Gradient Boosting

- [Gradient boosting (https://en.wikipedia.org/wiki/Gradient_boosting)](https://en.wikipedia.org/wiki/Gradient_boosting) is another ensemble technique for classification and regression. It can be viewed as a "series circuit" of base learners.
- The idea of gradient boosting originates from [Leo Breiman (https://en.wikipedia.org/wiki/Leo_Breiman)](https://en.wikipedia.org/wiki/Leo_Breiman) and [Jerome Friedman (https://en.wikipedia.org/wiki/Jerome_H._Friedman)](https://en.wikipedia.org/wiki/Jerome_H._Friedman) (1999).
- The diversity of the base learners is achieved by training them on different targets.
- The base learners are regressors, both for classification and regression.
- Usually, the base learners are decision trees regressors, but in theory they could be any regression algorithm.
- Gradient Boosted Decision Trees (or Gradient Boosting Machine) is a "swiss army knife" method in machine learning. It is invariant to the scale of the feature values and performs well on a wide variety of problems.

Pseudo Code of Training (w/o Learning Rate)

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:
$$F_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma).$$
2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:
    $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$
    2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:
    $$\gamma_m = \arg\min_\gamma \sum_{i=1}^n L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$
    4. Update the model:
    $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$
3. Output $F_M(x)$.

Learning Rate

- instead of step size γ_m, we use $\eta \cdot \gamma_m$, where $\eta \in (0, 1]$
- $\eta < 1$ implements the "slow cooking" idea, and in practice leads to better ensembles than $\eta = 1$

Special Case: Gradient Boosting for Regression

- the loss function is the squared loss: $L(y, F(x)) = \frac{1}{2}(y - F(x))^2$
- the initial model is the average target: $F_0(x) = \frac{1}{n}\sum_{i=1}^n y_i$
- pseudo-residuals: $r_{im} = y_i - F_{m-1}(x_i)$
- optimal multiplier: $\gamma_m = \left[\sum_{i=1}^n h_m(x_i) r_{im}\right] / \left[\sum_{i=1}^n (h_m(x_i))^2\right]$

Exercise 1: Implement a tree based gradient boosting regressor and evaluate it on the Boston Housing data set using 3-fold cross-validation! Use a maximal tree depth of 3!

In [1]:

```
# Load the Boston Housing data set.
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) # data shuffling
X = df.values[:, :-1] # input matrix
y = df['MEDV'].values # target vector
```

In [2]:

```
X.shape
```

Out[2]:

```
(506, 12)
```

In [3]:

```
y.shape
```

Out[3]:

```
(506,)
```

In [12]:

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
```

In [16]:

```
class SimpleGradiantBoostingRegressor:
    def __init__(self, n_trees=100, eta=0.1, max_depth=3):
        self.n_trees = n_trees
        self.eta = eta
        self.max_depth= max_depth

    def fit(self, X, y):
        self.F0 = np.mean(y)  # best constant model
        r = y - self.F0 # pseudo-residuals

        self.trees = []
        for m in range(self.n_trees):
            tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=m) #random State
            tree.fit(X, r) # fit base learner
            rhat = tree.predict(X) # prediction of the base learner
            gamma = (rhat@r)/(rhat@rhat) # optimal step size
            w = self.eta * gamma
            self.trees.append((w, tree)) # save tree and w
            r -= (w * rhat)


    def predict(self, X):
        yhat = np.ones(len(X)) * self.F0
        for w, tree in self.trees:
            yhat += w * tree.predict(X)
        return yhat
```

In [17]:

```
re = SimpleGradiantBoostingRegressor()
re.fit(X, y)
```

In [18]:

```
re.predict(X)
```

Out[18]:

