Practice Lab - Trees Ensemble

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt


RANDOM_STATE = 55
```

1. Introduction

Datatset

- This dataset is obtained from Kaggle: Heart Failure Prediction Dataset

In [4]:
```
# Load the dataset
df = pd.read_csv("...")
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina
0	40	M	ATA	140	289	0	Normal	172	N
1	49	F	NAP	160	180	0	Normal	156	N
2	37	M	ATA	130	283	0	ST	98	N
3	48	F	ASY	138	214	0	Normal	108	Y
4	54	M	NAP	150	195	0	Normal	122	N

2. One-hot encoding using Pandas

In [5]:
```
# remove binary data, hot encode columns with 3 or more values
cat_variables = ['Sex',
'ChestPainType',
'RestingECG',
'ExerciseAngina',
'ST_Slope'
]
```

In [6]:
```
# replace the columns with the one-hot encoded ones and keep the columns outside 'colu
df = pd.get_dummies(data = df,
```

```
                    prefix = cat_variables,
                    columns = cat_variables)
```

In [7]:
```
df.head()
```

Out[7]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	Sex_F	Sex_M	ChestPain
0	40	140	289	0	172	0.0	0	0	1	
1	49	160	180	0	156	1.0	1	1	0	
2	37	130	283	0	98	0.0	0	0	1	
3	48	138	214	0	108	1.5	1	1	0	
4	54	150	195	0	122	0.0	0	0	1	

5 rows × 21 columns

In [10]:
```
features = [x for x in df.columns if x not in 'HeartDisease'] ## Removing target varic
```

In [9]:
```
# feature variables after one-hot encoding
print(len(features))
```

```
20
```

3. Splitting the Dataset

In [13]:
```
X_train, X_val, y_train, y_val = train_test_split(df[features], df['HeartDisease'], tr
```

In [14]:
```
print(f'train samples: {len(X_train)}\ntest samples: {len(X_val)}')
print(f'target proportion: {sum(y_train)/len(y_train):.4f}')
```

```
train samples: 734
test samples: 184
target proportion: 0.5518
```

4. Building the Models

4.1 Decision Tree

- min_samples_split: The minimum number of samples required to split an internal node.
  - Choosing a higher min_samples_split can reduce the number of splits and may help to reduce overfitting.
- max_depth: The maximum depth of the tree.
  - Choosing a lower max_depth can reduce the number of splits and may help to reduce overfitting.

In [15]:
```
min_samples_split_list = [2,10, 30, 50, 100, 200, 300, 700] ## If the number is an int
max_depth_list = [1,2, 3, 4, 8, 16, 32, 64, None] # None means that there is no depth
```

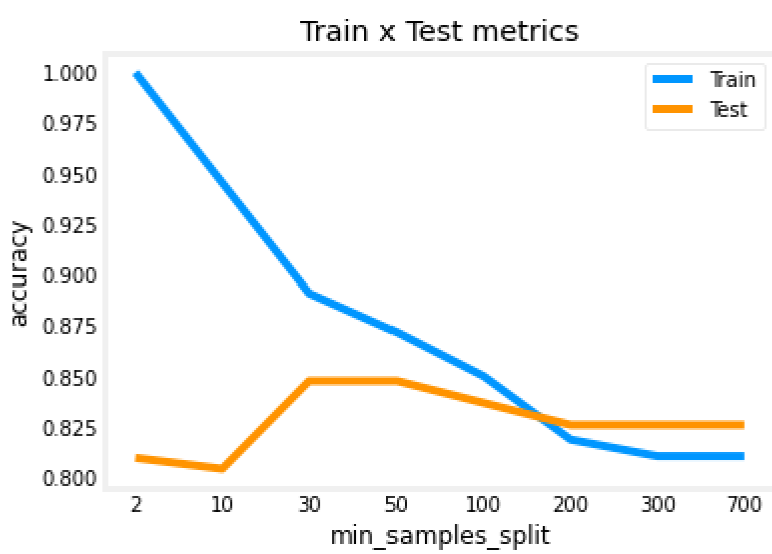
In [16]:

```python
accuracy_list_train = []
accuracy_list_val = []
for min_samples_split in min_samples_split_list:
    # fit the model at the same time we define it, because the fit function returns th
    model = DecisionTreeClassifier(min_samples_split = min_samples_split,
                                   random_state = RANDOM_STATE).fit(X_train,y_train)
    predictions_train = model.predict(X_train) ## The predicted values for the train d
    predictions_val = model.predict(X_val) ## The predicted values for the test datase
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_val = accuracy_score(predictions_val,y_val)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_val.append(accuracy_val)

plt.title('Train x Test metrics')
plt.xlabel('min_samples_split')
plt.ylabel('accuracy')
plt.xticks(ticks = range(len(min_samples_split_list )),labels=min_samples_split_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_val)
plt.legend(['Train','Test'])
```

Out[16]:

```
<matplotlib.legend.Legend at 0x7f15eccb0e90>
```



- Increasing min_samples_split from 10 to 30, and from 30 to 50 improves the validation accuracy (while bringing the training accuracy closer to the validation accuracy).

