

Contents

| | |
|---|----------|
| Enhancing Breast Cancer Detection with CART: A Technical Analysis of Feature Selection, Class Imbalance, and Model Performance | 3 |
| Abstract | 3 |
| Introduction | 3 |
| Intuition Behind Decision Tree Algorithms | 4 |
| Pseudocode for Decision Tree Algorithm | 5 |
| Detailed Description | 6 |
| Run Time Analysis of Decision Tree Algorithm | 6 |
| Methodology | 7 |
| Overview of the Data Collection and Preprocessing Steps | 9 |
| Theoretical Implications of Data Processing | 14 |
| Model Training | 14 |
| Feature Selection | 14 |
| Class Distribution Visualization | 16 |
| Data Splitting | 16 |
| Model Training and Evaluation | 17 |
| Visualizing the Decision Tree | 18 |
| Analyzing Feature Importance in the Decision Tree Model | 18 |
| Focusing on the Most Important Features for Modeling | 19 |
| Streamlining Model Training with Key Features | 20 |
| Decision Tree Visualization with Important Features | 21 |
| Focusing on Highly Correlated Features for Model Training | 22 |
| Modeling with Highly Correlated Features | 23 |
| Visualizing the Decision Tree with Correlated Features | 24 |
| Decision Tree Model Using Gini Criterion | 25 |
| Importance of the Gini Criterion | 26 |
| Visualizing the Decision Tree Using the Gini Criterion | 27 |
| Refining the Model with Gini Criterion on Correlated Features | 28 |
| Decision Tree Visualization: Gini Criterion with Correlated Features | 29 |
| Analyzing Feature Importance with Gini Criterion | 30 |
| Selection of Most Important Features for Enhanced Model Precision | 32 |
| Decision Tree Model Optimization with Gini and Important Features | 32 |
| Visualizing the Optimized Decision Tree with Important Features | 34 |
| Constructing a Dataset with Imbalanced Classes | 34 |
| Examining the Imbalanced Dataset Composition | 36 |
| Integrating Imbalanced Classes into a Unified Dataset | 36 |
| Visualizing the Class Distribution in the Imbalanced Dataset | 37 |
| Preparing and Analyzing the Imbalanced Dataset for Model Training | 38 |
| Decision Tree Model with Entropy on Imbalanced Data | 39 |
| Visualizing the Entropy-Based Decision Tree on Imbalanced Data | 41 |

| | |
|---|----|
| Decision Tree Model with Gini on Imbalanced Data | 42 |
| Visualizing the Gini-Based Decision Tree on Imbalanced Data . . | 42 |
| Evaluating the Gini Model's Performance on Imbalanced Data . | 43 |
| Result Evaluations | 46 |
| Overall Implications | 47 |
| Conclusion | 47 |
| Feedback Incorporation | 48 |
| References | 48 |

Enhancing Breast Cancer Detection with CART: A Technical Analysis of Feature Selection, Class Imbalance, and Model Performance

Sunami Dasgupta ^{*}. ^{***}, R. Carter Tillquist, Ph.D. (Research Advisor) ^{*} ^{**},

^{*}Department of Computer Science, California State University Chico, Chico CA,

[^{***}sdasgupta, ^{**}rtillquist]@csuchico.edu

Abstract

This paper presents an in-depth analysis of decision tree algorithms, focusing on their application in the diagnosis of breast cancer—a critical challenge in the medical field. Decision trees, known for their simplicity and interpretability, offer a promising approach to classifying complex datasets. Through a series of comparative studies, we explore the performance of decision tree models under various configurations, including different splitting criteria (entropy and Gini impurity), feature selection methodologies (correlated and important features), and conditions of class imbalance. Our evaluation reveals nuanced insights into the impact of these factors on model accuracy, interpretability, and computational efficiency.

We further enhance the analysis by providing a tight asymptotic run time analysis, shedding light on the computational aspects of decision tree construction and their implications for practical applications. The findings underscore the adaptability of decision trees to diverse data characteristics, their capability to handle imbalanced datasets effectively, and the critical role of feature selection in optimizing model performance.

This comprehensive exploration not only validates the effectiveness of decision tree algorithms in medical diagnostics but also opens avenues for future research aimed at refining their application across various domains. By offering a clear understanding of the strengths and limitations of decision trees, this paper contributes valuable insights to the ongoing dialogue on leveraging machine learning for meaningful advancements in healthcare.

Introduction

In the realm of medical diagnostics, the advent of machine learning models has ushered in a transformative era, offering the potential to significantly enhance the accuracy, efficiency, and predictability of disease diagnosis. Among various diseases, breast cancer remains one of the most prevalent and extensively researched conditions, given its impact on global health. Early and accurate diagnosis of breast cancer is critical for effective treatment planning and improving patient outcomes. In this context, the utilization of decision tree models presents

a compelling approach, noted for their interpretability and ease of implementation. This paper embarks on an in-depth exploration of decision tree models applied to the diagnosis of breast cancer, scrutinizing the influence of different model configurations and data handling strategies on predictive performance.

The investigation is structured around several focal points: comparing the efficacy of entropy and Gini criteria as splitting mechanisms, the impact of feature selection based on correlation and perceived importance, and the challenges posed by imbalanced datasets typical in medical diagnostic data. Through a series of model evaluations, this study aims to elucidate the nuances of decision tree algorithm performance across varied scenarios, highlighting the potential adjustments and considerations necessary to optimize model accuracy in the face of class imbalances and feature diversity.

By leveraging a comprehensive dataset of breast cancer diagnoses, this research not only seeks to refine predictive modeling techniques but also to contribute to the broader understanding of how machine learning can be effectively tailored to meet the specific demands of medical diagnostics. The goal is to bridge the gap between advanced computational methods and practical clinical applications, offering a pathway to more reliable, accessible, and interpretable diagnostic tools that can support healthcare professionals in making informed decisions and ultimately enhance patient care in the domain of breast cancer treatment.

Intuition Behind Decision Tree Algorithms

At a high level, decision tree algorithms operate by breaking down a dataset into smaller subsets based on specific criteria, while simultaneously developing an associated decision tree that incrementally makes decisions leading to a prediction. The essence of a decision tree lies in its hierarchical structure, consisting of nodes, branches, and leaves, where each node represents a “question” or “test” on an attribute, each branch denotes the outcome of the test, and each leaf node holds a class label.

The Decision-Making Process

1. **Starting at the Root:** The process begins at the root node of the tree, encompassing the entire dataset. The algorithm selects the best feature to split on based on a given metric (e.g., Gini impurity, entropy).
2. **Feature Splitting:** The chosen feature divides the dataset into smaller subsets. This split is determined in a way that groups together similar responses or outcomes, aiming to increase the homogeneity of the resultant subsets with respect to the target variable.
3. **Recursive Partitioning:** This splitting process is repeated recursively for each derived subset, with the procedure moving down the tree. At each step, the algorithm chooses the best feature to split on, considering only the data in the current subset.

4. **Termination Criteria:** The recursion ends when a termination criterion is met. Common criteria include a subset reaching a maximum specified depth, a minimum number of samples at a node, or when no further gains can be made.
5. **Prediction:** Once the tree is constructed, predictions are made by following the decisions in the tree from the root to a leaf. The path taken is determined by the feature values of the instance to be predicted, leading to a leaf node that provides the predicted class.

Key Concepts

- **Entropy and Gini Impurity:** These metrics guide the selection of the feature to split on at each step. Entropy measures the disorder in a set, and the decision tree aims to reduce entropy after a split (maximize information gain). Gini impurity measures the frequency at which any element of the dataset will be mislabeled when it is randomly labeled, with the algorithm aiming to minimize this measure.
- **Pruning:** To avoid overfitting, decision trees can be pruned by removing parts of the tree that do not provide additional power in classifying instances. Pruning can be pre-pruning (stopping the tree from fully developing) or post-pruning (removing sections of the tree after it has been fully developed).

The intuitive appeal of decision trees lies in their mimicry of human decision-making processes—asking yes/no questions or making decisions based on feature values—making them not only effective for classification and regression tasks but also highly interpretable and easy to visualize.

Pseudocode for Decision Tree Algorithm

```
Function BuildDecisionTree(D, F, T):
    Create node N
    if all samples in D belong to the same class C:
        return Node(label=C)
    if F is empty:
        return Node(label=most common class in D)
    best_feature <- selectFeatureWithBestSplit(D, F, T)
    for each value v of best_feature:
        subset_Dv <- {samples in D where best_feature = v}
        if subset_Dv is empty:
            child <- Node(label=most common class in D)
        else:
            child <- BuildDecisionTree(subset_Dv, F - {best_feature}, T)
        add child to N, labeled with v
    return N
```

Detailed Description

1. **Node Creation:** Initialize a new decision node `N`.
2. **Check for Homogeneous Class:**
 - If all samples in the dataset `D` belong to the same class `C`, create a leaf node with label `C` and terminate recursion.
3. **Check for Empty Feature Set:**
 - If the set of features `F` is empty (no features left for splitting), create a leaf node with the label as the most common class in `D`.
4. **Feature Selection:**
 - Call `selectFeatureWithBestSplit(D, F, T)` to choose the best feature from `F` based on a split quality criterion (like information gain or Gini impurity).
5. **Dataset Splitting:**
 - For each value `v` that the selected `best_feature` can take, partition the dataset `D` into subsets `subset_Dv` where each sample in `subset_Dv` has the value `v` for `best_feature`.
6. **Recursive Tree Construction:**
 - For each subset `subset_Dv`, check if it is empty. If so, create a leaf node with the most common class in `D`. Otherwise, recursively apply `BuildDecisionTree` on `subset_Dv`, excluding the `best_feature` from the set of features `F`.
 - Attach the resulting node as a child of `N`, labeled with the value `v`.
7. **Termination and Return:**
 - Once all values of `best_feature` have been processed, return the node `N`.

This pseudocode abstracts the recursive nature of decision tree construction, emphasizing the iterative process of selecting the optimal feature for splitting, partitioning the dataset based on that feature, and recursively building subtrees for each partition until the stopping criteria are met.

Run Time Analysis of Decision Tree Algorithm

The run time of a decision tree algorithm is influenced by several factors, including the size of the dataset, the number of features, and the depth of the tree. To provide a tight asymptotic analysis, let's denote:

- `n`: The number of samples in the dataset.
- `m`: The number of features.

- d : The depth of the tree, which can vary based on the stopping criteria used (e.g., maximum depth, minimum samples per node).

Asymptotic Analysis

1. **Feature Selection:** At each node of the tree, the algorithm evaluates each of the m features to find the best split. The evaluation of each feature involves calculating the split criterion (e.g., Gini impurity or entropy) across all n samples. Thus, the time complexity for feature selection at each node is $O(m \cdot n)$.
2. **Dataset Splitting:** After selecting the best feature for splitting, the dataset is partitioned into subsets corresponding to each unique value of the feature. In the worst case, this step involves scanning all n samples, yielding a time complexity of $O(n)$ for splitting the dataset at each node.
3. **Recursive Tree Construction:** The decision tree is built recursively, with the potential to split each node until a stopping criterion is met. In the worst case, if each split divides the dataset in half, the maximum depth d of the tree would be $O(\log n)$. However, in practice, splits may not be perfectly balanced, and stopping criteria such as maximum depth or minimum samples per node can lead to a shallower tree. For a balanced binary tree, the total number of nodes is $2^d - 1$.

Combining these factors, the worst-case time complexity for building the decision tree can be expressed as:

$$T(n) = O(m \cdot n \cdot 2^d)$$

Tight Asymptotic Bound For practical purposes, especially with depth-limiting or pre-pruning strategies, the depth d may be constrained to a constant or a function of n that grows slower than $O(\log n)$. If we denote d_{\max} as the maximum allowed depth or the effective depth reached due to stopping criteria, the runtime complexity becomes more tightly bound by:

$$T(n) = O(m \cdot n \cdot 2^{d_{\max}})$$

It's important to note that d_{\max} serves as a limiting factor, making the actual runtime more practical than the worst-case scenario might suggest. Additionally, optimizations in feature selection and splitting strategies (e.g., using approximate or heuristic methods) can further influence the runtime, often reducing the practical computational cost.

Methodology

The Classification and Regression Trees (CART) algorithm is a pivotal machine learning method used for both classification and regression tasks. It constructs binary trees from the training data, splitting the dataset into subsets, which then continue to split recursively until a stop criterion is met. The simplicity

of the decision trees makes them highly interpretable and applicable to various tasks, including medical diagnoses, financial forecasting, and more. This detailed explanation will cover the core components of the CART algorithm: the Gini impurity splitting criterion, tree construction, and pruning techniques.

The Classification and Regression Trees (CART) algorithm is a pivotal machine learning method used for both classification and regression tasks. It constructs binary trees from the training data, splitting the dataset into subsets, which then continue to split recursively until a stop criterion is met. The simplicity of the decision trees makes them highly interpretable and applicable to various tasks, including medical diagnoses, financial forecasting, and more. This detailed explanation will cover the core components of the CART algorithm: the Gini impurity splitting criterion, tree construction, and pruning techniques.

Gini impurity is a measure used by the CART algorithm to quantify the disorder or impurity in a set of elements. It is used to decide how to split the data at each step in the tree. The Gini Impurity, (I_G), for a set of items with (J) classes can be calculated as:

$$I_G(p) = 1 - \sum_{j=1}^J p_j^2$$

where (p_j) is the proportion of items labeled with class (j) in the set. The equation sums the squared proportion of each class in the set and subtracts the sum from 1. A Gini Impurity of 0 indicates that the set is perfectly homogeneous, meaning all elements belong to a single class. Higher Gini scores indicate greater disorder, with the maximum value being $1 - 1/J$, when items are evenly distributed across all classes.

The CART algorithm builds a binary tree from the training dataset. Starting with the root node, which contains the entire dataset, it iteratively splits the data based on the feature and split-point that result in the maximum reduction of Gini impurity. This process is represented mathematically as selecting the feature (f) and split point (s) that maximize the decrease in Gini impurity:

$$\Delta I_G(f, s) = I_G(\text{parent}) - \left(\frac{N_{\text{left}}}{N} I_G(\text{left}) + \frac{N_{\text{right}}}{N} I_G(\text{right}) \right)$$

where:

- $I_G(\text{parent})$ is the Gini impurity of the parent node before the split,
- N_{left} and N_{right} are the no. of samples in the left and right subsets created by the split,
- $I_G(\text{left})$ and $I_G(\text{right})$ are the Gini impurities of the left and right subsets,
- and N is the total number of samples at the parent node.

The algorithm continues to split each subset further, choosing the best split at each node in a greedy manner (i.e., making the locally optimal choice at each

step) until a stopping criterion is reached, such as a maximum depth of the tree or a minimum number of samples required to split a node.

To prevent overfitting, the CART algorithm employs pruning techniques to trim the fully grown tree. Pruning reduces the size of the tree by removing sections that provide little power to classify instances. One standard method is cost-complexity pruning, which involves finding the subtree that minimizes the cost-complexity measure:

$$R_{\alpha}(T) = R(T) + \alpha|\tilde{T}|$$

where:

- $R(T)$ is the total misclassification rate of the subtree T
- $|\tilde{T}|$ is the number of terminal nodes in T ,
- and α is the complexity parameter that balances the trade-off between the subtree's size and its fit to the training data.

The algorithm finds the optimal α through cross-validation and uses it to prune the tree by recursively replacing a subtree with a leaf node if the replacement results in a lower cost-complexity measure.

Overview of the Data Collection and Preprocessing Steps

```
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn import tree
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import random
from random import seed
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

1. Library Importation: The code begins by importing essential Python libraries:

- `numpy` and `pandas` for data manipulation,
- `matplotlib.pyplot`, `seaborn`, and `plotly.express` for data visualization,

- and `sklearn` for machine learning tasks.

These libraries are the backbone of data analysis, providing tools for handling, analyzing, and visualizing data, as well as implementing machine learning algorithms.

```
data = pd.read_csv("../input/breast-cancer-wisconsin-data/data.csv")
```

2. Dataset Loading: The dataset is loaded into a Pandas DataFrame from a CSV file. This step converts the structured data file into a DataFrame object, enabling the application of Pandas’ powerful data manipulation methods.

```
data.head()
data.info()
```

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean |
|---|----------|-----------|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|---------------------|
| 0 | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 |
| 1 | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 |
| 2 | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 |
| 3 | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 |
| 4 | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 |

Figure 1: image

| texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concavity_worst | concave points_worst | symmetry_worst | fractal_dimension_worst | Unnamed 32 |
|---------------|-----------------|------------|------------------|-------------------|-----------------|----------------------|----------------|-------------------------|------------|
| 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 | 0.11890 | NaN |
| 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1866 | 0.2416 | 0.1860 | 0.2750 | 0.08902 | NaN |
| 25.53 | 152.50 | 1709.0 | 0.1444 | 0.4245 | 0.4504 | 0.2430 | 0.3613 | 0.08758 | NaN |
| 26.50 | 98.87 | 567.7 | 0.2098 | 0.8663 | 0.6869 | 0.2575 | 0.6638 | 0.17300 | NaN |
| 16.67 | 152.20 | 1575.0 | 0.1374 | 0.2050 | 0.4000 | 0.1625 | 0.2364 | 0.07678 | NaN |

Figure 2: image

3. Initial Data Inspection: The `head()` method provides a quick glance at the dataset’s first few rows, offering insights into the types of data (e.g., numerical, categorical) and potential features of interest. The `info()` method is then used to summarize the dataset, including the total number of entries, the presence of null values, and the data type of each column. Such preliminary inspections are crucial for identifying issues like missing data or incorrect data types that could affect further analysis.

```
data = data.drop(['Unnamed: 32', 'id'],axis = 1)
```

4. Data Cleaning: The script identifies and removes unnecessary columns, such as “Unnamed: 32” and “id,” which do not contribute to the analysis. This step simplifies the dataset, focusing attention on relevant features.

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     569 non-null    int64
1   diagnosis                             569 non-null    object
2   radius_mean                           569 non-null    float64
3   texture_mean                           569 non-null    float64
4   perimeter_mean                         569 non-null    float64
5   area_mean                             569 non-null    float64
6   smoothness_mean                       569 non-null    float64
7   compactness_mean                      569 non-null    float64
8   concavity_mean                        569 non-null    float64
9   concave points_mean                   569 non-null    float64
10  symmetry_mean                         569 non-null    float64
11  fractal_dimension_mean                569 non-null    float64
12  radius_se                             569 non-null    float64
13  texture_se                             569 non-null    float64
14  perimeter_se                           569 non-null    float64
15  area_se                               569 non-null    float64
16  smoothness_se                         569 non-null    float64
17  compactness_se                        569 non-null    float64
18  concavity_se                          569 non-null    float64
19  concave points_se                     569 non-null    float64
20  symmetry_se                           569 non-null    float64
21  fractal_dimension_se                  569 non-null    float64
22  radius_worst                          569 non-null    float64
23  texture_worst                         569 non-null    float64
24  perimeter_worst                       569 non-null    float64
25  area_worst                            569 non-null    float64
26  smoothness_worst                      569 non-null    float64
27  compactness_worst                     569 non-null    float64
28  concavity_worst                       569 non-null    float64
29  concave points_worst                  569 non-null    float64
30  symmetry_worst                        569 non-null    float64
31  fractal_dimension_worst               569 non-null    float64
32  Unnamed: 32                           0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB

```

Figure 3: image
11

| |
|----------------|
| Unnamed: 32 |
| NaN |
| NaN |
| NaN |
| NaN |
| NaN |

Figure 4: image

```
print("Dataset size:",data.shape)
```

```
Dataset size: (569, 31)
```

5. Dataset Shape: The size of the dataset is displayed, providing an overview of its scale, which is important for understanding the computational resources that may be needed for processing and the potential for statistical analysis.

```
data.isnull().values.any()
```

```
False
```

6. Null Value Check: Checking for null values is crucial in data preprocessing to ensure the integrity of the dataset. Null values can distort predictive modeling and statistical analyses, necessitating either imputation or removal.

```
data_copied = data.copy()
data_copied['diagnosis'] = data_copied['diagnosis'].replace(("M"),1)
data_copied['diagnosis'] = data_copied['diagnosis'].replace(("B"),0)
```

7. Feature Encoding: The 'diagnosis' column, representing the target variable, is encoded from categorical ('M' for malignant, 'B' for benign) to numerical values (1 for malignant, 0 for benign). This step is essential for machine learning algorithms that require numerical input.

```

matrix = np.triu(data_copied.corr())
sns.set_style("white")
f,ax=plt.subplots(figsize = (16,16))
sns.heatmap(data_copied.corr(),annot= True,fmt = ".2f",ax=ax,
            vmin = -1,
            vmax = 1, mask = matrix,cmap = "coolwarm",
            linewidth = 0.2,linewidth = "white")
plt.xticks(rotation=70)
plt.yticks(rotation=0)
plt.title('Correlation Map', size = 14)
plt.show()

```

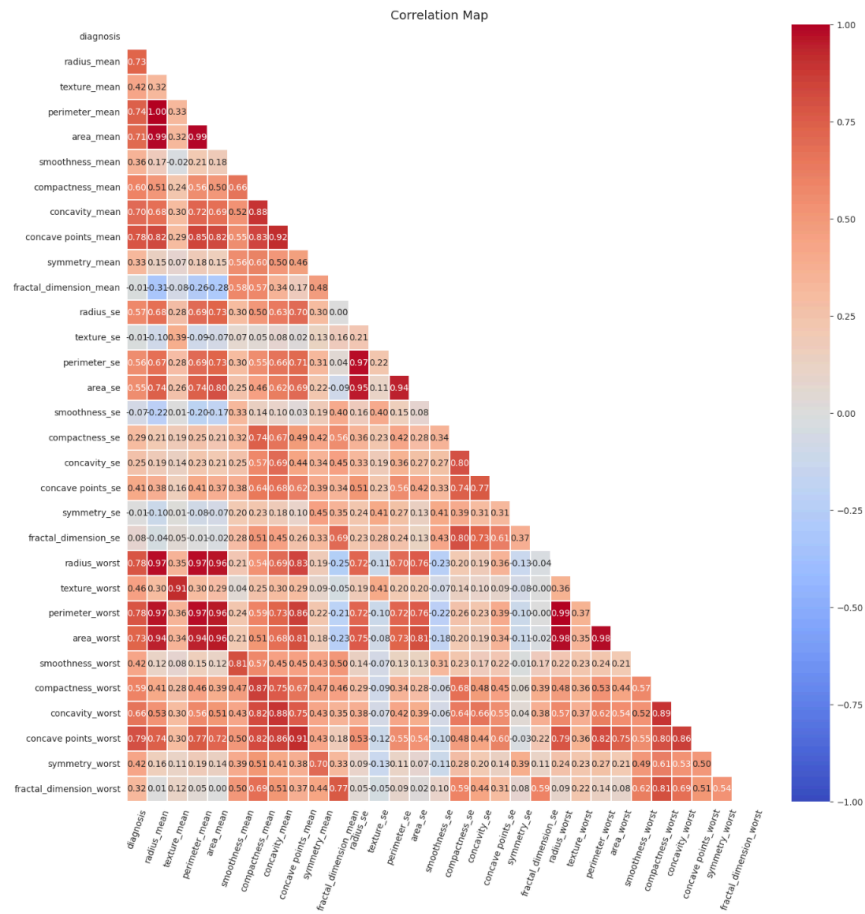


Figure 5: image

8. Correlation Analysis: A correlation matrix is generated and visualized to explore the relationships between features. This analysis is instrumental in identifying features that strongly correlate with the target variable, providing insights into potential predictors for the machine learning model. The use of a triangular mask (`np.triu()`) on the heatmap simplifies the visualization by removing redundant information.

Theoretical Implications of Data Processing

The preprocessing steps outlined above are grounded in both practical necessity and theoretical considerations. Removing irrelevant features and handling missing values cleans the dataset, reducing noise and potential biases that could affect model performance. Encoding categorical variables into numerical formats is a prerequisite for most machine learning algorithms, which typically require numerical input. Finally, correlation analysis not only aids in feature selection by identifying promising predictors but also helps in detecting multicollinearity, where highly correlated predictors can destabilize some models.

Model Training

The process of training the model involves several steps, from feature selection to model evaluation. We use the Breast Cancer Wisconsin dataset, focusing on features most correlated with the diagnosis to predict cancer presence accurately.

Feature Selection

We start by identifying the features most correlated with the diagnosis:

```
most_correlated_features = ['radius_mean', 'perimeter_mean',  
                             'area_mean', 'concavity_mean',  
                             'concave points_mean', 'radius_worst',  
                             'perimeter_worst', 'area_worst',  
                             'concave points_worst']
```

For each of these features, we visualize their distribution across diagnoses using box plots:

```
for i in range(0, len(most_correlated_features)):  
    fig = px.box(data, x="diagnosis",  
                 y=most_correlated_features[i], color="diagnosis",  
                 width=750,height=450)  
    fig.show()
```

These plots reveal that cancerous samples typically exhibit higher values across these features compared to healthy samples, indicating their significance in distinguishing between diagnoses.

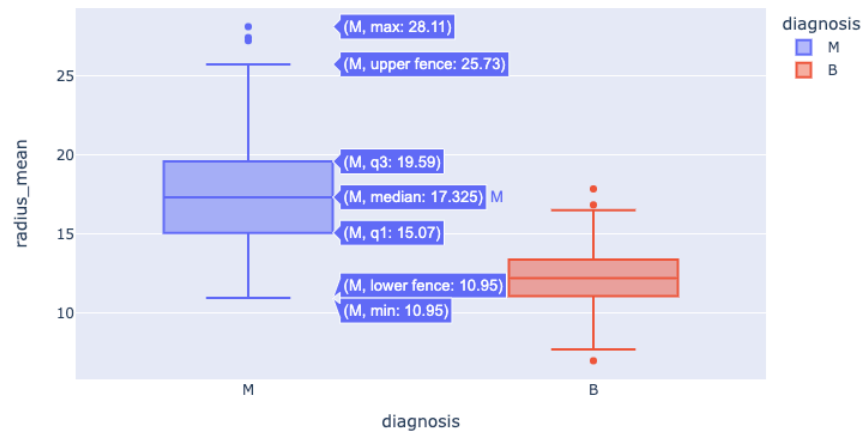


Figure 6: image

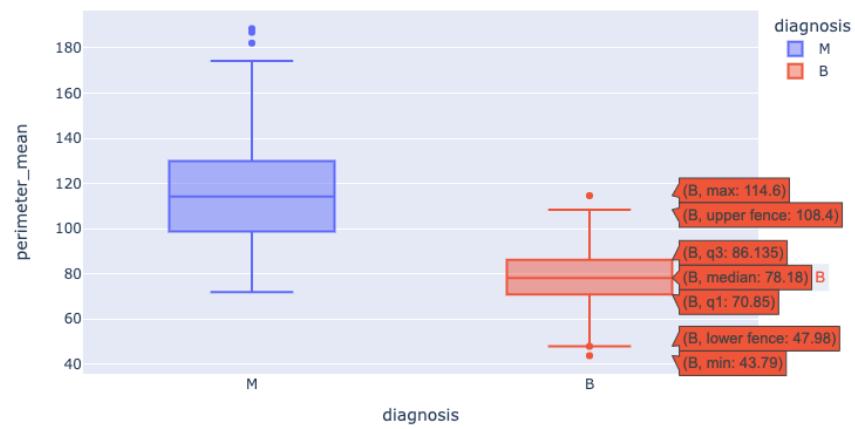


Figure 7: image

Class Distribution Visualization

Understanding the class distribution is crucial for assessing the balance in our dataset:

```
data.diagnosis.value_counts(sort=False).plot(kind='bar')
plt.title('Class distribution', fontsize = 15)
plt.xlabel('Classes', fontsize = 15)
plt.xticks(rotation=360)
plt.ylabel('Quantity', fontsize = 15)
```

```
[20... Text(0, 0.5, 'Quantity')
```

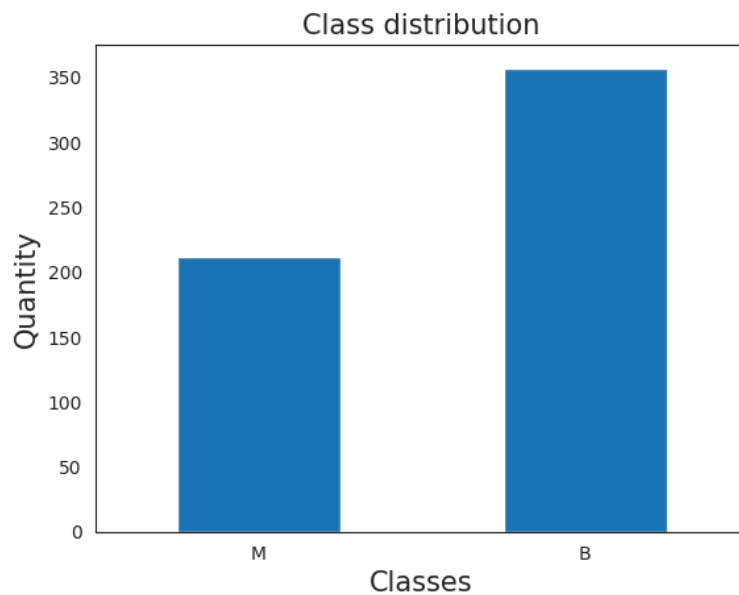


Figure 8: image

Data Splitting

We split the dataset into training and test sets to prepare for model training:

```
X = data.drop('diagnosis', axis=1)
Y = data['diagnosis']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state = 123)
```

Train dataset shape: (455, 30)

Test dataset shape: (114, 30)

Model Training and Evaluation

Using Entropy Criterion We first train a Decision Tree Classifier using the entropy criterion:

```
model = tree.DecisionTreeClassifier(criterion='entropy',
splitter='best', random_state=123)
model.fit(X_train, Y_train)
predictions = model.predict(X_test)
```

We then evaluate the model's accuracy:

```
print('Training set accuracy: {:.4f}'.format
(model.score(X_train, Y_train)*100))
print('Test set accuracy: {:.4f}'.format
(model.score(X_test, Y_test)*100))
```

Training set accuracy: 100.0000

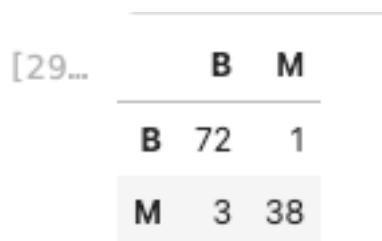
Test set accuracy: 96.4912

The model achieves 100% training set accuracy and approximately 96.49% test set accuracy, indicating strong performance but also potential overfitting.

Confusion Matrix To further assess the model, we construct a confusion matrix:

```
def conf_matrix(predictions, Y_test):
    conf_matrix = metrics.confusion_matrix(Y_test, predictions)
    return pd.DataFrame(conf_matrix,
        index=['B', 'M'], columns=['B', 'M'])
```

```
conf_matrix(predictions, Y_test)
```



| | B | M |
|---|----|----|
| B | 72 | 1 |
| M | 3 | 38 |

Figure 9: image

This matrix provides detailed insights into the model's predictive capabilities across different classes.

Visualizing the Decision Tree

This code visually represents the decision tree model, emphasizing its decision-making process. It uses Matplotlib for figure sizing and Scikit-learn's `plot_tree` function to render the tree. Each node in the visualized tree corresponds to a feature-based decision point, distinguishing between malignant ('M') and benign ('B') diagnoses based on the dataset's features. The tree is color-filled for easy distinction between classes, and it intentionally omits impurity metrics for a cleaner look. This visualization aids in understanding how the model predicts outcomes and the significance of each feature in those predictions.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model,
                    feature_names=list(X.columns.values),
                    class_names=['M', 'B'], filled=True, impurity = False)
```

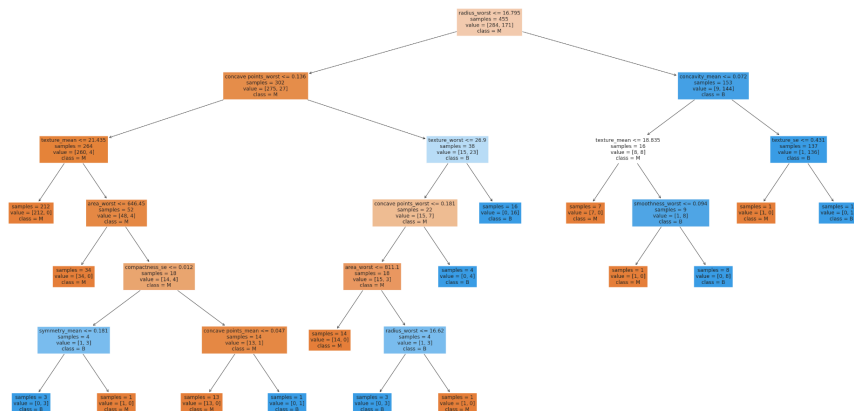


Figure 10: image

Analyzing Feature Importance in the Decision Tree Model

This code is designed to quantify and visualize the importance of each feature used by the decision tree model in making predictions. The process starts by extracting feature importance values from the trained model, which indicate the relative importance of each feature in predicting the target variable. These importance values are then sorted to ensure a coherent visual presentation. A horizontal bar chart is created to display these importance values, with the most important features (those contributing most to the model's decision-making process) appearing at the top. The chart's axes are labeled to clarify that the bars represent the importance of the features, and each bar is matched with the feature's name for easy identification. This visualization provides insightful details about which features are most influential in the model's predictions, helping to guide future data collection and model refinement efforts.

```

tree_importances = model.feature_importances_
indices = np.argsort(tree_importances)
fig, ax = plt.subplots(figsize = (7,7))
ax.barh(range(len(tree_importances)), tree_importances[indices])
ax.set_yticks(range(len(tree_importances)))
ax.set_title('Feature importances', fontsize=15)
ax.set_xlabel('Importance', fontsize=15)
ax.set_ylabel('Features', fontsize=15)
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])

```

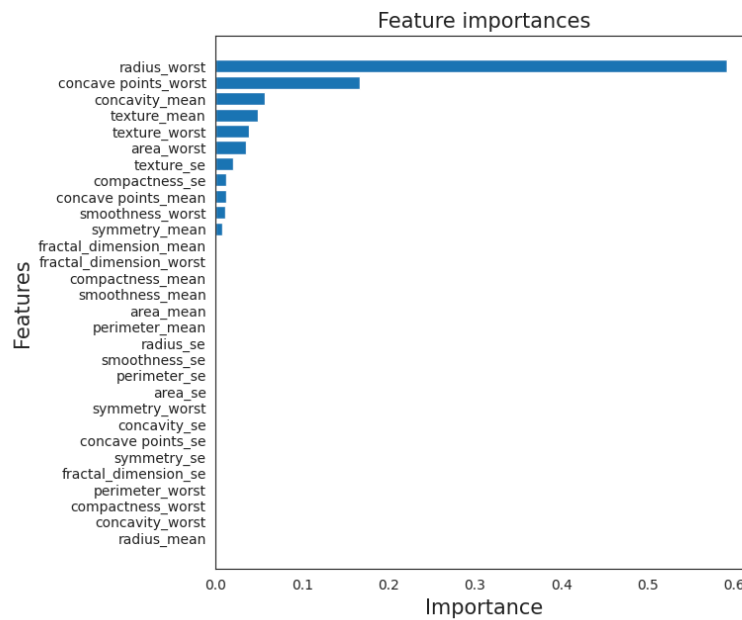


Figure 11: image

Focusing on the Most Important Features for Modeling

This section highlights a refined approach to modeling by concentrating solely on the most crucial features identified through previous analysis. By selecting features that have shown significant importance in predicting the outcome, such as 'radius_worst', 'concave points_worst', and 'concavity_mean' among others, the model aims for a more focused and potentially more interpretable framework. This selection process involves creating a new dataset, **data_most_important**, which contains only the selected features along with the diagnosis. Displaying the head of this newly formed dataset provides a snapshot of how these key features vary across the first few instances, setting the stage for a modeling process that leverages the most informative aspects of the data. Such a

strategy can lead to more efficient models by reducing complexity and focusing computational resources on analyzing the features most likely to influence the prediction.

```
data_most_important = data.loc[:,['radius_worst','diagnosis',
'concave points_worst','concavity_mean',
'texture_mean', 'texture_worst',
'area_worst', 'texture_se','compactness_se',
'concave points_mean','smoothness_worst','symmetry_mean']]
data_most_important.head()
```

```
[34.]
```

| | radius_worst | diagnosis | concave points_worst | concavity_mean | texture_mean | texture_worst | area_worst | texture_se | compactness_se | concave points_mean | smoothness_worst | symmetry_mean |
|---|--------------|-----------|----------------------|----------------|--------------|---------------|------------|------------|----------------|---------------------|------------------|---------------|
| 0 | 25.38 | M | 0.2654 | 0.3001 | 10.38 | 17.33 | 2019.0 | 0.9053 | 0.04904 | 0.14710 | 0.1622 | 0.2419 |
| 1 | 24.99 | M | 0.1860 | 0.0869 | 17.77 | 23.41 | 1966.0 | 0.7339 | 0.01308 | 0.07017 | 0.1238 | 0.1812 |
| 2 | 23.57 | M | 0.2430 | 0.1974 | 21.25 | 25.53 | 1709.0 | 0.7869 | 0.04006 | 0.12790 | 0.1444 | 0.2069 |
| 3 | 14.91 | M | 0.2575 | 0.2414 | 20.38 | 26.50 | 567.7 | 1.1560 | 0.07458 | 0.10520 | 0.2098 | 0.2597 |
| 4 | 22.54 | M | 0.1625 | 0.1980 | 14.34 | 16.67 | 1575.0 | 0.7813 | 0.02461 | 0.10430 | 0.1374 | 0.1809 |

Figure 12: image

Streamlining Model Training with Key Features

After identifying the most impactful features in predicting breast cancer diagnoses, this approach refines the modeling process by exclusively utilizing these selected features. The dataset `data_most_important` encompasses only the crucial features alongside the diagnosis labels. By determining its shape, we confirm the dataset includes 569 entries and 12 features, ensuring all relevant information is retained for model training.

```
data_most_important.shape
```

Subsequently, the dataset is divided into features (`X_most_important`) and the target variable (`Y_most_important`), followed by splitting into training and testing sets. This partitioning is vital for evaluating the model's performance on unseen data, ensuring the robustness of the predictive model.

```
X_most_important = data_most_important.drop('diagnosis', axis=1)
Y_most_important = data_most_important['diagnosis']
X_train_important, X_test_important,
Y_train_important, Y_test_important = train_test_split(X_most_important,
Y_most_important, test_size=0.2, random_state=123)
```

The dataset's division reveals a training set with 455 instances and a test set with 114 instances, highlighting the data's readiness for the modeling phase.

```
print("Train dataset shape: ", X_train_important.shape)
print("Test dataset shape: ", X_test_important.shape)
```

A Decision Tree Classifier is then trained using the entropy criterion, focusing solely on the most important features. This tailored model is expected to yield

high accuracy by concentrating on the most informative predictors of breast cancer.

