In [36]:
```python
import csv
import math
from collections import Counter
```

In [37]:
```python
# ---------- 1. Preprocessing ----------
```

In [38]:
```python
def read_csv(path):
    with open(path, 'r') as file:
        reader = csv.reader(file, delimiter=';')
        header = next(reader)
        data = [row for row in reader]
    return header, data
```

In [39]:
```python
def encode_data(header, data):
    encoders = {}
    encoded_data = []
    target_col_index = len(header) - 1
    for col_index in range(len(header)):
        col_values = [row[col_index].strip('"') for row in data]
        if col_index == target_col_index:
            mapping = {'no': 0, 'yes': 1}
            for row in data:
                row[col_index] = mapping[row[col_index].strip('"')]
            encoders[header[col_index]] = mapping
            continue
        try:
            [float(v) for v in col_values]
            encoders[header[col_index]] = None
        except:
            unique_vals = sorted(set(col_values))
            mapping = {val: idx for idx, val in enumerate(unique_vals)}
            for row in data:
                row[col_index] = mapping[row[col_index].strip('"')]
            encoders[header[col_index]] = mapping
    for row in data:
        encoded_data.append([float(value) for value in row])
    return encoded_data, encoders
```

In [40]:
```python
def split_dataset(data):
    return data[:4000], data[4000:4400], data[4400:]
```

In [41]:
```python
def split_features_labels(dataset):
    X = [row[:-1] for row in dataset]
    y = [row[-1] for row in dataset]
    return X, y
```

In [42]:
```python
# ---------- 2. ID3 ----------
```

In [43]:
```python
def entropy(labels):
    total = len(labels)
    counts = Counter(labels)
    return -sum((count / total) * math.log2(count / total) for count in counts.values())
```

In [44]:
```python
def info_gain(X, y, feature_index):
    total_entropy = entropy(y)
    subsets = {}
    for xi, yi in zip(X, y):
        key = xi[feature_index]
        if key not in subsets:
            subsets[key] = {'X': [], 'y': []}
        subsets[key]['X'].append(xi)
        subsets[key]['y'].append(yi)
    weighted_entropy = 0
    total = len(y)
    for group in subsets.values():
        proportion = len(group['y']) / total
        weighted_entropy += proportion * entropy(group['y'])
    return total_entropy - weighted_entropy
```

In [45]:
```python
def build_id3(X, y, features):
    if len(set(y)) == 1:
        return {'label': y[0]}
    if not features:
        return {'label': Counter(y).most_common(1)[0][0]}
    gains = [info_gain(X, y, f) for f in features]
    best_feature = features[gains.index(max(gains))]
    tree = {'feature': best_feature, 'branches': {}}
    for value in set([xi[best_feature] for xi in X]):
        subset_X, subset_y = [], []
        for xi, yi in zip(X, y):
            if xi[best_feature] == value:
```

```python
                new_xi = xi[:best_feature] + xi[best_feature+1:]
                subset_X.append(new_xi)
                subset_y.append(yi)
        new_features = [f if f < best_feature else f - 1 for f in features if f != best_feature]
        tree['branches'][value] = build_id3(subset_X, subset_y, new_features)
    return tree
```

In [46]:
```python
def majority_class(tree):
    labels = []
    def collect_labels(t):
        if 'label' in t:
            labels.append(t['label'])
        else:
            for child in t['branches'].values():
                collect_labels(child)
    collect_labels(tree)
    return Counter(labels).most_common(1)[0][0]
```

In [47]:
```python
def predict_id3(tree, x):
    while 'label' not in tree:
        feature_index = tree['feature']
        value = x[feature_index]
        if value in tree['branches']:
            tree = tree['branches'][value]
            x = x[:feature_index] + x[feature_index+1:]
        else:
            return majority_class(tree)
    return tree['label']
```

In [48]:
```python
# ---------- 3. CART ----------
```

In [49]:
```python
def gini_index(groups, classes):
    total = sum(len(group) for group in groups)
    gini = 0.0
    for group in groups:
        size = len(group)
        if size == 0:
            continue
        score = 0.0
        counts = Counter(group)
        for c in classes:
            p = counts[c] / size
            score += p * p
        gini += (1 - score) * (size / total)
    return gini
```

In [50]:
```python
def get_best_split(X, y):
    best_index, best_value, best_score, best_groups = None, None, float('inf'), None
    classes = list(set(y))
    for index in range(len(X[0])):
        values = set([row[index] for row in X])
        for value in values:
            left_y = [y[i] for i in range(len(X)) if X[i][index] == value]
            right_y = [y[i] for i in range(len(X)) if X[i][index] != value]
            gini = gini_index([left_y, right_y], classes)
            if gini < best_score:
                best_index, best_value, best_score = index, value, gini
                best_groups = (
                    [X[i] for i in range(len(X)) if X[i][index] == value],
                    [y[i] for i in range(len(X)) if X[i][index] == value],
                    [X[i] for i in range(len(X)) if X[i][index] != value],
                    [y[i] for i in range(len(X)) if X[i][index] != value]
                )
    return {'index': best_index, 'value': best_value, 'groups': best_groups}
```

In [51]:
```python
def build_cart(X, y, max_depth, depth=0):
    if len(set(y)) == 1 or depth >= max_depth:
        return {'label': Counter(y).most_common(1)[0][0]}
    node = get_best_split(X, y)
    left_X, left_y, right_X, right_y = node['groups']
    node['left'] = build_cart(left_X, left_y, max_depth, depth+1)
    node['right'] = build_cart(right_X, right_y, max_depth, depth+1)
    return node
```

