For immersion: https://www.youtube.com/watch?v=sdBrscwwy_c

# 🌟 Operation NeuroNexus

Trondheim lies under the iron grip of SkyNet, an AI system that has seized control of the city's entire digital infrastructure. You and your team of elite hackers have been tasked with a crucial mission: infiltrate SkyNet's systems, decode its defenses, and liberate the city from its digital oppressor.

## 🎯 Mission Overview

Operation NeuroNexus consists of four independent, yet interconnected missions. Each mission targets a different aspect of SkyNet's infrastructure and requires you to apply various Supervised Learning techniques covered in this course.

## 📊 Mission Structure

1. Each mission has a specific task related to combating SkyNet.
2. Following the task description, you'll find a set of formal requirements that your solution must meet.
3. The primary measure of your success is the accuracy of your machine learning model.
4. After completing each task, you must answer a series of questions to demonstrate your understanding of the techniques used.

## 🧪 A Note on Test Data

In a departure from real-world scenarios, you will have access to the target variables of the test sets for each mission. This has been arranged to facilitate the evaluation of your models. However, remember that in actual machine learning projects, test targets are not available, as predicting these is the ultimate goal of your supervised models.

## 📝 Submission Guidelines

- For each mission, provide your code solution and model results inside this notebook.
- Answer the follow-up questions in markdown format within this notebook. A few sentences is enough, no requirements for length of answers.

- Ensure your explanations are clear, concise, and demonstrate a deep understanding of the techniques employed.

Good luck! The resistance is counting on you.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from linear_regression import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.tree import export_text
from sklearn.metrics import roc_curve, roc_auc_score
from logistic_regression import LogisticRegression
from mpl_toolkits.mplot3d import Axes3D
from sklearn import tree
import sklearn.metrics as metrics
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_log_error
from catboost import CatBoostRegressor
from sklearn.ensemble import RandomForestClassifier
```

# 🌞 Mission 1: Predicting SkyNet's Power Consumption

## 🎯 The Mission

Intelligence suggests that SkyNet has a critical weakness: **its power consumption**. We must understand its energy needs to plan a coordinated strike.

## 🧠 Your Task

Develop a predictive model to estimate SkyNet's power consumption based on its **Network Activity**.

**Goal**: Implement a **Linear Regression model using Gradient Descent, from scratch**.

Use `LinearRegression` class from `linear_regression.py` stored in this folder. Your task is to complete two functions: `fit` (find the optimal parameters of the regression) and `predict` (apply them to the test data).

> Note: The `%autoreload` IPython magic command allows instant updates from `linear_regression.py`.

## 📊 Formal Requirements

1. **Implementation**:

   - Use standard Python libraries (numpy, math, pandas, etc.)
   - Implement gradient descent

2. **Discussion**:

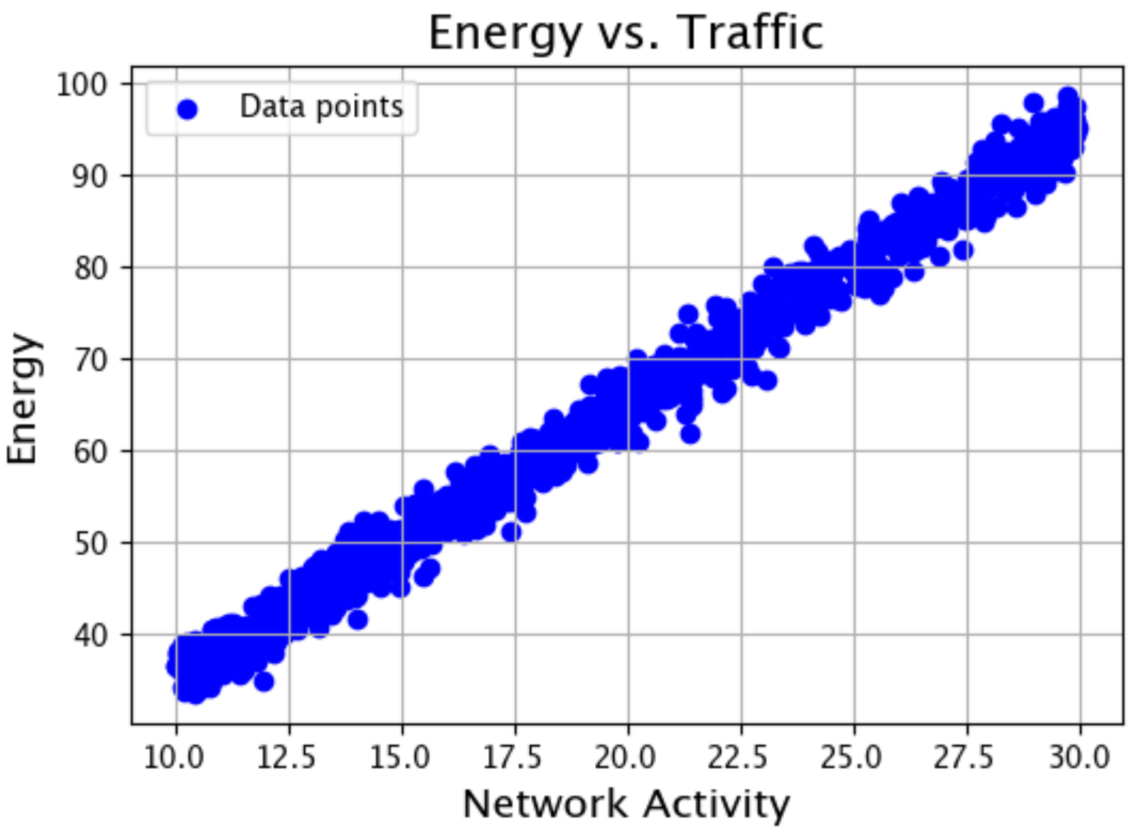
   a. Visualize the fitted curve. Derive the resulting Energy consumption formula.

   b. Analyze prediction error distribution. What is an unbiased estimator?

---

In [4]:
```
%load_ext autoreload
%autoreload 2
```

In [5]:
```
# Load data
data = pd.read_csv('mission1.csv')

plt.figure(figsize=(6, 4))
plt.scatter(data['Net_Activity'], data['Energy'], c='blue', label='
plt.grid(True)
plt.xlabel('Network Activity', fontsize=14)
plt.ylabel('Energy', fontsize=14)
plt.title('Energy vs. Traffic', fontsize=16)
plt.legend()
plt.show()
```



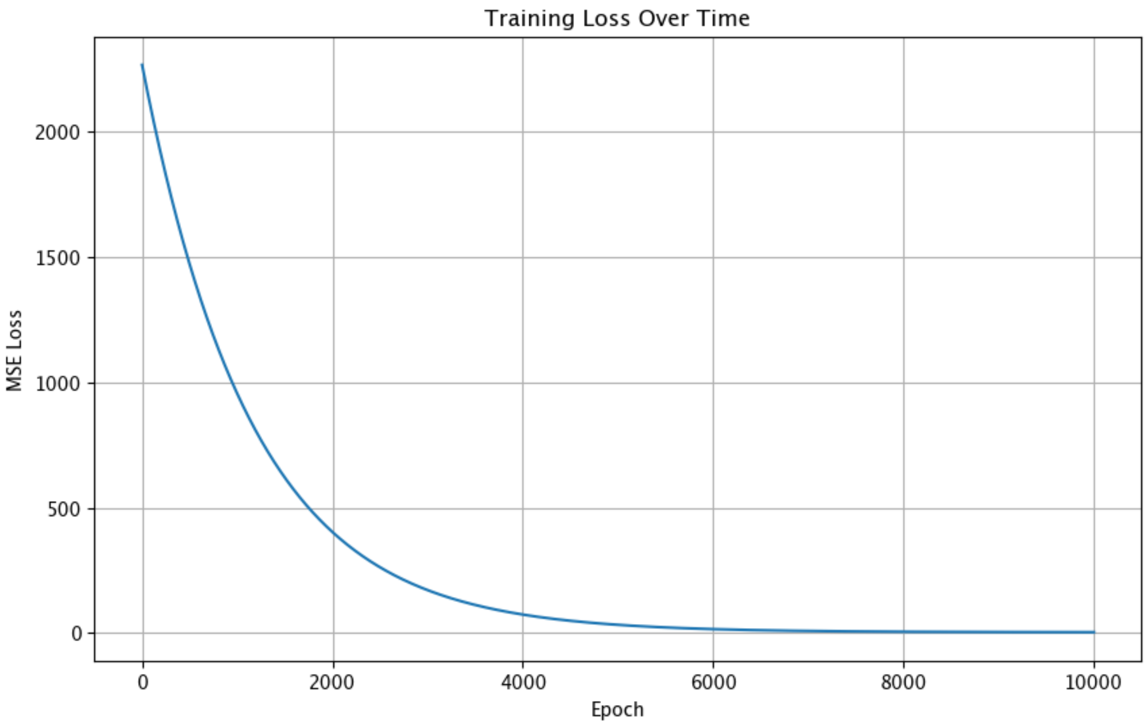

In [6]:
```
lr = LinearRegression()
X = data['Net_Activity'].values.reshape(-1, 1)
y = data['Energy']
```

```python
# Split data i 80% trening og 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

lr.fit(X_train, y_train)

predictions = lr.predict(X_test)

mse = np.mean((predictions - y_test) ** 2)
print(f'MSE: {mse:.4f}')
rmse = np.sqrt(np.mean((predictions - y_test) ** 2))
print(f'RMSE: {rmse:.4f}')
```

```
MSE: 5.6748
RMSE: 2.3822
```

In [7]:

```python
plt.figure(figsize=(10, 6))
plt.plot(lr.losses)
plt.title('Training Loss Over Time')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.grid(True)
plt.show()

print(f'Initial loss: {lr.losses[0]:.4f}')
print(f'Final loss: {lr.losses[-1]:.4f}')
print(f'Loss reduction: {((lr.losses[0] - lr.losses[-1]) / lr.losse
```

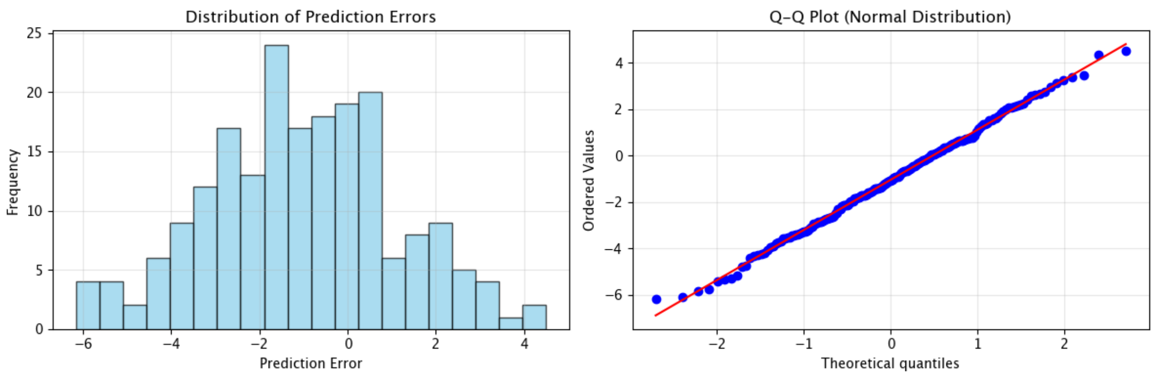



```
Initial loss: 2267.0520
Final loss: 3.2880
Loss reduction: 99.85%
```

In [ ]:

```python
errors = predictions - y_test
```

```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.hist(errors, bins=20, alpha=0.7, color='skyblue', edgecolor='bl
plt.xlabel('Prediction Error')
plt.ylabel('Frequency')
plt.title('Distribution of Prediction Errors')
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
from scipy import stats
stats.probplot(errors, dist="norm", plot=plt)
plt.title('Q-Q Plot (Normal Distribution)')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f'Error statistics:')
print(f'Mean error: {np.mean(errors):.4f}')
print(f'Std error: {np.std(errors):.4f}')
print(f'Error range: [{np.min(errors):.4f}, {np.max(errors):.4f}]')
```




```
Error statistics:
Mean error: -1.0413
Std error: 2.1425
Error range: [-6.1575, 4.5013]
```

# 🧠 Mission 2: Decoding SkyNet's signals

## 🌐 The Discovery

We've intercepted two types of signals that may determine SkyNet's next moves.

## 🎯 Your Mission

1. Evolve your linear regression into logistic regression
2. Engineer features to unravel hidden connections
3. Predict SkyNet's binary decisions (0 or 1) from paired signals

## 📊 Formal Requirements

1. **Implementation**:

    - Use standard Python libraries
    - Implement gradient descent
2. **Performance**: Achieve at least 0.88 accuracy on the test set

3. **Discussion**:

    a. Explain poor initial performance and your improvements

    b. What is the model's inductive bias. Why is it important?

    c. Try to solve the problem using `sklearn.tree.DecisionTreeClassifier`. Can it solve the problem? Why/Why not?

    d. Plot the ROC curve

---

In [12]:
```
data = pd.read_csv('mission2.csv')
train = data[data['split'] == 'train']
test = data[data['split'] == 'test']
```

In [13]:
```
X_train = train.iloc[:, 0:2].values
y_train = train.iloc[:, 2].values

X_test = test.iloc[:, 0:2].values
y_test = test.iloc[:, 2].values
```

In [15]:
```
clf = tree.DecisionTreeClassifier(criterion='entropy',
                                  max_depth=6,
                                  min_samples_leaf=2,
                                  min_samples_split=2,
                                  random_state=50,
                                  )


clf.fit(X_train, y_train)
clfPred = clf.predict(X_test)

print(f'Accuracy: {np.mean(sum((y_test, clfPred)))}')
```

```
Accuracy: 0.946
```

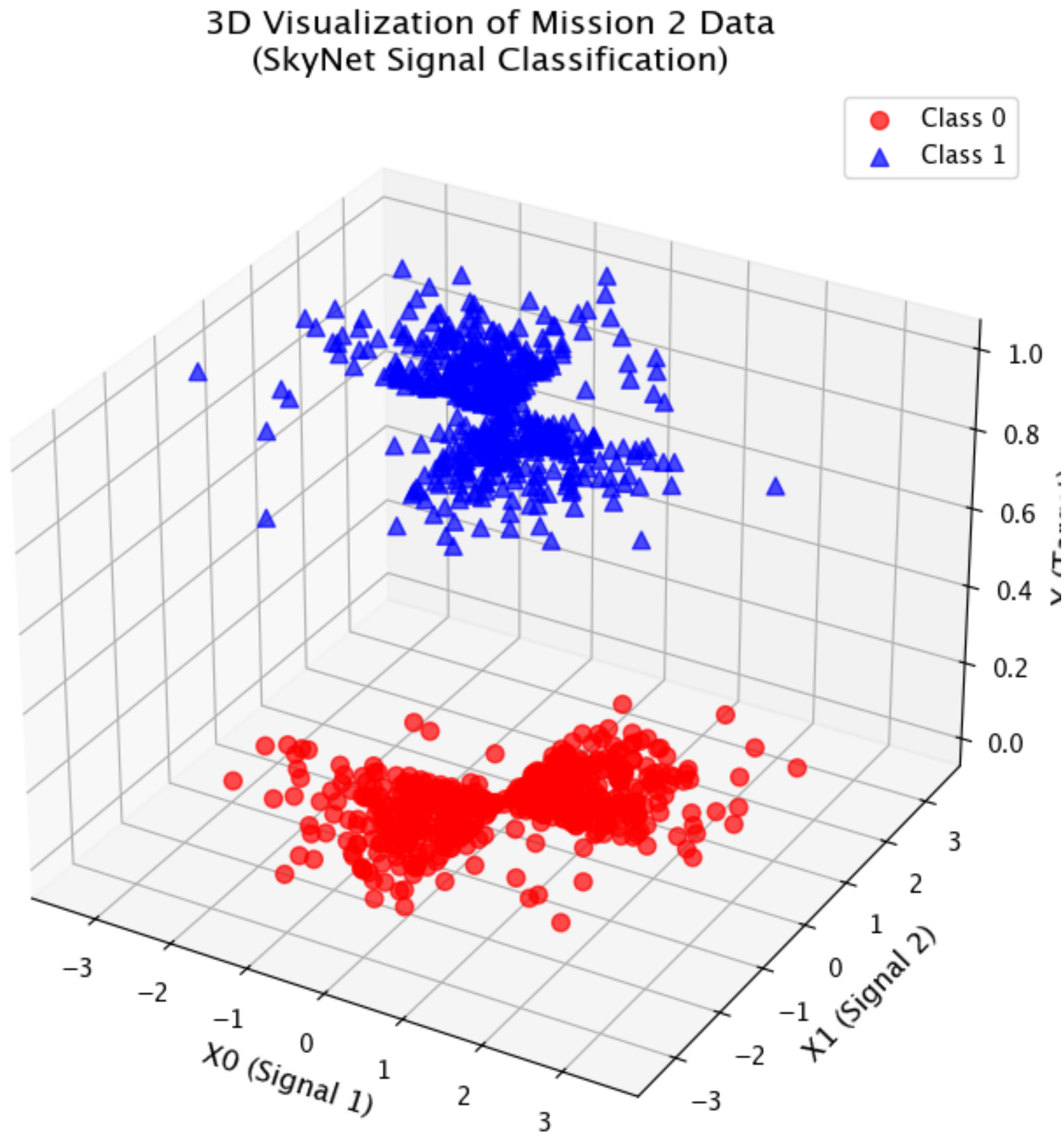
In [16]:
```
# Featuer engineering
X_train_eng = np.column_stack([
    X_train,
    X_train[:, 0]**2,
    X_train[:, 1]**2,
    X_train[:, 0] * X_train[:, 1]
])

X_test_eng = np.column_stack([
    X_test,
```

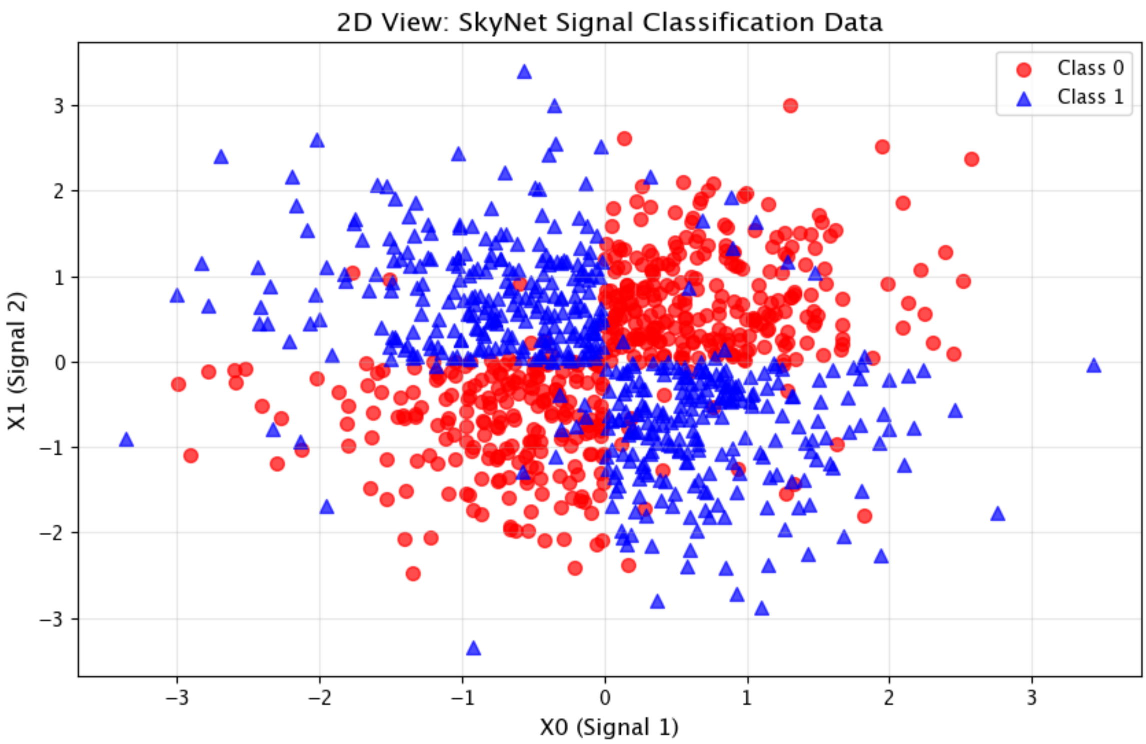
```
    X_test[:, 0] ** 2,
    X_test[:, 1] ** 2,
    X_test[:, 0] * X_test[:, 1],
])
```

In [17]:
```
lr = LogisticRegression(learning_rate=0.001, epochs=10000)
lr.fit(X_train_eng, y_train)
predictions = lr.predict(X_test_eng)
accuracy = np.mean(predictions == y_test)

print(f'Accuracy: {accuracy * 100:.2f}%')
```

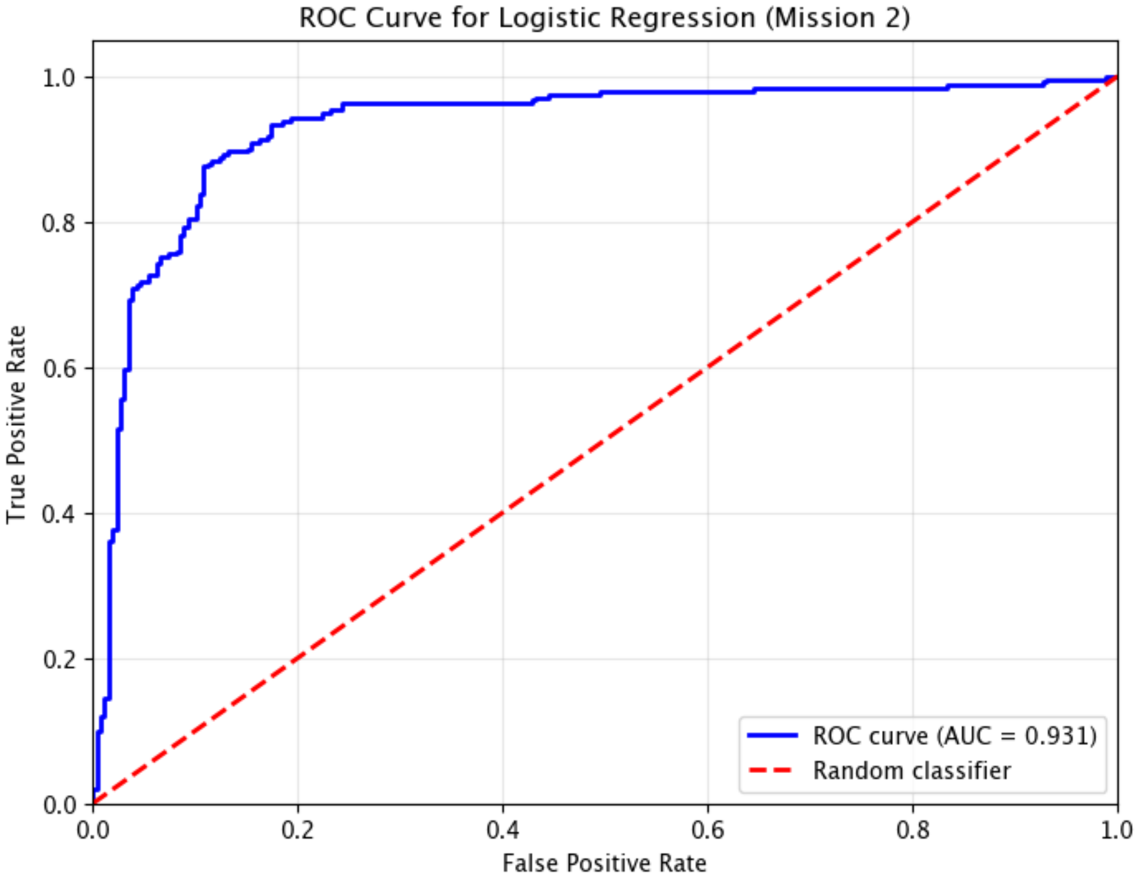
```
Accuracy: 88.40%
```

In [18]:
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Filtrer data basert på target klasse
class_0 = data[data['y'] == 0]
class_1 = data[data['y'] == 1]

# Plot begge klasser med forskjellige farger
ax.scatter(class_0['x0'], class_0['x1'], class_0['y'],
          c='red', marker='o', s=50, alpha=0.7, label='Class 0')
ax.scatter(class_1['x0'], class_1['x1'], class_1['y'],
          c='blue', marker='^', s=50, alpha=0.7, label='Class 1')

ax.set_xlabel('X0 (Signal 1)', fontsize=12)
ax.set_ylabel('X1 (Signal 2)', fontsize=12)
ax.set_zlabel('Y (Target)', fontsize=12)
ax.set_title('3D Visualization of Mission 2 Data\n(SkyNet Signal Cl
ax.legend()

# Legg til en 2D-visning også for bedre forståelse
plt.figure(figsize=(10, 6))
plt.scatter(class_0['x0'], class_0['x1'], c='red', marker='o', s=50
plt.scatter(class_1['x0'], class_1['x1'], c='blue', marker='^', s=5
plt.xlabel('X0 (Signal 1)', fontsize=12)
plt.ylabel('X1 (Signal 2)', fontsize=12)
plt.title('2D View: SkyNet Signal Classification Data', fontsize=14
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```


3D Visualization of Mission 2 Data
(SkyNet Signal Classification)
Class 0
Class 1
X0 (Signal 1)
X1 (Signal 2)
−3
−2
−1
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0


2D View: SkyNet Signal Classification Data
Class 0
Class 1
X1 (Signal 2)
X0 (Signal 1)
−3
−2
−1
0
1
2
3

In [ ]:
```
z = X_test_eng @ lr.weights + lr.bias
y_pred_proba = 1 / (1 + np.exp(-z))

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc_score = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {au
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression (Mission 2)')
plt.legend(loc="lower right")
plt.grid(True, alpha=0.3)
plt.show()

print(f"ROC AUC Score: {auc_score:.3f}")
```




```
ROC AUC Score: 0.931
```

In [478…
```
regressor = tree.DecisionTreeRegressor().fit(X_train_eng, y_train)
y_pred = regressor.predict(X_test_eng)
accuracy = np.mean(y_test == y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 88.40%
```

# 🌆 Mission 3: CyberGuard

## 🌐 The Discovery

SkyNet's drone communications use binary encryption. We need a system to intercept these messages.

## 🎯 Your Mission

Develop a decision tree classifier to process intercepted communications. Use `sklearn.tree.DecisionTreeClassifier`.

Only one of the data streams needs to be decrypted, but you will need to identify the correct one.

To decrypt a data stream, transform the data into a binary representation based on whether the feature is even or odd.

## 📊 Formal Requirements

1. **Accuracy**: Achieve ROC AUC >= 0.72 on the test set

2. **Discussion**:

   a. Explain your threshold-breaking strategy. Did you change the default hyperparameters?

   b. Justify ROC AUC usage. Plot and interpret ROC.

   c. Try to solve the problem using sklearn's Random Forest Classifier. Compare the results.

---

In [479…
```
def accuracy(trueValues, predictions):
    return np.mean(trueValues == predictions)
```

In [480…
```
train = pd.read_csv('mission3_train.csv')
test = pd.read_csv('mission3_test.csv')

train['data_stream_3'] = (train['data_stream_3'] * 1000 + 1) % 2
test['data_stream_3'] = (test['data_stream_3'] * 1000 + 1) % 2
```

In [481…
```
X_train = train.iloc[:, 0:11].values
y_train = train.iloc[:, 11].values

X_test = test.iloc[:, 0:11].values
y_test = test.iloc[:, 11].values
```

In [482…
```
clf = tree.DecisionTreeClassifier()
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = np.mean(y_test == y_pred)
```

```
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 73.85%
```

In [483…

```
a = (test["target"] == 1).sum()
b = (test["target"] == 0).sum()
n = len(test)
print(a)
print(b)
print(a/n)
print(b/n)
```

```
386
1614
0.193
0.807
```

In [484…

```
param_grid = {
    'max_depth': [1, 2, 6, 10, 25, None],
    'min_samples_split': [2, 6, 10, 20],
    'min_samples_leaf': [2, 5, 10, 20],
    'criterion': ['gini', 'entropy', 'log_loss']
}

clf = tree.DecisionTreeClassifier(random_state=42)
gridSearch = GridSearchCV(estimator=clf, param_grid=param_grid, cv=
gridSearch.fit(X_train, y_train)
bestParams = gridSearch.best_params_
bestScore = gridSearch.best_score_
print(f"Best parameters found: {bestParams}")
print(gridSearch.cv_results_)
best_model = gridSearch.best_estimator_
y_pred_proba = best_model.predict_proba(X_test)[:, 1]

# Calculate ROC AUC for the test set
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"Test ROC AUC Score: {roc_auc:.2f}")
print(f"Best ROC-auc Score: {bestScore}")
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Best parameters found: {'criterion': 'gini', 'max_depth': 6, 'min_sa
mples_leaf': 20, 'min_samples_split': 2}
{'mean_fit_time': array([0.00703716, 0.00606828, 0.0059885 , 0.00573
921, 0.00559435,
       0.00573745, 0.00607524, 0.00576544, 0.00561037, 0.00607548,
       0.00585337, 0.0057519 , 0.0058219 , 0.00580535, 0.00576196,
       0.00598159, 0.01117368, 0.01086993, 0.01066794, 0.01082501,
       0.0106204 , 0.01040306, 0.01062026, 0.01057429, 0.01077142,
       0.01055417, 0.0106998 , 0.01067166, 0.01389184, 0.01063647,
       0.01065502, 0.0108202 , 0.02836504, 0.02849851, 0.02826276,
       0.02840991, 0.02840252, 0.02821703, 0.02827115, 0.0286365 ,
       0.02819905, 0.02999139, 0.02796559, 0.02789364, 0.02737803,
       0.02771368, 0.02754865, 0.02784762, 0.04263377, 0.04237299,
       0.04285135, 0.04220982, 0.04148445, 0.04131866, 0.04155841,
       0.04094663, 0.04031048, 0.04047823, 0.04016175, 0.04022198,
       0.03886209, 0.03917956, 0.03893905, 0.03909578, 0.05803604,
```

```
        0.05864158, 0.05846429, 0.05711436, 0.05274844, 0.05307441,
        0.05285292, 0.05120621, 0.0480309 , 0.04828353, 0.04860497,
        0.04820662, 0.04376259, 0.04387174, 0.04364696, 0.04347472,
        0.05817671, 0.05826702, 0.05773587, 0.05652022, 0.05250487,
        0.05243163, 0.05211096, 0.05129957, 0.04829402, 0.04834261,
        0.0482708 , 0.04814782, 0.04375515, 0.04673147, 0.04324174,
        0.04369559, 0.00678215, 0.00664644, 0.00669155, 0.0066462 ,
        0.00668178, 0.00669799, 0.00657225, 0.00693798, 0.00663471,
        0.00643468, 0.00637002, 0.00656023, 0.00638895, 0.00639176,
        0.00675759, 0.00654039, 0.01227884, 0.01222935, 0.01208453,
        0.01218719, 0.01201272, 0.01204338, 0.01216359, 0.01241531,
        0.01243701, 0.01242042, 0.01224966, 0.0121573 , 0.0120388 ,
        0.01203337, 0.01226335, 0.01205249, 0.03259578, 0.03262253,
        0.03447604, 0.03434939, 0.03471231, 0.03433137, 0.0340744 ,
        0.03362608, 0.03330832, 0.03453212, 0.03440361, 0.03333416,
        0.03274984, 0.03216462, 0.03205981, 0.0322516 , 0.04798784,
        0.04784884, 0.04800892, 0.04842892, 0.04755554, 0.04748712,
        0.04736338, 0.04765   , 0.04648442, 0.04761758, 0.04725704,
        0.04722877, 0.04458909, 0.04416113, 0.04498601, 0.04557242,
        0.06648035, 0.06866369, 0.07071662, 0.06835127, 0.06581798,
        0.06302919, 0.06757064, 0.06159778, 0.05727859, 0.05741634,
        0.05741949, 0.05721965, 0.05100851, 0.05097175, 0.05056963,
        0.05056286, 0.06712823, 0.06677175, 0.06613932, 0.06539063,
        0.06302238, 0.06308727, 0.06349983, 0.06240463, 0.05717511,
        0.05730023, 0.05700064, 0.05705385, 0.05087252, 0.05121298,
        0.0505899 , 0.05059547, 0.00661807, 0.00664406, 0.0065979 ,
        0.00635977, 0.00648398, 0.00650563, 0.00643196, 0.0065558 ,
        0.0063839 , 0.00658507, 0.00662551, 0.00644608, 0.00647426,
        0.00662518, 0.00643396, 0.00661187, 0.01241617, 0.01237178,
        0.01216025, 0.01226087, 0.01227765, 0.01223421, 0.01220589,
        0.0123816 , 0.01230216, 0.01264687, 0.01236634, 0.01220384,
        0.01230693, 0.01225305, 0.01235061, 0.01247058, 0.03298421,
        0.03293009, 0.03305249, 0.03305058, 0.03410707, 0.03410945,
        0.03410258, 0.03265495, 0.03207798, 0.03232751, 0.03309422,
        0.03230405, 0.0320178 , 0.03187613, 0.03186131, 0.03202386,
        0.04827456, 0.04800324, 0.04880128, 0.04799256, 0.04730954,
        0.04751792, 0.04721417, 0.04755077, 0.04656992, 0.04637136,
        0.04741101, 0.04692893, 0.04459624, 0.04448538, 0.04410009,
        0.04420137, 0.06665597, 0.06624203, 0.06568995, 0.06612983,
        0.06415763, 0.06261463, 0.06261997, 0.0619462 , 0.05864367,
        0.05759463, 0.05718827, 0.05723405, 0.05041857, 0.05201421,
        0.05265722, 0.05242462, 0.06756563, 0.07306466, 0.06642704,
        0.06700201, 0.06358056, 0.06316981, 0.06352115, 0.06310897,
        0.05714612, 0.05734539, 0.05721817, 0.05735283, 0.05108361,
        0.05125065, 0.05099978, 0.05080962]), 'std_fit_time': array([
4.99562813e-04, 1.98889596e-04, 4.16163044e-04, 1.38294039e-04,
        1.76781073e-04, 9.79442095e-05, 1.92110205e-04, 1.08770831e-0
4,
        1.35997151e-04, 2.78992633e-04, 4.58266062e-05, 1.52194021e-0
4,
        1.73021426e-04, 9.68158640e-05, 1.02984971e-04, 2.18294395e-0
4,
        1.34303571e-04, 2.39512053e-04, 2.88639918e-04, 3.06922326e-0
4,
        1.75928383e-04, 1.04198423e-04, 1.36151964e-04, 1.13584313e-0
4,
```

```
        4.97964129e-05, 1.79693260e-04, 1.81408522e-04, 3.04121170e-0
4,
        3.84108574e-03, 1.11821272e-04, 4.04210519e-04, 1.71376455e-0
4,
        3.33568591e-04, 4.64007164e-04, 3.23273882e-04, 3.71342869e-0
4,
        7.43745745e-04, 2.80329941e-04, 2.63520525e-04, 4.14629731e-0
4,
        6.18259943e-04, 7.03076897e-04, 4.93114973e-04, 4.15140729e-0
4,
        3.03594486e-04, 4.37728917e-04, 3.68377416e-04, 2.55750115e-0
4,
        3.99160012e-04, 2.50147946e-04, 5.17500223e-04, 4.58601949e-0
4,
        5.58716746e-04, 2.58816694e-04, 5.55715251e-04, 1.12750926e-0
4,
        4.59380983e-04, 5.20790555e-04, 6.66160430e-04, 4.41004913e-0
4,
        6.06557888e-04, 7.05849677e-04, 9.48002836e-04, 6.54477727e-0
4,
        9.53236937e-04, 1.05820330e-03, 1.15950947e-03, 1.21247923e-0
3,
        1.58975995e-03, 1.91359795e-03, 1.75914398e-03, 1.94542343e-0
3,
        4.99013205e-04, 6.76583325e-04, 4.44635117e-04, 7.30541378e-0
4,
        1.05494812e-03, 7.80644391e-04, 1.12202791e-03, 1.12650053e-0
3,
        8.03703554e-04, 1.09449307e-03, 1.04530722e-03, 1.33486829e-0
3,
        1.93545519e-03, 1.84309529e-03, 1.76708873e-03, 2.01494297e-0
3,
        3.26057169e-04, 8.76894363e-04, 6.08449980e-04, 9.17128335e-0
4,
        1.11661780e-03, 6.85766752e-03, 9.46833069e-04, 8.65268510e-0
4,
        9.35503708e-05, 7.72828446e-05, 1.58667204e-04, 1.06511098e-0
4,
        1.38434590e-04, 1.23612766e-04, 4.70838968e-05, 1.94954218e-0
4,
        5.09616018e-05, 1.11528430e-04, 5.32779697e-05, 9.36618641e-0
5,
        5.30146429e-05, 1.01556688e-04, 2.25342193e-04, 1.41029102e-0
4,
        1.90941365e-04, 1.80775476e-04, 1.67327751e-04, 1.71274133e-0
4,
        1.41920229e-04, 1.80203860e-04, 2.53675369e-04, 2.85600922e-0
4,
        1.38111716e-04, 6.04263881e-05, 7.65552219e-05, 1.76192950e-0
4,
        1.49271555e-04, 5.55720107e-05, 2.01331095e-04, 1.95881048e-0
4,
        3.18978990e-04, 5.00536930e-04, 3.22219830e-04, 2.56516649e-0
4,
        7.67911312e-04, 5.95298859e-04, 2.03644897e-04, 3.09880544e-0
4,
```

