Ensemble Methods

- ensemble estimate: $\bar{y} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k$
- squared error of the ensemble estimate: $(\bar{y} - y)^2$
- decomposition into an accuracy and diversity term: $(\bar{y} - y)^2 = \frac{1}{K} \sum_{k=1}^{K} (\hat{y}_k - y)^2 - \frac{1}{K} \sum_{k=1}^{K} (\hat{y}_k - \bar{y})^2$
- the ensemble will work well, if the members are diverse and individually accurate

Random Forest

Random forest (https://en.wikipedia.org/wiki/Random_forest) is a tree based algorithm for classification and regression. It can be viewed as a "parallel circuit" of decision trees. The output of the forest is the average output of its trees. Some methods to grow an accurate and diverse set of trees are as follows:

- Bootstrapping (https://en.wikipedia.org/wiki/Bootstrap_aggregating): Each tree is trained on a data set that was sampled from the original data with replacement.
- Feature bagging (https://en.wikipedia.org/wiki/Random_subspace_method): Each tree is trained on a random subset of the features.
- Extreme randomization (https://en.wikipedia.org/wiki/Random_forest#ExtraTrees): For each feature, a random split value is used instead of the optimal one.

Exercise 1: Implement a bootstrapping based random forest regressor and evaluate it on the Boston Housing data set using 3-fold cross-validation! The evaluation metric should be RMSE!

In [1]:

```
# Load the data to DataFrame.
import pandas as pd
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
         'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MEDV']
df = pd.read_csv('housing_data.txt', delim_whitespace=True, names=names)
df = df.sample(len(df), random_state=42) # shuffle the data
X = df[df.columns[:-1]].values           # input matrix
y = df['MEDV'].values                    # target vector
```

In [2]:

```
class SimpleRandomForestRegressor:
    def __init__(self, n_trees=100, max_depth=None):
        self.n_trees = n_trees
        self.max_depth = max_depth

    def fit(self, X, y):
        rs = np.random.RandomState(42)

        self.trees = []
        for k in range(self.n_trees):
            idxs = rs.randint(0, len(y), len(y))                                  # bootstrapping
            re = DecisionTreeRegressor(max_depth=self.max_depth, random_state=k) # create decision tree
            re.fit(X[idxs], y[idxs])                                              # train decision tree
            self.trees.append(re)

    def predict(self, X):
        yhat = np.zeros(len(X))
        for re in self.trees:
            yhat += re.predict(X)
        return yhat / len(self.trees)
```

In [3]:

```
# bootstrapping
import numpy as np
from sklearn.tree import DecisionTreeRegressor

rs = np.random.RandomState(42)

idxs = rs.randint(0, len(y), len(y))
```

In [4]:

```
len(idxs), len(set(idxs))
```

Out[4]:

```
(506, 302)
```

In [5]:

```
re = DecisionTreeRegressor()
re.fit(X[idxs], y[idxs])
```

Out[5]:

DecisionTreeRegressor

DecisionTreeRegressor()

In [6]:

```
# testing the fit in the class
re = SimpleRandomForestRegressor()
re.fit(X, y)
re.trees[1].tree_.feature
```

Out[6]:

```
array([11,  5,  7, -2,  5,  9, -2,  5,  9, 10,  6, -2, -2,  2,  0, -2,  0,
       10,  4, -2, -2, -2,  7, -2, -2, 11,  4, -2, -2,  9,  0,  1, -2, -2,
        9, 11, -2, -2, 11, -2, -2, -2, -2,  5,  2,  9, -2, 11,  0,  1,  6,
        4, -2, -2,  1,  0, -2, -2,  1, -2, -2, -2,  9, -2,  8, -2,  7, -2,
       -2, -2, 10,  5,  2, -2, -2, 11, -2, -2, 10,  6, -2,  1,  5, -2, -2,
       -2,  5,  5, 11, -2, -2, -2,  0,  4, -2,  0, -2, 10, -2, -2,  7, -2,
       -2, -2,  5, 11,  5,  2,  5, 11,  3,  5, -2, -2, -2,  6,  8, -2, -2,
        7, -2,  2, -2, -2, -2, 11, -2, -2,  2,  9, -2,  7,  2, -2, -2, -2,
       -2,  4, 10, 11,  0, 11, -2, -2, 11, -2, -2,  2, -2, -2,  8, -2, -2,
       10,  7, -2,  6,  7,  6, -2,  7, 11, -2, -2,  8, -2, -2,  3, -2, -2,
       10, -2, -2,  6, 10, 11, -2, -2,  9, -2, -2,  9,  6,  2, -2, -2, -2,
       -2,  9,  5,  1,  7, -2, -2,  0, 11, -2, -2,  9, -2, -2,  3, -2, -2,
       11, -2,  4, 11,  5,  7, -2, -2,  4, -2, -2, -2,  4, -2, -2,  0,  8,
        4, 10, -2, -2, -2,  0,  0,  5, -2,  4, 10, -2, -2,  0, -2, -2, 10,
       -2, -2,  5, -2, -2, -2,  4, 11,  5,  7, -2,  5,  5,  4,  6,  5, 11,
       -2,  5, -2, -2,  8,  7,  5, -2, -2, -2, -2, 11, -2, -2,  5,  0, -2,
       -2,  6, -2, -2,  7,  6,  4,  6,  9, -2, -2,  7,  5, 11, -2, -2, -2,
       -2, 11, 10,  0,  6, -2, -2, 11, -2, -2,  7, 11,  4, -2, -2, 10,  7,
       -2, -2, -2,  2, -2,  7,  4, -2,  4, -2, -2,  9, -2, -2,  5,  0, -2,
       -2,  3, -2, -2,  5,  7,  9,  8, -2, -2, -2,  6, 11,  0, -2,  5,  5,
        0, -2, -2, -2, -2, -2, -2, 11,  7, -2,  6, -2, -2, -2,  4,  5,  0,
       -2, -2, -2,  7, 11, -2, -2,  0,  0, -2, -2, -2,  0,  6, -2, -2,  5,
        9, -2, -2, -2,  4, -2,  8, -2, -2, 10,  0, 10, -2,  8, -2, -2,  7,
        2,  5, -2,  6, -2, -2,  0, 10, -2, -2,  0, -2,  5, 10, -2, -2,  4,
       -2, -2,  7,  6, -2,  6,  1, -2, -2, -2,  8, -2, -2,  5,  5,  9, 11,
       -2, -2,  7, -2, -2,  7,  4, -2,  0, -2, -2,  0,  0,  7, -2, -2,  0,
        5, -2, -2,  0, -2, -2, -2, 11, 10, 11, -2, -2,  1,  5,  0, -2, -2,
        0, -2, -2, -2, 11,  5,  8, -2, -2,  7,  8, -2, -2, 11, 11, -2, -2,
        6, -2, -2,  0, -2,  2, -2, 11, -2, -2,  0, 11,  6,  0,  3,  5, 11,
       -2,  7, -2, -2, -2,  6,  7, -2, -2, 11, -2, -2,  7, -2, -2,  5,  6,
       -2, -2, -2, 11,  6,  0,  5, -2, -2,  0,  7, 11,  0, -2, -2,  5, -2,
        0, -2, -2,  5, -2,  0, -2,  0, -2,  7, -2, -2, -2,  7, -2,  5, -2,
       -2,  3,  0, 11,  4, -2, -2,  7, -2, -2,  6, 11, -2, -2, -2, -2,  7,
        5, 11, -2, -2, -2,  7, 11,  0, 11, -2,  5, 11, -2,  4, -2, -2,  5,
        5, -2, -2,  7, -2, -2, -2,  5,  6, -2, 11, -2, -2,  5, -2, -2,  0,
        7, -2,  6, -2, -2,  0, -2,  0, -2, -2], dtype=int64)
```

In [7]:

```
# testing the predict in the class
re = SimpleRandomForestRegressor()
re.fit(X, y)
re.predict(X)
```

Out[7]:

```
array([23.37 , 31.963, 14.525, 22.815, 15.96 , 20.401, 18.19 , 14.204,
       20.182, 17.925, 21.025, 18.99 ,  7.346, 21.122, 18.789, 28.384,
       19.137,  9.301, 49.132, 14.137, 25.322, 27.413, 13.561, 22.907,
       14.574, 13.988, 20.518, 14.814, 20.857, 18.934, 22.116, 23.808,
       22.788, 20.974, 17.365, 18.08 , 34.564, 19.34 , 23.346, 23.554,
       19.532, 28.426, 49.217, 17.864, 22.625, 14.733, 13.7  , 24.118,
       19.29 , 25.845, 19.697, 34.86 , 15.083, 26.543, 44.135, 21.424,
       17.244, 30.019, 23.423, 19.232, 25.186, 34.898, 31.569, 19.894,
       25.418, 19.074, 13.455, 24.331, 29.993, 14.163, 20.001, 25.581,
       10.439, 20.714, 20.958,  5.637, 19.987, 48.466, 10.805,  9.544,
       20.842, 16.466, 20.496,  9.745, 19.684, 28.353, 16.1  , 24.501,
       24.773, 17.428, 22.672,  9.688, 19.953, 17.88 , 25.142, 21.852,
       45.143, 15.814, 10.894, 16.385, 21.752, 22.197, 12.542, 19.919,
       20.045, 13.853, 18.735, 24.512, 20.793, 23.827,  8.814, 25.076,
       21.881, 33.406, 31.697, 12.55 , 41.104, 14.557, 20.828, 23.845,
       17.9  , 24.626,  8.469, 19.621, 24.988, 20.796, 23.568, 37.252,
       16.833, 46.33 , 15.918, 22.772, 16.054, 18.693, 16.277, 18.464,
       20.569, 31.435, 29.06 , 15.759, 17.573, 23.281, 19.651, 19.497,
        8.255, 20.815, 17.251, 15.478, 14.497, 46.758, 15.086, 13.495,
       26.901, 21.008, 19.395, 22.375, 16.728, 24.387, 33.825,  6.504,
       28.464, 18.778, 21.203, 23.435, 22.392, 21.369, 47.878, 14.39 ,
       16.687, 22.608, 20.098, 19.653, 20.917, 37.16 , 21.366, 20.242,
       19.602, 28.566, 36.173, 24.411, 12.722, 12.676, 11.587, 16.024,
       32.5  , 27.346, 13.968, 13.904, 45.958, 21.289, 20.102, 22.808,
       18.133, 11.891,  6.616, 30.245, 26.262, 19.002, 15.775, 14.331,
       22.81 , 17.452, 27.199, 35.138, 23.123, 24.265, 23.921, 49.625,
       34.209, 32.007, 24.381, 22.848, 14.556, 42.113, 19.42 , 31.629,
       25.437, 21.3  , 21.359,  9.305, 45.073, 43.306, 32.123,  9.359,
       16.092, 20.106, 33.711, 17.124, 43.755, 20.622, 23.065, 12.06 ,
       20.052, 23.599, 29.625, 27.778, 22.181, 29.868, 25.477, 32.995,
       23.512, 20.974, 24.205,  9.85 , 25.366, 22.752, 16.23 ,  8.72 ,
       19.503, 24.6  , 23.038, 19.778, 22.61 , 14.191, 30.745, 26.623,
       33.633, 13.375, 16.173, 13.231, 14.201, 11.696, 24.497, 17.294,
        9.244, 21.578, 14.821, 24.696, 32.249, 40.376, 30.293, 20.031,
       31.319, 44.232, 23.491, 20.969, 23.075, 18.786, 17.872,  7.689,
       21.162, 20.603, 27.535, 28.401, 20.849, 22.688, 14.678, 12.854,
       35.815, 17.875, 15.839, 22.029, 22.145, 20.623, 21.741, 45.436,
        8.889, 19.031, 35.305, 17.728, 18.828, 22.467, 20.8  , 14.246,
       46.454, 24.106, 22.345, 17.289, 31.321, 36.012, 43.762, 18.68 ,
       22.116, 19.708, 44.771, 12.337, 19.434, 16.415, 21.996,  7.803,
       20.153, 12.878, 13.546, 17.921, 21.509, 21.343, 21.906, 23.752,
       49.106, 26.783, 21.93 , 46.983,  8.814, 24.278, 21.654, 19.491,
       18.807, 18.454, 14.023, 11.47 , 27.318, 20.328, 27.853, 20.878,
       24.262, 13.145, 11.256, 28.578, 27.972, 10.109, 20.427, 23.732,
       45.527, 22.008,  8.746, 35.24 , 43.258, 19.604, 26.442, 41.162,
       17.621, 26.798, 12.989, 20.137, 43.165, 28.153, 24.248, 17.114,
       17.734, 15.459, 48.602, 21.6  , 33.697, 15.43 , 14.917, 14.158,
       23.38 , 33.457, 22.689, 49.601, 10.239, 14.032, 22.798, 17.683,
       17.677, 27.336, 16.5  , 23.429, 18.457, 32.604, 25.881, 18.719,
       14.539, 34.426, 10.894, 20.368, 30.201, 22.577, 33.886, 12.439,
       23.313, 18.939, 25.81 , 21.493, 22.11 , 27.852, 20.084, 25.576,
       22.879, 21.38 , 32.946, 23.11 , 20.813, 23.445, 47.449, 24.703,
       26.593,  7.97 , 34.803, 20.401, 29.61 , 19.668, 22.367, 11.636,
       22.118, 14.709, 14.992, 19.346, 10.676, 27.922, 22.24 , 13.915,
       14.342, 31.713, 12.883, 21.146, 22.227, 19.544, 20.659, 25.398,
       18.339, 35.794, 20.416, 20.217, 17.419, 25.995,  7.908, 38.839,
       49.65 , 19.78 , 21.183, 15.662, 34.108, 18.829, 21.185, 23.817,
       18.932, 17.301, 19.952, 17.966, 22.152, 11.753, 34.663, 20.048,
       20.785, 30.106, 22.173, 23.272, 16.295, 31.048, 23.685, 30.988,
       19.278, 21.959, 18.99 , 26.139, 34.216, 14.931, 30.407,  7.669,
       23.722, 15.194, 25.105, 47.76 , 25.212, 15.091, 19.531, 19.25 ,
       21.914, 32.659, 45.435, 22.292, 15.254, 20.633, 21.909, 18.226,
       21.327, 15.245, 14.012, 29.515, 21.615, 18.884, 21.01 , 25.593,
       12.662, 19.319])
```