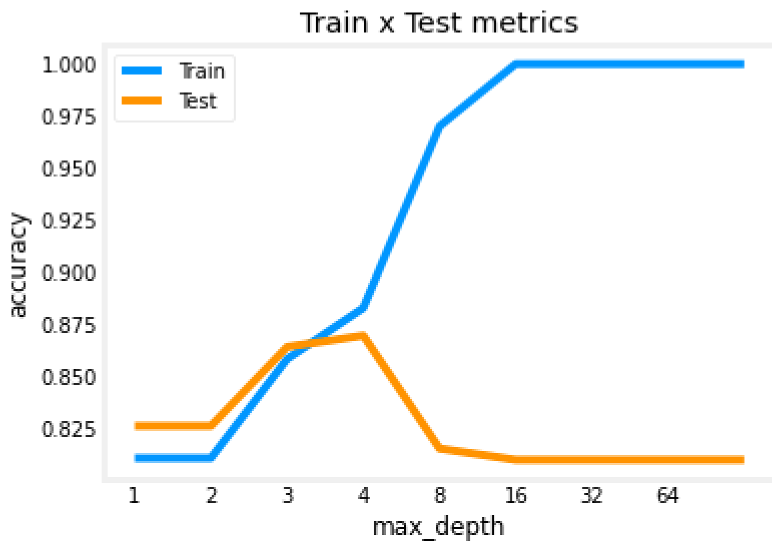
In [17]:

```python
accuracy_list_train = []
accuracy_list_val = []
for max_depth in max_depth_list:
    # fit the model at the same time we define it, because the fit function returns th
    model = DecisionTreeClassifier(max_depth = max_depth,
                                   random_state = RANDOM_STATE).fit(X_train,y_train)
    predictions_train = model.predict(X_train) ## The predicted values for the train d
    predictions_val = model.predict(X_val) ## The predicted values for the test datase
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_val = accuracy_score(predictions_val,y_val)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_val.append(accuracy_val)

plt.title('Train x Test metrics')
```

```python
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.xticks(ticks = range(len(max_depth_list )),labels=max_depth_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_val)
plt.legend(['Train','Test'])
```

Out[17]:
```
<matplotlib.legend.Legend at 0x7f15ec750c90>
```



we can choose the best values for these two hyper-parameters for our model to be:

- max_depth = 3
- min_samples_split = 50

In [18]:
```python
decision_tree_model = DecisionTreeClassifier(min_samples_split = 50,
                                             max_depth = 3,
                                             random_state = RANDOM_STATE).fit(X_train,
```

In [19]:
```python
print(f"Metrics train:\n\tAccuracy score: {accuracy_score(decision_tree_model.predict(
print(f"Metrics test:\n\tAccuracy score: {accuracy_score(decision_tree_model.predict(X
```

```
Metrics train:
	Accuracy score: 0.8583
Metrics test:
	Accuracy score: 0.8641
```

4.2 Random Forest

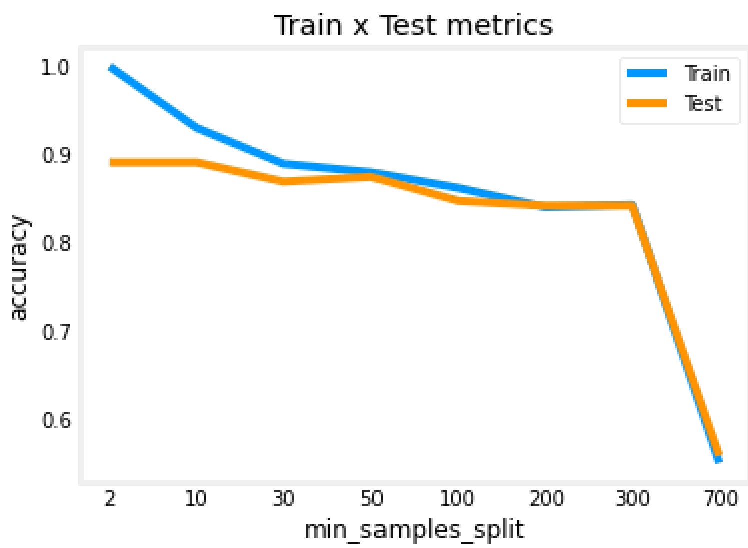
In [20]:
```python
min_samples_split_list = [2,10, 30, 50, 100, 200, 300, 700]  ## If the number is an in
                                             ## If it is a float, then it is the perce
max_depth_list = [2, 4, 8, 16, 32, 64, None]
n_estimators_list = [10,50,100,500]
```