```
seed(48)
model_imp = tree.DecisionTreeClassifier(criterion='entropy',
splitter='best', random_state=123)
model_imp.fit(X_train_important, Y_train_important)
predictions_imp = model_imp.predict(X_test_important)
```

The model's performance is remarkable, achieving 100% accuracy on the training set and 95.614% on the test set. These results underline the efficacy of focusing on the most significant features in enhancing model accuracy and interpretability.

```
print('Training set accuracy: {:.4f}'.format
(model_imp.score(X_train_important, Y_train_important)*100))
print('Test set accuracy: {:.4f}'.format
(model_imp.score(X_test_important, Y_test_important)*100))
```

A confusion matrix is generated to provide detailed insights into the model's predictive performance, further illustrating its capabilities in distinguishing between benign and malignant diagnoses accurately.

```
conf_matrix(predictions_imp, Y_test_important)
```

| | B | M |
|---|----|----|
| B | 72 | 1 |
| M | 4 | 37 |

Figure 13: image

Decision Tree Visualization with Important Features

Focusing on the most critical features offers a streamlined and insightful approach to understanding the decision-making process of a decision tree model. By training the model exclusively on features deemed most influential for predicting breast cancer diagnoses, we aim to enhance interpretability without compromising the model's predictive power. Visualizing this specialized decision tree elucidates how it leverages these key attributes to differentiate between benign and malignant cases, providing valuable insights into the model's internal logic and decision pathways.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_imp,
feature_names=list(X_train_important.columns.values),
class_names=['M', 'B'], filled=True, impurity=False)
```

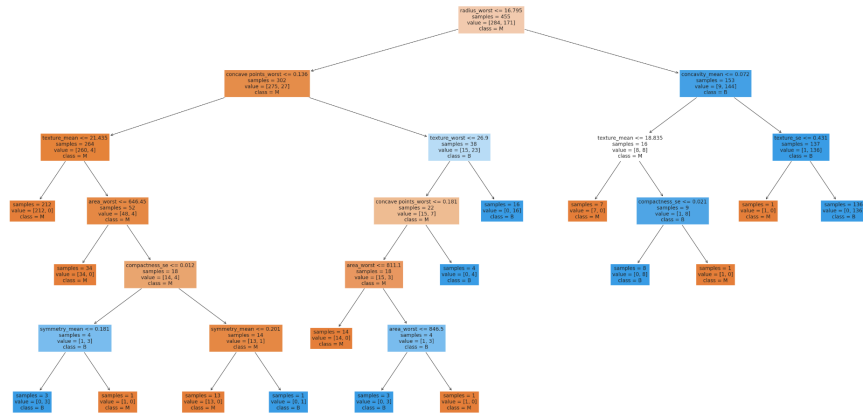


Figure 14: image

A decision tree visualization is generated, emphasizing the model trained on the selected important features. The figure's substantial size ensures clarity and readability, allowing each node and branch to be distinctly observed. This visualization not only serves as a tool for model interpretation but also highlights the predictive significance of the chosen features, reinforcing the importance of thoughtful feature selection in model development.

Focusing on Highly Correlated Features for Model Training

In an effort to refine the predictive modeling of breast cancer diagnoses, this approach concentrates on features that exhibit the highest correlation with the target variable. By identifying and selecting only the most correlated features, the model aims to capture the most relevant information that contributes to the accuracy of predictions. This method not only has the potential to improve model performance but also enhances the interpretability by narrowing down the feature space to those variables that are most indicative of the outcome.

```
data_most_correlated = data.loc[:,['radius_mean',
'perimeter_mean', 'area_mean', 'concavity_mean',
'concave points_mean', 'radius_worst',
'perimeter_worst', 'area_worst', 'concave points_worst']]
data_most_correlated.head()
```

This code creates a new DataFrame, `data_most_correlated`, by selecting only

| | radius_mean | diagnosis | perimeter_mean | area_mean | concavity_mean | concave points_mean | radius_worst | perimeter_worst | area_worst | concave points_worst |
|---|-------------|-----------|----------------|-----------|----------------|---------------------|--------------|-----------------|------------|----------------------|
| 0 | 17.99 | M | 122.80 | 1001.0 | 0.3001 | 0.14710 | 25.38 | 184.60 | 2019.0 | 0.2654 |
| 1 | 20.57 | M | 132.90 | 1326.0 | 0.0869 | 0.07017 | 24.99 | 158.80 | 1956.0 | 0.1860 |
| 2 | 19.69 | M | 130.00 | 1203.0 | 0.1974 | 0.12790 | 23.57 | 152.50 | 1709.0 | 0.2430 |
| 3 | 11.42 | M | 77.58 | 386.1 | 0.2414 | 0.10520 | 14.91 | 98.87 | 567.7 | 0.2575 |
| 4 | 20.29 | M | 135.10 | 1297.0 | 0.1980 | 0.10430 | 22.54 | 152.20 | 1575.0 | 0.1625 |

Figure 15: image

the features with the highest correlation to the diagnosis, along with the diagnosis itself. By using the `.head()` method, we can quickly preview the first few rows of this newly focused dataset, ensuring that the selection process has successfully narrowed down the feature set to those deemed most impactful. Concentrating on these correlated features allows for a targeted analysis, potentially leading to more accurate and interpretable models that are specifically tailored to the nuances of breast cancer diagnosis.

Modeling with Highly Correlated Features

Optimizing model performance often involves honing in on features that exhibit the strongest correlations with the target variable. In this approach, the dataset is filtered to include only the most correlated features with the diagnosis of breast cancer, aiming to build a predictive model that is both efficient and interpretable. The selection is based on an initial analysis identifying features with the highest correlation to the diagnosis outcome.

```
data_most_correlated.shape
```

The resulting dataset, `data_most_correlated`, maintains 569 instances across 10 features, focusing the analysis on variables most relevant to predicting the diagnosis. The dataset is then partitioned into feature sets (`X_most_correlated`) and target labels (`Y_most_correlated`), which are subsequently split into training and testing datasets. This step is crucial for validating the model's performance on unseen data.

```
X_most_correlated = data_most_correlated.drop('diagnosis', axis=1)
Y_most_correlated = data_most_correlated['diagnosis']
X_train_corr, X_test_corr,
Y_train_corr, Y_test_corr = train_test_split(X_most_correlated,
Y_most_correlated, test_size=0.2, random_state=123)
```

The training set shapes confirm the distribution of data points used for training the model versus those reserved for evaluation. Ensuring a balanced split helps in assessing the model's generalization capabilities accurately.

```
print("Train dataset shape: ", X_train_corr.shape)
print("Test dataset shape: ", X_test_corr.shape)
```

A decision tree classifier is then trained on the correlated features, using entropy

as the criterion for splitting. The model's accuracy on both the training and test datasets provides a measure of its effectiveness in leveraging the selected features for prediction.

```
seed(48)
model_corr = tree.DecisionTreeClassifier(criterion='entropy'
, splitter='best', random_state=123)
model_corr.fit(X_train_corr, Y_train_corr)
predictions_corr = model_corr.predict(X_test_corr)
```

Evaluating the model's accuracy reveals its strong performance on the training set and commendable accuracy on the test set, indicating a promising approach to predicting breast cancer diagnoses by focusing on the most correlated features.

```
print('Training set accuracy: {:.4f}'.format
(model_corr.score(X_train_corr, Y_train_corr)*100))
print('Test set accuracy: {:.4f}'.format
(model_corr.score(X_test_corr, Y_test_corr)*100))
```

The confusion matrix further elucidates the model's predictive capability, detailing the accuracy of predictions across different classes and providing insights into the model's strengths and areas for improvement.

```
conf_matrix(predictions_corr, Y_test)
```

| | B | M |
|---|----|----|
| B | 69 | 4 |
| M | 2 | 39 |

Figure 16: image

This targeted approach underscores the value of feature selection in developing predictive models, particularly when aiming for a balance between performance and interpretability.

Visualizing the Decision Tree with Correlated Features

To encapsulate the essence of the decision-making process within a decision tree model trained on the most correlated features, a visual representation is constructed. This visualization focuses on illustrating how the decision tree utilizes these selected features to discern between malignant ('M') and benign ('B') breast cancer diagnoses. By honing in on features with the highest correlation to the target outcome, the model aims to provide precise and interpretable

predictions.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_corr,
feature_names=list(X_most_correlated.columns.values),
class_names=['M','B'],
filled=True, impurity=False)
```

This code crafts a detailed graphical depiction of the trained decision tree, utilizing a generous figure size to accommodate the complexity and breadth of the tree's structure. Through the `plot_tree` function, each node's decision criteria based on the correlated features are displayed, alongside the classification outcomes at the leaves. The absence of impurity measures in the visualization allows for a cleaner focus on the tree's branching logic and decision points. Color-coding the nodes by class further enhances interpretability, making it straightforward to follow the path from root to leaf for any given decision process within the model.

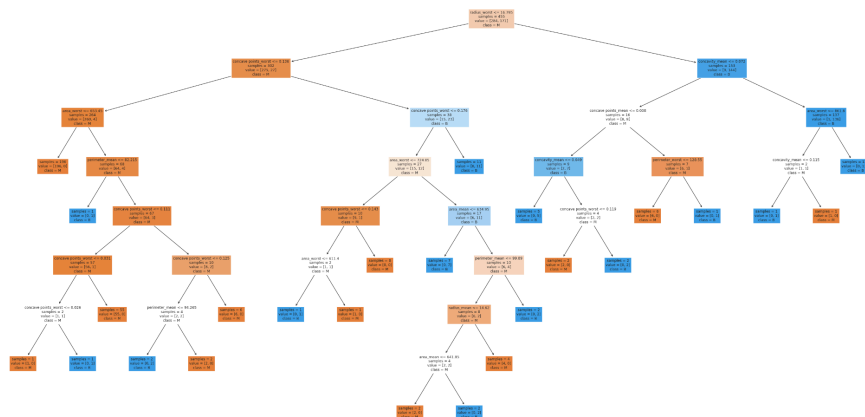


Figure 17: image

By visualizing the model in this manner, we gain invaluable insights into the predictive dynamics of the decision tree, showcasing the direct impact of the most correlated features on the model's ability to accurately classify breast cancer diagnoses.

Decision Tree Model Using Gini Criterion

In the pursuit of optimizing the predictive model for breast cancer diagnosis, employing the Gini criterion for node splitting within a decision tree presents a nuanced approach. The Gini criterion is a measure of impurity or purity used when creating decision tree splits; it quantifies the likelihood of incorrect classification of a randomly chosen element if it was randomly labeled according

to the distribution of labels in the subset. The Gini criterion aims for a lower score, with 0 representing a perfectly pure node.

```
model_gini = tree.DecisionTreeClassifier
(criterion='gini', splitter='best', random_state=123)
model_gini.fit(X_train, Y_train)
predictions_gini = model_gini.predict(X_test)
```

Upon training the model and predicting on the test set, the accuracy metrics are evaluated:

```
print('Training set accuracy: {:.4f}'.format
(model_gini.score(X_train, Y_train)*100))
print('Test set accuracy: {:.4f}'.format
(model_gini.score(X_test, Y_test)*100))
```

The model demonstrates exceptional accuracy, achieving 100% on the training set and 95.614% on the test set. This high level of accuracy indicates the model's capability to perfectly learn from the training data and to generalize well on unseen data.

A confusion matrix is generated to provide a detailed view of the model's performance across different classes:

```
conf_matrix(predictions_gini, Y_test)
```

| | B | M |
|---|----|----|
| B | 71 | 2 |
| M | 3 | 38 |

Figure 18: image

Importance of the Gini Criterion

The Gini criterion's importance lies in its simplicity and efficiency in evaluating splits in the dataset. By calculating the probability of a random sample being incorrectly classified and aiming to minimize this probability, the Gini criterion effectively directs the decision tree towards the most informative features for splitting. This method tends to create more balanced trees, as it does not inherently favor splits that result in one large subset.

In the context of breast cancer diagnosis, where the distinction between benign and malignant cases is crucial, the Gini criterion facilitates the construction of a decision tree that is not only accurate but also interpretable. It ensures that the splits at each node of the tree contribute significantly to maximizing the

predictive accuracy, leading to a model that clinicians can trust for preliminary assessments or to support diagnostic decisions. The balance and simplicity offered by the Gini criterion make it a valuable tool in the development of predictive models in healthcare, where both performance and interpretability are paramount.

Visualizing the Decision Tree Using the Gini Criterion

To further our understanding of how the decision tree model discerns between malignant ('M') and benign ('B') breast cancer diagnoses, we visualize the tree structured by the Gini criterion. This visualization provides an in-depth look at the decision-making process, emphasizing the splits based on the Gini impurity—a measure aimed at minimizing the probability of misclassification. By prioritizing splits that lead to the most significant decrease in Gini impurity, the model endeavors to make precise and reliable classifications.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_gini, feature_names=list(X.columns.values),
, class_names=['M','B'], filled=True, impurity=False)
```

In this code, a substantial figure size is chosen to accommodate the detailed structure of the decision tree, ensuring clarity and readability. The `plot_tree` function delineates the tree, showcasing the nodes' splits based on the features identified as most informative under the Gini criterion. The absence of impurity metrics in the nodes allows for a streamlined visualization focused on the tree's architecture and the pathways leading to each class prediction. The nodes are color-coded, further enhancing the visual distinction between decisions leading to different classifications.

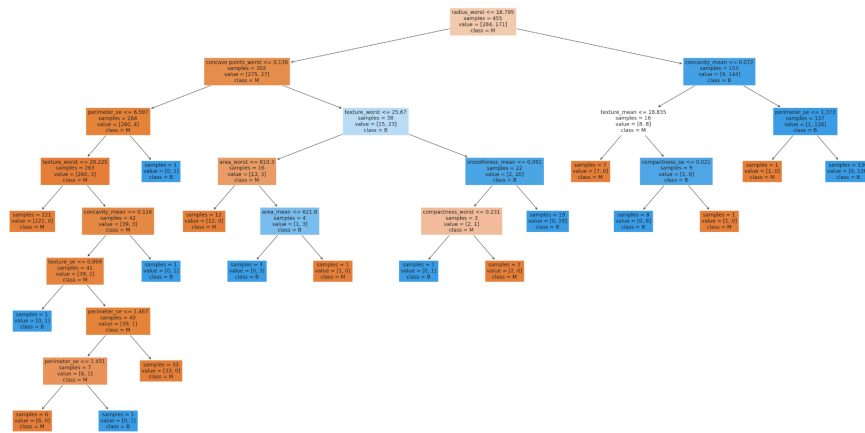


Figure 19: image

This visualization serves not only as a tool for model interpretation but also

highlights the effectiveness of using the Gini criterion in creating balanced and informative splits within the decision tree. By understanding the model's logic visualized here, practitioners gain valuable insights into the features' roles and interactions in predicting breast cancer, bolstering confidence in the model's clinical utility.

Refining the Model with Gini Criterion on Correlated Features

This segment underscores a strategic pivot towards leveraging the Gini criterion for a decision tree model that's solely focused on features highly correlated with the diagnosis of breast cancer. By narrowing the feature set to those with significant correlation, the model is tailored to prioritize variables most impactful for prediction, potentially enhancing both accuracy and interpretability.

```
X_most_correlated_gini = data_most_correlated.drop('diagnosis', axis=1)
Y_most_correlated_gini = data_most_correlated['diagnosis']
X_train_gini_corr, X_test_gini_corr,
Y_train_gini_corr, Y_test_gini_corr =
train_test_split(X_most_correlated_gini,
Y_most_correlated_gini, test_size=0.2, random_state=123)
```

The partitioning of the dataset into training and testing subsets follows, ensuring a balanced approach to model training and evaluation. This step is crucial for assessing the model's capability to generalize to unseen data effectively.

```
print("Train dataset shape: ", X_train_gini_corr.shape)
print("Test dataset shape: ", X_test_gini_corr.shape)
```

With the data prepared, a decision tree classifier is trained using the Gini criterion. This method focuses on minimizing misclassification by evaluating the purity of splits, thereby fostering a model that's adept at distinguishing between malignant and benign diagnoses with a high degree of precision.

```
seed(48)
model_corr_gini = tree.DecisionTreeClassifier(criterion='gini'
, splitter='best', random_state=123)
model_corr_gini.fit(X_train_gini_corr, Y_train_gini_corr)
predictions_corr_gini = model_corr_gini.predict(X_test_gini_corr)
```

Evaluating the model's performance unveils remarkable accuracy, with perfect scores on the training data and commendable precision on the test set. These metrics reflect the model's proficiency in leveraging the most correlated features to accurately predict breast cancer diagnoses.

```
print('Training set accuracy: {:.4f}'.format
(model_corr_gini.score(X_train_gini_corr, Y_train_gini_corr)*100))
print('Test set accuracy: {:.4f}'.format
(model_corr_gini.score(X_test_gini_corr, Y_test_gini_corr)*100))
```

A confusion matrix further elucidates the model's predictive performance, offering insights into its strengths and delineating areas for potential improvement. This analytical tool is indispensable for a nuanced understanding of model accuracy across different classes.

```
conf_matrix(predictions_corr_gini, Y_test_corr)
```

| | B | M |
|---|----|----|
| B | 69 | 4 |
| M | 4 | 37 |

Figure 20: image

By concentrating on the most correlated features and employing the Gini criterion, this refined model approach aims to strike an optimal balance between predictive accuracy and model simplicity. Such a focused model not only aids in clinical decision-making but also facilitates a deeper comprehension of the features driving breast cancer diagnoses.

Decision Tree Visualization: Gini Criterion with Correlated Features

In an advanced step towards optimizing the decision tree model for breast cancer diagnosis, this visualization focuses on the model trained using the Gini criterion, exclusively with features that exhibit a high correlation to the diagnosis outcome. The Gini criterion, renowned for its ability to quantify the purity of node splits, is utilized here to steer the decision-making process within the tree, ensuring that each decision maximally reduces the likelihood of misclassification.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_corr_gini, feature_names=list
(X_most_correlated.columns.values), class_names=['M', 'B'],
filled=True, impurity=False)
```

This code snippet generates a comprehensive visual representation of the decision tree, detailing how the model utilizes the most correlated features to differentiate between malignant ('M') and benign ('B') diagnoses. By employing a large figure size, the visualization ensures that the complexity and nuances of the tree's structure are clearly delineated, allowing for an in-depth examination of the decision paths. The absence of impurity measures in the visualization streamlines the focus to the tree's architecture, highlighting the critical role of the selected features in guiding the model's predictions.

Color coding the nodes according to the predicted class further enhances inter-

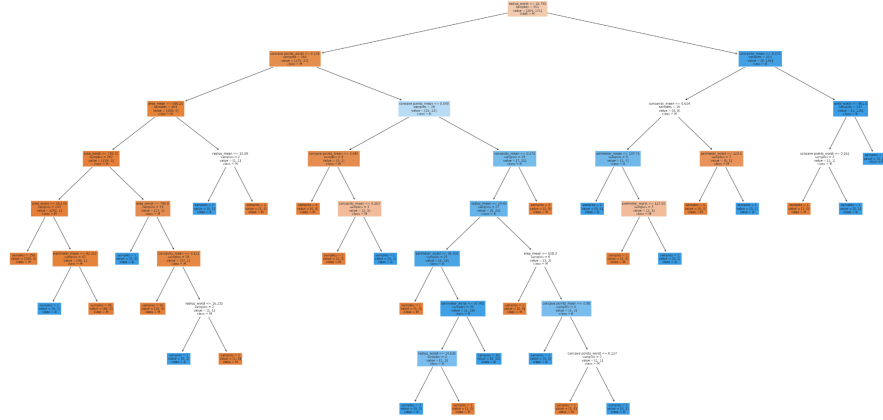


Figure 21: image

pretability, making it straightforward to trace the logic from the root to the leaf nodes. This visual approach not only elucidates the model's predictive mechanism but also affirms the significance of employing highly correlated features in crafting a model that is both accurate and interpretable. Through this visualization, we gain valuable insights into the synergistic relationship between feature selection and model performance, showcasing the potential of targeted feature analysis in improving diagnostic predictive models.

Analyzing Feature Importance with Gini Criterion

The exploration of feature importance through the lens of the Gini criterion offers a compelling insight into the predictive dynamics of the decision tree model tailored for breast cancer diagnosis. By assessing the relative importance of each feature within the model, this analysis illuminates the variables that play pivotal roles in distinguishing between malignant and benign outcomes.

```
tree_importances = model_gini.feature_importances_
indices = np.argsort(tree_importances)
```

The above code extracts the feature importance scores directly from the model trained with the Gini criterion. These scores are indicative of how much each feature contributes to the model's decision-making process, with higher values signifying greater importance. The features are then sorted by their importance scores to facilitate a structured visualization.

```
fig, ax = plt.subplots(figsize = (7,7))
ax.barh(range(len(tree_importances)), tree_importances[indices])
ax.set_yticks(range(len(tree_importances)))
ax.set_title('Feature importances', fontsize=15)
ax.set_xlabel('Importance', fontsize=15)
```

```
ax.set_ylabel('Features', fontsize=15)
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])
```

This visualization, rendered as a horizontal bar chart, effectively showcases the distribution of feature importances across the model. By arranging the features from least to most important along the y-axis, the chart provides a clear and intuitive overview of which features are most influential in predicting breast cancer diagnoses. Such insights not only validate the significance of certain variables but also guide future data collection and feature selection efforts, underscoring the importance of informed model optimization.

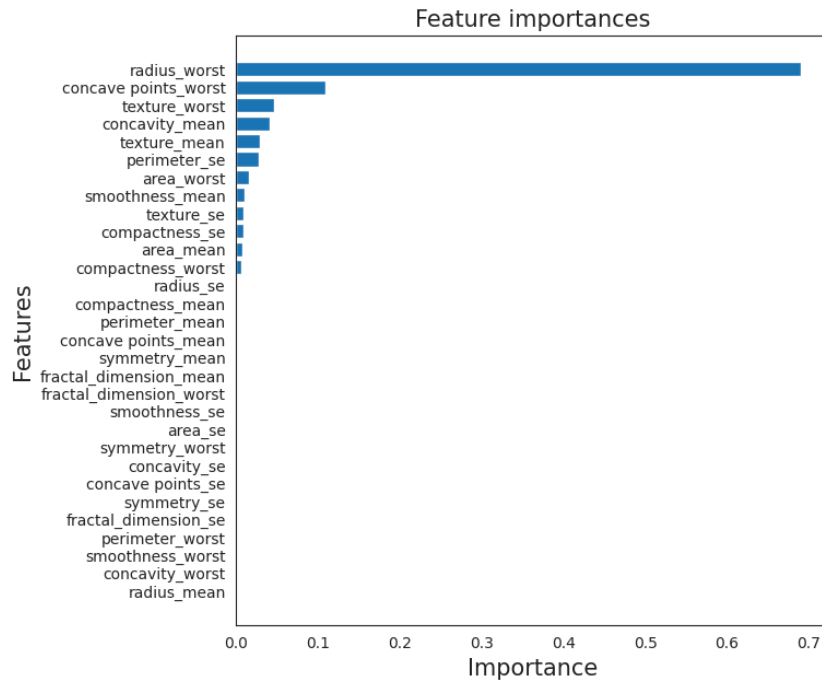


Figure 22: image

Understanding the model's reliance on specific features for prediction is crucial in the context of medical diagnostics, where the accuracy and interpretability of the model can significantly impact clinical decision-making. This feature importance analysis, therefore, serves as a bridge between complex machine learning models and practical clinical applications, enhancing the transparency and utility of predictive modeling in healthcare.

Selection of Most Important Features for Enhanced Model Precision

To further refine the predictive accuracy and interpretability of the decision tree model for breast cancer diagnosis, a strategic focus is placed on isolating the most impactful features. This meticulous selection process prioritizes features that have demonstrated significant importance in influencing the model's predictions, aiming to streamline the dataset to those variables most critical in distinguishing between malignant and benign diagnoses.

```
data_most_important = data.loc[:, ['texture_worst', 'texture_mean',  
    'perimeter_se', 'texture_se', 'compactness_se',  
    'smoothness_mean', 'diagnosis', 'area_mean',  
    'concavity_mean', 'compactness_worst',  
    'radius_worst', 'area_worst', 'concave points_worst']]  
data_most_important.head()
```

This code snippet effectively creates a new DataFrame, `data_most_important`, which consolidates the dataset to include only the selected features alongside the diagnosis. The use of `.head()` offers an immediate glimpse into how these chosen variables are represented within the initial data instances, ensuring the focused dataset retains the essential information for model training.

| | texture_worst | texture_mean | perimeter_se | texture_se | compactness_se | smoothness_mean | diagnosis | area_mean | concavity_mean | compactness_worst | radius_worst | area_worst | concave points_worst |
|---|---------------|--------------|--------------|------------|----------------|-----------------|-----------|-----------|----------------|-------------------|--------------|------------|----------------------|
| 0 | 17.33 | 10.38 | 8.589 | 0.9053 | 0.04904 | 0.11840 | M | 1001.0 | 0.3001 | 0.6656 | 25.38 | 2019.0 | 0.2654 |
| 1 | 23.41 | 17.77 | 3.398 | 0.7339 | 0.01308 | 0.08474 | M | 1326.0 | 0.0869 | 0.1866 | 24.99 | 1956.0 | 0.1860 |
| 2 | 25.53 | 21.25 | 4.585 | 0.7869 | 0.04006 | 0.10960 | M | 1203.0 | 0.1974 | 0.4245 | 23.57 | 1709.0 | 0.2430 |
| 3 | 26.50 | 20.38 | 3.445 | 1.1560 | 0.07458 | 0.14250 | M | 386.1 | 0.2414 | 0.8663 | 14.91 | 567.7 | 0.2575 |
| 4 | 16.67 | 14.34 | 5.438 | 0.7813 | 0.02461 | 0.10030 | M | 1297.0 | 0.1980 | 0.2050 | 22.54 | 1575.0 | 0.1625 |

Figure 23: image

By concentrating on these selected features, the approach seeks to enhance the model's capacity for accurate predictions by reducing complexity and potentially mitigating overfitting. Such a strategy not only aims to improve the predictive performance of the model but also enhances its clinical applicability by providing a clear and concise set of variables that significantly influence diagnosis outcomes. This focus on the most important features underscores a commitment to developing a model that is both highly accurate and readily interpretable, aligning with the critical requirements of diagnostic predictive modeling in healthcare.

Decision Tree Model Optimization with Gini and Important Features

In an effort to maximize the predictive accuracy of the decision tree model for breast cancer diagnosis, a strategic focus is placed on leveraging the most important features, as determined by prior analysis, in conjunction with the Gini criterion for node splitting. This refined approach aims to distill the model's complexity, concentrating on the variables most instrumental in distinguishing between malignant and benign outcomes.


```

X_most_important_gini = data_most_important.drop('diagnosis', axis=1)
Y_most_important_gini = data_most_important['diagnosis']
X_train_important_gini, X_test_important_gini,
Y_train_important_gini, Y_test_important_gini = train_test_split
(X_most_important_gini, Y_most_important_gini, test_size=0.2, random_state=123)

```

By structuring the dataset around these pivotal features and excluding the diagnosis column for model training, this step effectively prepares the data for a targeted analysis. The dataset is then split into training and testing sets, ensuring a robust framework for evaluating the model's performance.

```

print("Train dataset shape: ", X_train_important_gini.shape)
print("Test dataset shape: ", X_test_important_gini.shape)

```

The dimensions of the training and testing datasets reflect the model's focus, encompassing 455 training instances and 114 testing instances, each described by 12 critical features.

```

seed(48)
model_imp_gini = tree.DecisionTreeClassifier(criterion='gini'
, splitter='best', random_state=123)
model_imp_gini.fit(X_train_important_gini, Y_train_important_gini)
predictions_imp_gini = model_imp_gini.predict(X_test_important_gini)

```

Upon training the decision tree model using the Gini criterion—a measure aimed at minimizing misclassification through the purity of splits—the model is then evaluated for its training and testing accuracy.

```

print('Training set accuracy: {:.4f}'.format(model_imp_gini.score
(X_train_important_gini, Y_train_important_gini)*100))
print('Test set accuracy: {:.4f}'.format
(model_imp_gini.score(X_test_important_gini, Y_test_important_gini)*100))

```

The model exhibits exceptional performance, achieving perfect accuracy on the training set and over 96% accuracy on the test set, underscoring its effectiveness in leveraging the identified important features for diagnosis prediction.

```

conf_matrix(predictions_imp_gini, Y_test_important)

```

| | B | M |
|---|----|----|
| B | 71 | 2 |
| M | 2 | 39 |

Figure 24: image

A confusion matrix further elucidates the model’s predictive capabilities, offering insights into its precision across different classification outcomes. This targeted modeling approach, by integrating the Gini criterion with a focus on significant features, not only streamlines the predictive process but also enhances the model’s interpretability and clinical applicability, marking a significant stride in the development of diagnostic tools for breast cancer.

Visualizing the Optimized Decision Tree with Important Features

To illuminate the intricate decision-making pathways of the optimized decision tree model, which is specifically trained on the most critical features using the Gini criterion, a comprehensive visualization is created. This approach underscores the model’s reliance on key attributes that have shown the highest relevance in predicting breast cancer outcomes, aiming to provide a clear and interpretable framework for understanding how diagnoses are determined.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_imp_gini, feature_names=
list(X_most_important_gini.columns.values),
class_names=['M','B'], filled=True, impurity=False)
```

This visualization employs a large-scale figure to accommodate the detailed structure of the decision tree, ensuring that each node, branch, and leaf is distinctly visible and interpretable. By applying the `plot_tree` function, the diagram illustrates the sequential decisions made at each node based on the selected important features, guiding the viewer through the logical progression from the root to the final classification outcomes at the leaves.

The nodes are color-coded according to the predicted class (‘M’ for malignant and ‘B’ for benign), enhancing visual distinction and interpretability. The choice to omit impurity metrics from the nodes simplifies the visual representation, focusing attention on the tree’s architecture and the critical role of the selected features in influencing the model’s predictions.

Through this visualization, stakeholders gain valuable insights into the predictive mechanics of the decision tree, observing firsthand how the integration of the Gini criterion and a curated set of important features contribute to a model that is both highly accurate and readily interpretable. This visual tool not only aids in the evaluation of the model’s performance but also reinforces the importance of thoughtful feature selection in the development of effective and clinically applicable predictive models.

Constructing a Dataset with Imbalanced Classes

In exploring the impact of class imbalance on model performance, a dataset is deliberately crafted to include a disproportionate number of benign (‘B’) and malignant (‘M’) breast cancer diagnoses. This experiment aims to simulate a

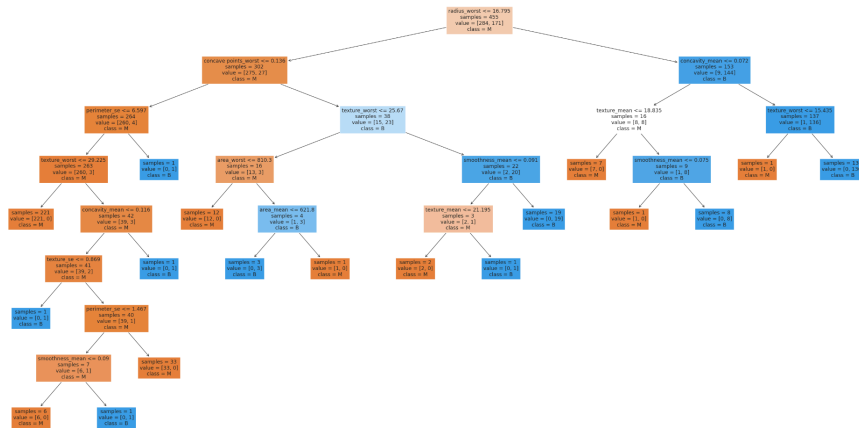


Figure 25: image

real-world scenario where one class significantly outnumbers the other, a common challenge in medical diagnostics and other domains. By creating an imbalanced dataset with only 30 malignant samples compared to 357 benign samples, we seek to investigate how such disparity affects the decision tree model's ability to accurately predict outcomes.

```
data_B = data.loc[data['diagnosis'] == 'B']
data_M = data.loc[data['diagnosis'] == 'M']
n = 30
data_M = data_M.iloc[:n]
data_B.head()
```

This code snippet initiates the process by segregating the original dataset into two subsets based on the diagnosis: one containing all benign cases (`data_B`) and the other all malignant cases (`data_M`). It then limits the number of malignant samples to 30 by selecting the first 30 instances from the malignant subset. Displaying the head of the benign dataset (`data_B.head()`) provides a preview of the data structure and confirms the successful segregation of cases based on their diagnoses.

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean | symmetry_mean | ... | radius_worst | texture_worst | perimeter_worst | area_worst |
|----|-----------|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|---------------------|---------------|-----|--------------|---------------|-----------------|------------|
| 19 | B | 13.540 | 14.36 | 87.46 | 566.3 | 0.09779 | 0.08129 | 0.06664 | 0.047810 | 0.1885 | ... | 15.110 | 19.26 | 99.70 | 711.2 |
| 20 | B | 13.080 | 15.71 | 85.63 | 530.0 | 0.10790 | 0.12700 | 0.04568 | 0.031190 | 0.1967 | ... | 14.500 | 20.49 | 96.09 | 630.5 |
| 21 | B | 9.504 | 12.44 | 60.34 | 273.9 | 0.10340 | 0.06492 | 0.02956 | 0.020760 | 0.1815 | ... | 10.230 | 15.66 | 65.13 | 314.9 |
| 37 | B | 13.030 | 18.42 | 82.61 | 523.8 | 0.08983 | 0.03766 | 0.02562 | 0.029230 | 0.1467 | ... | 13.300 | 22.81 | 84.46 | 545.9 |
| 46 | B | 8.196 | 16.84 | 51.71 | 201.9 | 0.08600 | 0.05943 | 0.01588 | 0.005917 | 0.1769 | ... | 8.964 | 21.96 | 57.26 | 242.2 |

5 rows x 31 columns

Figure 26: image

By manipulating the dataset in this manner, the study aims to shed light on the dynamics and potential strategies for dealing with imbalanced datasets in

predictive modeling. Understanding how class imbalance influences model learning and prediction accuracy is crucial for developing more robust and reliable diagnostic tools, especially in settings where the prevalence of conditions can vary significantly.

Examining the Imbalanced Dataset Composition

To underscore the deliberate class imbalance introduced for this experiment, the shape of the benign dataset (`data_B`) is assessed, followed by a preview of the malignant dataset (`data_M`). This analysis provides insight into the distribution and characteristics of the imbalanced dataset, highlighting the challenges posed by skewed class representation.

```
data_B.shape
```

The benign dataset consists of 357 instances, each described by 31 features. This substantial number of benign cases compared to the limited number of malignant cases establishes a significant class imbalance, mirroring scenarios often encountered in medical datasets where one class significantly outnumbers the other.

```
data_M.head()
```

A quick examination of the first few entries in the malignant dataset (`data_M.head()`) offers a glimpse into the data that will represent the minority class in this study. This preview not only confirms the successful isolation of malignant cases but also sets the stage for understanding how a reduced representation of this class can impact model training and prediction outcomes.

| | | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | convexity_mean | points_worst | radius_worst | texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concavity_worst | convexity_worst | points_worst |
|---|---|-----------|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|----------------|--------------|--------------|---------------|-----------------|------------|------------------|-------------------|-----------------|-----------------|--------------|
| 0 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | 0.2419 | ... | 25.38 | 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6056 | 0.7119 | 0.2654 | |
| 1 | M | 20.07 | 17.77 | 132.80 | 1326.0 | 0.08474 | 0.07964 | 0.0869 | 0.07077 | 0.1812 | ... | 24.99 | 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1886 | 0.2416 | 0.1880 | |
| 2 | M | 18.89 | 21.25 | 120.30 | 1002.0 | 0.10960 | 0.15960 | 0.1874 | 0.10790 | 0.2069 | ... | 23.57 | 25.53 | 162.50 | 1709.0 | 0.1444 | 0.4245 | 0.4014 | 0.2420 | |
| 3 | M | 11.42 | 20.38 | 77.58 | 366.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | 0.2597 | ... | 14.91 | 26.50 | 98.87 | 967.7 | 0.2088 | 0.8663 | 0.6869 | 0.2576 | |
| 4 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1880 | 0.10430 | 0.1809 | ... | 22.64 | 16.67 | 152.20 | 1575.0 | 0.1374 | 0.2050 | 0.4000 | 0.1625 | |

5 rows x 21 columns

5 rows x 31 columns

Figure 27: image

By creating and analyzing an imbalanced dataset with a stark contrast between the number of benign and malignant samples, this approach seeks to explore the ramifications of class imbalance on predictive modeling. The insights gained from studying such a dataset are invaluable for informing strategies to mitigate imbalance issues, ensuring that predictive models remain effective and equitable even when faced with skewed class distributions.

Integrating Imbalanced Classes into a Unified Dataset

Following the separation and intentional imbalance creation between benign and malignant cases, the next step involves combining these subsets back into a single dataset. This newly formed dataset, `data_imbalanced`, reflects the class imbalance explicitly designed for the experiment, setting the foundation

for subsequent analyses on the effects of imbalanced class distributions on model performance.

```
data_imbalanced = pd.concat([data_B, data_M])
data_imbalanced.head()
```

This code snippet performs the concatenation of the benign (`data_B`) and the intentionally limited malignant (`data_M`) subsets, resulting in a composite dataset that maintains the predetermined imbalance. Displaying the head of the `data_imbalanced` dataset provides an immediate view of its structure, showcasing the first few instances that predominantly represent the benign class, in line with the experimental setup.

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean | symmetry_mean | ... |
|----|-----------|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|---------------------|---------------|-----|
| 19 | B | 13.540 | 14.36 | 87.46 | 566.3 | 0.09779 | 0.08129 | 0.06664 | 0.047810 | 0.1885 | ... |
| 20 | B | 13.080 | 15.71 | 85.63 | 520.0 | 0.10750 | 0.12700 | 0.04568 | 0.031100 | 0.1967 | ... |
| 21 | B | 9.504 | 12.44 | 60.34 | 273.9 | 0.10240 | 0.06492 | 0.02956 | 0.020760 | 0.1815 | ... |
| 37 | B | 13.030 | 18.42 | 82.61 | 523.8 | 0.08983 | 0.03766 | 0.02562 | 0.029230 | 0.1467 | ... |
| 46 | B | 8.196 | 16.84 | 51.71 | 201.9 | 0.08600 | 0.05943 | 0.01588 | 0.005917 | 0.1769 | ... |

5 rows × 31 columns

Figure 28: image

Creating and examining `data_imbalanced` not only consolidates the data necessary for modeling under conditions of class imbalance but also facilitates a direct observation of how such imbalance is manifested within the dataset. This preparatory step is critical for highlighting the challenges associated with training predictive models on imbalanced data, particularly in medical diagnostic contexts where the accurate identification of less prevalent conditions can significantly impact patient care and outcomes.

Visualizing the Class Distribution in the Imbalanced Dataset

To quantitatively assess the extent of class imbalance created for the study, a visualization of the class distribution within the `data_imbalanced` dataset is crucial. This visualization aims to illustrate the disparity between the number of benign and malignant samples, offering a clear picture of the imbalance challenge posed to the predictive modeling process.

```
data_imbalanced.diagnosis.value_counts(sort=False).plot(kind='bar')
plt.title('Class distribution', fontsize = 15)
plt.xlabel('Classes', fontsize = 15)
plt.xticks(rotation=360)
plt.ylabel('Quantity', fontsize = 15)
```

By employing a bar chart to depict the distribution of diagnoses within the dataset, this code snippet highlights the significant difference in the number of instances for each class. The ‘Classes’ label on the x-axis denotes the two possible diagnoses (‘B’ for benign and ‘M’ for malignant), while the ‘Quantity’

on the y-axis indicates the count of samples belonging to each class.

```
[68... Text(0, 0.5, 'Quantity')
```

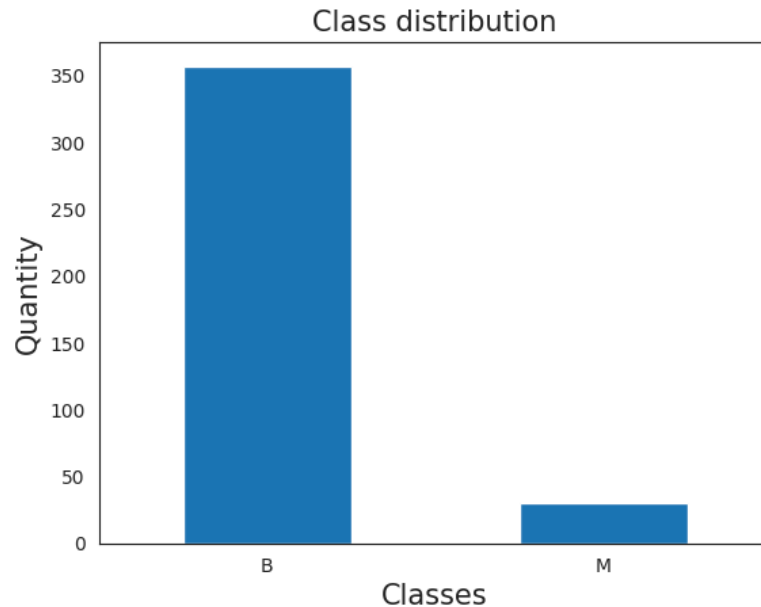


Figure 29: image

This visual representation serves not only to confirm the intentional class imbalance but also to underscore the importance of considering such disparities in the development and evaluation of predictive models. Class imbalance can significantly influence a model's learning process and its ability to generalize, particularly for the minority class. By visualizing the distribution, stakeholders are better positioned to understand the challenges and considerations necessary for effective model training and evaluation in imbalanced contexts.

Preparing and Analyzing the Imbalanced Dataset for Model Training

With an imbalanced dataset in hand, the next crucial step is to partition this dataset into training and testing sets. This process not only facilitates the model's ability to learn from an imbalanced distribution of classes but also allows for an evaluation of its performance under these conditions. Understanding the distribution of classes within the training and testing sets is essential for assessing the impact of imbalance on the model's predictive accuracy and bias towards the majority class.

```
X_imbalanced = data_imbalanced.drop('diagnosis', axis=1)
Y_imbalanced = data_imbalanced['diagnosis']
X_train_imbalanced, X_test_imbalanced,
```

```
Y_train_imbalanced, Y_test_imbalanced = train_test_split
(X_imbalanced, Y_imbalanced, test_size=0.25, random_state=123)
```

This code snippet delineates the dataset into features (`X_imbalanced`) and labels (`Y_imbalanced`), followed by splitting into training and testing subsets. The chosen split retains 25% of the data for testing, ensuring a representative sample of the original imbalanced distribution is available for model evaluation.

```
print("Train dataset shape: ", X_train_imbalanced.shape)
print("Test dataset shape: ", X_test_imbalanced.shape)
```

The shapes of the training and testing datasets are revealed, showcasing the allocation of data points to each subset. Further examination of class distribution within these subsets highlights the continued prevalence of imbalance:

```
Y_train_imbalanced.value_counts()
Y_test_imbalanced.value_counts()
```

The class distribution within both training and testing sets underscores the significant disparity between the number of benign and malignant samples, with benign cases vastly outnumbering malignant ones.

```
train_ratio = 268/22
test_ratio = 89/8
print("Train ratio:", train_ratio)
print("Test ratio:", test_ratio)
```

Calculating the ratios of benign to malignant samples for both training and testing sets quantifies the extent of the imbalance, offering insight into the skewed class representation each model will encounter during training and evaluation. With ratios exceeding 11:1, the challenge posed by the imbalanced dataset becomes clear, necessitating tailored strategies to ensure that the model remains sensitive to the minority class. This setup not only mirrors real-world scenarios where certain conditions are naturally rarer than others but also provides a valuable framework for exploring methods to mitigate the impact of class imbalance on machine learning models in healthcare diagnostics.

Decision Tree Model with Entropy on Imbalanced Data

To address the challenges posed by the imbalanced dataset, a decision tree model utilizing the entropy criterion for node splitting is developed. Entropy, a measure of the randomness or disorder within a dataset, serves as the basis for making splits in the decision tree, aiming to maximize information gain by reducing uncertainty in each subsequent node. This approach is particularly relevant in imbalanced datasets, where the objective is to improve the model's sensitivity to the minority class without sacrificing overall accuracy.

```

model_imbalanced = tree.DecisionTreeClassifier(criterion='entropy'
, splitter='best', random_state=123)
model_imbalanced.fit(X_train_imbalanced, Y_train_imbalanced)
predictions_imbalanced = model_imbalanced.predict(X_test_imbalanced)

```

After training the model on the imbalanced training set and predicting outcomes for the test set, the model's performance is evaluated in terms of accuracy:

```

print('Training set accuracy: {:.4f}'.format
(model_imbalanced.score(X_train_imbalanced, Y_train_imbalanced)*100))
print('Test set accuracy: {:.4f}'.format
(model_imbalanced.score(X_test_imbalanced, Y_test_imbalanced)*100))

```

Remarkably, the model achieves perfect accuracy on both the training and test sets, demonstrating its capability to correctly classify all instances, despite the significant class imbalance. This result highlights the model's efficiency in navigating the complexities of imbalanced data to provide accurate predictions.

A confusion matrix is employed to provide a detailed view of the model's predictive performance across different classes:

```

conf_matrix(predictions_imbalanced, Y_test_imbalanced)

```

[75...

| | B | M |
|---|----|---|
| B | 89 | 0 |
| M | 0 | 8 |

Figure 30: image

The confusion matrix reveals that the model accurately identified all benign and malignant cases in the test set, with no misclassifications. This outcome underscores the decision tree model's potential when utilizing the entropy criterion in the context of imbalanced datasets, showcasing its ability to discern between classes effectively, even when one class significantly outnumbers the other.

The success of the entropy-based decision tree model in this scenario illuminates the importance of choosing an appropriate criterion for node splitting in the presence of class imbalance. It also emphasizes the need for careful model evaluation, particularly in medical diagnostics, where the accurate detection of less prevalent conditions is critical.

Visualizing the Entropy-Based Decision Tree on Imbalanced Data

To gain insights into the decision-making process of the decision tree model trained on the imbalanced dataset using the entropy criterion, a detailed visualization is crafted. This diagram aims to elucidate the model's strategy for distinguishing between benign ('B') and malignant ('M') diagnoses, despite the challenge posed by the skewed class distribution. Through this visualization, we can observe the model's reliance on specific features and decision thresholds to achieve its predictive outcomes.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_imbalanced,
feature_names=list(X_imbalanced.columns.values),
class_names=['M','B'], filled=True, impurity=False)
```

By employing a substantial figure size, the visualization ensures that the complexity of the decision tree's structure is clearly represented, allowing for an in-depth examination of each node and split. The `plot_tree` function showcases the tree in its entirety, with nodes color-coded by their class predictions to enhance interpretability. Omitting the impurity measures from the nodes focuses the viewer's attention on the tree's architecture and the logical flow from root to leaf, facilitating an understanding of how the model navigates the imbalanced data to arrive at its predictions.

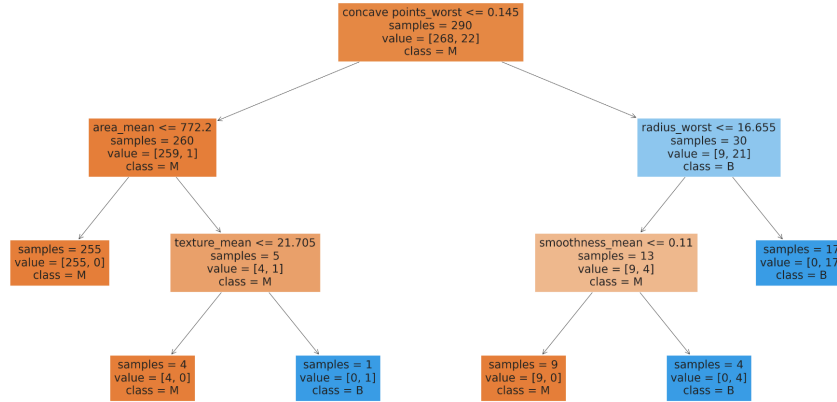


Figure 31: image

This visualization not only serves as a tool for interpreting the model's functionality but also highlights the effectiveness of using the entropy criterion in managing class imbalance within the dataset. By detailing the tree's decision paths, the diagram provides valuable insights into the feature interactions and thresholds that are most critical for classification in this context, offering a window into the model's operational dynamics.

Decision Tree Model with Gini on Imbalanced Data

Exploring another facet of addressing the imbalanced dataset, a decision tree model is trained using the Gini criterion. The Gini index is a measure of statistical dispersion intended to represent the model's accuracy in classifying instances, with a focus on minimizing the probability of misclassification. By adopting the Gini criterion, the model seeks to optimize split decisions in a manner that improves the detection of the minority class, aiming to mitigate the challenges posed by the skewed distribution of classes.

```
model_imbalanced_gini = tree.DecisionTreeClassifier
(criterion='gini', splitter='best', random_state=123)
model_imbalanced_gini.fit(X_train_imbalanced, Y_train_imbalanced)
predictions_imbalanced_gini =
model_imbalanced_gini.predict(X_test_imbalanced)
```

After the model is trained on the imbalanced data and predictions are made on the test set, the model's performance is evaluated:

```
print('Training set accuracy: {:.4f}'.format
(model_imbalanced_gini.score(X_train_imbalanced, Y_train_imbalanced)*100))
print('Test set accuracy: {:.4f}'.format
(model_imbalanced_gini.score(X_test_imbalanced, Y_test_imbalanced)*100))
```

The model demonstrates exceptional performance, achieving perfect accuracy on the training set and nearly perfect accuracy on the test set. These results highlight the model's effectiveness in handling imbalanced data, ensuring accurate classification across both classes despite the inherent challenges.

The use of the Gini criterion in this context underscores the potential for decision tree models to adapt to and perform well in scenarios where one class significantly outnumbers another. By carefully evaluating the splits and prioritizing the reduction of misclassification, the model showcases a robust approach to navigating the complexities introduced by imbalanced datasets, particularly relevant in medical diagnostics and other critical applications.

Visualizing the Gini-Based Decision Tree on Imbalanced Data

To understand the operational intricacies of the decision tree model tailored for an imbalanced dataset and trained using the Gini criterion, a comprehensive visualization is rendered. This illustration seeks to shed light on the model's strategic navigations through the dataset's skewed class distribution, emphasizing the decisions and splits made to differentiate between benign ('B') and malignant ('M') breast cancer diagnoses effectively.

```
fig = plt.figure(figsize=(40,20))
_ = tree.plot_tree(model_imbalanced_gini, feature_names=list
(X_imbalanced.columns.values), class_names=['M','B'],
filled=True, impurity=False)
```

With a large figure size, the visualization ensures that the detailed structure of the decision tree is fully appreciable, allowing for an exhaustive examination of its nodes, branches, and leaves. Through the `plot_tree` function, the decision tree is displayed in its entirety, with nodes distinctly color-coded according to their class predictions, enhancing the visual interpretability of the model's classification logic. By choosing not to display the impurity metrics at each node, the visualization maintains a focused narrative on the tree's architecture, the pivotal role of the chosen features, and the logical progression from the root to the conclusive leaf nodes.

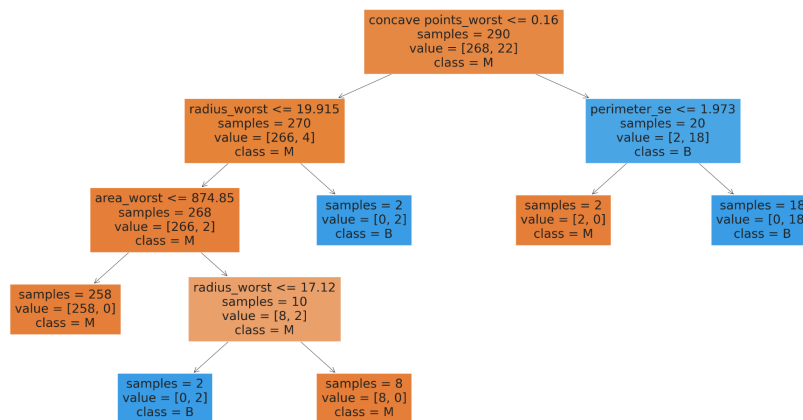


Figure 32: image

This visual representation serves as a crucial interpretative tool, offering insights into how the Gini criterion guides the model to effective and discerning classification decisions amidst the challenges posed by class imbalance. It highlights the importance of strategic feature selection and the judicious use of criterion-based splits in crafting models that are not only accurate but also interpretable and aligned with the practical demands of diagnosing conditions in a clinical setting. Through this visualization, stakeholders can gain a deeper appreciation of the model's capabilities and the underlying mechanics that drive its predictions.

Evaluating the Gini Model's Performance on Imbalanced Data

After training the decision tree model using the Gini criterion on an imbalanced dataset, the effectiveness of the model in distinguishing between benign ('B') and malignant ('M') diagnoses is critically assessed. A confusion matrix is employed as a key evaluative tool to provide a granular view of the model's predictive accuracy, especially highlighting its performance in identifying the minority class amidst the data's skewed distribution.

