\nFinding optimal classification thresholds...")

def find\_optimal\_thresholds(model, X, y\_true, metric\_name='f1', fn\_cost=5, fp\_cost=1):

"""

Find the optimal classification threshold based on various metrics.

Parameters:

-----------

model : estimator

Trained classifier with predict\_proba method

X : array-like

Input features

y\_true : array-like

True class labels

metric\_name : str, optional

Metric to optimize ('f1', 'cost', 'precision', 'recall', 'roi')

fn\_cost : float, optional

Cost of a false negative

fp\_cost : float, optional

Cost of a false positive

Returns:

--------

dict

Dictionary with optimal thresholds for different metrics

"""

# Get predicted probabilities

y\_probs = model.predict\_proba(X)[:, 1]

# Initialize variables

thresholds = np.linspace(0.01, 0.99, 99)

metrics = {

'threshold': [],

'accuracy': [],

'precision': [],

'recall': [],

'f1': [],

'business\_cost': [],

'roi': []

}

# Calculate metrics for each threshold

for threshold in thresholds:

y\_pred = (y\_probs >= threshold).astype(int)

# Calculate standard metrics

tn, fp, fn, tp = confusion\_matrix(y\_true, y\_pred).ravel()

accuracy = (tp + tn) / (tp + tn + fp + fn)

precision = tp / (tp + fp) if (tp + fp) > 0 else 0

recall = tp / (tp + fn) if (tp + fn) > 0 else 0

f1 = 2 \* precision \* recall / (precision + recall) if (precision + recall) > 0 else 0

# Calculate business cost

business\_cost = (fn \* fn\_cost) + (fp \* fp\_cost)

# Calculate ROI (simplified)

avg\_customer\_value = 1000

avg\_intervention\_cost = 100

potential\_savings = tp \* avg\_customer\_value - (tp + fp) \* avg\_intervention\_cost

roi = (potential\_savings / ((tp + fp) \* avg\_intervention\_cost)) if (tp + fp) > 0 else 0

# Store results

metrics['threshold'].append(threshold)

metrics['accuracy'].append(accuracy)

metrics['precision'].append(precision)

metrics['recall'].append(recall)

metrics['f1'].append(f1)

metrics['business\_cost'].append(business\_cost)

metrics['roi'].append(roi)

# Find optimal thresholds

results = {

'accuracy': thresholds[np.argmax(metrics['accuracy'])],

'precision': thresholds[np.argmax(metrics['precision'])],

'recall': thresholds[np.argmax(metrics['recall'])],

'f1': thresholds[np.argmax(metrics['f1'])],

'business\_cost': thresholds[np.argmin(metrics['business\_cost'])],

'roi': thresholds[np.argmax(metrics['roi'])]

}

# Return threshold based on specified metric

if metric\_name == 'cost':

optimal\_threshold = results['business\_cost']

else:

optimal\_threshold = results[metric\_name]

return {

'optimal\_threshold': optimal\_threshold,

'all\_thresholds': results,

'metrics': pd.DataFrame(metrics)

}

# Find optimal thresholds for each model

thresholds\_baseline = find\_optimal\_thresholds(

baseline\_model, X\_test\_processed, y\_test, metric\_name='f1')

thresholds\_smote = find\_optimal\_thresholds(

smote\_model, X\_test\_processed, y\_test, metric\_name='f1')

thresholds\_weighted = find\_optimal\_thresholds(

weighted\_model, X\_test\_processed, y\_test, metric\_name='f1')

# Print optimal thresholds

print("\nOptimal Thresholds (F1 Score):")

print(f"Baseline model: {thresholds\_baseline['optimal\_threshold']:.4f}")

print(f"SMOTE model: {thresholds\_smote['optimal\_threshold']:.4f}")

print(f"Weighted model: {thresholds\_weighted['optimal\_threshold']:.4f}")

# Also find optimal thresholds for business cost

thresholds\_cost\_baseline = find\_optimal\_thresholds(

baseline\_model, X\_test\_processed, y\_test, metric\_name='cost')

thresholds\_cost\_smote = find\_optimal\_thresholds(

smote\_model, X\_

**Learning Outcomes for Task 2**

* Understanding the challenges posed by class imbalance in binary classification problems
* Learning practical techniques to handle imbalanced data, including SMOTE and class weighting
* Developing custom evaluation metrics that align with business objectives rather than just statistical accuracy
* Understanding how classification thresholds affect different performance metrics
* Learning to translate model performance into business value through cost-benefit analysis
* Gaining experience in visualizing and communicating model performance from a business perspective
* Understanding the tradeoffs between different optimization objectives when deploying models

**Key Takeaways for Task 2**

* **Class imbalance significantly impacts model performance**: Traditional accuracy metrics can be misleading when classes are imbalanced, as models may achieve high accuracy by simply predicting the majority class
* **Different imbalance handling techniques have different strengths**: SMOTE helps by creating synthetic samples, while class weighting adjusts the importance of classes during training without changing the data
* **Default classification thresholds (0.5) are rarely optimal**: Finding the optimal threshold depends on business objectives and can dramatically improve model performance on relevant metrics
* **Business costs should drive model optimization**: In churn prediction, the cost of missing a churner (false negative) is typically much higher than the cost of incorrectly flagging a loyal customer (false positive)
* **ROI and business metrics tell a more complete story**: When presenting results to stakeholders, business metrics like potential savings and ROI are more meaningful than technical metrics like accuracy or AUC
* **Precision-recall tradeoff is crucial in churn prediction**: Understanding and visualizing this tradeoff helps in selecting the right model and threshold for deployment
* **Model evaluation should match deployment context**: The evaluation framework should reflect how the model will be used in production and what business decisions it will inform