In [8]:

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

def evaluate(re, X, y):
    cv = KFold(3, random_state=42, shuffle=True)

    scores_tr = []
    scores_te = []
    for tr, te in cv.split(X):
        re.fit(X[tr], y[tr])
        yhat = re.predict(X)
        scores_tr.append(mean_squared_error(y[tr], yhat[tr])**0.5)
        scores_te.append(mean_squared_error(y[te], yhat[te])**0.5)

    return np.mean(scores_tr), np.mean(scores_te)
```

In [9]:

```
evaluate(SimpleRandomForestRegressor(), X, y)
```

Out[9]:

```
(1.2496535188109161, 3.5344184688013613)
```

In [10]:

```
evaluate(DecisionTreeRegressor(random_state=42), X, y)
```

Out[10]:

```
(0.0, 4.867211481330203)
```

In [11]:

```
from sklearn.linear_model import Ridge
evaluate(Ridge(), X, y)
```

Out[11]:

```
(4.710820669783009, 4.927289718141437)
```

In [12]:

```
# How the test RMSE depends on the number of trees.
res = []
for n_trees in range(1, 101):
    print(n_trees)
    re = SimpleRandomForestRegressor(n_trees=n_trees)
    _, rmse_te = evaluate(re, X, y)
    res.append({'n_trees': n_trees, 'rmse_te': rmse_te})
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

In [13]:

```
df_res = pd.DataFrame(res).set_index('n_trees')
df_res
```

Out[13]:

n_trees	rmse_te
1	4.732256
2	4.056692
3	3.960604
4	3.891775
5	3.900339
...	...
96	3.539245
97	3.538955
98	3.541483
99	3.542396
100	3.534418

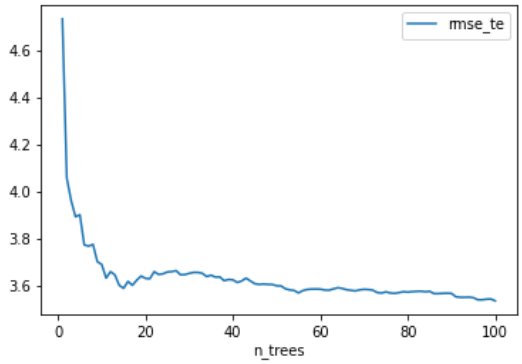
100 rows × 1 columns

In [14]:

```
df_res.plot()
```

Out[14]:

```
<AxesSubplot: xlabel='n_trees'>
```



Exercise 2: Implement a feature bagging based random forest regressor and evaluate it on the Boston Housing data set using 3-fold cross-validation!

In [15]:

```
class SimpleRandomForestRegressor:
    def __init__(self, n_trees=100, max_depth=None, bootstrap=True, feat_bag=False):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.bootstrap = bootstrap
        self.feat_bag = feat_bag


    def fit(self, X, y):
        rs = np.random.RandomState(42)
        n, d = X.shape

        self.trees = []
        for k in range(self.n_trees):
            row_idxs = np.arange(n)
            if self.bootstrap:
                row_idxs = rs.randint(0, n, n)  # bootstrapping

            col_idxs = np.arange(d)
            if self.feat_bag:
                col_idxs = rs.randint(0, d, d)  # featur bagging


            tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=k) # create decision tree
            tree.fit(X[row_idxs][:, col_idxs], y[row_idxs])                                    # train decision tree
            self.trees.append((tree, col_idxs))

    def predict(self, X):
        yhat = np.zeros(len(X))
        for tree, col_idxs in self.trees:
            yhat += tree.predict(X[:, col_idxs])
        return yhat / len(self.trees)
```

In [16]:

```
evaluate(SimpleRandomForestRegressor(bootstrap=False, feat_bag=False), X, y)
```

Out[16]:

```
(3.221869811461758e-14, 4.563463998771837)
```

In [17]:

```
evaluate(SimpleRandomForestRegressor(bootstrap=True, feat_bag=False), X, y)
```

Out[17]:

```
(1.2496535188109161, 3.5344184688013613)
```

In [18]:

```
evaluate(SimpleRandomForestRegressor(bootstrap=False, feat_bag=True), X, y)
```

Out[18]:

```
(0.018271831222768983, 3.746377378818943)
```

In [19]:

```
evaluate(SimpleRandomForestRegressor(bootstrap=True, feat_bag=True), X, y)
```

Out[19]:

```
(1.3106116530969623, 3.65460581683147)
```

Exercise 3: Repeat the previous experiments using scikit-learn's RandomForestRegressor class!

In [20]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [21]:

```
# bootstrapping No feature bagging
evaluate(RandomForestRegressor(bootstrap=True, max_features=1.0, random_state=42), X, y)
```

Out[21]:

```
(1.249721947092092, 3.578281219483339)
```

In [22]:

```
# No bootstrapping feature bagging with 50% strategy
evaluate(RandomForestRegressor(bootstrap=False, max_features=0.5, random_state=42), X, y)
```

Out[22]:

```
(0.0008610326361854645, 3.432009311114937)
```

In [23]:

```
# bootstrapping feature bagging with 50% strategy
evaluate(RandomForestRegressor(bootstrap=True, max_features=0.5, random_state=42), X, y)
```

Out[23]:

```
(1.2368681562001127, 3.464869293320867)
```

In [24]:

```
# No bootstrapping No feature bagging
evaluate(RandomForestRegressor(bootstrap=False, max_features=1.0, random_state=42), X, y)
```

Out[24]:

```
(3.221869811461758e-14, 4.534326812091633)
```

Exercise 4: Plot the training and the test RMSE as a function of the number of trees!

In [25]:

```
# How the test RMSE depends on the number of trees.
# No bootstrapping with feature bagging
res = []
for n_trees in range(1, 101):
    print(n_trees, end=' ')
    re = RandomForestRegressor(n_estimators=n_trees, bootstrap=False, max_features=0.5, random_state=42)
    rmse_tr, rmse_te = evaluate(re, X, y)
    res.append({'n_trees': n_trees, 'rmse_tr': rmse_tr, 'rmse_te': rmse_te})
```

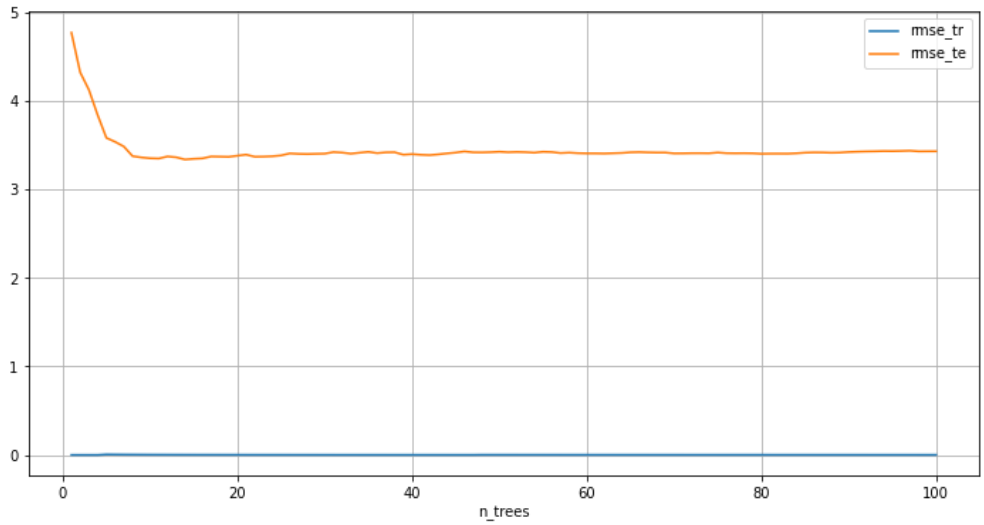
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

In [27]:

```
df_res = pd.DataFrame(res).set_index('n_trees')
df_res.plot(figsize=(12,6), grid=True)
```

Out[27]:

```
<AxesSubplot: xlabel='n_trees'>
```



In [28]:

```
# How the test RMSE depends on the number of trees.
# bootstrapping No feature bagging
res = []
for n_trees in range(1, 101):
    print(n_trees, end=' ')
    re = RandomForestRegressor(n_estimators=n_trees, bootstrap=True, max_features=1.0, random_state=42)
    rmse_tr, rmse_te = evaluate(re, X, y)
    res.append({'n_trees': n_trees, 'rmse_tr': rmse_tr, 'rmse_te': rmse_te})
```

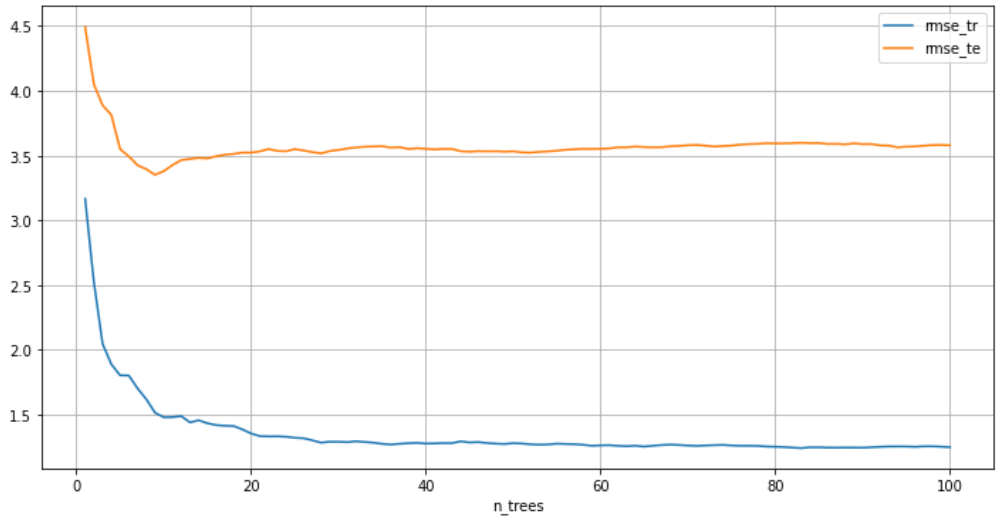
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

In [29]:

```
df_res = pd.DataFrame(res).set_index('n_trees')
df_res.plot(figsize=(12,6), grid=True)
```

Out[29]:

```
<AxesSubplot: xlabel='n_trees'>
```



Exercise 5: Repeat the previous experiment using the "extreme randomization" strategy!

In [30]:

```
from sklearn.ensemble import ExtraTreesRegressor
```

In [31]:

```
evaluate(ExtraTreesRegressor(random_state=42), X, y)
```

Out[31]:

```
(3.221869811461758e-14, 3.5508313733547276)
```

In [32]:

```
res = []
for n_trees in range(1, 101):
    print(n_trees, end=' ')
    re = ExtraTreesRegressor(n_estimators=n_trees, random_state=42)
    rmse_tr, rmse_te = evaluate(re, X, y)
    res.append({'n_trees': n_trees, 'rmse_tr': rmse_tr, 'rmse_te': rmse_te})
```

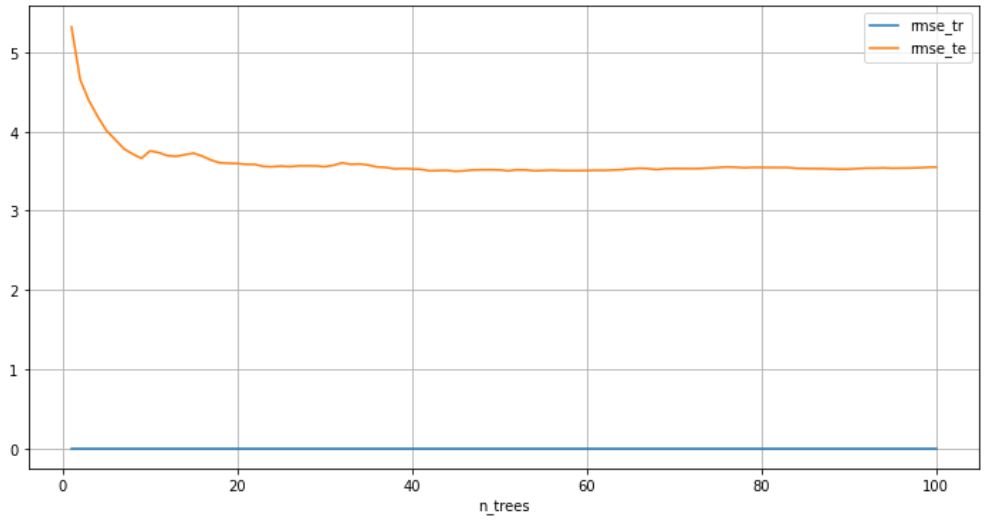
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

In [33]:

```
df_res = pd.DataFrame(res).set_index('n_trees')
df_res.plot(figsize=(12,6), grid=True)
```

Out[33]:

```
<AxesSubplot: xlabel='n_trees'>
```



In [ ]: