Machine Learning Assignment 2
Report

Name: Sistla Keerthana

Date: 5th October 2025

Topic: Using Random Forest to Learn Imbalanced Data

Reference Paper: https://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf

Github Repository: github.com/KeerthanaSistla/ML/blob/main/Assignment2/Report.doc

1. Project Overview

Imbalanced datasets are a major challenge in machine learning, where the minority class (e.g., malignant tumors) is underrepresented, leading to biased model performance and poor detection of critical cases.

This project focuses on improving classification performance on the Mammographic Masses dataset by using Weighted Random Forest (WRF) models with class weighting and decision threshold tuning. The goal is to maximize minority class detection (malignant tumors) while maintaining a reasonable majority class performance (benign tumors).

Unlike previous studies that used fixed thresholds and unweighted Random Forest (or balanced Random Forest), this work demonstrates hyperparameter tuning and threshold optimization to enhance model performance on imbalanced data.

2. Technologies and Tools Used

- **Programming Language:** Python
- **Dataset:** Mammographic Masses Dataset (UCI Repository)
- **Libraries:** scikit-learn, pandas, numpy, matplotlib, seaborn
- **Methods and Model:** Weighted Random Forest, Label Encoding, Threshold Tuning

3. System Architecture

Raw Dataset (CSV): Load mammographic mass data from CSV.

Data Preprocessing: Handle missing values, encode categorical variables, and normalize numerical features.

Weighted Random Forest: Train a Random Forest classifier with higher weights assigned to the minority class, improving model focus on underrepresented classes.

Hyperparameter Tuning: Optimize parameters like class_weight to improve model performance.

Threshold Optimization: Adjust decision threshold by maximizing G-mean to balance minority and majority class recall.

Evaluate Metrics: Assess performance using Acc+ (recall for malignant), Acc- (recall for benign), Precision, F1-score, G-mean, and Weighted Accuracy.

The feedback loop between the Weighted Random Forest and Hyperparameter Tuning blocks represents iterative optimization, ensuring the best performing model configuration is achieved before final evaluation.

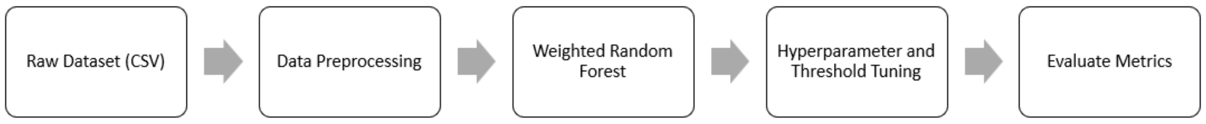


Fig 3.1: Workflow Diagram

4. Methodology

Step 1: Data Loading & Preprocessing

Dataset: /content/mammographic_masses.data

Features: BI-RADS, Age, Shape, Margin, Density, Severity

Encoding: Converting categorical values into numeric.

Step 2: Handle Class Imbalance

Class distribution: Benign (0): 516, Malignant (1): 445

WRF assigns higher weight to the minority class (class_weight={0:1, 1:2} and {0:1, 1:3})

Step 3: Model Building

Model Used: Weighted Random Forest (WRF)

Hyperparameters:

- n_estimators = 300
- max_depth = None (full depth)
- class_weight = 1:2 and 1:3

