File Edit View Run Kernel Nbgrader Tabs Settings

Assignments | Gradient_Boosting_Regressic | Random_Forest_From_Scratc | phone_battery_decision_tree

/ elite / Gradient_Boosting_Regression /

Name | Modified

Gradient_Boosting... 36m ago

Markdown | Validate | Notebook | Python 3 (ipykernel)

Phone Battery Replacement Prediction (Decision Tree)

" "This notebook is a **complete, runnable** mini-project for a campus phone-repair desk.

" "## Goal " "Predict whether a phone battery should be replaced using numeric features: " "- `cycle_count`: number of full charge cycles " "- `capacity_pct`: battery health percentage " "- `max_temp_c`: maximum operating temperature (°C) " "- `age_months`: phone age (months)

" "Target: " "- `replace` = 0 (No) " "- `replace` = 1 (Yes)

" "We will: " "1. Create / load a small dataset " "2. Train a **Decision Tree from scratch** (CART with Gini) " "3. Train a **scikit-learn** DecisionTreeClassifier for comparison " "4. Evaluate and interpret feature importances "

[1]:
```
import numpy as np
import pandas as pd

# For plots (sklearn tree visualization)
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

1) Create a small dataset

" "Below is a small, realistic dataset. You can replace it later with your own real repair-desk data. "

[2]:
```
data = [
    # cycle_count, capacity_pct, max_temp_c, age_months, replace
    [120, 94, 34,  8, 0],
    [260, 88, 36, 14, 0],
    [410, 84, 38, 18, 0],
    [520, 79, 39, 22, 1],
    [680, 76, 41, 26, 1],
    [750, 73, 42, 28, 1],
    [ 90, 97, 33,  6, 0],
    [300, 86, 35, 15, 0],
    [480, 81, 40, 20, 1],
    [610, 78, 41, 23, 1],
    [200, 90, 35, 12, 0],
    [560, 80, 39, 21, 1],
    [330, 87, 37, 16, 0],
    [820, 70, 44, 30, 1],
    [150, 92, 34, 10, 0],
    [700, 74, 43, 27, 1],
    [430, 83, 38, 19, 0],
    [590, 77, 41, 24, 1],
    [240, 89, 36, 13, 0],
    [900, 68, 45, 32, 1],
]

cols = ["cycle_count", "capacity_pct", "max_temp_c", "age_months", "replace"]
df = pd.DataFrame(data, columns=cols)
df.head()
```

[2]:

	cycle_count	capacity_pct	max_temp_c	age_months	replace
0	120	94	34	8	0
1	260	88	36	14	0
2	410	84	38	18	0
3	520	79	39	22	1
4	680	76	41	26	1

[3]:
```
# Optional: save as CSV for reuse
csv_path = "battery_replace.csv"
df.to_csv(csv_path, index=False)
csv_path
```

[3]: 'battery_replace.csv'

2) Train/Test split

" "We'll split the dataset for evaluation. "

[ ]:
```
X = df.drop(columns=["replace"])
y = df["replace"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)

X_train.shape, X_test.shape
```

3) Decision Tree **from scratch** (CART with Gini)

" "This implementation uses: " "- binary splits of the form `feature < threshold` " "- Gini impurity to pick the best split " "- stopping rules: `max_depth` and `min_size` "

[ ]:
```
from dataclasses import dataclass
from typing import Optional, Tuple, List, Any

def gini_for_groups(left_y: np.ndarray, right_y: np.ndarray, classes: np.ndarray) -> float:
    # weighted gini across the two child groups
    n_left, n_right = len(left_y), len(right_y)
    n_total = n_left + n_right
    gini = 0.0

    for group_y, n in [(left_y, n_left), (right_y, n_right)]:
        # 1. If the group is empty, skip it (why?)
        # 2. Compute the proportion (p) of each class in this group
        # 3. Compute the sum of squared proportions (Σ p²)
        # 4. Compute gini for this group as (1 - Σ p²)
        # 5. Weight the group gini by (group_size / total_size)
        # 6. Add it to the overall gini

        # YOUR CODE HERE
        raise NotImplementedError()
    return gini

def best_split(X: np.ndarray, y: np.ndarray) -> Tuple[int, float, float, np.ndarray, np.ndarray]:
    classes = np.unique(y)
    n_samples, n_features = X.shape

    best_feature = -1
    best_thresh = 0.0
    best_gini = float("inf")
    best_left_idx = None
    best_right_idx = None

    for f in range(n_features):
        # Candidate thresholds from observed values
        thresholds = np.unique(X[:, f])
```

```
        for t in thresholds:
            # 1. Split samples into left and right using feature f and threshold t
            #    (Hint: left = feature < threshold)
            # 2. Compute gini impurity for this split using gini_for_groups
            # 3. If this gini is smaller than the best so far:
            #       - update best_gini
            #       - store best_feature, best_thresh
            #       - store left and right indices

            # YOUR CODE HERE
            raise NotImplementedError()
    return best_feature, best_thresh, best_gini, best_left_idx, best_right_idx

def majority_class(y: np.ndarray) -> int:
    vals, counts = np.unique(y, return_counts=True)
    return int(vals[np.argmax(counts)])

@dataclass
class Node:
    feature: Optional[int] = None
    thresh: Optional[float] = None
    left: Optional[Any] = None   # Node or int class
    right: Optional[Any] = None  # Node or int class

def build_tree(X: np.ndarray, y: np.ndarray, max_depth: int, min_size: int, depth: int = 1) -> Any:
    # stopping rules
    if len(np.unique(y)) == 1:
        return int(y[0])
    if depth >= max_depth or len(y) <= min_size:
        return majority_class(y)

    f, t, g, left_idx, right_idx = best_split(X, y)

    # If split is degenerate, stop
    if left_idx is None or right_idx is None or left_idx.sum() == 0 or right_idx.sum() == 0:
        return majority_class(y)

    left_subtree = build_tree(X[left_idx], y[left_idx], max_depth, min_size, depth + 1)
    right_subtree = build_tree(X[right_idx], y[right_idx], max_depth, min_size, depth + 1)

    return Node(feature=f, thresh=t, left=left_subtree, right=right_subtree)

def predict_one(node: Any, row: np.ndarray) -> int:
    # 1. If the current node is NOT a Node instance:
    #       - it is a leaf, return the class
    # 2. Otherwise:
    #       - compare row[node.feature] with node.thresh
    #       - if less, recurse to left child
    #       - else, recurse to right child
    # YOUR CODE HERE
    raise NotImplementedError()

def predict(node: Any, X: np.ndarray) -> np.ndarray:
    return np.array([predict_one(node, row) for row in X], dtype=int)

def print_tree(node: Any, feature_names: List[str], indent: str = ""):
    if not isinstance(node, Node):
        print(indent + f"Predict: {node}")
        return
    name = feature_names[node.feature]
    print(indent + f"if {name} < {node.thresh}:")
    print_tree(node.left, feature_names, indent + "  ")
    print(indent + "else:")
    print_tree(node.right, feature_names, indent + "  ")
```

[ ]:
```
# Train from-scratch tree
Xtr = X_train.values.astype(float)
ytr = y_train.values.astype(int)

scratch_tree = build_tree(Xtr, ytr, max_depth=4, min_size=2)

print("From-scratch tree structure:")
print_tree(scratch_tree, feature_names=X.columns.tolist())
```

[ ]:
```
# Evaluate from-scratch tree
Xte = X_test.values.astype(float)
yte = y_test.values.astype(int)

pred_scratch = predict(scratch_tree, Xte)

print("From-scratch accuracy:", accuracy_score(yte, pred_scratch))
print("Confusion matrix:\n", confusion_matrix(yte, pred_scratch))
print("\nClassification report:\n", classification_report(yte, pred_scratch, digits=3))
```

4) scikit-learn Decision Tree (comparison)

" "We'll train a standard sklearn tree and visualize it. "

[ ]:
```
sk_tree = DecisionTreeClassifier(
    criterion="gini",
    max_depth=4,
    min_samples_leaf=2,
    random_state=42
)
sk_tree.fit(X_train, y_train)

pred_sk = sk_tree.predict(X_test)

print("sklearn accuracy:", accuracy_score(y_test, pred_sk))
print("Confusion matrix:\n", confusion_matrix(y_test, pred_sk))
print("\nClassification report:\n", classification_report(y_test, pred_sk, digits=3))
```

[ ]:
```
# Feature importances
imp = pd.Series(sk_tree.feature_importances_, index=X.columns).sort_values(ascending=False)
imp
```

[ ]:
```
# Visualize the sklearn tree (no Graphviz needed)
plt.figure(figsize=(14, 7))
plot_tree(
    sk_tree,
    feature_names=X.columns,
    class_names=["NoReplace", "Replace"],
    filled=True,
    rounded=True,
    impurity=True
)
plt.title("Decision Tree (sklearn)")
plt.show()
```

5) Try a few new predictions

" "Provide a few new phones and see the model output. "

[ ]:
```
new_phones = pd.DataFrame([
    {"cycle_count": 350, "capacity_pct": 85, "max_temp_c": 37, "age_months": 16},
    {"cycle_count": 780, "capacity_pct": 72, "max_temp_c": 45, "age_months": 29},
    {"cycle_count": 520, "capacity_pct": 82, "max_temp_c": 39, "age_months": 22},
])
```

```
print("New phones:")
display(new_phones)

print("\nPredictions (sklearn):", sk_tree.predict(new_phones).tolist(), "  (1=Replace, 0=No)")
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: