```
========================================
Both hour.csv and day.csv have the following fields, except hr which is not
available in day.csv

    - instant: record index
    - dteday : date
    - season : season (1:springer, 2:summer, 3:fall, 4:winter)
    - yr : year (0: 2011, 1:2012)
    - mnth : month ( 1 to 12)
    - hr : hour (0 to 23)
    - holiday : weather day is holiday or not (extracted from
http://dchr.dc.gov/page/holiday-schedule)
    - weekday : day of the week
    - workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
    + weathersit :
        - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
        - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
        - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain
+ Scattered clouds
        - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
    - temp : Normalized temperature in Celsius. The values are divided to 41 (max)
    - atemp: Normalized feeling temperature in Celsius. The values are divided to
50 (max)
    - hum: Normalized humidity. The values are divided to 100 (max)
    - windspeed: Normalized wind speed. The values are divided to 67 (max)
    - casual: count of casual users
    - registered: count of registered users
    - cnt: count of total rental bikes including both casual and registered
```

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
# 导入数据
df_train = pd.read_csv("hour.csv")
df_train.head()
```

Out[2]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	at
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2

In [3]:

```
df_train.isnull().sum()
```

Out[3]:

```
instant       0
dteday        0
season        0
yr            0
mnth          0
hr            0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

In [4]:

```
# 数据集更新了，年月日已经拆分了
# df_train["year"] = pd.DatatimeIndex(df_train["datatime"]).year
```

In [5]:

```
fig, axs = plt.subplots(2,2, sharey=True)
df_train.groupby("weathersit").plot(y="cnt", marker="o", ax=axs[0,0])
df_train.groupby("hum").mean().plot(y="cnt", marker="o",figsize=(24,16), ax=axs[0,1
df_train.groupby("temp").mean().plot(y="cnt", marker="o", ax=axs[1,0])
df_train.groupby("windspeed").plot(y="cnt", marker="o", ax=axs[1,1])
plt.show
```

Out[5]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```

In [6]:

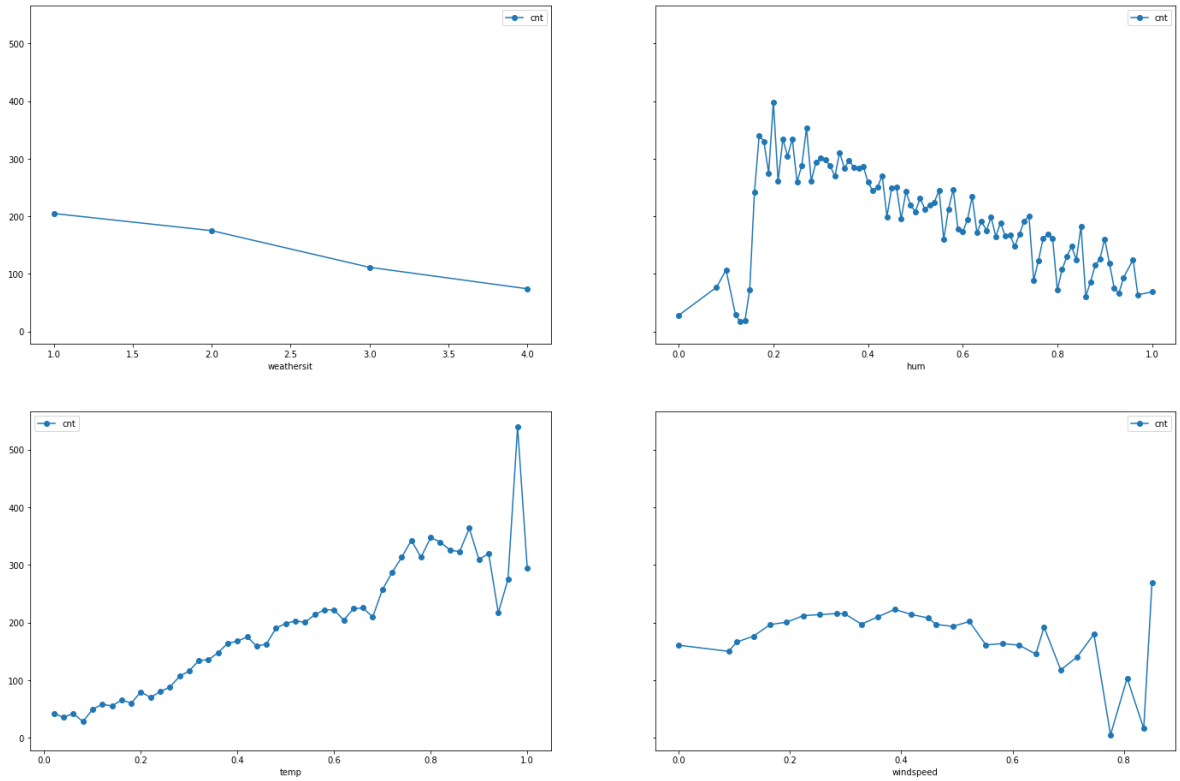
```
# 从上面数据看到有的特征不求均值找不到规律
```

In [7]:

```
fig, axs = plt.subplots(2,2, sharey=True)
df_train.groupby("weathersit").mean().plot(y="cnt", marker="o", ax=axs[0,0])
df_train.groupby("hum").mean().plot(y="cnt", marker="o",figsize=(24,16), ax=axs[0,1
df_train.groupby("temp").mean().plot(y="cnt", marker="o", ax=axs[1,0])
df_train.groupby("windspeed").mean().plot(y="cnt", marker="o", ax=axs[1,1])
plt.show
```