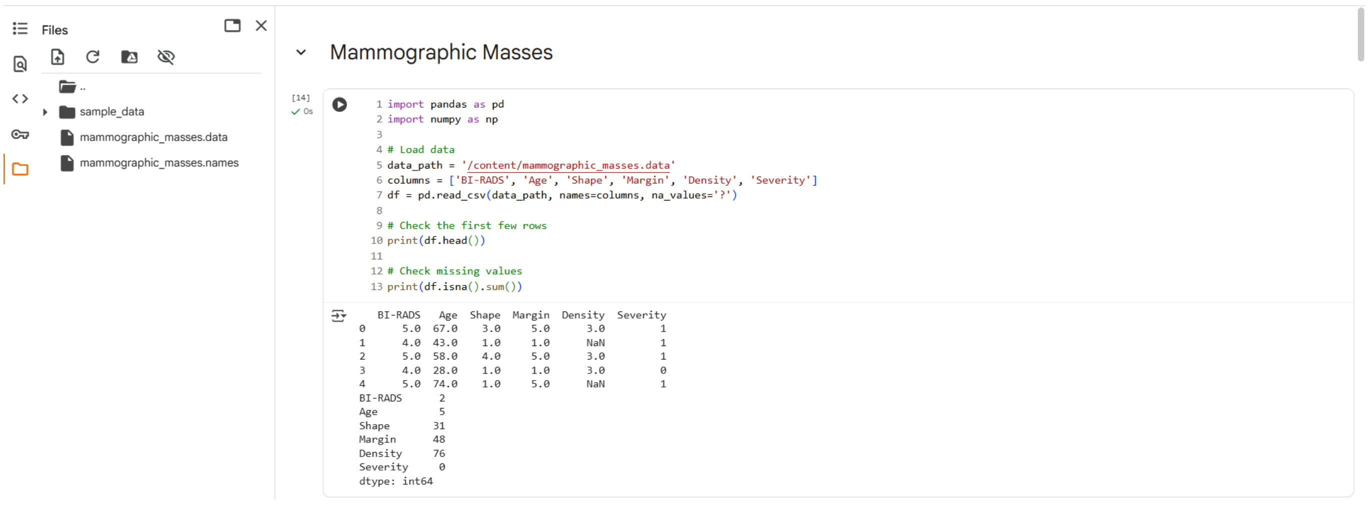
Train-test split: 70% training, 30% testing

Step 4: Threshold Optimization

Predicted probabilities and varied thresholds from 0.1 - 0.9

Optimal threshold selected by maximizing G-mean, balancing minority and majority recall.

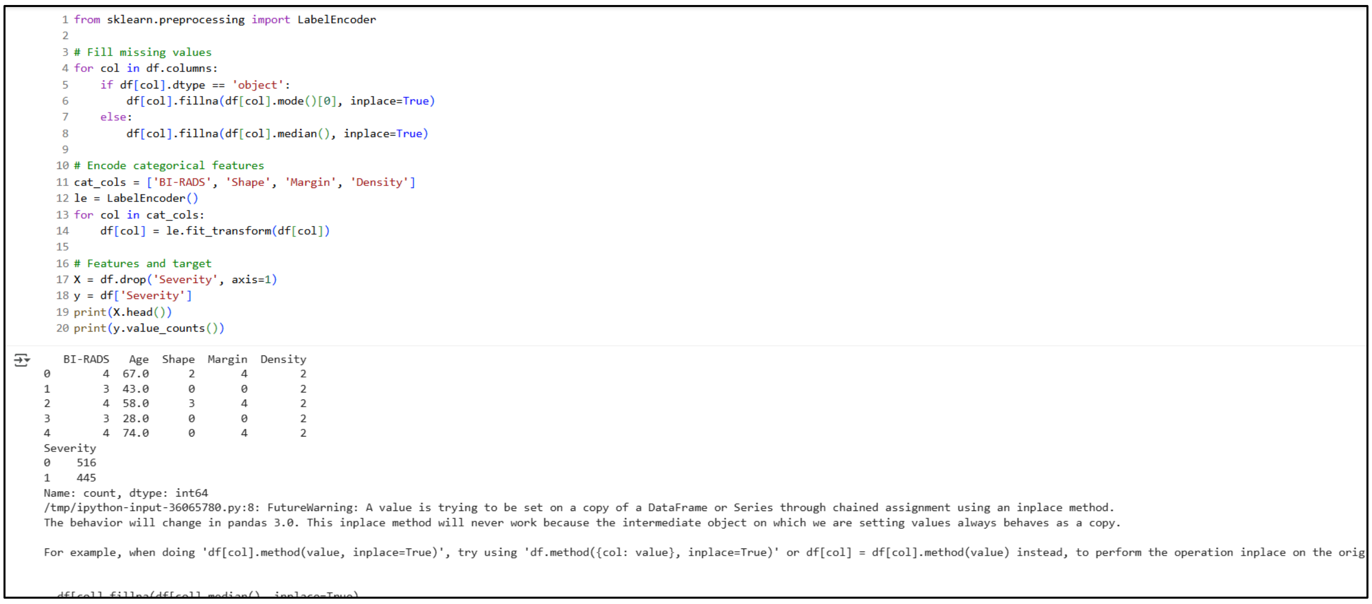
5. Screenshots



```python
import pandas as pd
import numpy as np

# Load data
data_path = '/content/mammographic_masses.data'
columns = ['BI-RADS', 'Age', 'Shape', 'Margin', 'Density', 'Severity']
df = pd.read_csv(data_path, names=columns, na_values='?')

# Check the first few rows
print(df.head())

# Check missing values
print(df.isna().sum())
```

```
   BI-RADS   Age  Shape  Margin  Density  Severity
0      5.0  67.0    3.0     5.0      3.0         1
1      4.0  43.0    1.0     1.0      NaN         1
2      5.0  58.0    4.0     5.0      3.0         1
3      4.0  28.0    1.0     1.0      3.0         0
4      5.0  74.0    1.0     5.0      NaN         1
BI-RADS      2
Age          5
Shape       31
Margin      48
Density     76
Severity     0
dtype: int64
```

