This is version 2 where I used the GradientBoostingRegressor (gbr) with default parameters and I added a few new features in addition to Day, Month and Season (from the previous version) such as the tax per square foot etc. I removed some of the features with large number of missing values and changed the dtypes to keep the properties frame under 0.7GB and the number of features at 64. On its own this gave LB 0.06445 but when combined (simply averaged) with some of the other public kernels that have similar score on their own the result was 0.06426 (top 5%). Suprisingly both lgbm and xgb gave worse score with the extra features. The rest of the notebook compares the feature importances between gbr and lgbm and xgb and looks at the impact of each feature. The test set score seems to converge at about 30 features but keeping all 64 gave a better LB score.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import  GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import MiniBatchKMeans
import lightgbm as lgb
import xgboost as xgb
import datetime as dt
import gc

print('loading files...')
prop = pd.read_csv('properties_2016.csv',low_memory=False)
prop.rename(columns={'parcelid': 'ParcelId'}, inplace=True)   # make it the same as
train = pd.read_csv('train_2016_v2.csv')
train = train[train['logerror'] < 0]
train.rename(columns={'parcelid': 'ParcelId'},inplace=True)
#sample = pd.read_csv('sample_submission.csv')
#print(train.shape, prop.shape, sample.shape)
```

```
loading files...
```

In [2]:

```
print('preprocessing, fillna, outliters, dtypes ...')

prop['longitude']=prop['longitude'].fillna(prop['longitude'].median()) / 1e6   #  cc
prop['latitude'].fillna(prop['latitude'].median()) / 1e6
prop['censustractandblock'].fillna(prop['censustractandblock'].median()) / 1e12
train = train[train['logerror'] <  train['logerror'].quantile(0.9975)]  # exclude 0.
train = train[train['logerror'] >  train['logerror'].quantile(0.0025)]

print('qualitative ...')
qualitative = [f for f in prop.columns if prop.dtypes[f] == object]
prop[qualitative] = prop[qualitative].fillna('Missing')
for c in qualitative:  prop[c] = LabelEncoder().fit(list(prop[c].values)).transform(

print('smallval ...')
smallval = [f for f in prop.columns if np.abs(prop[f].max())<100]
prop[smallval] = prop[smallval].fillna('Missing')
for c in smallval:  prop[c] = LabelEncoder().fit(list(prop[c].values)).transform(lis

print('other ...')
other=['regionidcounty','fips','propertycountylandusecode','propertyzoningdesc','pro
prop[other] = prop[other].fillna('Missing')
for c in other:  prop[c] = LabelEncoder().fit(list(prop[c].values)).transform(list(p

randomyears=pd.Series(np.random.choice(prop['yearbuilt'].dropna().values,len(prop)))
prop['yearbuilt']=prop['yearbuilt'].fillna(randomyears).astype(int)
med_yr=prop['yearbuilt'].quantile(0.5)
prop['New']=prop['yearbuilt'].apply(lambda x: 1 if x > med_yr else 0).astype(np.int8

randomyears=pd.Series(np.random.choice(prop['assessmentyear'].dropna().values,len(pr
prop['assessmentyear']=prop['assessmentyear'].fillna(randomyears).astype(int)

prop['unitcnt'] = prop['unitcnt'].fillna(1).astype(int)

feat_to_drop=[ 'finishedsquarefeet50', 'finishedfloor1squarefeet', 'finishedsquarefe
prop.drop(feat_to_drop,axis=1,inplace=True)   # drop because too many missing values
prop['lotsizesquarefeet'].fillna(prop['lotsizesquarefeet'].quantile(0.001),inplace=T
prop['finishedsquarefeet12'].fillna(prop['finishedsquarefeet12'].quantile(0.001),inp
prop['calculatedfinishedsquarefeet'].fillna(prop['finishedsquarefeet12'],inplace=Tru
prop['taxamount'].fillna(prop['taxamount'].quantile(0.001),inplace=True)
prop['landtaxvaluedollarcnt'].fillna(prop['landtaxvaluedollarcnt'].quantile(0.001),i
prop.fillna(0,inplace=True)

print('quantitative ...')
quantitative = [f for f in prop.columns if prop.dtypes[f] == np.float64]
prop[quantitative] = prop[quantitative].astype(np.float32)

cfeatures = list(prop.select_dtypes(include = ['int64', 'int32', 'uint8', 'int8']).c
for c in qualitative:  prop[c] = LabelEncoder().fit(list(prop[c].values)).transform(

# some quantitative features have a limited number of values (eg ZIP code)
for c in ['rawcensustractandblock',  'regionidcity',  'regionidneighborhood',  'regi
    prop[c] = LabelEncoder().fit(list(prop[c].values)).transform(list(prop[c].values

# other quantitative features were probably transformed when Zillow first calculate
for c in ['calculatedfinishedsquarefeet', 'finishedsquarefeet12', 'lotsizesquarefeet
    'structuretaxvaluedollarcnt',  'taxvaluedollarcnt',  'landtaxvaluedollarcnt',  '
    prop[c] = np.log1p(prop[c].values)

gc.collect()
```

```
preprocessing, fillna, outliters, dtypes ...
qualitative ...
smallval ...
other ...
quantitative ...
```

Out[2]:

```
0
```

In [3]:

```
print('create new features and the final dataframes frames ...')

#replace latitudes and longitudes with 500 clusters  (similar to ZIP codes)
coords = np.vstack(prop[['latitude', 'longitude']].values)
sample_ind = np.random.permutation(len(coords))[:1000000]
kmeans = MiniBatchKMeans(n_clusters=500, batch_size=100000).fit(coords[sample_ind])
prop['Cluster'] = kmeans.predict(prop[['latitude', 'longitude']])

prop['Living_area_prop'] = prop['calculatedfinishedsquarefeet'] / prop['lotsizesquar
prop['Value_ratio'] = prop['taxvaluedollarcnt'] / prop['taxamount']
prop['Value_prop'] = prop['structuretaxvaluedollarcnt'] / prop['landtaxvaluedollarcn
prop['Taxpersqrtfoot']=prop['finishedsquarefeet12']/prop['taxamount']

train['transactiondate'] = pd.to_datetime(train.transactiondate)
train['Month'] = train['transactiondate'].dt.month.astype(np.int8)
train['Day'] = train['transactiondate'].dt.day.astype(np.int8)
train['Season'] = train['Month'].apply(lambda x: 1 if x in [1,2,9,10,11,12] else 0).

month_err=(train.groupby('Month').aggregate({'logerror': lambda x: np.mean(x)})- tra
train['Meanerror']=train['Month'].apply(lambda x: month_err[x-1]).astype(np.float)

train['abserror']=train['logerror'].abs()
month_abs_err=(train.groupby('Month').aggregate({'abserror': lambda x: np.mean(x)})-
train['Meanabserror']=train['Month'].apply(lambda x: month_abs_err[x-1]).astype(np.f
train.drop(['abserror'], axis=1,inplace=True)

X = train.merge(prop, how='left', on='ParcelId')
y = X['logerror']
X.drop(['ParcelId', 'logerror', 'transactiondate'], axis=1,inplace=True)
features=list(X.columns)

print(X.shape, y.shape)
gc.collect()
```

```
create new features and the final dataframes frames ...

/Users/akhileshpothuri/opt/anaconda3/lib/python3.9/site-packages/sklea
rn/base.py:443: UserWarning: X has feature names, but MiniBatchKMeans
was fitted without feature names
  warnings.warn(
/var/folders/94/qbcd9c954nnf3wh_kk6kyj380000gn/T/ipykernel_6504/801038
739.py:20: DeprecationWarning: `np.float` is a deprecated alias for th
e builtin `float`. To silence this warning, use `float` by itself. Doi
ng this will not modify any behavior and is safe. If you specifically
wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://nump
y.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.or
g/devdocs/release/1.20.0-notes.html#deprecations)
  train['Meanerror']=train['Month'].apply(lambda x: month_err[x-1]).as
type(np.float)
/var/folders/94/qbcd9c954nnf3wh_kk6kyj380000gn/T/ipykernel_6504/801038
739.py:24: DeprecationWarning: `np.float` is a deprecated alias for th
e builtin `float`. To silence this warning, use `float` by itself. Doi
ng this will not modify any behavior and is safe. If you specifically
 wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://nump
y.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.or
g/devdocs/release/1.20.0-notes.html#deprecations)
```

```
  train['Meanabserror']=train['Month'].apply(lambda x: month_abs_err[x
-1]).astype(np.float)
```

```
(38774, 64) (38774,)
```

Out[3]:

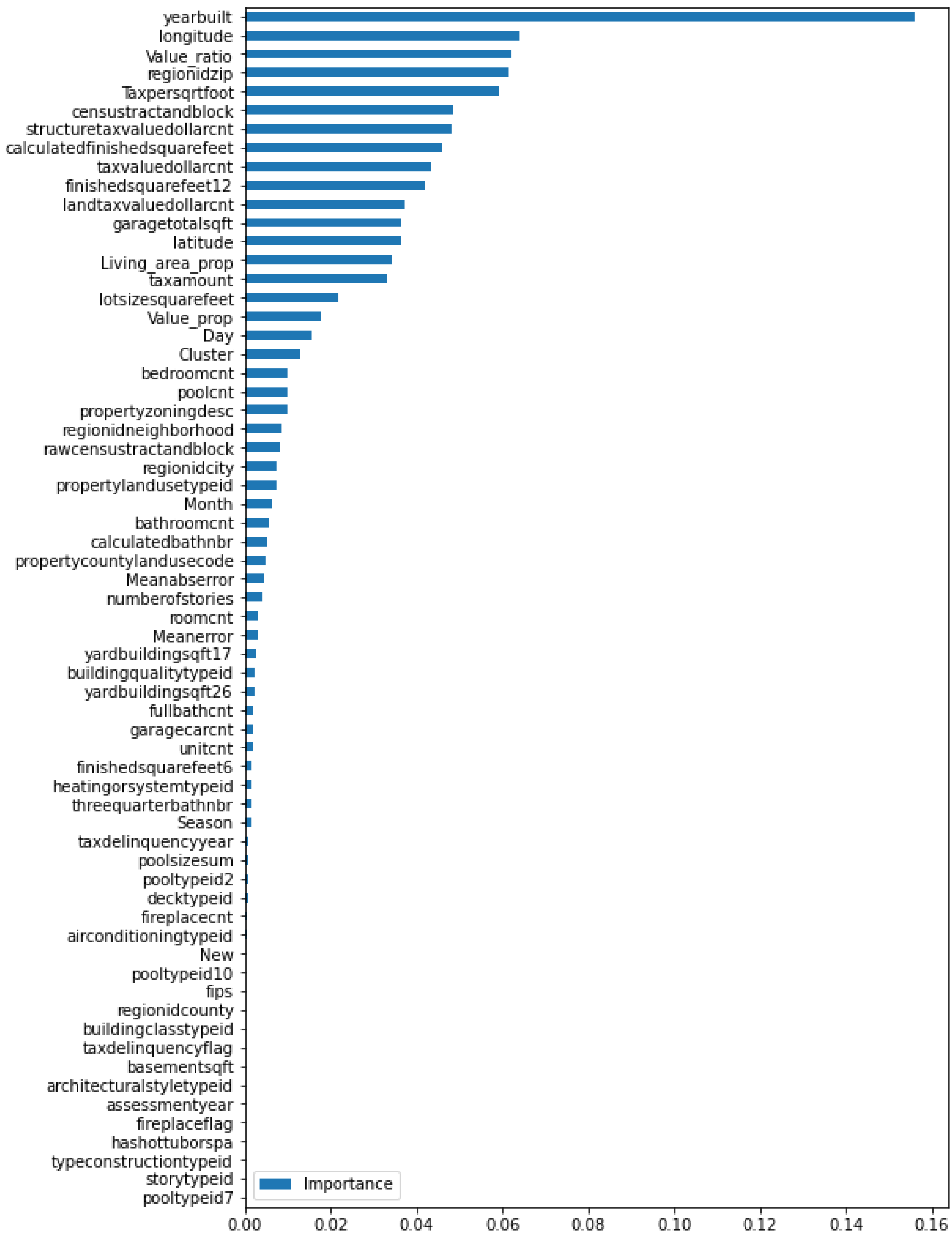
```
18
```

In [4]:

```
print(' Training GB ...')
X_train=X
y_train=y
n_estimators=800
clf = GradientBoostingRegressor(loss='lad',   n_estimators=n_estimators,  verbose=1)
clf.fit(X_train, y_train)
print('MAE train  {:.4f}'.format(np.mean(np.abs(y_train-clf.predict(X_train)) )))
gc.collect()

submit=False     # change to create the submission file
features=X_train.columns
if submit:
    print('predict and submit ...')
    X_test = (sample.merge(prop, on='ParcelId', how='left')).loc[:,features]

    if 'Season' in features: X_test['Season']=np.int8(1)
    if 'Day' in features: X_test['Day']=np.int8(15)

    for month in [10, 11, 12]:
        print('month ',month)
        if 'Month' in features: X_test['Month']=np.int8(month)
        if 'Meanerror' in features: X_test['Meanerror']=np.float(month_err[month-1])
        if 'Meanabserror' in features: X_test['Meanabserror']=np.float(month_abs_err
        sample['2016' + str(month)] = clf.predict(X_test)
        print(' MAE {}  {:.4f}'.format(month,np.mean(np.abs(sample['2016' + str(mont

    sample.to_csv('submission_GBR6445.csv', index = False, float_format = '%.5f')

FeatImp=pd.DataFrame(clf.feature_importances_, index=X_train.columns, columns=['Impo
FeatImp=FeatImp.sort_values('Importance')
FeatImp.plot(kind='barh', figsize=(8,14))
plt.show()
```

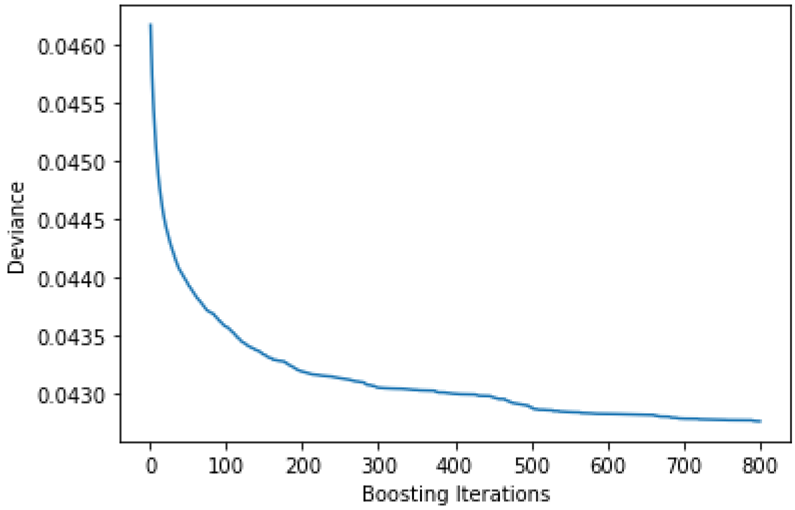
```
 Training GB ...
      Iter       Train Loss   Remaining Time 
         1           0.0462            2.53m

/Users/akhileshpothuri/opt/anaconda3/lib/python3.9/site-packages/sklea
rn/ensemble/_gb.py:293: FutureWarning: The loss 'lad' was deprecated i
n v1.0 and will be removed in version 1.2. Use 'absolute_error' which
is equivalent.
  warnings.warn(

         2           0.0459            2.53m
         3           0.0457            2.52m
         4           0.0456            2.52m
         5           0.0454            2.50m
         6           0.0453            2.50m
         7           0.0452            2.49m
         8           0.0451            2.49m
         9           0.0450            2.49m
        10           0.0450            2.49m
        20           0.0445            2.46m
        30           0.0442            2.43m
        40           0.0441            2.40m
        50           0.0439            2.37m
        60           0.0438            2.34m
        70           0.0438            2.31m
```

```
         80          0.0437           2.28m
         90          0.0436           2.25m
        100          0.0436           2.22m
        200          0.0432           1.91m
        300          0.0431           1.60m
        400          0.0430           1.28m
        500          0.0429          57.54s
        600          0.0428          38.38s
        700          0.0428          19.20s
        800          0.0428           0.00s
 MAE train  0.0428
```



In [5]:

```
# Plot training deviance - 600 iteration seems enough
plt.plot(np.arange(n_estimators)+1, clf.train_score_)
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
plt.show()
```



In [6]:

```
# starting with 10 most important features I gradually add more features to see thei
# there are 64 features but after the first 30 the gain is quite small
print('feature impact ...')

X_train, X_eval, y_train, y_eval = train_test_split(X,y, test_size=0.5, random_state
clf = GradientBoostingRegressor(loss='lad',   n_estimators=600,  verbose=0)
clf.fit(X_train, y_train)
FeatImp=pd.DataFrame(clf.feature_importances_, index=X_train.columns, columns=['Impo
FeatImp=FeatImp.sort_values('Importance', ascending = False)
print( FeatImp.iloc[0:10].index.values )
Errors_train = []
Errors_eval = []

istart=10
iend=len(FeatImp)+1
iend = 30  # remove if time is no constraint and let run to the end (ie 64)
for i in range(istart,iend):
    X_train_temp = X_train[FeatImp.iloc[0:i].index.values]
    X_eval_temp = X_eval[FeatImp.iloc[0:i].index.values]
    clf.fit(X_train_temp, y_train)
    Err_eval = np.mean(np.abs(y_eval-clf.predict(X_eval_temp) ) )
    Err_train=np.mean(np.abs(y_train-clf.predict(X_train_temp) ) )
    print('{:<30} train {:.3f} eval {:.3f} '.format(FeatImp.index[i-1],1000*Err_trai
    Errors_train = Errors_train+[Err_train]
    Errors_eval = Errors_eval+[Err_eval]

plt.figure(figsize=(20,10))
plt.plot(range(istart,iend),Errors_train,label='train')
plt.plot(range(istart,iend),Errors_eval,label='eval')
plt.xticks(range(istart,iend), features[10:],rotation=90)
plt.ylabel('Errors')
plt.legend()
plt.show()
```

```
-> 1720         raise ValueError(
   1721             "The number of FixedLocator locations"
   1722             f" ({len(locator.locs)}), usually from a call to"
   1723             " set_ticks, does not match"
   1724             f" the number of ticklabels ({len(ticklabels)}).")
   1725     tickd = {loc: lab for loc, lab in zip(locator.locs, tickla
bels)}
   1726     func = functools.partial(self._format_with_dict, tickd)

ValueError: The number of FixedLocator locations (20), usually from a
 call to set_ticks, does not match the number of ticklabels (54).
```

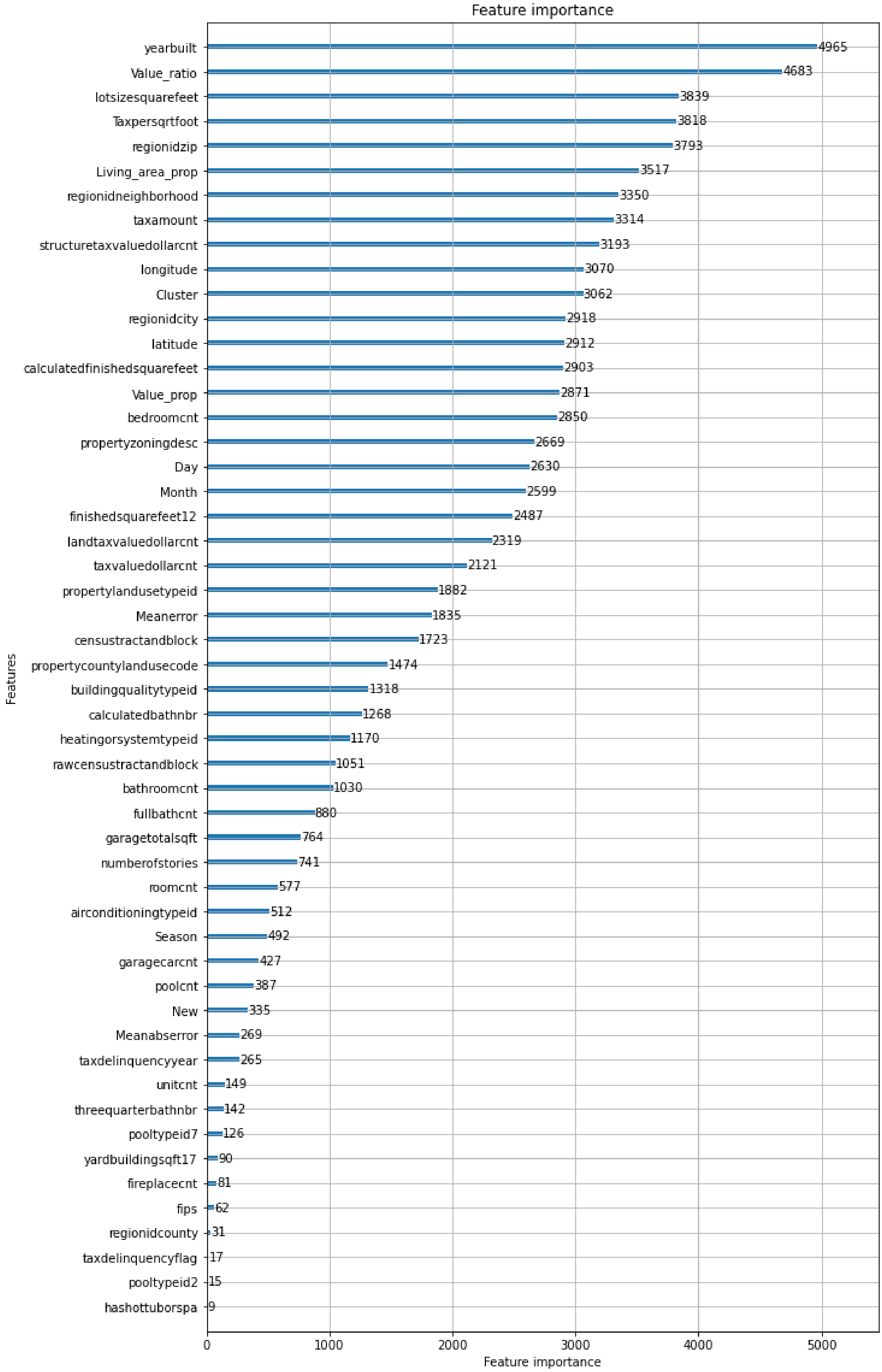
In [7]:

```
print('Training lgbm ...')
features=list(X.columns)
cfeatures = list(X.select_dtypes(include = ['int64', 'int32', 'uint8', 'int8']).colu

params = {'metric': 'mae', 'learning_rate' : 0.005, 'max_depth':10, 'max_bin':10,
         'feature_fraction': 0.95,'bagging_fraction':0.95,'bagging_freq':10,'min_dat

# using eval or not (set CV to True or False)
CV=False
if CV:

    X_train, X_eval, y_train, y_eval = train_test_split(X,y, test_size=0.5, random_s
    lgb_train = lgb.Dataset(X_train.values, y_train.values)
    lgb_eval = lgb.Dataset(X_eval.values, y_eval.values, reference = lgb_train)
    lgb_model = lgb.train(params, lgb_train, num_boost_round = 3000, valid_sets = lg
              feature_name=features, early_stopping_rounds=100, verbose_eval = 100)
    pred1 = lgb_model.predict(X_train.values, num_iteration = lgb_model.best_iterati
    pred2 = lgb_model.predict(X_eval.values, num_iteration = lgb_model.best_iteratio
    print(' MAE train  {:.4f}'.format(np.mean(np.abs(y_train.values-pred1) )))
    print(' MAE eval   {:.4f}'.format(np.mean(np.abs(y_eval.values-pred2) )))
    del lgb_train,pred1,lgb_eval, pred2

else:

    X_train=X
    y_train=y
    lgb_train = lgb.Dataset(X_train.values, y_train.values)
    lgb_model = lgb.train(params, lgb_train, num_boost_round = 3000, feature_name=fe
    pred1 = lgb_model.predict(X_train.values, num_iteration = lgb_model.best_iterati
    print(' MAE train  {:.4f}'.format(np.mean(np.abs(y_train.values-pred1) )))
    del lgb_train, pred1


lgb_model.save_model('model.txt')
#bst = lgb.Booster(model_file='model.txt')
gc.collect()
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:

-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

In [8]:

```
#check feature importance
lgb.plot_importance(lgb_model,  figsize=(10,20))
plt.show()
gc.collect()
```


Feature importance
Features
Feature importance
yearbuilt 4965
Value_ratio 4683
lotsizesquarefeet 3839
Taxpersqrtfoot 3818
regionidzip 3793
Living_area_prop 3517
regionidneighborhood 3350
taxamount 3314
structuretaxvaluedollarcnt 3193
longitude 3070
Cluster 3062
regionidcity 2918
latitude 2912
calculatedfinishedsquarefeet 2903
Value_prop 2871
bedroomcnt 2850
propertyzoningdesc 2669
Day 2630
Month 2599
finishedsquarefeet12 2487
landtaxvaluedollarcnt 2319
taxvaluedollarcnt 2121
propertylandusetypeid 1882
Meanerror 1835
censustractandblock 1723
propertycountylandusecode 1474
buildingqualitytypeid 1318
calculatedbathnbr 1268
heatingorsystemtypeid 1170
rawcensustractandblock 1051
bathroomcnt 1030
fullbathcnt 880
garagetotalsqft 764
numberofstories 741
roomcnt 577
airconditioningtypeid 512
Season 492
garagecarcnt 427
poolcnt 387
New 335
Meanabserror 269
taxdelinquencyyear 265
unitcnt 149
threequarterbathnbr 142
pooltypeid7 126
yardbuildingsqft17 90
fireplacecnt 81
fips 62
regionidcounty 31
taxdelinquencyflag 17
pooltypeid2 15
hashottuborspa 9
0
1000
2000
3000
4000
5000

Out[8]:

12814

In [9]:

```
# I do not know if there is a more pythonic way to get the lgbm feature importances
# could do is by parsing the model file
ind=[]
data=[]
with open('model.txt') as f: FI = list(f)[-100:-2]
FI=FI[FI.index('feature importances:\n') +1:]
for i in range(len(FI)):
    FI[i]=FI[i][:-1]
    ind=ind+[FI[i].split('=')[0]]
    data=data+[int(FI[i].split('=')[1])]
FeatImp=pd.DataFrame(data, index=ind, columns=['Importance'])
del f,ind,data
FeatImp.head()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
 last)
Input In [9], in <cell line: 6>()
      4 data=[]
      5 with open('model.txt') as f: FI = list(f)[-100:-2]
----> 6 FI=FI[FI.index('feature importances:\n') +1:]
      7 for i in range(len(FI)):
      8     FI[i]=FI[i][:-1]

ValueError: 'feature importances:\n' is not in list
```

Obviously the feature importances depend on the choice of the parameters but still the first attempt to compare gbr and lgbm leads to quite large differences and so it makes sense to combine the different methods. Finally, let us compare with xgb.

In [10]:

```
print('training xgboost ...')
X_train=X
y_train=y
y_mean = np.mean(y_train)
xgb_params = {'eta': 0.037, 'max_depth': 5, 'subsample': 0.80,  'eval_metric': 'mae'
              'lambda': 0.8,   'alpha': 0.4, 'base_score': y_mean, 'silent': 1 }
dtrain = xgb.DMatrix(X_train, y_train)
model = xgb.train(xgb_params, dtrain, num_boost_round=250)
pred1 = model.predict(dtrain)
print(' xgb MAE train  {:.4f}'.format(np.mean(np.abs(y_train.values-pred1) )))
del dtrain, pred1
gc.collect()
```

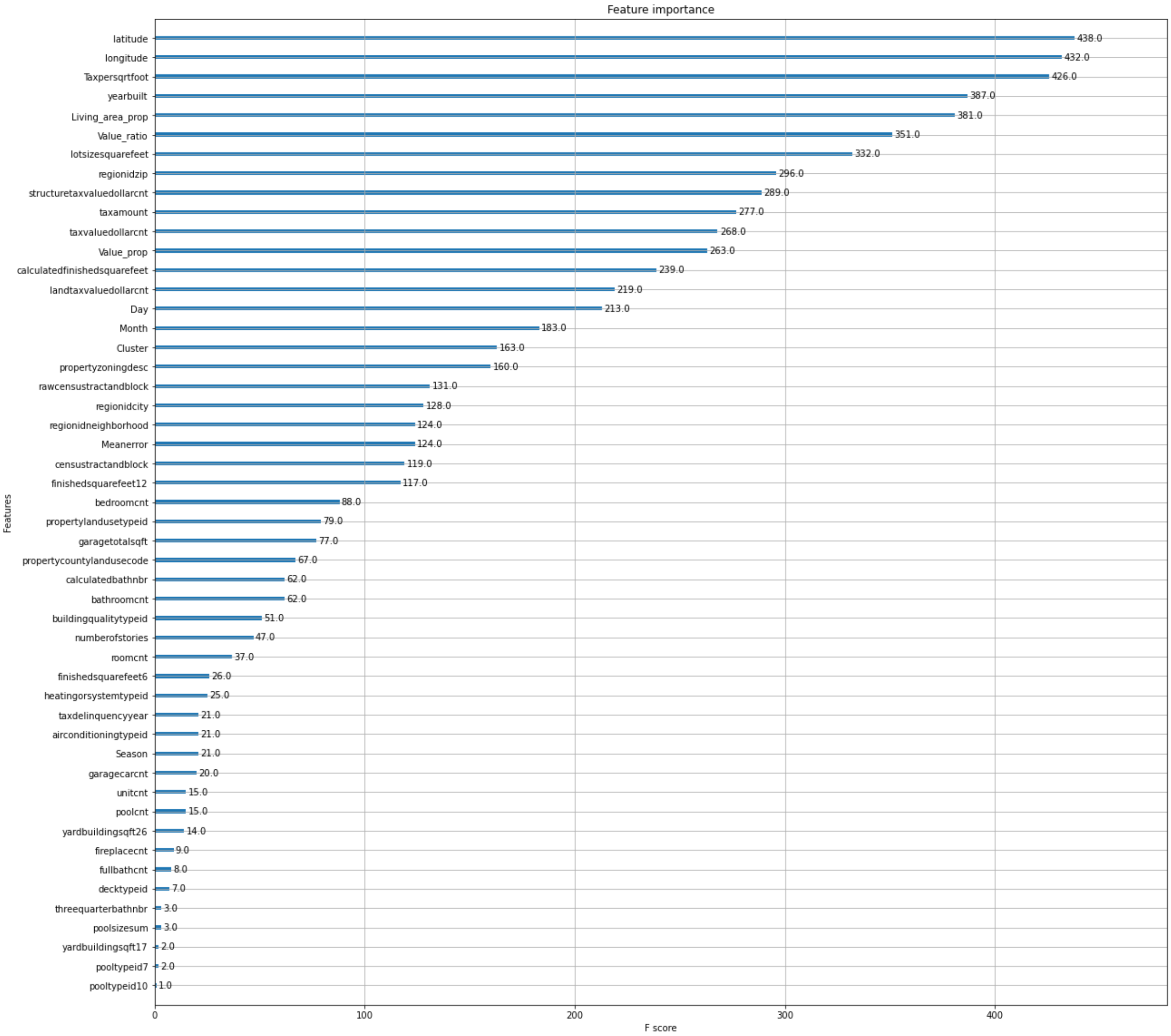
```
training xgboost ...
[11:43:00] WARNING: /Users/runner/work/xgboost/xgboost/python-package/
build/temp.macosx-10.9-x86_64-cpython-38/xgboost/src/learner.cc:767:
Parameters: { "silent" } are not used.

 xgb MAE train  0.0454
```

Out[10]:

```
4715
```

In [11]:

```
fig, ax = plt.subplots(figsize=(20, 20))
xgb.plot_importance(model, ax=ax)
plt.show()
gc.collect()
```



Out[11]:

0

Work in progress ...

Type *Markdown* and LaTeX: α^2