In [52]:
```python
def predict_cart(tree, x):
    if 'label' in tree:
        return tree['label']
    if x[tree['index']] == tree['value']:
        return predict_cart(tree['left'], x)
    else:
        return predict_cart(tree['right'], x)
```

In [53]:
```python
# ---------- 4. Naive Bayes ----------
```

In [54]:
```python
def train_naive_bayes(X, y):
    summaries = {}
    class_counts = Counter(y)
    total = len(y)
    for class_val in class_counts:
        indices = [i for i in range(len(y)) if y[i] == class_val]
        class_data = [X[i] for i in indices]
        summaries[class_val] = {
            'prior': class_counts[class_val] / total,
            'features': list(zip(*class_data))
        }
    return summaries
```

In [55]:
```python
def predict_naive_bayes(model, x):
    probs = {}
    for class_val, info in model.items():
        prob = info['prior']
        for i in range(len(x)):
            values = info['features'][i]
            count = values.count(x[i])
            prob *= (count + 1) / (len(values) + len(set(values)))  # Laplace smoothing
        probs[class_val] = prob
    return max(probs, key=probs.get)
```

In [56]:
```python
# ---------- 5. Accuracy ----------
```

In [57]:
```python
def accuracy(y_true, y_pred):
    return sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp) / len(y_true)
```

In [58]:
```python
# ---------- 6. Run Everything ----------
```

In [59]:
```python
header, raw_data = read_csv('bank.csv')
encoded_data, encoders = encode_data(header, raw_data)
train_set, val_set, pred_set = split_dataset(encoded_data)
```

In [60]:
```python
X_train, y_train = split_features_labels(train_set)
X_val, y_val = split_features_labels(val_set)
X_pred = [row[:-1] for row in pred_set]
```

In [61]:
```python
# ID3
features = list(range(len(X_train[0])))
id3_tree = build_id3(X_train, y_train, features)
id3_preds = [predict_id3(id3_tree, x) for x in X_val]
acc_id3 = accuracy(y_val, id3_preds)
```

In [62]:
```python
# CART
cart_tree = build_cart(X_train, y_train, max_depth=5)
cart_preds = [predict_cart(cart_tree, x) for x in X_val]
acc_cart = accuracy(y_val, cart_preds)
```

In [63]:
```python
# Naive Bayes
nb_model = train_naive_bayes(X_train, y_train)
nb_preds = [predict_naive_bayes(nb_model, x) for x in X_val]
acc_nb = accuracy(y_val, nb_preds)
```

In [64]:
```python
# Comparison
print(f"ID3 Accuracy: {round(acc_id3 * 100, 2)}%")
print(f"CART Accuracy: {round(acc_cart * 100, 2)}%")
print(f"Naive Bayes Accuracy: {round(acc_nb * 100, 2)}%")
```

```
ID3 Accuracy: 85.0%
CART Accuracy: 88.5%
Naive Bayes Accuracy: 78.0%
```

In [65]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_val_true = [int(y) for y in y_val]
id3_pred = [int(p) for p in id3_preds]
cart_pred = [int(p) for p in cart_preds]
nb_pred = [int(p) for p in nb_preds]
def print_metrics(name, y_true, y_pred):
    print(f"\n{name} Metrics:")
    print(f"Accuracy : {accuracy_score(y_true, y_pred) * 100:.2f}%")
    print(f"Precision: {precision_score(y_true, y_pred) * 100:.2f}%")
    print(f"Recall   : {recall_score(y_true, y_pred) * 100:.2f}%")
    print(f"F1 Score : {f1_score(y_true, y_pred) * 100:.2f}%")
print_metrics("ID3", y_val_true, id3_pred)
print_metrics("CART", y_val_true, cart_pred)
print_metrics("Naive Bayes", y_val_true, nb_pred)
```

```
ID3 Metrics:
Accuracy : 85.00%
Precision: 23.81%
Recall   : 10.20%
F1 Score : 14.29%

CART Metrics:
Accuracy : 88.50%
Precision: 61.54%
Recall   : 16.33%
F1 Score : 25.81%

Naive Bayes Metrics:
Accuracy : 78.00%
Precision: 27.59%
Recall   : 48.98%
F1 Score : 35.29%
```

In [66]:
```python
#best model classifier
best_model = max([('ID3', acc_id3), ('CART', acc_cart), ('NB', acc_nb)], key=lambda x: x[1])[0]
print(f"Best Algorithm: {best_model}")
```

```
Best Algorithm: CART
```

In [67]:
```python
# Final Predictions
if best_model == 'ID3':
    final_preds = [predict_id3(id3_tree, x) for x in X_pred]
elif best_model == 'CART':
    final_preds = [predict_cart(cart_tree, x) for x in X_pred]
else:
    final_preds = [predict_naive_bayes(nb_model, x) for x in X_pred]
```

In [68]:
```python
# Reverse label encoding
label_decoder = {v: k for k, v in encoders['y'].items()} if 'y' in encoders else {0: 'no', 1: 'yes'}
final_preds_decoded = [label_decoder[int(p)] for p in final_preds]
```

In [69]:
```python
print("Final Predictions:")
print(" ".join(final_preds_decoded))
```

```
Final Predictions:
no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no yes no no no no no no
no no no no no no no no no no no no no no no no no no yes yes no no no no no no no no no no no no no no no no no no no no
no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no n
o
```

In [ ]: