Benz - playground

June 21, 2017

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        sns.set(font_scale = 1.7)
        import matplotlib.pyplot as plt
        import sklearn
        %matplotlib inline
```

```
In [2]: from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "last_expr"
        # http://ipython.readthedocs.io/en/stable/config/options/terminal.html
        import warnings
        warnings.filterwarnings("ignore")
```

```
In [35]: train = pd.read_csv('data/train.csv')
         test = pd.read_csv('data/test.csv')
```

```
In [83]: import lightgbm as lgb
         import xgboost as xgb
```

```
In [84]: train_x = train.drop('y', axis = 1)
         train_y = train['y']
         test_x = test.copy()
```

1 Stack then average models

https://www.kaggle.com/hakeem/stacked-then-averaged-models-0-5697

```
In [85]: from sklearn.base import BaseEstimator,TransformerMixin, ClassifierMixin
         from sklearn.preprocessing import LabelEncoder
         from sklearn.decomposition import PCA, FastICA, TruncatedSVD
         from sklearn.random_projection import GaussianRandomProjection, SparseRand
         from sklearn.pipeline import make_pipeline, make_union
         from sklearn.linear_model import LassoLarsCV, ElasticNetCV
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.utils import check_array
         from sklearn.metrics import r2_score
```

In [86]:
```
# http://danielhnyk.cz/creating-your-own-estimator-scikit-learn/

class StackingEstimator(BaseEstimator, TransformerMixin):
    def __init__(self, estimator):
        self.estimator = estimator

    def fit(self, X, y = None, **fit_params):
        self.estimator.fit(X, y, **fit_params)
        return self
    def transform(self, X):
        X = check_array(X)
        X_transformed = np.copy(X)
        if issubclass(self.estimator.__class__, ClassifierMixin) and hasat
            X_transformed = np.hstack((self.estimator.predict_proba(X),X))
        X_transformed = np.hstack((np.reshape(self.estimator.predict(X),(-

        return X_transformed
```

In [87]:
```
for col in train.columns:
    if train[col].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(train[col].values) + list(test[col].values))
        train[col] = lbl.transform(list(train[col].values))
        test[col] = lbl.transform(list(test[col].values))
```

In [88]:
```
n_comp = 12

# tSVD
tsvd = TruncatedSVD(n_components=n_comp, random_state=420)
tsvd_results_train = tsvd.fit_transform(train.drop(["y"], axis=1))
tsvd_results_test = tsvd.transform(test)

# PCA
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(train.drop(["y"], axis=1))
pca2_results_test = pca.transform(test)

# ICA
ica = FastICA(n_components=n_comp, random_state=420)
ica2_results_train = ica.fit_transform(train.drop(["y"], axis=1))
ica2_results_test = ica.transform(test)

# GRP
grp = GaussianRandomProjection(n_components=n_comp, eps=0.1, random_state=
grp_results_train = grp.fit_transform(train.drop(["y"], axis=1))
grp_results_test = grp.transform(test)
```

```
        # SRP
        srp = SparseRandomProjection(n_components=n_comp, dense_output=True, rando
        srp_results_train = srp.fit_transform(train.drop(["y"], axis=1))
        srp_results_test = srp.transform(test)
```

In [89]:
```
        #save columns list before adding the decomposition components

        usable_columns = list(set(train.columns) - set(['y']))

        # Append decomposition components to datasets
        for i in range(1, n_comp + 1):
            train['pca_' + str(i)] = pca2_results_train[:, i - 1]
            test['pca_' + str(i)] = pca2_results_test[:, i - 1]

            train['ica_' + str(i)] = ica2_results_train[:, i - 1]
            test['ica_' + str(i)] = ica2_results_test[:, i - 1]

            train['tsvd_' + str(i)] = tsvd_results_train[:, i - 1]
            test['tsvd_' + str(i)] = tsvd_results_test[:, i - 1]

            train['grp_' + str(i)] = grp_results_train[:, i - 1]
            test['grp_' + str(i)] = grp_results_test[:, i - 1]

            train['srp_' + str(i)] = srp_results_train[:, i - 1]
            test['srp_' + str(i)] = srp_results_test[:, i - 1]
```

In [94]:
```
        y_mean = np.mean(y_train)
        id_test = test['ID'].values
        #finaltrainset and finaltestset are data to be used only the stacked model
        finaltrainset = train[usable_columns].values
        finaltestset = test[usable_columns].values
```

In [95]:
```
        import xgboost as xgb
        import lightgbm as lgb
```

In [247]:
```
        xgb_params = {
            'n_trees': 520,
            'eta': 0.0045,
            'max_depth': 4,
            'subsample': 0.93,
            'objective': 'reg:linear',
            'eval_metric': 'rmse',
            'base_score': y_mean, # base prediction = mean(target)
            'silent': 1
        }
        # NOTE: Make sure that the class is labeled 'class' in the data file

        dtrain = xgb.DMatrix(train_x, y_train)
```

```
            dtest = xgb.DMatrix(test_x)

            num_boost_rounds = 1250
            # train model
            model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=num
            y_pred = model.predict(dtest)
```

```
In [248]: sub = pd.DataFrame()
          sub['ID'] = id_test
          sub['y'] = y_pred
          sub.to_csv('Submission/only xgb 2' + '.csv', index = False)
```

```
In [96]: xgb_params = {'n_trees':1000,
                       'eta' : 0.003,
                     'max_depth' : 5,
                       'subsample' : 0.91,
                       'eval_metric':'rmse',
                       'base_score' : y_mean,
                       'silent' : 1
                      }
```

```
In [97]: dtrain = xgb.DMatrix(train_x, label = train_y)
```

```
In [98]: dtest = xgb.DMatrix(test)
```

```
In [99]: num_boost_rounds = 1250
```

```
In [100]: %%time
          model = xgb.train(xgb_params, dtrain, num_boost_round=num_boost_rounds)

CPU times: user 1min 16s, sys: 824 ms, total: 1min 16s
Wall time: 1min 20s
```

```
In [101]: y_pred = model.predict(dtest)
```

```
In [102]: r2_score(model.predict(dtrain), train_y)

Out[102]: 0.4229675336193659
```

1.1 XGB cross-validation

```
In [200]: from sklearn.model_selection import GridSearchCV
```

```
In [195]: xgb_params1 = {'n_estimators':2000,
                         'eta' : 0.001,
                       'max_depth' : 3,
                         'subsample' : 0.9,
                         'eval_metric':'rmse',
                         'silent' : 1,
```

```
                        'min_child_weight':1,
                        'colsample_bytree':0.7,
                        'gamma':0.1,
                        'reg_alpha':0,
                        'reg_lambda':0
                       }
```

```
In [196]: cv_folds = 10
          nrounds = 1000
```

```
In [197]: %%time
          cvresult = xgb.cv(xgb_params1, dtrain, num_boost_round=nrounds, nfold=cv_
                      metrics='rmse', early_stopping_rounds=20)
```

```
CPU times: user 4min 17s, sys: 3.62 s, total: 4min 21s
Wall time: 4min 43s
```

```
In [198]: cvresult.shape[0]
```

```
Out[198]: 1000
```

1.1.1 Tune max_depth and min_child_weight

tune 1

```
In [207]: %%time
          param_test1 = {
           'max_depth':range(3,10,2),
           'min_child_weight':range(1,6,2)
          }
          gsearch1 = GridSearchCV(estimator = xgb.XGBRegressor( learning_rate =0.1,
           min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8,
           objective= 'reg:linear', nthread=4, scale_pos_weight=1, seed=2017),
           param_grid = param_test1, scoring='neg_mean_squared_error',n_jobs=4,iid=
          gsearch1.fit(train_x,train_y)
```

```
CPU times: user 37.3 s, sys: 685 ms, total: 38 s
Wall time: 10min 19s
```

```
In [205]: gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
Out[205]: ([mean: -1.87053, std: 4.38716, params: {'min_child_weight': 1, 'max_dept
            mean: -0.84927, std: 2.24561, params: {'min_child_weight': 3, 'max_dept
            mean: -0.68878, std: 2.08043, params: {'min_child_weight': 5, 'max_dept
            mean: -2.13067, std: 5.21152, params: {'min_child_weight': 1, 'max_dept
            mean: -0.50769, std: 1.73564, params: {'min_child_weight': 3, 'max_dept
            mean: -0.74552, std: 2.28462, params: {'min_child_weight': 5, 'max_dept
            mean: -2.38524, std: 5.49466, params: {'min_child_weight': 1, 'max_dept
```

```
 mean: -0.67995, std: 2.09491, params: {'min_child_weight': 3, 'max_dept
 mean: -0.84350, std: 2.39400, params: {'min_child_weight': 5, 'max_dept
 mean: -2.70380, std: 6.27409, params: {'min_child_weight': 1, 'max_dept
 mean: -0.45347, std: 1.68096, params: {'min_child_weight': 3, 'max_dept
 mean: -1.85142, std: 4.53422, params: {'min_child_weight': 5, 'max_dept
{'max_depth': 9, 'min_child_weight': 3},
-0.45346553388605892)
```

```
In [208]: gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
Out[208]: ([mean: -439.74693, std: 668.63479, params: {'min_child_weight': 1, 'max_
 mean: -282.78193, std: 340.77344, params: {'min_child_weight': 3, 'max_
 mean: -259.69633, std: 315.89252, params: {'min_child_weight': 5, 'max_
 mean: -482.32536, std: 793.81750, params: {'min_child_weight': 1, 'max_
 mean: -232.56322, std: 263.17522, params: {'min_child_weight': 3, 'max_
 mean: -269.71983, std: 347.18711, params: {'min_child_weight': 5, 'max_
 mean: -519.58660, std: 837.53767, params: {'min_child_weight': 1, 'max_
 mean: -259.33132, std: 318.16289, params: {'min_child_weight': 3, 'max_
 mean: -284.25183, std: 363.84805, params: {'min_child_weight': 5, 'max_
 mean: -569.52391, std: 956.14624, params: {'min_child_weight': 1, 'max_
 mean: -225.02831, std: 254.95108, params: {'min_child_weight': 3, 'max_
 mean: -439.23657, std: 690.53919, params: {'min_child_weight': 5, 'max_
{'max_depth': 9, 'min_child_weight': 3},
-225.0283077699093)
```

tune 2

```
In [226]: %%time
          param_test1 = {
           'max_depth':range(3,8,1),
           'min_child_weight':range(1,6,2)
          }
          gsearch1 = GridSearchCV(estimator = xgb.XGBRegressor( learning_rate =0.00
           min_child_weight=1, gamma=0, subsample=0.93,
           objective= 'reg:linear', nthread=4, scale_pos_weight=1, seed=2017),
           param_grid = param_test1, scoring='r2',n_jobs=4,iid=False, cv=5)
          gsearch1.fit(train_x,train_y)
```

```
CPU times: user 22.6 s, sys: 680 ms, total: 23.3 s
Wall time: 13min 21s
```

```
In [227]: gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
Out[227]: ([mean: -0.13521, std: 0.29988, params: {'min_child_weight': 1, 'max_dept
 mean: -0.12079, std: 0.27557, params: {'min_child_weight': 3, 'max_dept
 mean: 0.03006, std: 0.14854, params: {'min_child_weight': 5, 'max_depth
 mean: -0.63993, std: 1.28969, params: {'min_child_weight': 1, 'max_dept
 mean: -0.21504, std: 0.45262, params: {'min_child_weight': 3, 'max_dept
```

```
 mean: -0.01474, std: 0.13486, params: {'min_child_weight': 5, 'max_dept
 mean: -1.87480, std: 3.75046, params: {'min_child_weight': 1, 'max_dept
 mean: -0.23705, std: 0.48683, params: {'min_child_weight': 3, 'max_dept
 mean: -0.02103, std: 0.13202, params: {'min_child_weight': 5, 'max_dept
 mean: -1.87813, std: 3.75125, params: {'min_child_weight': 1, 'max_dept
 mean: -0.24058, std: 0.48926, params: {'min_child_weight': 3, 'max_dept
 mean: -0.02436, std: 0.12883, params: {'min_child_weight': 5, 'max_dept
 mean: -1.88462, std: 3.75652, params: {'min_child_weight': 1, 'max_dept
 mean: -0.24476, std: 0.49283, params: {'min_child_weight': 3, 'max_dept
 mean: -0.02857, std: 0.12804, params: {'min_child_weight': 5, 'max_dept
{'max_depth': 3, 'min_child_weight': 5},
0.030060294110519092)
```

tune 3

In [233]:
```
from sklearn.metrics import make_scorer
```

In [240]:
```
%%time
# def MSE(y_true,y_pred):
#     mse = mean_squared_error(y_true, y_pred)

#     return mse

def R2(y_true,y_pred):
     r2 = r2_score(y_true, y_pred)

     return r2

# def two_score(y_true,y_pred):
#     MSE(y_true,y_pred) #set score here and not below if using MSE in Gr
#     score = R2(y_true,y_pred)
#     return score

# def two_scorer():
#     return make_scorer(two_score, greater_is_better=True) # change for


def scorer():
    return make_scorer(R2)

param_test1 = {
 'max_depth':range(3,8,1),
 'min_child_weight':range(1,6,2)
}
gsearch1 = GridSearchCV(estimator = xgb.XGBRegressor( learning_rate =0.00
 min_child_weight=1, gamma=0, subsample=0.93,
 objective= 'reg:linear', nthread=4, scale_pos_weight=1, seed=2017),
 param_grid = param_test1, scoring=scorer(),n_jobs=4,iid=False, cv=5)
gsearch1.fit(train_x,train_y)
```

```
CPU times: user 49.3 s, sys: 1.43 s, total: 50.7 s
Wall time: 37min 8s
```

```
In [241]: gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
Out[241]: ([mean: 0.19502, std: 0.70223, params: {'min_child_weight': 1, 'max_depth
            mean: -0.22679, std: 1.53276, params: {'min_child_weight': 3, 'max_dept
            mean: -0.14945, std: 1.40121, params: {'min_child_weight': 5, 'max_dept
            mean: -1.38027, std: 3.81728, params: {'min_child_weight': 1, 'max_dept
            mean: -0.33839, std: 1.70357, params: {'min_child_weight': 3, 'max_dept
            mean: -0.15771, std: 1.39376, params: {'min_child_weight': 5, 'max_dept
            mean: -5.47435, std: 11.99259, params: {'min_child_weight': 1, 'max_dep
            mean: -0.44355, std: 1.89476, params: {'min_child_weight': 3, 'max_dept
            mean: -0.25229, std: 1.56226, params: {'min_child_weight': 5, 'max_dept
            mean: -6.24469, std: 13.53401, params: {'min_child_weight': 1, 'max_dep
            mean: -0.46754, std: 1.93083, params: {'min_child_weight': 3, 'max_dept
            mean: -0.29473, std: 1.63887, params: {'min_child_weight': 5, 'max_dept
            mean: -6.92931, std: 14.90799, params: {'min_child_weight': 1, 'max_dep
            mean: -0.48046, std: 1.95542, params: {'min_child_weight': 3, 'max_dept
            mean: -0.32518, std: 1.68888, params: {'min_child_weight': 5, 'max_dept
           {'max_depth': 3, 'min_child_weight': 1},
           0.19502205893623775)
```

```
In [242]: xgb_params = {
              'n_trees': 520,
              'eta': 0.0045,
              'max_depth': 3,
              'subsample': 0.93,
              'objective': 'reg:linear',
              'eval_metric': 'rmse',
              'base_score': y_mean, # base prediction = mean(target)
              'silent': 1,
              'min_child_weight':1
          }
```

```
In [244]: sub = pd.DataFrame()
          sub['ID'] = id_test
          sub['y'] = y_pred
          sub.to_csv('Submission/xgb_cv3' + '.csv', index = False)
```

```
In [249]: xgb_params_4 = {
              'n_trees': 520,
              'eta': 0.0045,
              'max_depth': 4,
              'subsample': 0.93,
              'objective': 'reg:linear',
              'eval_metric': 'rmse',
              'base_score': y_mean, # base prediction = mean(target)
```

```
            'silent': 1,
            'min_child_weight':1
            }

In [250]: %%time
          dtrain = xgb.DMatrix(train_x, train_y)
          dtest = xgb.DMatrix(test_x)

          num_boost_rounds = 1250
          # train model
          model = xgb.train(dict(xgb_params_4, silent=0), dtrain, num_boost_round=n
          y_pred = model.predict(dtest)

CPU times: user 1min 1s, sys: 523 ms, total: 1min 1s
Wall time: 1min 2s


In [251]: sub = pd.DataFrame()
          sub['ID'] = id_test
          sub['y'] = y_pred
          sub.to_csv('Submission/xgb_cv4' + '.csv', index = False)
```

tune gamma

```
In [253]: gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_

Out[253]: ([mean: -5.47435, std: 11.99259, params: {'gamma': 0.0},
            mean: -5.45500, std: 11.95379, params: {'gamma': 0.1},
            mean: -5.48847, std: 12.01484, params: {'gamma': 0.2},
            mean: -5.48811, std: 12.01501, params: {'gamma': 0.3},
            mean: -5.49255, std: 12.02505, params: {'gamma': 0.4}],
           {'gamma': 0.1},
           -5.4549992290303502)

In [254]: xgb_params_5 = {
              'n_trees': 520,
              'eta': 0.0045,
              'max_depth': 4,
              'subsample': 0.93,
              'objective': 'reg:linear',
              'eval_metric': 'rmse',
              'base_score': y_mean, # base prediction = mean(target)
              'silent': 1,
              'min_child_weight':1,
              'gamma':0.1
          }

In [255]: %%time
          dtrain = xgb.DMatrix(train_x, train_y)
```

```
    dtest = xgb.DMatrix(test_x)

    num_boost_rounds = 1250
    # train model
    model = xgb.train(dict(xgb_params_5, silent=0), dtrain, num_boost_round=n
    y_pred = model.predict(dtest)
```

```
CPU times: user 1min 5s, sys: 924 ms, total: 1min 6s
Wall time: 1min 12s
```

In [256]:
```
sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = y_pred
sub.to_csv('Submission/xgb_cv5' + '.csv', index = False)
```

In [ ]:
```
%%time
# def MSE(y_true,y_pred):
#     mse = mean_squared_error(y_true, y_pred)

#     return mse

def R2(y_true,y_pred):
     r2 = r2_score(y_true, y_pred)

     return r2

# def two_score(y_true,y_pred):
#     MSE(y_true,y_pred) #set score here and not below if using MSE in Gric
#     score = R2(y_true,y_pred)
#     return score

# def two_scorer():
#     return make_scorer(two_score, greater_is_better=True) # change for fa


def scorer():
    return make_scorer(R2)

param_test6 = {
 'gamma':np.arange(0.05,0.15,0.01),

}
gsearch1 = GridSearchCV(estimator = xgb.XGBRegressor( learning_rate =0.0045
 min_child_weight=1, gamma=0, subsample=0.93,
 objective= 'reg:linear', nthread=4, scale_pos_weight=1, seed=2017),
 param_grid = param_test6, scoring=scorer(),n_jobs=4,iid=False, cv=5)
gsearch1.fit(train_x,train_y)
```

```
In [ ]:

In [ ]:

In [213]: kf = KFold(10)

In [217]: %%time
          y_preds = []
          for train_index, test_index in kf.split(train_x):
              dtrain = xgb.DMatrix(train_x.ix[train_index], label = train_y.ix[trai
              model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round
              y_pred = model.predict(xgb.DMatrix(train_x.ix[test_index]))
              score = r2_score(train_y.ix[test_index], y_pred)
              y_preds.append(score)

CPU times: user 20min 5s, sys: 8.45 s, total: 20min 13s
Wall time: 20min 35s


In [221]: np.mean(y_preds)

Out[221]: 0.52009639614048475

In [222]: y_preds

Out[222]: [0.41084962858293206,
           0.58284129573549814,
           0.3402203136351869,
           0.49281114491526601,
           0.65214878239799678,
           0.5125278457743172,
           0.47485432475854339,
           0.56627732922148066,
           0.61354609801642601,
           0.55488719836719924]

In [ ]:

In [61]: '''Train the stacked models then predict the test data'''

Out[61]: 'Train the stacked models then predict the test data'

In [62]: stacked_pipeline = make_pipeline(
             StackingEstimator(estimator = LassoLarsCV(normalize = True)),
             StackingEstimator(estimator = GradientBoostingRegressor(learning_rate
                                                                      loss = 'huber'
                                                                      min_samples_le

             LassoLarsCV())
```

```
In [167]: %%time
          stacked_pipeline.fit(finaltrainset, y_train)
          results = stacked_pipeline.predict(finaltestset)

CPU times: user 5.41 s, sys: 355 ms, total: 5.76 s
Wall time: 6.29 s


In [ ]:

In [ ]: '''R2 Score on the entire Train data when averaging'''

        print('R2 score on train data:')
        print(r2_score(y_train,stacked_pipeline.predict(finaltrainset)*0.2855 + mod

In [ ]: '''Average the preditionon test data  of both models then save it on a csv

        sub = pd.DataFrame()
        sub['ID'] = id_test
        sub['y'] = y_pred*0.75 + results*0.25
        sub.to_csv('stacked-models.csv', index=False)

In [ ]: pwd

In [ ]: r2_score(y_train,y_pred*0.25 + results*0.75)

In [ ]: r2_score(y_train, model.predict(dtrain))

In [ ]: def submission_generation(name):
            sub = pd.DataFrame()
            sub['ID'] = id_test
            sub['y'] = y_pred
            sub.to_csv('Submission/' + name+'.csv', index = False)

In [ ]: submission_generation('only xgboost')

In [ ]:

In [166]: def runXGB(train_X, train_y, test_X, test_y = None, feature_names = None,
              param = {}
              param['objective'] = 'reg:linear'
              param['eta'] = 0.001
              param['max_depth'] = 5
              param['silent'] = 1

              param['eval_metric'] = "rmse"
              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_sto
    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 [ ]: # %%time
        # cv_scores = []
        # kf = KFold(n_splits=10, shuffle=True, random_state=2016)
        # for dev_index, val_index in kf.split(range(train_x.shape[0])):
        #         dev_X, val_X = train_x.ix[dev_index,:], train_x.ix[val_index,:]
        #         dev_y, val_y = train_y.ix[dev_index], train_y.ix[val_index]
        #         preds, model = runXGB(dev_X, dev_y, val_X, val_y)
        #         cv_scores.append(r2_score(val_y, preds))
        #         print(cv_scores)
        #         break
```

2 Stacking

```
In [106]: import xgboost as xgb
```

```
In [148]: class SklearnWrapper(object):
              def __init__(self, clf, params=None): ## remove parameter seed = 0
                  # params['random_state'] = seed
                  self.clf = clf(**params)

              def fit2(self, x_train, y_train):
                  self.clf.fit(x_train, y_train)

              def predict2(self, x):
                  return self.clf.predict(x)
```

```
In [108]: class XGBWrapper(object):
              def __init__(self, seed = 0, params = None):
                  self.params = params
                  self.params['seed'] = seed
                  self.nrounds = params['nround']
```

```
        def fit2(self, x_train, y_train):
            dtrain = xgb.DMatrix(x_train, label = y_train)
            self.gbdt = xgb.train(self.params, dtrain, self.nrounds)

        def predict2(self, x):
            return self.gbdt.predict(xgb.DMatrix(x))
```

In [109]:
```
from sklearn.model_selection import KFold
kf = KFold(5)
```

In [110]:
```
def get_oof(clf, train_x, train_y, test_x):
    n = train_x.shape[0]
    oof_train = np.zeros((n,1))
    oof_test = np.zeros((n,1))
    oof_test_temp = np.zeros((n,1))

    for train_index, test_index in kf.split(train_x):
        model = clf.fit2(train_x.iloc[train_index], train_y.iloc[train_ir
        oof_train[test_index] = clf.predict2(train_x.iloc[test_index]).re
        oof_test_temp = clf.predict2(test_x).reshape(-1,1)
        oof_test = oof_test + oof_test_temp

    oof_test = oof_test / 5

    return oof_train, oof_test
```

2.0.1 XGBoost

In [245]:
```
xg_params = {
    'objective' : 'reg:linear',
    'n_trees':1250,
    'eta' : 0.0045,
    'max_depth' : 3,
    'subsample' : 0.93,
    'eval_metric':'rmse',
    'base_score' : y_mean,
    'silent' : 1 ,
    'nround':1000,
    'min_child_weight':1
            }
```

In [246]:
```
xg = XGBWrapper(seed = 0, params = xg_params)
```

In [ ]:

In [113]:
```
%%time
xg_oof_train, xg_oof_test = get_oof(xg, train_x, train_y, test_x)
```

```
CPU times: user 3min 54s, sys: 958 ms, total: 3min 55s
Wall time: 3min 58s
```

2.0.2 sklearn models

```
In [129]: from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,Gra
          from sklearn.svm import SVR
          from sklearn.linear_model import LassoLarsCV, ElasticNetCV , ElasticNet

In [116]: ## http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Rand
          rf_params = {
              'n_jobs': 16,
              'n_estimators': 1000,
              'max_features': 0.5,
              'max_depth': 10,
              'min_samples_leaf': 2,
          }

In [120]: ## http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Grad
          gbr_params = {'loss':'ls',
                        'learning_rate':0.01,
                        'n_estimators':1000,
                        'max_depth':5,
                        'min_sample_leaf':2,

          }

In [154]: gbr_params2 = {
              'learning_rate' : 0.001,
              'n_estimators':2000,
              'loss' : 'huber',
              'max_depth' : 3,
              'max_features': 0.55,
              'min_samples_leaf' : 18,
              'min_samples_split' : 14,
              'subsample': 0.7
          }

In [121]: ## http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Extr


          et_params = {
              'n_jobs': -1,
              'n_estimators': 1000,
              'max_features': 0.5,
              'max_depth': 10,
              'min_samples_leaf': 2,
          }
```

```
In [140]: ## http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.
          ll_params = {
             # 'fit_intercept':True
              'n_jobs':-1
          }

In [126]: ## http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.
          en_params = {
              'alpha':0.1,
              'l1_ratio':0.7

          }

In [122]: ## http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
          svr_params = {
              'C' : 1,
              'eplison':0.1,
              'kernel':'rbf',
              'gamma':'auto'

          }

In [155]: rf = SklearnWrapper(clf=RandomForestRegressor, params=rf_params)
          gbr = SklearnWrapper(clf = GradientBoostingRegressor,  params = gbr_param
          et = SklearnWrapper(clf = ExtraTreesRegressor,  params = et_params)

          en = SklearnWrapper(clf = ElasticNet, params = en_params)

In [142]: # rf = SklearnWrapper(clf=RandomForestRegressor, seed=2017, params=rf_par
          # gb = SklearnWrapper(clf = GradientBoostingRegressor, seed = 2017, param
          # et = SklearnWrapper(clf = ExtraTreesRegressor, seed = 2017, params = et
          # ll = SklearnWrapper(clf = LassoLarsCV, seed = 2017, params = ll_params)
          # en = SklearnWrapper(clf = ElasticNet, seed = 2017, params = en_params)
          # svr = SklearnWrapper(clf = SVR, seed = 2017, params = svr_params)

In [143]: ll =  LassoLarsCV()

In [146]: svr = SVR()

In [151]: %%time
          rf_oof_train, rf_oof_test =  get_oof(rf, train_x, train_y, test)

CPU times: user 7min 56s, sys: 3.21 s, total: 7min 59s
Wall time: 2min 21s


In [156]: %%time
          gb_oof_train, gb_oof_test = get_oof(gbr, train_x, train_y, test)
```

```
CPU times: user 4min 19s, sys: 782 ms, total: 4min 20s
Wall time: 4min 20s
```

```
In [157]: %%time
          et_oof_train, et_oof_test = get_oof(et, train_x, train_y, test)

Exception ignored in: <bound method DMatrix.__del__ of <xgboost.core.DMatrix object
Traceback (most recent call last):
  File "/Users/Aaron/anaconda/lib/python3.5/site-packages/xgboost-0.6-py3.5.egg/xgb
    _check_call(_LIB.XGDMatrixFree(self.handle))
AttributeError: 'DMatrix' object has no attribute 'handle'
```

```
CPU times: user 4min 5s, sys: 2.26 s, total: 4min 8s
Wall time: 1min 18s
```

```
In [163]: %%time
          en_oof_train, en_oof_test = get_oof(en, train_x, train_y, test)

CPU times: user 8.73 s, sys: 123 ms, total: 8.85 s
Wall time: 7.5 s
```

```
In [159]: def get_oof_extra(clf, train_x, train_y, test_x):
              n = train_x.shape[0]
              oof_train = np.zeros((n,1))
              oof_test = np.zeros((n,1))
              oof_test_temp = np.zeros((n,1))

              for train_index, test_index in kf.split(train_x):
                  model = clf.fit(train_x.iloc[train_index], train_y.iloc[train_inc
                  oof_train[test_index] = clf.predict(train_x.iloc[test_index]).res
                  oof_test_temp = clf.predict(test_x).reshape(-1,1)
                  oof_test = oof_test + oof_test_temp

              oof_test = oof_test / 5

              return oof_train, oof_test
```

```
In [160]: %%time
          svr_oof_train, svr_oof_test = get_oof_extra(svr, train_x, train_y, test)

CPU times: user 1min 5s, sys: 292 ms, total: 1min 5s
Wall time: 1min 5s
```

```
In [161]: %%time
          ll_oof_train, ll_oof_test = get_oof_extra(ll, train_x, train_y, test)

CPU times: user 2.35 s, sys: 367 ms, total: 2.72 s
Wall time: 1.69 s


In [164]: ### Combining all model 1
          x_train_after_stack = np.concatenate((xg_oof_train, et_oof_train, rf_oof_
                                                , ll_oof_train, en_oof_train, svr_oo
          x_test_after_stack = np.concatenate((xg_oof_test, et_oof_test, rf_oof_tes
                                               ll_oof_test, en_oof_test, svr_oof_te

In [168]: %%time
          p, model = runXGB(x_train_after_stack, train_y, x_test_after_stack, num_r

CPU times: user 1.3 s, sys: 26.3 ms, total: 1.32 s
Wall time: 1.4 s


In [170]: x_train_after_stack.shape

Out[170]: (4209, 7)

In [171]: p[:5]

Out[171]: array([ 54.88514328,  60.32428741,  54.88514328,  54.88514328,  68.589157

In [172]: sub = pd.DataFrame()
          sub['ID'] = id_test
          sub['y'] = p
          sub.to_csv('stack1.csv', index = False)

In [173]: sub = pd.DataFrame()
          sub['ID'] = id_test
          sub['y'] = rf_oof_test
          sub.to_csv('rf.csv', index = False)

In [174]: rf_oof_test.shape

Out[174]: (4209, 1)

In [176]: rf_oof_test[:5]

Out[176]: array([[  81.39255755],
                 [ 108.00938191],
                 [  81.43177215],
                 [  82.68408714],
                 [ 119.37385729]])
```

```
In [178]: train_x.head()
```

Out[178]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	pca_11	ica_11
0	0	37	23	20	0	3	27	9	14	0	...	1.360747	-0.016158
1	6	37	21	22	4	3	31	11	14	0	...	-2.803449	-0.025003
2	7	24	24	38	2	3	30	9	23	0	...	3.667395	0.025212
3	9	24	21	38	5	3	30	11	4	0	...	4.234178	0.021985
4	13	24	23	38	5	3	14	3	13	0	...	4.837340	0.013433

	tsvd_11	grp_11	srp_11	pca_12	ica_12	tsvd_12	grp
0	0.534548	-508.356480	-3.981069	4.238816	-0.026000	1.349757	1218.909
1	-0.298973	-515.250317	-2.185324	2.358387	0.001018	-2.785076	1209.804
2	-4.359028	-485.908782	8.223169	-0.968324	-0.025087	4.349495	1188.930
3	-3.811860	-507.908450	-1.033786	-1.581056	-0.018093	4.806347	1179.433
4	-1.678387	-502.634869	2.039579	-1.653438	-0.024047	5.160166	1186.215

	srp_12
0	-14.707015
1	-15.233844
2	-52.632749
3	-22.133079
4	-63.204101

[5 rows x 437 columns]

```
In [183]: r2_score(rf_oof_train, train_y)

Out[183]: -0.18665155841770664

In [181]: rf_oof_train.shape

Out[181]: (4209, 1)

In [182]: test.shape

Out[182]: (4209, 437)

In [185]: r2_score(rf.fit2(train_x, train_y).predict2(train_x), train_y)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-185-795cd0156d22> in <module>()
----> 1 r2_score(rf.fit2(train_x, train_y).predict2(train_x), train_y)

AttributeError: 'NoneType' object has no attribute 'predict2'
```

```
In [186]: r2_score(gb_oof_train, train_y)

Out[186]: -0.11056341949576431

In [187]: r2_score(ll_oof_train, train_y)

Out[187]: -0.10988915924298714

In [188]: r2_score(en_oof_train, train_y)

Out[188]: -0.0099200917466468752

In [189]: r2_score(et_oof_train, train_y)

Out[189]: 0.067769074918884065

In [190]: r2_score(svr_oof_train, train_y)

Out[190]: -8327.3718559387071

In [ ]:
```