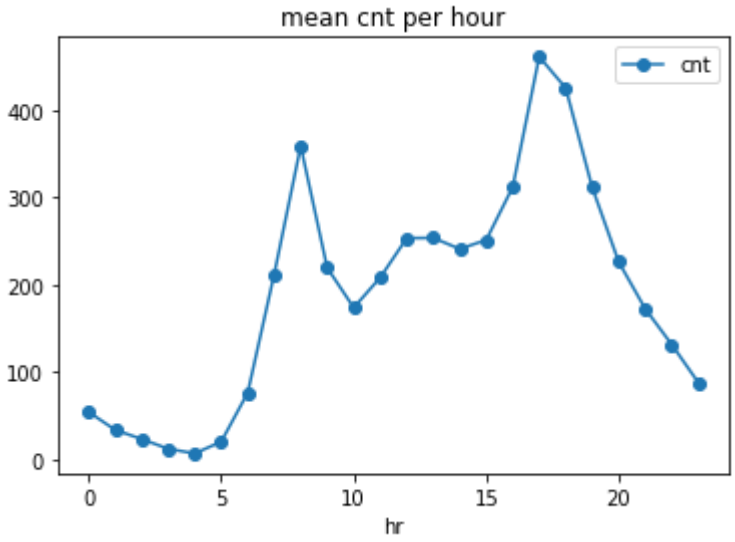
Out[7]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



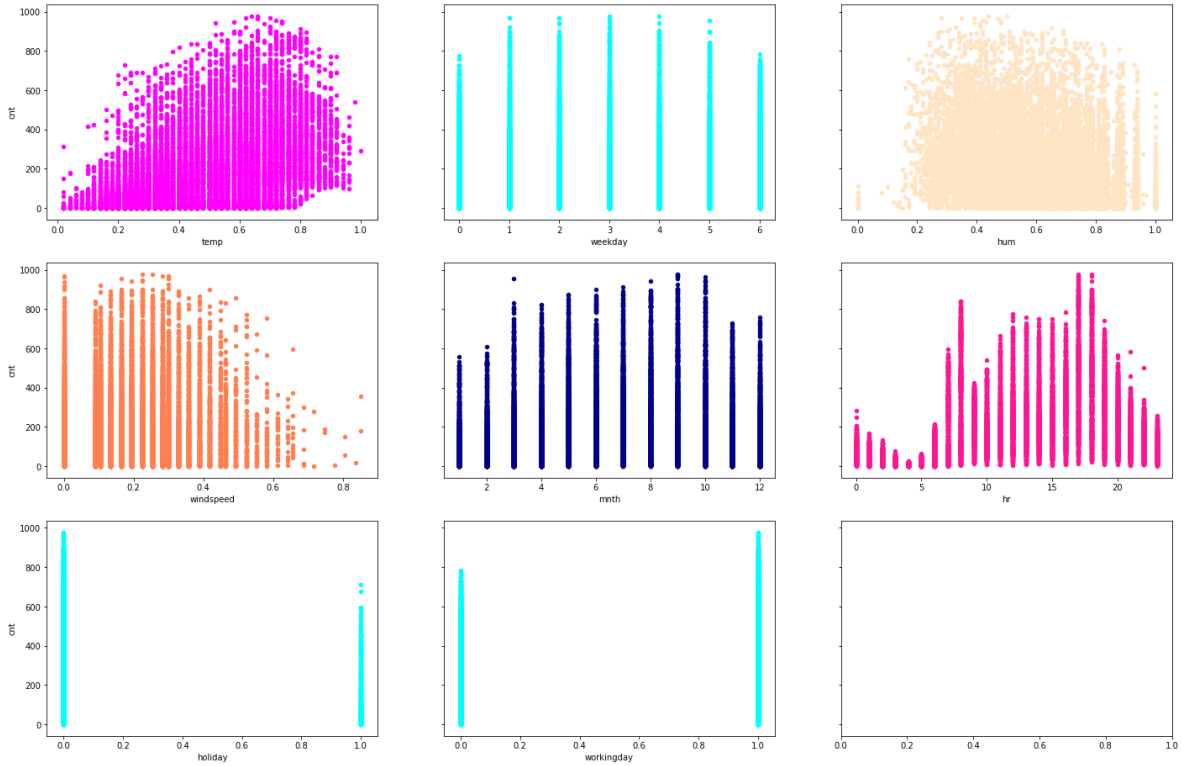
In [8]:

```
df_train.groupby("hr").mean().plot(y="cnt", marker="o")
plt.title("mean cnt per hour")
plt.show()
```

