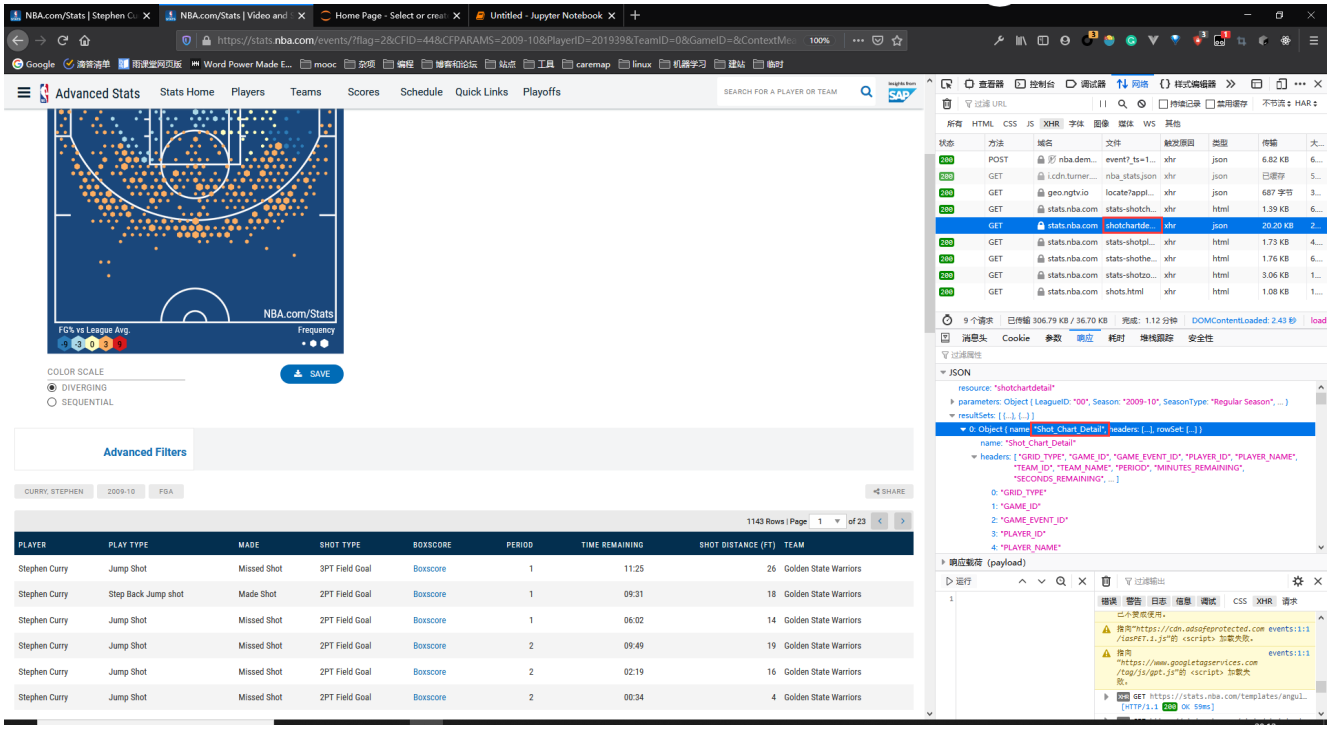
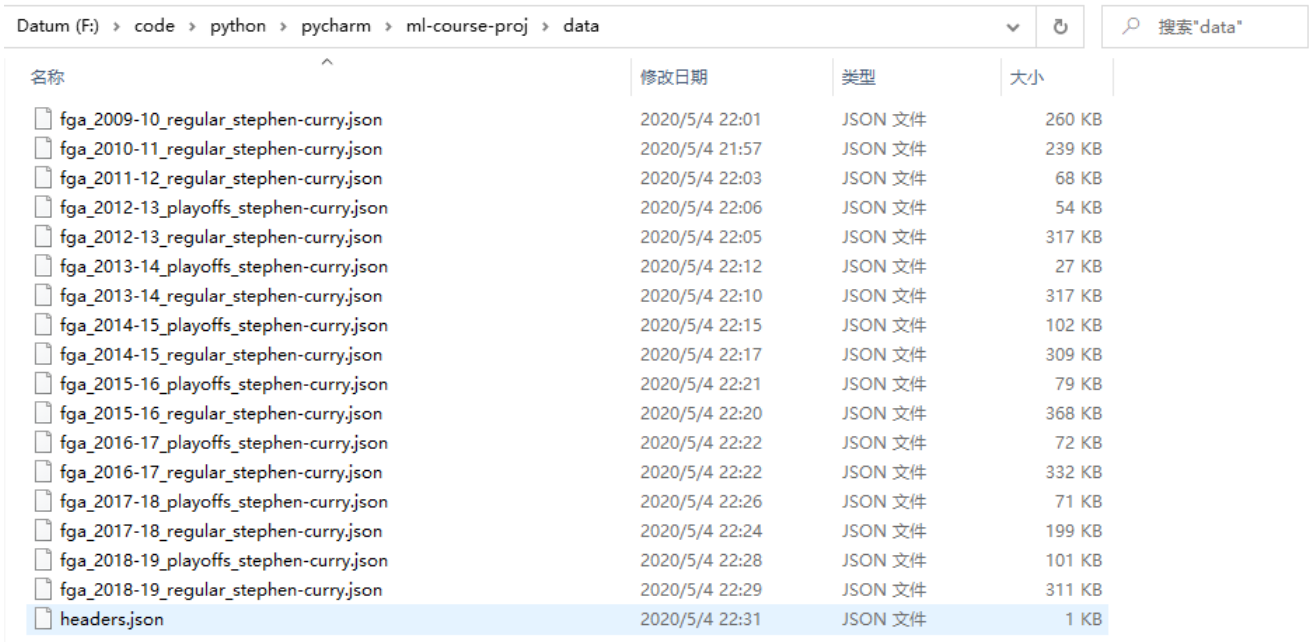
机器学习课程作业 —— Stephen Curry 历史进球预测

by 余梓俊 2018211991

Data Collecting

在 stats.nba.com 中找到库里的信息，点击 FGA（Field Goal Attempt）。在浏览器调试工具的 XHR 网络中请求找到请求的赛季数据，手动保存下 JSON 数据。





Preprocessing

先看看 headers 里面都有哪些字段。

In [1]:

```
import pandas
```

```
pandas.read_json('data/headers.json')
```

Out[1]:

	headers
0	GRID_TYPE
1	GAME_ID
2	GAME_EVENT_ID
3	PLAYER_ID
4	PLAYER_NAME
5	TEAM_ID
6	TEAM_NAME
7	PERIOD
8	MINUTES_REMAINING
9	SECONDS_REMAINING
10	EVENT_TYPE
11	ACTION_TYPE
12	SHOT_TYPE
13	SHOT_ZONE_BASIC
14	SHOT_ZONE_AREA
15	SHOT_ZONE_RANGE
16	SHOT_DISTANCE
17	LOC_X
18	LOC_Y
19	SHOT_ATTEMPTED_FLAG
20	SHOT_MADE_FLAG
21	GAME_DATE
22	HTM
23	VTM

很多信息对于预测某次出手能否进球是没有用的，比如 team_id, grid_type 等等。还有一些是可能有用的信息，但是我们不想用的，比如比赛日期——很有可能某一场比赛库里的手感很好，总是能进球，但是我们希望预测模型专注于某次出手本身，比如投篮类型和距离，而不是利用比赛日期等“取巧”的方式来进行预测。

注意到表中没有表明某一次出手的那一场比赛是常规赛还是季后赛，这可能也是一个影响因素，比如季后赛可能手感更好，专注度更高，但防守强度也可能更高。我们通过文件名包含的信息来加上这个字段。

In [2]:

```
import pandas as pd
raw_data = pd.DataFrame(columns=pd.read_json('data/headers.json').to_numpy().flatten())
raw_data.insert(0, 'PLAYOFFS', None)
print('表格字段')
raw_data
```

表格字段

Out[2]:

PLAYOFFS	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_ID	TEAM_NAME	PERIOD	MINUTES_REMAINING

0 rows × 25 columns

In [3]:

```
import os
```

```
headers = pd.read_json('data/headers.json').to_numpy().flatten().tolist()
data_dir = 'data'
num_regular = 0
num_playoffs = 0

print('读取赛季数据')
for file in os.listdir(data_dir):
    if file.find('regular') != -1:
        playoffs = 0
        num_regular += 1
    elif file.find('playoffs') != -1:
        playoffs = 1
        num_playoffs += 1
    elif file.find('headers') != -1:
        continue
    else: raise Exception(f'wrong filename: {file}')

    print(file)
    one_data = pd.read_json(os.path.join(data_dir, file), orient='values')
    one_data.columns = headers
    one_data.insert(0, 'PLAYOFFS', playoffs)
    raw_data = raw_data.append(one_data)

print(f'Total: {num_regular} regulars, {num_playoffs} playoffs')
raw_data
```

```
读取赛季数据
fga_2009-10_regular_stephen-curry.json
fga_2010-11_regular_stephen-curry.json
fga_2011-12_regular_stephen-curry.json
fga_2012-13_playoffs_stephen-curry.json
fga_2012-13_regular_stephen-curry.json
fga_2013-14_playoffs_stephen-curry.json
fga_2013-14_regular_stephen-curry.json
fga_2014-15_playoffs_stephen-curry.json
fga_2014-15_regular_stephen-curry.json
fga_2015-16_playoffs_stephen-curry.json
fga_2015-16_regular_stephen-curry.json
fga_2016-17_playoffs_stephen-curry.json
fga_2016-17_regular_stephen-curry.json
fga_2017-18_playoffs_stephen-curry.json
fga_2017-18_regular_stephen-curry.json
fga_2018-19_playoffs_stephen-curry.json
fga_2018-19_regular_stephen-curry.json
Total: 10 regulars, 7 playoffs
```

Out[3]:

	PLAYOFFS	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_ID	TEAM_NAME	PERIOD	MINUTES_RE
0	0	Shot Chart Detail	20900015	4	201939	Stephen Curry	1610612744	Golden State Warriors	1	
1	0	Shot Chart Detail	20900015	17	201939	Stephen Curry	1610612744	Golden State Warriors	1	
2	0	Shot Chart Detail	20900015	53	201939	Stephen Curry	1610612744	Golden State Warriors	1	
3	0	Shot Chart Detail	20900015	141	201939	Stephen Curry	1610612744	Golden State Warriors	2	
4	0	Shot Chart Detail	20900015	249	201939	Stephen Curry	1610612744	Golden State Warriors	2	
...	...	...	...	...	...	...	...	...	...	...
1335	0	Shot Chart Detail	21801205	533	201939	Stephen Curry	1610612744	Golden State Warriors	3	
1336	0	Shot Chart Detail	21801215	25	201939	Stephen Curry	1610612744	Golden State Warriors	1	
1337	0	Shot Chart Detail	21801215	48	201939	Stephen Curry	1610612744	Golden State Warriors	1	
1338	0	Shot Chart Detail	21801215	61	201939	Stephen Curry	1610612744	Golden State Warriors	1	

	PLAYOFFS	Shot Type	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_ID	Team Name	PERIOD	MINUTES_RE
1339	0	Detail	21601213	114	201939	Stephen Curry	1610612744	Warriors	1	

14020 rows × 25 columns

按照之前的想法，选择需要用到的信息。

In [4]:

```
raw_data['HOME_TEAM'] = raw_data.apply(lambda x: 1 if x.HTM == 'GSW' else 0, axis=1)
raw_data = raw_data.loc[:, ['PLAYOFFS', 'PERIOD', 'MINUTES_REMAINING', 'ACTION_TYPE', \
                'SHOT_TYPE', 'SHOT_ZONE_BASIC', 'SHOT_ZONE_AREA', \
                'SHOT_DISTANCE', 'SHOT_MADE_FLAG', 'LOC_X', 'LOC_Y', 'HOME_TEAM']]
raw_data
```

Out[4]:

	PLAYOFFS	PERIOD	MINUTES_REMAINING	ACTION_TYPE	SHOT_TYPE	SHOT_ZONE_BASIC	SHOT_ZONE_AREA	SHOT_DISTANCE	S
0	0	1	11	Jump Shot	3PT Field Goal	Above the Break 3	Right Side Center(RC)	26	
1	0	1	9	Step Back Jump shot	2PT Field Goal	Mid-Range	Left Side Center(LC)	18	
2	0	1	6	Jump Shot	2PT Field Goal	In The Paint (Non-RA)	Center(C)	14	
3	0	2	9	Jump Shot	2PT Field Goal	Mid-Range	Left Side(L)	19	
4	0	2	2	Jump Shot	2PT Field Goal	Mid-Range	Left Side Center(LC)	16	
...	...	...	...	...	...	...	...	...	...
1335	0	3	0	Step Back Jump shot	3PT Field Goal	Above the Break 3	Left Side Center(LC)	30	
1336	0	1	9	Step Back Jump shot	2PT Field Goal	Mid-Range	Left Side(L)	18	
1337	0	1	8	Jump Shot	3PT Field Goal	Above the Break 3	Center(C)	26	
1338	0	1	6	Jump Shot	3PT Field Goal	Above the Break 3	Right Side Center(RC)	23	
1339	0	1	2	Jump Shot	2PT Field Goal	Mid-Range	Left Side(L)	12	

14020 rows × 12 columns

In [5]:

```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14020 entries, 0 to 1339
Data columns (total 12 columns):
PLAYOFFS             14020 non-null object
PERIOD               14020 non-null object
MINUTES_REMAINING    14020 non-null object
ACTION_TYPE          14020 non-null object
SHOT_TYPE            14020 non-null object
SHOT_ZONE_BASIC      14020 non-null object
SHOT_ZONE_AREA       14020 non-null object
SHOT_DISTANCE        14020 non-null object
SHOT_MADE_FLAG       14020 non-null object
LOC_X                14020 non-null object
LOC_Y                14020 non-null object
HOME_TEAM            14020 non-null int64
dtypes: int64(1), object(11)
memory usage: 1.4+ MB
```

playoffs, period, minutes_remaining, shot_distance, shot_made_flag, loc_x, loc_y 这些字段应当为整数，先进

playoffs, period, minutes_remaining, shot_distance, shot_made_flag, loc_x, loc_y 这些字段应当为整数，先进行一下转换。

In [6]:

```
for field in ['PLAYOFFS', 'PERIOD', 'MINUTES_REMAINING', \
              'SHOT_DISTANCE', 'SHOT_MADE_FLAG', 'LOC_X', 'LOC_Y', 'HOME_TEAM']:
    raw_data[field] = raw_data[field].astype('int')

raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14020 entries, 0 to 1339
Data columns (total 12 columns):
PLAYOFFS             14020 non-null int32
PERIOD               14020 non-null int32
MINUTES_REMAINING    14020 non-null int32
ACTION_TYPE          14020 non-null object
SHOT_TYPE            14020 non-null object
SHOT_ZONE_BASIC      14020 non-null object
SHOT_ZONE_AREA       14020 non-null object
SHOT_DISTANCE        14020 non-null int32
SHOT_MADE_FLAG       14020 non-null int32
LOC_X                14020 non-null int32
LOC_Y                14020 non-null int32
HOME_TEAM            14020 non-null int32
dtypes: int32(8), object(4)
memory usage: 985.8+ KB
```

保存下这个 raw_data，后面就在这个 raw_data 上开展工作。

In [7]:

```
import os
import pandas as pd

if not os.path.exists('raw_data.csv'):
    raw_data.to_csv('raw_data.csv', index=0)
raw_data = pd.read_csv('raw_data.csv')
```

先用 pandas_profiling 这个包生成一份数据报告，大致浏览一下数据。

In [8]:

```
import pandas_profiling as ppf
ppf.ProfileReport(raw_data)
```

```
variables: 100%|| 12/12 [00:05<0
0:00,  2.40it/s]
correlations [recoded]: 100%|| 6/6 [00:06<0
0:00,  1.06s/it]
interactions [continuous]: 100%|| 25/25 [00:05<0
0:00,  4.89it/s]
table: 100%|| 1/1 [00:00<0
0:00,  9.62it/s]
missing [matrix]: 100%|| 2/2 [00:00<0
0:00,  2.76it/s]
warnings [correlations]: 100%|| 3/3 [00:00<00
:00, 428.63it/s]
package: 100%|| 1/1 [00:00<0
0:00, 83.33it/s]
build report structure: 100%|| 1/1 [00:04<0
0:00,  4.82s/it]
```

Out[8]:

首先注意到 action type 的种类有五十多个，数据非常不平衡，并且 one-hot 编码会导致字段太多，因此把出现次数少于100的 action type 都归为单独的一个 other（或者使用 clustering 等技术，不太懂，不考虑）。

In [9]:

```
action_type = raw_data.ACTION_TYPE.value_counts()

print(action_type)
rearly_used_boundary = 0
for count in action_type.data:
    if count < 100: break
    rearly_used_boundary += 1
rearly_used_actions = action_type.index.to_list()[rearly_used_boundary : ]
print(rearly_used_actions)
raw_data['ACTION_TYPE'] = raw_data.apply(lambda x: x.ACTION_TYPE if not x.ACTION_TYPE in  rearly_used_a
ctions else 'Other', axis=1)
print(raw_data.ACTION_TYPE.value_counts())
```

```
Jump Shot                           6682
Pullup Jump shot                    2123
Step Back Jump shot                  939
Driving Layup Shot                   773
Layup Shot                           604
Floating Jump shot                   455
Driving Finger Roll Layup Shot       432
Running Jump Shot                    252
Driving Reverse Layup Shot           194
Running Layup Shot                   158
Cutting Layup Shot                   141
Driving Floating Jump Shot           128
```

```
Jump Bank Shot                          125
Reverse Layup Shot                      124
Turnaround Jump Shot                    113
Fadeaway Jump Shot                      113
Running Bank shot                        75
Running Finger Roll Layup Shot           70
Running Pull-Up Jump Shot                68
Driving Floating Bank Jump Shot          60
Driving Bank shot                        45
Finger Roll Layup Shot                   42
Running Reverse Layup Shot               38
Driving Jump shot                        36
Turnaround Fadeaway shot                 33
Cutting Finger Roll Layup Shot           31
Turnaround Bank shot                     23
Pullup Bank shot                         22
Putback Layup Shot                       12
Driving Hook Shot                        10
Driving Dunk Shot                        10
Hook Shot                                10
Running Hook Shot                        10
Slam Dunk Shot                            7
Tip Layup Shot                            7
Tip Shot                                  7
Dunk Shot                                 7
No Shot                                   6
Fadeaway Bank shot                        6
Running Dunk Shot                         6
Turnaround Hook Shot                      4
Turnaround Fadeaway Bank Jump Shot        3
Step Back Bank Jump Shot                  3
Driving Slam Dunk Shot                    2
Jump Hook Shot                            2
Driving Bank Hook Shot                    2
Alley Oop Layup shot                      1
Putback Dunk Shot                         1
Hook Bank Shot                            1
Running Slam Dunk Shot                    1
Turnaround Bank Hook Shot                 1
Cutting Dunk Shot                         1
Running Bank Hook Shot                    1
Name: ACTION_TYPE, dtype: int64
['Running Bank shot', 'Running Finger Roll Layup Shot', 'Running Pull-Up Jump Shot', 'Driving Floating
Bank Jump Shot', 'Driving Bank shot', 'Finger Roll Layup Shot', 'Running Reverse Layup Shot', 'Driving
Jump shot', 'Turnaround Fadeaway shot', 'Cutting Finger Roll Layup Shot', 'Turnaround Bank shot', 'Pull
up Bank shot', 'Putback Layup Shot', 'Driving Hook Shot', 'Driving Dunk Shot', 'Hook Shot', 'Running Ho
ok Shot', 'Slam Dunk Shot', 'Tip Layup Shot', 'Tip Shot', 'Dunk Shot', 'No Shot', 'Fadeaway Bank shot',
'Running Dunk Shot', 'Turnaround Hook Shot', 'Turnaround Fadeaway Bank Jump Shot', 'Step Back Bank Jump
Shot', 'Driving Slam Dunk Shot', 'Jump Hook Shot', 'Driving Bank Hook Shot', 'Alley Oop Layup shot', 'P
utback Dunk Shot', 'Hook Bank Shot', 'Running Slam Dunk Shot', 'Turnaround Bank Hook Shot', 'Cutting Du
nk Shot', 'Running Bank Hook Shot']
Jump Shot                         6682
Pullup Jump shot                  2123
Step Back Jump shot                939
Driving Layup Shot                 773
Other                              664
Layup Shot                         604
Floating Jump shot                 455
Driving Finger Roll Layup Shot     432
Running Jump Shot                  252
Driving Reverse Layup Shot         194
Running Layup Shot                 158
Cutting Layup Shot                 141
Driving Floating Jump Shot         128
Jump Bank Shot                     125
Reverse Layup Shot                 124
Fadeaway Jump Shot                 113
Turnaround Jump Shot               113
Name: ACTION_TYPE, dtype: int64
```

还看到 Profile 中提示 shot zone basic 和 shot distance 存在高相关性，同时在 Correlations 图表中观察到 shot distance 和 loc_y 也存在一定程度的正相关，这些都属于意料之内，暂时不知道是否应该处理以及如何处理。

特征工程就做这样简单的几步，接下来对离散值进行 one-hot 编码，然后分隔训练集和测试集，因为没有什么特殊的字段需要考虑，方便起见，直接整体随机三七分。

In [10]:

```
from category_encoders.one_hot import OneHotEncoder

one_hot_encoder = OneHotEncoder(return_df=True, use_cat_names=True)
raw_data = one_hot_encoder.fit_transform(raw_data)
raw_data
```

Out[10]:

	PLAYOFFS	PERIOD	MINUTES_REMAINING	ACTION_TYPE_Jump Shot	ACTION_TYPE_Step Back Jump shot	ACTION_TYPE_Pullup Jump shot	ACTION_TYPE_Layup Shot
0	0	1	11	1	0	0	0
1	0	1	9	0	1	0	0
2	0	1	6	1	0	0	0
3	0	2	9	1	0	0	0
4	0	2	2	1	0	0	0
...	...	...	...	...	...	...	...
14015	0	3	0	0	1	0	0
14016	0	1	9	0	1	0	0
14017	0	1	8	1	0	0	0
14018	0	1	6	1	0	0	0
14019	0	1	2	1	0	0	0

14020 rows × 40 columns

In [11]:

```
from sklearn.model_selection import train_test_split

y = raw_data['SHOT_MADE_FLAG'].copy()
raw_data.drop(['SHOT_MADE_FLAG'], axis=1, inplace=True)
X = raw_data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.3)

print(f'X_train size: {X_train.shape}, X_test size: {X_test.shape}\ny_train size: {y_train.shape}, y_te
st size: {y_test.shape}')
```

```
X_train size: (9814, 39), X_test size: (4206, 39)
y_train size: (9814,), y_test size: (4206,)
```

Model Selecting and Training

课程中学习的分类器有

- 逻辑回归
- 神经网络
- 支持向量机

分别应用这三种分类器进行训练

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import validation_curve
```

In [13]:

```
def plot_learning_curve(estimator, title, X, y, train_sizes=np.linspace(.1, 1.0, 5)):
```

```
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, train_sizes=train_sizes, return_times=True)
    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)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    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.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.text(train_sizes, train_scores_mean, train_scores_mean)
#     plt.text(train_sizes, test_scores_mean, test_scores_mean)
    for a, b in zip(train_sizes, train_scores_mean):
#         plt.text(a, b, b, ha='center', va='bottom', fontsize=20)
        plt.text(a, b, '%.2f' % b, fontsize=10)
    for a, b in zip(train_sizes, test_scores_mean):
        plt.text(a, b, '%.2f' % b, fontsize=10)
    plt.legend(loc="best")
    plt.show()
```

In [14]:

```
def plot_validation_curve(estimator, title, X, y, param_name, param_range):
    train_scores, test_scores = validation_curve(estimator, X, y, param_name, param_range)
    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)
    plt.title(title)
    plt.xlabel(param_name)
    plt.ylabel("Score")

    plt.plot(param_range, train_scores_mean, 'o-', color="r",
                 label="Training score")
    plt.plot(param_range, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    plt.fill_between(param_range, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
#     plt.text(param_range, train_scores_mean, train_scores_mean)
#     plt.text(param_range, test_scores_mean, test_scores_mean)
    for a, b in zip(param_range, train_scores_mean):
        plt.text(a, b, '%.2f' % b, fontsize=10)
    for a, b in zip(param_range, test_scores_mean):
        plt.text(a, b, '%.2f' % b, fontsize=10)
    plt.legend(loc="best")
    plt.show()
```

逻辑回归

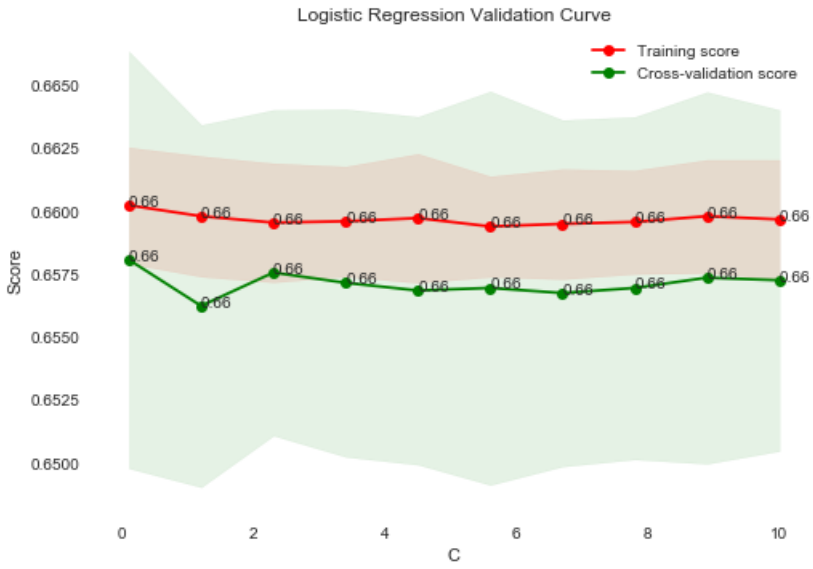
使用 l2 正则化，画出在一系列不同的 C（lambda 的倒数）下模型的训练集和验证集得分的变化情况。

In [15]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=5000)
```

```
plot_validation_curve(lr, 'Logistic Regression Validation Curve', X_train, y_train, 'C', np.linspace(0.1, 10, 10))
```



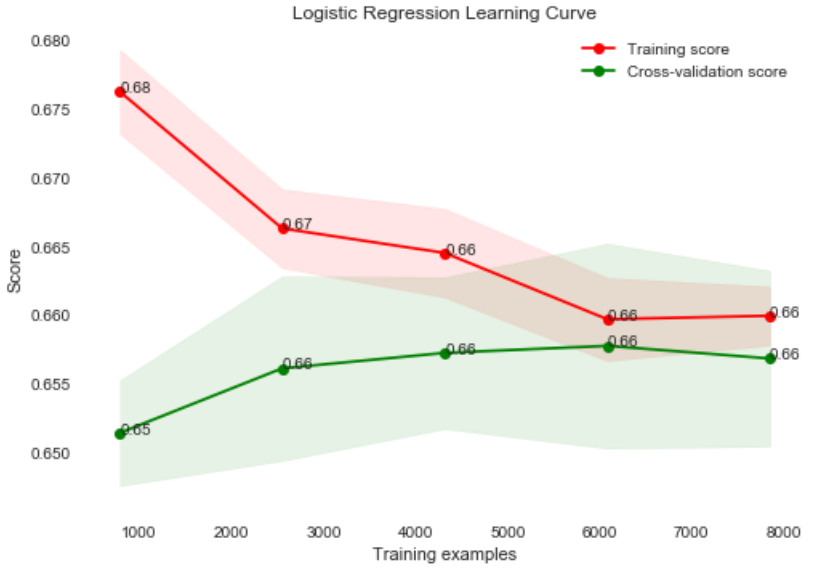
从图中可以看出，模型“很好”地拟合了验证集，这里的“很好”当然是相对于训练集来说，在训练集和测试集上的准确率分别为 65% 和 66%，而且几乎不随惩罚项系数 C 的变化而变化，这说明虽然逻辑回归虽然完全没有过拟合，但碍于线性模型有限的能力，表现出了明显的欠拟合（乱猜的猜中的概率也有 50%）。

当然也有可能是这些特征提供的信息本身就确实不足以预测进球，毕竟能否进球是非常“主观”的，很大程度上取决于出手时的手感。

既然 C 对交叉验证的结果影响很小，这里就直接用逻辑回归的默认参数（C=1）来计算一下测试集上的得分。

In [16]:

```
lr = LogisticRegression(max_iter=5000)
plot_learning_curve(lr, 'Logistic Regression Learning Curve', X_train, y_train)
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```



Out[16]:

```
0.6507370423204946
```

神经网络

神经网络的超参数就多了，鉴于我对机器学习模型的 insight 还非常浅薄，NN 的丹并不好炼，这里直接用 scikit-learn MLPClassifier 的默认超参数先进行尝试（隐藏层默认为 1 层，包含 100 个神经元，这里改为 10 个，其他保持默认）。

In [17]:

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(hidden_layer_sizes=(10,))
nn.fit(X_train, y_train)
nn.score(X_test, y_test)
```

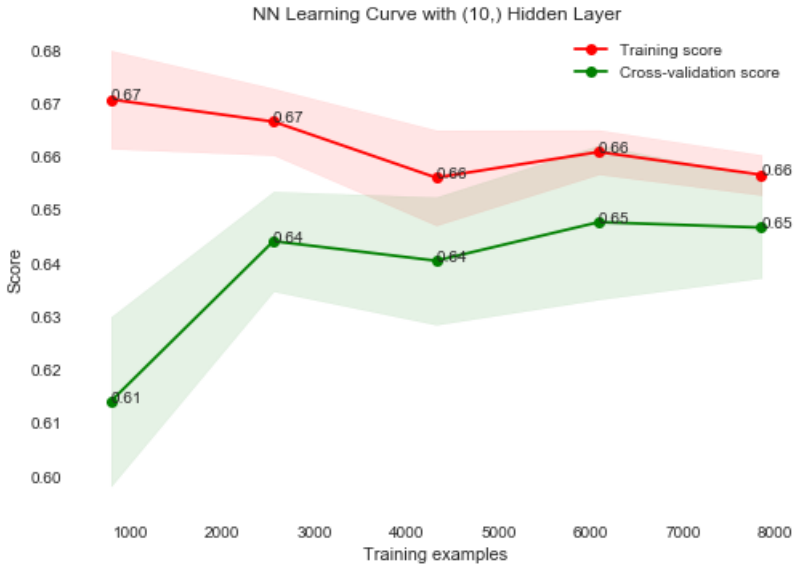
Out[17]:

```
0.6438421302900618
```

结果居然比逻辑回归还要低了 1 个百分点，绘制一下它的学习曲线看一下。

In [18]:

```
plot_learning_curve(nn, 'NN Learning Curve with (10,) Hidden Layer', X_train, y_train)
```

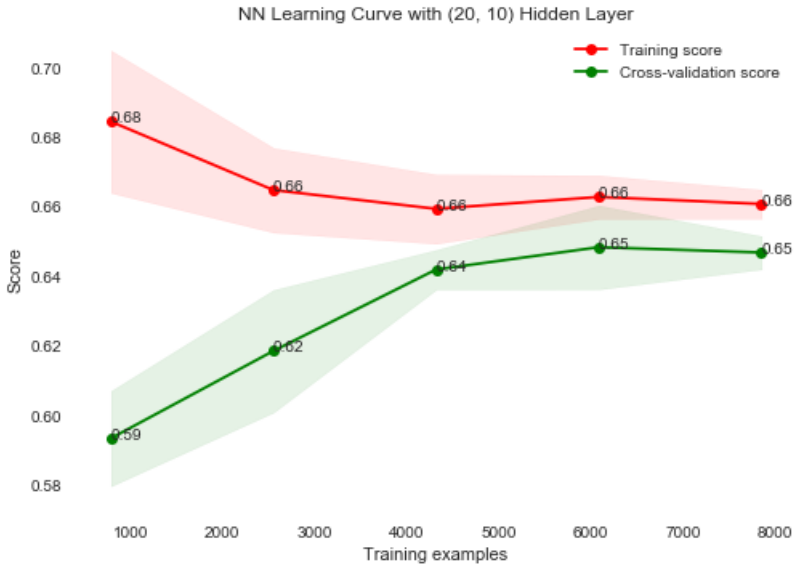


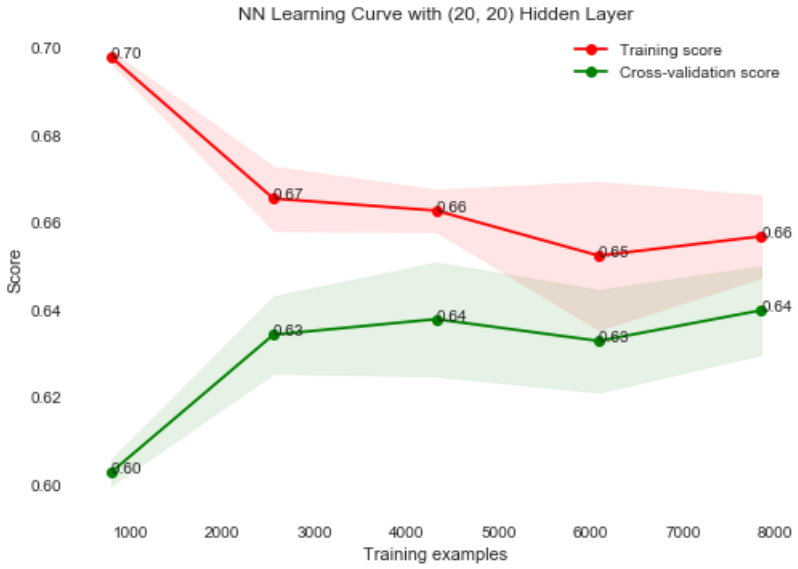
没有看出什么有用的信息，似乎仍然是欠拟合。

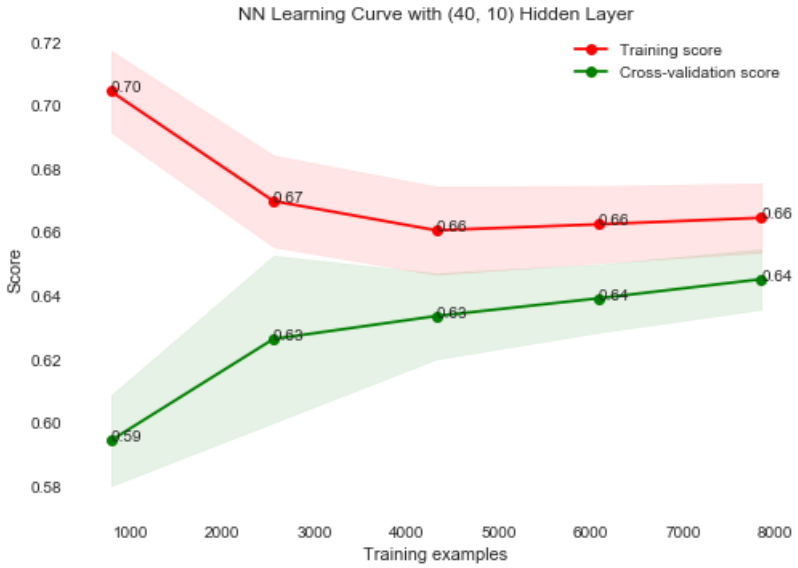
随意修改一下隐藏层，试一下（20，10），（20，20），（40，10），（40，20）

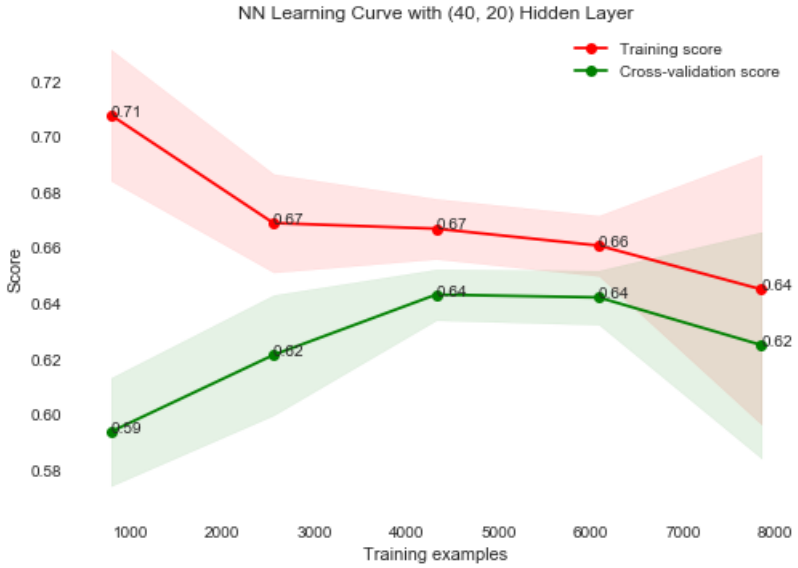
In [19]:

```
for hidden_layers in [(20, 10), (20, 20), (40, 10), (40, 20)]:
    nn = MLPClassifier(hidden_layer_sizes=hidden_layers)
    plot_learning_curve(nn, f'NN Learning Curve with {hidden_layers} Hidden Layer', X_train, y_train)
```









情况还是类似，上面四种更好的应该是（20，10）这个隐藏层结构，但是除了方差变小，总体效果也依然和逻辑回归相差无几。

支持向量机

课堂上讲解过线性核和高斯核的支持向量机分类器，这里也采用这两种核，先用 scikit-learn 的 GridSearchCV 类进行粗略的网格搜

索，找到两个核中得分更高的核及其合适的超参数。

In [20]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

param_grid = [
  {'C': [0.01, 0.1, 1, 5, 10, 50], 'kernel': ['linear']},
  {'C': [0.01, 0.1, 1, 5, 10, 50], 'gamma': [0.0001, 0.001, 0.01], 'kernel': ['rbf']},
]

search = GridSearchCV(SVC(), param_grid)
# search.fit(X_train, y_train)
# svc = search.best_estimator_

# print(search.best_params_)
# print(f'Train set score: {svc.score(X_train, y_train)}')
# print(f'Test set score: {svc.score(X_test, y_test)}')
```

```
{'C': 1, 'kernel': 'linear'}
Train set score: 0.6610963929080905
Test set score: 0.6516880646695198
```

结果表明在初次粗略的网格搜索中，最优的 SVC 是 C=1 的线性核，但结果仍然只有 65%。

SVM 的训练内存和时间随样本增大而增大，这里的 8000 多个样本也已经非常耗时（scikit-learn 无法使用 GPU 加速，此处也没有设置 n_jobs 参数来并行计算），故不再对 SVM 进行调优。

由于 SVM 训练较慢，我在这里放一张上面的 cell 的运行结果截图，写完整个 notebook 后再 run all 一次 kernel，届时会注释掉上面的 cell，让其不再运行。

随机森林

考虑到上面的模型的结果都不是很理想，这里再额外尝试一个集成模型：随机森林。

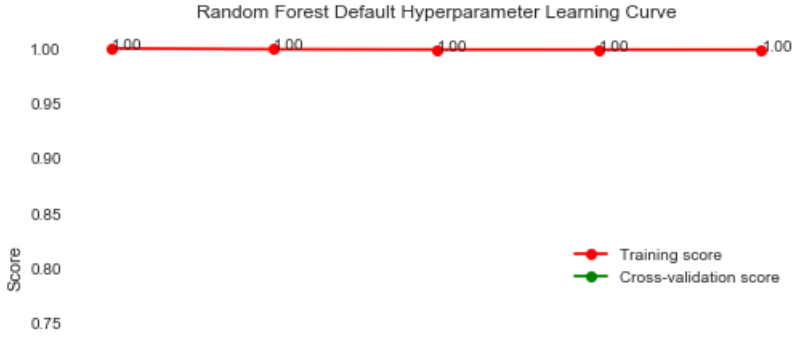
> The main parameters to adjust when using these methods is `n_estimators` and `max_features`. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are `max_features=None` (always considering all features instead of a random subset) for regression problems, and `max_features="sqrt"` (using a random subset of size sqrt(n_features)) for classification tasks (where n_features is the number of features in the data). Good results are often achieved when setting `max_depth=None` in combination with `min_samples_split=2` (i.e., when fully developing the trees).

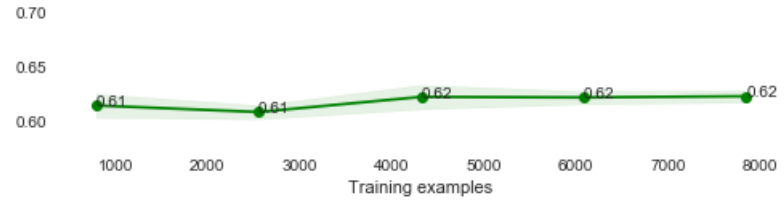
参考 scikit-learn User Guide 里的讲解，通常树的数量越多模型会越好，分类问题通常选用 `max_features=="sqrt"`，这里先采用默认参数（树的数量为 100）绘制一下学习曲线，并用 n_jobs 来使用多核 CPU 来并行加速。

In [21]:

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(random_state=0, n_jobs=-1)
plot_learning_curve(forest, 'Random Forest Default Hyperparameter Learning Curve', X_train, y_train)
```



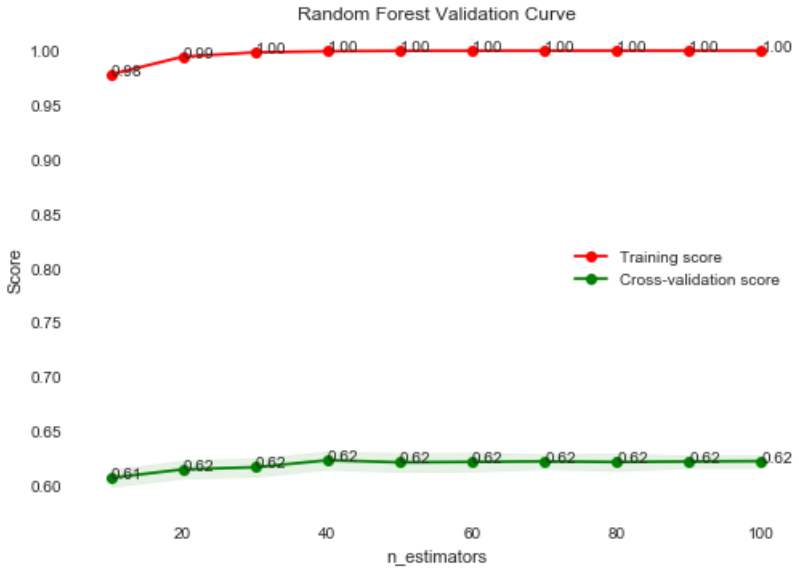


可以看到，随机森林明显地过拟合了,并且从一个很小的样本量开始，它的验证集得分就几乎不再增长，但即使如此，其验证集得分并没有比其他模型差太多。因此我们再探索一下随机森林。

先调树的数量这个超参数。

In [22]:

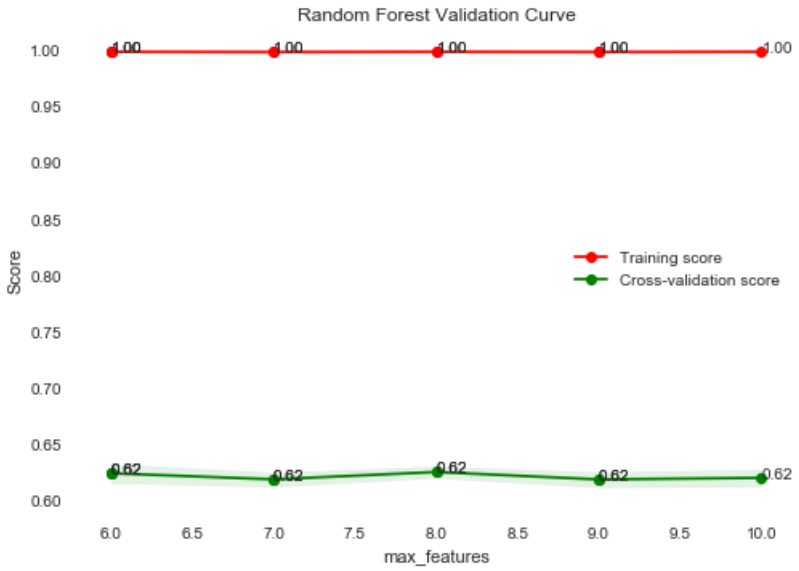
```
plot_validation_curve(forest, 'Random Forest Validation Curve', X_train, y_train, 'n_estimators', np.li
nspace(10, 100, 10, dtype=int))
```



树的数量增加对模型的过拟合并没有起到很好的作用，接下来选定树的数量为 40，再试一下调整 max feature。

In [23]:

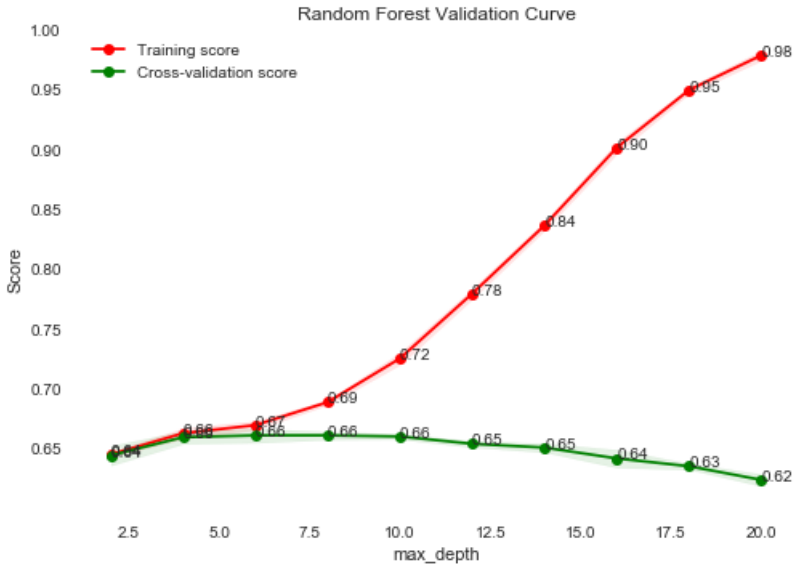
```
forest = RandomForestClassifier(random_state=0, n_jobs=-1, n_estimators=40)
plot_validation_curve(forest, 'Random Forest Validation Curve', X_train, y_train, 'max_features', np.li
nspace(6, 10, 10, dtype=int))
```



效果依然不佳，随意选定 max features 为 8，再试一下调整 max_depth。

In [24]:

```
forest = RandomForestClassifier(random_state=0, n_jobs=-1, n_estimators=40, max_features=8)
plot_validation_curve(forest, 'Random Forest Validation Curve', X_train, y_train, 'max_depth', np.linsp
ace(2, 20, 10, dtype=int))
```



可以看到，在 n_estimator=40, max_features=8 的基础上，随着树最大深度的增加，随机森林的过拟合明显加剧。

再细微地搜索一下这三个参数。

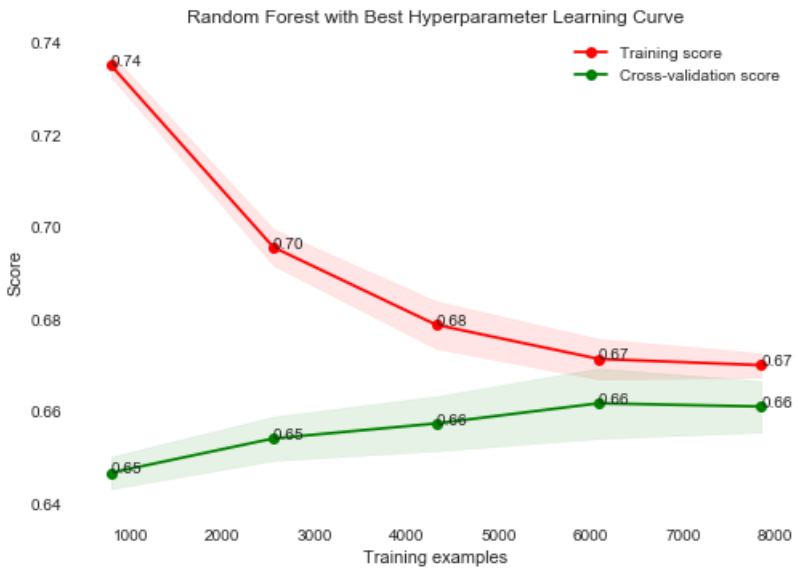
In [25]:

```
from sklearn.model_selection import GridSearchCV
# import pandas as pd

forest = RandomForestClassifier(random_state=0, n_jobs=-1)
param_grid = {'n_estimators': [15, 20, 25, 30], 'max_features': [5, 6, 7, 8, 9], 'max_depth': [4, 5, 6]
}
search = GridSearchCV(forest, param_grid)
search.fit(X_train, y_train)

forest = search.best_estimator_
print(search.best_params_)
plot_learning_curve(forest, 'Random Forest with Best Hyperparameter Learning Curve', X_train, y_train)
```

```
{'max_depth': 6, 'max_features': 8, 'n_estimators': 25}
```



随机森林的最终结果依然和前几个模型差不多。

Model Ensembling

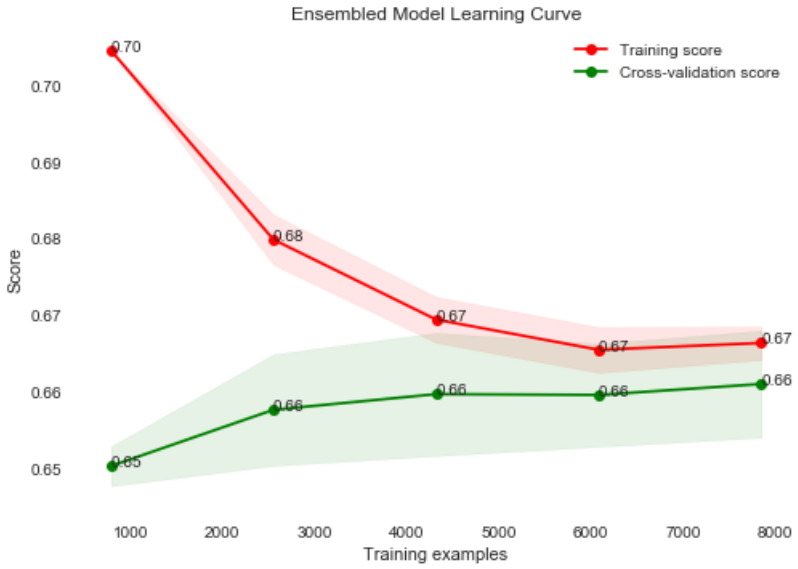
上面用了四个模型进行训练，但最终准确率无一例外都在 66% 左右，因此有理由怀疑，这些模型可能都找到了相同的规律，但这些规律并不是很有效的规律，也就是说这些特征——出手距离、投篮区域、投篮类型等等，并不能很好地预测投篮能否进球（当然也有可能是因为炼丹水平太差），所以模型融合可能不会带来太大的提升。