In [21]:
```python
accuracy_list_train = []
accuracy_list_val = []
for min_samples_split in min_samples_split_list:
    # fit the model at the same time we define it, because the fit function returns th
    model = RandomForestClassifier(min_samples_split = min_samples_split,
                                   random_state = RANDOM_STATE).fit(X_train,y_train)
```

```
    predictions_train = model.predict(X_train) ## The predicted values for the train d
    predictions_val = model.predict(X_val) ## The predicted values for the test datase
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_val = accuracy_score(predictions_val,y_val)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_val.append(accuracy_val)

plt.title('Train x Test metrics')
plt.xlabel('min_samples_split')
plt.ylabel('accuracy')
plt.xticks(ticks = range(len(min_samples_split_list )),labels=min_samples_split_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_val)
plt.legend(['Train','Test'])
```

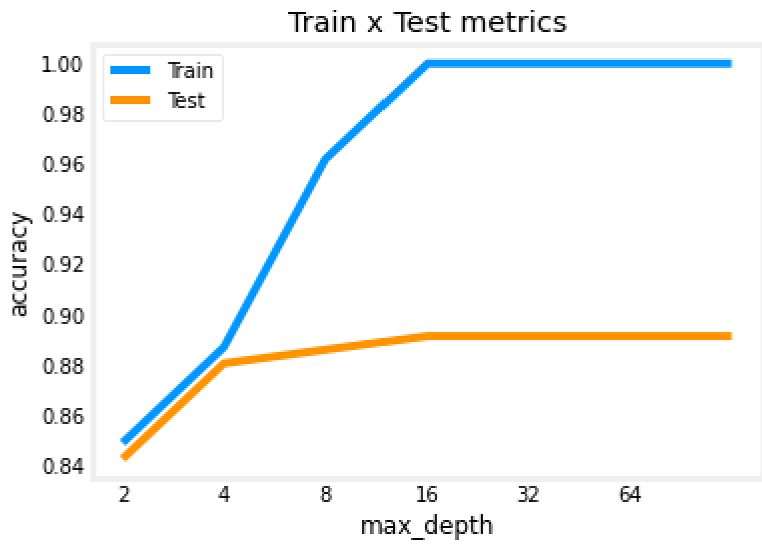
Out[21]: <matplotlib.legend.Legend at 0x7f15e73508d0>



In [22]:
```
accuracy_list_train = []
accuracy_list_val = []
for max_depth in max_depth_list:
    # You can fit the model at the same time you define it, because the fit function r
    model = RandomForestClassifier(max_depth = max_depth,
                                   random_state = RANDOM_STATE).fit(X_train,y_train)
    predictions_train = model.predict(X_train) ## The predicted values for the train d
    predictions_val = model.predict(X_val) ## The predicted values for the test datase
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_val = accuracy_score(predictions_val,y_val)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_val.append(accuracy_val)

plt.title('Train x Test metrics')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.xticks(ticks = range(len(max_depth_list )),labels=max_depth_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_val)
plt.legend(['Train','Test'])
```

Out[22]: <matplotlib.legend.Legend at 0x7f15e72daad0>



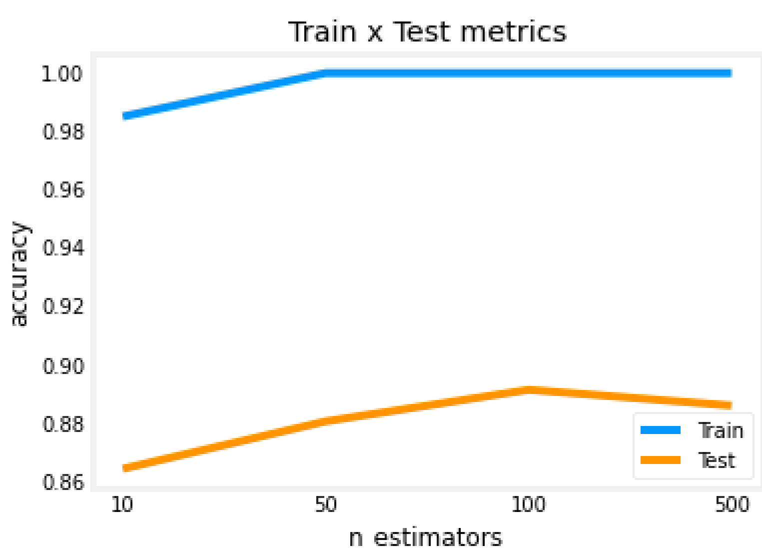
In [23]:

```
accuracy_list_train = []
accuracy_list_val = []
for n_estimators in n_estimators_list:
    # fit the model at the same time we define it, because the fit function returns th
    model = RandomForestClassifier(n_estimators = n_estimators,
                                   random_state = RANDOM_STATE).fit(X_train,y_train)
    predictions_train = model.predict(X_train) ## The predicted values for the train d
    predictions_val = model.predict(X_val) ## The predicted values for the test datase
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_val = accuracy_score(predictions_val,y_val)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_val.append(accuracy_val)

plt.title('Train x Test metrics')
plt.xlabel('n_estimators')
plt.ylabel('accuracy')
plt.xticks(ticks = range(len(n_estimators_list )),labels=n_estimators_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_val)
plt.legend(['Train','Test'])
```

Out[23]:

```
<matplotlib.legend.Legend at 0x7f15e72475d0>
```



we fit a random forest with the following parameters:

- max_depth: 8
- min_samples_split: 10
- n_estimators: 100

In [24]:
```
random_forest_model = RandomForestClassifier(n_estimators = 100,
                                             max_depth = 8,
                                             min_samples_split = 10).fit(X_train,y_tra
```

In [25]:
```
print(f"Metrics train:\n\tAccuracy score: {accuracy_score(random_forest_model.predict(
```

```
Metrics train:
        Accuracy score: 0.9183
Metrics test:
        Accuracy score: 0.9022
```

4.3 XGBoost

In [26]:
```
n = int(len(X_train)*0.8) ## we use 80% to train and 20% to eval
```

In [27]:
```
X_train_fit, X_train_eval, y_train_fit, y_train_eval = X_train[:n], X_train[n:], y_tra
```

We can then set a large number of estimators, because we can stop if the cost function stops decreasing.

In [28]:
```
xgb_model = XGBClassifier(n_estimators = 500, learning_rate = 0.1,verbosity = 1, rando
xgb_model.fit(X_train_fit,y_train_fit, eval_set = [(X_train_eval,y_train_eval)], early
```

```
[0]     validation_0-logloss:0.64479
[1]     validation_0-logloss:0.60569
[2]     validation_0-logloss:0.57481
[3]     validation_0-logloss:0.54947
[4]     validation_0-logloss:0.52973
[5]     validation_0-logloss:0.51331
[6]     validation_0-logloss:0.49823
[7]     validation_0-logloss:0.48855
[8]     validation_0-logloss:0.47888
[9]     validation_0-logloss:0.47068
[10]    validation_0-logloss:0.46507
[11]    validation_0-logloss:0.45832
[12]    validation_0-logloss:0.45557
[13]    validation_0-logloss:0.45030
[14]    validation_0-logloss:0.44653
[15]    validation_0-logloss:0.44213
[16]    validation_0-logloss:0.43948
[17]    validation_0-logloss:0.44088
[18]    validation_0-logloss:0.44358
[19]    validation_0-logloss:0.44493
[20]    validation_0-logloss:0.44294
[21]    validation_0-logloss:0.44486
[22]    validation_0-logloss:0.44586
[23]    validation_0-logloss:0.44680
[24]    validation_0-logloss:0.44925
[25]    validation_0-logloss:0.45383
```

Out[28]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=500,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=55,
              reg_alpha=0, reg_lambda=1, ...)
```

In [29]:
```
xgb_model.best_iteration
```

Out[29]:
```
16
```

The best round of training was round 16, with a log loss of 4.3948.

In [30]:
```
print(f"Metrics train:\n\tAccuracy score: {accuracy_score(xgb_model.predict(X_train),y
```

```
Metrics train:
        Accuracy score: 0.9251
Metrics test:
        Accuracy score: 0.8641
```

In this example, both Random Forest and XGBoost had similar performance (test accuracy).