In [1]:
```
!pip install xgboost
from xgboost import XGBClassifier
import xgboost as xgb

from sklearn.model_selection import GridSearchCV

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

```
Collecting xgboost
  Downloading xgboost-1.7.1-py3-none-win_amd64.whl (89.1 MB)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.7.3)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.1
```

In [2]:
```
train = pd.read_csv(r"D:\Otto_train.csv")
```

In [3]:
```
# drop ids and get labels
y_train = train['target']

#XGBoost只接受数字型标签
y_train = y_train.map(lambda s: s[6:])

y_train = y_train.map(lambda s: int(s)-1)
print(y_train.head())
X_train = train.drop(["id", "target"], axis=1)

#保存特征名字以备后用（可视化）
feat_names = X_train.columns
#X_train.info()
#sklearn的学习器大多之一稀疏数据输入，模型训练会快很多
#查看一个学习器是否支持稀疏数据，可以看fit函数是否支持：X: {array-like, sparse matrix}.
#可自行用timeit比较稠密数据和稀疏数据的训练时间
from scipy.sparse import csr_matrix
X_train = csr_matrix(X_train)
print(X_train)
```

```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
  (0, 0)        1
  (0, 10)       1
  (0, 16)       2
  (0, 21)       1
  (0, 23)       4
  (0, 24)       1
  (0, 25)       1
  (0, 28)       2
  (0, 34)       1
  (0, 39)       1
```

(0, 39) 1
(0, 41) 5
(0, 47) 2
(0, 53) 1
(0, 56) 2
(0, 59) 11
(0, 61) 1
(0, 62) 1
(0, 64) 1
(0, 66) 7
(0, 70) 1
(0, 78) 2
(0, 79) 1
(0, 84) 1
(1, 7) 1
(1, 17) 2
: :
(61877, 12) 3
(61877, 14) 1
(61877, 15) 1
(61877, 16) 1
(61877, 17) 1
(61877, 21) 3
(61877, 23) 2
(61877, 24) 1
(61877, 28) 9
(61877, 35) 2
(61877, 40) 3
(61877, 41) 1
(61877, 47) 1
(61877, 52) 2
(61877, 53) 5
(61877, 54) 1
(61877, 56) 1
(61877, 59) 3
(61877, 61) 4

```
(61877, 66)    10
(61877, 67)    2
(61877, 73)    3
(61877, 74)    1
(61877, 82)    1
(61877, 91)    2
```

In [4]:
```
from sklearn.model_selection import StratifiedKFold  #交叉验证+分层抽样
kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=3)
```

In [5]:
```
MAX_ROUNDS = 1000

#直接调用xgboost内嵌的交叉验证（cv），可对连续的n_estimators参数进行快速交叉验证
#而GridSearchCV只能对有限个参数进行交叉验证
def get_n_estimators(params, X_train , y_train , early_stopping_rounds=10):
    xgb_params = params.copy()

    #准备数据
    xgb_train = xgb.DMatrix(X_train, label = y_train)

    #模型训练/超参数调优
    cvresult = xgb.cv(xgb_params, xgb_train, num_boost_round=MAX_ROUNDS, nfold=3,
             metrics='mlogloss', early_stopping_rounds=early_stopping_rounds,seed=3)
    cvresult.to_csv('1_nestimators.csv', index_label = 'n_estimators')

    #最佳参数n_estimators
    n_estimators = cvresult.shape[0]


    print('best n_estimators:' , n_estimators)
    print('best cv score:' , cvresult['test-mlogloss-mean'][n_estimators-1])

    return n_estimators
```

In [6]:
```
#初始参数
params = {'learning_rate': 0.1,
          #'n_estimators': 1000
          'min_child_weight': 1,
          'max_depth': 5,
          'subsample': 0.7,
          'colsample_bytree': 0.7,
          'objective': 'multi:softprob',
          'num_class':9,
          'n_jobs': 4
         }
#objective [默认= reg: squarederror]  multi:softmax：设置XGBoost以使用softmax目标进行多类分类，还需要设置num_class（类数）
#multi:softprob：与softmax相同，但输出向量，可以进一步将其整形为矩阵。结果包含属于每个类别的每个数据点的预测概率。ndata * nclass

n_estimators_1 = get_n_estimators(params , X_train , y_train)
```

```
best n_estimators: 557
best cv score: 0.481484536342689
```

In [7]:
```
#max_depth 建议3-10,  min_child_weight=1 / sqrt(ratio_rare_event) =5.5
max_depth = range(5,10,2)
min_child_weight = range(1,6,2)
tuned_params = dict(max_depth=max_depth, min_child_weight=min_child_weight)
```

In [8]:
```
params = {'learning_rate': 0.1,
          'n_estimators': 636,   #第一轮参数调整得到的n_estimators最优值
          #'min_child_weight': 1,
          #'max_depth': 5,
          'subsample': 0.7,
          'colsample_bytree': 0.7,
          #'tree_method':'hist',
          #'max_bin':127,
          'objective': 'multi:softprob',
          'nthread': 4
         }

xgb_g = XGBClassifier(silent=False,  **params) #http://www.3dwindy.com/article/339803
```

In [9]:
```
grid_search = GridSearchCV(xgb_g, param_grid = tuned_params, scoring='neg_log_loss',n_jobs=4, cv=kfold,verbose=5, refit = False)
grid_search.fit(X_train , y_train)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

Out[9]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=3, shuffle=True),
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.7,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     gpu_id=None, grow_policy=None,
                                     importance_type=...
                                     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=636, n_jobs=None, nthread=4,
                                     num_parallel_tree=None,
                                     objective='multi:softprob', ...),
             n_jobs=4,
             param_grid={'max_depth': range(5, 10, 2),
                         'min_child_weight': range(1, 6, 2)},
             refit=False, scoring='neg_log_loss', verbose=5)
```

In [10]:
```
# summarize results
print("Best: %f using %s" % (grid_search.best_score_, grid_search.best_params_))
test_means = grid_search.cv_results_[ 'mean_test_score' ]
#test_stds = grid_search.cv_results_[ 'std_test_score' ]
#train_means = grid_search.cv_results_[ 'mean_train_score' ]
#train_stds = grid_search.cv_results_[ 'std_train_score' ]

pd.DataFrame(grid_search.cv_results_).to_csv('maxdepth_min_child_weights_1.csv')

# plot results
test_scores = np.array(test_means).reshape(len(max_depth), len(min_child_weight))
#train_scores = np.array(train_means).reshape(len(max_depth), len(min_child_weight))

for i, value in enumerate(max_depth):
    plt.plot(min_child_weight, -test_scores[i], label= 'test_max_depth:'    + str(value))

plt.legend()
plt.xlabel( 'min_child_weight' )
plt.ylabel( 'Log Loss' )
plt.savefig('max_depth_and_min_child_weght_1.png' )
```

```
Best: -0.481058 using {'max_depth': 5, 'min_child_weight': 1}
```