In [26]:

```
from sklearn.ensemble import StackingClassifier

estimators = [
    ('lr', LogisticRegression(max_iter=5000, n_jobs=-1, random_state=0)),
    ('nn', MLPClassifier(hidden_layer_sizes=(20, 10), random_state=0)),
    # svm is too slow
    # ('svc',SVC(kernel='linear', C=1, random_state=0)),
    ('rf', RandomForestClassifier(n_estimators=25, max_features=8, max_depth=6, n_jobs=-1, random_state
=0))
]

clf = StackingClassifier(estimators=estimators)
plot_learning_curve(clf, 'Ensembled Model Learning Curve', X_train, y_train)
```



Conclusion

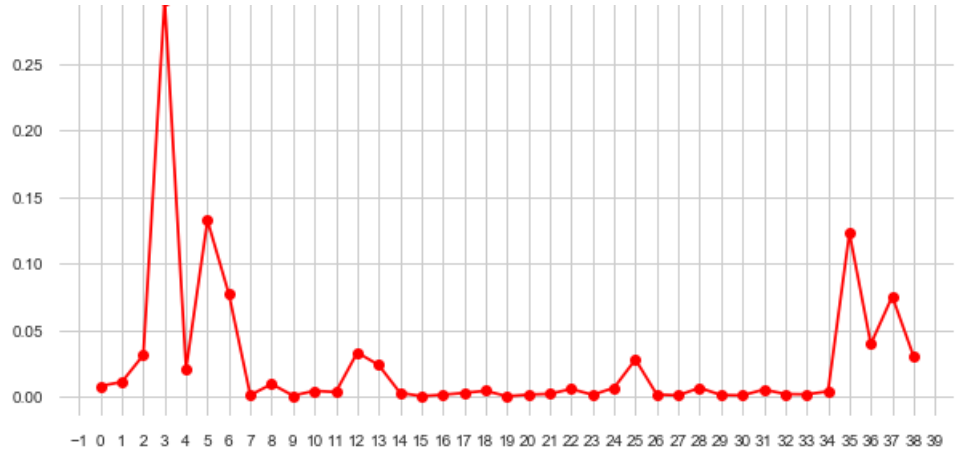
大作业总结

上面说过，很可能这些模型都学到了一样的规律，而这些规律不足以预测进球。这里通过绘制一下前面的随即森林和逻辑回归学到的各个特征的权重，来看一下这两个模型学习到了怎样的规律。

In [27]:

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator

plt.figure(figsize=(10, 5))
plt.title('Random Forest Feature Importance')
plt.plot([x for x in range(39)], forest.feature_importances_, 'o-', color="r")
plt.gca().xaxis.set_major_locator(MultipleLocator(1))
plt.grid()
plt.show()
```



In [28]:

```
features = X_train.columns
features[3], features[5], features[6], features[35], features[37]
```

Out[28]:

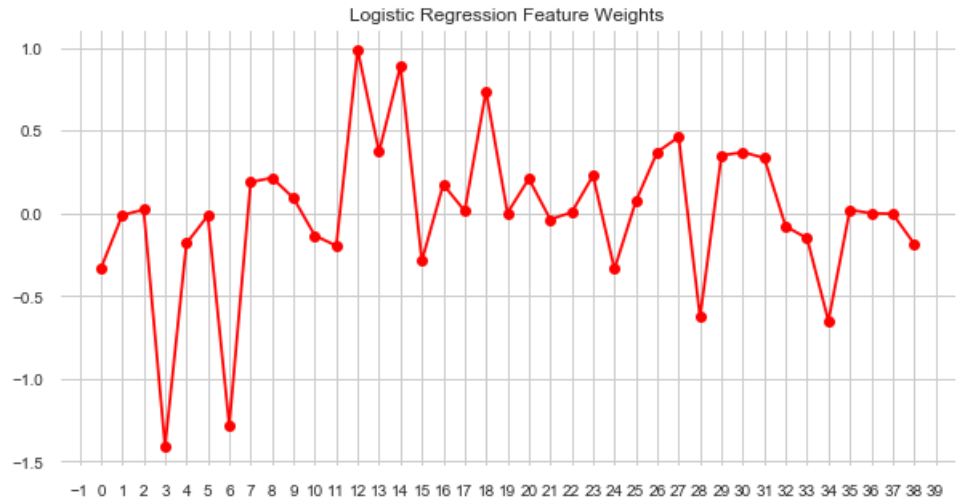
```
('ACTION_TYPE_Jump Shot',
 'ACTION_TYPE_Pullup Jump shot',
 'ACTION_TYPE_Layup Shot',
 'SHOT_DISTANCE',
 'LOC_Y')
```

从结果可以看到，跳投、急停跳投、上篮、出手距离、Y 轴距离（和出手距离有较大的相关性）最为重要的几个特征，这符合我们的直觉。

再来看看逻辑回归。

In [29]:

```
plt.figure(figsize=(10, 5))
plt.title('Logistic Regression Feature Weights')
plt.plot([x for x in range(39)], lr.coef_.reshape(-1), 'o-', color="r")
plt.gca().xaxis.set_major_locator(MultipleLocator(1))
plt.grid()
plt.show()
```



In [30]:

```
features = X_train.columns
features[3], features[6], features[12], features[14], features[18], features[28], features[34]
```

Out[30]:

```
('ACTION_TYPE_Jump Shot',
 'ACTION_TYPE_Layup Shot',
 'ACTION_TYPE_Driving Finger Roll Layup Shot',
 'ACTION_TYPE_Driving Reverse Layup Shot',
 'ACTION_TYPE_Cutting Layup Shot',
 'SHOT_ZONE_BASIC_Backcourt',
 'SHOT_ZONE_AREA_Back Court(BC)')
```

逻辑回归学习到的规律和随机森林还是有所区别的，例如在逻辑回归认为比较重要的特征中，有另外三个出手类型比较重要，但是随机森林并不认为。逻辑回归还指出在后场出手的球可能不会进，这和随机森林表明的 Y 轴距离也是高相关的特征。

总体而言，这个小项目并不算成功，在乱猜也的正确概率为 50%（二分类）的情况下，所有尝试过的模型在验证集上都只能取得 66% 左右的准确率，在测试集上也是 65、66 上下的水平（有些模型没有计算最后在测试集上的得分，但能预想地到得分也就是在 66% 左右，毕竟除了逻辑回归，其他模型的交叉验证的方差都是比较小的）。

课程总结

课程总体时间不长，介绍的机器学习模型也有限，但相比直接参考 API 文档调包来“学习”机器学习，课堂中进行的数学公式推导的学习、数学模型直观理解的学习，更能加深对机器模型本质的理解，也让人对机器模型背后的数学产生敬意。例如：在上神经网络的时候，我通过学习网上的文章: 矩阵推导术（上）(https://zhuanlan.zhihu.com/p/24709748)，自己动手依样画瓢写了一遍矩阵形式的反向传播推导过程，不仅简单复习了一下大一的数学，也使得以后在使用逻辑回归和多层感知机时更有底气和自信。

但在完成作业尤其是这个大作业的过程中，我依然感觉到对模型本质的不理解带来的参数调优和模型选择的困难，同时我也认识到，只有加强对模型本质的理解，才能理解模型到底学到了什么知识，才能理解模型好或者不好的原因是什么。

但性格使然，学习到后期时动力便有所下降，支持向量机到现在依然是一知半解，面对这么多的核以及对应的超参数，调参时感到无从下手。可能相比于机器学习这样“费时”且不能百分之一百看到预期结果的工程，我更倾向于传统的软件开发工程，能带给我更多的踏实感（当然可能是因为数学基础不够，不能理解模型和算法）。

总之，课程的收获还是不少的，也希望以后能慢慢地掌握更多数学工具，学会更多机器学习的方法，并在实际中使用它们。