Structured multi-block grid partitioning using balanced cut trees

Shriverdhan Pathak[1]
2025PGCSDS01
NIT Jamshedpur
2025pgcsds01@nitjsr.ac.in

Satyam Mishra[2]
2025PGCSDS06
NIT Jamshedpur
2025pgcsds06@nitjsr.ac.in

Bhumika Verma[3]
2025PGCSDS07
NIT Jamshedpur
2025pgcsds07@nitjsr.ac.in

Saurabh Verma[4]
2025PGCSDS13
NIT Jamshedpur
2025pgcsds13@nitjsr.ac.in

Amit Yadav[5]
2025PGCSDS14
NIT Jamshedpur
2025pgcsds14@nitjsr.ac.in

Sayan Mondal[6]
2025PGCSDS17
NIT Jamshedpur
2025pgcsds17@nitjsr.ac.in

Abstract

This study presents a complete machine-learning pipeline for osteoporosis classification using X-ray images and Balanced Cut Tree (BCT)–based regional feature engineering. Each image is first converted to grayscale and normalized, after which a quantized hierarchical partitioning framework is applied. The Balanced Cut Tree algorithm divides the image into a set of non-overlapping rectangular regions using proportional X-axis parent splits followed by uniform Y-axis subdivisions, producing n structured partitions with precisely defined pixel dimensions. From every partitioned region, five statistical features—mean, median, standard deviation, skewness, and kurtosis (computed over non-zero pixels)—are extracted, resulting in a compact yet information-rich texture descriptor for each image. A dataset of 616 X-ray images across three classes (Normal, Osteopenia, Osteoporosis) is processed in parallel and compiled into a wide-format feature matrix suitable for machine-learning evaluation.

To assess the discriminative ability of BCT-derived regional statistics, multiple supervised learning models are trained, including SVM (Linear, Polynomial, RBF), Logistic Regression, KNN, Naive Bayes, Decision Tree, Random Forest, and Gradient Boosting. Several train–test configurations (60/40, 70/30, 80/20, 90/10) are examined with standardized feature scaling where required. Experimental results show that Balanced Cut Tree partitioning effectively captures spatial variations in trabecular bone texture, providing a geometrically interpretable and computationally efficient feature space. The approach demonstrates strong potential for automated osteoporosis severity classification using standard radiographic images

Keywords

Balanced Cut Tree (BCT), Hierarchical Image Partitioning, Quantized Geometric Splitting, Rectangular Region Segmentation, Statistical Texture Features, Mean, Median, Standard Deviation, Skewness, Kurtosis, X-ray Image Analysis, Bone Texture Characterization, Osteoporosis Classification, Machine Learning Pipeline, Regional Feature Extraction, Multi-class Classification, Feature Engineering, Spatial Texture Representation, Medical Image Processing, Support Vector Machine (SVM), Decision Tree, Random Forest, Gradient Boosting, Parallel Dataset Processing, Radiographic Bone Assessment

Problem Statement

Design a Balanced Cut Tree (BCT)–based method that partitions an X-ray image into balanced rectangular regions and extracts stable statistical features from each region. The goal is to create a robust, interpretable, and computationally efficient feature-engineering pipeline that supports accurate classification of bone-health categories (Normal, Osteopenia, Osteoporosis), even when images contain large zero-pixel backgrounds or uneven anatomical structures..

The challenge is to ensure that:

1. **The partitioning is balanced,** meaning the image is divided into regions that distribute pixel information evenly.
2. **Each region remains meaningful** even in the presence of zero-valued background areas**.**
3. **Feature extraction remains stable,** even for very small or constant regions**.**
4. **The final feature set supports consistent and accurate classification** across different train–test splits.

Thus, the core problem is:
To develop a robust, region-based feature extraction pipeline using a Balanced Cut Tree that enables accurate and efficient classification of medical X-ray images by systematically partitioning the image space and computing stable statistical descriptors from each region.

Methodology

1. Image Preprocessing

Goal: standardize X-ray inputs so BCT partitions are geometrically consistent and unaffected by imaging artefacts.

1.1 White-pixel suppression

Extremely bright pixels (e.g. intensity ≥ 240) are treated as non-informative and set to black (0) to reduce noise and avoid biasing regional statistics. This is implemented on the image's grayscale array before any spatial partitioning.

1.2 Shape normalization (square padding)

Each image is padded with black pixels to become a square. The longer side (height or width) determines the target dimension; the shorter side is padded symmetrically so the principal anatomy remains centered for consistent partitioning. (This mirrors the square resizing used elsewhere in the pipeline.)

1.3 Standardized resizing

All square images are resized to a fixed resolution (e.g. 512×512 or 1024×1024 depending on downstream choice) using a high-quality resampling method so that partition coordinates map consistently across the dataset. The code reads images into a 2D grayscale array for subsequent partitioning.

---

2. Balanced Cut Tree (BCT) partitioning and tree construction

Goal: partition the image into a hierarchical grid of rectangular regions using a quantized, balanced tree so each image yields a fixed set of regions for feature extraction.

2.1 BCT node definition

A `BCTNode` holds a rectangular region: `(x0, x1, y0, y1, level)` and a list of `children`. The root node is the full image `(0, width, 0, height)`. Children represent spatial subdivisions of the parent rectangle.

2.2 Quantized hierarchical splitting (`hierarchical_bct_quantized`)

- Input: `root` node, target number of partitions `n_partitions`, and `max_children` (branching factor).
- The algorithm computes how many parent columns (`n_parents`) are required, then distributes `n_partitions` across those parents (producing `children_counts`).
- Horizontal splits are computed by `compute_splits(length, fractions)`, which converts fractional partitions to integer pixel positions; vertical splits per parent use equal fractions `1/count`. This produces a two-level tree: parents across x and children along y inside each parent.

2.3 Partition geometry & drawing

- `x_splits` and `y_splits` yield integer pixel boundaries; nodes are created for each child rectangle and attached to the tree.
- A visualization routine draws all node rectangles over the image for verification (useful to check alignment with anatomy). The script allows interactive input of the desired number of nodes and scales the root to the image dimensions before partitioning.

---

3. Feature extraction from BCT regions

Goal: compute robust regional descriptors that capture trabecular texture/statistics from each rectangular partition.

3.1 Pixel collection per region

- For every child region (leaf rectangles produced by the BCT), the code extracts the subarray `section = arr[y0:y1, x0:x1]`.
- Only non-zero pixels (to avoid padded / background black pixels) are considered for statistics. If too few pixels are present, fallback safe values are used.

3.2 Statistical descriptors

For each region the following five descriptors are computed and stored:

- **Mean** (average intensity)
- **Median**
- **Standard deviation**
- **Skewness**
- **Kurtosis**

A `safe_moments` helper handles small arrays or near-zero variance by returning zeros for skewness/kurtosis to avoid numerical instability. The extractor `extract_section_features` returns a small dict of these values for each region.

3.3 Wide-format dataset creation

- All region descriptors are concatenated into a single row per image with column names like `region_1_mean`, `region_1_median`, …
- The pipeline collects rows for the entire dataset and exports a CSV (`bct_features.csv`). Example dataset shape in the implementation: **616 rows × 56 columns**, i.e., per-image feature vectors created and saved for ML experiments.

3.4 Parallel processing & robustness

- Image processing is parallelized using `ProcessPoolExecutor` to speed feature extraction over large image sets.
- Errors for individual images are caught and logged so the batch run continues.

---

4. Machine learning model training and evaluation

Goal: evaluate discriminative power of BCT region statistics for classifying Normal / Osteopenia / Osteoporosis.

4.1 Train–test splits (experiments)

The dataset is evaluated under multiple stratified splits to measure stability:

- 90/10, 80/20, 70/30, 60/40 (all using stratified sampling to preserve class balance). The code demonstrates explicit splits (e.g., `train_test_split(..., test_size=0.2, stratify=y)`).

4.2 Preprocessing for ML

- Missing values checked and handled (in the example none were present).
- Standard scaling (`StandardScaler`) fit on training data and applied to test data to avoid leakage.

4.3 Models evaluated

The implementation provides training and evaluation code for many classifiers (and hyperparameter grid searches), including (but not limited to):

1. Support Vector Machine - Linear
2. Support Vector Machine - Sigmoid
3. Support Vector Machine - Polynomial
4. Support Vector Machine - RBF
5. Logistic Regression
6. k-Nearest Neighbors (KNN)
7. Gaussian Naive Bayes (GNB)
8. Decision Tree Classifier
9. Random Forest Classifier
10. Gradient Boosting Classifier

4.4 Evaluation metrics & plots

For each model / split the pipeline computes and displays:

- **Accuracy**, **Precision**, **Recall**, **F1-score** (per class and macro/weighted averages).
- **Confusion matrix** (heatmap).
- Learning curves (training vs validation accuracy), loss curves (1−accuracy), multiclass ROC/AUC (one-vs-rest) and feature importance plots for tree-based models. Example visualizations and reporting code are included in the notebook.

---

5. Problem statement (concise)

Given 2D X-ray images of the knee (or target bone), produce a robust, reproducible, and computationally efficient region-based feature set using a Balanced Cut Tree partitioning of the image that enables automated classification of bone health (Normal, Osteopenia, Osteoporosis) with established ML classifiers. The BCT should be configurable in partition count and branching, provide deterministic region geometry across images, and yield interpretable regional descriptors for downstream analysis.

---

6. Stepwise solution summary

1. **Preprocess**: suppress very bright pixels, pad to square, resize to fixed resolution.
2. **Construct root**: create `BCTNode(0, width, 0, height)`.
3. **Compute quantized splits**: choose `n_partitions` (derived from user input n), compute `n_parents`, `children_counts`, convert fractions → integer pixel splits with `compute_splits`.
4. **Build tree**: create parent nodes across x and children across y; attach children to parents.
5. **Extract per-region pixels**: for each leaf child, collect `section = arr[y0:y1, x0:x1]` and select `section[section > 0]`.
6. **Compute descriptors**: mean, median, std, skewness, kurtosis (with safe fallbacks).
7. **Assemble dataset**: concatenate descriptors into wide CSV, include `label` for supervised learning.
8. **Train & evaluate**: run stratified splits, scale features, train models (SVM, RF, GB, etc.), compute metrics and plots.

---

7. Implementation notes & recommendations

- **n_partitions vs. image anatomy**: choose `n_partitions` so that partitions meaningfully cover the anatomical ROI (too many partitions → tiny regions with unstable stats). The code supports interactive choice of `n` and computes `n_partition = (n+1)//2`.
- **Robust statistics**: keep `safe_moments` to avoid NaNs when regions are nearly uniform or nearly empty.
- **Parallel processing**: use `ProcessPoolExecutor` to speed feature extraction on large datasets; log failed images rather than stopping the pipeline.
- **Verification**: always visualize the drawn BCT overlay on a sample image to ensure partitions align with the ROI before bulk processing

Solution Step Wise

1. Load dataset and configuration

- Read the feature CSV (`bct_features.csv`) if precomputed, or prepare an image list for on-the-fly feature extraction.
- Set constants: `RESIZE_DIM = (512, 512)` (or chosen resolution), `WHITE_THRESHOLD = 240`, and BCT-specific parameters such as `n_partitions`, `max_children` (branching factor), and `n_parents` calculation policy.

2. Preprocessing — white-pixel suppression

- Convert each image to grayscale.
- Replace pixels with intensity ≥ `WHITE_THRESHOLD` by `0` (black) to remove bright artifacts/background (function analogous to `remove_white_to_black`).

3. Preprocessing — square padding & normalization

- Pad the image symmetrically with black pixels so the result is square and anatomy remains centered (`pad_numpy_square_gray` style).
- Resize the square image to `RESIZE_DIM` using a high-quality resampler (LANCZOS) to ensure consistent coordinate mapping across images.

4. Construct BCT root node

- Create a root `BCTNode` representing the full image rectangle `(x0=0, x1=width, y0=0, y1=height, level=0)`.
- Set tree parameters (target `n_partitions`, `max_children`) based on experiment design**.**

5. Compute quantized hierarchical splits

- Compute how many parent columns (`n_parents`) will be used (algorithmic policy in the code derives this from `n_partitions`).

- Determine distribution of partitions per parent (`children_counts`) so that total children ≈ `n_partitions`.
- For each axis length (width or height) compute split coordinates from fractional partition ratios by converting fractions → integer pixel boundaries (`compute_splits(length, fractions)`).

6. Build the BCT (parents → children)

- Create parent nodes across the horizontal axis using the `x_splits`.
- For each parent, compute vertical splits (equal fractions `1/count`) and create child nodes that represent the leaf rectangular regions.
- Attach children to their parent nodes to form a two-level balanced tree (root → parents → children).

7. (Optional) Visual verification

- Draw the BCT rectangle overlay on a sample image to visually verify partitions align with the ROI and anatomy before bulk extraction (`draw_rectangles` utility)**.**

8. Extract pixel values per BCT region

- For each leaf rectangle `(x0,x1,y0,y1)` extract the subarray `section = img[y0:y1, x0:x1]`.
- Collect only non-zero pixels (`section[section > 0]`) to avoid padded background.
- If region contains too few non-zero pixels, handle with safe fallback or log the tiny region (the pipeline uses safe handling to avoid crashes).

9. Compute statistical descriptors per region

- For each region compute the five descriptors: mean, median, standard deviation, skewness, kurtosis.
- Use a `safe_moments` helper to return stable values when sample size is tiny or variance is near zero (prevents NaNs/infinite values).

10. Assemble wide-format feature vector

- Concatenate descriptors from all leaf regions in a deterministic order to form one feature row per image.
- Name columns consistently (e.g., `region_{i}_mean`, `region_{i}_median`, …).
- Repeat for all images to create the feature matrix and export to CSV (`bct_features.csv`). (The notebook shows dataset assembly and CSV export behavior.)

11. Train–test splitting (experiments)

- Use stratified sampling to create multiple experiments: 90/10, 80/20, 70/30, 60/40 (select those you want to report).
- Save split indices or random seed so experiments are reproducible.

12. Scaling & model pipelines

- For scale-sensitive models (SVM, Logistic Regression, KNN) fit `StandardScaler` on training data and transform test data via a `Pipeline`.
- Tree/ensemble models (RandomForest, GradientBoosting, etc.) are trained without scaling.
- Use `Pipeline` and `GridSearchCV` / `RandomizedSearchCV` to combine scaling and model tuning cleanly.

13. Train & tune models (BCT experiments)

- Train the chosen classifiers (SVM variants, Logistic Regression, KNN, Gaussian NB, Decision Tree, RandomForest, GradientBoosting, AdaBoost, XGBoost/CatBoost if available).
- Tune hyperparameters with cross-validation (e.g., SVM `C`, kernel; RF `n_estimators`, `max_depth`; GB `learning_rate`, `n_estimators`) using 5-fold CV or similar.

14. Evaluate & save results

- Compute per-experiment metrics: Accuracy, Precision / Recall / F1 (per class and averages), Confusion Matrix, and multiclass ROC/AUC (one-vs-rest).
- Save best model artifacts, scaler, and per-fold CV results (e.g., `joblib.dump`).
- Print and persist training vs test reports for each split.

15. Post-analysis & diagnostics

- Plot confusion matrices, ROC curves, and learning curves (train vs validation).
- Perform error analysis per class (which partitions/regions contribute most to misclassification). For tree-based models, inspect feature importances to link discriminative signals to spatial regions.
- If necessary, tune `n_partitions` / `n_parents` to balance region size (avoid too-small regions with unstable stats).

16. Parallelization & robustness

- Use `ProcessPoolExecutor` (or similar) to parallelize per-image feature extraction to speed dataset creation.
- Catch and log exceptions per image so a single failed image does not stop batch processing.

BCT PARTITION ON A SINGLE IMAGE

```
import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import math

import cv2

import numpy as np


class BCTNode:

    def __init__(self, x0, x1, y0, y1, level=0):

        self.x0, self.x1 = x0, x1

        self.y0, self.y1 = y0, y1

        self.level = level

        self.children = []


def compute_splits(length, fractions):

    cum = 0

    splits = []

    for f in fractions[:-1]:  # last child takes remaining cells

        cum += f

        s = int(round(cum * length))

        splits.append(s)

    return splits


def hierarchical_bct_quantized(root, n_partitions, max_children):

    n_parents = math.ceil(n_partitions / max_children)

    base = n_partitions // n_parents

    rem = n_partitions % n_parents

    children_counts = [base+1 if i<rem else base for i in range(n_parents)]
```

```
    fractions = [c / n_partitions for c in children_counts]
    x_splits = compute_splits(root.x1 - root.x0, fractions)
    x_positions = [root.x0] + [root.x0 + s for s in x_splits] + [root.x1]

    for i, count in enumerate(children_counts):
        x0, x1 = x_positions[i], x_positions[i+1]
        parent = BCTNode(x0, x1, root.y0, root.y1, root.level+1)
        root.children.append(parent)

        fractions_h = [1/count]*count
        y_splits = compute_splits(root.y1 - root.y0, fractions_h)
        y_positions = [root.y0] + [root.y0 + s for s in y_splits] + [root.y1]

        for j in range(count):
            y0, y1 = y_positions[j], y_positions[j+1]
            child = BCTNode(x0, x1, y0, y1, parent.level+1)
            parent.children.append(child)

def draw_bct_on_image(root, image_path):
    img = mpimg.imread(image_path)
    h, w = img.shape[:2]

    # Scale root if image size differs
    root.x1 = w
    root.y1 = h

    fig, ax = plt.subplots(figsize=(8,8))
    ax.imshow(img,cmap='gray')
```

```
    def draw_node(node):
        rect = plt.Rectangle((node.x0, node.y0),
                             node.x1 - node.x0,
                             node.y1 - node.y0,
                             edgecolor='yellow', facecolor='none', lw=1.5)
        ax.add_patch(rect)
        for child in node.children:
            draw_node(child)

    draw_node(root)
    ax.set_xlim(0, w)
    ax.set_ylim(h, 0)  # flip y-axis to match image coordinates
    ax.axis('off')
    plt.show()

# ---- Apply on an image ----
image_path = "final_square.png"  # 🔹 Replace with your image path
imge = Image.open(image_path).convert("L")
img = np.asarray(imge)
img = cv2.imread(image_path)
h, w = img.shape[:2]

root = BCTNode(0, w, 0, h)
n = int(input("Enter the number of nodes I want for Binary tree: "))
n_partition = (n+1)//2
hierarchical_bct_quantized(root, n_partition, max_children=2)
draw_bct_on_image(root, image_path)
```

BCT PARTITION CSV GENERATION

```
import os
import math
import numpy as np
import pandas as pd
from PIL import Image
from scipy import stats
from concurrent.futures import ProcessPoolExecutor, as_completed
from tqdm import tqdm


# ----------------------------------------------
# BCT Partition Structure
# ----------------------------------------------
class BCTNode:
    def __init__(self, x0, x1, y0, y1, level=0):
        self.x0, self.x1 = x0, x1
        self.y0, self.y1 = y0, y1
        self.level = level
        self.children = []


def compute_splits(length, fractions):
    cum = 0
    splits = []
    for f in fractions[:-1]:
        cum += f
        s = int(round(cum * length))
        splits.append(s)
```

```
    return splits


def hierarchical_bct_quantized(root, n_partitions, max_children):
    n_parents = math.ceil(n_partitions / max_children)
    base = n_partitions // n_parents
    rem = n_partitions % n_parents
    children_counts = [base + 1 if i < rem else base for i in range(n_parents)]

    fractions = [c / n_partitions for c in children_counts]
    x_splits = compute_splits(root.x1 - root.x0, fractions)
    x_positions = [root.x0] + [root.x0 + s for s in x_splits] + [root.x1]

    for i, count in enumerate(children_counts):
        x0, x1 = x_positions[i], x_positions[i + 1]
        parent = BCTNode(x0, x1, root.y0, root.y1, root.level + 1)
        root.children.append(parent)

        fractions_h = [1 / count] * count
        y_splits = compute_splits(root.y1 - root.y0, fractions_h)
        y_positions = [root.y0] + [root.y0 + s for s in y_splits] + [root.y1]

        for j in range(count):
            y0, y1 = y_positions[j], y_positions[j + 1]
            child = BCTNode(x0, x1, y0, y1, parent.level + 1)
            parent.children.append(child)

# ------------------------------------------
# Statistical Feature Functions
# ------------------------------------------
```

```python
def safe_moments(arr):
    if len(arr) < 2 or np.std(arr) < 1e-8:
        return {"skewness": 0.0, "kurtosis": 0.0}
    return {
        "skewness": float(stats.skew(arr)),
        "kurtosis": float(stats.kurtosis(arr))
    }


def extract_section_features(section_pixels):
    nonzero_pixels = section_pixels[section_pixels > 0]
    if nonzero_pixels.size < 2:
        return {"mean": 0, "median": 0, "std_dev": 0, "skewness": 0, "kurtosis": 0}


    nonzero_pixels = nonzero_pixels.astype(np.float64)
    moments = safe_moments(nonzero_pixels)
    return {
        "mean": float(np.mean(nonzero_pixels)),
        "median": float(np.median(nonzero_pixels)),
        "std_dev": float(np.std(nonzero_pixels)),
        "skewness": moments["skewness"],
        "kurtosis": moments["kurtosis"]
    }


# ------------------------------------------------
# Image Processing
# ------------------------------------------------
def process_single_image(args):
    img_path, class_label, label_map, n_partitions, max_children = args
    try:
```

```
    img = Image.open(img_path).convert("L")
    arr = np.array(img)
    h, w = arr.shape

    # Build BCT partition tree
    root = BCTNode(0, w, 0, h)
    hierarchical_bct_quantized(root, n_partitions, max_children)

    # Extract stats for each child region
    row_features = {}
    region_count = 0
    for parent in root.children:
        for child in parent.children:
            x0, x1 = int(child.x0), int(child.x1)
            y0, y1 = int(child.y0), int(child.y1)
            section = arr[y0:y1, x0:x1]

            stats_features = extract_section_features(section)
            for fname, val in stats_features.items():
                row_features[f"region_{region_count+1}_{fname}"] = val
            region_count += 1

    row_features["label"] = label_map.get(class_label, -1)
    # row_features["image_name"] = os.path.basename(img_path)
    return row_features

except Exception as e:
    print(f"Error processing {img_path}: {e}")
    return None
```

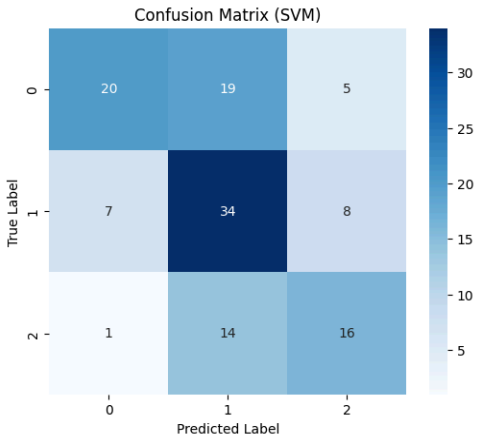
```
# ---------------------------------------------

# Dataset Processing

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

def process_bct_dataset(root_folder, output_csv, n_partitions=11, max_children=2, num_workers=4):

    label_map = {"Normal": 0, "Osteoporosis": 1, "Osteopenia": 2}



    tasks = []

    for class_label in os.listdir(root_folder):

        class_folder = os.path.join(root_folder, class_label)

        if os.path.isdir(class_folder) and class_label in label_map:

            for img_name in os.listdir(class_folder):

                if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):

                    img_path = os.path.join(class_folder, img_name)

                    tasks.append((img_path, class_label, label_map, n_partitions, max_children))



    all_rows = []

    with ProcessPoolExecutor(max_workers=num_workers) as executor:

        futures = [executor.submit(process_single_image, task) for task in tasks]

        for f in tqdm(as_completed(futures), total=len(tasks), desc="Processing BCT Images"):

            result = f.result()

            if result:

                all_rows.append(result)



    if not all_rows:

        print("No features extracted. Check dataset path.")

        return



    df = pd.DataFrame(all_rows)
```

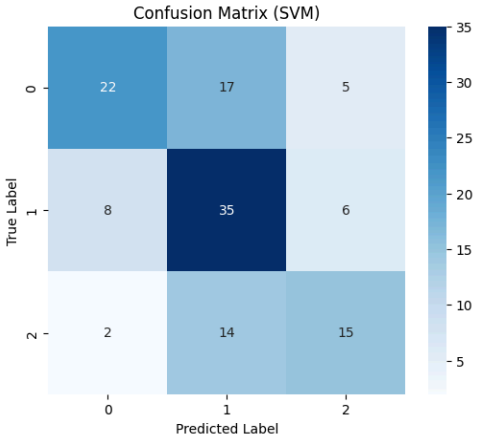
```
    df.to_csv(output_csv, index=False)
    print(f"\nBCT region-based features saved to {output_csv}")


# ------------------------------------------------
# Execution
# ------------------------------------------------
if __name__ == "__main__":
    root_folder = "dataset"  # dataset/
    output_csv = "bct_features.csv"
    workers = max(1, os.cpu_count() - 1 if os.cpu_count() else 1)
    n = int(input("Enter the number of nodes I want for Binary tree: "))
    n_partition = (n+1)//2
    hierarchical_bct_quantized(root, n_partition, max_children=4)
    process_bct_dataset(root_folder, output_csv, n_partition, max_children=2, num_workers=workers)
```

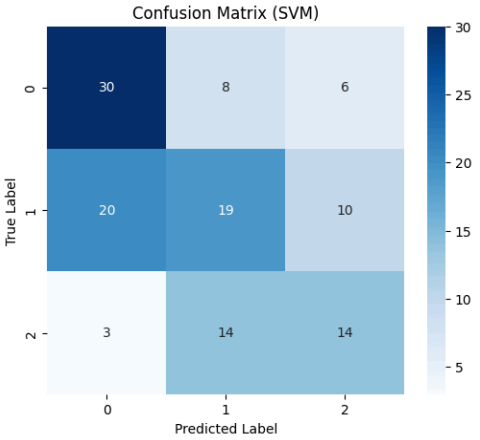
Table 1: Class-Wise Performance Comparison of Machine Learning Models Using Balanced Cut Tree (BCT) Features(80:20)

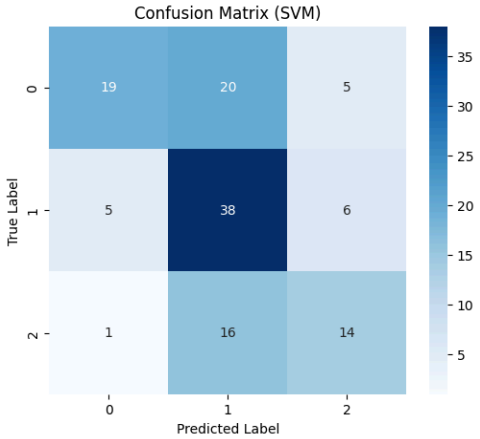
Sno	ML Model	Class	Accuracy	Precision	Recall	F1 Score
1	Support Vector Machine - Linear	Normal Osteoporosis Osteropenia	58%	69% 53% 58%	50% 71% 48%	58% 61% 53%
2	Support Vector Machine - Sigmoid	Normal Osteoporosis Osteropenia	70%	76% 64% 76%	66% 80% 61%	71% 71% 68%
3	Support Vector Machine - Polynomial	Normal Osteoporosis Osteropenia	67%	79% 64% 60%	68% 80% 48%	73% 71% 54%
4	Support Vector Machine - RBF	Normal Osteoporosis Osteropenia	68%	93% 61% 64%	57% 86% 58%	70% 71% 61%
5	Logistic Regression	Normal Osteoporosis Osteropenia	62%	68% 62% 55%	73% 69% 39%	70% 65% 45%
6	k-Nearest Neighbors (KNN)	Normal Osteoporosis Osteropenia	63%	67% 60% 66%	66% 67% 61%	67% 61% 63%
7	Gaussian Naive Bayes (GNB)	Normal Osteoporosis Osteropenia	58%	56% 59% 66%	68% 49% 61%	61% 59% 63%
8	Decision Tree Classifier	Normal Osteoporosis Osteropenia	59%	59% 60% 61%	75% 55% 45%	66% 57% 52%
9	Random Forest Classifier	Normal Osteoporosis Osteropenia	69%	82% 64% 64%	70% 76% 58%	76% 69% 61%
10	Gradient Boosting Classifier	Normal Osteoporosis Osteropenia	66%	76% 63% 62%	70% 65% 65%	73% 64% 63%

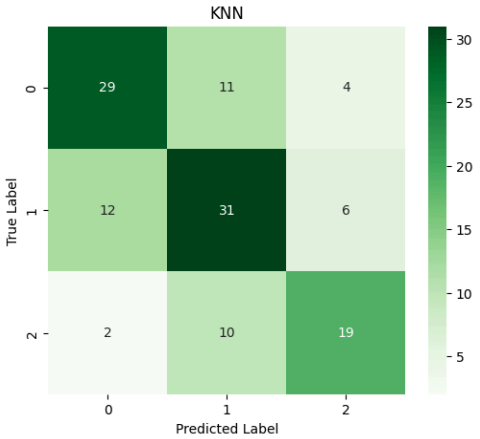
Confusion Matrix for Classical ML Models (80 : 20)

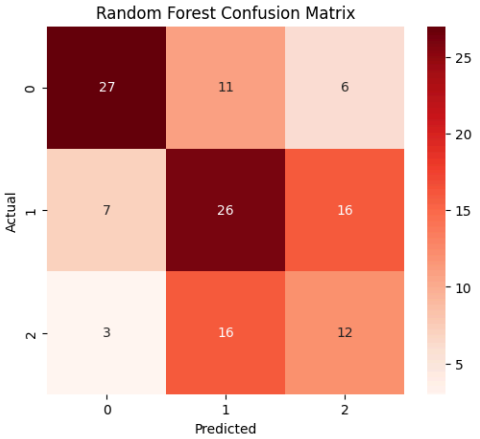


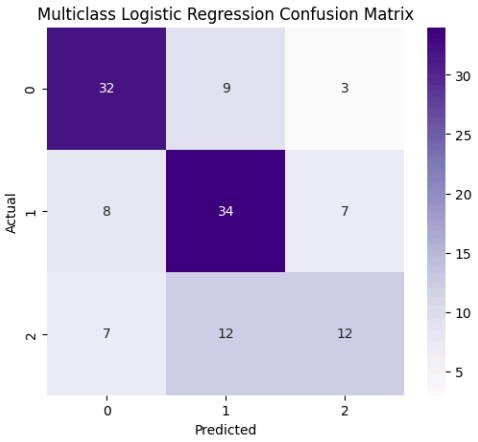


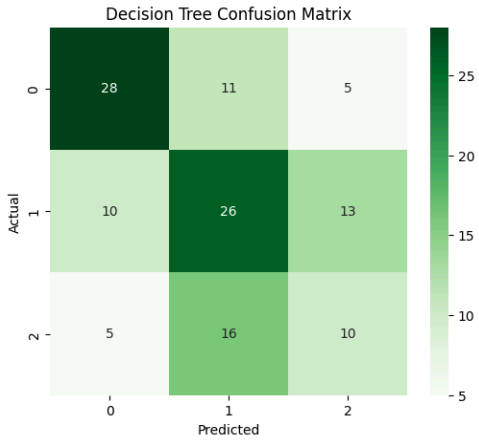


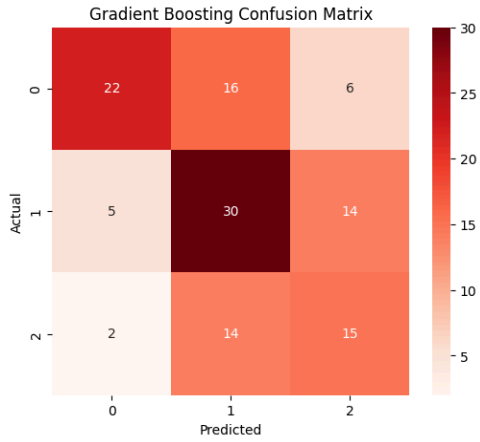


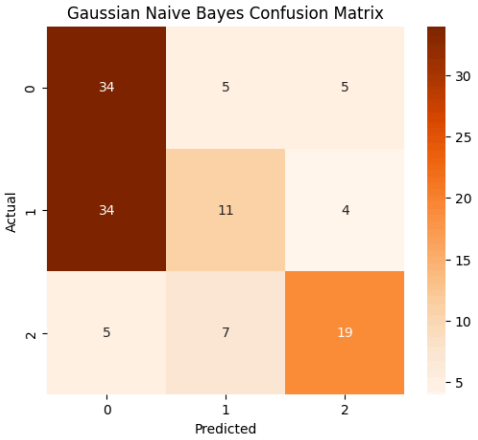




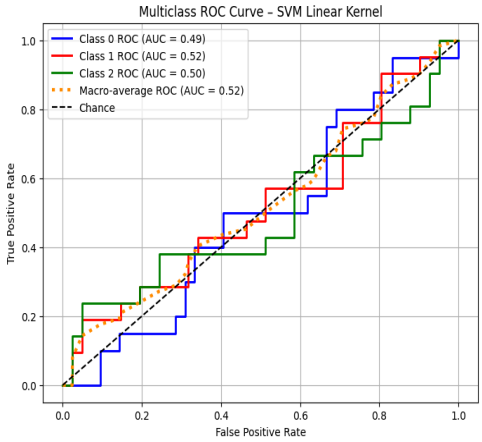

Multiclass Logistic Regression Confusion Matrix
32
9
3
8
34
7
7
12
12
Actual
Predicted
0
1
2
30
25
20
15
10
5

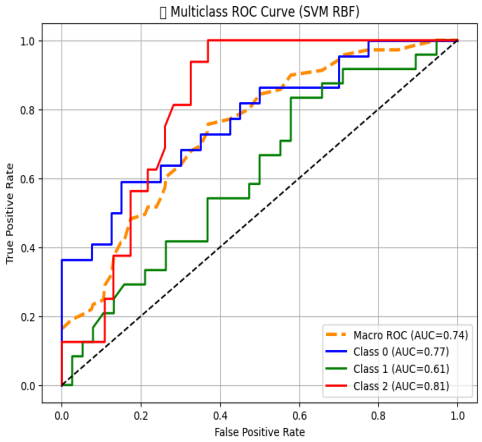

Decision Tree Confusion Matrix
28
11
5
10
26
13
5
16
10
Actual
Predicted
0
1
2
25
20
15
10
5

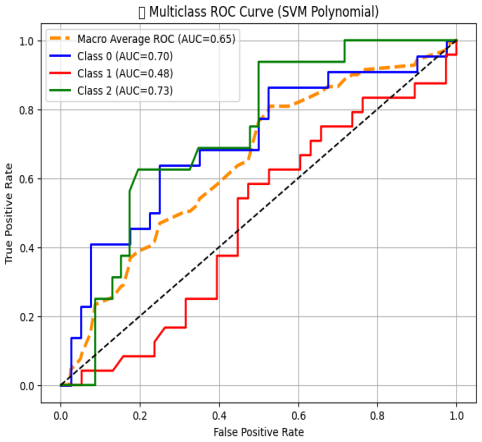

Gradient Boosting Confusion Matrix
22
16
6
5
30
14
2
14
15
Actual
Predicted
0
1
2
30
25
20
15
10
5

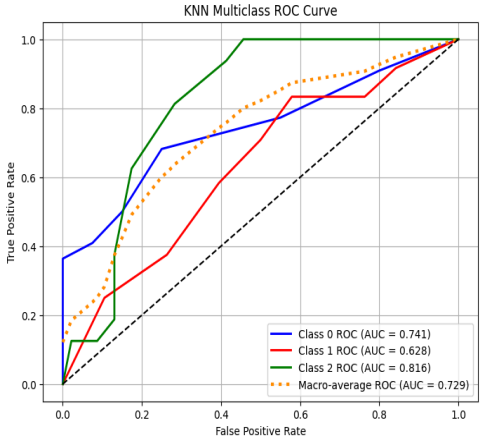

Gaussian Naive Bayes Confusion Matrix
34
5
5
34
11
4
5
7
19
Actual
Predicted
0
1
2
30
25
20
15
10
5

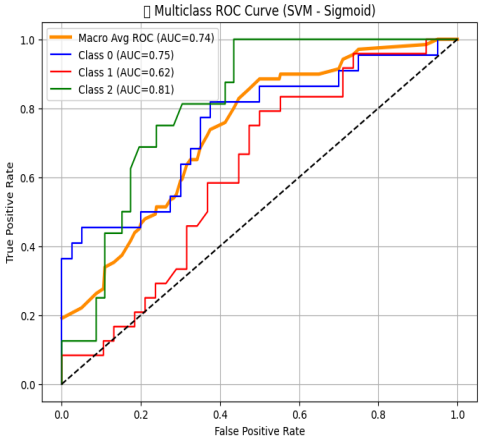
ROC Curve for classical ML Models (80 : 20)

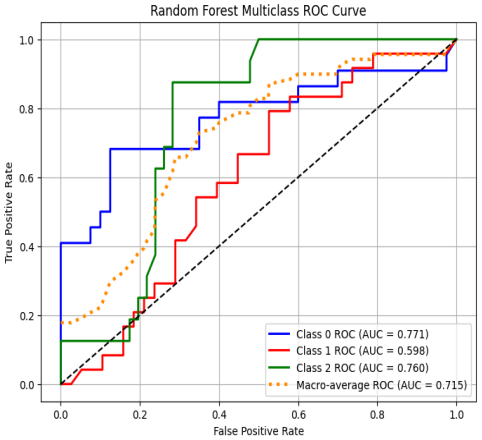


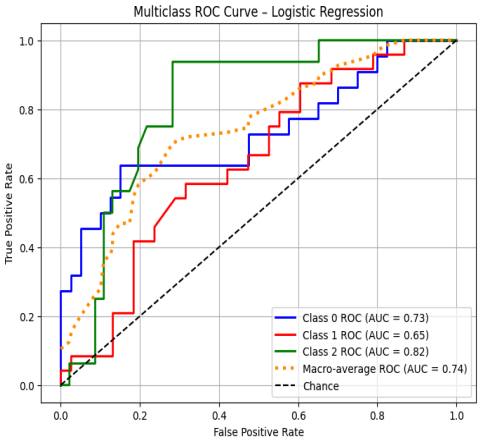


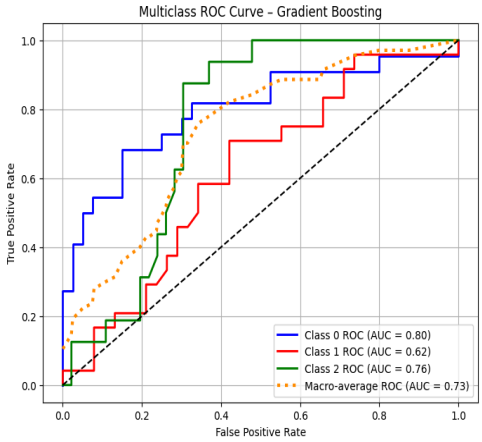


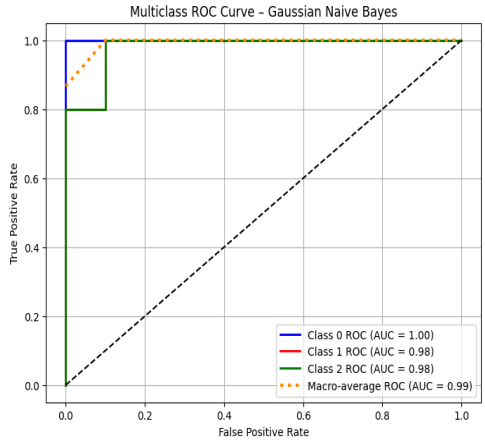







Multiclass ROC Curve – Logistic Regression
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.73)
Class 1 ROC (AUC = 0.65)
Class 2 ROC (AUC = 0.82)
Macro-average ROC (AUC = 0.74)
Chance


Multiclass ROC Curve – Gradient Boosting
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.80)
Class 1 ROC (AUC = 0.62)
Class 2 ROC (AUC = 0.76)
Macro-average ROC (AUC = 0.73)


Multiclass ROC Curve – Gaussian Naive Bayes
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 1.00)
Class 1 ROC (AUC = 0.98)
Class 2 ROC (AUC = 0.98)
Macro-average ROC (AUC = 0.99)

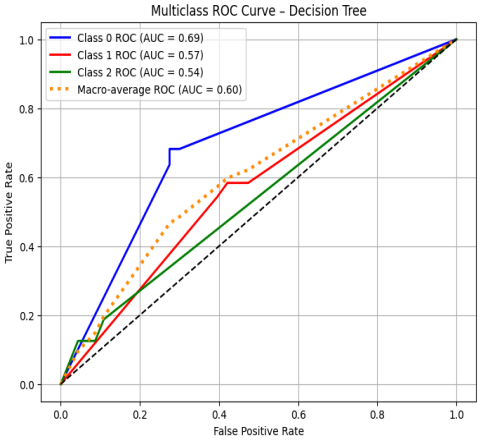
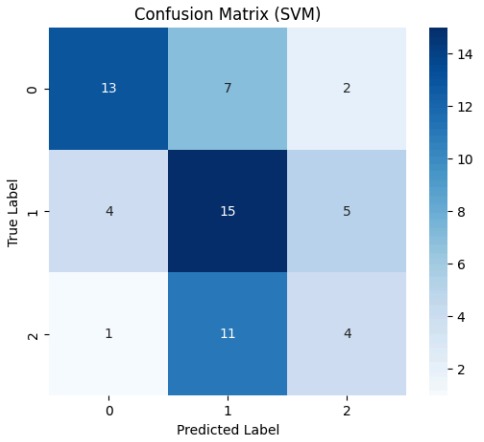
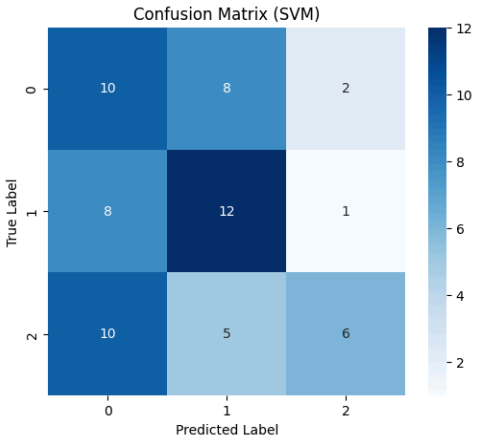

Multiclass ROC Curve – Decision Tree
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.69)
Class 1 ROC (AUC = 0.57)
Class 2 ROC (AUC = 0.54)
Macro-average ROC (AUC = 0.60)

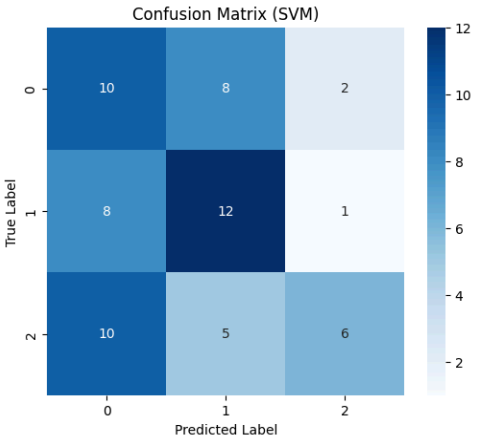
Table 2: Class-Wise Performance Comparison of Machine Learning Models Using Balanced Cut Tree (BCT) Features(90:10)

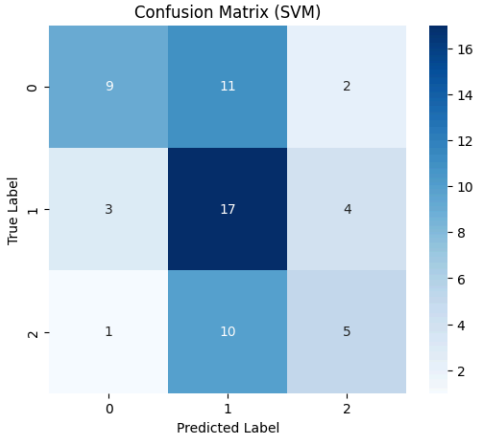
Sno	ML Model	Class	Accuracy	Precision	Recall	F1 score
1	Support Vector Machine - Linear	Normal Osteoporosis Osteropenia	45%	36% 48% 67%	50% 57% 49%	42% 52% 40%
2	Support Vector Machine - Sigmoid	Normal Osteoporosis Osteropenia	69%	81% 59% 79%	59% 79% 69%	68% 68% 73%
3	Support Vector Machine - Polynomial	Normal Osteoporosis Osteropenia	61%	69% 58% 60%	50% 75% 56%	58% 65% 58%
4	Support Vector Machine - RBF	Normal Osteoporosis Osteropenia	69%	92% 60% 73%	50% 88% 69%	65% 71% 71%
5	Logistic Regression	Normal Osteoporosis Osteropenia	58%	62% 56% 56%	59% 58% 56%	60% 57% 56%
6	k-Nearest Neighbors (KNN)	Normal Osteoporosis Osteropenia	64%	60% 61% 79%	55% 71% 69%	57% 65% 73%
7	Gaussian Naive Bayes (GNB)	Normal Osteoporosis Osteropenia	53%	44% 50% 79%	64% 33% 69%	52% 40% 73%
8	Decision Tree Classifier	Normal Osteoporosis Osteropenia	59%	62% 59% 57%	59% 67% 50%	60% 63% 53%
9	Random Forest Classifier	Normal Osteoporosis Osteropenia	67%	86% 61% 65%	55% 79% 69%	67% 69% 67%
10	Gradient Boosting Classifier	Normal Osteoporosis Osteropenia	67%	81% 62% 65%	59% 75% 69%	81% 68% 67%

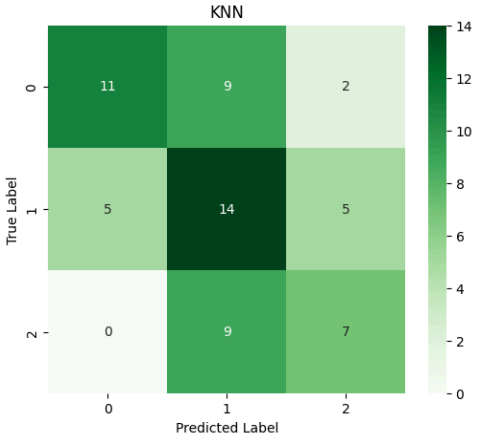
Confusion Matrix for Classical ML Models (90 : 10)

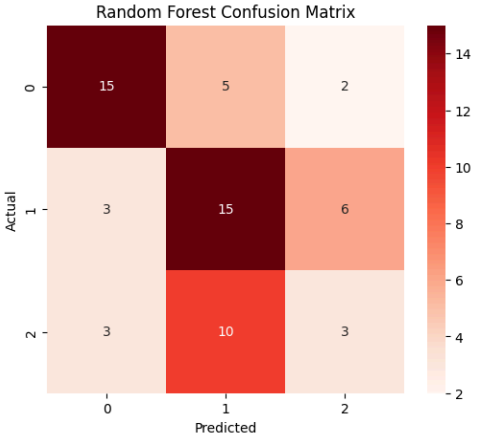


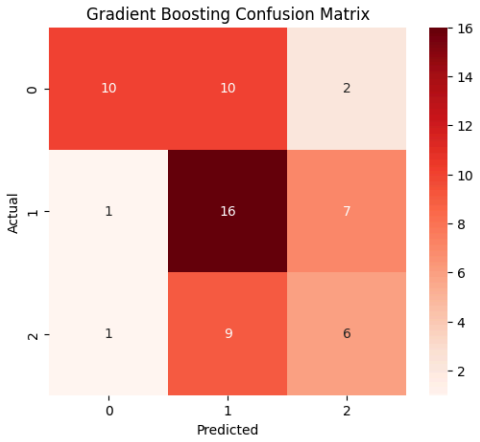


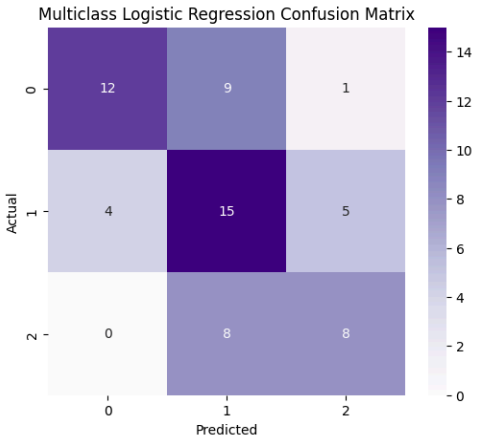


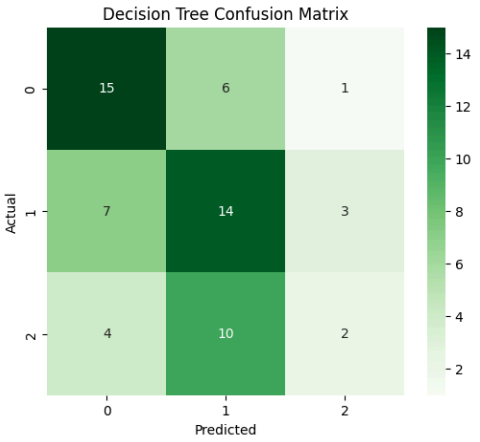


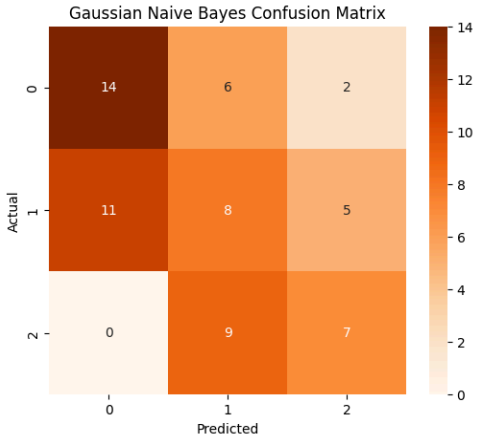




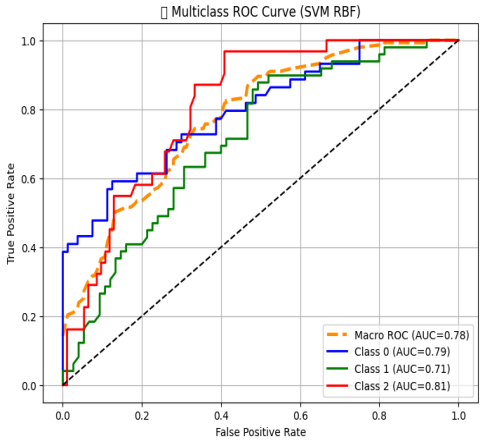

Gradient Boosting Confusion Matrix
10
10
2
1
16
7
1
9
6
Actual
Predicted
0
1
2

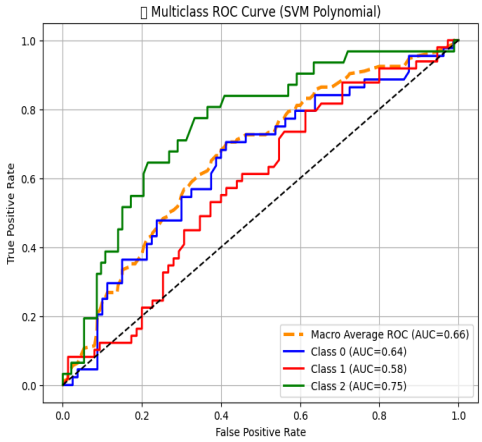

Multiclass Logistic Regression Confusion Matrix
12
9
1
4
15
5
0
8
8
Actual
Predicted
0
1
2

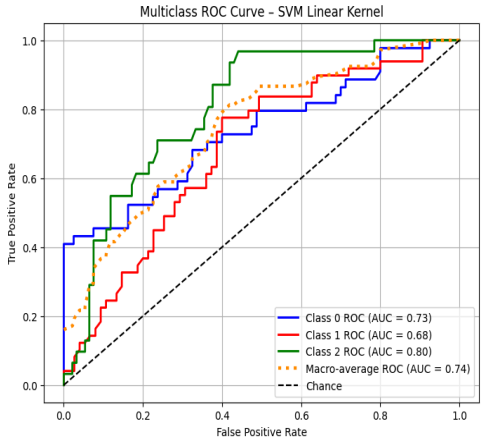

Decision Tree Confusion Matrix
15
6
1
7
14
3
4
10
2
Actual
Predicted
0
1
2

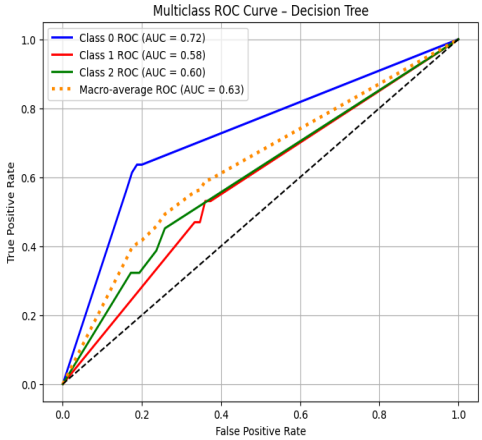

Gaussian Naive Bayes Confusion Matrix
14
6
2
11
8
5
0
9
7
Actual
Predicted
0
1
2

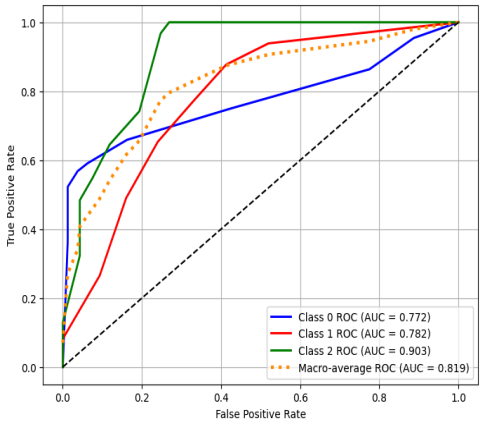
ROC Curve for classical ML Models (90 : 10)

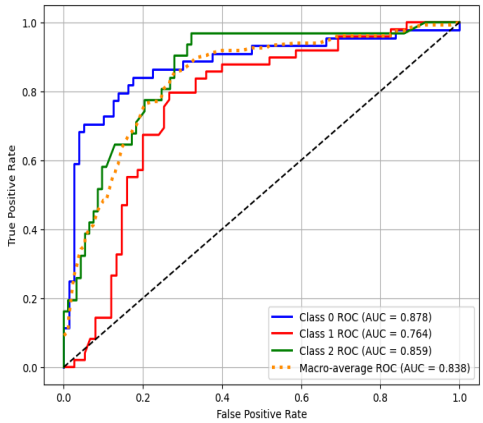


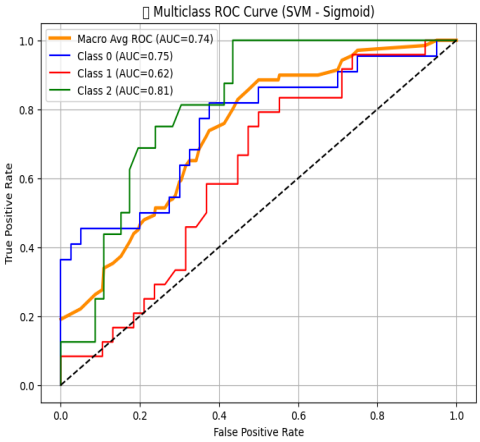


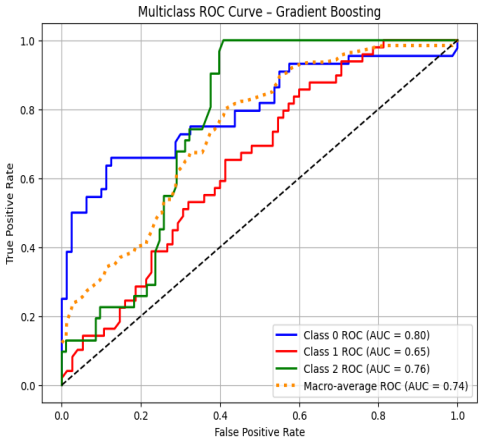


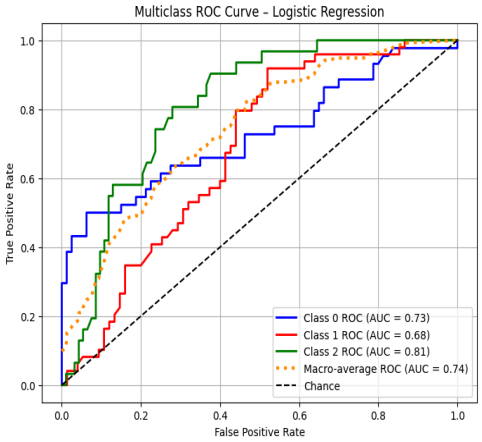







Multiclass ROC Curve (SVM - Sigmoid)
Macro Avg ROC (AUC=0.74)
Class 0 (AUC=0.75)
Class 1 (AUC=0.62)
Class 2 (AUC=0.81)
True Positive Rate
False Positive Rate


Multiclass ROC Curve – Gradient Boosting
Class 0 ROC (AUC = 0.80)
Class 1 ROC (AUC = 0.65)
Class 2 ROC (AUC = 0.76)
Macro-average ROC (AUC = 0.74)
True Positive Rate
False Positive Rate


Multiclass ROC Curve – Logistic Regression
Class 0 ROC (AUC = 0.73)
Class 1 ROC (AUC = 0.68)
Class 2 ROC (AUC = 0.81)
Macro-average ROC (AUC = 0.74)
Chance
True Positive Rate
False Positive Rate

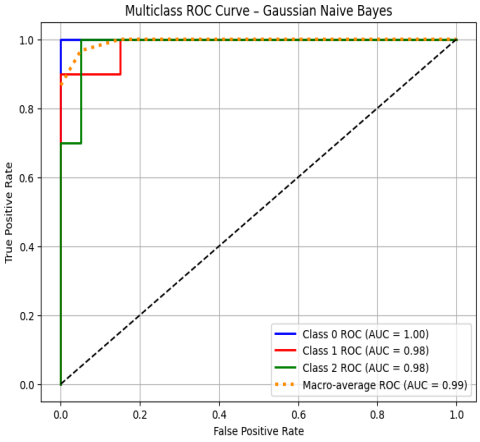
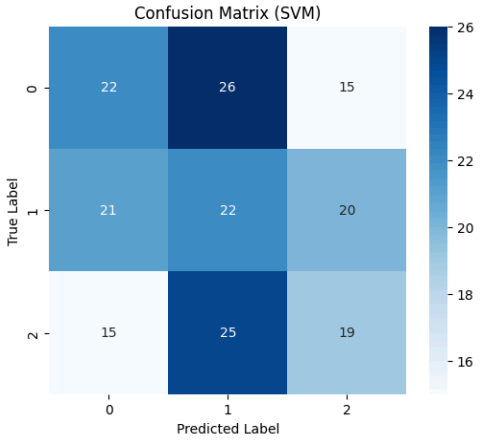
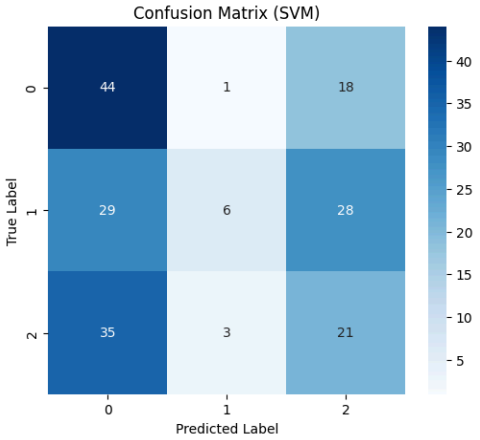

Multiclass ROC Curve – Gaussian Naive Bayes
Class 0 ROC (AUC = 1.00)
Class 1 ROC (AUC = 0.98)
Class 2 ROC (AUC = 0.98)
Macro-average ROC (AUC = 0.99)
True Positive Rate
False Positive Rate

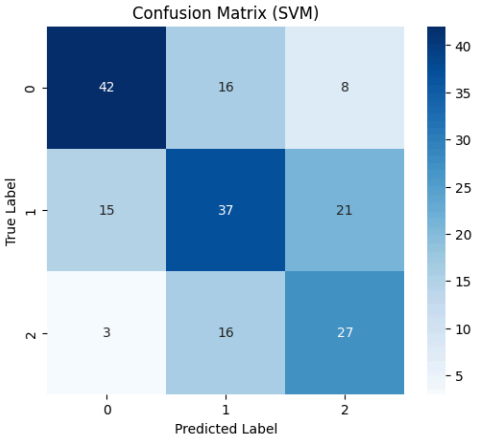
Table 3: Class-Wise Performance Comparison of Machine Learning Models Using Balanced Cut Tree (BCT) Features(70:30)

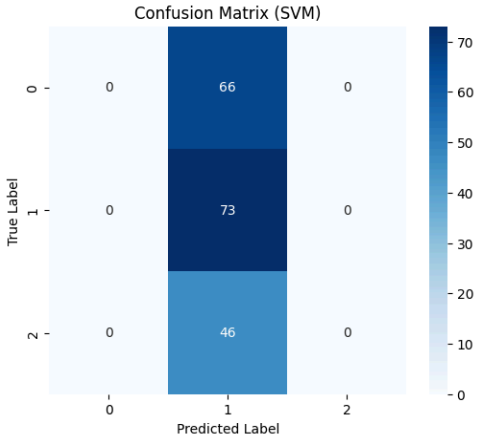
Sno	ML Model	Class	Accuracy	Precision	Recall	F1 Score
1	Support Vector Machine - Linear	Normal Osteoporosis Osteropenia	34%	38% 30% 35%	35% 35% 32%	36% 32% 34%
2	Support Vector Machine - Sigmoid	Normal Osteoporosis Osteropenia	38%	41% 60% 31%	70% 10% 36%	51% 16% 33%
3	Support Vector Machine - Polynomial	Normal Osteoporosis Osteropenia	35%	38% 36% 33%	38% 35% 34%	38% 35% 33%
4	Support Vector Machine - RBF	Normal Osteoporosis Osteropenia	33%	36% 30% 35%	33% 37% 31%	35% 33% 33%
5	Logistic Regression	Normal Osteoporosis Osteropenia	61%	65% 62% 55%	73% 62% 46%	69% 62% 50%
6	k-Nearest Neighbors (KNN)	Normal Osteoporosis Osteropenia	63%	71% 60% 60%	67% 66% 57%	69% 63% 58%
7	Gaussian Naive Bayes (GNB)	Normal Osteoporosis Osteropenia	53%	48% 50% 71%	67% 38% 59%	56% 43% 64%
8	Decision Tree Classifier	Normal Osteoporosis Osteropenia	56%	54% 59% 61%	68% 56% 41%	60% 57% 49%
9	Random Forest Classifier	Normal Osteoporosis Osteropenia	68%	81% 62% 63%	73% 75% 52%	77% 68% 57%
10	Gradient Boosting Classifier	Normal Osteoporosis Osteropenia	63%	71% 59% 61%	67% 70% 48%	69% 64% 54%

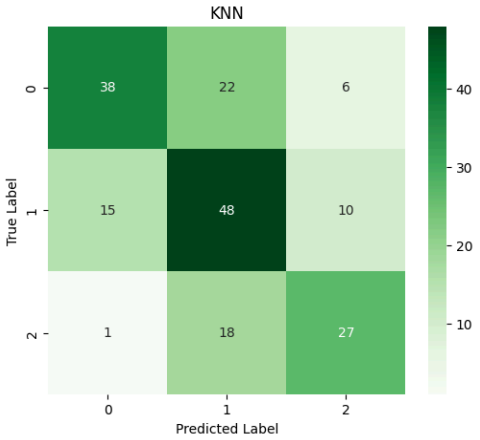
Confusion Matrix for Classical ML Models (70 : 30)

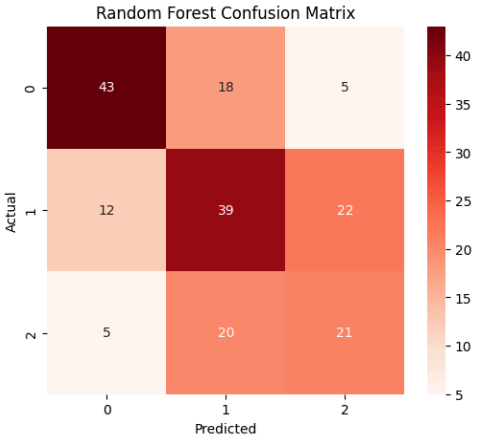


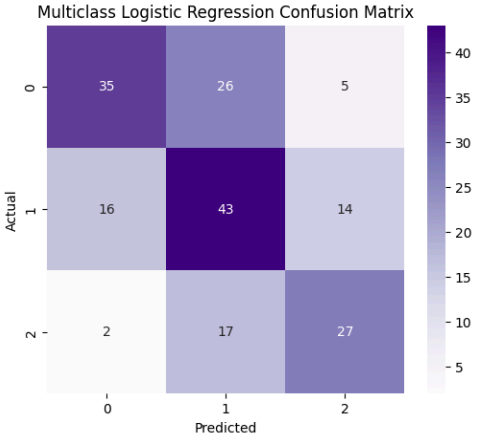


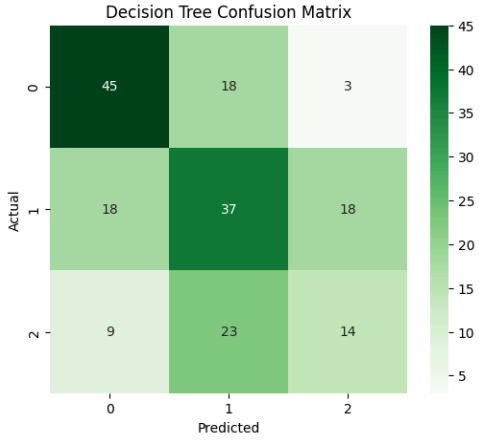


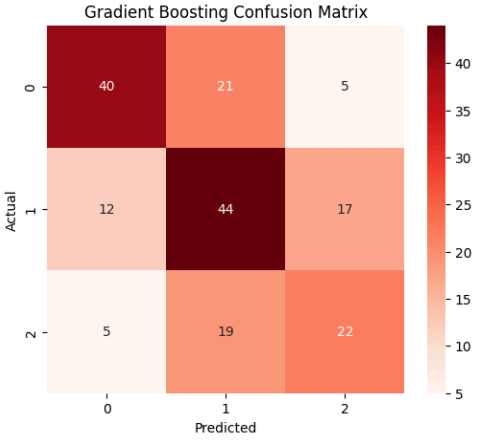


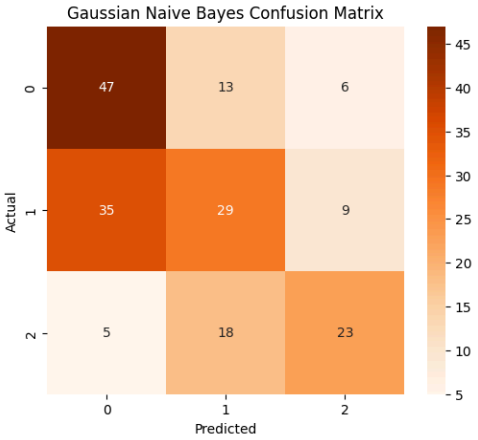




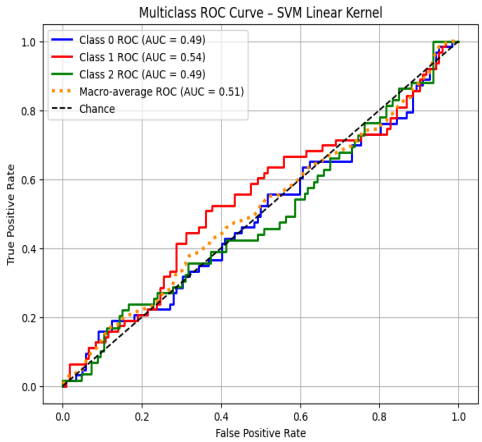

Multiclass Logistic Regression Confusion Matrix
35
26
5
16
43
14
2
17
27
Actual
Predicted
0
1
2
40
35
30
25
20
15
10
5

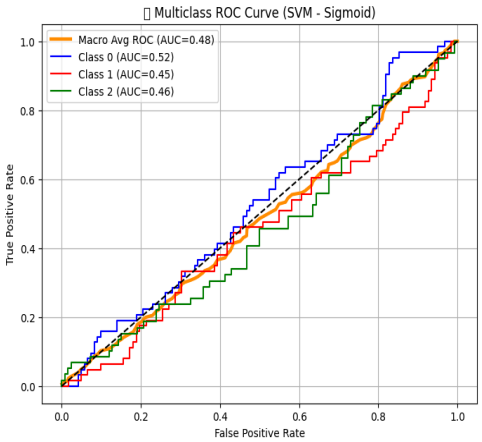

Decision Tree Confusion Matrix
45
18
3
18
37
18
9
23
14
Actual
Predicted
0
1
2
45
40
35
30
25
20
15
10
5

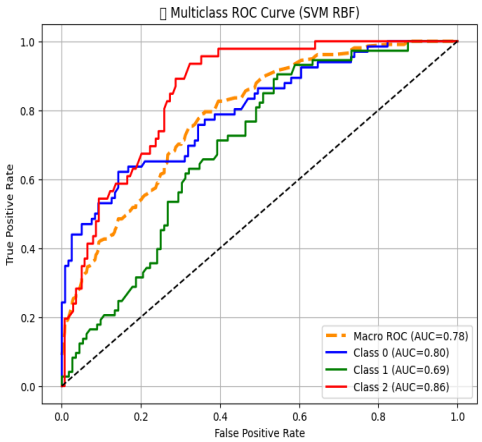

Gradient Boosting Confusion Matrix
40
21
5
12
44
17
5
19
22
Actual
Predicted
0
1
2
40
35
30
25
20
15
10
5

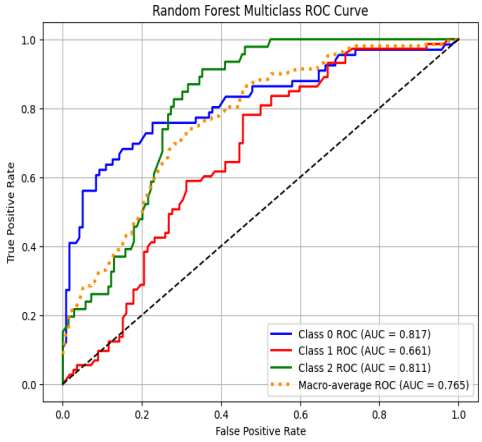

Gaussian Naive Bayes Confusion Matrix
47
13
6
35
29
9
5
18
23
Actual
Predicted
0
1
2
45
40
35
30
25
20
15
10
5

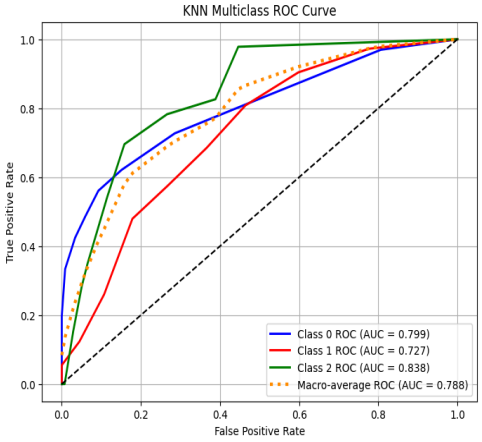
ROC Curve for classical ML Models (70 : 30)

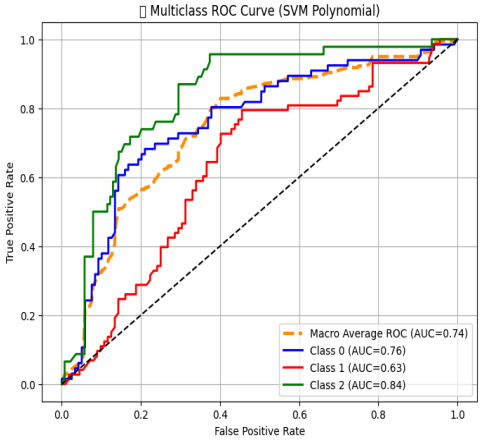


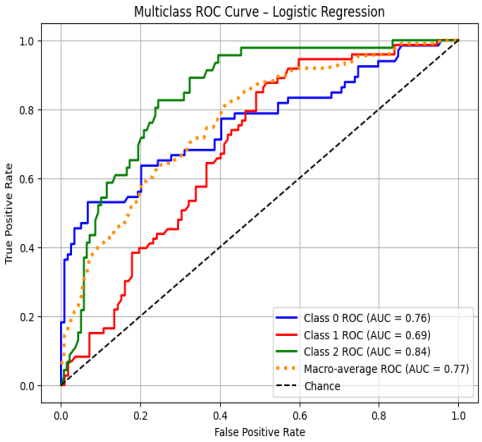


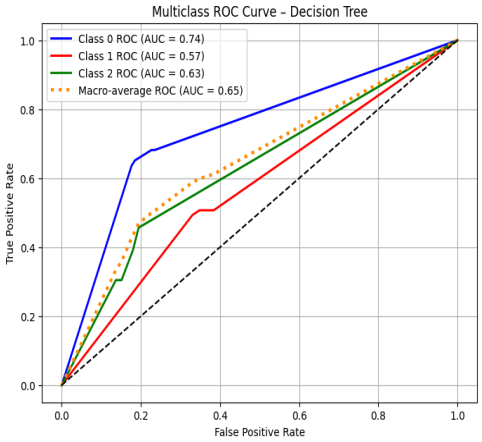


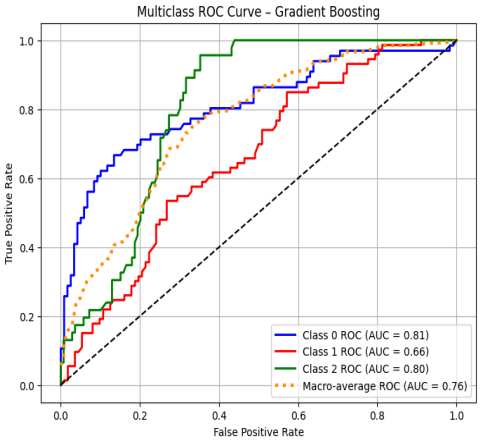







Multiclass ROC Curve – Logistic Regression
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.76)
Class 1 ROC (AUC = 0.69)
Class 2 ROC (AUC = 0.84)
Macro-average ROC (AUC = 0.77)
Chance


Multiclass ROC Curve – Decision Tree
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.74)
Class 1 ROC (AUC = 0.57)
Class 2 ROC (AUC = 0.63)
Macro-average ROC (AUC = 0.65)


Multiclass ROC Curve – Gradient Boosting
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.81)
Class 1 ROC (AUC = 0.66)
Class 2 ROC (AUC = 0.80)
Macro-average ROC (AUC = 0.76)

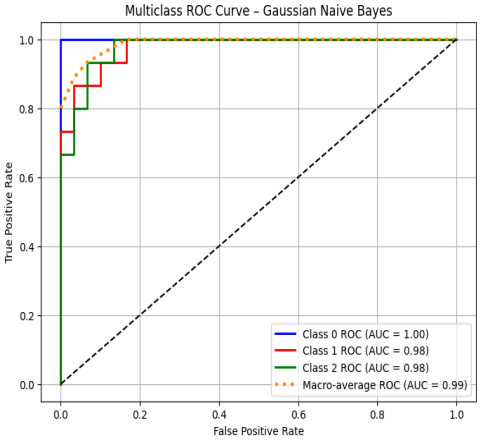
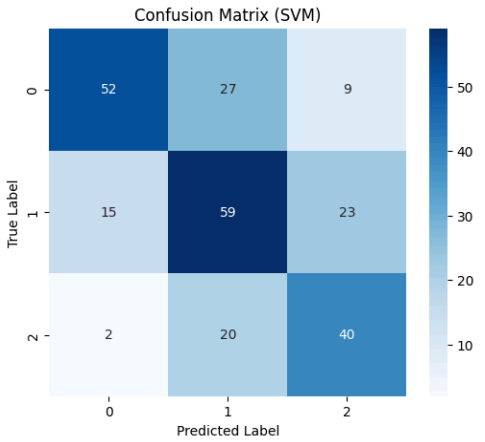
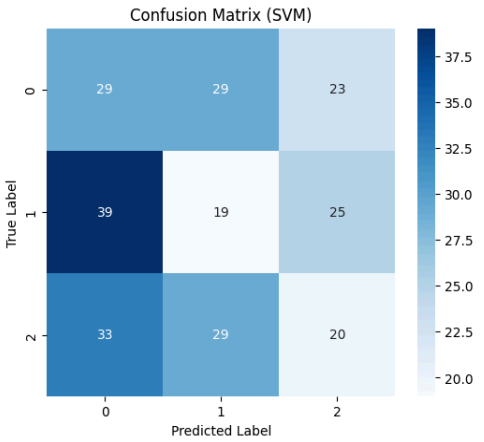

Multiclass ROC Curve – Gaussian Naive Bayes
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 1.00)
Class 1 ROC (AUC = 0.98)
Class 2 ROC (AUC = 0.98)
Macro-average ROC (AUC = 0.99)

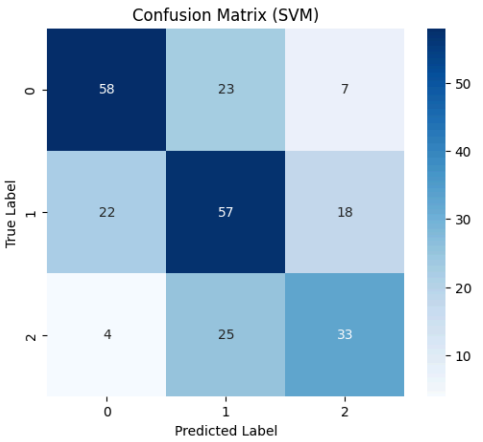
Table 4: Class-Wise Performance Comparison of Machine Learning Models Using Balanced Cut Tree (BCT) Features (60:40)

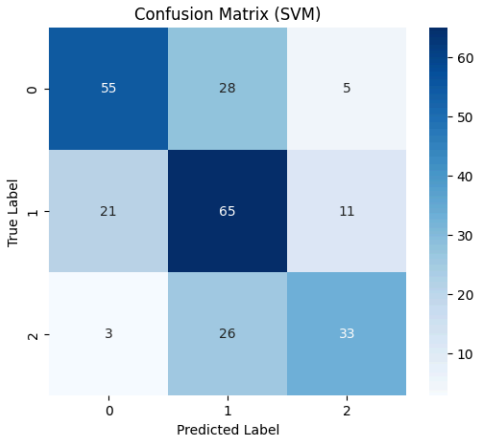
Sno	ML Model	Class	Accuracy	Precision	Recall	F1 Score
1	Support Vector Machine - Linear	Normal Osteoporosis Osteropenia	27%	29% 25% 29%	36% 23% 24%	32% 24% 27%
2	Support Vector Machine - Sigmoid	Normal Osteoporosis Osteropenia	64%	75% 58% 67%	59% 73% 60%	66% 65% 63%
3	Support Vector Machine - Polynomial	Normal Osteoporosis Osteropenia	63%	76% 58% 57%	62% 73% 48%	69% 65% 52%
4	Support Vector Machine - RBF	Normal Osteoporosis Osteropenia	67%	88% 59% 67%	57% 84% 56%	69% 69% 61%
5	Logistic Regression	Normal Osteoporosis Osteropenia	59%	65% 57% 55%	62% 61% 52%	64% 59% 53%
6	k-Nearest Neighbors (KNN)	Normal Osteoporosis Osteropenia	61%	63% 59% 64%	36% 61% 56%	64% 60% 60%
7	Gaussian Naive Bayes (GNB)	Normal Osteoporosis Osteropenia	56%	52% 54% 67%	62% 48% 60%	57% 51% 63%
8	Decision Tree Classifier	Normal Osteoporosis Osteropenia	53%	57% 52% 50%	70% 49% 37%	63% 51% 43%
9	Random Forest Classifier	Normal Osteoporosis Osteropenia	65%	82% 59% 62%	64% 75% 45%	72% 62% 58%
10	Gradient Boosting Classifier	Normal Osteoporosis Osteropenia	64%	76% 59% 60%	64% 70% 56%	69% 64% 58%

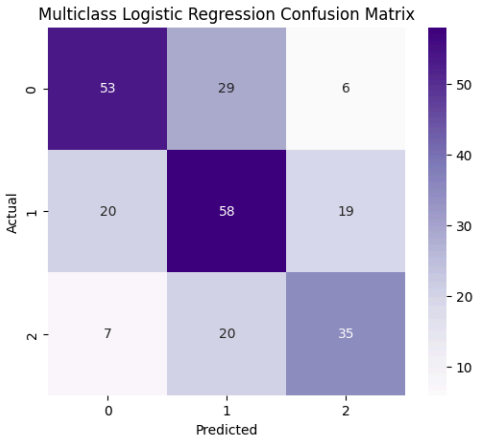
Confusion Matrix for Classical ML Models (60 : 40)

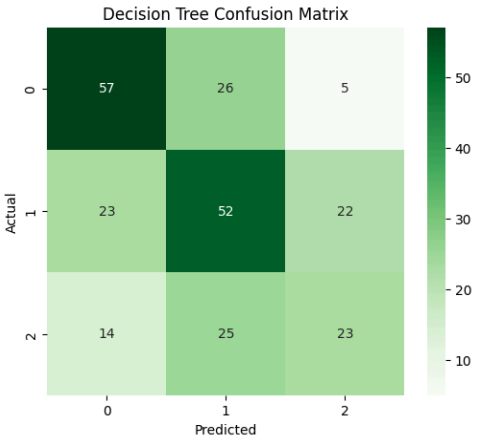


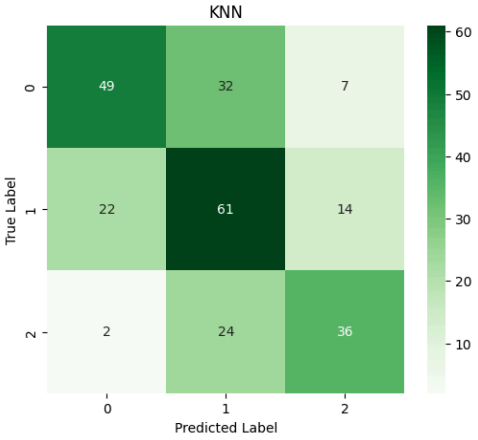


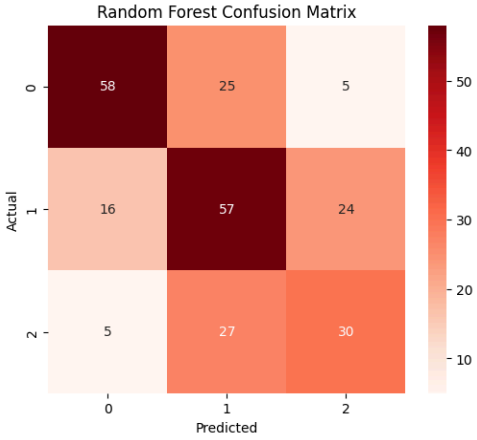


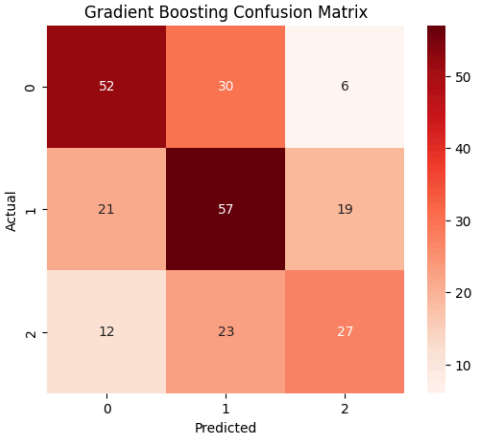


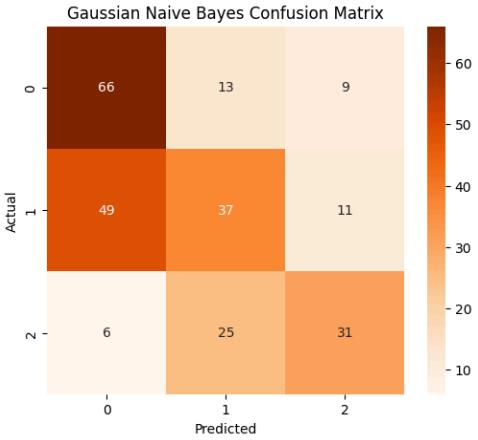




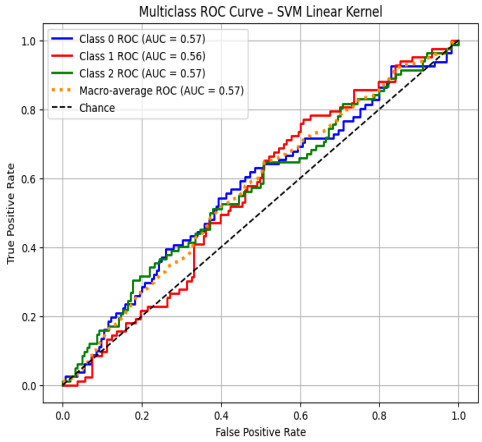

KNN
True Label
Predicted Label
49
32
7
22
61
14
2
24
36

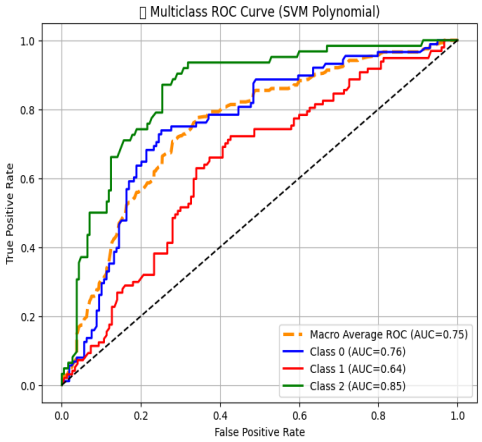

Random Forest Confusion Matrix
Actual
Predicted
58
25
5
16
57
24
5
27
30

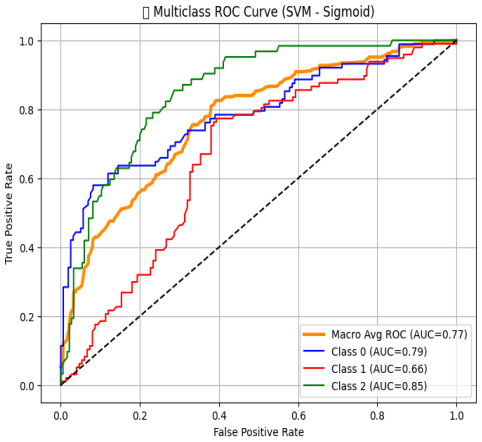

Gradient Boosting Confusion Matrix
Actual
Predicted
52
30
6
21
57
19
12
23
27

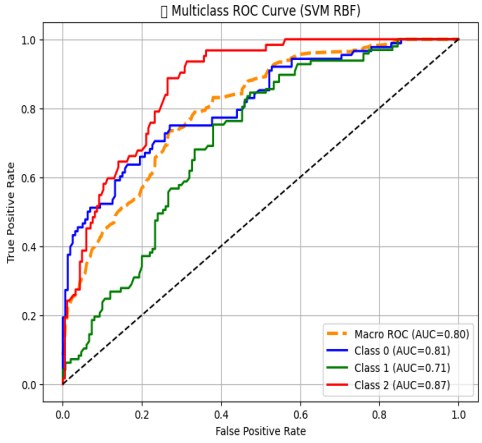

Gaussian Naive Bayes Confusion Matrix
Actual
Predicted
66
13
9
49
37
11
6
25
31

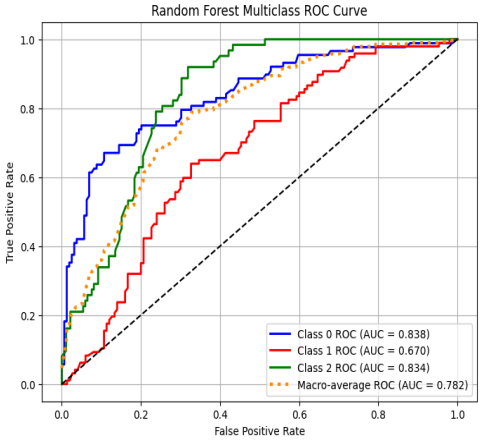
ROC Curve for classical ML Models (60 : 40)

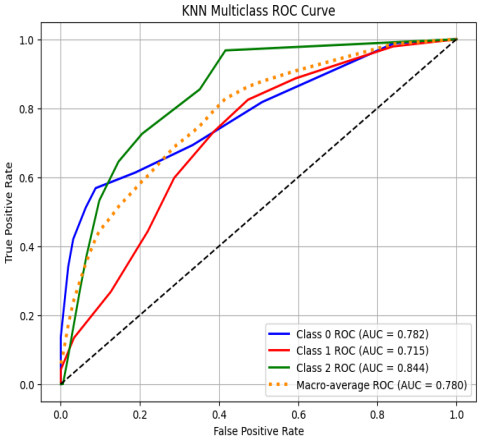


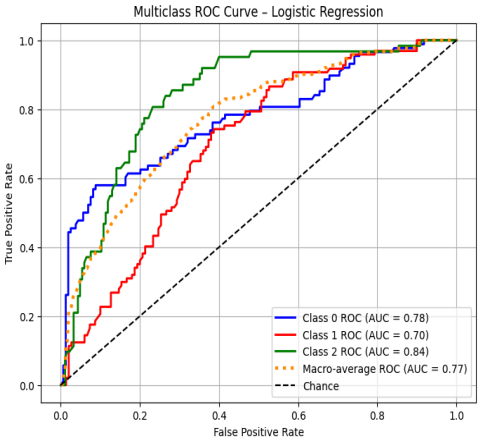


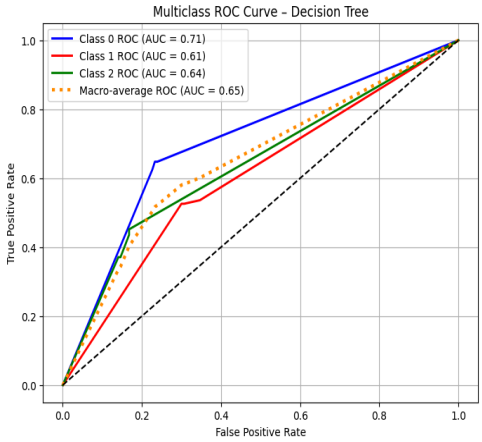


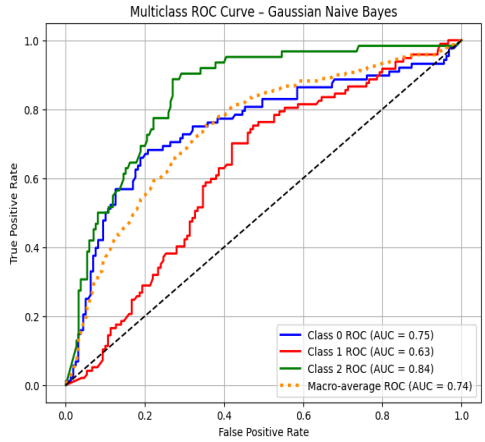


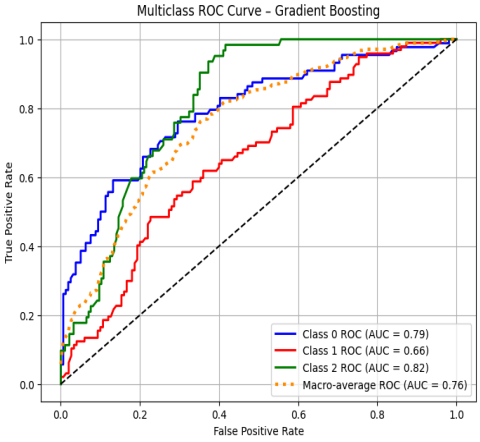





Multiclass ROC Curve – Logistic Regression
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.78)
Class 1 ROC (AUC = 0.70)
Class 2 ROC (AUC = 0.84)
Macro-average ROC (AUC = 0.77)
Chance


Multiclass ROC Curve – Decision Tree
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.71)
Class 1 ROC (AUC = 0.61)
Class 2 ROC (AUC = 0.64)
Macro-average ROC (AUC = 0.65)


Multiclass ROC Curve – Gaussian Naive Bayes
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.75)
Class 1 ROC (AUC = 0.63)
Class 2 ROC (AUC = 0.84)
Macro-average ROC (AUC = 0.74)


Multiclass ROC Curve – Gradient Boosting
True Positive Rate
False Positive Rate
Class 0 ROC (AUC = 0.79)
Class 1 ROC (AUC = 0.66)
Class 2 ROC (AUC = 0.82)
Macro-average ROC (AUC = 0.76)

Conclusion

The Balanced Cut Tree (BCT) approach provides an effective and computationally efficient framework for analyzing X-ray images for osteoporosis assessment. By partitioning each image into balanced rectangular regions and extracting stable statistical features from non-zero pixel areas, the method captures essential variations in trabecular bone texture. The experimental results across multiple machine-learning models and train–test ratios demonstrate that BCT-derived regional statistics support consistent multi-class classification of Normal, Osteopenia, and Osteoporosis cases.

Overall, the study shows that BCT partitioning is a reliable and interpretable feature-engineering strategy that performs well even with noisy backgrounds, uneven anatomy, and low-contrast X-rays. It offers a promising direction for automated, scalable bone-health assessment using standard radiographic imaging