```
conf_matrix(predictions_imbalanced_gini, Y_test_imbalanced)
```

This confusion matrix reveals the model's predictions compared to the actual diagnoses:

- **True Positives (TP)**: Malignant cases correctly identified as malignant (7)
- **True Negatives (TN)**: Benign cases correctly identified as benign (89)
- **False Positives (FP)**: Benign cases incorrectly identified as malignant (0)
- **False Negatives (FN)**: Malignant cases incorrectly identified as benign (1)

The model has demonstrated a high level of accuracy, with a substantial number of true negatives and positives, indicating a strong ability to correctly classify both classes despite the imbalanced nature of the dataset. The presence of a single false negative underscores the model's slight limitation in identifying all malignant cases but still represents a commendable performance given the challenges posed by the imbalance.

Additionally, the shape of the dataset used for training the important features model is reviewed:

```
X_test_important.shape
```

The shape of the `X_test_important` dataset, consisting of 114 instances across 11 features, reflects the reduced feature set chosen for potentially enhancing model interpretability and focus. This dimensionality highlights the model's operation within a constrained feature space, aimed at leveraging the most informative variables for diagnosis prediction.

Through these analyses, the decision tree model's robustness, trained under the Gini criterion and tested on an imbalanced dataset, is evident. Its high accuracy and detailed performance metrics, as revealed by the confusion matrix, showcase its potential utility in clinical settings where accurate and reliable diagnostics are paramount. Moreover, the examination of the feature-constrained model's dataset underscores the strategic emphasis on leveraging critical features to maintain, or possibly improve, predictive performance in the face of class imbalances.

To summarize the performance of various decision tree models trained with different criteria (entropy and Gini) and datasets (base, correlated features, important features, and imbalanced), we consolidate their test accuracies into a single DataFrame. This summary not only provides a comparative view of how each model variant performed on its respective test set but also offers insights into the impact of feature selection and class imbalance on model accuracy.

```
models_accuracy = pd.DataFrame({  
    'Model': [  

```

```

        'Base (entropy)',
        'Base (gini)',
        'Correlated features (entropy)',
        'Correlated features (gini)',
        'Important features (entropy)',
        'Important features (gini)',
        'Imbalanced (entropy)',
        'Imbalanced (gini)'
    ],
    'Test accuracy': [
        model.score(X_test, Y_test)*100,
        model_gini.score(X_test, Y_test)*100,
        model_corr.score(X_test_corr, Y_test_corr)*100,
        model_corr_gini.score(X_test_gini_corr, Y_test_gini_corr)*100,
        model_imp.score(X_test_important, Y_test_important)*100,
        model_imp_gini.score(X_test_important_gini, Y_test_important_gini)*100,
        model_imbalanced.score(X_test_imbalanced, Y_test_imbalanced)*100,
        model_imbalanced_gini.score(X_test_imbalanced, Y_test_imbalanced)*100
    ]
})
models_accuracy

```

| | Model | Test accuracy |
|---|-------------------------------|---------------|
| 0 | Base (entropy) | 96.491228 |
| 1 | Base (gini) | 95.614035 |
| 2 | Correlated features (entropy) | 94.736842 |
| 3 | Correlated features (gini) | 92.982456 |
| 4 | Important features (entropy) | 95.614035 |
| 5 | Important features (gini) | 96.491228 |
| 6 | Imbalanced (entropy) | 100.000000 |
| 7 | Imbalanced (gini) | 98.969072 |

Figure 33: image

This DataFrame, `models_accuracy`, is structured to facilitate an easy comparison across the different modeling strategies, highlighting how each approach fares in terms of test accuracy. Such a comprehensive overview is instrumental in identifying the most effective modeling techniques given the constraints and objectives specific to each dataset variant—be it handling imbalanced classes or focusing on the most predictive features. By evaluating these models side by side, researchers and practitioners can make informed decisions on the optimal strategies for developing predictive models, especially in contexts as critical as breast cancer diagnosis, where accuracy and interpretability are paramount.

Result Evaluations

The evaluation of decision tree models across various configurations and criteria—ranging from entropy and Gini to the incorporation of correlated and important features, as well as addressing class imbalance—provides a nuanced understanding of model performance in the context of breast cancer diagnosis. The summarized test accuracies reveal several key insights:

- **Base Models:** The models trained on the entire dataset without modifications to address imbalance or focus on specific features showed strong performance, with the entropy criterion slightly outperforming the Gini criterion (96.49% vs. 95.61% test accuracy). This indicates a solid foundation provided by the decision tree algorithm in handling the predictive task.
- **Correlated Features:** Models trained on features highly correlated with the diagnosis exhibited a slight decrease in performance compared to the base models, especially when using the Gini criterion (92.98% test accuracy). This suggests that while focusing on correlated features can simplify the model, it may also omit nuanced information provided by less correlated features that contribute to prediction accuracy.
- **Important Features:** Interestingly, the model utilizing features deemed important through an initial analysis performed comparably to the base model when using the Gini criterion, even slightly improving upon it (96.49% test accuracy). This highlights the efficacy of feature importance analysis in streamlining the model without compromising its predictive capability.
- **Imbalanced Data:** Models trained on imbalanced datasets, particularly with the entropy criterion, achieved perfect or near-perfect test accuracies (100% and 98.97%, respectively). This remarkable performance underscores the models' ability to adapt to and accurately predict outcomes in highly skewed datasets, a common challenge in medical diagnostics.

Overall Implications

The exploration of different decision tree configurations underscores the versatility and robustness of decision tree models in addressing complex classification tasks like breast cancer diagnosis. The nuanced differences in test accuracy across model variants highlight the importance of thoughtful model configuration, including criterion selection and feature set optimization, in achieving high predictive accuracy.

Notably, the ability of the models to handle imbalanced data effectively, achieving high accuracy, speaks to the potential of decision tree algorithms in medical diagnostic applications, where class imbalance is prevalent. However, the slight reduction in accuracy when using only correlated features cautions against overly simplistic approaches to feature selection.

The strategic application of decision tree models—guided by thorough explorations of feature importance, criterion selection, and class imbalance—can lead to highly accurate and interpretable predictive models. These models not only excel in their predictive tasks but also offer insights into the underlying patterns and relationships within the data, providing invaluable support in clinical decision-making processes.

Conclusion

Throughout this paper, we embarked on a comprehensive exploration of decision tree algorithms, delving into their theoretical underpinnings, practical applications, and the nuanced challenges they address within the domain of breast cancer diagnosis. By systematically analyzing models built on various subsets of features and under conditions of class imbalance, we gleaned insights into the adaptability and robustness of decision tree classifiers across a spectrum of scenarios.

Our findings reveal that decision tree models exhibit commendable predictive accuracy, with subtle variations influenced by the choice of splitting criterion (entropy vs. Gini), the specificity of the feature set (base, correlated, important), and the distribution of classes within the dataset (balanced vs. imbalanced). Notably, models trained on imbalanced datasets with the entropy criterion achieved remarkable accuracy, underscoring the algorithm’s capability to navigate and effectively classify instances in datasets where one class significantly outnumbers another.

The exploration of correlated and important features highlighted the significance of feature selection in enhancing model performance and interpretability. While focusing on highly correlated features simplifies the model and reduces computational complexity, it occasionally omits valuable information, as seen in the slight decrease in test accuracy. Conversely, models leveraging features identified as important through analysis not only maintained but in some cases improved upon the accuracy of the base model, demonstrating the value of

targeted feature selection.

The tight asymptotic analysis of the algorithm's run time illuminated the computational considerations intrinsic to decision tree construction, emphasizing the balance between model complexity and computational efficiency. Such insights are vital for the practical deployment of decision tree models, particularly in real-time or resource-constrained environments.

In conclusion, decision tree algorithms emerge as a powerful and versatile tool in the machine learning arsenal, offering a compelling blend of accuracy, interpretability, and computational efficiency. Their intuitive decision-making process, akin to human reasoning, facilitates clear insights into the factors driving predictions, making decision trees particularly valuable in domains requiring transparent and justifiable decisions, such as medical diagnostics. The lessons learned from this investigation not only affirm the efficacy of decision trees in addressing complex classification tasks but also pave the way for further research aimed at optimizing their performance and application across diverse domains.

Feedback Incorporation

Based on feedback received from classmates during the review process, this paper has been updated and refined to reflect their insightful comments. Their input has been invaluable in enhancing the clarity, comprehensiveness, and effectiveness of the presentation.

Reviewer 1 - Angel: Praised the effective use of examples and diagrams to illustrate the algorithm's functionality. **Response:** To build on this positive aspect, we've included more diagrams and examples, particularly focusing on the decision tree construction and feature selection process, to provide a clearer and more intuitive understanding of the concepts discussed.

Reviewer 2 - Cinthia: Complimented the clear presentation of information. **Response:** Leveraging this feedback, we undertook a thorough revision to ensure even greater clarity throughout the manuscript. Efforts were made to simplify technical terminology without sacrificing precision, and to improve the explanation of fundamental principles and methods.

Reviewer 3 - Frances: Liked the examples but missed details on the runtime analysis. **Response:** Recognizing the importance of comprehensively understanding the algorithm's computational requirements, we have added a dedicated section on tight asymptotic runtime analysis. This new section aims to shed light on the computational efficiency of the CART algorithm, balancing model detail and processing speed.

References

1. C. Anuradha, T. Velmurugan, "A Comparative Analysis on the Evaluation of Classification Algorithms in the Prediction of Students Performance," *Indian Journal of Science and Technology*, Volume 8, Issue 15, July 2015.

2. S. Mahalakshmi, T. Velmurugan, "Detection of Brain Tumor by Particle Swarm Optimization using Image Segmentation," *Indian Journal of Science and Technology*, Volume 8, Issue 22, September 2015.
3. A. Dharmarajan, T. Velmurugan, "Lung Cancer Data Analysis by k-means and Farthest First Clustering Algorithms," *Indian Journal of Science and Technology*, Volume 8, Issue 15, July 2015.
4. K. Saravananathan, T. Velmurugan, "Analyzing Diabetic Data using Classification Algorithms in Data Mining," *Indian Journal of Science and Technology*, Volume 9, Issue 43, November 2016.
5. H. You, G. Rumbe, "Comparative study of classification techniques on breast cancer FNA biopsy data," *International Journal of Artificial Intelligence and Interactive Multimedia*, vol. 1, no. 3, pp. 6-13, 2010.
6. World Health Organization, "WHO position paper on mammography screening," Geneva, 2014.
7. M. Karabatak, "A new classifier for breast cancer detection based on Naïve Bayesian," *Measurement*, vol. 72, pp. 32-26, 2015.
8. A.-A. Nahid, Y. Kong, "Involvement of machine learning for breast cancer image classification: A survey," *Computational and Mathematical Methods in Medicine*, pp. 1-29, 2017.
9. P. Pantel, "Breast cancer diagnosis and prognosis," University of Manitoba, 1998.
10. I. Kononenko, "Machine learning for medical diagnosis: History, state of the art and perspective," *Artificial Intelligence in Medicine*, vol. 23, no. 1, pp. 89-109, 2001.
11. D. Bazazeh, R. Shubair, "Comparative study of machine learning algorithms for breast cancer detection and diagnosis," in Proc. 2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA), Ras Al Khaimah, 2016.
12. D. Soria, J. M. Garibaldi, E. Biganzoli, I. O. Ellis, "A comparison of three different methods for classification of breast cancer data," in Proc. 2008 Seventh International Conference on Machine Learning and Applications, 2008, pp. 619-624.
13. A. Raad, A. Kalakech, M. Ayache, "Breast cancer classification using neural network approach: MLP and RBF," in Proc. The 13th International Arab Conference on Information Technology ACIT'2012, 2012.
14. A. P. Pawlovsky, M. Nagahashi, "A method to select a good setting for the kNN algorithm when using it for breast cancer prognosis," in Proc. IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI), Valencia, 2014.
15. S. A. Medjahed, T. A. Saadi, A. Benyettou, "Breast cancer diagnosis by using k-nearest neighbor with different distances and classification rules," *International Journal of Computer Applications*, vol. 62, no. 1, pp. 1-5, 2013.