In [99]:

```
#Import libraries:
import pandas as pd
import numpy as np
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn import model_selection, metrics   #Additional scklearn functio
from sklearn.model_selection import GridSearchCV   #Performing grid search
import matplotlib.pylab as plt
%matplotlib inline
```

Load Data

In [100]:

```
train = pd.read_json('./../Data/train.json')
```

In [101]:

```
train.shape
```

Out[101]:

```
(49352, 15)
```

In [102]:

```
def calculate_length(df):
    df["num_photos"] = df["photos"].apply(len)
    df["num_features"] = df["features"].apply(len)
    df["num_description_words"] = df["description"].apply(lambda x: len(x.st
    return(df)
def calculate_date(df):
    df["created"] = pd.to_datetime(df["created"])
    df["created_year"] = df["created"].dt.year
    df["created_month"] = df["created"].dt.month
    df["created_day"] = df["created"].dt.day
    df["created_hour"] = df["created"].dt.hour
    return(df)
```

Add some additional features

In [103]:

```
train = calculate_length(train)
train = calculate_date(train)
train.shape
```

Out[103]:

```
(49352, 22)
```

Label encode certain categorical features

In [104]:

```
train.dtypes.unique()
```

Out[104]:

```
array([dtype('float64'), dtype('int64'), dtype('O'), dtype('<M8[ns]')], d
type=object)
```

In [105]:

```
features = [x for x in train.select_dtypes(include=['int64', 'float64']).col
```

```
In [106]: features
```

```
Out[106]: [u'bathrooms',
           u'bedrooms',
           u'latitude',
           u'longitude',
           u'price',
           'num_photos',
           'num_features',
           'num_description_words',
           'created_year',
           'created_month',
           'created_day',
           'created_hour']
```

```
In [107]: train.select_dtypes(include=['object']).columns
```

```
Out[107]: Index([u'building_id', u'description', u'display_address', u'features',
                 u'interest_level', u'manager_id', u'photos', u'street_address'],
                dtype='object')
```

```
In [108]: from sklearn import preprocessing
```

Label encode some categorical variables (different from one-hot encoding!!)

```
In [109]: def LabelEncoder(df, columns, features, append=False):
              for cols in columns:
                  label = preprocessing.LabelEncoder()
                  label.fit(df[cols].values)
                  df[cols] = label.transform(df[cols].values)
                  if append:
                      features.append(cols)
              return df, features
```

```
In [110]: categorical = ["display_address", "manager_id", "building_id", "street_addre
          train, features = LabelEncoder(train, categorical, features, True)
```

```
In [111]: train.building_id.head(5)
```

```
Out[111]: 10        2431
          10000     5862
          100004    5806
          100007    1201
          100013       0
          Name: building_id, dtype: int64
```

Create tf-idf matrix from text features

```
In [112]: train['features'].values

Out[112]: array([[],
                 [u'Doorman', u'Elevator', u'Fitness Center', u'Cats Allowed', u'Do
          gs Allowed'],
                 [u'Laundry In Building', u'Dishwasher', u'Hardwood Floors', u'Pets
          Allowed Case by Case'],
                 ...,
                 [u'Doorman', u'Elevator', u'Pre-War', u'Dogs Allowed', u'Cats Allo
          wed'],
                 [u'Doorman', u'Elevator', u'Pre-War', u'Dogs Allowed', u'Cats Allo
          wed'],
                 [u'Hardwood Floors']], dtype=object)
```

```
In [113]: #train['features'] = train["features"].apply(lambda x: " ".join(["_".join(i
          train['features'] = train['features'].apply(lambda x: ",".join(x))
```

```
In [117]: train['features'].values

Out[117]: array(['', u'Doorman,Elevator,Fitness Center,Cats Allowed,Dogs Allowed',
                 u'Laundry In Building,Dishwasher,Hardwood Floors,Pets Allowed Case
          by Case',
                 ..., u'Doorman,Elevator,Pre-War,Dogs Allowed,Cats Allowed',
                 u'Doorman,Elevator,Pre-War,Dogs Allowed,Cats Allowed',
                 u'Hardwood Floors'], dtype=object)
```

```
 In [76]: from sklearn.feature_extraction import text
```

```
In [118]: tfidf = text.CountVectorizer(stop_words='english', max_features=200, ngram_r
          tr_sparse = tfidf.fit_transform(train["features"])
```

```
In [126]: tr_sparse.shape, train[features].shape

Out[126]: ((49352, 200), (49352, 16))
```

```
In [127]: from scipy import sparse
```

```
In [136]: train_X = sparse.hstack([train[features], tr_sparse]).tocsr()
          target_num_map = {'high':0, 'medium':1, 'low':2}
          train_y = np.array(train['interest_level'].apply(lambda x: target_num_map[x]
          train_X.shape, train_y.shape

Out[136]: ((49352, 216), (49352,))
```

```
In [155]: np.set_printoptions(threshold=100)
          train_X.toarray()[0]

Out[155]: array([  1.5   ,   3.    ,  40.7145, ...,   0.    ,   0.    ,   0.    ])
```

In [152]:

```
train[features].head(3)
```

Out[152]:

	bathrooms	bedrooms	latitude	longitude	price	num_photos	num_features	num_
10	1.5	3	40.7145	-73.9425	3000	5	0	94
10000	1.0	2	40.7947	-73.9667	5465	11	5	1
100004	1.0	1	40.7388	-74.0018	2850	8	4	93

Train model

In [192]:

```
def xgbfit(model, dtrain, output, useTrainCV=True, cv_folds=5, early_stoppi

    if useTrainCV:
        xgb_param = model.get_xgb_params()
        xgb_param['num_class'] = 3

        xgtrain = xgb.DMatrix(dtrain, label=output)
        cvresult = xgb.cv(xgb_param, xgtrain, num_boost_round=model.get_para
                          metrics='mlogloss', early_stopping_rounds=early_st
        model.set_params(n_estimators=cvresult.shape[0])

    #Fit the model algorithm on the data
    model.fit(dtrain, output, eval_metric='mlogloss')

    #Predict training set:
    dtrain_predictions = model.predict(dtrain)

    #Print model report:
    print "\nModel Report"
    #print "R-Square: %.3f" % metrics.r2_score(output, dtrain_predictions)
    print "Log Loss : %.3f" % np.sqrt(metrics.log_loss(output, dtrain_predic
    print "Optimal CV Score:"
    print(cvresult.iloc[len(cvresult)-1,:])
    print "Optimal iteration: %d" %(len(cvresult)-1)
    #print "Cross Validation Result: "
    #print(cvresult)

    plt.figure()
    cvresult.loc[:,["test-mlogloss", "train-mlogloss"]].plot()
    return (len(cvresult))
```

```
In [193]: features
```

```
Out[193]: [u'bathrooms',
           u'bedrooms',
           u'latitude',
           u'longitude',
           u'price',
           'num_photos',
           'num_features',
           'num_description_words',
           'created_year',
           'created_month',
           'created_day',
           'created_hour',
           'display_address',
           'manager_id',
           'building_id',
           'street_address']
```

```
In [195]: xgb1 = XGBClassifier(
           learning_rate =0.1,
           n_estimators=1000,
           max_depth=6,
           min_child_weight=1,
           #gamma=0,
           subsample=0.7,
           colsample_bytree=0.7,
           objective='multi:softprob',
           nthread=4,
           scale_pos_weight=1,
           seed=189)
          n_estimators = xgbfit(xgb1, train_X, train_y)
```

```
Model Report
```

In [231]:

```
def runXGB(train_X, train_y, test_X, test_y=None, feature_names=None, seed_v
    param = {}
    param['objective'] = 'multi:softprob'
    param['eta'] = 0.1
    param['max_depth'] = 6
    param['silent'] = True
    param['num_class'] = 3
    param['eval_metric'] = "mlogloss"
    param['min_child_weight'] = 1
    param['subsample'] = 0.7
    param['colsample_bytree'] = 0.7
    param['seed'] = seed_val
    num_rounds = num_rounds

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)

    if test_y is not None:
        xgtest = xgb.DMatrix(test_X, label=test_y)
        watchlist = [ (xgtrain,'train'), (xgtest, 'test') ]
        model = xgb.train(plst, xgtrain, num_rounds, watchlist, early_stoppi
    else:
        xgtest = xgb.DMatrix(test_X)
        model = xgb.train(plst, xgtrain, num_rounds)

    pred_test_y = model.predict(xgtest)
    return pred_test_y, model
```

In [234]:

```
cv_scores = []
kf = model_selection.KFold(n_splits=5, shuffle=True, random_state=9594)
for dev_index, val_index in kf.split(range(train_X.shape[0])):
    # This leads to 5 iterations for 5 splits.
    # dev_index has 80% of the data, val_index has 20% since dev_index takes
    dev_X, val_X = train_X[dev_index,:], train_X[val_index,:] #training and
    dev_y, val_y = train_y[dev_index], train_y[val_index] #training and vali
    preds, model = runXGB(dev_X, dev_y, val_X, val_y, num_rounds=10)
    cv_scores.append(metrics.log_loss(val_y, preds))
print("cv_scores is: \n")
print(cv_scores)
```

```
[0]     train-mlogloss:1.04231  test-mlogloss:1.04335
Multiple eval metrics have been passed: 'test-mlogloss' will be used for
 early stopping.

Will train until test-mlogloss hasn't improved in 30 rounds.
[1]     train-mlogloss:0.988808 test-mlogloss:0.991342
[2]     train-mlogloss:0.9447   test-mlogloss:0.947949
[3]     train-mlogloss:0.905032 test-mlogloss:0.909496
[4]     train-mlogloss:0.873738 test-mlogloss:0.879315
[5]     train-mlogloss:0.8452   test-mlogloss:0.851965
[6]     train-mlogloss:0.821483 test-mlogloss:0.828979
[7]     train-mlogloss:0.798081 test-mlogloss:0.806763
[8]     train-mlogloss:0.779337 test-mlogloss:0.789054
[9]     train-mlogloss:0.763191 test-mlogloss:0.773825
[0]     train-mlogloss:1.04218  test-mlogloss:1.04306
Multiple eval metrics have been passed: 'test-mlogloss' will be used for
 early stopping.

Will train until test-mlogloss hasn't improved in 30 rounds.
[1]     train-mlogloss:0.98889  test-mlogloss:0.990589
[2]     train-mlogloss:0.944869 test-mlogloss:0.947613
[3]     train-mlogloss:0.905808 test-mlogloss:0.909032
[4]     train-mlogloss:0.874403 test-mlogloss:0.878286
[5]     train-mlogloss:0.846063 test-mlogloss:0.85052
[6]     train-mlogloss:0.82223  test-mlogloss:0.82755
[7]     train-mlogloss:0.798562 test-mlogloss:0.804737
[8]     train-mlogloss:0.780015 test-mlogloss:0.78687
[9]     train-mlogloss:0.764084 test-mlogloss:0.771618
[0]     train-mlogloss:1.03727  test-mlogloss:1.03705
Multiple eval metrics have been passed: 'test-mlogloss' will be used for
 early stopping.

Will train until test-mlogloss hasn't improved in 30 rounds.
[1]     train-mlogloss:0.984484 test-mlogloss:0.984005
[2]     train-mlogloss:0.944027 test-mlogloss:0.943723
[3]     train-mlogloss:0.905473 test-mlogloss:0.905309
[4]     train-mlogloss:0.872217 test-mlogloss:0.872441
[5]     train-mlogloss:0.84519  test-mlogloss:0.845638
[6]     train-mlogloss:0.821019 test-mlogloss:0.821717
[7]     train-mlogloss:0.797477 test-mlogloss:0.798822
[8]     train-mlogloss:0.777528 test-mlogloss:0.779398
[9]     train-mlogloss:0.758671 test-mlogloss:0.761119
[0]     train-mlogloss:1.03672  test-mlogloss:1.03845
Multiple eval metrics have been passed: 'test-mlogloss' will be used for
 early stopping.
```

```
Will train until test-mlogloss hasn't improved in 30 rounds.
[1]	train-mlogloss:0.984133 test-mlogloss:0.986989
[2]	train-mlogloss:0.943447 test-mlogloss:0.94748
[3]	train-mlogloss:0.904404 test-mlogloss:0.909973
[4]	train-mlogloss:0.871004 test-mlogloss:0.877333
[5]	train-mlogloss:0.843771 test-mlogloss:0.850999
[6]	train-mlogloss:0.819389 test-mlogloss:0.827512
[7]	train-mlogloss:0.795706 test-mlogloss:0.804846
[8]	train-mlogloss:0.77611  test-mlogloss:0.785978
[9]	train-mlogloss:0.757212 test-mlogloss:0.767794
[0]	train-mlogloss:1.03689  test-mlogloss:1.03828
Multiple eval metrics have been passed: 'test-mlogloss' will be used for
 early stopping.

Will train until test-mlogloss hasn't improved in 30 rounds.
[1]	train-mlogloss:0.983946 test-mlogloss:0.986643
[2]	train-mlogloss:0.943248 test-mlogloss:0.946835
[3]	train-mlogloss:0.904048 test-mlogloss:0.909118
[4]	train-mlogloss:0.870739 test-mlogloss:0.877103
[5]	train-mlogloss:0.843342 test-mlogloss:0.850627
[6]	train-mlogloss:0.818933 test-mlogloss:0.82725
[7]	train-mlogloss:0.795661 test-mlogloss:0.805011
[8]	train-mlogloss:0.776003 test-mlogloss:0.78638
[9]	train-mlogloss:0.756996 test-mlogloss:0.768503
cv_scores is:

[0.77382506840606557, 0.77161818340597343, 0.76111855259299155, 0.7677942
3060448582, 0.7685032073548016]
```