```
array([23.94338117, 31.84403964, 14.5974109 , 23.18869335, 16.53389674,
       20.51149196, 17.62557483, 13.57396085, 20.02974902, 19.1537357 ,
       21.00365235, 18.77811641,  7.77255095, 20.58009835, 19.25299176,
       26.15302545, 19.35018065,  9.13319693, 48.57348633, 15.22903246,
       24.7196627 , 26.65527172, 13.86831478, 21.97822747, 14.95097855,
       15.33661623, 21.58127832, 14.67157346, 19.68720584, 20.47668502,
       20.2642894 , 23.61148237, 16.48740369, 19.79431243, 16.01197271,
       17.44559748, 33.9641267 , 19.12315572, 22.28414065, 23.87685238,
       18.91158592, 29.47223172, 49.05900514, 18.92279053, 23.09613799,
       14.45276541, 14.89674794, 23.87685238, 17.95920624, 25.79465137,
       20.31190662, 35.96893202, 15.80590052, 25.46493393, 46.0662084 ,
       21.29196354, 16.20583887, 31.3541363 , 22.92447383, 17.90951381,
       24.1399609 , 35.04744678, 31.05704042, 19.29881036, 24.63048326,
       18.61714486, 13.63618977, 23.80222937, 29.11514983, 16.50801582,
       20.80001181, 23.94133523, 10.30781219, 20.94226247, 21.637198  ,
        5.56506069, 20.06917987, 48.58622726, 11.04404007,  9.62619819,
       21.17633358, 14.94284172, 20.06246491,  9.8898435 , 19.8148173 ,
       27.03040893, 15.54321935, 23.72991859, 25.59834601, 18.05814359,
       22.60743354,  8.70476568, 20.01023594, 18.65530393, 25.77200581,
       20.28259633, 48.78200179, 15.84392654, 12.33805394, 18.26469352,
       [illegible], [illegible], 12.33541787, 20.08118332, 20.05643001,
       [illegible], [illegible], [illegible], 20.59959848, 23.87750559,
        9.08553407, 22.82305562, 22.06299019, 31.91028132, 32.09155153,
       [illegible], 41.7049744 , 15.1208986 , 20.70884696, 23.78328633,
       [illegible], 23.51199935,  7.80699263, 20.41511813, 24.14561736,
       [illegible], [illegible], 38.49091195, 17.73296402, 46.04448157,
       [illegible], [illegible], 18.27060821, 19.07947045, 15.53622527,
       [illegible], [illegible], 31.24760232, 29.15255873, 15.54772281,
       [illegible], [illegible], [illegible], [illegible], [illegible],
       [illegible], [illegible], [illegible], [illegible], [illegible],
       16.53389674, 14.4823683 , 27.23235591, 21.71325974, 20.85076335,
       [illegible], [illegible], [illegible], [illegible],  6.20276483,
       27.71506165, 18.55719647, 21.16234316, 23.61148237, 22.05445276,
       20.43440803, 47.3127483 , 15.23359837, 16.77341824, 21.0866933 ,
       19.22433077, 20.06917987, 22.24159265, 38.76863876, 20.49393345,
       19.58143706, 21.41995845, 29.25856698, 36.14065647, 23.8729553 ,
       13.41636513, 13.27697366, 11.16487116, 16.35655085, 31.49254756,
       29.05132918, 14.56919768, 13.91854614, 49.19874073, 20.92634571,
       19.88657069, 22.42079331, 18.12221124, 12.69631187,  7.80943809,
       30.06691216, 25.94846649, 18.1178712 , 14.99069729, 14.48726275,
       23.35541423, 17.97229239, 27.34120677, 34.72100189, 23.14173045,
       22.34497229, 23.28724383, 50.01282765, 33.72454409, 32.05293992,
       24.57660515, 23.35020137, 16.53389674, 42.39924604, 20.9675943 ,
       31.0271021 , 24.12559993, 20.66605879, 20.66619643,  9.65114062,
       45.07299996, 44.0964027 , 33.3598017 ,  9.40412597, 17.09786167,
       21.41090285, 34.40489748, 16.61728077, 48.95011956, 19.98685239,
       23.40403421, 11.47069706, 20.52753475, 23.50022496, 30.4450431 ,
       30.54829822, 21.44329524, 23.5380056 , 24.39954545, 31.96615913,
       25.08710939, 20.34544749, 23.77452114, 10.71668747, 24.82650903,
       22.31115735, 17.94976399,  9.01746252, 20.18053097, 24.55787872,
       23.20356282, 19.00346334, 21.2753031 , 14.75526853, 30.73128232,
       [illegible], [illegible], [illegible], [illegible], 13.05931079,
       14.01233715, 11.02133698, 24.55468784, 17.57157583,  9.48477268,
       22.36969719, 14.3574069 , 25.36942505, 32.63422488, 39.75081863,
       30.77718454, 19.54826771, 30.90963836, 43.67350441, 21.9271521 ,
       [illegible], [illegible], 18.99691351, 17.19603056,  8.03224294,
       21.13641042, 19.73475349, 27.79295037, 26.29989839, 18.85214119,
       23.53743781, 15.67118714, 14.28893323, 34.6581908 , 18.48875961,
       11.31388105, 21.33862702, 21.50323762, 19.08380711, 21.61941626,
       48.70010917,  8.61620531, 18.75844577, 37.22269773, 19.29909642,
       18.59065693, 22.30554283, 21.00658132, 15.20315154, 45.53844009,
       23.92627597, 21.54766349, 16.06193831, 30.06106969, 34.87128768,
       42.69923995, 18.85726669, 22.34281364, 19.5190534 , 45.8386138 ,
       13.54354803, 20.62475724, 15.47410901, 22.20126708,  8.32811969,
       20.06917987, 14.22484007, 14.13622895, 17.91132848, 22.29388057,
       22.36394993, 21.48990219, 24.70958183, 49.36144447, 26.10592653,
       20.9826723 , 49.13247245,  8.9076255 , 24.68975108, 20.27977161,
       21.15649136, 18.46549262, 20.03173972, 13.98552463, 11.02133698,
       26.57170469, 20.3989323 , 26.98142753, 21.89954354, 25.11305702,
       12.75018369, 11.32198808, 29.56999801, 28.3799249 ,  9.51018397,
       18.62400296, 22.97355312, 45.37493621, 22.31822592,  9.61681826,
       35.3152476 , 43.04664917, 15.85069933, 24.54167677, 40.72144701,
       17.19505671, 25.87042247, 13.99221051, 21.2753031 , 43.59262221,
       27.6567932 , 24.13619933, 17.27562194, 17.81686259, 13.80311657,
       48.05056526, 21.00298048, 33.26131745, 15.97643774, 15.08151668,
       14.42406501, 22.28279528, 34.51824077, 22.84958464, 49.7005193 ,
       10.49528037, 13.64087617, 22.38400339, 17.02984294, 17.16915836,
       27.35022716, 17.02455525, 24.06629699, 18.62807604, 32.4954863 ,
       26.42841251, 18.73785301, 16.65772224, 34.24958377, 11.38441512,
       20.61401392, 28.71303461, 22.56862867, 35.12773675, 11.45456386,
       23.67979453, 19.38493389, 25.62545238, 20.32871843, 21.65657997,
       27.55675713, 19.36385403, 25.77763182, 21.30842263, 20.98560186,
       33.17912815, 22.49580438, 21.87915238, 22.69237596, 49.17654587,
       24.14561736, 26.13959404,  8.82022703, 32.97673581, 20.49867839,
       32.08527032, 17.45463707, 21.72623844, 11.32198808, 21.79019007,
       15.87753279, 15.02449898, 18.62516946, 10.22441633, 27.40572498,
```

In [13]:

```
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 [19]:

```
evaluate(SimpleGradiantBoostingRegressor(), X, y)
```

Out[19]:

```
(1.2716614748909418, 3.363000134587033)
```

Exercise 2: Repeat the previous experiment using scikit-learn!

In [15]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

In [20]:

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

Out[20]:

```
(1.2716614748909418, 3.36820868174328)
```

Exercise 3: Which tree depth gives the most accurate ensemble?

```
       23.76632069, 14.71893486, 15.76422952, 31.48445399, 11.90378456,
       19.28711466, 22.10144168, 19.50792664, 19.77773943, 25.19864246,
       18.82732995, 34.87128768, 20.51338874, 20.23355623, 19.32917714,
       27.06011499,  8.46146737, 38.21135689, 49.88352537, 19.53494766,
       21.02285957, 16.40964185, 33.84292501, 18.46314173, 21.60872548,
       24.76702986, 17.62759944, 17.71570929, 20.48291761, 18.39583915,
       22.5462147 , 10.79330954, 34.03961182, 19.11057791, 19.97173206,
       28.7263535 , 22.42079331, 22.79617935, 16.57900309, 31.38486286,
       22.89353353, 31.87851094, 18.08245003, 22.21974  , 18.01088659,
       26.19137898, 33.77820738, 13.78436769, 29.45196712,  8.57804525,
       24.04767225, 14.729936  , 25.7726188 , 50.16718936, 25.84797917,
       14.56548017, 20.01023594, 19.41916554, 20.63633342, 33.22054316,
       48.31400185, 23.17413704, 15.41411439, 22.01096261, 22.77029625,
       17.12884637, 21.23025849, 14.62203331, 13.83904853, 30.12374063,
       20.68799646, 17.45463707, 21.02343742, 24.98958987, 12.46448683,
       20.45289135])
```

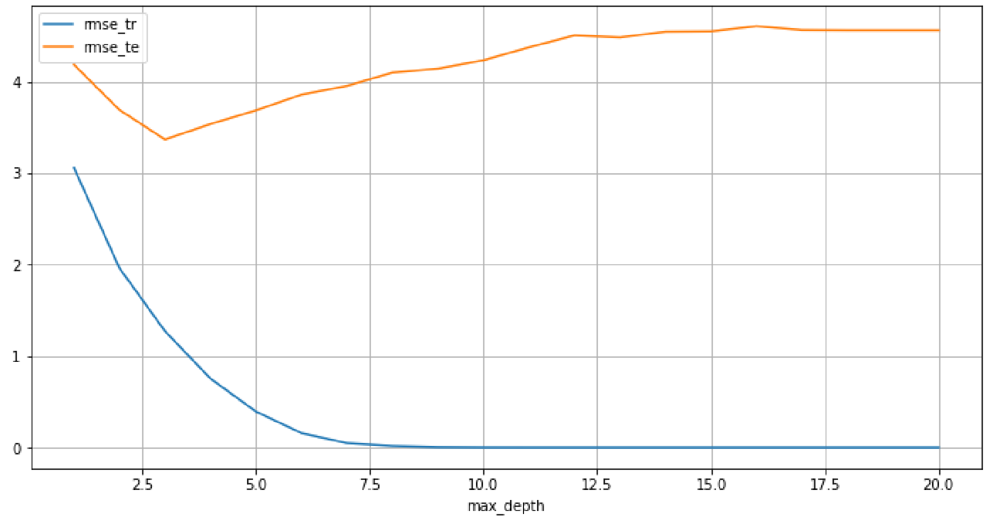
In [29]:

```
# Professor solution
res = []
for max_depth in range(1, 21):
    print(max_depth, end=' ')
    re = GradientBoostingRegressor(max_depth=max_depth, random_state=42)
    rmse_tr, rmse_te = evaluate(re, X, y)
    res.append({'max_depth': max_depth, 'rmse_tr': rmse_tr, 'rmse_te': rmse_te})

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

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Out[29]:

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



In [30]:

```
# Optimal max_depth.
df_res['rmse_te'].idxmin()
```

Out[30]:

```
3
```

Exercise 3/B: How the training and test RMSE changes with the number of trees? (Use a simple train-test split for this experiment!)

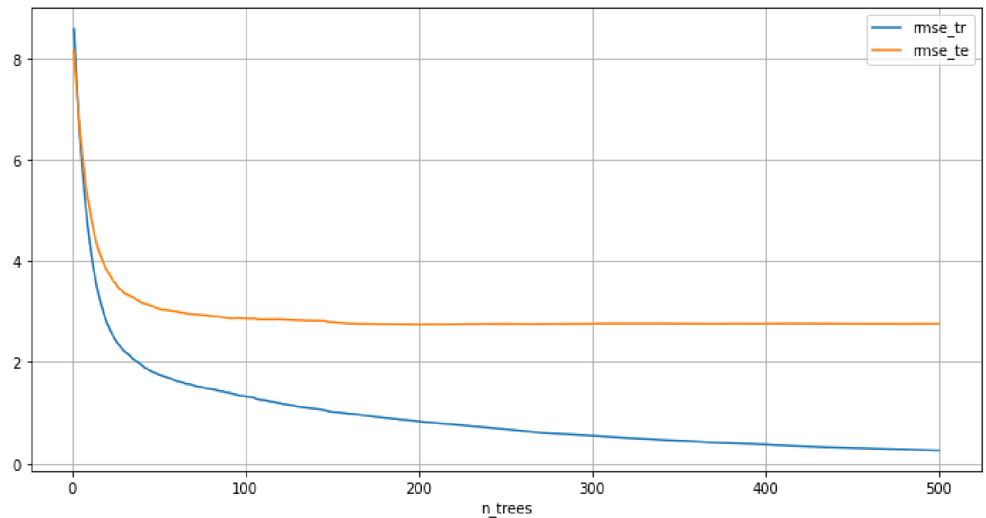
In [32]:

```
from sklearn.model_selection import ShuffleSplit
tr, te = next(ShuffleSplit(test_size=0.3, random_state=42).split(X))

res = []
re = GradientBoostingRegressor(n_estimators=1, random_state=42)
for n_trees in range(1, 501):
    print(n_trees, end=' ')
    if n_trees > 1:
        re.warm_start = True
        re.n_estimators += 1
    re.fit(X[tr], y[tr])
    yhat = re.predict(X)
    rmse_tr = mean_squared_error(y[tr], yhat[tr])**0.5
    rmse_te = mean_squared_error(y[te], yhat[te])**0.5
    res.append({'n_trees': n_trees, 'rmse_tr': rmse_tr, 'rmse_te': rmse_te})

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

```
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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 11
4 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 1
43 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229
230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316
317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345
346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432
433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461
462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490
491 492 493 494 495 496 497 498 499 500
```

Out[32]:

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



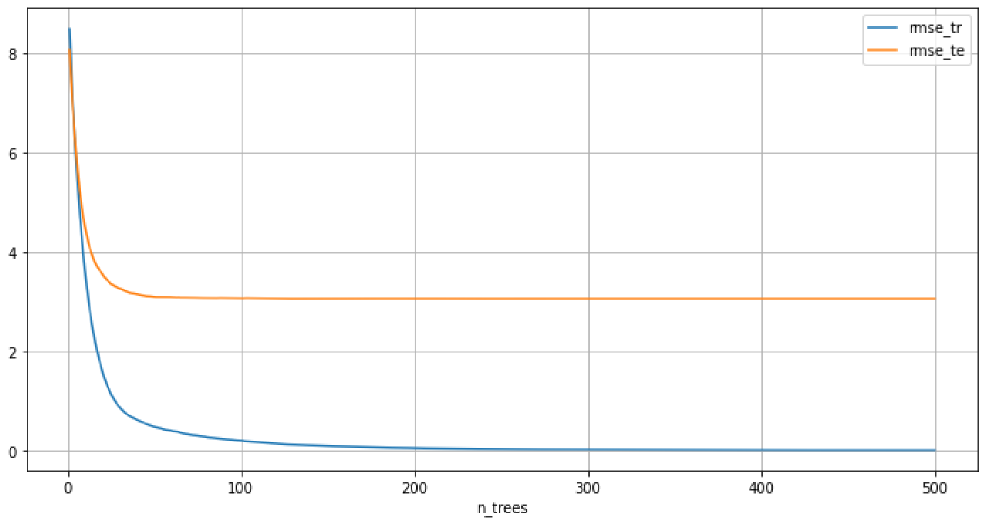
In [33]:

```
# What happens if we use deeper trees?
from sklearn.model_selection import ShuffleSplit
tr, te = next(ShuffleSplit(test_size=0.3, random_state=42).split(X))

res = []
re = GradientBoostingRegressor(n_estimators=1, max_depth=6, random_state=42)
for n_trees in range(1, 501):
    print(n_trees, end=' ')
    if n_trees > 1:
        re.warm_start = True
        re.n_estimators += 1
    re.fit(X[tr], y[tr])
    yhat = re.predict(X)
    rmse_tr = mean_squared_error(y[tr], yhat[tr])**0.5
    rmse_te = mean_squared_error(y[te], yhat[te])**0.5
    res.append({'n_trees': n_trees, 'rmse_tr': rmse_tr, 'rmse_te': rmse_te})

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

```
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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 11
4 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 1
43 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229
230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316
317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345
346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432
433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461
462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490
491 492 493 494 495 496 497 498 499 500
```

Out[33]:

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



Exercise 4: Apply a random forest and a gradient boosting classifier on the Wisconsin Breast Cancer data set! Use stratified 10-fold cross-validation! The evaluation metric should be the ratio of correct classifications. For both ensemble methods, determine the maximal tree depth that gives the highest accuracy!

In [1]:

```
# Load the Wisconsin Breast Cancer data set.
import pandas as pd
names = [
    'Sample_code_number', 'Clump_Thickness', 'Uniformity_of_Cell_Size',
    'Uniformity_of_Cell_Shape', 'Marginal_Adhesion', 'Single_Epithelial_Cell_Size',
    'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class'
]
df = pd.read_csv('wisconsin_data.txt', sep=',', names=names, na_values='?')
df = df.sample(len(df), random_state=42) # data shuffling
df['Bare_Nuclei'].fillna(0, inplace=True)
X = df[df.columns[1: -1]].values
y = (df['Class'].values / 2 - 1).astype('int')
```

In [2]:

```
X.shape, y.shape
```

Out[2]:

```
((699, 9), (699,))
```

In [11]:

```
# evaluate function
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
import numpy as np
```

In [12]:

```
def evaluate(cl, X, y):
    cv = StratifiedKFold(10, shuffle=True, random_state=42)
    scores = []
    for tr, te in cv.split(X, y):
        cl.fit(X[tr], y[tr])
        yhat = cl.predict(X) # log_loss or regressors use predict_propa
        score = accuracy_score(y[te], yhat[te])
        scores.append(score)
    return np.mean(scores)
```

In [13]:

```
# Dummy classifier's accuracy.
from sklearn.dummy import DummyClassifier
```

In [14]:

```
evaluate(DummyClassifier(), X, y)
```

Out[14]:

```
0.6552173913043479
```

In [15]:

```
# Logistic regression's accuracy.
from sklearn.linear_model import LogisticRegression
```

In [16]:

```
evaluate(LogisticRegression(), X, y)
```

Out[16]:

```
0.9656314699792962
```

In [17]:

```
# gradient boosting, random forest, different max_depth values, ...
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [29]:

```
GB = []
RF = []

for i in range (1,11):
    print(i)
    gb = evaluate(GradientBoostingClassifier(random_state=42 ,max_depth=i), X, y)
    GB.append({'max_depth': i, 'eva_GB': gb})
    rf = evaluate(RandomForestClassifier(random_state=42, max_depth=i), X, y)
    RF.append({'max_depth': i, 'eva_RF': rf})
```

```
1
2
3
4
5
6
7
8
9
10
```

In [31]:

```
data1 = pd.DataFrame(GB).set_index('max_depth')
data2 = pd.DataFrame(RF).set_index('max_depth')
```

In [32]:

```
data1
```

Out[32]:

	eva
max_depth	
1	0.962774
2	0.957081
3	0.952754
4	0.958509
5	0.958509
6	0.948489
7	0.949917
8	0.945611
9	0.941346
10	0.937060

In [33]:

```
data2
```

Out[33]:

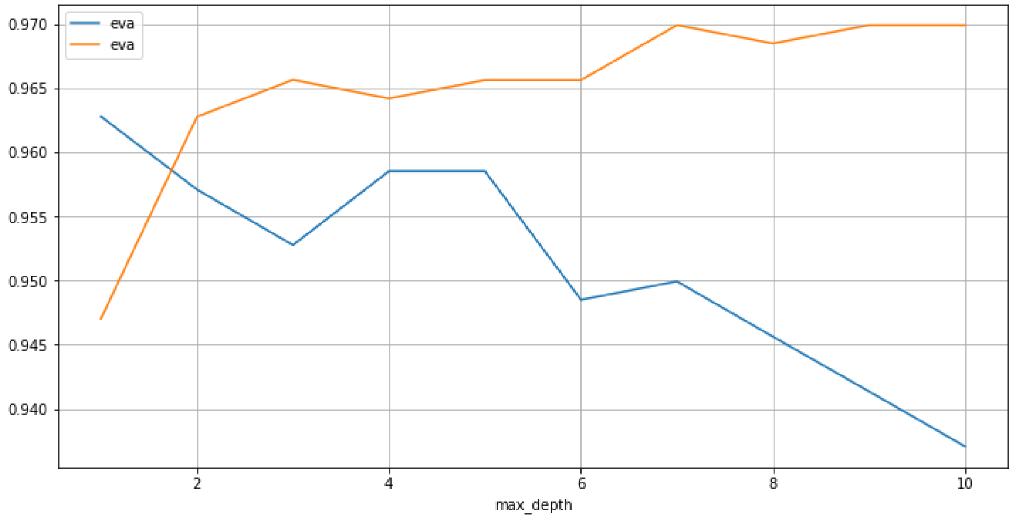
	eva
max_depth	
1	0.947019
2	0.962754
3	0.965631
4	0.964182
5	0.965611
6	0.965611
7	0.969896
8	0.968468
9	0.969896
10	0.969896

In [41]:

```
ax = data1.plot(figsize=(12,6))
data2.plot(ax=ax, grid=True)
```

Out[41]:

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



In [42]:

```
# Professor Solutions
# gradient boosting, random forest, different max_depth values, ...
res = []
for max_depth in list(range(1, 11)):
    print(max_depth, end=' ')
    gb = GradientBoostingClassifier(max_depth=max_depth, random_state=42)
    rf = RandomForestClassifier(max_depth=max_depth, random_state=42)
    res.append({'max_depth': max_depth, 'RF_acc': evaluate(rf, X, y), 'GB_acc': evaluate(gb, X, y)})
```

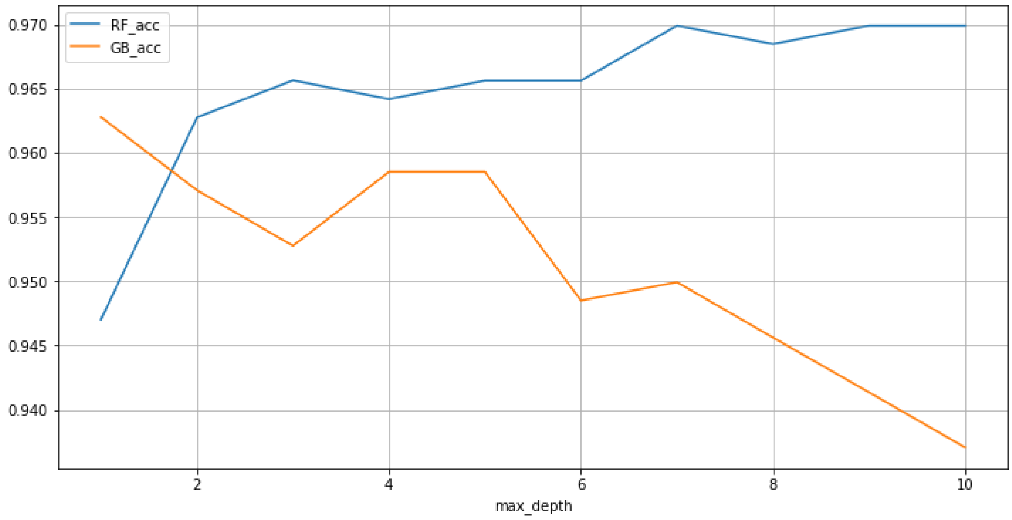
```
1 2 3 4 5 6 7 8 9 10
```

In [43]:

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

Out[43]:

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



Gradient Boosting on Steroids

- [XGBoost (https://en.wikipedia.org/wiki/XGBoost)](https://en.wikipedia.org/wiki/XGBoost) and [LightGBM (https://en.wikipedia.org/wiki/LightGBM)](https://en.wikipedia.org/wiki/LightGBM) are a highly efficient and flexible implementations of gradient boosting.
- XGBoost started as a research project by Tianqi Chen (in 2014).
- LightGBM was introduced by Microsoft Research (in 2016).
- note: another [LightGBM (https://lightgbm.readthedocs.io/en/stable/Features.html)](https://lightgbm.readthedocs.io/en/stable/Features.html)

Exercise 5: Compare XGBoost, LightGBM and scikit-learn's GradientBoostingClassifier on the Wisconsin Breast Cancer problem, in terms of speed and accuracy!

In [44]:

```
import xgboost
xgboost.__version__
```

Out[44]:

```
'1.7.3'
```

In [45]:

```
import lightgbm
lightgbm.__version__
```

Out[45]:

```
'3.3.5'
```

In [56]:

```
from xgboost import XGBClassifier
evaluate(XGBClassifier(n_estimators=100, max_depth=3), X, y)
```

Out[56]:

```
0.9585093167701864
```

In [55]:

```
XGBClassifier()
```

Out[55]:

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
```

In [66]:

```
import time

t0 = time.time()
print('acc ', evaluate(XGBClassifier(n_estimators=100, max_depth=3), X, y), '\ntime ', (time.time() - t0))
```

```
acc  0.9585093167701864
time  0.43979501724243164
```

In [67]:

```
from lightgbm import LGBMClassifier
evaluate(LGBMClassifier(), X, y)
```

Out[67]:

```
0.9542443064182194
```

In [68]:

```
import time

t0 = time.time()
print('acc ', evaluate(LGBMClassifier(n_estimators=100, num_leaves=8), X, y), '\ntime ', (time.time() - t0))
```

```
acc  0.9585300207039337
time  0.28715991973876953
```

In [72]:

```
import time

t0 = time.time()
print('acc ', evaluate(GradientBoostingClassifier(n_estimators=100, max_depth=3, random_state=42), X, y), '\ntime ', (time.time()
```

```
acc  0.9585093167701864
time  1.2569808959960938
```

In [77]:

```
# Since version 0.21, scikit-learn includes a histogram based
# gradient boosting algorithm that was inspired by LightGBM.

from sklearn.ensemble import HistGradientBoostingClassifier
import time
from sklearn.ensemble import HistGradientBoostingClassifier
t0= time.time()
print('acc:', evaluate(HistGradientBoostingClassifier(max_iter=100, max_leaf_nodes=8, random_state=42), X, y), '\ntime:', time.ti
```

```
acc: 0.9585093167701864
time: 1.2880544662475586
```

In [ ]: