hw2-trees

June 4, 2025

1 hw2:

1.0.1

: 1. sklearn .
. 2.
sklearn.
.

, , .

1.0.2

« » ( ). --- 10 .
/ .

1.0.3

`hw2-trees.ipynb` `hw2code.py`
Github. Google- <> ,
Kaggle.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from matplotlib.colors import Colormap, ListedColormap
import pandas as pd
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')
```

2 1. . .

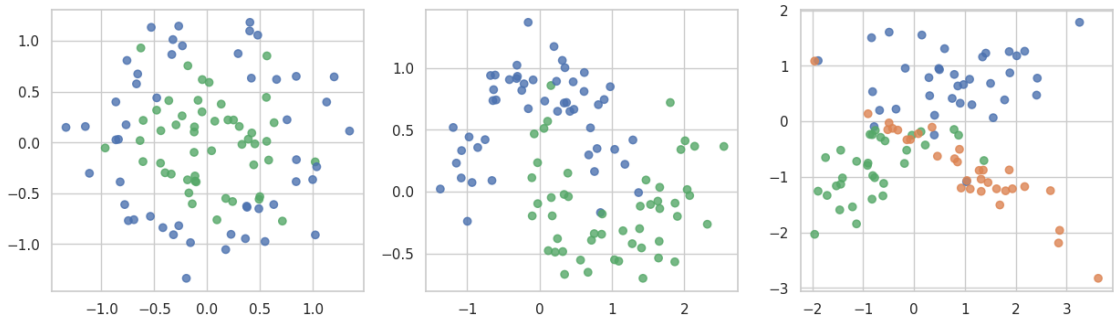
make_moons, make_circles

.

[2]:
```
from sklearn.datasets import make_moons, make_circles, make_classification
datasets = [
    make_circles(noise=0.2, factor=0.5, random_state=42),
    make_moons(noise=0.2, random_state=42),
    make_classification(n_classes=3, n_clusters_per_class=1, n_features=2,␣
 ↪class_sep=.8, random_state=3,
                        n_redundant=0, )
]
```

[3]:
```
palette = sns.color_palette(n_colors=3)
cmap = ListedColormap(palette)
```

[4]:
```
plt.figure(figsize=(15, 4))
for i, (x, y) in enumerate(datasets):
    plt.subplot(1, 3, i + 1)
    plt.scatter(x[:, 0], x[:, 1], c=y, cmap=cmap, alpha=.8)
```



1. (1)

, .

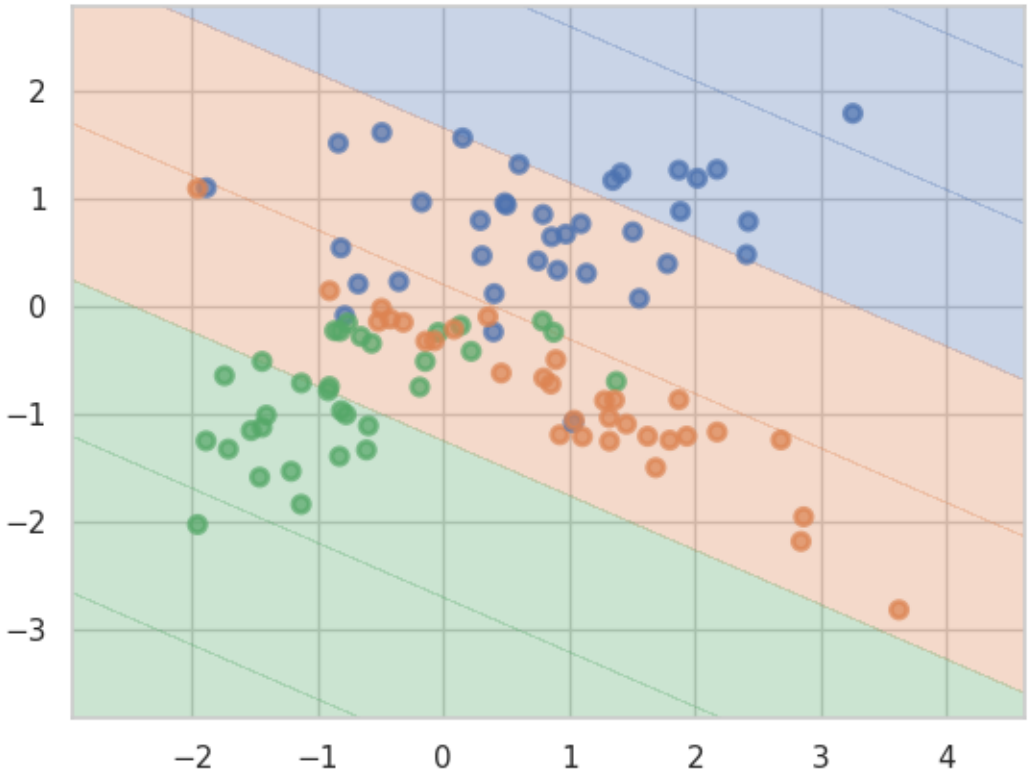
(plot_surface,). accuracy .

?

[5]:
```
def plot_surface(clf, X, y):
    plot_step = 0.01
    palette = sns.color_palette(n_colors=len(np.unique(y)))
    cmap = ListedColormap(palette)
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))
```

```
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.3)

    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap, alpha=.7,
                edgecolors=np.array(palette)[y], linewidths=2)
```

[6]:
```
#    :
from sklearn.linear_model import LinearRegression
X, y = datasets[2]
lr  = LinearRegression().fit(X, y)
plot_surface(lr, X, y)
```



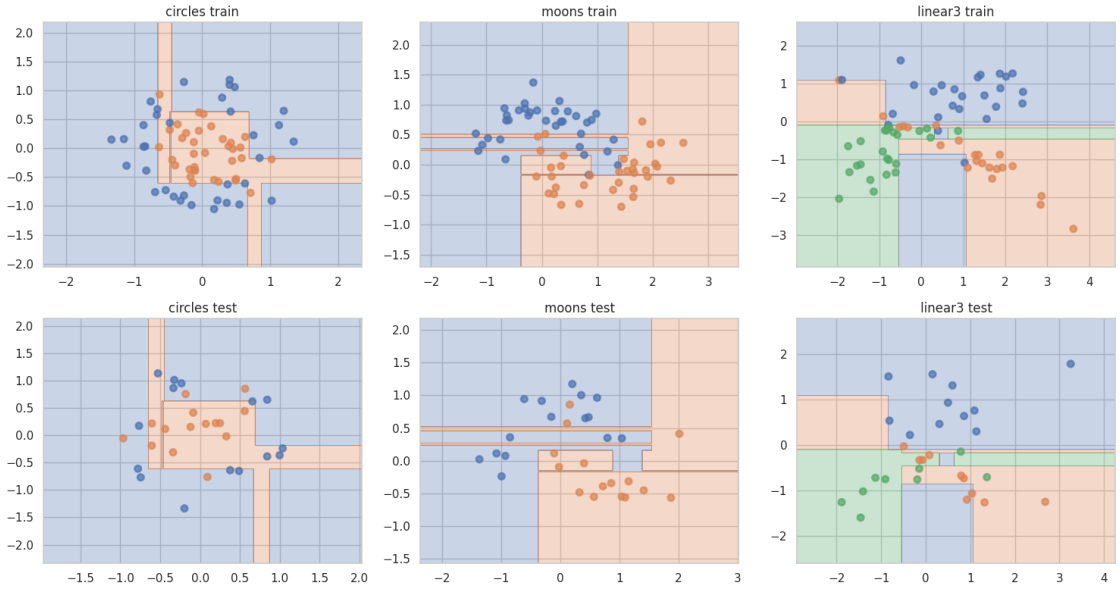
[7]:
```
### ( °   ° )  *:
```

[8]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

named_datasets = [
```

```
    ('circles', make_circles(noise=0.2, factor=0.5, random_state=42)),
    ('moons',   make_moons(noise=0.2, random_state=42)),
    ('linear3', make_classification(n_classes=3, n_clusters_per_class=1,
                                    n_features=2, class_sep=0.8,
                                    random_state=3, n_redundant=0))
]
```

[9]:
```
fig, axes = plt.subplots(2, 3, figsize=(15, 8))
for idx, (name, (X, y)) in enumerate(named_datasets):

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, stratify=y, random_state=42
    )
    clf = DecisionTreeClassifier(random_state=42)
    clf.fit(X_train, y_train)


    acc_train = accuracy_score(y_train, clf.predict(X_train))
    acc_test  = accuracy_score(y_test,  clf.predict(X_test))
    print(f"{name:8s} - train_acc = {acc_train:.3f}, test_acc = {acc_test:.3f}")

    plt.sca(axes[0, idx])
    plot_surface(clf, X_train, y_train)
    axes[0, idx].set_title(f"{name} train")

    plt.sca(axes[1, idx])
    plot_surface(clf, X_test, y_test)
    axes[1, idx].set_title(f"{name} test")

fig.tight_layout()
plt.show()
```

```
circles  - train_acc = 1.000, test_acc = 0.700
moons    - train_acc = 1.000, test_acc = 0.933
linear3  - train_acc = 1.000, test_acc = 0.667
```



:

train_acc 1 test_acc - , . ,
.

2. (1.5)

(. max_depth, min_samples_leaf).
, . - ,
, (,
?). . ,
?

[10]:
```
### ( °  ° )  *:
```

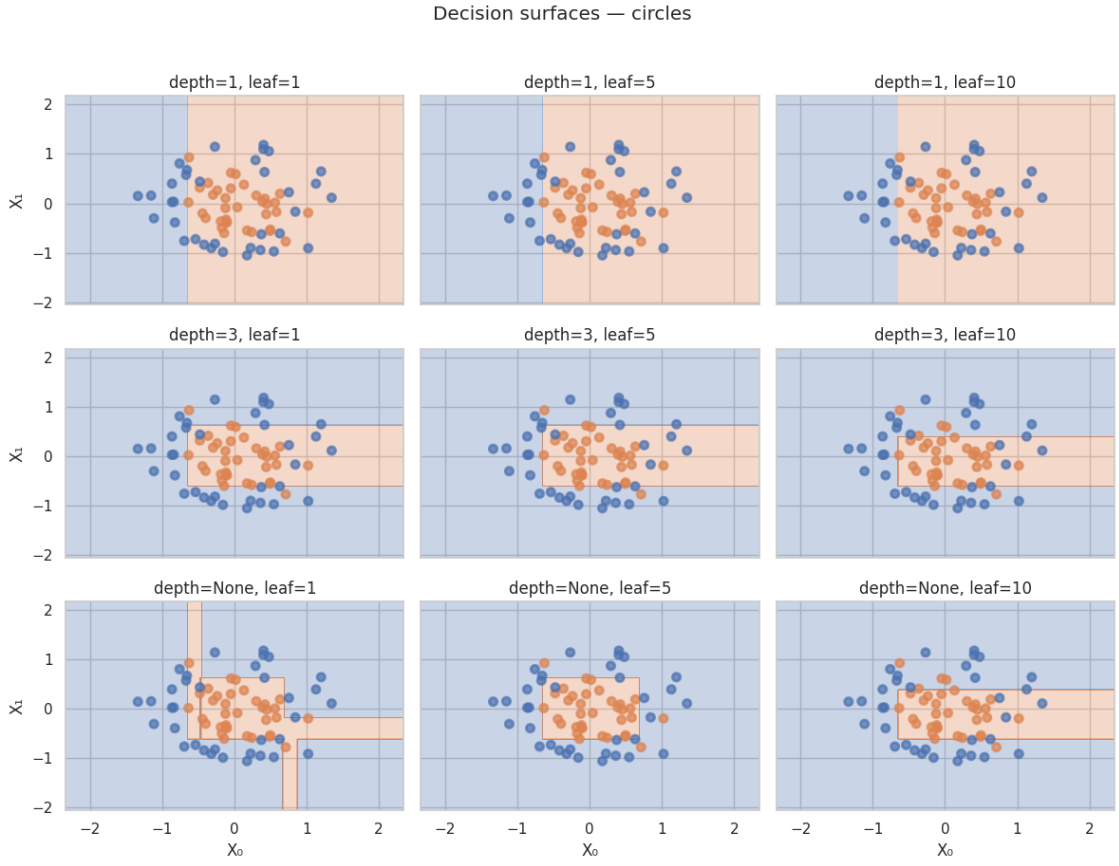
[11]:
```
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.model_selection import train_test_split

named_datasets = [
    ('circles', make_circles(noise=0.2, factor=0.5, random_state=42)),
    ('moons',   make_moons(noise=0.2, random_state=42)),
    ('linear3', make_classification(
        n_classes=3, n_clusters_per_class=1,
        n_features=2, class_sep=0.8,
        random_state=3, n_redundant=0
    ))
]

splits = {}
```

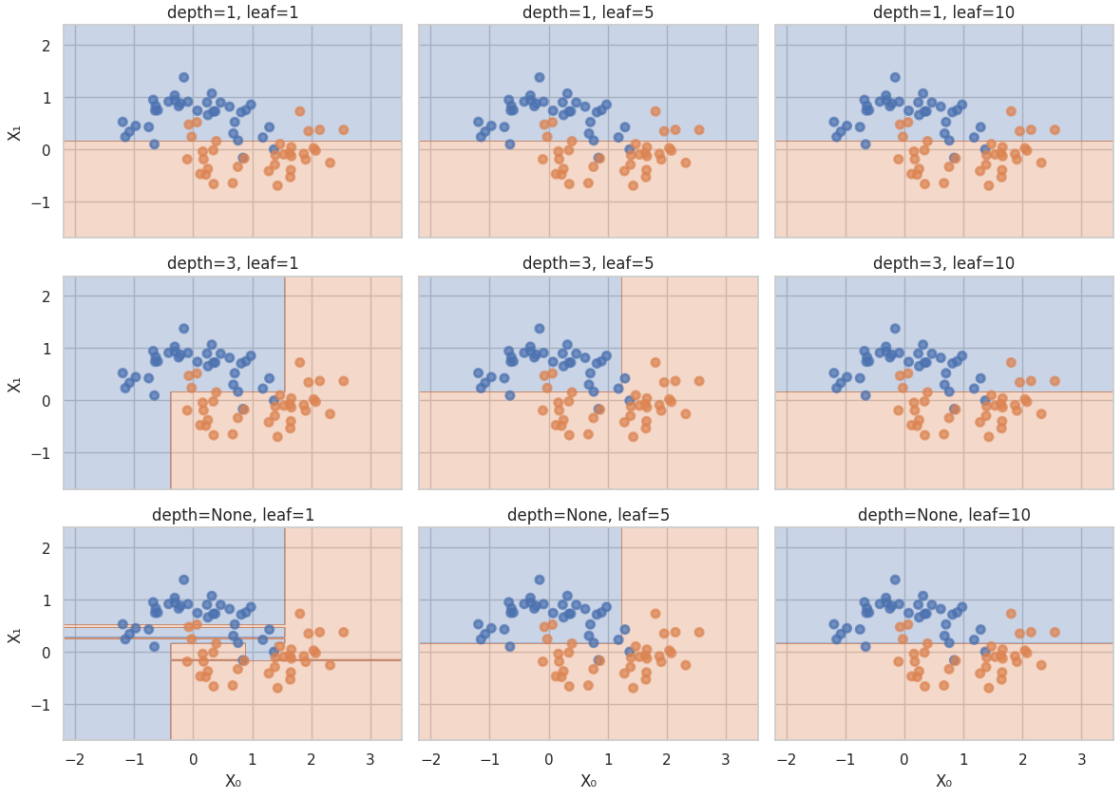
```
for name, (X, y) in named_datasets:
    X_tr, X_te, y_tr, y_te = train_test_split(
        X, y, test_size=0.3, stratify=y, random_state=42
    )
    splits[name] = (X_tr, X_te, y_tr, y_te)

param_grid = {
    'max_depth':        [1, 3, None],
    'min_samples_leaf':[1, 5, 10]
}
```

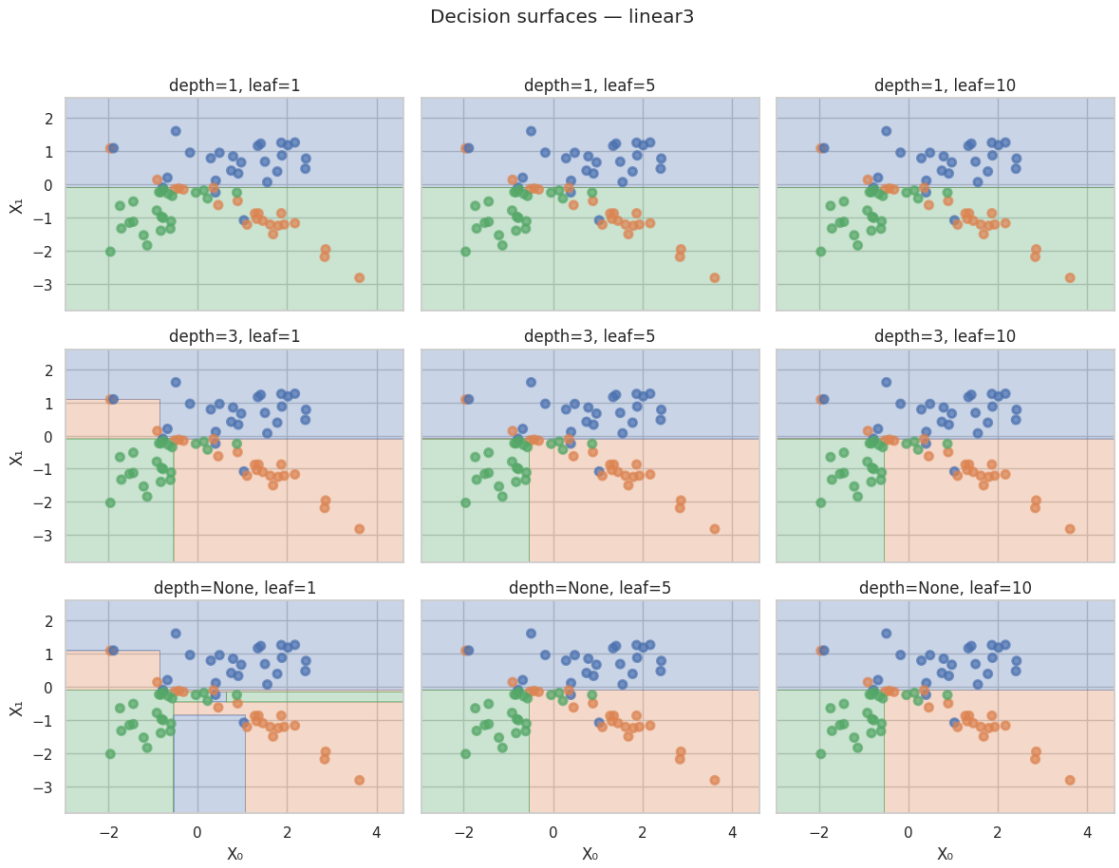
[12]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import pandas as pd

all_rows = []
for name, (X_tr, X_te, y_tr, y_te) in splits.items():
    for depth in param_grid['max_depth']:
        for leaf in param_grid['min_samples_leaf']:
            clf = DecisionTreeClassifier(
                max_depth=depth,
                min_samples_leaf=leaf,
                random_state=42
            )
            clf.fit(X_tr, y_tr)
            all_rows.append({
                'dataset':          name,
                'max_depth':        depth,
                'min_samples_leaf':leaf,
                'acc_train':        accuracy_score(y_tr, clf.predict(X_tr)),
                'acc_test':         accuracy_score(y_te, clf.predict(X_te))
            })

df_all = pd.DataFrame(all_rows)
```

[13]:
```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier


for name, (X_tr, X_te, y_tr, y_te) in splits.items():
    n_d = len(param_grid['max_depth'])
    n_l = len(param_grid['min_samples_leaf'])
    fig, axes = plt.subplots(n_d, n_l, figsize=(4*n_l, 3*n_d), sharex=True,␣
 ↪sharey=True)

    for i, depth in enumerate(param_grid['max_depth']):
        for j, leaf in enumerate(param_grid['min_samples_leaf']):
```

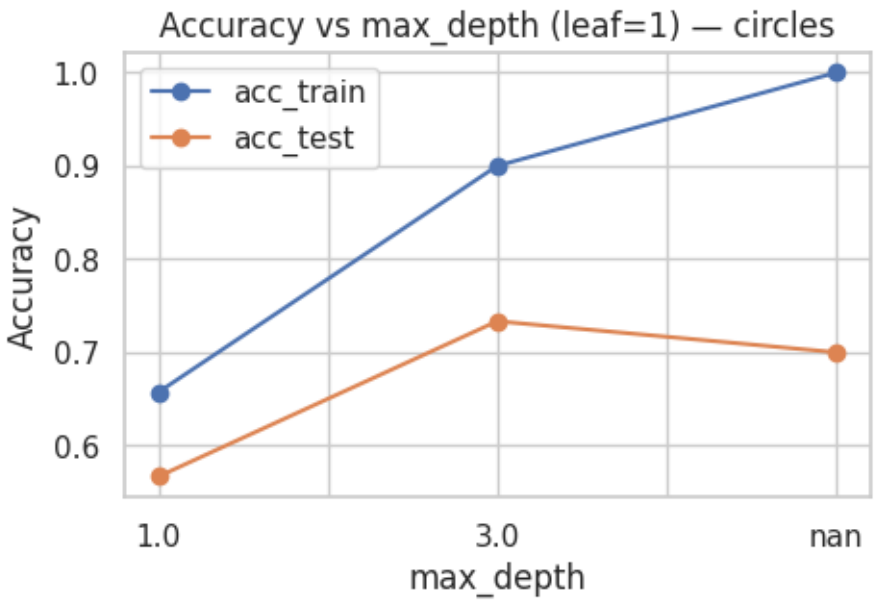
```
            clf = DecisionTreeClassifier(
                max_depth=depth,
                min_samples_leaf=leaf,
                random_state=42
            )
            clf.fit(X_tr, y_tr)

            ax = axes[i, j]
            plt.sca(ax)
            plot_surface(clf, X_tr, y_tr)
            ax.set_title(f"depth={depth}, leaf={leaf}")
            if i == n_d - 1: ax.set_xlabel("X ")
            if j == 0:       ax.set_ylabel("X ")

    fig.suptitle(f"Decision surfaces - {name}", y=1.02)
    plt.tight_layout()
    plt.show()
```

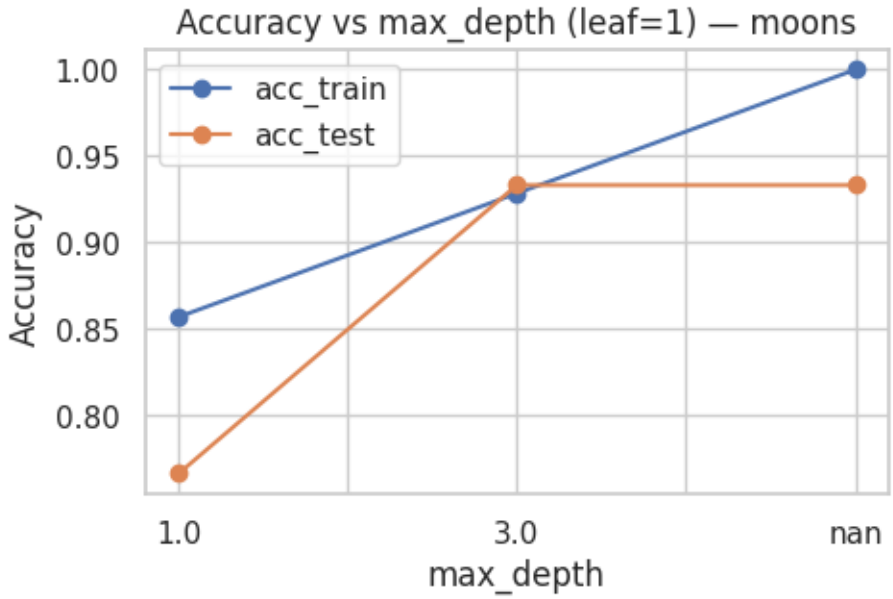


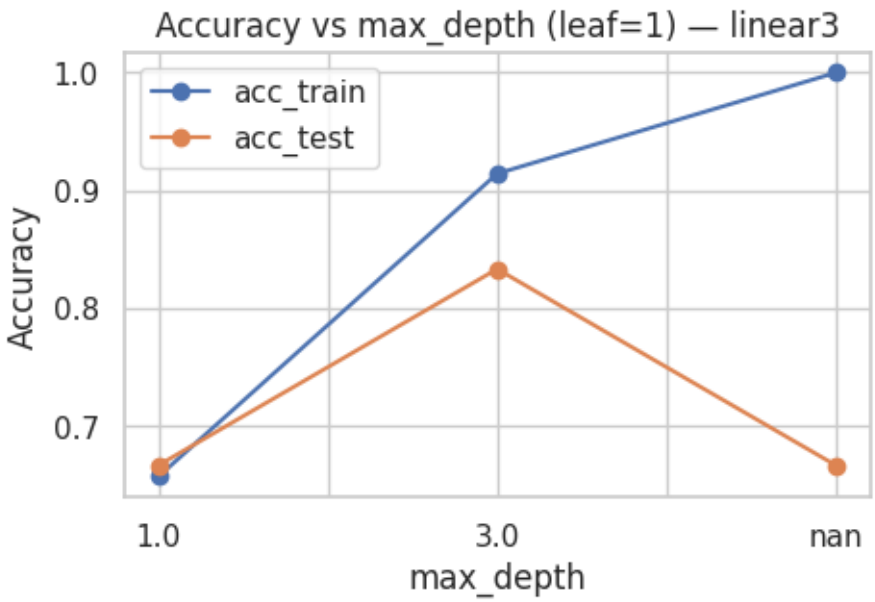

Decision surfaces — moons
depth=1, leaf=1
depth=1, leaf=5
depth=1, leaf=10
depth=3, leaf=1
depth=3, leaf=5
depth=3, leaf=10
depth=None, leaf=1
depth=None, leaf=5
depth=None, leaf=10
X₁
X₀



[14]:
```
import matplotlib.pyplot as plt

for name in df_all['dataset'].unique():
    df = df_all[(df_all['dataset'] == name) & (df_all['min_samples_leaf'] ==␣
 ↪1)].copy()
    df['max_depth_str'] = df['max_depth'].astype(str)

    df.plot(
        x='max_depth_str',
        y=['acc_train','acc_test'],
        marker='o',
        figsize=(5,3),
        title=f"Accuracy vs max_depth (leaf=1) - {name}"
    )
    plt.xlabel("max_depth")
    plt.ylabel("Accuracy")
    plt.grid(True)
    plt.show()
```

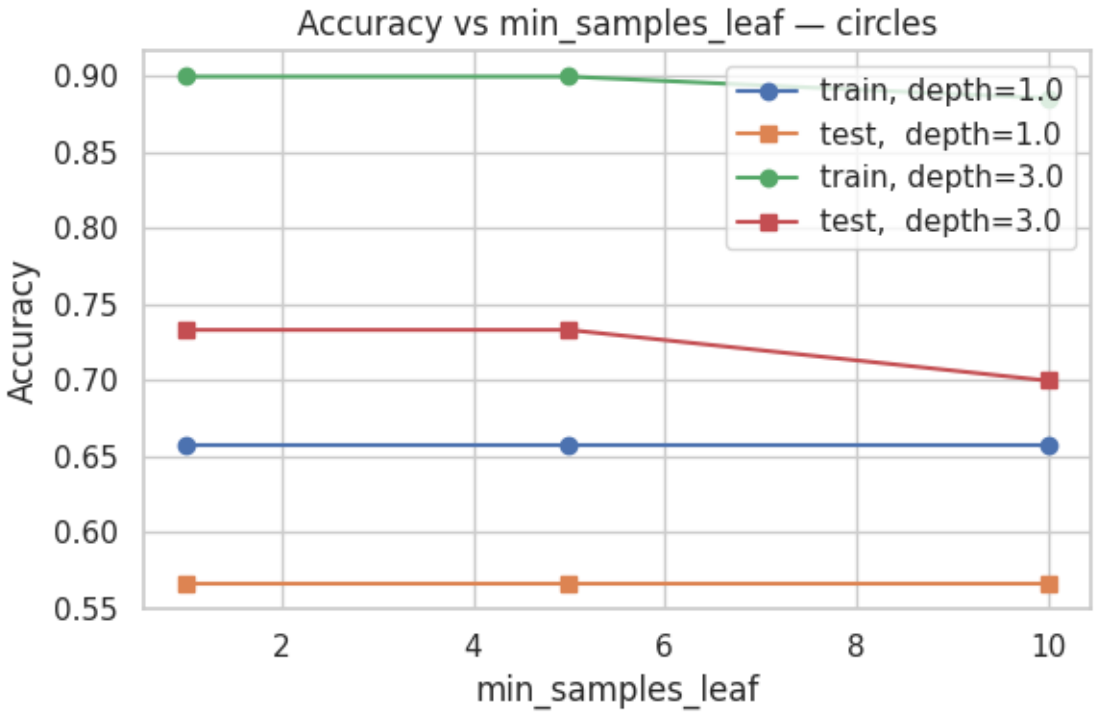

Accuracy vs max_depth (leaf=1) — circles
acc_train
acc_test
Accuracy
1.0
0.9
0.8
0.7
0.6
1.0
3.0
nan
max_depth

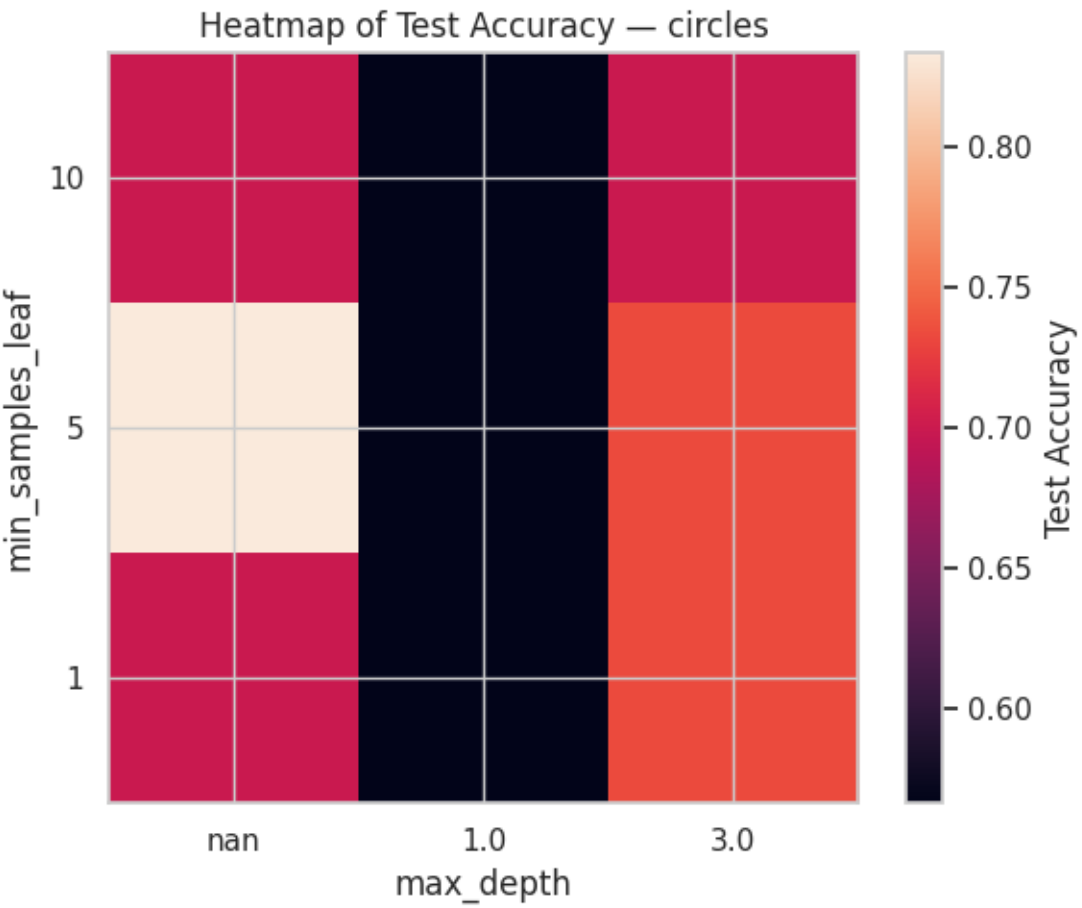

Accuracy vs max_depth (leaf=1) — moons
acc_train
acc_test
Accuracy
1.00
0.95
0.90
0.85
0.80
1.0
3.0
nan
max_depth

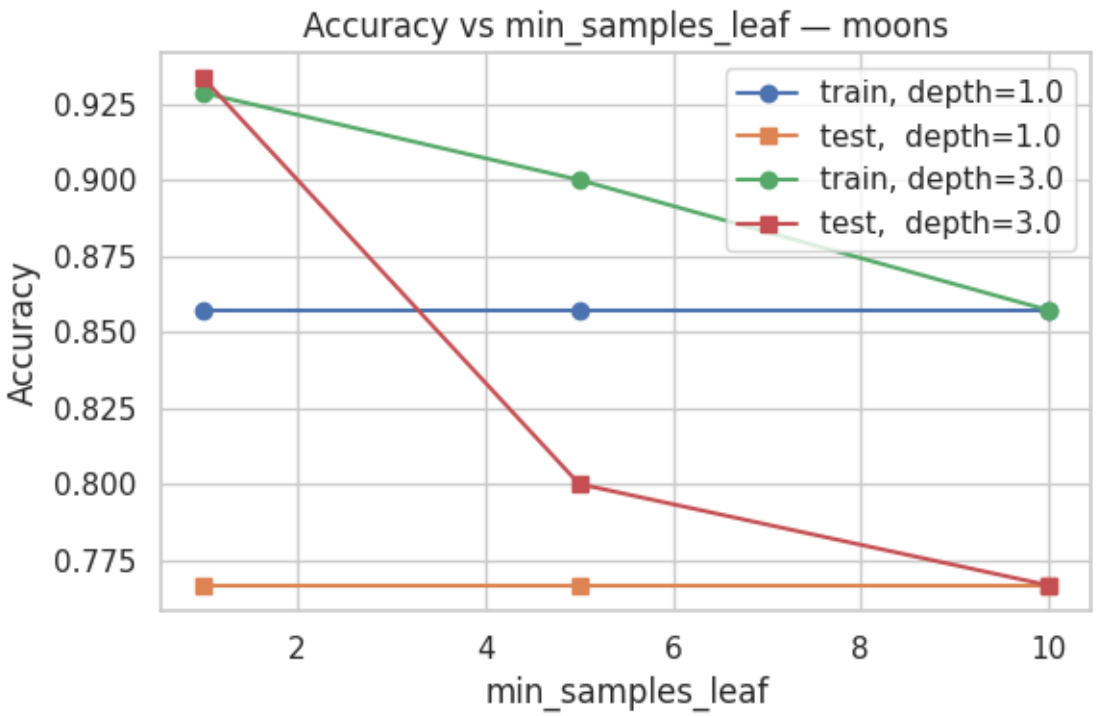


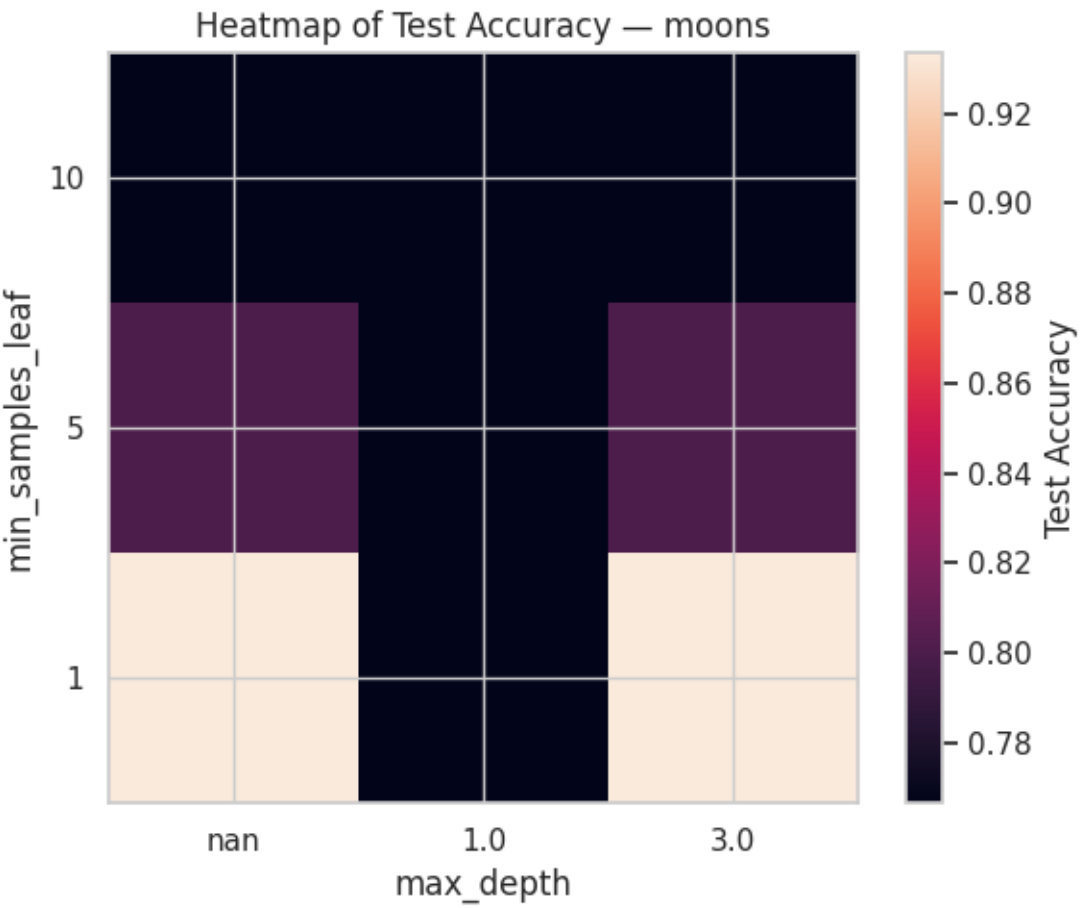
[15]:
```
import matplotlib.pyplot as plt

for name in df_all['dataset'].unique():
    df = df_all[df_all['dataset'] == name]

    plt.figure(figsize=(6,4))
    for depth in sorted(df['max_depth'].dropna().unique(), key=lambda x: (x is␣
 ↪None, x)):
        df_d = df[df['max_depth'] == depth]
        plt.plot(df_d['min_samples_leaf'], df_d['acc_train'],
                 marker='o', label=f"train, depth={depth}")
        plt.plot(df_d['min_samples_leaf'], df_d['acc_test'],
                 marker='s', label=f"test,  depth={depth}")
    plt.title(f"Accuracy vs min_samples_leaf - {name}")
    plt.xlabel("min_samples_leaf")
    plt.ylabel("Accuracy")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    pivot = df.pivot(index='min_samples_leaf', columns='max_depth',␣
 ↪values='acc_test')
    plt.figure(figsize=(6,5))
    plt.imshow(pivot.values, aspect='auto', origin='lower')
    plt.colorbar(label='Test Accuracy')
```

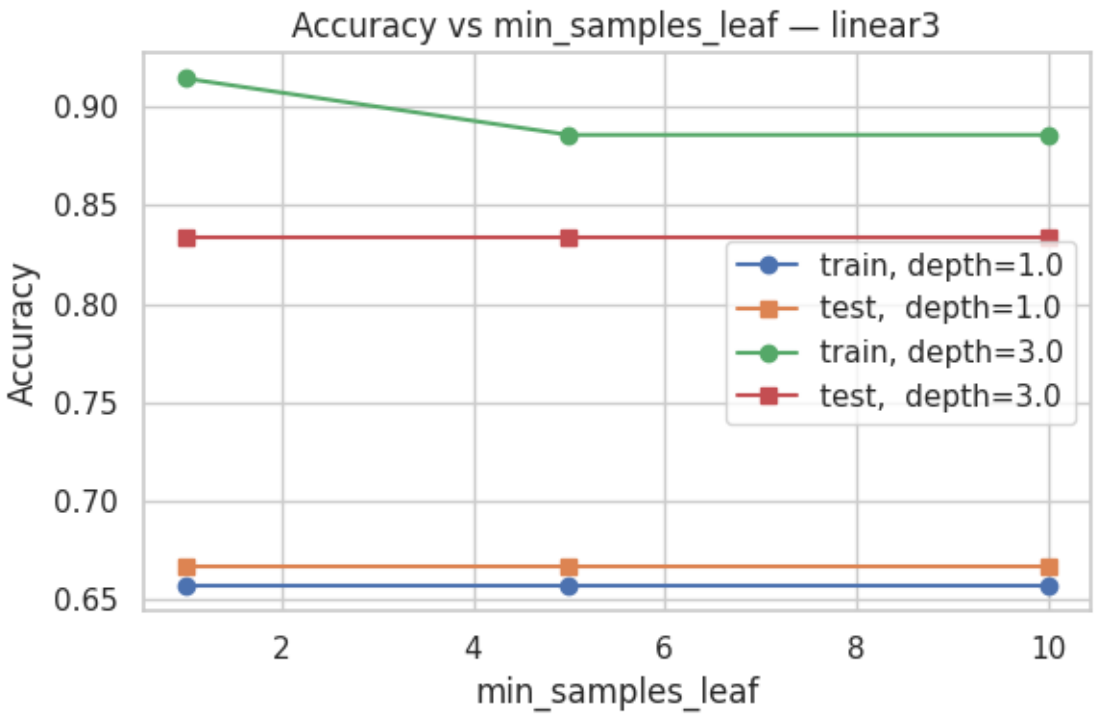
```
    plt.xticks(range(len(pivot.columns)), pivot.columns)
    plt.yticks(range(len(pivot.index)), pivot.index)
    plt.xlabel('max_depth')
    plt.ylabel('min_samples_leaf')
    plt.title(f"Heatmap of Test Accuracy - {name}")
    plt.tight_layout()
    plt.show()
```

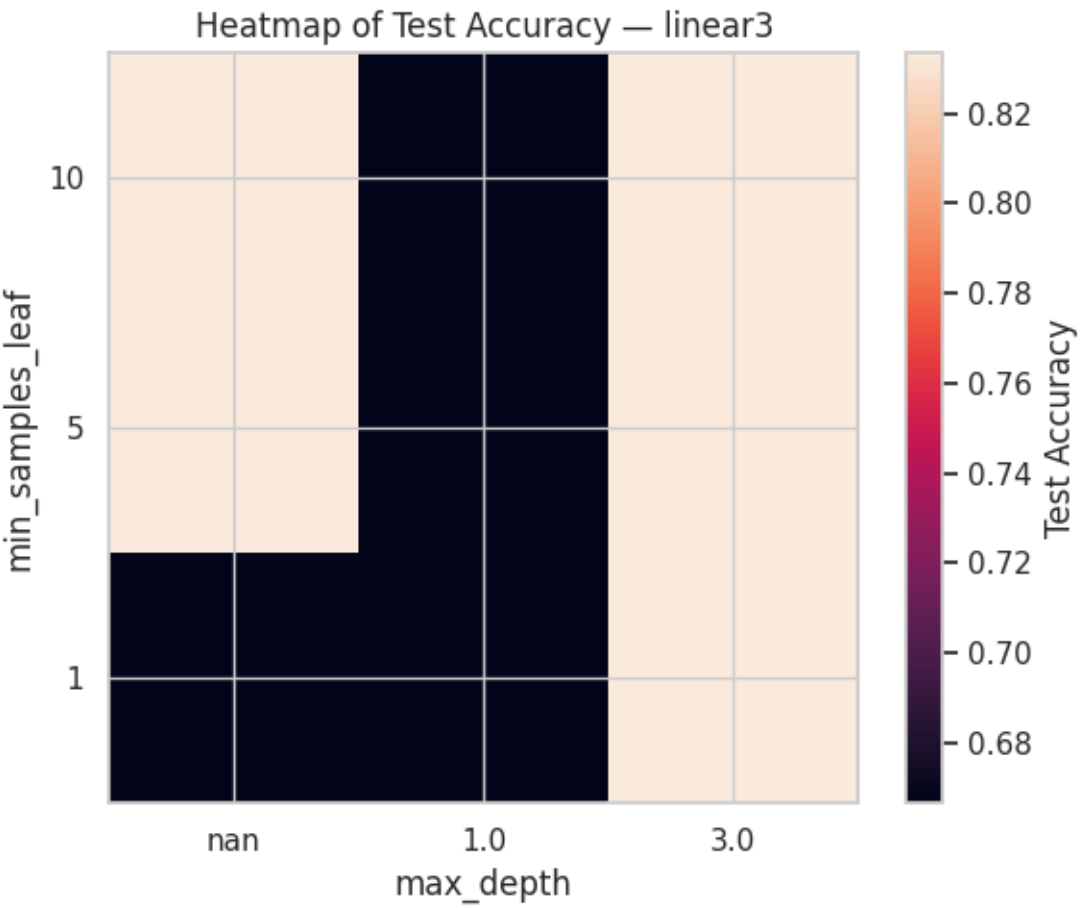



Heatmap of Test Accuracy — circles
min_samples_leaf
10
5
1
nan
1.0
3.0
max_depth
0.80
0.75
0.70
0.65
0.60
Test Accuracy


Accuracy vs min_samples_leaf — moons
train, depth=1.0
test, depth=1.0
train, depth=3.0
test, depth=3.0
Accuracy
0.925
0.900
0.875
0.850
0.825
0.800
0.775
2
4
6
8
10
min_samples_leaf


Heatmap of Test Accuracy — moons
min_samples_leaf
10
5
1
nan
1.0
3.0
max_depth
Test Accuracy
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78


Accuracy vs min_samples_leaf — linear3
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
2
4
6
8
10
min_samples_leaf
train, depth=1.0
test, depth=1.0
train, depth=3.0
test, depth=3.0



:

: 3 . ,
100%, .

: 5 . , ,
. **moon**, 5 , depth=3.
.

: - , --- , .
- , , 1. , - .
. - linear, ,
--- .

3 2.

.

3. (1.5)

find_best_split hw2code.py

Done

4. (0.5)

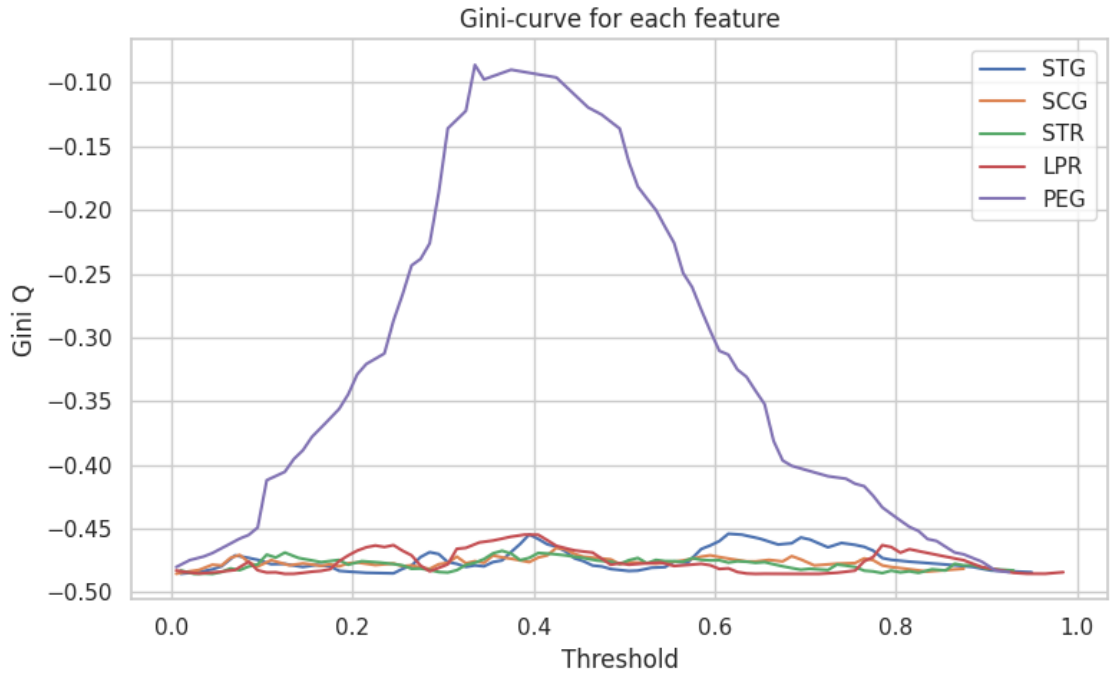
students.csv (User Knowledge).

, (: 0 1). `` ---

'' . scatter- `` --- '' .

[16]:
```
### ( °  ° )  *:
```

[17]:
```
import pandas as pd
import matplotlib.pyplot as plt
from hw2code import find_best_split

df = pd.read_csv("datasets/students.csv")
X = df.iloc[:, :5].values
y = df.iloc[:, 5].values
feature_names = df.columns[:5]

fig, ax = plt.subplots(figsize=(8, 5))

for i, name in enumerate(feature_names):
    feature_vector = X[:, i]

    thresholds, ginis, thresh_best, gini_best = find_best_split(feature_vector,␣
 ↪y)

    ax.plot(thresholds, ginis, label=name)

ax.set_xlabel("Threshold")
ax.set_ylabel("Gini Q")
ax.set_title("Gini-curve for each feature")
ax.legend()
plt.tight_layout()
plt.show()

fig2, axes = plt.subplots(1, 5, figsize=(15, 3), sharey=True)
for i, name in enumerate(feature_names):
    axes[i].scatter(X[:, i], y, alpha=0.6)
    axes[i].set_xlabel(name)
    if i == 0:
        axes[i].set_ylabel("Class (0 or 1)")
fig2.suptitle("Feature values vs Class")
plt.tight_layout()
plt.show()
```



PEG, , scatter- - 0.35

5. (0.5)

, ?

scatter- ? `` '' , ?

, ?

:

1-2. PEG ~0.35 . scatter.

, , Gini- .

3. .

4. .

6. (1.5).

DecisionTree hw2code.py. _fit_node.
_predict_node.

, .
. : . :
. , « ».

Done

7. (0.5)

mushrooms. agaricus-lepiota.data (
), pandas, LabelEncoder (sklearn),
. --- (e --- edible, p --- poisonous)
accuracy, , 1, --- 0.
( ) . accuracy.

accuracy, ( ), .

[18]:
```
### ( °  ° )  *:
```

[19]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from hw2code import DecisionTree

df = pd.read_csv("datasets/agaricus-lepiota.data", header=None)

encoders = {}
for col in df.columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    encoders[col] = le

y = df.iloc[:, 0].values
X = df.iloc[:, 1:].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,␣
 ↪stratify=y, random_state=42)

feature_types = ["categorical"] * X.shape[1]
tree = DecisionTree(feature_types)
tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy on mushrooms: {accuracy:.3f}")
```

```
def tree_depth(node):
    if node["type"] == "terminal":
        return 0
    left_depth  = tree_depth(node["left_child"])
    right_depth = tree_depth(node["right_child"])
    return 1 + max(left_depth, right_depth)

depth = tree_depth(tree._tree)
print(f"Depth of the mushroom tree: {depth}")
```

```
Accuracy on mushrooms: 1.000
Depth of the mushroom tree: 7
```

8. (, 1)

DecisionTree max_depth, min_samples_split min_samples_leaf DecisionTreeClassifier.
tic-tac-toe (.).

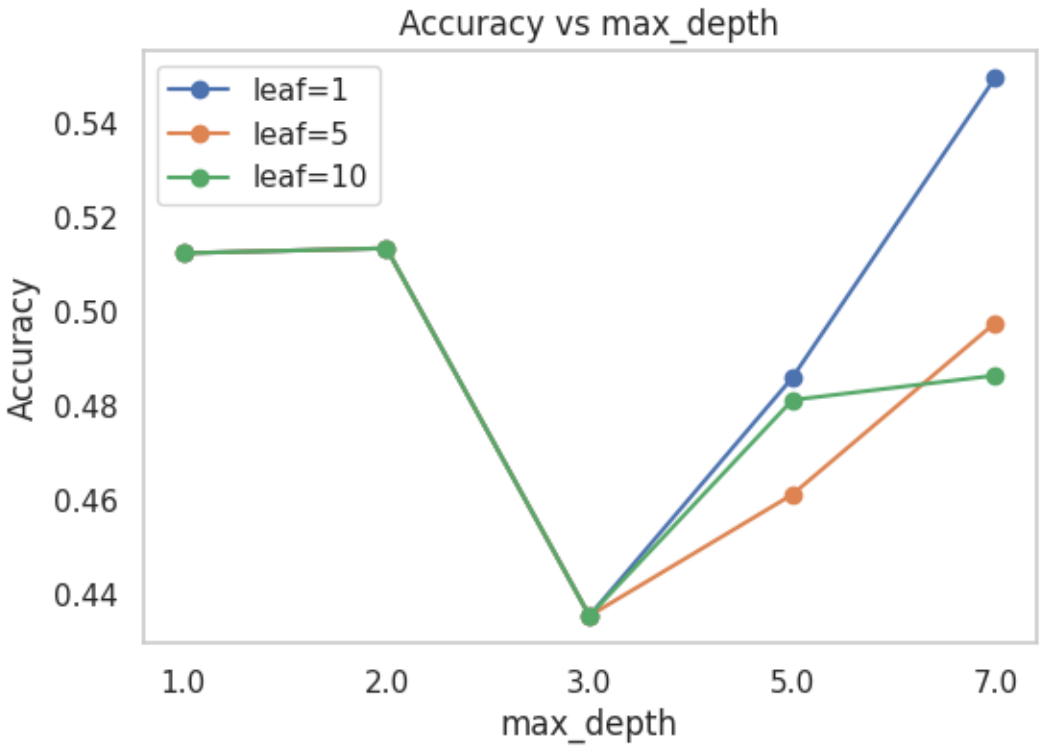
[20]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("datasets/tic-tac-toe-endgame.csv", header=None)

X_ttt = df.iloc[:, :-1]
y_ttt = df.iloc[:, -1]


X_ttt = X_ttt.apply(LabelEncoder().fit_transform)

le_target = LabelEncoder()
y_ttt = le_target.fit_transform(y_ttt)
```

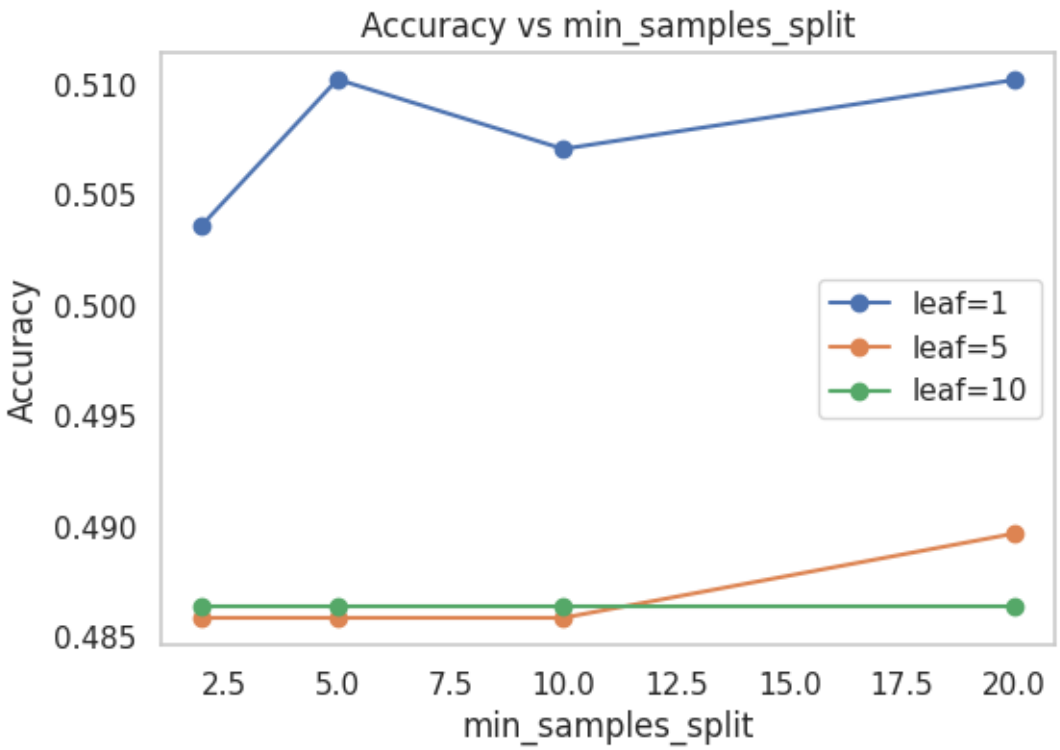
[21]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score

param_grid = {
    'max_depth':        [None, 1, 2, 3, 5, 7],
    'min_samples_split':[2, 5, 10, 20],
    'min_samples_leaf': [1, 5, 10, 20]
}

results = []

for md in param_grid['max_depth']:
    for mss in param_grid['min_samples_split']:
        for msl in param_grid['min_samples_leaf']:
            tree = DecisionTree(feature_types,
```

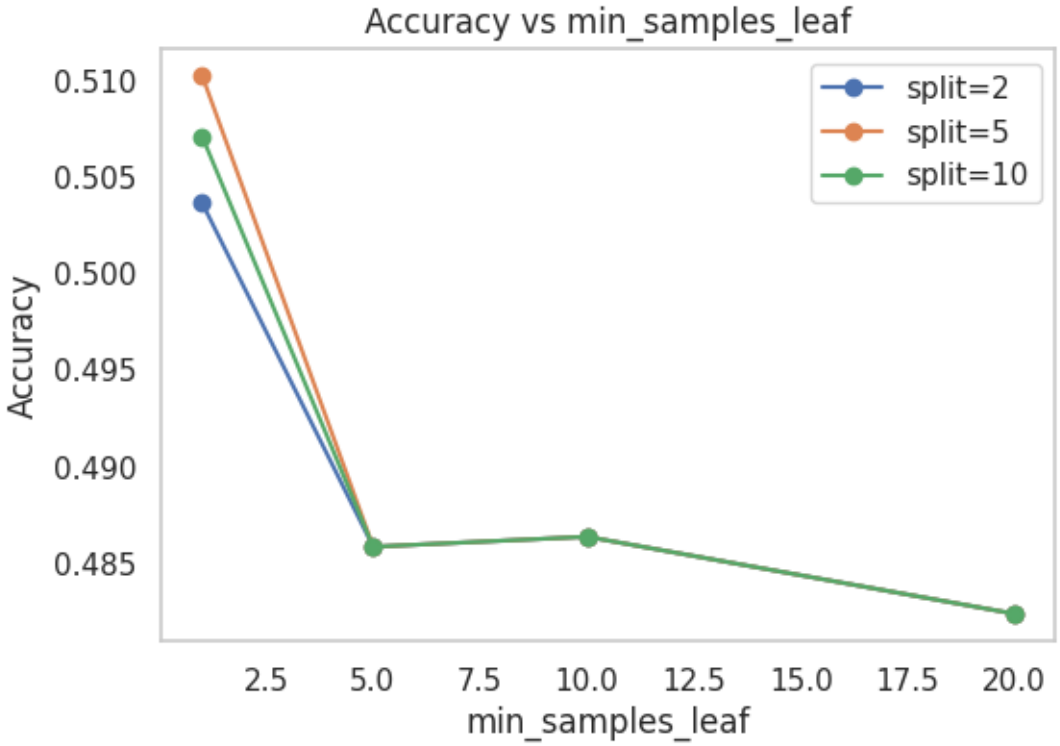
```
                                max_depth=md,
                                min_samples_split=mss,
                                min_samples_leaf=msl)
            scores = cross_val_score(
                tree,
                X_ttt.values,
                y_ttt,
                cv=5,
                scoring='accuracy'
            )
            results.append({
                'max_depth': md,
                'min_samples_split': mss,
                'min_samples_leaf': msl,
                'accuracy_mean': scores.mean()
            })

df_res = pd.DataFrame(results)
```

[22]:
```
plt.figure(figsize=(6,4))
for msl in [1,5,10]:
    sub = df_res[df_res['min_samples_leaf']==msl]
    means = sub.groupby('max_depth')['accuracy_mean'].mean()
    plt.plot(means.index.astype(str), means.values, marker='o',␣
 ↪label=f"leaf={msl}")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Accuracy vs max_depth")
plt.grid()
plt.show()

plt.figure(figsize=(6,4))
for msl in [1,5,10]:
    sub = df_res[df_res['min_samples_leaf']==msl]
    means = sub.groupby('min_samples_split')['accuracy_mean'].mean()
    plt.plot(means.index, means.values, marker='o', label=f"leaf={msl}")
plt.xlabel("min_samples_split")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Accuracy vs min_samples_split")
plt.grid()
plt.show()

plt.figure(figsize=(6,4))
for mss in [2,5,10]:
    sub = df_res[df_res['min_samples_split']==mss]
```

```
    means = sub.groupby('min_samples_leaf')['accuracy_mean'].mean()
    plt.plot(means.index, means.values, marker='o', label=f"split={mss}")
plt.xlabel("min_samples_leaf")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Accuracy vs min_samples_leaf")
plt.grid()
plt.show()
```




Accuracy vs min_samples_split
0.510
0.505
0.500
0.495
0.490
0.485
Accuracy
leaf=1
leaf=5
leaf=10
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
min_samples_split


Accuracy vs min_samples_leaf
0.510
0.505
0.500
0.495
0.490
0.485
Accuracy
split=2
split=5
split=10
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
min_samples_leaf

: --- 7 .

, min_samples_leaf 1--2.

min_samples_split , .

, accuracy (~0.55) , .

9. (2)

(, pandas url, *.data),

(Data Folder, *.names):* mushrooms (,) * tic-tac-toe (,) * cars (, unacc, acc --- 0, good, vgood --- 1) * nursery (, , not_recom recommend --- 0, very_recom, priority, spec_prior --- 1).

, LabelEncoder. cross_val_score (cv=10) accuracy : * DecisionTree, * DecisionTree, * DecisionTree, + one-hot-encoding * DecisionTreeClassifier sklearn. pd.DataFrame (--- , ---).

: * cross_val_score , scoring=make_scorer(accuracy_score), sklearn.metrics. * ( ), sparse=False OneHotEncoder (, ,). , (, ,).

[23]:
```
### ( °  ° )  *:
```

[25]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.tree import DecisionTreeClassifier
from hw2code import DecisionTree

all_data = {}

df = pd.read_csv("datasets/agaricus-lepiota.data", header=None)
for col in df.columns:
    df[col] = LabelEncoder().fit_transform(df[col])
X = df.iloc[:,1:].values
y = df.iloc[:,0].values
all_data['mushrooms'] = (X, y)

df = pd.read_csv("datasets/tic-tac-toe-endgame.csv", header=None)
```

```
X = df.iloc[:,:-1].apply(LabelEncoder().fit_transform).values
y = LabelEncoder().fit_transform(df.iloc[:,-1].values)
all_data['tic-tac-toe'] = (X, y)

url_cars = "https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.
 ↪data"
cols = ["buying","maint","doors","persons","lug_boot","safety","class"]
df = pd.read_csv(url_cars, names=cols)

df['class'] = df['class'].map(lambda v: 0 if v in ('unacc','acc') else 1)
X = df.iloc[:,:-1].apply(LabelEncoder().fit_transform).values
y = df['class'].values
all_data['cars'] = (X,y)

url_nursery = "https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪nursery/nursery.data"
cols = ["parents","has_nurs","form","children","housing","finance",
        "social","health","class"]
df = pd.read_csv(url_nursery, names=cols)

df['class'] = df['class'].map(lambda v: 0 if v in ('not_recom','recommend')␣
 ↪else 1)
X = df.iloc[:,:-1].apply(LabelEncoder().fit_transform).values
y = df['class'].values
all_data['nursery'] = (X, y)

scorer = make_scorer(accuracy_score)

def make_algorithms(X,y):
    n_feat = X.shape[1]
    algos = {
        "DT_real":          DecisionTree(["real"]*n_feat),
        "DT_cat":           DecisionTree(["categorical"]*n_feat),
        "DT_real_ohe":      None,
        "SK_DT":            DecisionTreeClassifier(random_state=42)
    }
    ohe = OneHotEncoder(sparse_output=False)
    X_ohe = ohe.fit_transform(X)
    algos["DT_real_ohe"] = DecisionTree(["real"]*X_ohe.shape[1])
    return algos, X_ohe

results = []
for name, (X,y) in all_data.items():
    algos, X_ohe = make_algorithms(X,y)
    for alg_name, alg in algos.items():
        X_use = X_ohe if alg_name == 'DT_real_ohe' else X
        scores = cross_val_score(alg, X_use, y, cv=10, scoring=scorer)
```

```
        results.append({
            "dataset": name,
            "algorithm": alg_name,
            "accuracy": scores.mean(),
        })

df_res = pd.DataFrame(results)

table = df_res.pivot_table(
    index="dataset",
    columns="algorithm",
    values="accuracy"
).round(3)

print(table)
```

```
algorithm    DT_cat  DT_real  DT_real_ohe  SK_DT
dataset
cars          0.968    0.943        0.919  0.943
mushrooms     1.000    0.999        1.000  0.960
nursery       1.000    1.000        1.000  1.000
tic-tac-toe   0.601    0.497        0.595  0.785
```

10. (1)

. ? , .

. - ?

?

:

mushrooms nursery , 1.0. ,

. , .

cars . .

One-hot , .

tic-tac-toe sklearn- . -

.

. .

.

: , sklearn

.

, random_state shuffle - .

, , .

, - . , ,
.