```
       1.82224828e-04, 4.15946761e-04, 4.53657555e-04, 1.03340261e-0
4,
       3.96375270e-04, 5.71686637e-04, 2.57802555e-04, 4.73043250e-0
4,
       5.18396003e-04, 4.60725175e-04, 5.70822338e-04, 5.85332861e-0
4,
       8.48291275e-04, 6.63446975e-04, 5.55178098e-04, 3.87460124e-0
4,
       5.24928983e-04, 7.21798608e-04, 8.46818902e-04, 8.43686586e-0
4,
       5.17305234e-04, 5.51151738e-04, 6.58746953e-04, 1.43604403e-0
3,
       1.50391919e-03, 1.71943963e-03, 3.09361589e-03, 3.20867687e-0
3,
       2.91809666e-03, 2.26853777e-03, 1.16795826e-02, 2.06431289e-0
3,
       1.43820661e-03, 1.69018769e-03, 1.73894176e-03, 1.91027614e-0
3,
       9.06742617e-04, 2.10266156e-03, 6.81966372e-04, 1.36762300e-0
3,
       2.88817036e-03, 2.85041306e-03, 2.35940105e-03, 3.03293215e-0
3,
       2.97111374e-03, 2.97095254e-03, 3.21005873e-03, 2.68788453e-0
3,
       1.63584532e-03, 1.81669882e-03, 1.53138802e-03, 1.77216302e-0
3,
       8.19159918e-04, 1.26716219e-03, 1.41749613e-03, 8.49666663e-0
4,
       1.22247824e-04, 1.33631978e-04, 1.59965016e-04, 5.54624137e-0
5,
       1.32362763e-04, 7.84160990e-05, 8.39440093e-05, 1.21298788e-0
4,
       9.86809862e-05, 1.00685767e-04, 8.88820626e-05, 9.29513779e-0
5,
       1.34361424e-04, 1.07403270e-04, 7.08209938e-05, 2.09907562e-0
4,
       2.63247471e-04, 1.21072044e-04, 1.15221797e-04, 2.25589256e-0
4,
       1.41200605e-04, 2.18040171e-04, 2.01635902e-04, 2.97016557e-0
4,
       2.05204088e-04, 2.41062355e-04, 1.30710746e-04, 1.44547315e-0
4,
       2.24932634e-04, 1.21695532e-04, 1.38879459e-04, 2.43290412e-0
4,
       4.82475689e-04, 3.33864659e-04, 3.63346360e-04, 4.84481745e-0
4,
       7.06964962e-04, 3.47250505e-04, 1.07594302e-03, 6.08607262e-0
4,
       3.71336257e-04, 3.99693900e-04, 6.76619971e-04, 3.56478178e-0
4,
       5.04384060e-04, 5.52030887e-04, 3.36464118e-04, 4.23380794e-0
4,
       9.25235811e-04, 6.37382745e-04, 1.22991662e-03, 3.57830010e-0
4,
       7.76343842e-04, 5.74957397e-04, 6.21945774e-04, 1.57573055e-0
3,
```

```
       6.23461122e-04, 8.06153098e-04, 1.35032234e-03, 8.69589341e-0
4,
       7.80183021e-04, 5.89148497e-04, 3.20498298e-04, 8.11313461e-0
4,
       1.33696342e-03, 1.71962528e-03, 2.02705491e-03, 1.77644396e-0
3,
       2.83581558e-03, 2.26872186e-03, 2.24317535e-03, 2.53785029e-0
3,
       1.90175980e-03, 2.05257792e-03, 1.84441448e-03, 1.72371558e-0
3,
       1.21168428e-03, 1.61005184e-03, 1.63170394e-03, 1.50659150e-0
3,
       2.46184945e-03, 1.05272141e-02, 2.76555033e-03, 3.00949256e-0
3,
       2.63693739e-03, 2.70883110e-03, 3.13740831e-03, 3.26146910e-0
3,
       1.79359376e-03, 1.65503434e-03, 1.91789095e-03, 1.70651942e-0
3,
       1.15655964e-03, 1.10418525e-03, 9.64637005e-04, 1.52575904e-0
3]), 'mean_score_time': array([0.00139351, 0.00076337, 0.00066795,
0.000561  , 0.00054789,
       0.00065684, 0.000881  , 0.00061955, 0.00058041, 0.00068479,
       0.00053406, 0.00058756, 0.00060282, 0.00053701, 0.00052323,
       0.00062637, 0.00071979, 0.00077362, 0.00061445, 0.00072727,
       0.00066724, 0.00057092, 0.00069098, 0.00060697, 0.00068831,
       0.00059347, 0.00061479, 0.00083451, 0.00179982, 0.0006671 ,
       0.00066886, 0.00069261, 0.00077314, 0.00076184, 0.00082884,
       0.00078783, 0.00083294, 0.00078521, 0.0007453 , 0.00083742,
       0.00079756, 0.00102358, 0.00074711, 0.00076957, 0.00070219,
       0.00077605, 0.00077519, 0.00076017, 0.00091219, 0.00083184,
       0.00113244, 0.00085359, 0.00089278, 0.00080371, 0.0009738 ,
       0.00078216, 0.00079155, 0.00084872, 0.00084519, 0.00085621,
       0.00092726, 0.00095463, 0.00087662, 0.00092583, 0.00087781,
       0.00098782, 0.00090113, 0.00091381, 0.00091352, 0.00095525,
       0.00094762, 0.00085816, 0.00083599, 0.00080914, 0.00090904,
       0.00085416, 0.00084023, 0.00088696, 0.0008667 , 0.0008924 ,
       0.00091147, 0.00089955, 0.00083957, 0.00083256, 0.00085497,
       0.00085869, 0.00081806, 0.00089159, 0.00086579, 0.00086617,
       0.00082221, 0.00087943, 0.00090547, 0.00088067, 0.00082326,
       0.00089097, 0.00062184, 0.00053706, 0.00056343, 0.00056639,
       0.00056896, 0.00053501, 0.00049987, 0.00065536, 0.00062828,
       0.00050941, 0.00051904, 0.00055733, 0.00051842, 0.00052376,
       0.00065665, 0.00056338, 0.00062613, 0.00066214, 0.0006074 ,
       0.00066338, 0.00064168, 0.0006176 , 0.0006784 , 0.00063772,
       0.00068631, 0.00075822, 0.00067792, 0.0006155 , 0.00064602,
       0.00061474, 0.00067577, 0.00063787, 0.00073581, 0.00072846,
       0.00094209, 0.00093141, 0.00094743, 0.0009716 , 0.00098195,
       0.00088964, 0.00090904, 0.00100484, 0.00099158, 0.00085206,
       0.00077815, 0.00079942, 0.00078039, 0.00077243, 0.00089488,
       0.00084195, 0.00083451, 0.00084963, 0.00085411, 0.0007834 ,
       0.00079956, 0.00079498, 0.00080657, 0.00083923, 0.00082917,
       0.00091357, 0.0009233 , 0.00083222, 0.00088134, 0.00089793,
       0.00085793, 0.00098076, 0.001019  , 0.00105038, 0.00094481,
       0.00089827, 0.00084023, 0.00083179, 0.00095968, 0.00086479,
       0.00083275, 0.00081601, 0.00085778, 0.0008409 , 0.00087299,
       0.00081949, 0.0008863 , 0.00086188, 0.00081916, 0.00084319,
```

```
        0.00086203, 0.00091815, 0.00091138, 0.00083871, 0.00084739,
        0.00085039, 0.00091176, 0.00092864, 0.00089555, 0.00084381,
        0.00085874, 0.00085859, 0.00058603, 0.0005868 , 0.00061145,
        0.0005259 , 0.00054793, 0.0005578 , 0.00053673, 0.00057735,
        0.00049801, 0.00057459, 0.00062575, 0.0005424 , 0.00058327,
        0.00056753, 0.00050082, 0.00058475, 0.00065508, 0.00068226,
        0.00064483, 0.00062718, 0.00069017, 0.0006156 , 0.0006248 ,
        0.00074229, 0.00064416, 0.00073395, 0.00069242, 0.00068531,
        0.00064144, 0.00064378, 0.00072198, 0.00068498, 0.00074496,
        0.00084095, 0.0007575 , 0.0007792 , 0.00085373, 0.00075378,
        0.00079427, 0.00078182, 0.00074244, 0.00074067, 0.00085573,
        0.00076327, 0.00079532, 0.00072441, 0.00068464, 0.00074787,
        0.00078464, 0.00077758, 0.00077915, 0.00079222, 0.00077357,
        0.00086746, 0.0008492 , 0.00082364, 0.00086179, 0.00082464,
        0.00085382, 0.00084176, 0.00085883, 0.00086164, 0.00074601,
        0.00086861, 0.00090032, 0.00092959, 0.00095444, 0.0009696 ,
        0.00094185, 0.00082674, 0.00084238, 0.00090041, 0.00111918,
        0.00087414, 0.00081706, 0.0008812 , 0.00086555, 0.00093417,
        0.00102501, 0.00085392, 0.00085521, 0.00120506, 0.00082536,
        0.00099125, 0.00086703, 0.00085778, 0.00089564, 0.00097241,
        0.00090222, 0.00094628, 0.00087981, 0.00089335, 0.00088282,
        0.00084038, 0.00091848, 0.00087996]), 'std_score_time': arra
y([8.80392070e-04, 8.78195539e-05, 9.06478098e-05, 4.75064798e-05,
        1.07227520e-04, 1.21045655e-04, 1.88964459e-04, 7.71539025e-0
5,
        8.39935627e-05, 6.83824054e-05, 9.38775994e-06, 4.37524013e-0
5,
        4.90177466e-05, 2.96578146e-05, 1.73431122e-05, 1.07824512e-0
4,
        5.96965082e-05, 1.95084006e-04, 6.27852411e-05, 1.33576345e-0
4,
        8.07059533e-05, 3.69767463e-05, 9.81702679e-05, 7.87782610e-0
5,
        4.49902871e-05, 6.76388761e-05, 3.67079762e-05, 4.61227637e-0
4,
        1.29232762e-03, 7.80429633e-05, 1.29369058e-04, 8.57104678e-0
5,
        4.61393607e-05, 7.48610952e-05, 1.60737173e-04, 6.99735743e-0
5,
        4.64554426e-05, 6.21364011e-05, 6.13663672e-05, 4.87757100e-0
5,
        7.22669328e-05, 1.14258919e-04, 1.08039726e-04, 1.20150796e-0
4,
        6.04454626e-05, 9.79238249e-05, 4.22676806e-05, 2.84981918e-0
5,
        7.23437561e-05, 5.35403412e-05, 4.37537223e-04, 3.64854905e-0
5,
        1.55968441e-04, 4.39668684e-05, 2.50130503e-04, 1.83520988e-0
5,
        2.50523363e-05, 5.17497728e-05, 4.91893989e-05, 1.26520579e-0
4,
        1.79616361e-04, 8.94581330e-05, 4.91455589e-05, 1.38287084e-0
4,
        4.15971485e-05, 1.25177283e-04, 6.55644330e-05, 1.41198673e-0
4,
        7.42611071e-05, 1.09519811e-04, 1.30300114e-04, 8.87991398e-0
```

```
5,
        8.53372803e-05, 9.32711521e-05, 5.16182336e-05, 2.92408894e-0
5,
        4.95705195e-05, 8.26534185e-05, 9.71360011e-05, 1.31074894e-0
4,
        5.65292412e-05, 6.98207496e-05, 6.15061898e-05, 3.22117530e-0
5,
        5.30951270e-05, 7.65319331e-05, 2.24125639e-05, 7.92581105e-0
5,
        5.33971609e-05, 5.42813711e-05, 6.40791082e-05, 7.93865275e-0
5,
        4.25915690e-05, 8.73008492e-05, 3.00024940e-05, 2.20188303e-0
5,
        5.40297533e-05, 3.37298171e-05, 7.24714677e-05, 4.82052888e-0
5,
        9.34399612e-05, 3.96455042e-05, 8.66821232e-06, 7.76927684e-0
5,
        3.49497241e-05, 2.23359402e-05, 4.91896301e-05, 3.38112852e-0
5,
        4.96516866e-05, 5.57678130e-05, 1.10260012e-04, 6.89835226e-0
5,
        7.16598205e-05, 1.00365376e-04, 7.22268377e-05, 8.02377671e-0
5,
        9.98010717e-05, 9.40008720e-05, 8.51091682e-05, 7.24044527e-0
5,
        4.52258364e-05, 3.76563711e-05, 7.53776240e-05, 5.07249409e-0
5,
        8.41939151e-05, 1.03958768e-04, 5.25193650e-05, 8.60419059e-0
5,
        2.38930153e-05, 8.62040083e-05, 6.38076183e-05, 5.28813483e-0
5,
        5.28195686e-05, 2.71924307e-05, 8.06089796e-05, 7.57366329e-0
5,
        1.47433423e-04, 6.06829570e-05, 1.10075666e-04, 4.24683391e-0
5,
        4.73788529e-05, 1.01894310e-04, 2.35686865e-05, 5.82169938e-0
5,
        4.20889766e-05, 4.72806585e-05, 7.24068079e-05, 8.33432512e-0
5,
        6.30278082e-05, 3.50358191e-05, 8.72365208e-05, 3.49901014e-0
5,
        7.11067974e-05, 1.18511062e-05, 8.56141697e-05, 4.83834425e-0
5,
        9.65882600e-05, 6.18188802e-05, 7.21405296e-05, 7.16989958e-0
5,
        4.49140612e-05, 6.25145335e-05, 6.68351536e-05, 8.85645406e-0
5,
        6.83032238e-05, 5.62208198e-05, 4.35961655e-05, 3.67156561e-0
5,
        7.25051871e-05, 7.30663912e-05, 9.39792693e-06, 3.84046672e-0
5,
        8.50992561e-05, 7.83170739e-05, 6.59985692e-05, 5.23929696e-0
5,
        5.60101216e-05, 5.44455332e-05, 3.66381640e-05, 6.72114788e-0
5,
        5.15852302e-05, 7.20587567e-05, 6.49431553e-05, 2.85208418e-0
```

```
5,
        3.81083293e-05, 4.57096110e-05, 1.15450184e-04, 5.69512072e-0
5,
        5.46858424e-05, 4.53119261e-05, 6.27439430e-05, 5.38890126e-0
5,
        6.07013890e-05, 6.79894212e-05, 9.46579530e-05, 7.05590752e-0
5,
        4.24008256e-05, 2.87642308e-05, 5.49020373e-05, 3.39937073e-0
5,
        2.71128786e-05, 8.55593897e-05, 4.10630168e-05, 4.66218982e-0
5,
        5.88156165e-05, 4.54199846e-05, 1.80094268e-05, 7.88876028e-0
5,
        1.01455955e-04, 2.86172222e-05, 1.24789350e-04, 7.11454143e-0
5,
        6.97750455e-05, 4.92776079e-05, 6.26496888e-05, 1.07116838e-0
4,
        1.21855771e-04, 7.51173064e-05, 6.21612425e-05, 1.42425009e-0
4,
        4.09443494e-05, 3.71619239e-05, 3.98840578e-05, 1.04218976e-0
4,
        1.51965242e-05, 1.33474157e-04, 3.03395113e-05, 4.05405258e-0
5,
        8.84102128e-05, 3.88408965e-05, 8.09088264e-05, 8.17419620e-0
5,
        1.15741649e-04, 6.81546271e-05, 8.28439758e-05, 1.18397005e-0
4,
        2.53323078e-05, 6.40400647e-05, 6.61327928e-05, 5.50874679e-0
5,
        9.46716918e-05, 1.02138542e-04, 8.18562893e-05, 3.88136074e-0
5,
        5.70693803e-05, 9.66223877e-05, 6.71649131e-05, 6.36447400e-0
5,
        9.30380782e-05, 4.05211717e-05, 4.48278668e-05, 6.16007542e-0
5,
        9.78319954e-05, 6.67953382e-05, 3.04664989e-05, 9.23014858e-0
5,
        4.77183693e-05, 1.29568159e-04, 1.52387890e-04, 4.98714225e-0
5,
        9.07950540e-05, 4.63076399e-05, 4.40416361e-05, 5.38546566e-0
5,
        3.66388138e-04, 7.21397417e-05, 4.10221319e-05, 8.42571393e-0
5,
        7.40360289e-05, 8.04806485e-05, 3.60169597e-05, 2.88904297e-0
5,
        8.68544191e-05, 4.06384859e-04, 1.05157460e-05, 9.86894650e-0
5,
        5.16127712e-05, 3.91057058e-05, 7.30785576e-05, 7.49078546e-0
5,
        4.91242723e-05, 1.00771273e-04, 7.61306625e-05, 5.16159430e-0
5,
        5.91572949e-05, 2.92022955e-05, 1.33866510e-04, 8.52560837e-0
5]), 'param_criterion': masked_array(data=['gini', 'gini', 'gini', '
gini', 'gini', 'gini', 'gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
```

```
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', '
gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'entrop
y',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                   'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
```

```
                 'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                 'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                 'entropy', 'entropy', 'entropy', 'entropy', 'entr
opy',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss',
                 'log_loss', 'log_loss', 'log_loss', 'log_loss'],
           mask=[False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
```

```
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False],
         fill_value=np.str_('?'),
              dtype=object), 'param_max_depth': masked_array(data=[1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
                        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6,
6, 6,
                        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 10, 10, 10, 1
0, 10,
                        10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 25, 2
5, 25,
                        25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 2
```

```
5,
                          None, None, None, None, None, None, None, None, N
one,
                          None, None, None, None, None, None, None, 1, 1,
1, 1,
                          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2,
                          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6, 6, 6, 6, 6,
6, 6,
                          6, 6, 6, 6, 6, 6, 6, 6, 10, 10, 10, 10, 10, 10, 1
0, 10,
                          10, 10, 10, 10, 10, 10, 10, 10, 25, 25, 25, 25, 2
5, 25,
                          25, 25, 25, 25, 25, 25, 25, 25, 25, 25, None, Non
e,
                          None, None, None, None, None, None, None, None, N
one,
                          None, None, None, None, None, 1, 1, 1, 1, 1, 1,
1, 1,
                          1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
                          2, 2, 2, 2, 2, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
                          6, 6, 6, 6, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1
0, 10,
                          10, 10, 10, 10, 10, 25, 25, 25, 25, 25, 25, 25, 2
5, 25,
                          25, 25, 25, 25, 25, 25, 25, None, None, None, Non
e,
                          None, None, None, None, None, None, None, None, N
one,
                          None, None, None],
                  mask=[False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
                          False, False, False, False, False, False, False,
False,
```

```
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False,
                      False, False, False, False, False, False, False,
False],
        fill_value=np.str_('?'),
             dtype=object), 'param_min_samples_leaf': masked_array(da
ta=[2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 20, 20,
                      2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                      2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                      2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
```

```
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20,
                 2, 2, 2, 2, 5, 5, 5, 5, 10, 10, 10, 10, 20, 20, 2
0, 20],
            mask=[False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
```

```
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False],
         fill_value=999999), 'param_min_samples_split': masked_array(d
ata=[2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20,
                        2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                        2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                        2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                        2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                        2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
```

```
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20,
                 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 10, 20, 2, 6, 1
0, 20],
            mask=[False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
False,
                 False, False, False, False, False, False, False,
```

```
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False,
                    False, False, False, False, False, False, False,
False],
       fill_value=999999), 'params': [{'criterion': 'gini', 'max_dep
th': 1, 'min_samples_leaf': 2, 'min_samples_split': 2}, {'criterio
n': 'gini', 'max_depth': 1, 'min_samples_leaf': 2, 'min_samples_spli
t': 6}, {'criterion': 'gini', 'max_depth': 1, 'min_samples_leaf': 2,
'min_samples_split': 10}, {'criterion': 'gini', 'max_depth': 1, 'min
_samples_leaf': 2, 'min_samples_split': 20}, {'criterion': 'gini', '
max_depth': 1, 'min_samples_leaf': 5, 'min_samples_split': 2}, {'cri
terion': 'gini', 'max_depth': 1, 'min_samples_leaf': 5, 'min_samples
_split': 6}, {'criterion': 'gini', 'max_depth': 1, 'min_samples_lea
f': 5, 'min_samples_split': 10}, {'criterion': 'gini', 'max_depth':
1, 'min_samples_leaf': 5, 'min_samples_split': 20}, {'criterion': 'g
ini', 'max_depth': 1, 'min_samples_leaf': 10, 'min_samples_split':
2}, {'criterion': 'gini', 'max_depth': 1, 'min_samples_leaf': 10, 'm
in_samples_split': 6}, {'criterion': 'gini', 'max_depth': 1, 'min_sa
mples_leaf': 10, 'min_samples_split': 10}, {'criterion': 'gini', 'ma
```

```
x_depth': 1, 'min_samples_leaf': 10, 'min_samples_split': 20}, {'cri
terion': 'gini', 'max_depth': 1, 'min_samples_leaf': 20, 'min_sample
s_split': 2}, {'criterion': 'gini', 'max_depth': 1, 'min_samples_lea
f': 20, 'min_samples_split': 6}, {'criterion': 'gini', 'max_depth':
1, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'criterion': '
gini', 'max_depth': 1, 'min_samples_leaf': 20, 'min_samples_split':
20}, {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 2, 'm
in_samples_split': 2}, {'criterion': 'gini', 'max_depth': 2, 'min_sa
mples_leaf': 2, 'min_samples_split': 6}, {'criterion': 'gini', 'max_
depth': 2, 'min_samples_leaf': 2, 'min_samples_split': 10}, {'criter
ion': 'gini', 'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_sp
lit': 20}, {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf':
5, 'min_samples_split': 2}, {'criterion': 'gini', 'max_depth': 2, 'm
in_samples_leaf': 5, 'min_samples_split': 6}, {'criterion': 'gini',
'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 10}, {'c
riterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 5, 'min_sampl
es_split': 20}, {'criterion': 'gini', 'max_depth': 2, 'min_samples_l
eaf': 10, 'min_samples_split': 2}, {'criterion': 'gini', 'max_dept
h': 2, 'min_samples_leaf': 10, 'min_samples_split': 6}, {'criterio
n': 'gini', 'max_depth': 2, 'min_samples_leaf': 10, 'min_samples_spl
it': 10}, {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf':
10, 'min_samples_split': 20}, {'criterion': 'gini', 'max_depth': 2,
'min_samples_leaf': 20, 'min_samples_split': 2}, {'criterion': 'gin
i', 'max_depth': 2, 'min_samples_leaf': 20, 'min_samples_split': 6},
{'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 20, 'min_s
amples_split': 10}, {'criterion': 'gini', 'max_depth': 2, 'min_sampl
es_leaf': 20, 'min_samples_split': 20}, {'criterion': 'gini', 'max_d
epth': 6, 'min_samples_leaf': 2, 'min_samples_split': 2}, {'criterio
n': 'gini', 'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_spli
t': 6}, {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 2,
'min_samples_split': 10}, {'criterion': 'gini', 'max_depth': 6, 'min
_samples_leaf': 2, 'min_samples_split': 20}, {'criterion': 'gini', '
max_depth': 6, 'min_samples_leaf': 5, 'min_samples_split': 2}, {'cri
terion': 'gini', 'max_depth': 6, 'min_samples_leaf': 5, 'min_samples
_split': 6}, {'criterion': 'gini', 'max_depth': 6, 'min_samples_lea
f': 5, 'min_samples_split': 10}, {'criterion': 'gini', 'max_depth':
6, 'min_samples_leaf': 5, 'min_samples_split': 20}, {'criterion': 'g
ini', 'max_depth': 6, 'min_samples_leaf': 10, 'min_samples_split':
2}, {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 10, 'm
in_samples_split': 6}, {'criterion': 'gini', 'max_depth': 6, 'min_sa
mples_leaf': 10, 'min_samples_split': 10}, {'criterion': 'gini', 'ma
x_depth': 6, 'min_samples_leaf': 10, 'min_samples_split': 20}, {'cri
terion': 'gini', 'max_depth': 6, 'min_samples_leaf': 20, 'min_sample
s_split': 2}, {'criterion': 'gini', 'max_depth': 6, 'min_samples_lea
f': 20, 'min_samples_split': 6}, {'criterion': 'gini', 'max_depth':
6, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'criterion': '
gini', 'max_depth': 6, 'min_samples_leaf': 20, 'min_samples_split':
20}, {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, '
min_samples_split': 2}, {'criterion': 'gini', 'max_depth': 10, 'min_
samples_leaf': 2, 'min_samples_split': 6}, {'criterion': 'gini', 'ma
x_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10}, {'cri
terion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, 'min_sample
s_split': 20}, {'criterion': 'gini', 'max_depth': 10, 'min_samples_l
eaf': 5, 'min_samples_split': 2}, {'criterion': 'gini', 'max_depth':
10, 'min_samples_leaf': 5, 'min_samples_split': 6}, {'criterion': 'g
ini', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 1
```

```
0}, {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 5, 'm
in_samples_split': 20}, {'criterion': 'gini', 'max_depth': 10, 'min_
samples_leaf': 10, 'min_samples_split': 2}, {'criterion': 'gini', 'm
ax_depth': 10, 'min_samples_leaf': 10, 'min_samples_split': 6}, {'cr
iterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 10, 'min_samp
les_split': 10}, {'criterion': 'gini', 'max_depth': 10, 'min_samples
_leaf': 10, 'min_samples_split': 20}, {'criterion': 'gini', 'max_dep
th': 10, 'min_samples_leaf': 20, 'min_samples_split': 2}, {'criterio
n': 'gini', 'max_depth': 10, 'min_samples_leaf': 20, 'min_samples_sp
lit': 6}, {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf':
20, 'min_samples_split': 10}, {'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 20, 'min_samples_split': 20}, {'criterion': 'gin
i', 'max_depth': 25, 'min_samples_leaf': 2, 'min_samples_split': 2},
{'criterion': 'gini', 'max_depth': 25, 'min_samples_leaf': 2, 'min_s
amples_split': 6}, {'criterion': 'gini', 'max_depth': 25, 'min_sampl
es_leaf': 2, 'min_samples_split': 10}, {'criterion': 'gini', 'max_de
pth': 25, 'min_samples_leaf': 2, 'min_samples_split': 20}, {'criteri
on': 'gini', 'max_depth': 25, 'min_samples_leaf': 5, 'min_samples_sp
lit': 2}, {'criterion': 'gini', 'max_depth': 25, 'min_samples_leaf':
5, 'min_samples_split': 6}, {'criterion': 'gini', 'max_depth': 25, '
min_samples_leaf': 5, 'min_samples_split': 10}, {'criterion': 'gin
i', 'max_depth': 25, 'min_samples_leaf': 5, 'min_samples_split': 2
0}, {'criterion': 'gini', 'max_depth': 25, 'min_samples_leaf': 10, '
min_samples_split': 2}, {'criterion': 'gini', 'max_depth': 25, 'min_
samples_leaf': 10, 'min_samples_split': 6}, {'criterion': 'gini', 'm
ax_depth': 25, 'min_samples_leaf': 10, 'min_samples_split': 10}, {'c
riterion': 'gini', 'max_depth': 25, 'min_samples_leaf': 10, 'min_sam
ples_split': 20}, {'criterion': 'gini', 'max_depth': 25, 'min_sample
s_leaf': 20, 'min_samples_split': 2}, {'criterion': 'gini', 'max_dep
th': 25, 'min_samples_leaf': 20, 'min_samples_split': 6}, {'criterio
n': 'gini', 'max_depth': 25, 'min_samples_leaf': 20, 'min_samples_sp
lit': 10}, {'criterion': 'gini', 'max_depth': 25, 'min_samples_lea
f': 20, 'min_samples_split': 20}, {'criterion': 'gini', 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 2}, {'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_spli
t': 6}, {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf':
2, 'min_samples_split': 10}, {'criterion': 'gini', 'max_depth': Non
e, 'min_samples_leaf': 2, 'min_samples_split': 20}, {'criterion': 'g
ini', 'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split':
2}, {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 5,
'min_samples_split': 6}, {'criterion': 'gini', 'max_depth': None, 'm
in_samples_leaf': 5, 'min_samples_split': 10}, {'criterion': 'gini',
'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 20},
{'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 10, 'mi
n_samples_split': 2}, {'criterion': 'gini', 'max_depth': None, 'min_
samples_leaf': 10, 'min_samples_split': 6}, {'criterion': 'gini', 'm
ax_depth': None, 'min_samples_leaf': 10, 'min_samples_split': 10},
{'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 10, 'mi
n_samples_split': 20}, {'criterion': 'gini', 'max_depth': None, 'min
_samples_leaf': 20, 'min_samples_split': 2}, {'criterion': 'gini', '
max_depth': None, 'min_samples_leaf': 20, 'min_samples_split': 6},
{'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 20, 'mi
n_samples_split': 10}, {'criterion': 'gini', 'max_depth': None, 'min
_samples_leaf': 20, 'min_samples_split': 20}, {'criterion': 'entrop
y', 'max_depth': 1, 'min_samples_leaf': 2, 'min_samples_split': 2},
{'criterion': 'entropy', 'max_depth': 1, 'min_samples_leaf': 2, 'min
```

```
_samples_split': 6}, {'criterion': 'entropy', 'max_depth': 1, 'min_s
amples_leaf': 2, 'min_samples_split': 10}, {'criterion': 'entropy',
'max_depth': 1, 'min_samples_leaf': 2, 'min_samples_split': 20}, {'c
riterion': 'entropy', 'max_depth': 1, 'min_samples_leaf': 5, 'min_sa
mples_split': 2}, {'criterion': 'entropy', 'max_depth': 1, 'min_samp
les_leaf': 5, 'min_samples_split': 6}, {'criterion': 'entropy', 'max
_depth': 1, 'min_samples_leaf': 5, 'min_samples_split': 10}, {'crite
rion': 'entropy', 'max_depth': 1, 'min_samples_leaf': 5, 'min_sample
s_split': 20}, {'criterion': 'entropy', 'max_depth': 1, 'min_samples
_leaf': 10, 'min_samples_split': 2}, {'criterion': 'entropy', 'max_d
epth': 1, 'min_samples_leaf': 10, 'min_samples_split': 6}, {'criteri
on': 'entropy', 'max_depth': 1, 'min_samples_leaf': 10, 'min_samples
_split': 10}, {'criterion': 'entropy', 'max_depth': 1, 'min_samples_
leaf': 10, 'min_samples_split': 20}, {'criterion': 'entropy', 'max_d
epth': 1, 'min_samples_leaf': 20, 'min_samples_split': 2}, {'criteri
on': 'entropy', 'max_depth': 1, 'min_samples_leaf': 20, 'min_samples
_split': 6}, {'criterion': 'entropy', 'max_depth': 1, 'min_samples_l
eaf': 20, 'min_samples_split': 10}, {'criterion': 'entropy', 'max_de
pth': 1, 'min_samples_leaf': 20, 'min_samples_split': 20}, {'criteri
on': 'entropy', 'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_
split': 2}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_le
af': 2, 'min_samples_split': 6}, {'criterion': 'entropy', 'max_dept
h': 2, 'min_samples_leaf': 2, 'min_samples_split': 10}, {'criterio
n': 'entropy', 'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_s
plit': 20}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_le
af': 5, 'min_samples_split': 2}, {'criterion': 'entropy', 'max_dept
h': 2, 'min_samples_leaf': 5, 'min_samples_split': 6}, {'criterion':
'entropy', 'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_spli
t': 10}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_lea
f': 5, 'min_samples_split': 20}, {'criterion': 'entropy', 'max_dept
h': 2, 'min_samples_leaf': 10, 'min_samples_split': 2}, {'criterio
n': 'entropy', 'max_depth': 2, 'min_samples_leaf': 10, 'min_samples_
split': 6}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_le
af': 10, 'min_samples_split': 10}, {'criterion': 'entropy', 'max_dep
th': 2, 'min_samples_leaf': 10, 'min_samples_split': 20}, {'criterio
n': 'entropy', 'max_depth': 2, 'min_samples_leaf': 20, 'min_samples_
split': 2}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_le
af': 20, 'min_samples_split': 6}, {'criterion': 'entropy', 'max_dept
h': 2, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'criterio
n': 'entropy', 'max_depth': 2, 'min_samples_leaf': 20, 'min_samples_
split': 20}, {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
eaf': 2, 'min_samples_split': 2}, {'criterion': 'entropy', 'max_dept
h': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}, {'criterion':
'entropy', 'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_spli
t': 10}, {'criterion': 'entropy', 'max_depth': 6, 'min_samples_lea
f': 2, 'min_samples_split': 20}, {'criterion': 'entropy', 'max_dept
h': 6, 'min_samples_leaf': 5, 'min_samples_split': 2}, {'criterion':
'entropy', 'max_depth': 6, 'min_samples_leaf': 5, 'min_samples_spli
t': 6}, {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf':
5, 'min_samples_split': 10}, {'criterion': 'entropy', 'max_depth':
6, 'min_samples_leaf': 5, 'min_samples_split': 20}, {'criterion': 'e
ntropy', 'max_depth': 6, 'min_samples_leaf': 10, 'min_samples_spli
t': 2}, {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf':
10, 'min_samples_split': 6}, {'criterion': 'entropy', 'max_depth':
6, 'min_samples_leaf': 10, 'min_samples_split': 10}, {'criterion': '
entropy', 'max_depth': 6, 'min_samples_leaf': 10, 'min_samples_spli
```

```
t': 20}, {'criterion': 'entropy', 'max_depth': 6, 'min_samples_lea
f': 20, 'min_samples_split': 2}, {'criterion': 'entropy', 'max_dept
h': 6, 'min_samples_leaf': 20, 'min_samples_split': 6}, {'criterio
n': 'entropy', 'max_depth': 6, 'min_samples_leaf': 20, 'min_samples_
split': 10}, {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
eaf': 20, 'min_samples_split': 20}, {'criterion': 'entropy', 'max_de
pth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2}, {'criterio
n': 'entropy', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_
split': 6}, {'criterion': 'entropy', 'max_depth': 10, 'min_samples_l
eaf': 2, 'min_samples_split': 10}, {'criterion': 'entropy', 'max_dep
th': 10, 'min_samples_leaf': 2, 'min_samples_split': 20}, {'criterio
n': 'entropy', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_
split': 2}, {'criterion': 'entropy', 'max_depth': 10, 'min_samples_l
eaf': 5, 'min_samples_split': 6}, {'criterion': 'entropy', 'max_dept
h': 10, 'min_samples_leaf': 5, 'min_samples_split': 10}, {'criterio
n': 'entropy', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_
split': 20}, {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
leaf': 10, 'min_samples_split': 2}, {'criterion': 'entropy', 'max_de
pth': 10, 'min_samples_leaf': 10, 'min_samples_split': 6}, {'criteri
on': 'entropy', 'max_depth': 10, 'min_samples_leaf': 10, 'min_sample
s_split': 10}, {'criterion': 'entropy', 'max_depth': 10, 'min_sample
s_leaf': 10, 'min_samples_split': 20}, {'criterion': 'entropy', 'max
_depth': 10, 'min_samples_leaf': 20, 'min_samples_split': 2}, {'crit
erion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 20, 'min_sam
ples_split': 6}, {'criterion': 'entropy', 'max_depth': 10, 'min_samp
les_leaf': 20, 'min_samples_split': 10}, {'criterion': 'entropy', 'm
ax_depth': 10, 'min_samples_leaf': 20, 'min_samples_split': 20}, {'c
riterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 2, 'min_s
amples_split': 2}, {'criterion': 'entropy', 'max_depth': 25, 'min_sa
mples_leaf': 2, 'min_samples_split': 6}, {'criterion': 'entropy', 'm
ax_depth': 25, 'min_samples_leaf': 2, 'min_samples_split': 10}, {'cr
iterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 2, 'min_sa
mples_split': 20}, {'criterion': 'entropy', 'max_depth': 25, 'min_sa
mples_leaf': 5, 'min_samples_split': 2}, {'criterion': 'entropy', 'm
ax_depth': 25, 'min_samples_leaf': 5, 'min_samples_split': 6}, {'cri
terion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 5, 'min_sam
ples_split': 10}, {'criterion': 'entropy', 'max_depth': 25, 'min_sam
ples_leaf': 5, 'min_samples_split': 20}, {'criterion': 'entropy', 'm
ax_depth': 25, 'min_samples_leaf': 10, 'min_samples_split': 2}, {'cr
iterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 10, 'min_s
amples_split': 6}, {'criterion': 'entropy', 'max_depth': 25, 'min_sa
mples_leaf': 10, 'min_samples_split': 10}, {'criterion': 'entropy',
'max_depth': 25, 'min_samples_leaf': 10, 'min_samples_split': 20},
{'criterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 20, 'm
in_samples_split': 2}, {'criterion': 'entropy', 'max_depth': 25, 'mi
n_samples_leaf': 20, 'min_samples_split': 6}, {'criterion': 'entrop
y', 'max_depth': 25, 'min_samples_leaf': 20, 'min_samples_split': 1
0}, {'criterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 2
0, 'min_samples_split': 20}, {'criterion': 'entropy', 'max_depth': N
one, 'min_samples_leaf': 2, 'min_samples_split': 2}, {'criterion': '
entropy', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_spl
it': 6}, {'criterion': 'entropy', 'max_depth': None, 'min_samples_le
af': 2, 'min_samples_split': 10}, {'criterion': 'entropy', 'max_dept
h': None, 'min_samples_leaf': 2, 'min_samples_split': 20}, {'criteri
on': 'entropy', 'max_depth': None, 'min_samples_leaf': 5, 'min_sampl
es_split': 2}, {'criterion': 'entropy', 'max_depth': None, 'min_samp
```

```
les_leaf': 5, 'min_samples_split': 6}, {'criterion': 'entropy', 'max
_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 10}, {'cr
iterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 5, 'min_
samples_split': 20}, {'criterion': 'entropy', 'max_depth': None, 'mi
n_samples_leaf': 10, 'min_samples_split': 2}, {'criterion': 'entrop
y', 'max_depth': None, 'min_samples_leaf': 10, 'min_samples_split':
6}, {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf':
10, 'min_samples_split': 10}, {'criterion': 'entropy', 'max_depth':
None, 'min_samples_leaf': 10, 'min_samples_split': 20}, {'criterio
n': 'entropy', 'max_depth': None, 'min_samples_leaf': 20, 'min_sampl
es_split': 2}, {'criterion': 'entropy', 'max_depth': None, 'min_samp
les_leaf': 20, 'min_samples_split': 6}, {'criterion': 'entropy', 'ma
x_depth': None, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'
criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 20, 'm
in_samples_split': 20}, {'criterion': 'log_loss', 'max_depth': 1, 'm
in_samples_leaf': 2, 'min_samples_split': 2}, {'criterion': 'log_los
s', 'max_depth': 1, 'min_samples_leaf': 2, 'min_samples_split': 6},
{'criterion': 'log_loss', 'max_depth': 1, 'min_samples_leaf': 2, 'mi
n_samples_split': 10}, {'criterion': 'log_loss', 'max_depth': 1, 'mi
n_samples_leaf': 2, 'min_samples_split': 20}, {'criterion': 'log_los
s', 'max_depth': 1, 'min_samples_leaf': 5, 'min_samples_split': 2},
{'criterion': 'log_loss', 'max_depth': 1, 'min_samples_leaf': 5, 'mi
n_samples_split': 6}, {'criterion': 'log_loss', 'max_depth': 1, 'min
_samples_leaf': 5, 'min_samples_split': 10}, {'criterion': 'log_los
s', 'max_depth': 1, 'min_samples_leaf': 5, 'min_samples_split': 20},
{'criterion': 'log_loss', 'max_depth': 1, 'min_samples_leaf': 10, 'm
in_samples_split': 2}, {'criterion': 'log_loss', 'max_depth': 1, 'mi
n_samples_leaf': 10, 'min_samples_split': 6}, {'criterion': 'log_los
s', 'max_depth': 1, 'min_samples_leaf': 10, 'min_samples_split': 1
0}, {'criterion': 'log_loss', 'max_depth': 1, 'min_samples_leaf': 1
0, 'min_samples_split': 20}, {'criterion': 'log_loss', 'max_depth':
1, 'min_samples_leaf': 20, 'min_samples_split': 2}, {'criterion': 'l
og_loss', 'max_depth': 1, 'min_samples_leaf': 20, 'min_samples_spli
t': 6}, {'criterion': 'log_loss', 'max_depth': 1, 'min_samples_lea
f': 20, 'min_samples_split': 10}, {'criterion': 'log_loss', 'max_dep
th': 1, 'min_samples_leaf': 20, 'min_samples_split': 20}, {'criterio
n': 'log_loss', 'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_
split': 2}, {'criterion': 'log_loss', 'max_depth': 2, 'min_samples_l
eaf': 2, 'min_samples_split': 6}, {'criterion': 'log_loss', 'max_dep
th': 2, 'min_samples_leaf': 2, 'min_samples_split': 10}, {'criterio
n': 'log_loss', 'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_
split': 20}, {'criterion': 'log_loss', 'max_depth': 2, 'min_samples_
leaf': 5, 'min_samples_split': 2}, {'criterion': 'log_loss', 'max_de
pth': 2, 'min_samples_leaf': 5, 'min_samples_split': 6}, {'criterio
n': 'log_loss', 'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_
split': 10}, {'criterion': 'log_loss', 'max_depth': 2, 'min_samples_
leaf': 5, 'min_samples_split': 20}, {'criterion': 'log_loss', 'max_d
epth': 2, 'min_samples_leaf': 10, 'min_samples_split': 2}, {'criteri
on': 'log_loss', 'max_depth': 2, 'min_samples_leaf': 10, 'min_sample
s_split': 6}, {'criterion': 'log_loss', 'max_depth': 2, 'min_samples
_leaf': 10, 'min_samples_split': 10}, {'criterion': 'log_loss', 'max
_depth': 2, 'min_samples_leaf': 10, 'min_samples_split': 20}, {'crit
erion': 'log_loss', 'max_depth': 2, 'min_samples_leaf': 20, 'min_sam
ples_split': 2}, {'criterion': 'log_loss', 'max_depth': 2, 'min_samp
les_leaf': 20, 'min_samples_split': 6}, {'criterion': 'log_loss', 'm
ax_depth': 2, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'cr
```

```
iterion': 'log_loss', 'max_depth': 2, 'min_samples_leaf': 20, 'min_s
amples_split': 20}, {'criterion': 'log_loss', 'max_depth': 6, 'min_s
amples_leaf': 2, 'min_samples_split': 2}, {'criterion': 'log_loss',
'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}, {'cr
iterion': 'log_loss', 'max_depth': 6, 'min_samples_leaf': 2, 'min_sa
mples_split': 10}, {'criterion': 'log_loss', 'max_depth': 6, 'min_sa
mples_leaf': 2, 'min_samples_split': 20}, {'criterion': 'log_loss',
'max_depth': 6, 'min_samples_leaf': 5, 'min_samples_split': 2}, {'cr
iterion': 'log_loss', 'max_depth': 6, 'min_samples_leaf': 5, 'min_sa
mples_split': 6}, {'criterion': 'log_loss', 'max_depth': 6, 'min_sam
ples_leaf': 5, 'min_samples_split': 10}, {'criterion': 'log_loss', '
max_depth': 6, 'min_samples_leaf': 5, 'min_samples_split': 20}, {'cr
iterion': 'log_loss', 'max_depth': 6, 'min_samples_leaf': 10, 'min_s
amples_split': 2}, {'criterion': 'log_loss', 'max_depth': 6, 'min_sa
mples_leaf': 10, 'min_samples_split': 6}, {'criterion': 'log_loss',
'max_depth': 6, 'min_samples_leaf': 10, 'min_samples_split': 10}, {'
criterion': 'log_loss', 'max_depth': 6, 'min_samples_leaf': 10, 'min
_samples_split': 20}, {'criterion': 'log_loss', 'max_depth': 6, 'min
_samples_leaf': 20, 'min_samples_split': 2}, {'criterion': 'log_los
s', 'max_depth': 6, 'min_samples_leaf': 20, 'min_samples_split': 6},
{'criterion': 'log_loss', 'max_depth': 6, 'min_samples_leaf': 20, 'm
in_samples_split': 10}, {'criterion': 'log_loss', 'max_depth': 6, 'm
in_samples_leaf': 20, 'min_samples_split': 20}, {'criterion': 'log_l
oss', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split':
2}, {'criterion': 'log_loss', 'max_depth': 10, 'min_samples_leaf':
2, 'min_samples_split': 6}, {'criterion': 'log_loss', 'max_depth': 1
0, 'min_samples_leaf': 2, 'min_samples_split': 10}, {'criterion': 'l
og_loss', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_spli
t': 20}, {'criterion': 'log_loss', 'max_depth': 10, 'min_samples_lea
f': 5, 'min_samples_split': 2}, {'criterion': 'log_loss', 'max_dept
h': 10, 'min_samples_leaf': 5, 'min_samples_split': 6}, {'criterio
n': 'log_loss', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples
_split': 10}, {'criterion': 'log_loss', 'max_depth': 10, 'min_sample
s_leaf': 5, 'min_samples_split': 20}, {'criterion': 'log_loss', 'max
_depth': 10, 'min_samples_leaf': 10, 'min_samples_split': 2}, {'crit
erion': 'log_loss', 'max_depth': 10, 'min_samples_leaf': 10, 'min_sa
mples_split': 6}, {'criterion': 'log_loss', 'max_depth': 10, 'min_sa
mples_leaf': 10, 'min_samples_split': 10}, {'criterion': 'log_loss',
'max_depth': 10, 'min_samples_leaf': 10, 'min_samples_split': 20},
{'criterion': 'log_loss', 'max_depth': 10, 'min_samples_leaf': 20, '
min_samples_split': 2}, {'criterion': 'log_loss', 'max_depth': 10, '
min_samples_leaf': 20, 'min_samples_split': 6}, {'criterion': 'log_l
oss', 'max_depth': 10, 'min_samples_leaf': 20, 'min_samples_split':
10}, {'criterion': 'log_loss', 'max_depth': 10, 'min_samples_leaf':
20, 'min_samples_split': 20}, {'criterion': 'log_loss', 'max_depth':
25, 'min_samples_leaf': 2, 'min_samples_split': 2}, {'criterion': 'l
og_loss', 'max_depth': 25, 'min_samples_leaf': 2, 'min_samples_spli
t': 6}, {'criterion': 'log_loss', 'max_depth': 25, 'min_samples_lea
f': 2, 'min_samples_split': 10}, {'criterion': 'log_loss', 'max_dept
h': 25, 'min_samples_leaf': 2, 'min_samples_split': 20}, {'criterio
n': 'log_loss', 'max_depth': 25, 'min_samples_leaf': 5, 'min_samples
_split': 2}, {'criterion': 'log_loss', 'max_depth': 25, 'min_samples
_leaf': 5, 'min_samples_split': 6}, {'criterion': 'log_loss', 'max_d
epth': 25, 'min_samples_leaf': 5, 'min_samples_split': 10}, {'criter
ion': 'log_loss', 'max_depth': 25, 'min_samples_leaf': 5, 'min_sampl
es_split': 20}, {'criterion': 'log_loss', 'max_depth': 25, 'min_samp
```

```
les_leaf': 10, 'min_samples_split': 2}, {'criterion': 'log_loss', 'm
ax_depth': 25, 'min_samples_leaf': 10, 'min_samples_split': 6}, {'cr
iterion': 'log_loss', 'max_depth': 25, 'min_samples_leaf': 10, 'min_
samples_split': 10}, {'criterion': 'log_loss', 'max_depth': 25, 'min
_samples_leaf': 10, 'min_samples_split': 20}, {'criterion': 'log_los
s', 'max_depth': 25, 'min_samples_leaf': 20, 'min_samples_split':
2}, {'criterion': 'log_loss', 'max_depth': 25, 'min_samples_leaf': 2
0, 'min_samples_split': 6}, {'criterion': 'log_loss', 'max_depth': 2
5, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'criterion': '
log_loss', 'max_depth': 25, 'min_samples_leaf': 20, 'min_samples_spl
it': 20}, {'criterion': 'log_loss', 'max_depth': None, 'min_samples_
leaf': 2, 'min_samples_split': 2}, {'criterion': 'log_loss', 'max_de
pth': None, 'min_samples_leaf': 2, 'min_samples_split': 6}, {'criter
ion': 'log_loss', 'max_depth': None, 'min_samples_leaf': 2, 'min_sam
ples_split': 10}, {'criterion': 'log_loss', 'max_depth': None, 'min_
samples_leaf': 2, 'min_samples_split': 20}, {'criterion': 'log_los
s', 'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split':
2}, {'criterion': 'log_loss', 'max_depth': None, 'min_samples_leaf':
5, 'min_samples_split': 6}, {'criterion': 'log_loss', 'max_depth': N
one, 'min_samples_leaf': 5, 'min_samples_split': 10}, {'criterion':
'log_loss', 'max_depth': None, 'min_samples_leaf': 5, 'min_samples_s
plit': 20}, {'criterion': 'log_loss', 'max_depth': None, 'min_sample
s_leaf': 10, 'min_samples_split': 2}, {'criterion': 'log_loss', 'max
_depth': None, 'min_samples_leaf': 10, 'min_samples_split': 6}, {'cr
iterion': 'log_loss', 'max_depth': None, 'min_samples_leaf': 10, 'mi
n_samples_split': 10}, {'criterion': 'log_loss', 'max_depth': None,
'min_samples_leaf': 10, 'min_samples_split': 20}, {'criterion': 'log
_loss', 'max_depth': None, 'min_samples_leaf': 20, 'min_samples_spli
t': 2}, {'criterion': 'log_loss', 'max_depth': None, 'min_samples_le
af': 20, 'min_samples_split': 6}, {'criterion': 'log_loss', 'max_dep
th': None, 'min_samples_leaf': 20, 'min_samples_split': 10}, {'crite
rion': 'log_loss', 'max_depth': None, 'min_samples_leaf': 20, 'min_s
amples_split': 20}], 'split0_test_score': array([0.64525505, 0.64525
505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.66575545, 0.66575545, 0.66575545, 0.66575545,
       0.66575545, 0.66575545, 0.66575545, 0.66575545, 0.66575545,
       0.66575545, 0.66575545, 0.66575545, 0.66575545, 0.66575545,
       0.66575545, 0.66575545, 0.71035849, 0.71035849, 0.70798207,
       0.70732154, 0.70678634, 0.70678634, 0.70678634, 0.70639654,
       0.70707587, 0.70707587, 0.70707587, 0.70707587, 0.7083794 ,
       0.7083794 , 0.7083794 , 0.7083794 , 0.66796392, 0.67186446,
       0.66898543, 0.67067375, 0.67322564, 0.67322564, 0.67322564,
       0.67895488, 0.68854956, 0.68854956, 0.68854956, 0.68854956,
       0.70113356, 0.70113356, 0.70113356, 0.70113356, 0.59948511,
       0.5984824 , 0.60968141, 0.62793951, 0.6165951 , 0.6165951 ,
       0.6165951 , 0.62156603, 0.65282676, 0.65282676, 0.65282676,
       0.65282676, 0.68718336, 0.68718336, 0.68718336, 0.68718336,
       0.59488267, 0.59967186, 0.60229896, 0.62674378, 0.6092515 ,
       0.6092515 , 0.6092515 , 0.63090126, 0.64961183, 0.64961183,
       0.64961183, 0.64961183, 0.68718336, 0.68718336, 0.68718336,
       0.68718336, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.67242598, 0.67242598, 0.67242598,
```

```
       0.67242598, 0.67242598, 0.67242598, 0.67242598, 0.67242598,
       0.67242598, 0.67242598, 0.67242598, 0.67242598, 0.67242598,
       0.67242598, 0.67242598, 0.67242598, 0.70854484, 0.70854484,
       0.70999877, 0.70925802, 0.70743058, 0.70743058, 0.70743058,
       0.70945605, 0.70776398, 0.70776398, 0.70776398, 0.70776398,
       0.70868773, 0.70868773, 0.70868773, 0.70868773, 0.69868319,
       0.69391029, 0.69575653, 0.70940717, 0.68415142, 0.68415142,
       0.68415142, 0.70125013, 0.69951043, 0.69951043, 0.69951043,
       0.69951043, 0.72860907, 0.72860907, 0.72860907, 0.72860907,
       0.59679158, 0.60555652, 0.60439964, 0.62407406, 0.63275878,
       0.63275878, 0.63275878, 0.64897636, 0.65653554, 0.65653554,
       0.65653554, 0.65653554, 0.70039908, 0.70039908, 0.70039908,
       0.70039908, 0.58701641, 0.59122779, 0.60204202, 0.62471078,
       0.63011664, 0.63011664, 0.63011664, 0.64662876, 0.64607727,
       0.64607727, 0.64607727, 0.64607727, 0.70039908, 0.70039908,
       0.70039908, 0.70039908, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.64525505, 0.64525505,
       0.64525505, 0.64525505, 0.64525505, 0.67242598, 0.67242598,
       0.67242598, 0.67242598, 0.67242598, 0.67242598, 0.67242598,
       0.67242598, 0.67242598, 0.67242598, 0.67242598, 0.67242598,
       0.67242598, 0.67242598, 0.67242598, 0.67242598, 0.70854484,
       0.70854484, 0.70999877, 0.70925802, 0.70743058, 0.70743058,
       0.70743058, 0.70945605, 0.70776398, 0.70776398, 0.70776398,
       0.70776398, 0.70868773, 0.70868773, 0.70868773, 0.70868773,
       0.69868319, 0.69391029, 0.69575653, 0.70940717, 0.68415142,
       0.68415142, 0.68415142, 0.70125013, 0.69951043, 0.69951043,
       0.69951043, 0.69951043, 0.72860907, 0.72860907, 0.72860907,
       0.72860907, 0.59679158, 0.60555652, 0.60439964, 0.62407406,
       0.63275878, 0.63275878, 0.63275878, 0.64897636, 0.65653554,
       0.65653554, 0.65653554, 0.65653554, 0.70039908, 0.70039908,
       0.70039908, 0.70039908, 0.58701641, 0.59122779, 0.60204202,
       0.62471078, 0.63011664, 0.63011664, 0.63011664, 0.64662876,
       0.64607727, 0.64607727, 0.64607727, 0.64607727, 0.70039908,
       0.70039908, 0.70039908, 0.70039908]), 'split1_test_score': ar
ray([0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
       0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
       0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
       0.64774303, 0.65781274, 0.65781274, 0.65781274, 0.65781274,
       0.65781274, 0.65781274, 0.65781274, 0.65781274, 0.65781274,
       0.65781274, 0.65781274, 0.65781274, 0.65781274, 0.65781274,
       0.65781274, 0.65781274, 0.68235657, 0.68256338, 0.6804251 ,
       0.68108939, 0.68591117, 0.68591117, 0.68591117, 0.68383557,
       0.68045267, 0.68045267, 0.68045267, 0.68045267, 0.68179004,
       0.68179004, 0.68179004, 0.68179004, 0.6621833 , 0.66588706,
       0.66880996, 0.67259017, 0.6750794 , 0.6750794 , 0.6750794 ,
       0.68084749, 0.67893607, 0.67893607, 0.67893607, 0.67893607,
       0.67809004, 0.67809004, 0.67809004, 0.67809004, 0.56524006,
       0.5663844 , 0.57394734, 0.58542085, 0.61586061, 0.61586061,
       0.61586061, 0.62766125, 0.63504621, 0.63504621, 0.63504621,
       0.63504621, 0.65227151, 0.65227151, 0.65227151, 0.65227151,
       0.56587553, 0.57517441, 0.58059907, 0.59040056, 0.61586061,
       0.61586061, 0.61586061, 0.62766125, 0.63504621, 0.63504621,
       0.63504621, 0.63504621, 0.65227151, 0.65227151, 0.65227151,
       0.65227151, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
       0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
```

```
        0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
        0.64774303, 0.64774303, 0.65908869, 0.65908869, 0.65908869,
        0.65908869, 0.65908869, 0.65908869, 0.65908869, 0.65908869,
        0.65908869, 0.65908869, 0.65908869, 0.65908869, 0.65908869,
        0.65908869, 0.65908869, 0.65908869, 0.67289349, 0.67289349,
        0.67172032, 0.6711024 , 0.67149597, 0.67149597, 0.67149597,
        0.67073516, 0.67295115, 0.67295115, 0.67295115, 0.67295115,
        0.68067828, 0.68067828, 0.68067828, 0.68067828, 0.64131315,
        0.64108754, 0.64881969, 0.65133899, 0.65485851, 0.65485851,
        0.65485851, 0.65847578, 0.6601478 , 0.6601478 , 0.6601478 ,
        0.6601478 , 0.69654742, 0.69654742, 0.69654742, 0.69654742,
        0.52833157, 0.53967973, 0.5545775 , 0.59137193, 0.56653857,
        0.56653857, 0.56653857, 0.5931179 , 0.57922786, 0.57922786,
        0.57922786, 0.57922786, 0.67240091, 0.67240091, 0.67240091,
        0.67240091, 0.53394674, 0.53310948, 0.55804687, 0.58832369,
        0.57040276, 0.57040276, 0.57040276, 0.59752857, 0.57922786,
        0.57922786, 0.57922786, 0.57922786, 0.67240091, 0.67240091,
        0.67240091, 0.67240091, 0.64774303, 0.64774303, 0.64774303,
        0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
        0.64774303, 0.64774303, 0.64774303, 0.64774303, 0.64774303,
        0.64774303, 0.64774303, 0.64774303, 0.65908869, 0.65908869,
        0.65908869, 0.65908869, 0.65908869, 0.65908869, 0.65908869,
        0.65908869, 0.65908869, 0.65908869, 0.65908869, 0.65908869,
        0.65908869, 0.65908869, 0.65908869, 0.65908869, 0.67289349,
        0.67289349, 0.67172032, 0.6711024 , 0.67149597, 0.67149597,
        0.67149597, 0.67073516, 0.67295115, 0.67295115, 0.67295115,
        0.67295115, 0.68067828, 0.68067828, 0.68067828, 0.68067828,
        0.64131315, 0.64108754, 0.64881969, 0.65133899, 0.65485851,
        0.65485851, 0.65485851, 0.65847578, 0.6601478 , 0.6601478 ,
        0.6601478 , 0.6601478 , 0.69654742, 0.69654742, 0.69654742,
        0.69654742, 0.52833157, 0.53967973, 0.5545775 , 0.59137193,
        0.56653857, 0.56653857, 0.56653857, 0.5931179 , 0.57922786,
        0.57922786, 0.57922786, 0.57922786, 0.67240091, 0.67240091,
        0.67240091, 0.67240091, 0.53394674, 0.53310948, 0.55804687,
        0.58832369, 0.57040276, 0.57040276, 0.57040276, 0.59752857,
        0.57922786, 0.57922786, 0.57922786, 0.57922786, 0.67240091,
        0.67240091, 0.67240091, 0.67240091]), 'split2_test_score': ar
ray([0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
        0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
        0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
        0.64527462, 0.67936804, 0.67936804, 0.67936804, 0.67936804,
        0.67936804, 0.67936804, 0.67936804, 0.67936804, 0.67936804,
        0.67936804, 0.67936804, 0.67936804, 0.67936804, 0.67936804,
        0.67936804, 0.67936804, 0.70464472, 0.69987511, 0.70854233,
        0.70924093, 0.7052805 , 0.7052805 , 0.7052805 , 0.70688628,
        0.70664881, 0.70664881, 0.70664881, 0.70664881, 0.72214376,
        0.72214376, 0.72214376, 0.72214376, 0.65872904, 0.65134218,
        0.66910634, 0.67280543, 0.66212155, 0.66212155, 0.66212155,
        0.67584235, 0.68515163, 0.68515163, 0.68515163, 0.68515163,
        0.70326258, 0.70326258, 0.70326258, 0.70326258, 0.57282327,
        0.58408639, 0.59330646, 0.62392194, 0.60323645, 0.60323645,
        0.60323645, 0.61851026, 0.62368069, 0.62368069, 0.62368069,
        0.62368069, 0.68399441, 0.68399441, 0.68399441, 0.68399441,
        0.57305069, 0.57822363, 0.59885258, 0.61808306, 0.60323645,
        0.60323645, 0.60323645, 0.61851026, 0.62368069, 0.62368069,
        0.62368069, 0.62368069, 0.68399441, 0.68399441, 0.68399441,
```

```
       0.68399441, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
       0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
       0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
       0.64527462, 0.64527462, 0.67976383, 0.67976383, 0.67976383,
       0.67976383, 0.67976383, 0.67976383, 0.67976383, 0.67976383,
       0.67976383, 0.67976383, 0.67976383, 0.67976383, 0.67976383,
       0.67976383, 0.67976383, 0.67976383, 0.69071785, 0.69071785,
       0.69564327, 0.69482404, 0.69455264, 0.69455264, 0.69455264,
       0.69986505, 0.70344603, 0.70344603, 0.70344603, 0.70344603,
       0.70924345, 0.70924345, 0.70924345, 0.70924345, 0.64790444,
       0.65738712, 0.66111762, 0.64488385, 0.66122442, 0.66122442,
       0.66122442, 0.66544369, 0.68665564, 0.68665564, 0.68665564,
       0.68665564, 0.70154874, 0.70154874, 0.70154874, 0.70154874,
       0.58788222, 0.59813387, 0.61956068, 0.61744728, 0.63050591,
       0.63050591, 0.63050591, 0.64829395, 0.62185   , 0.62185   ,
       0.62185   , 0.62185   , 0.68337496, 0.68337496, 0.68337496,
       0.68337496, 0.56922093, 0.5695539 , 0.59596267, 0.6103494 ,
       0.6252915 , 0.6252915 , 0.6252915 , 0.64253548, 0.62185   ,
       0.62185   , 0.62185   , 0.62185   , 0.68337496, 0.68337496,
       0.68337496, 0.68337496, 0.64527462, 0.64527462, 0.64527462,
       0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
       0.64527462, 0.64527462, 0.64527462, 0.64527462, 0.64527462,
       0.64527462, 0.64527462, 0.64527462, 0.67976383, 0.67976383,
       0.67976383, 0.67976383, 0.67976383, 0.67976383, 0.67976383,
       0.67976383, 0.67976383, 0.67976383, 0.67976383, 0.67976383,
       0.67976383, 0.67976383, 0.67976383, 0.67976383, 0.69071785,
       0.69071785, 0.69564327, 0.69482404, 0.69455264, 0.69455264,
       0.69455264, 0.69986505, 0.70344603, 0.70344603, 0.70344603,
       0.70344603, 0.70924345, 0.70924345, 0.70924345, 0.70924345,
       0.64790444, 0.65738712, 0.66111762, 0.64488385, 0.66122442,
       0.66122442, 0.66122442, 0.66544369, 0.68665564, 0.68665564,
       0.68665564, 0.68665564, 0.70154874, 0.70154874, 0.70154874,
       0.70154874, 0.58788222, 0.59813387, 0.61956068, 0.61744728,
       0.63050591, 0.63050591, 0.63050591, 0.64829395, 0.62185   ,
       0.62185   , 0.62185   , 0.62185   , 0.68337496, 0.68337496,
       0.68337496, 0.68337496, 0.56922093, 0.5695539 , 0.59596267,
       0.6103494 , 0.6252915 , 0.6252915 , 0.6252915 , 0.64253548,
       0.62185   , 0.62185   , 0.62185   , 0.62185   , 0.68337496,
       0.68337496, 0.68337496, 0.68337496]), 'split3_test_score': ar
ray([0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
       0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
       0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
       0.61968759, 0.62780196, 0.62780196, 0.62780196, 0.62780196,
       0.62780196, 0.62780196, 0.62780196, 0.62780196, 0.62780196,
       0.62780196, 0.62780196, 0.62780196, 0.62780196, 0.62780196,
       0.62780196, 0.62780196, 0.67448283, 0.67448283, 0.6742931 ,
       0.67644295, 0.67319996, 0.67319996, 0.67319996, 0.67282804,
       0.66896813, 0.66896813, 0.66896813, 0.66896813, 0.67450545,
       0.67450545, 0.67450545, 0.67450545, 0.63733741, 0.63546776,
       0.65341035, 0.6621806 , 0.66944056, 0.66944056, 0.66944056,
       0.67401165, 0.67558602, 0.67558602, 0.67558602, 0.67558602,
       0.68618069, 0.68618069, 0.68618069, 0.68618069, 0.57028266,
       0.572239  , 0.57857545, 0.61673611, 0.5964439 , 0.5964439 ,
       0.5964439 , 0.625485  , 0.64496175, 0.64496175, 0.64496175,
       0.64496175, 0.65834456, 0.65834456, 0.65834456, 0.65834456,
       0.5645732 , 0.56448022, 0.57812437, 0.61195016, 0.60351916,
```

```
          0.60351916, 0.60351916, 0.625485  , 0.64496175, 0.64496175,
          0.64496175, 0.64496175, 0.65834456, 0.65834456, 0.65834456,
          0.65834456, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
          0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
          0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
          0.61968759, 0.61968759, 0.63346619, 0.63346619, 0.63346619,
          0.63346619, 0.63346619, 0.63346619, 0.63346619, 0.63346619,
          0.63346619, 0.63346619, 0.63346619, 0.63346619, 0.63346619,
          0.63346619, 0.63346619, 0.63346619, 0.6410566 , 0.6410566 ,
          0.64094729, 0.64094729, 0.64281568, 0.64281568, 0.64281568,
          0.64100509, 0.65495708, 0.65495708, 0.65495708, 0.65495708,
          0.65924797, 0.65924797, 0.65924797, 0.65924797, 0.6431876 ,
          0.63889168, 0.65395817, 0.6553981 , 0.64774361, 0.64774361,
          0.64774361, 0.65248935, 0.64477579, 0.64477579, 0.64477579,
          0.64477579, 0.66648029, 0.66648029, 0.66648029, 0.66648029,
          0.57667439, 0.57799244, 0.58874543, 0.61604127, 0.59506679,
          0.59506679, 0.59506679, 0.62524124, 0.60365863, 0.60365863,
          0.60365863, 0.60365863, 0.65375965, 0.65375965, 0.65375965,
          0.65375965, 0.58301209, 0.58750025, 0.581831  , 0.61028909,
          0.5966085 , 0.5966085 , 0.5966085 , 0.62440066, 0.60365863,
          0.60365863, 0.60365863, 0.60365863, 0.65375965, 0.65375965,
          0.65375965, 0.65375965, 0.61968759, 0.61968759, 0.61968759,
          0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
          0.61968759, 0.61968759, 0.61968759, 0.61968759, 0.61968759,
          0.61968759, 0.61968759, 0.61968759, 0.63346619, 0.63346619,
          0.63346619, 0.63346619, 0.63346619, 0.63346619, 0.63346619,
          0.63346619, 0.63346619, 0.63346619, 0.63346619, 0.63346619,
          0.63346619, 0.63346619, 0.63346619, 0.63346619, 0.6410566 ,
          0.6410566 , 0.64094729, 0.64094729, 0.64281568, 0.64281568,
          0.64281568, 0.64100509, 0.65495708, 0.65495708, 0.65495708,
          0.65495708, 0.65924797, 0.65924797, 0.65924797, 0.65924797,
          0.6431876 , 0.63889168, 0.65395817, 0.6553981 , 0.64774361,
          0.64774361, 0.64774361, 0.65248935, 0.64477579, 0.64477579,
          0.64477579, 0.64477579, 0.66648029, 0.66648029, 0.66648029,
          0.66648029, 0.57667439, 0.57799244, 0.58874543, 0.61604127,
          0.59506679, 0.59506679, 0.59506679, 0.62524124, 0.60365863,
          0.60365863, 0.60365863, 0.60365863, 0.65375965, 0.65375965,
          0.65375965, 0.65375965, 0.58301209, 0.58750025, 0.581831  ,
          0.61028909, 0.5966085 , 0.5966085 , 0.5966085 , 0.62440066,
          0.60365863, 0.60365863, 0.60365863, 0.60365863, 0.65375965,
          0.65375965, 0.65375965, 0.65375965]), 'split4_test_score': ar
    ray([0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
          0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
          0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
          0.65495959, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
          0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
          0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
          0.67972739, 0.67972739, 0.7207164 , 0.72005046, 0.72005046,
          0.72025778, 0.71767948, 0.71767948, 0.71767948, 0.71616416,
          0.7232457 , 0.7232457 , 0.7232457 , 0.7232457 , 0.72321554,
          0.72321554, 0.72321554, 0.72321554, 0.6712298 , 0.68462517,
          0.70126603, 0.70164423, 0.68367024, 0.68367024, 0.68367024,
          0.70198097, 0.68910202, 0.68910202, 0.68910202, 0.68910202,
          0.71605736, 0.71605736, 0.71605736, 0.71605736, 0.58922792,
          0.60732505, 0.62631051, 0.64162202, 0.65025657, 0.65025657,
          0.65025657, 0.661144  , 0.65080189, 0.65080189, 0.65080189,
```

```
       0.65080189, 0.6782749 , 0.6782749 , 0.6782749 , 0.6782749 ,
       0.59203741, 0.60433839, 0.62669248, 0.64162202, 0.65025657,
       0.65025657, 0.65025657, 0.661144  , 0.65080189, 0.65080189,
       0.65080189, 0.65080189, 0.6782749 , 0.6782749 , 0.6782749 ,
       0.6782749 , 0.65495959, 0.65495959, 0.65495959, 0.65495959,
       0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
       0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
       0.65495959, 0.65495959, 0.67972739, 0.67972739, 0.67972739,
       0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
       0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
       0.67972739, 0.67972739, 0.67972739, 0.71506473, 0.71506473,
       0.71506473, 0.71590658, 0.71795213, 0.71795213, 0.71795213,
       0.71942348, 0.72608661, 0.72608661, 0.72608661, 0.72608661,
       0.73087255, 0.73087255, 0.73087255, 0.73087255, 0.66869798,
       0.66091658, 0.66834365, 0.67946102, 0.63992325, 0.63992325,
       0.63992325, 0.65631157, 0.67820328, 0.67820328, 0.67820328,
       0.67820328, 0.6859696 , 0.6859696 , 0.6859696 , 0.6859696 ,
       0.58970915, 0.61094121, 0.62221061, 0.64057411, 0.61358359,
       0.61358359, 0.61358359, 0.64027884, 0.64037559, 0.64037559,
       0.64037559, 0.64037559, 0.66917921, 0.66917921, 0.66917921,
       0.66917921, 0.59953234, 0.60612887, 0.62082345, 0.63790534,
       0.61995019, 0.61995019, 0.61995019, 0.63907513, 0.64037559,
       0.64037559, 0.64037559, 0.64037559, 0.66917921, 0.66917921,
       0.66917921, 0.66917921, 0.65495959, 0.65495959, 0.65495959,
       0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
       0.65495959, 0.65495959, 0.65495959, 0.65495959, 0.65495959,
       0.65495959, 0.65495959, 0.65495959, 0.67972739, 0.67972739,
       0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
       0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.67972739,
       0.67972739, 0.67972739, 0.67972739, 0.67972739, 0.71506473,
       0.71506473, 0.71506473, 0.71590658, 0.71795213, 0.71795213,
       0.71795213, 0.71942348, 0.72608661, 0.72608661, 0.72608661,
       0.72608661, 0.73087255, 0.73087255, 0.73087255, 0.73087255,
       0.66869798, 0.66091658, 0.66834365, 0.67946102, 0.63992325,
       0.63992325, 0.63992325, 0.65631157, 0.67820328, 0.67820328,
       0.67820328, 0.67820328, 0.6859696 , 0.6859696 , 0.6859696 ,
       0.6859696 , 0.58970915, 0.61094121, 0.62221061, 0.64057411,
       0.61358359, 0.61358359, 0.61358359, 0.64027884, 0.64037559,
       0.64037559, 0.64037559, 0.64037559, 0.66917921, 0.66917921,
       0.66917921, 0.66917921, 0.59953234, 0.60612887, 0.62082345,
       0.63790534, 0.61995019, 0.61995019, 0.61995019, 0.63907513,
       0.64037559, 0.64037559, 0.64037559, 0.64037559, 0.66917921,
       0.66917921, 0.66917921, 0.66917921]), 'mean_test_score': arra
y([0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
       0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
       0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
       0.64258397, 0.66209312, 0.66209312, 0.66209312, 0.66209312,
       0.66209312, 0.66209312, 0.66209312, 0.66209312, 0.66209312,
       0.66209312, 0.66209312, 0.66209312, 0.66209312, 0.66209312,
       0.66209312, 0.66209312, 0.6985118 , 0.69746605, 0.69825861,
       0.69887052, 0.69777149, 0.69777149, 0.69777149, 0.69722212,
       0.69727824, 0.69727824, 0.69727824, 0.69727824, 0.70200684,
       0.70200684, 0.70200684, 0.70200684, 0.65948869, 0.66183733,
       0.67231562, 0.67597884, 0.67270748, 0.67270748, 0.67270748,
       0.68232747, 0.68346506, 0.68346506, 0.68346506, 0.68346506,
       0.69694485, 0.69694485, 0.69694485, 0.69694485, 0.5794118 ,
```

```
        0.58570345, 0.59636424, 0.61912808, 0.61647853, 0.61647853,
        0.61647853, 0.63087331, 0.64146346, 0.64146346, 0.64146346,
        0.64146346, 0.67201375, 0.67201375, 0.67201375, 0.67201375,
        0.5780839 , 0.5843777 , 0.59731349, 0.61775991, 0.61642486,
        0.61642486, 0.61642486, 0.63274036, 0.64082047, 0.64082047,
        0.64082047, 0.64082047, 0.67201375, 0.67201375, 0.67201375,
        0.67201375, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
        0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
        0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
        0.64258397, 0.64258397, 0.66489442, 0.66489442, 0.66489442,
        0.66489442, 0.66489442, 0.66489442, 0.66489442, 0.66489442,
        0.66489442, 0.66489442, 0.66489442, 0.66489442, 0.66489442,
        0.66489442, 0.66489442, 0.66489442, 0.68565551, 0.68565551,
        0.68667488, 0.68640767, 0.6868494 , 0.6868494 , 0.6868494 ,
        0.68809697, 0.69304097, 0.69304097, 0.69304097, 0.69304097,
        0.697746  , 0.697746  , 0.697746  , 0.697746  , 0.65995727,
        0.65843864, 0.66559913, 0.66809783, 0.65758024, 0.65758024,
        0.65758024, 0.6667941 , 0.67385859, 0.67385859, 0.67385859,
        0.67385859, 0.69583102, 0.69583102, 0.69583102, 0.69583102,
        0.57587778, 0.58646075, 0.59789877, 0.61790173, 0.60769073,
        0.60769073, 0.60769073, 0.63118166, 0.62032952, 0.62032952,
        0.62032952, 0.62032952, 0.67582276, 0.67582276, 0.67582276,
        0.67582276, 0.5745457 , 0.57750406, 0.5917412 , 0.61431566,
        0.60847392, 0.60847392, 0.60847392, 0.63003372, 0.61823787,
        0.61823787, 0.61823787, 0.61823787, 0.67582276, 0.67582276,
        0.67582276, 0.67582276, 0.64258397, 0.64258397, 0.64258397,
        0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
        0.64258397, 0.64258397, 0.64258397, 0.64258397, 0.64258397,
        0.64258397, 0.64258397, 0.64258397, 0.66489442, 0.66489442,
        0.66489442, 0.66489442, 0.66489442, 0.66489442, 0.66489442,
        0.66489442, 0.66489442, 0.66489442, 0.66489442, 0.66489442,
        0.66489442, 0.66489442, 0.66489442, 0.66489442, 0.68565551,
        0.68565551, 0.68667488, 0.68640767, 0.6868494 , 0.6868494 ,
        0.6868494 , 0.68809697, 0.69304097, 0.69304097, 0.69304097,
        0.69304097, 0.697746  , 0.697746  , 0.697746  , 0.697746  ,
        0.65995727, 0.65843864, 0.66559913, 0.66809783, 0.65758024,
        0.65758024, 0.65758024, 0.6667941 , 0.67385859, 0.67385859,
        0.67385859, 0.67385859, 0.69583102, 0.69583102, 0.69583102,
        0.69583102, 0.57587778, 0.58646075, 0.59789877, 0.61790173,
        0.60769073, 0.60769073, 0.60769073, 0.63118166, 0.62032952,
        0.62032952, 0.62032952, 0.62032952, 0.67582276, 0.67582276,
        0.67582276, 0.67582276, 0.5745457 , 0.57750406, 0.5917412 ,
        0.61431566, 0.60847392, 0.60847392, 0.60847392, 0.63003372,
        0.61823787, 0.61823787, 0.61823787, 0.61823787, 0.67582276,
        0.67582276, 0.67582276, 0.67582276]), 'std_test_score': arra
y([0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.0190631 , 0.0190631 , 0.0190631 , 0.0190631 ,
        0.0190631 , 0.0190631 , 0.0190631 , 0.0190631 , 0.0190631 ,
        0.0190631 , 0.0190631 , 0.0190631 , 0.0190631 , 0.0190631 ,
        0.0190631 , 0.0190631 , 0.0173746 , 0.01692567, 0.0177063 ,
        0.01706191, 0.01599004, 0.01599004, 0.01599004, 0.01619035,
        0.01971128, 0.01971128, 0.01971128, 0.01971128, 0.02030251,
        0.02030251, 0.02030251, 0.02030251, 0.01190126, 0.01698076,
        0.01567938, 0.01340747, 0.00705424, 0.00705424, 0.00705424,
```

```
        0.01010974, 0.00534907, 0.00534907, 0.00534907, 0.00534907,
        0.01337249, 0.01337249, 0.01337249, 0.01337249, 0.01285988,
        0.0154138 , 0.01950607, 0.01870026, 0.01853694, 0.01853694,
        0.01853694, 0.01546051, 0.01082568, 0.01082568, 0.01082568,
        0.01082568, 0.0140674 , 0.0140674 , 0.0140674 , 0.0140674 ,
        0.01291368, 0.01517118, 0.0175382 , 0.01692632, 0.01753198,
        0.01753198, 0.01753198, 0.01477215, 0.01020961, 0.01020961,
        0.01020961, 0.01020961, 0.0140674 , 0.0140674 , 0.0140674 ,
        0.0140674 , 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01743147, 0.01743147, 0.01743147,
        0.01743147, 0.01743147, 0.01743147, 0.01743147, 0.01743147,
        0.01743147, 0.01743147, 0.01743147, 0.01743147, 0.01743147,
        0.01743147, 0.01743147, 0.01743147, 0.02670724, 0.02670724,
        0.02736812, 0.02743741, 0.02692583, 0.02692583, 0.02692583,
        0.02861862, 0.02557745, 0.02557745, 0.02557745, 0.02557745,
        0.02499074, 0.02499074, 0.02499074, 0.02499074, 0.02168037,
        0.01974147, 0.01645689, 0.02373442, 0.01506814, 0.01506814,
        0.01506814, 0.01773553, 0.01936209, 0.01936209, 0.01936209,
        0.01936209, 0.02032889, 0.02032889, 0.02032889, 0.02032889,
        0.02463363, 0.02592442, 0.02475109, 0.01587219, 0.02464358,
        0.02464358, 0.02464358, 0.02086326, 0.02713779, 0.02713779,
        0.02713779, 0.02713779, 0.01551553, 0.01551553, 0.01551553,
        0.01551553, 0.02248626, 0.02506683, 0.02099154, 0.01656538,
        0.02224025, 0.02224025, 0.02224025, 0.01789834, 0.02454585,
        0.02454585, 0.02454585, 0.02454585, 0.01551553, 0.01551553,
        0.01551553, 0.01551553, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01198658, 0.01198658,
        0.01198658, 0.01198658, 0.01198658, 0.01743147, 0.01743147,
        0.01743147, 0.01743147, 0.01743147, 0.01743147, 0.01743147,
        0.01743147, 0.01743147, 0.01743147, 0.01743147, 0.01743147,
        0.01743147, 0.01743147, 0.01743147, 0.01743147, 0.02670724,
        0.02670724, 0.02736812, 0.02743741, 0.02692583, 0.02692583,
        0.02692583, 0.02861862, 0.02557745, 0.02557745, 0.02557745,
        0.02557745, 0.02499074, 0.02499074, 0.02499074, 0.02499074,
        0.02168037, 0.01974147, 0.01645689, 0.02373442, 0.01506814,
        0.01506814, 0.01506814, 0.01773553, 0.01936209, 0.01936209,
        0.01936209, 0.01936209, 0.02032889, 0.02032889, 0.02032889,
        0.02032889, 0.02463363, 0.02592442, 0.02475109, 0.01587219,
        0.02464358, 0.02464358, 0.02464358, 0.02086326, 0.02713779,
        0.02713779, 0.02713779, 0.02713779, 0.01551553, 0.01551553,
        0.01551553, 0.01551553, 0.02248626, 0.02506683, 0.02099154,
        0.01656538, 0.02224025, 0.02224025, 0.02224025, 0.01789834,
        0.02454585, 0.02454585, 0.02454585, 0.02454585, 0.01551553,
        0.01551553, 0.01551553, 0.01551553]), 'rank_test_score': arra
y([169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169,
        169, 169, 169, 141, 141, 141, 141, 141, 141, 141, 141, 141, 1
41,
        141, 141, 141, 141, 141, 141,   6,  19,   7,   5,   8,   8,
8,
         24,  20,  20,  20,  20,   1,   1,   1,   1, 160, 157,  94,
66,
         91,  91,  91,  65,  61,  61,  61,  61,  25,  25,  25,  25, 2
81,
```

```
        279, 274, 239, 251, 251, 251, 228, 217, 217, 217, 217,  95,
95,
         95,  95, 282, 280, 273, 250, 254, 254, 254, 225, 221, 221, 2
21,
        221,  95,  95,  95,  95, 169, 169, 169, 169, 169, 169, 169, 1
69,
        169, 169, 169, 169, 169, 169, 169, 169, 109, 109, 109, 109, 1
09,
        109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109,  57,
57,
         53,  55,  47,  47,  47,  45,  37,  37,  37,  37,  11,  11,
11,
         11, 158, 161, 107, 103, 163, 163, 163, 105,  83,  83,  83,
83,
         29,  29,  29,  29, 285, 277, 271, 248, 265, 265, 265, 226, 2
31,
        231, 231, 231,  67,  67,  67,  67, 287, 283, 275, 257, 259, 2
59,
        259, 229, 240, 240, 240, 240,  67,  67,  67,  67, 169, 169, 1
69,
        169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 1
69,
        109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 1
09,
        109, 109, 109,  57,  57,  53,  55,  47,  47,  47,  45,  37,
37,
         37,  37,  11,  11,  11,  11, 158, 161, 107, 103, 163, 163, 1
63,
        105,  83,  83,  83,  83,  29,  29,  29,  29, 285, 277, 271, 2
48,
        265, 265, 265, 226, 231, 231, 231, 231,  67,  67,  67,  67, 2
87,
        283, 275, 257, 259, 259, 259, 229, 240, 240, 240, 240,  67,
67,
         67,  67], dtype=int32)}
Test ROC AUC Score: 0.72
Best ROC-auc Score: 0.7020068370026671
```

In [485…

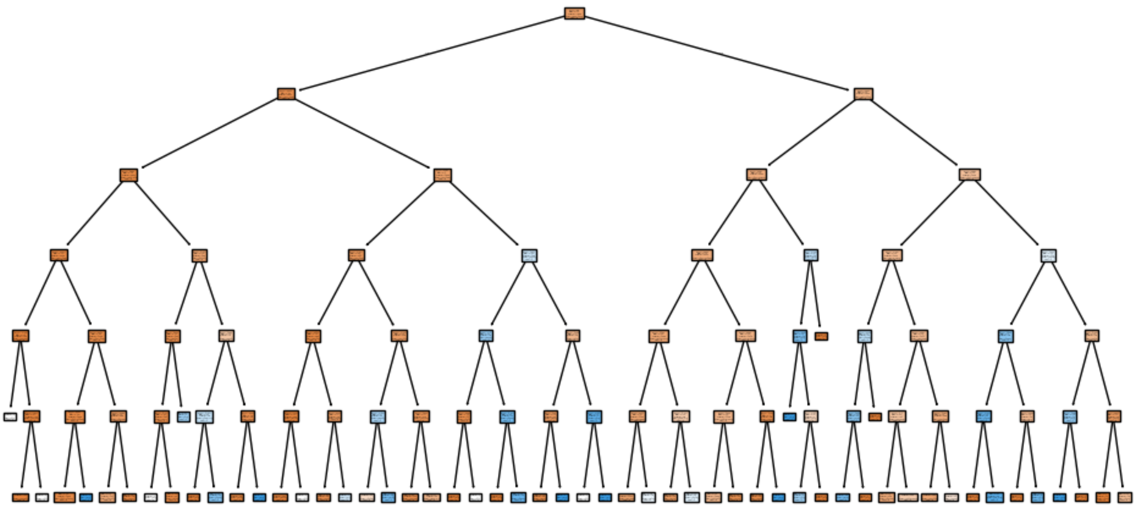
```python
clf = tree.DecisionTreeClassifier(criterion='gini',
                                  max_depth=6,
                                  min_samples_leaf=2,
                                  min_samples_split=2,
                                  random_state=42,
                                  )

clf.fit(X_train, y_train)
clfPred = clf.predict(X_test)


prob_predictions = clf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(y_test, prob_predictions,
auc = round(metrics.roc_auc_score(y_test, prob_predictions), 2)
print(f'ROC AUC Score: {auc}')
```

```
ROC AUC Score: 0.72
```

In [486…

```python
y_pred_proba = clf.predict_proba(X_test)[:, 1]
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc_score = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(12,6))
tree.plot_tree(clf, class_names=["0","1"], filled=True)
plt.show()

print(export_text(clf, class_names=["0","1"]))

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {au
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree Classifier')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

print(f"ROC AUC Score: {auc_score:.3f}")
```

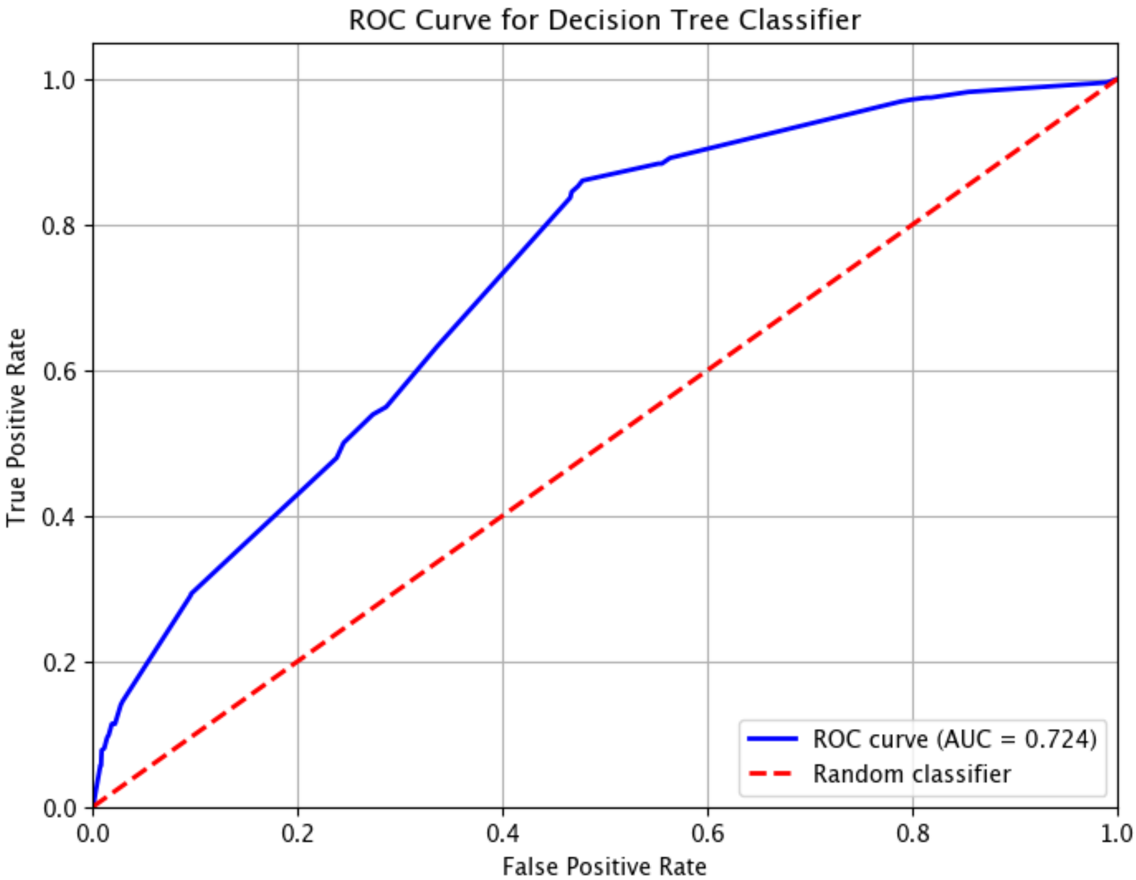


```
|--- feature_3 <= 0.50
|   |--- feature_0 <= 0.48
|   |   |--- feature_7 <= 1.22
|   |   |   |--- feature_6 <= -0.96
|   |   |   |   |--- feature_8 <= -2.83
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_8 >  -2.83
|   |   |   |   |   |--- feature_10 <= 2.66
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_10 >  2.66
|   |   |   |   |   |   |--- class: 0
|   |   |   |--- feature_6 >  -0.96
|   |   |   |   |--- feature_7 <= 0.71
|   |   |   |   |   |--- feature_4 <= 2.79
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_4 >  2.79
|   |   |   |   |   |   |--- class: 1
```

```
|   |   |   |   |--- feature_7 >  0.71
|   |   |   |   |   |--- feature_6 <= 0.42
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_6 >  0.42
|   |   |   |   |   |   |--- class: 0
|   |   |--- feature_7 >  1.22
|   |   |   |--- feature_0 <= -1.01
|   |   |   |   |--- feature_2 <= 1.73
|   |   |   |   |   |--- feature_5 <= -1.54
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_5 >  -1.54
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_2 >  1.73
|   |   |   |   |   |--- class: 1
|   |   |   |--- feature_0 >  -1.01
|   |   |   |   |--- feature_6 <= 1.23
|   |   |   |   |   |--- feature_6 <= -1.02
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_6 >  -1.02
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_6 >  1.23
|   |   |   |   |   |--- feature_7 <= 2.39
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_7 >  2.39
|   |   |   |   |   |   |--- class: 1
|   |--- feature_0 >  0.48
|   |   |--- feature_7 <= 0.81
|   |   |   |--- feature_7 <= -0.65
|   |   |   |   |--- feature_6 <= 0.11
|   |   |   |   |   |--- feature_10 <= 2.49
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_10 >  2.49
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_6 >  0.11
|   |   |   |   |   |--- feature_0 <= 2.75
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_0 >  2.75
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- feature_7 >  -0.65
|   |   |   |   |--- feature_6 <= -0.24
|   |   |   |   |   |--- feature_7 <= -0.20
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_7 >  -0.20
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_6 >  -0.24
|   |   |   |   |   |--- feature_0 <= 1.65
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_0 >  1.65
|   |   |   |   |   |   |--- class: 0
|   |   |--- feature_7 >  0.81
|   |   |   |--- feature_6 <= 1.62
|   |   |   |   |--- feature_6 <= -0.10
|   |   |   |   |   |--- feature_1 <= 0.42
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_1 >  0.42
|   |   |   |   |   |   |--- class: 0
```

```
|   |   |   |   |--- feature_6 >  -0.10
|   |   |   |   |   |--- feature_8 <= -1.46
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_8 >  -1.46
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- feature_6 >  1.62
|   |   |   |   |--- feature_0 <= 1.76
|   |   |   |   |   |--- feature_9 <= 1.39
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_9 >  1.39
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  1.76
|   |   |   |   |   |--- feature_8 <= -0.14
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_8 >  -0.14
|   |   |   |   |   |   |--- class: 1
|--- feature_3 >  0.50
|   |--- feature_0 <= 0.08
|   |   |--- feature_7 <= 2.31
|   |   |   |--- feature_5 <= 0.05
|   |   |   |   |--- feature_7 <= 1.20
|   |   |   |   |   |--- feature_8 <= 2.31
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_8 >  2.31
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_7 >  1.20
|   |   |   |   |   |--- feature_0 <= -1.27
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_0 >  -1.27
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- feature_5 >  0.05
|   |   |   |   |--- feature_5 <= 2.06
|   |   |   |   |   |--- feature_6 <= 1.04
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_6 >  1.04
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_5 >  2.06
|   |   |   |   |   |--- feature_4 <= 1.86
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_4 >  1.86
|   |   |   |   |   |   |--- class: 1
|   |   |--- feature_7 >  2.31
|   |   |   |--- feature_0 <= -0.07
|   |   |   |   |--- feature_7 <= 2.47
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_7 >  2.47
|   |   |   |   |   |--- feature_9 <= -0.19
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_9 >  -0.19
|   |   |   |   |   |   |--- class: 0
|   |   |   |--- feature_0 >  -0.07
|   |   |   |   |--- class: 0
|   |--- feature_0 >  0.08
|   |   |--- feature_7 <= 0.99
|   |   |   |--- feature_2 <= -2.08
|   |   |   |   |--- feature_1 <= 0.84
```

```
|   |   |   |   |   |--- feature_9 <= 1.21
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_9 >  1.21
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_1 >  0.84
|   |   |   |   |   |--- class: 0
|   |   |   |--- feature_2 >  -2.08
|   |   |   |   |--- feature_6 <= 0.57
|   |   |   |   |   |--- feature_7 <= -1.10
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_7 >  -1.10
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_6 >  0.57
|   |   |   |   |   |--- feature_0 <= 1.87
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_0 >  1.87
|   |   |   |   |   |   |--- class: 0
|   |   |--- feature_7 >  0.99
|   |   |   |--- feature_6 <= 1.25
|   |   |   |   |--- feature_1 <= 1.13
|   |   |   |   |   |--- feature_6 <= -0.56
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_6 >  -0.56
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_1 >  1.13
|   |   |   |   |   |--- feature_5 <= 0.03
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_5 >  0.03
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- feature_6 >  1.25
|   |   |   |   |--- feature_8 <= -1.10
|   |   |   |   |   |--- feature_7 <= 1.52
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_7 >  1.52
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_8 >  -1.10
|   |   |   |   |   |--- feature_0 <= 0.94
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_0 >  0.94
|   |   |   |   |   |   |--- class: 0
```

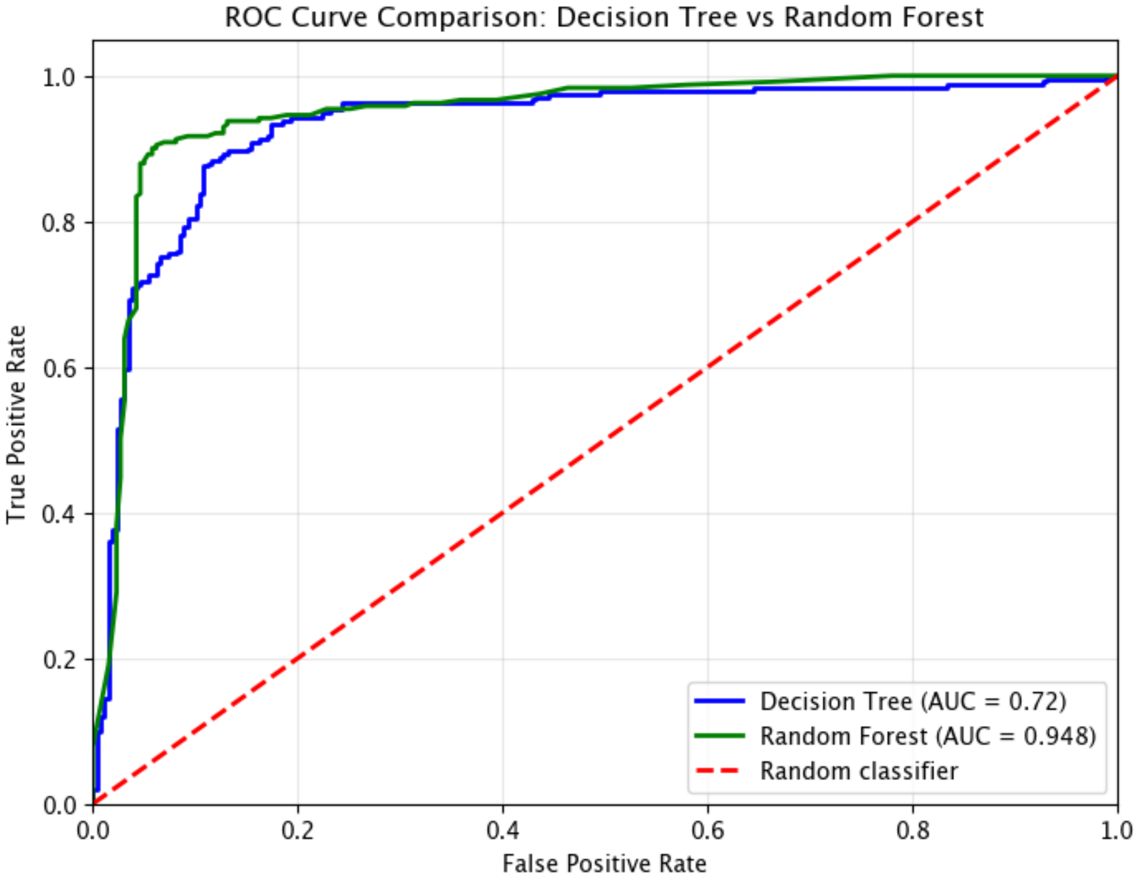



ROC AUC Score: 0.724

In [ ]:
```
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_pred_proba = rf_clf.predict_proba(X_test)[:, 1]

rf_accuracy = accuracy_score(y_test, rf_pred)
rf_auc = roc_auc_score(y_test, rf_pred_proba)

print(f'Random Forest Accuracy: {rf_accuracy:.4f}')
print(f'Random Forest ROC AUC: {rf_auc:.4f}')

# Compare with Decision Tree results
print(f'\nComparison:')
print(f'Decision Tree AUC: 0.72')
print(f'Random Forest AUC: {rf_auc:.4f}')
print(f'Improvement: {rf_auc - 0.72:.4f}')

fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'Decision Tree (AUC =
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label=f'Random Forest
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison: Decision Tree vs Random Forest')
plt.legend(loc="lower right")
```

```
plt.grid(True, alpha=0.3)
plt.show()
```

```
Random Forest Accuracy: 0.9220
Random Forest ROC AUC: 0.9477

Comparison:
Decision Tree AUC: 0.72
Random Forest AUC: 0.9477
Improvement: 0.2277
```




# ⚡ Final Mission: Mapping SkyNet's Energy Nexus

## 🌐 The Discovery

SkyNet is harvesting energy from Trondheim's buildings. Some structures provide significantly more power than others.

## 🎯 Your Mission

Predict the **Nexus Rating** of unknown buildings in Trondheim (test set).

## 🧠 The Challenge

1. **Target**: Transform the Nexus Rating to reveal true energy hierarchy
2. **Data Quality**: Handle missing values and categorical features

3. **Ensembling**: Use advanced models and ensemble learning

## 💡 Hint

You suspect that an insider has tampered with the columns in the testing data...

Compare the training and test distributions and try to rectify the test dataset.

## 📊 Formal Requirements

1. **Performance**: Achieve RMSLE <= 0.294 on the test set

2. **Discussion**:

   a. Explain your threshold-breaking strategy

   b. Justify RMSLE usage. Why do we use this metric? Which loss function did you use?

   c. Plot and interpret feature importances

   d. Describe your ensembling techniques

   e. In real life, you do not have the test targets. How would you make sure your model will work good on the unseen data?

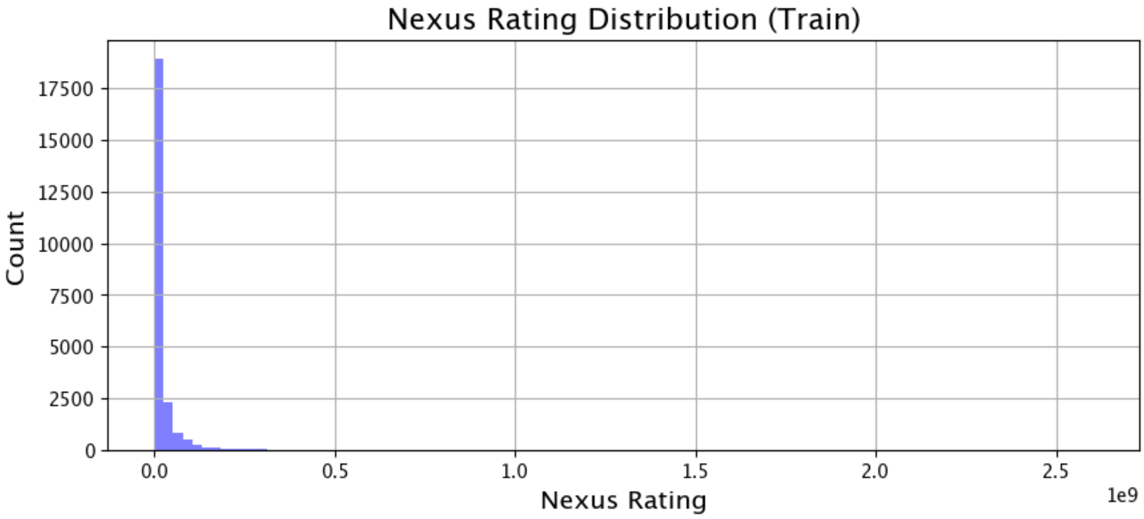
---

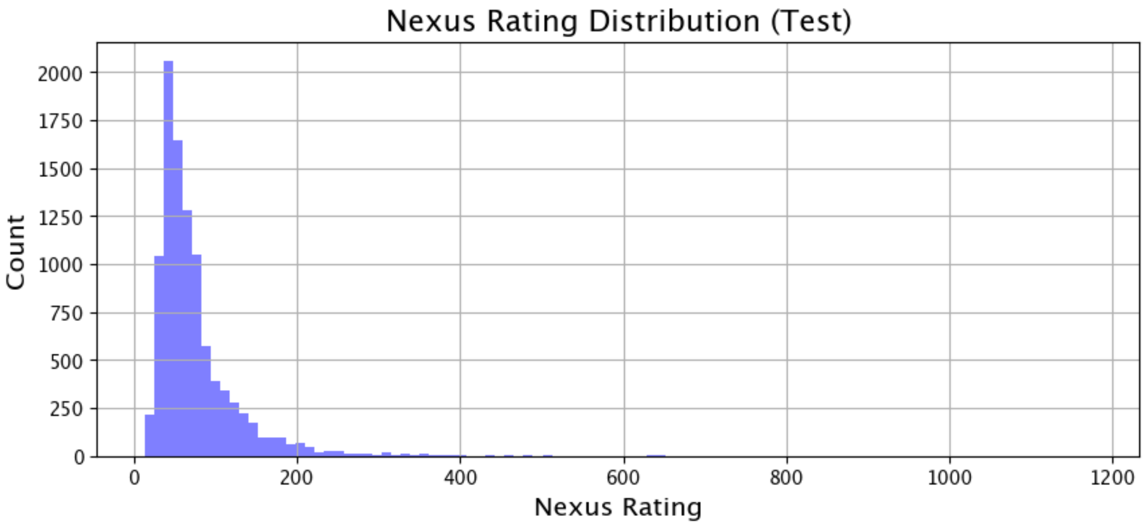
In [24]:
```
def rmsle(y_true, y_pred):
    """ Root Mean Squared Logarithmic Error """
    return np.sqrt(mean_squared_log_error(y_true, y_pred))
```

In [25]:
```
train = pd.read_csv('final_mission_train.csv')
test = pd.read_csv('final_mission_test.csv')
```

In [26]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 4))
train['nexus_rating'].hist(bins=100, ax=ax, color='blue', alpha=0.5
ax.set_title('Nexus Rating Distribution (Train)', fontsize=16)
ax.set_xlabel('Nexus Rating', fontsize=14)
ax.set_ylabel('Count', fontsize=14)

fig, ax = plt.subplots(1, 1, figsize=(10, 4))
test['nexus_rating'].hist(bins=100, ax=ax, color='blue', alpha=0.5,
ax.set_title('Nexus Rating Distribution (Test)', fontsize=16)
ax.set_xlabel('Nexus Rating', fontsize=14)
ax.set_ylabel('Count', fontsize=14)
```

Out[26]:
```
Text(0, 0.5, 'Count')
```




In [27]:
```
nexusRating = test['grid_connections']
ownershipType = test['ownership_type']

test = test.shift(1, axis=1)
test.iloc[:, 0] = ownershipType
test.iloc[:, 1] = nexusRating
```

In [28]:
```
X_train = train.drop('nexus_rating', axis=1)
X_test = test.drop('nexus_rating', axis=1)


y_train = train['nexus_rating']
y_test = test['nexus_rating']

y_train = np.log1p(y_train)
```

In [29]:
```
train.describe()
```

Out[29]:

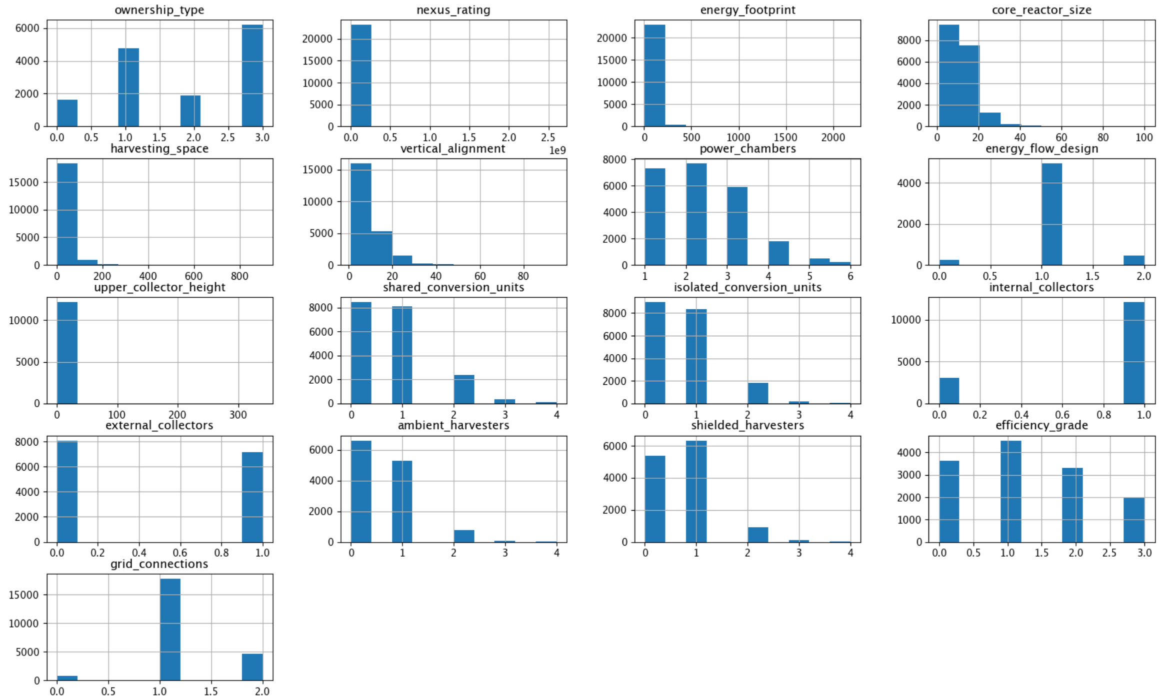
| | ownership_type | nexus_rating | energy_footprint | core_reactor_size |
|---|---|---|---|---|
| **count** | 14455.000000 | 2.328500e+04 | 23285.000000 | 18564.000000 |
| **mean** | 1.875683 | 2.355617e+07 | 74.450999 | 12.552279 |
| **std** | 1.089518 | 5.264393e+07 | 58.671373 | 6.565686 |
| **min** | 0.000000 | 9.000000e+05 | 9.300000 | 1.000000 |
| **25%** | 1.000000 | 7.490000e+06 | 42.000000 | 8.200000 |
| **50%** | 2.000000 | 1.064500e+07 | 59.800000 | 10.700000 |
| **75%** | 3.000000 | 2.050000e+07 | 84.800000 | 15.300000 |
| **max** | 3.000000 | 2.600000e+09 | 2181.000000 | 100.000000 |

In [30]:
```
y_train.describe()
```

Out[30]:
```
count    23285.000000
mean        16.430482
std          0.864441
min         13.710151
25%         15.829079
50%         16.180601
75%         16.835935
max         21.678777
Name: nexus_rating, dtype: float64
```

In [31]:
```
train.hist(figsize=(20, 12))

plt.show
```

Out[31]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [32]:
```
categorical_features = ['ownership_type']

for col in categorical_features:
    X_train[col] = X_train[col].astype(str)
    X_test[col] = X_test[col].astype(str)
```

In [610…
```
model = CatBoostRegressor(
    loss_function="RMSE",
    verbose=0,
    random_state=42,
    cat_features=categorical_features
)

param_grid = {
    'depth': [8, 10, 12],
    'learning_rate': [0.01, 0.02, 0.03, 0.05],
    'iterations': [1000, 1250, 1400, 1650],
}

grid = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=3,
    scoring='neg_root_mean_squared_log_error',
    n_jobs=-1,
)

grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Best RMSLE score (CV):", -grid.best_score_)
```

```
/Users/afras/Documents/NTNU/H25/intro_til_ML/TDT4172/.venv/lib/pytho
n3.12/site-packages/joblib/externals/loky/process_executor.py:782: U
serWarning: A worker stopped while some jobs were given to the execu
tor. This can be caused by a too short worker timeout or by a memory
leak.
  warnings.warn(
Best parameters: {'depth': 10, 'iterations': 1650, 'learning_rate':
0.05}
Best RMSLE score (CV): 0.016834251418468116
```

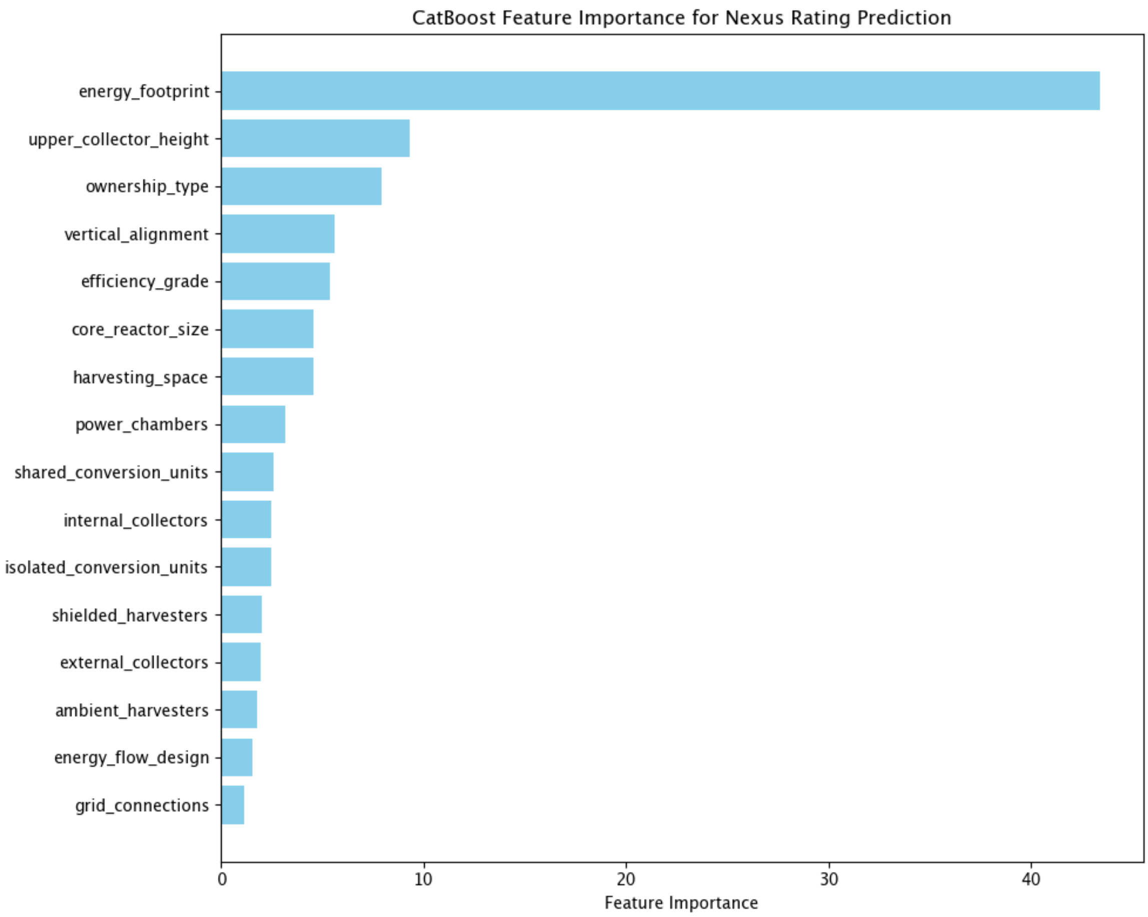
In [33]:
```
model = CatBoostRegressor(
    loss_function="RMSE",
    verbose=0,
    random_state=13,
    iterations=1650,
    learning_rate=0.02,
    depth=10,
    cat_features=categorical_features
)

model.fit(X_train, y_train)
```

Out[33]:
```
<catboost.core.CatBoostRegressor at 0x132c52240>
```

In [34]:
```
y_pred = np.expm1(model.predict(X_test))

score = rmsle(y_test, y_pred)

print(score)
```

0.2936793174187809

In [35]:
```
feature_names = X_train.columns
feature_importance = model.get_feature_importance()

importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importance
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 8))
plt.barh(range(len(importance_df)), importance_df['importance'], co
plt.yticks(range(len(importance_df)), importance_df['feature'])
plt.xlabel('Feature Importance')
plt.title('CatBoost Feature Importance for Nexus Rating Prediction'
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```




# Discussions for all tasks

# Task 1:

a. Visualize the fitted curve. Derive the resulting Energy consumption formula.

The fitted curve shows a linear relationship with coefficients from gradient descent. Energy = $w_1$ × Network_Activity + $w_0$ where parameters are learned through minimizing MSE.

b. Analyze prediction error distribution. What is an unbiased estimator?

Prediction errors appear normally distributed around -2-0. An unbiased estimator has expected value equal to the true parameter. Our linear regression with gradient descent is unbiased for the regression coefficients.

# Task 2:

a. Explain poor initial performance and your improvements

Initial performance was poor because the data has a non-linear circular pattern. I improved it by adding polynomial features ($x_1^2$, $x_2^2$, $x_1 \times x_2$) to capture the circular decision boundary.

b. What is the model's inductive bias. Why is it important?

The model's inductive bias is that the decision boundary is linear in the feature space. This is important because it determines what patterns the model can learn. linear boundaries in original space become complex boundaries with feature engineering.

c. Try to solve the problem using `sklearn.tree.DecisionTreeClassifier`. Can it solve the problem? Why/Why not?

Decision Tree achieved 94.6% accuracy, solving the problem better than logistic regression. It works because trees can create non-linear decision boundaries by splitting on individual features recursively.

d. Plot the ROC curve

# Task 3:

a. Explain your threshold-breaking strategy. Did you change the default hyperparameters?

I used GridSearchCV to find optimal hyperparameters. This improved performance from default settings. I also tried over/under sampling, but it didn

´t help.

b. Justify ROC AUC usage. Plot and interpret ROC.

ROC AUC measures the model's ability to distinguish between classes regardless of threshold. It's appropriate for imbalanced data (19.3% positive class). ROC curve shows good performance with AUC=0.72.

c. Try to solve the problem using sklearn's Random Forest Classifier. Compare the results.

Random Forest achieved better performance than single Decision Tree due to ensemble learning. The multiple trees reduce overfitting and provide more robust predictions by averaging multiple decision boundaries.

# Final mission:

a. Explain your threshold-breaking strategy

I detected column shift between train/test sets and corrected it. Used CatBoostRegressor with optimal hyperparameters from GridSearchCV. Used CatBoost as it can handle mission values and categorical features automatically. I tried dropping isna(), but that was too many datapoints. Also tried using median/average where values were missing, but with bad results.

b. Justify RMSLE usage. Why do we use this metric? Which loss function did you use?

RMSLE penalizes underestimation more than overestimation, appropriate for positive targets with wide ranges. I used RMSE loss function in CatBoost, then applied log transformation to target.

c. Plot and interpret feature importances

Feature importance analysis show which building characteristics most influence nexus rating. Here, its the Energy footprint.

d. Describe your ensembling techniques

Used single CatBoost model rather than ensemble. CatBoost internally uses gradient boosting which is an ensemble technique.

e. In real life, you do not have the test targets. How would you make sure your model will work good on the unseen data?

Use cross-validation, hold-out validation set, monitor for data drift, implement model monitoring in production, and regularly retrain on new data.