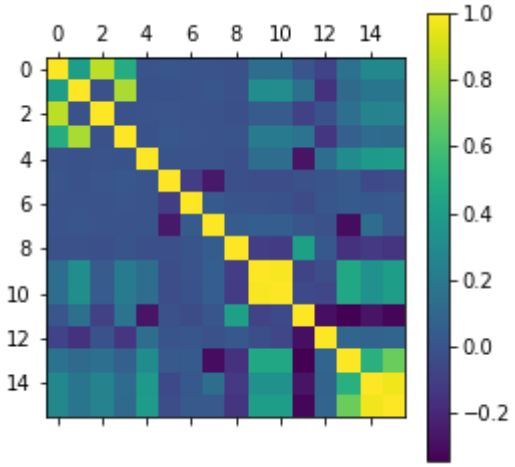


In [9]:

```
fig,axs = plt.subplots(3,3, sharey= True)
df_train.plot(x="temp", y="cnt", kind="scatter",figsize=(24,16),ax=axs[0,0],color="
df_train.plot(x="weekday", y="cnt", kind="scatter",ax=axs[0,1],color="cyan")
df_train.plot(x="hum", y="cnt", kind="scatter",ax=axs[0,2],color="bisque")
df_train.plot(x="windspeed", y="cnt", kind="scatter",ax=axs[1,0],color="coral")
df_train.plot(x="mnth", y="cnt", kind="scatter",ax=axs[1,1],color="darkblue")
df_train.plot(x="hr", y="cnt", kind="scatter",ax=axs[1,2],color="deeppink")
df_train.plot(x="holiday", y="cnt", kind="scatter",ax=axs[2,0],color="cyan")
df_train.plot(x="workingday", y="cnt", kind="scatter",ax=axs[2,1],color="cyan")


column = df_train.columns
corr = df_train[column].corr()
plt.figure()
# 特征间的相关矩阵
plt.matshow(corr)
plt.colorbar()
plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```



In [10]:

```
df = df_train.drop(["dteday"], axis=1)
```

In [11]:

```
df.head(3)
```

Out[11]:

	instant	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hu
0	1	1	0	1	0	0	6	0	1	0.24	0.2879	0.8
1	2	1	0	1	1	0	6	0	1	0.22	0.2727	0.8
2	3	1	0	1	2	0	6	0	1	0.22	0.2727	0.8

In [12]:

```
name = df.drop(['cnt','casual','registered'],axis=1).columns

target = df["cnt"].values
feature = df.drop(['cnt','casual','registered'],axis=1).values

from sklearn import preprocessing
# 对特征进行归一化等

feture = preprocessing.scale(feature)
```

In [13]:

```
print("name:", name)
print(feature)
print("target:", target)
```

```
name: Index(['instant', 'season', 'yr', 'mnth', 'hr', 'holiday', 'week
day',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspee
d'],
      dtype='object')
[[1.0000e+00 1.0000e+00 0.0000e+00 ... 2.8790e-01 8.1000e-01 0.0000e+0
0]
 [2.0000e+00 1.0000e+00 0.0000e+00 ... 2.7270e-01 8.0000e-01 0.0000e+0
0]
 [3.0000e+00 1.0000e+00 0.0000e+00 ... 2.7270e-01 8.0000e-01 0.0000e+0
0]
 ...
 [1.7377e+04 1.0000e+00 1.0000e+00 ... 2.5760e-01 6.0000e-01 1.6420e-0
1]
 [1.7378e+04 1.0000e+00 1.0000e+00 ... 2.7270e-01 5.6000e-01 1.3430e-0
1]
 [1.7379e+04 1.0000e+00 1.0000e+00 ... 2.7270e-01 6.5000e-01 1.3430e-0
1]]
target: [16 40 32 ... 90 61 49]
```

In [14]:

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve
```

In [15]:

```
# 高斯朴素贝叶斯估计器
from sklearn.naive_bayes import GaussianNB

cv = ShuffleSplit(n_splits=10, test_size=0.25, random_state=0)
estimator = GaussianNB()
X = feature; y= target
train_sizes, train_scores, test_scores = learning_curve( \
        estimator, X, y, cv=cv, n_jobs=4)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
```

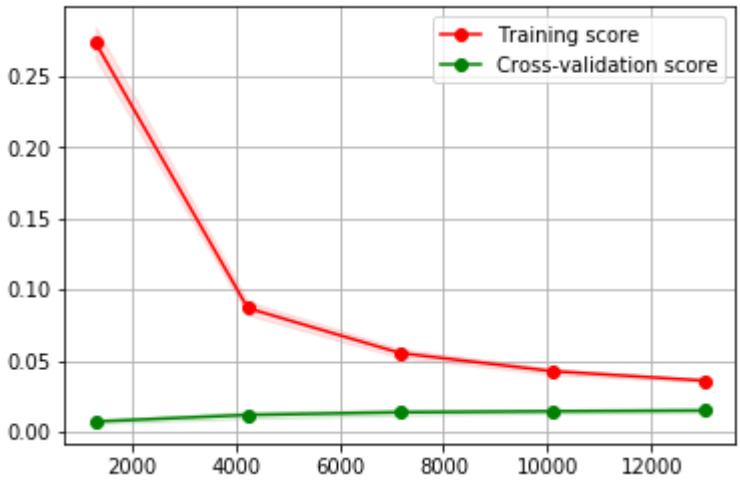
In [16]:

```
print(train_scores_mean)
print(train_scores_std)

print(test_scores_mean)
print(test_scores_std)
print(train_sizes)
```

```
[0.27336915 0.0868508  0.05528739 0.04253044 0.03589842]
[0.01112578 0.00475373 0.00310955 0.00220448 0.00113147]
[0.00720368 0.0118527  0.0136939  0.01440736 0.01493671]
[0.00143008 0.00249211 0.00216206 0.00155483 0.00156586]
[ 1303  4236  7168 10101 13034]
```

In [17]:

```
plt.grid()
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Cross-validation score")

plt.legend(loc="best")
plt.show()
# 高斯贝叶斯朴素估计模型
```



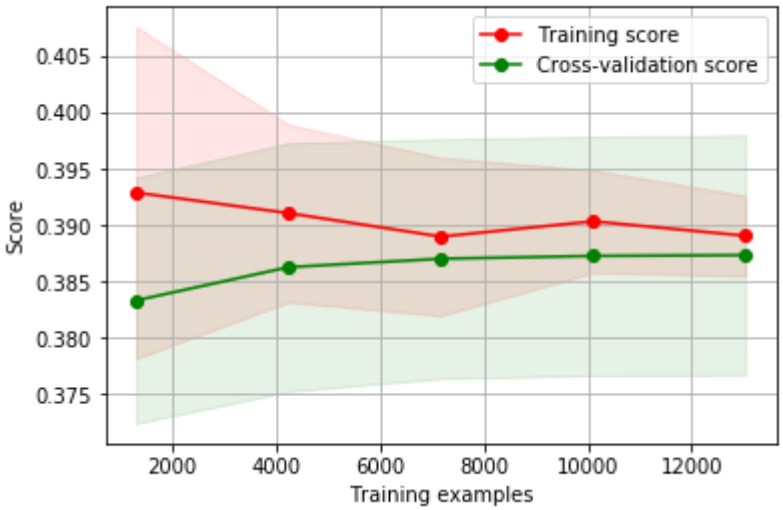
In [53]:

```
def estm(estimator, X, y, n_jobs=1):
    cv = ShuffleSplit(n_splits=10, test_size=0.25, random_state=0)

    train_sizes, train_scores, test_scores = learning_curve( \
        estimator, X, y, shuffle=True, cv=cv, n_jobs=n_jobs)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    print("train_mean:", train_scores_mean)
    print("test_mean:", test_scores_mean)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    plt.legend(loc="best")
    plt.show()
```

In [54]:

```
# 岭回归估计器
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1, fit_intercept=True)

estm(estimator = ridge, X = feature, y= target) # 岭回归
```

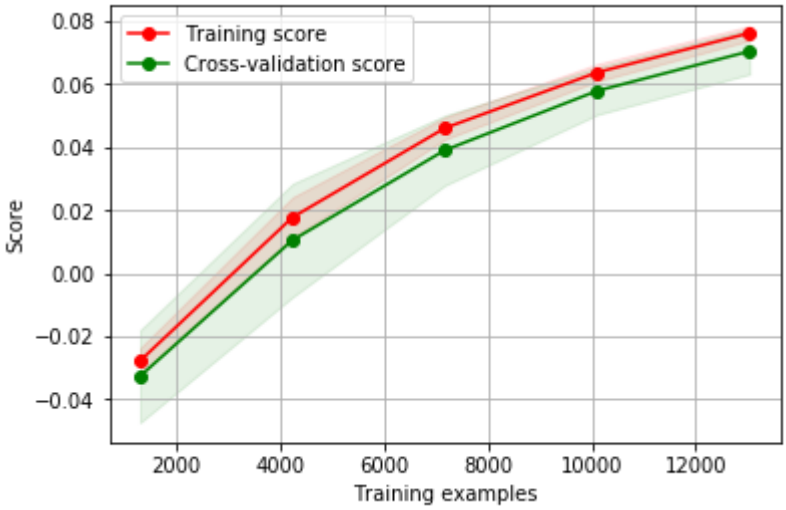
```
train_mean: [0.39290346 0.3910804  0.38899554 0.39035204 0.38907019]
test_mean: [0.38332278 0.38627641 0.38702503 0.38727722 0.38735053]
```



In [55]:

```
# svm估计器
from sklearn.svm import SVR
svm_est = SVR(gamma="scale")
estm(estimator = svm_est, X = feature, y= target)
```

```
train_mean: [-0.02780901  0.01771899  0.04605179  0.06373237  0.076250
03]
test_mean: [-0.03268469  0.01047544  0.03902897  0.05788598  0.0704484
5]
```



In [62]:

```
#跑上面的svm模型真的很慢
```

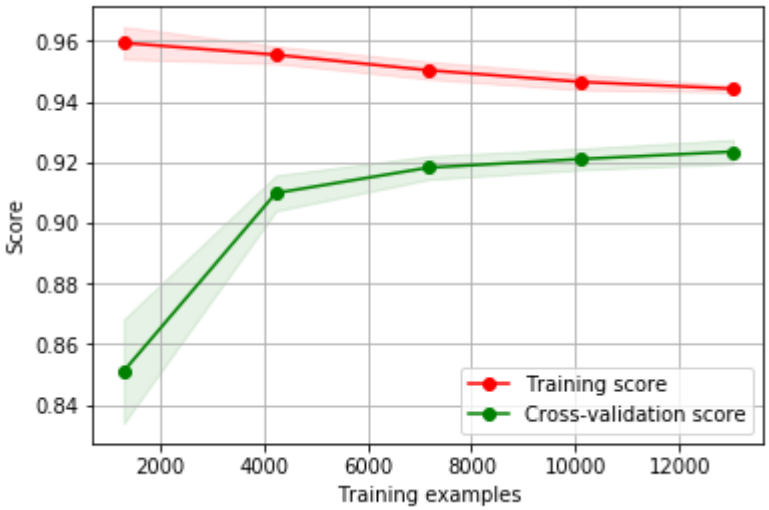
In [63]:

```
# 随机森林估计器
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators=100)
```

In [66]:

```
rfr_best = RandomForestRegressor(n_estimators=100, max_depth=10)
estm(rfr_best, X = feature, y= target)
```

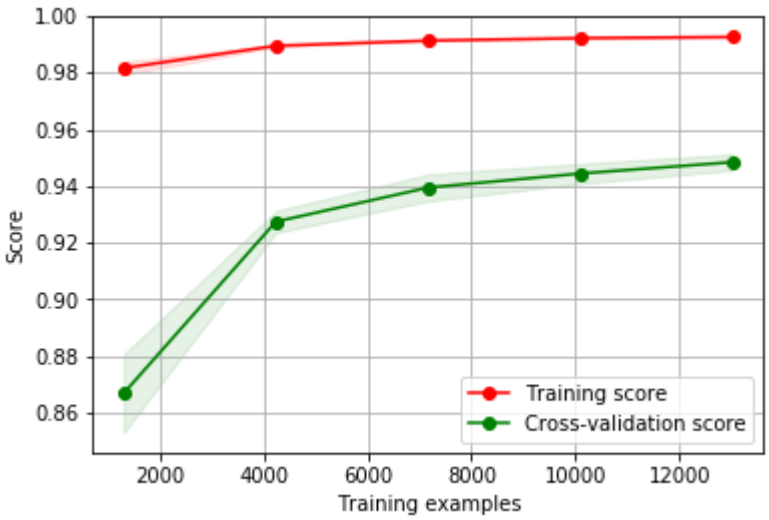
```
train_mean: [0.95935031 0.95542485 0.95032115 0.94648625 0.94422042]
test_mean: [0.85103462 0.90982391 0.91819348 0.92103033 0.92347536]
```



In [67]:

```
estm(rfr, X = feature, y= target)
```

```
train_mean: [0.98167846 0.98956945 0.99139242 0.99227577 0.99268494]
test_mean: [0.86680905 0.92746839 0.93951654 0.94448014 0.9484968 ]
```



In [68]:

```
df2 = df_train.drop(['holiday', 'weekday','workingday',
                     'dteday','casual','registered'],axis=1)
# 删掉三个可能不相关的特征 :holiday, weekday, workingday
```

In [69]:

```
df2.head(5)
```

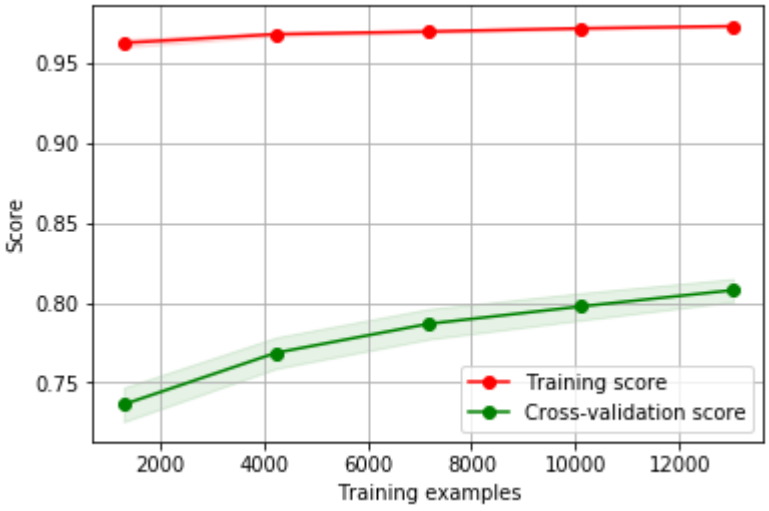
Out[69]:

	instant	season	yr	mnth	hr	weathersit	temp	atemp	hum	windspeed	cnt
0	1	1	0	1	0	1	0.24	0.2879	0.81	0.0	16
1	2	1	0	1	1	1	0.22	0.2727	0.80	0.0	40
2	3	1	0	1	2	1	0.22	0.2727	0.80	0.0	32
3	4	1	0	1	3	1	0.24	0.2879	0.75	0.0	13
4	5	1	0	1	4	1	0.24	0.2879	0.75	0.0	1

In [75]:

```
# 减少三个特征后的随机森林效果
target2 = df2["cnt"].values
feature2 = df2.drop(['cnt'],axis=1).values
estm(rfr, X = feature2, y= target2)
```

```
train_mean: [0.96254768 0.96800351 0.96966206 0.97165311 0.97301559]
test_mean: [0.73635969 0.76869966 0.78680719 0.79770389 0.80788745]
```



In [76]:

```
# 效果变差了，说明并不是冗余的特征
```

In [77]:

```
# 网格搜索法来寻找最优参数
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

X_train,X_test,y_train,y_test = train_test_split(feature, target, test_size=0.2,\
                                                 shuffle=True, random_state=None)

#构建网格参数
param_grid = {
    'n_estimators': [10, 100, 500],
    'max_depth': list(range(2,11))
            }

#初始化模型
forest = RandomForestRegressor()
#初始化网格搜索
grid_search = GridSearchCV(estimator=forest, param_grid=param_grid, cv=5,
                           n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

#查看最好的参数选择
print(grid_search.best_params_)

#使用网格搜索得到的最好的参数选择进行模型训练
best_forest = grid_search.best_estimator_
best_forest.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done  50 tasks      | elapsed:   49.6s
[Parallel(n_jobs=-1)]: Done 135 out of 135 | elapsed:  4.4min finished

{'max_depth': 10, 'n_estimators': 500}
```

Out[77]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=Non
e,
           oob_score=False, random_state=None, verbose=0, warm_start=F
alse)
```

In [78]:

```
# for params, mean_score, scores in grid_search.grid_
print(grid_search.cv_results_)
```

```
{'mean_fit_time': array([ 0.13319407,  1.45894413,  7.69540601,  0.215
43045,  2.12760754,
       10.19142137,  0.25360661,  2.50598874, 12.50833755,  0.3293362
1,
        3.05163474, 15.49317541,  0.40856681,  3.53402071, 17.5118323
8,
        0.4156352 ,  3.97993717, 19.81730919,  0.41597514,  4.4758298
4,
       22.97348342,  0.52254672,  5.08494382, 25.21887112,  0.5697337
2,
        5.89285698, 24.89119835]), 'std_fit_time': array([0.00597661,
0.06011933, 0.15387387, 0.06057044, 0.18664663,
       0.35262105, 0.05621877, 0.0846163 , 0.12028449, 0.03099818,
       0.16301284, 0.3957759 , 0.05232937, 0.18943767, 0.23094902,
       0.05952818, 0.08112081, 0.35501047, 0.02914394, 0.25369788,
       0.60628481, 0.07118387, 0.21263516, 0.33981729, 0.08466858,
       0.14657052, 4.06957062]), 'mean_score_time': array([0.00348597,
0.02038913, 0.11069765, 0.00612483, 0.03590779,
       0.14690261, 0.00495086, 0.03047194, 0.13486309, 0.00470958,
       0.02950215, 0.1779366 , 0.00487576, 0.03642316, 0.21307054,
       0.00582633, 0.04934931, 0.18337116, 0.00567908, 0.05458946,
       0.25079675, 0.00578904, 0.0800024 , 0.26320786, 0.01405897,
       0.05956931, 0.2289319 ]), 'std_score_time': array([0.00015086,
0.0009698 , 0.02113325, 0.00316835, 0.02203837,
       0.03937412, 0.00127093, 0.0046392 , 0.00709288, 0.00045844,
       0.00120708, 0.04808674, 0.00038644, 0.00786648, 0.0458313 ,
       0.00170539, 0.01187062, 0.01696695, 0.0008307 , 0.02729879,
       0.06539335, 0.00010715, 0.04716663, 0.05540412, 0.00835431,
       0.0166167 , 0.03914065]), 'param_max_depth': masked_array(data=
[2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7,
                   8, 8, 8, 9, 9, 9, 10, 10, 10],
             mask=[False, False, False, False, False, False, False, Fa
lse,
                   False, False, False, False, False, False, False, Fa
lse,
                   False, False, False, False, False, False, False, Fa
lse,
                   False, False, False],
       fill_value='?',
            dtype=object), 'param_n_estimators': masked_array(data=[1
0, 100, 500, 10, 100, 500, 10, 100, 500, 10, 100, 500,
                   10, 100, 500, 10, 100, 500, 10, 100, 500, 10, 100,
500,
                   10, 100, 500],
             mask=[False, False, False, False, False, False, False, Fa
lse,
                   False, False, False, False, False, False, False, Fa
lse,
                   False, False, False, False, False, False, False, Fa
lse,
                   False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'max_depth': 2, 'n_estimators':
10}, {'max_depth': 2, 'n_estimators': 100}, {'max_depth': 2, 'n_estima
tors': 500}, {'max_depth': 3, 'n_estimators': 10}, {'max_depth': 3, 'n
_estimators': 100}, {'max_depth': 3, 'n_estimators': 500}, {'max_dept
h': 4, 'n_estimators': 10}, {'max_depth': 4, 'n_estimators': 100}, {'m
```

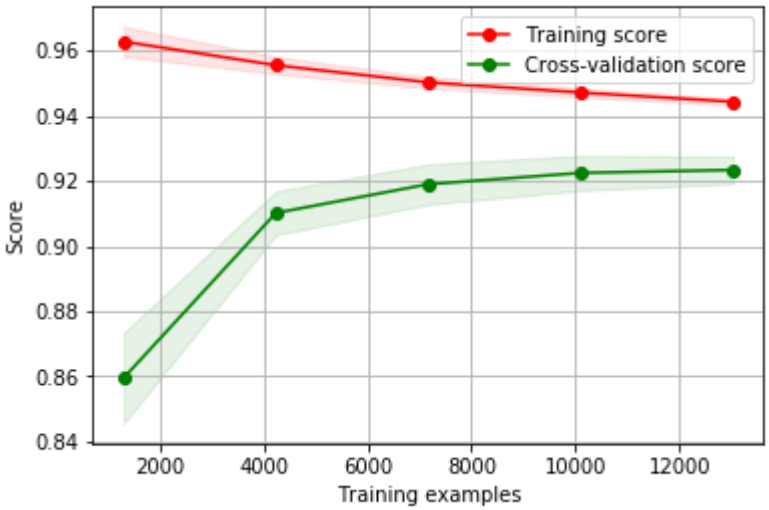
```
h': 4, 'n_estimators': 10}, {'max_depth': 4, 'n_estimators': 100}, {'m
ax_depth': 4, 'n_estimators': 500}, {'max_depth': 5, 'n_estimators': 1
0}, {'max_depth': 5, 'n_estimators': 100}, {'max_depth': 5, 'n_estimat
ors': 500}, {'max_depth': 6, 'n_estimators': 10}, {'max_depth': 6, 'n_
estimators': 100}, {'max_depth': 6, 'n_estimators': 500}, {'max_dept
h': 7, 'n_estimators': 10}, {'max_depth': 7, 'n_estimators': 100}, {'m
ax_depth': 7, 'n_estimators': 500}, {'max_depth': 8, 'n_estimators': 1
0}, {'max_depth': 8, 'n_estimators': 100}, {'max_depth': 8, 'n_estimat
ors': 500}, {'max_depth': 9, 'n_estimators': 10}, {'max_depth': 9, 'n_
estimators': 100}, {'max_depth': 9, 'n_estimators': 500}, {'max_dept
h': 10, 'n_estimators': 10}, {'max_depth': 10, 'n_estimators': 100},
{'max_depth': 10, 'n_estimators': 500}], 'split0_test_score': array
([0.4265695 , 0.43080089, 0.43333664, 0.52482063, 0.52868847,
       0.52751703, 0.60295818, 0.60333284, 0.60722655, 0.6880848 ,
       0.68472585, 0.68671101, 0.74101723, 0.73300685, 0.73615953,
       0.83156864, 0.8328769 , 0.83403787, 0.88214921, 0.88438457,
       0.88499887, 0.90604236, 0.9127264 , 0.9126119 , 0.92452809,
       0.92869364, 0.92876554]), 'split1_test_score': array([0.4356870
6, 0.44799795, 0.44385583, 0.52410611, 0.52709677,
       0.52811793, 0.5953118 , 0.60109326, 0.60361272, 0.66934986,
       0.66205907, 0.66365064, 0.69897809, 0.70714238, 0.70854428,
       0.79903639, 0.82000209, 0.82164397, 0.8671429 , 0.87276111,
       0.87276547, 0.89823378, 0.90009964, 0.90172258, 0.91248584,
       0.91660188, 0.91748047]), 'split2_test_score': array([0.4332172
6, 0.42151468, 0.42187655, 0.51774011, 0.51208051,
       0.51265898, 0.57885442, 0.58301569, 0.58369676, 0.65188192,
       0.65272815, 0.6526802 , 0.70556499, 0.70892637, 0.71218678,
       0.80873436, 0.81543464, 0.81552178, 0.86679453, 0.86753016,
       0.87116847, 0.90064449, 0.90381184, 0.90346152, 0.9161736 ,
       0.91912169, 0.92073253]), 'split3_test_score': array([0.4492938
2, 0.43668538, 0.43876207, 0.53115374, 0.53166857,
       0.53366559, 0.58874033, 0.59201368, 0.59359413, 0.66397063,
       0.66445918, 0.66281597, 0.71152393, 0.71379972, 0.71702243,
       0.81702849, 0.81725307, 0.8207766 , 0.87849206, 0.88226444,
       0.88122158, 0.89888095, 0.91028826, 0.91049059, 0.92404634,
       0.9293586 , 0.92856867]), 'split4_test_score': array([0.4093267
7, 0.40906575, 0.40999022, 0.51882518, 0.51824527,
       0.51787675, 0.56934065, 0.56913182, 0.57026932, 0.63981228,
       0.6448662 , 0.64454813, 0.69009841, 0.69197979, 0.6922864 ,
       0.80578953, 0.80934544, 0.80937264, 0.85942446, 0.86248597,
       0.86060782, 0.89261474, 0.89356756, 0.89303781, 0.90865972,
       0.91347295, 0.91428701]), 'mean_test_score': array([0.4308191 ,
0.42921384, 0.42956501, 0.52332891, 0.52355572,
       0.523967  , 0.58704223, 0.58971877, 0.5916813 , 0.66262144,
       0.66176871, 0.6620824 , 0.70943777, 0.71097219, 0.71324112,
       0.81243163, 0.81898325, 0.82027132, 0.8708009 , 0.87388547,
       0.87415291, 0.89928377, 0.90409905, 0.90426524, 0.91717884,
       0.92144976, 0.92196692]), 'std_test_score': array([0.01304282,
0.01323909, 0.01220871, 0.00480755, 0.00727445,
       0.00760069, 0.01187401, 0.01255947, 0.01348972, 0.01631609,
       0.01342985, 0.01418063, 0.01732113, 0.01321422, 0.01415489,
       0.01117714, 0.00777887, 0.0081609 , 0.00832797, 0.0083907 ,
       0.0085073 , 0.00431899, 0.00692399, 0.00695073, 0.00627327,
       0.00644297, 0.00583823]), 'rank_test_score': array([25, 27, 26,
24, 23, 22, 21, 20, 19, 16, 18, 17, 15, 14, 13, 12, 11,
       10,  9,  8,  7,  6,  5,  4,  3,  2,  1], dtype=int32), 'split0_
train_score': array([0.42601463, 0.42946586, 0.43192614, 0.52110995,
0.52461488,
       0.5237484 , 0.58845221, 0.58731621, 0.59194356, 0.6618245 ,
       0.65969967, 0.66122801, 0.72539732, 0.71564364, 0.71902706,
       0.82535444, 0.82712128, 0.82870432, 0.88415985, 0.88549359,
       0.88635991, 0.91569962, 0.92103971, 0.92095331, 0.94275207,
```

```
        0.94450779, 0.94449826]), 'split1_train_score': array([0.423982
61, 0.4378589 , 0.43313634, 0.52467323, 0.52697513,
        0.52796217, 0.58939228, 0.59543514, 0.59832749, 0.67583265,
        0.66969714, 0.67155511, 0.71801122, 0.72443821, 0.72567976,
        0.81395794, 0.83166674, 0.83248985, 0.87988359, 0.88652003,
        0.88689117, 0.91690117, 0.91990356, 0.92075336, 0.93930202,
        0.94386384, 0.94413779]), 'split2_train_score': array([0.443512
65, 0.43157532, 0.43184905, 0.53441443, 0.52803298,
        0.52833472, 0.58927397, 0.59361595, 0.59437662, 0.66204444,
        0.66184514, 0.66157483, 0.70696482, 0.71097096, 0.71479343,
        0.81751595, 0.82773331, 0.82808917, 0.8811396 , 0.88224729,
        0.88442762, 0.91551727, 0.92007969, 0.92003873, 0.94123196,
        0.94369118, 0.94453403]), 'split3_train_score': array([0.438328
  , 0.42591528, 0.42791695, 0.52071378, 0.52113498,
        0.52323859, 0.5881326 , 0.59101744, 0.59260369, 0.66530009,
        0.66624063, 0.66447488, 0.71097453, 0.71388843, 0.71665822,
        0.81099975, 0.81496985, 0.81856281, 0.88168968, 0.88491129,
        0.8837703 , 0.91048218, 0.91870837, 0.91917717, 0.93861616,
        0.94248367, 0.94236033]), 'split4_train_score': array([0.430717
21, 0.43098549, 0.43152661, 0.52696716, 0.52576168,
        0.5256326 , 0.59580671, 0.59585888, 0.59667716, 0.67613408,
        0.6768601 , 0.67622093, 0.729107  , 0.73067035, 0.73081437,
        0.83574275, 0.83807291, 0.83837898, 0.88949672, 0.89255113,
        0.89160827, 0.92314265, 0.92437289, 0.92455876, 0.94181603,
        0.94628481, 0.9471218 ]), 'mean_train_score': array([0.4325110
2, 0.43116017, 0.43127102, 0.52557571, 0.52530393,
        0.5257833 , 0.59021155, 0.59264872, 0.5947857 , 0.66822715,
        0.66686854, 0.66701075, 0.71809098, 0.71912232, 0.72139457,
        0.82071417, 0.82791282, 0.82924503, 0.88327389, 0.88634467,
        0.88661145, 0.91634858, 0.92082084, 0.92109627, 0.94074365,
        0.94416626, 0.94453044]), 'std_train_score': array([0.00738956,
0.00388473, 0.00176396, 0.00498909, 0.00237929,
        0.00209267, 0.00283807, 0.00316549, 0.00241363, 0.00645214,
        0.00608036, 0.00591437, 0.00835676, 0.00731512, 0.00597844,
        0.0089178 , 0.00755644, 0.00647605, 0.00340786, 0.00340935,
        0.00275503, 0.00404948, 0.00192448, 0.00184001, 0.00155092,
        0.00124524, 0.00152182])}
```

In [79]:

```
rfr_best = RandomForestRegressor(n_estimators=100, max_depth=10)
estm(rfr_best, X = feature, y= target)
```

```
train_mean: [0.96268762 0.95542684 0.95017855 0.94705958 0.94422869]
test_mean: [0.85949261 0.91012582 0.91893709 0.92240978 0.92330138]
```



In [80]:

```
# 尝试原来博客中的参数，使用五折交叉验证而不是shufflesplit，树的深度不设置上限
```

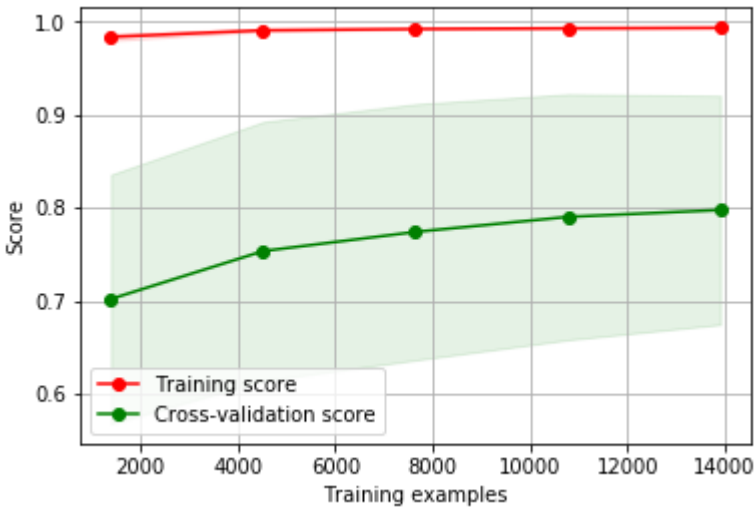
In [82]:

```
def estm2(estimator, X, y, n_jobs=1,cv = 5):

    train_sizes, train_scores, test_scores = learning_curve( \
        estimator, X, y, shuffle=True, cv=cv, n_jobs=n_jobs)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    print("train_mean:", train_scores_mean)
    print("test_mean:", test_scores_mean)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    plt.legend(loc="best")
    plt.show()
```

In [83]:

```
r = RandomForestRegressor(n_estimators=100)
estm2(r, X = feature, y= target)
```

```
train_mean: [0.98351078 0.99064278 0.99200975 0.99274763 0.99327549]
test_mean: [0.701878   0.75343972 0.77403515 0.79018347 0.79756075]
```



In [84]:

```
# 通过还原数据得到结论，树的深度前提可以避免过拟合，用shuffle以此拆交叉验证的效果要提升很多
```

In [ ]:

```

```

In [93]:

```
rfr_best = RandomForestRegressor(n_estimators=100,max_depth=10).fit(feature, target

test_scale = feature
pred = np.array(rfr_best.predict(test_scale))
dic = dict()
dic.update({'pred_result':pred, "datatime":df_train["dteday"], "hour":df_train["hr"
import pandas
df_pre_result = pandas.DataFrame(data=dic)
df_pre_result.head(15)
```

Out[93]:

	pred_result	datatime	hour
0	25.197721	2011-01-01	0
1	28.853542	2011-01-01	1
2	25.540445	2011-01-01	2
3	10.315996	2011-01-01	3
4	2.496642	2011-01-01	4
5	2.477209	2011-01-01	5
6	3.044991	2011-01-01	6
7	30.548464	2011-01-01	7
8	43.904548	2011-01-01	8
9	51.795345	2011-01-01	9
10	69.528911	2011-01-01	10
11	71.463207	2011-01-01	11
12	92.183035	2011-01-01	12
13	111.922991	2011-01-01	13
14	113.988334	2011-01-01	14

In [ ]:

```

```

In [ ]:

```

```