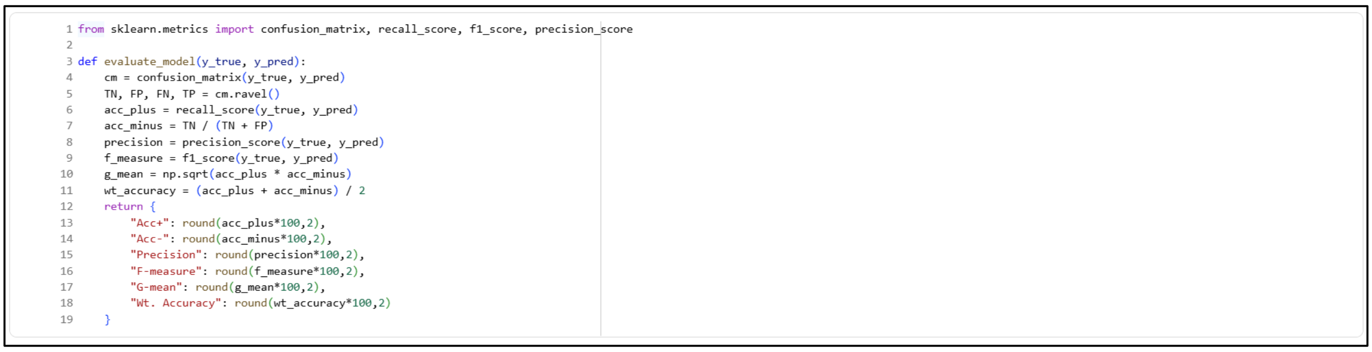
Fig. 5.1: Loading Data



```python
from sklearn.preprocessing import LabelEncoder

# Fill missing values
for col in df.columns:
    if df[col].dtype == 'object':
        df[col].fillna(df[col].mode()[0], inplace=True)
    else:
        df[col].fillna(df[col].median(), inplace=True)

# Encode categorical features
cat_cols = ['BI-RADS', 'Shape', 'Margin', 'Density']
le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])

# Features and target
X = df.drop('Severity', axis=1)
y = df['Severity']
print(X.head())
print(y.value_counts())
```

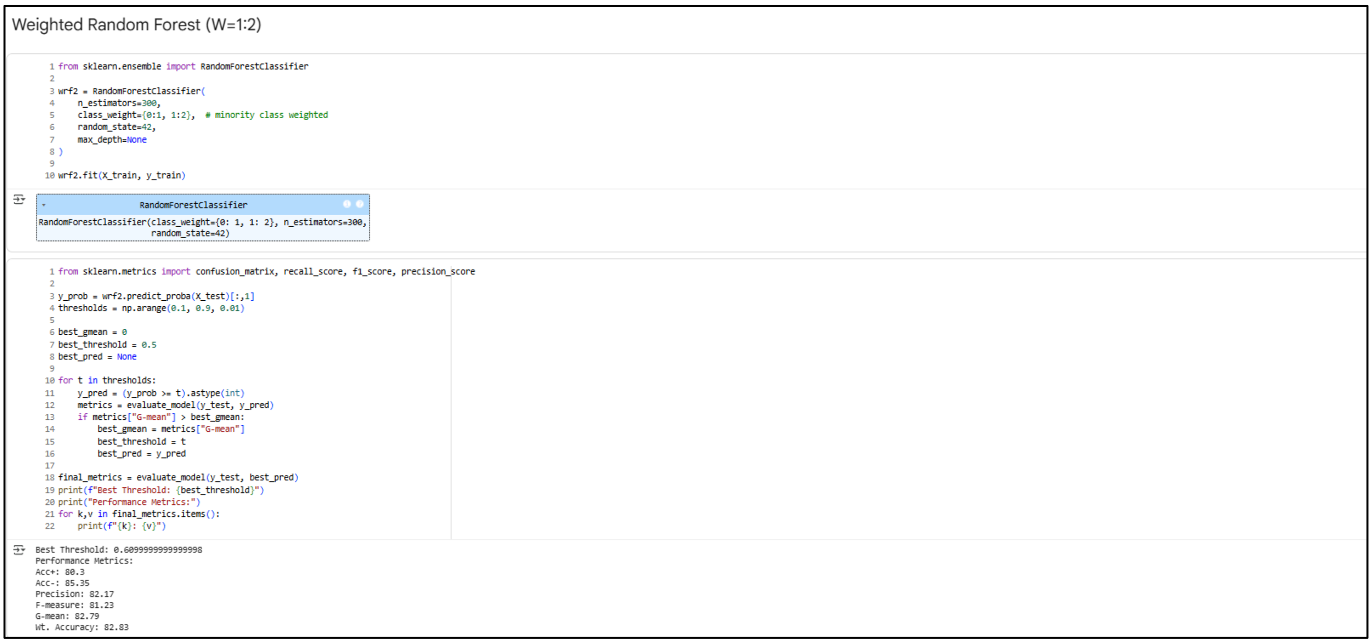
```
   BI-RADS   Age  Shape  Margin  Density
0        4  67.0      2       4        2
1        3  43.0      0       0        2
2        4  58.0      3       4        2
3        3  28.0      0       0        2
4        4  74.0      0       4        2
Severity
0    516
1    445
Name: count, dtype: int64
/tmp/ipython-input-36065780.py:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the orig

  df[col].fillna(df[col].median(), inplace=True)
```

Fig 5.2: Data Preprocessing



```python
from sklearn.metrics import confusion_matrix, recall_score, f1_score, precision_score

def evaluate_model(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    TN, FP, FN, TP = cm.ravel()
    acc_plus = recall_score(y_true, y_pred)
    acc_minus = TN / (TN + FP)
    precision = precision_score(y_true, y_pred)
    f_measure = f1_score(y_true, y_pred)
    g_mean = np.sqrt(acc_plus * acc_minus)
    wt_accuracy = (acc_plus + acc_minus) / 2
    return {
        "Acc+": round(acc_plus*100,2),
        "Acc-": round(acc_minus*100,2),
        "Precision": round(precision*100,2),
        "F-measure": round(f_measure*100,2),
        "G-mean": round(g_mean*100,2),
        "Wt. Accuracy": round(wt_accuracy*100,2)
    }
```

Fig 5.3: Model Evaluation Function



Weighted Random Forest (W=1:2)

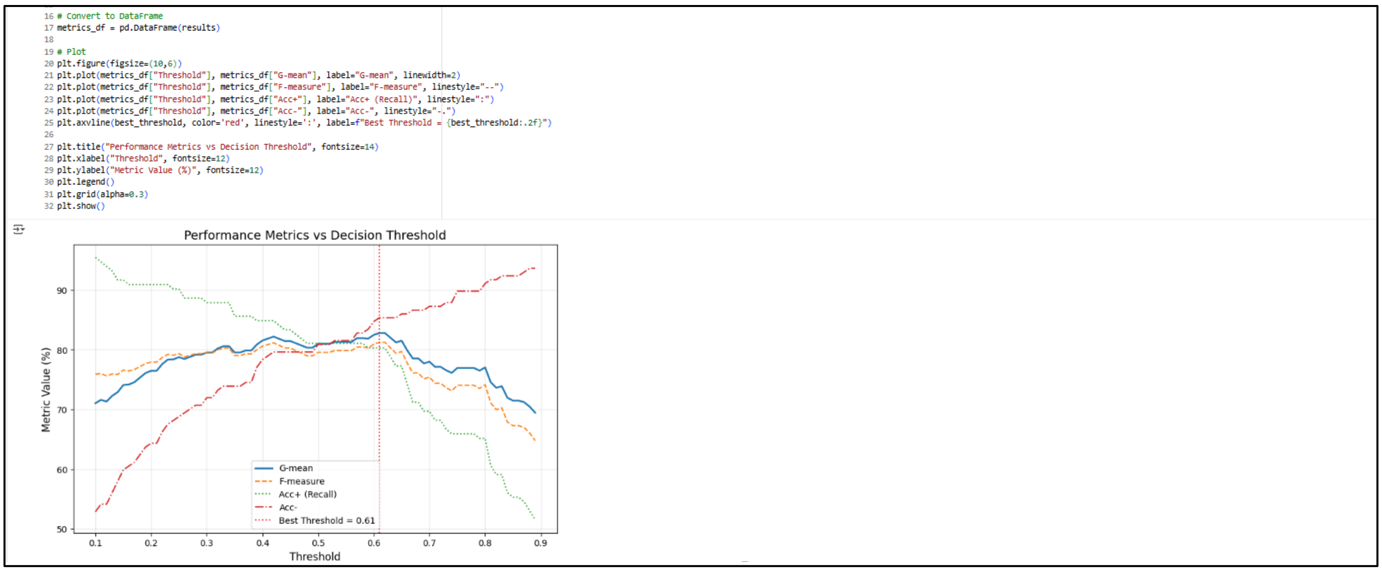
```python
from sklearn.ensemble import RandomForestClassifier

wrf2 = RandomForestClassifier(
    n_estimators=300,
    class_weight={0:1, 1:2},  # minority class weighted
    random_state=42,
    max_depth=None
)

wrf2.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight={0: 1, 1: 2}, n_estimators=300,
                       random_state=42)
```

```python
from sklearn.metrics import confusion_matrix, recall_score, f1_score, precision_score

y_prob = wrf2.predict_proba(X_test)[:,1]
thresholds = np.arange(0.1, 0.9, 0.01)

best_gmean = 0
best_threshold = 0.5
best_pred = None

for t in thresholds:
    y_pred = (y_prob >= t).astype(int)
    metrics = evaluate_model(y_test, y_pred)
    if metrics["G-mean"] > best_gmean:
        best_gmean = metrics["G-mean"]
        best_threshold = t
        best_pred = y_pred

final_metrics = evaluate_model(y_test, best_pred)
print(f"Best Threshold: {best_threshold}")
print("Performance Metrics:")
for k,v in final_metrics.items():
    print(f"{k}: {v}")
```

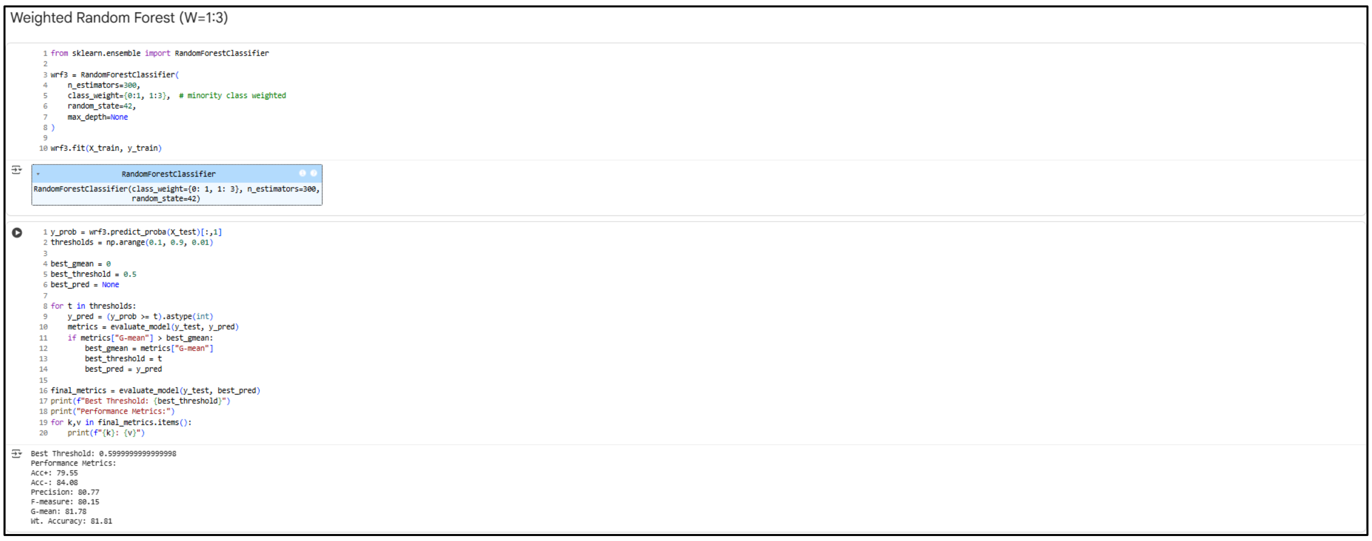
```
Best Threshold: 0.6099999999999998
Performance Metrics:
Acc+: 80.3
Acc-: 85.35
Precision: 82.17
F-measure: 81.23
G-mean: 82.79
Wt. Accuracy: 82.83
```

Fig 5.4: Weighted Random Forest (W=1:2)



```
# Convert to DataFrame
metrics_df = pd.DataFrame(results)

# Plot
plt.figure(figsize=(10,6))
plt.plot(metrics_df["Threshold"], metrics_df["G-mean"], label="G-mean", linewidth=2)
plt.plot(metrics_df["Threshold"], metrics_df["F-measure"], label="F-measure", linestyle="--")
plt.plot(metrics_df["Threshold"], metrics_df["Acc+"], label="Acc+ (Recall)", linestyle=":")
plt.plot(metrics_df["Threshold"], metrics_df["Acc-"], label="Acc-", linestyle="-.")
plt.axvline(best_threshold, color='red', linestyle=':', label=f"Best Threshold = {best_threshold:.2f}")

plt.title("Performance Metrics vs Decision Threshold", fontsize=14)
plt.xlabel("Threshold", fontsize=12)
plt.ylabel("Metric Value (%)", fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

Fig 5.5: Weighted Random Forest Plot



Weighted Random Forest (W=1:3)

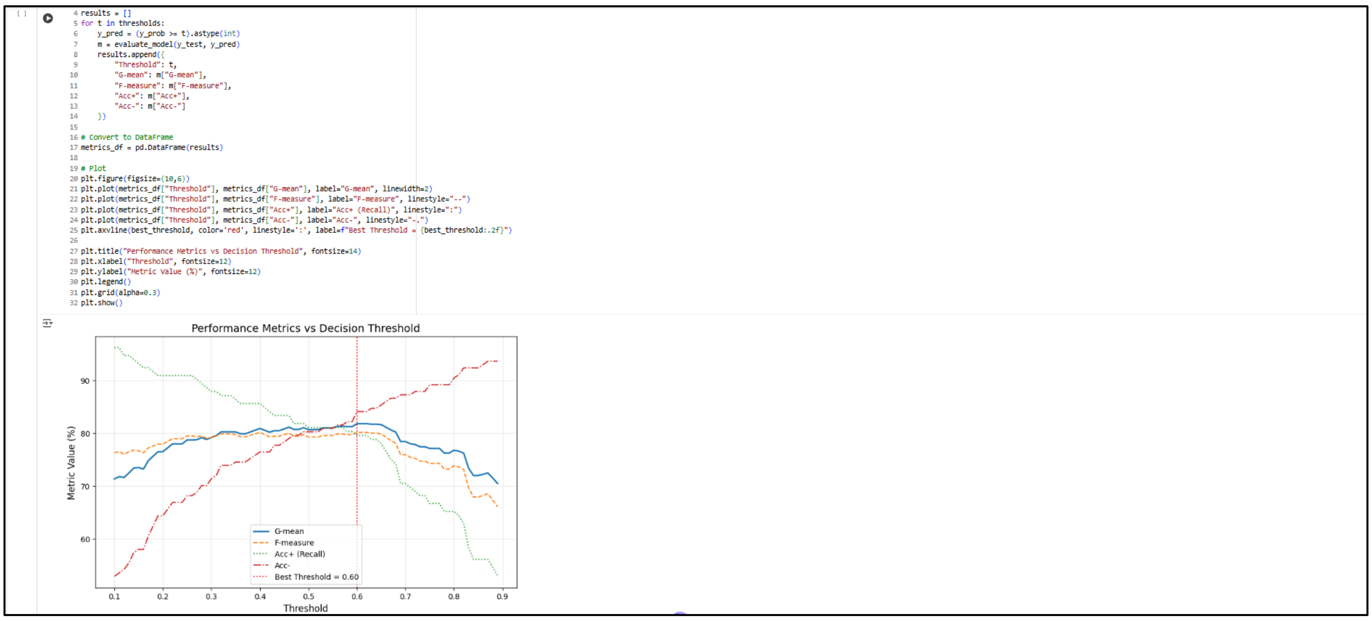
```
from sklearn.ensemble import RandomForestClassifier

wrf3 = RandomForestClassifier(
    n_estimators=300,
    class_weight={0:1, 1:3},  # minority class weighted
    random_state=42,
    max_depth=None
)

wrf3.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight={0: 1, 1: 3}, n_estimators=300,
                       random_state=42)
```

```
y_prob = wrf3.predict_proba(X_test)[:,1]
thresholds = np.arange(0.1, 0.9, 0.01)

best_gmean = 0
best_threshold = 0.5
best_pred = None

for t in thresholds:
    y_pred = (y_prob >= t).astype(int)
    metrics = evaluate_model(y_test, y_pred)
    if metrics["G-mean"] > best_gmean:
        best_gmean = metrics["G-mean"]
        best_threshold = t
        best_pred = y_pred

final_metrics = evaluate_model(y_test, best_pred)
print(f"Best Threshold: {best_threshold}")
print("Performance Metrics:")
for k,v in final_metrics.items():
    print(f"{k}: {v}")
```

```
Best Threshold: 0.5999999999999998
Performance Metrics:
Acc+: 79.55
Acc-: 84.08
Precision: 80.77
F-measure: 80.15
G-mean: 81.78
Wt. Accuracy: 81.81
```

Fig 5.6: Weighted Random Forest (W=1:3)



```
results = []
for t in thresholds:
    y_pred = (y_prob >= t).astype(int)
    m = evaluate_model(y_test, y_pred)
    results.append({
        "Threshold": t,
        "G-mean": m["G-mean"],
        "F-measure": m["F-measure"],
        "Acc+": m["Acc+"],
        "Acc-": m["Acc-"]
    })

# Convert to DataFrame
metrics_df = pd.DataFrame(results)

# Plot
plt.figure(figsize=(10,6))
plt.plot(metrics_df["Threshold"], metrics_df["G-mean"], label="G-mean", linewidth=2)
plt.plot(metrics_df["Threshold"], metrics_df["F-measure"], label="F-measure", linestyle="--")
plt.plot(metrics_df["Threshold"], metrics_df["Acc+"], label="Acc+ (Recall)", linestyle=":")
plt.plot(metrics_df["Threshold"], metrics_df["Acc-"], label="Acc-", linestyle="-.")
plt.axvline(best_threshold, color='red', linestyle=':', label=f"Best Threshold = {best_threshold:.2f}")

plt.title("Performance Metrics vs Decision Threshold", fontsize=14)
plt.xlabel("Threshold", fontsize=12)
plt.ylabel("Metric Value (%)", fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

Fig 5.7: Weighted Random Forest Plot

6. Testing and Results

Table 6.1: Comparison of Performance Metrics

Metric	WRF 2:1 (Paper)	WRF 3:1 (Paper)	WRF 2:1 (My Model)	WRF 3:1 (My Model)
Acc^+ (Recall +)	65.38	72.69	80.30	79.55
Acc^- (Recall -)	99.57	99.25	85.35	84.08
Precision	78.34	69.74	82.17	80.77
F-measure (F1)	71.28	71.18	81.23	80.15
G-mean	80.68	84.94	82.79	81.78
Weighted Accuracy	82.48	85.97	82.83	81.81
Best Threshold	N/A	N/A	0.6098	0.5998

Observation:

Minority Class Detection (Acc+): Custom WRF models improve malignant tumor recall, demonstrating better detection.

Majority Class Detection (Acc-): Slightly lower than paper's WRF, showing the trade-off in improving minority class detection.

Precision & F1-score: Both metrics are higher in custom models, indicating better balance between precision and recall.

G-mean & Weighted Accuracy: Comparable to paper's results, confirming balanced performance.

Threshold Tuning: Optimized thresholds (~0.6) outperform the paper's fixed cutoff (0.3), emphasizing minority class detection.

7. Learning Outcomes

- Learned to handle imbalanced datasets using class weighting.
- Understood threshold tuning to optimize G-mean.
- Improved understanding of evaluation metrics for imbalanced classification.
- Developed skills in hyperparameter tuning and model evaluation using Python.

8. Conclusion

The Weighted Random Forest with threshold optimization demonstrates significant improvement in minority class detection and F1-score on the Mammographic Masses dataset. This approach effectively balances the trade-off between minority and majority class performance, confirming the importance of class weighting and threshold tuning in handling imbalanced datasets.