kaggle-v6.0

December 17, 2018

In [1]:
```
from conf import *
```

In [2]:
```
%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

In [3]:
```
#eccezioni, non da droppare
target = ["fullVisitorId","totals.totalTransactionRevenue"]
#num cols
nums = ["visitStartTime","totals.totalTransactionRevenue"]
#cat cols
cats = ["trafficSource.adwordsClickInfo.gclId","trafficSource.referralPath","trafficSou

parameters = {
    #numero massimo di valori in una singola colonna per essere flattata, altrimenti d
    "max_new_feat":500,
    #inviare a kaggle tramite l' API
    "commit":0,
    #lgbm tuning parameters
    "n_leaves" : 512,
    "feature_fraction" : 0.99,
    "bagging_fraction" : 0.99,
    "learn_rate" : 0.004,
    #train rows
    "train_rows" : 10000,
    #test_rows, per submittare deve essere settato a -1
    "test_rows" : 500,
    #il metodo principale è lgbm ma si può testare anche la regressione lineare
    "test_also_lin_reg" : 1,
    #bagging frequency
    "bagging_freq" : 1,
    #transactionRevenue
    "transactionRevenue" : 0,
    #percentuale di dev e val
    "percentage" : 18,
    #grouping_mode_cats
    "grouping_mode_cats" : "mode",
```

```
            #score
            "final_score" : -1,
            #minio di alberi per il rf
            "min_child_samples" : -1
        }

        locals().update(parameters)
```

```
In [4]: if (commit==1) & (test_rows != -1):
            raise Exception("per submittare devi usare tutte le righe del test")
```

```
In [5]: %time train_df = load_df(train_file,train_rows,target)
```

```
Loaded train_v2.csv. Shape: (10000, 92)
CPU times: user 24.4 s, sys: 727 ms, total: 25.1 s
Wall time: 25.4 s
```

```
In [6]: %time test_df = load_df(test_file,test_rows,target)
```

```
Loaded test_v2.csv. Shape: (500, 76)
CPU times: user 2.81 s, sys: 67.9 ms, total: 2.88 s
Wall time: 2.91 s
```

```
In [7]: if "totals.totalTransactionRevenue" in test_df.columns:
            test_df = test_df.drop("totals.totalTransactionRevenue",axis=1)
```

```
In [8]: if not transactionRevenue:
            if "totals.transactionRevenue" in test_df.columns:
                test_df = test_df.drop("totals.transactionRevenue",axis=1)
```

```
In [9]: if not transactionRevenue:
            if "totals.transactionRevenue" in train_df.columns:
                train_df = train_df.drop("totals.transactionRevenue",axis=1)
```

```
In [10]: #controllare che nel nuovo test non droppi totalsRevenue perchè c'è solo nel train
         train_df,test_df = drop_uncommons(train_df,test_df,target)

         nums = find_num_cols(train_df,target,cats,nums)

         cats = find_cat_cols(train_df,target,nums)
```

```
In [11]: cc(train_df,test_df,cats,nums)
```

```
In [12]: for col in nums:
             if (col not in target) | (col=="totals.totalTransactionRevenue"):
                 train_df[col] = train_df[col].astype(float)
                 if col!="totals.totalTransactionRevenue":
                     test_df[col] = test_df[col].astype(float)
```

```
In [13]: # Impute 0 for missing target values
         train_df.fillna(0,inplace=True)
         test_df.fillna(0,inplace=True)

In [14]: train_df = stringify_cats(train_df,cats)
         test_df = stringify_cats(test_df,cats)

In [15]: #droppo le colonne che hanno troppa varianza
         train_df,test_df,cats = drop_exceeding(train_df,test_df,max_new_feat,cats,target)

In [16]: cc(train_df,test_df,cats,nums)

In [17]: train_df[cats] = train_df[cats].astype(str)
         test_df[cats] = test_df[cats].astype(str)

In [18]: ######QUI aggiungo il weekday

         #train_df["date"].weekday()

In [19]: %time train_df,test_df,cats = encode_cats(train_df,test_df,cats)

[**********]CPU times: user 5.75 s, sys: 2.63 s, total: 8.38 s
Wall time: 8.87 s


In [20]: cc(train_df,test_df,cats,nums)

In [21]: %time train_df = group_me(train_df,"fullVisitorId",cats,nums,grouping_mode_cats)

[******]CPU times: user 6min 33s, sys: 5.63 s, total: 6min 38s
Wall time: 6min 37s


In [22]: %time test_df = group_me(test_df,"fullVisitorId",cats,nums,grouping_mode_cats)

[******]CPU times: user 19.6 s, sys: 131 ms, total: 19.8 s
Wall time: 19.7 s


In [23]: cc(train_df,test_df,cats,nums)

In [24]: load = -1
         base = "./saved_conf/"

         if load == 1:
             print("hai scelto di importare il dataset da disco")
             train_df = pd.read_csv(base + "dump_train")
             test_df = pd.read_csv(base + "dump_test")
             with open(base + "dump_parameters", 'r') as file:
                 file.read(json.loads(parameters))
```

```
            locals().update(parameters)
        else:
            if load == 0:
                print("hai scelto di scrivere il dataset su disco")
                %time train_df.to_csv(path_or_buf=base + "dump_train", header=True, mode='w',i
                test_df.to_csv(path_or_buf=base + "dump_test", header=True, mode='w',index=Fal
                with open(base + "dump_parameters", 'w') as file:
                    file.write(json.dumps(parameters))
            else:
                print("hai scelto di non caricare nè scaricare il dataset")
```

```
hai scelto di non caricare nè scaricare il dataset
```

```
In [25]: train_id = train_df["fullVisitorId"].values
         test_id = test_df["fullVisitorId"].values

In [26]: #pulizia delle colonne con nomi assurdi
         #questa operazione può essere fatta in maniera safe perchè
         #a questo punto i due datasets hanno le stesse colonne con gli stessi nomi

In [27]: train_df.columns = [col[:30] for col in train_df.columns]
         test_df.columns = [col[:30] for col in test_df.columns]

In [28]: common_feats = list((set(train_df.columns).intersection(set(test_df.columns))).differe

In [29]: #qui viene bloccato il controllo di coerenza poichè le colonne cambiano, in particola
         #vengono accorciati ma è safe farlo perchè i nomi delle colonne sono importanti solo

In [30]: #cc(train_df,test_df,cats,nums)

In [31]: #train_df.head()

In [32]: #splitto il dataframe in development e validation ma cercando di mantenere in maniera
         #corretta il rapporto dei compratori che è circa dell' 1%

         #posso fare confronti con 0 perchè prima tutte le colonne sono state messe a 0 perciò

         #divido il train in 2 parti: quelli che hanno speso che sono l' 1% e quelli che non h
         #speso 99% e da ognuno estraggo il tot% quindi mantengo il rapporto tra i due
         train_money = train_df[train_df["totals.totalTransactionRevenue"]>0]
         train_no_money = train_df[train_df["totals.totalTransactionRevenue"]==0]

         percent = int(len(train_money)*percentage/100)
         train_money_val = train_money.iloc[:percent,]

         dev_df = train_money.iloc[percent:len(train_money),]
         val_df = train_money_val
```

```
percent = int(len(train_no_money)*percentage/100)
train_no_money_val = train_no_money.iloc[:percent,]

dev_df = dev_df.append(train_no_money.iloc[percent:len(train_no_money),])
val_df = val_df.append(train_no_money_val )


################################################
#voglio lavorare su un subset perciò provo a ridurre la grandezza
#mantengo il rapporto ma perdo info nelle features
#quantity=1

#dev_df=dev_df.iloc[:int(len(dev_df)*quantity),:]
#val_df=val_df.iloc[:int(len(val_df)*quantity),:]
################################################

#dev_y contiene la colonna addestramento in dev già log1p
dev_y = np.log1p(dev_df["totals.totalTransactionRevenue"].values)
#val_y contiene la colonna target in val già log1p
val_y = np.log1p(val_df["totals.totalTransactionRevenue"].values)

#dev_x contiene colonne numeriche e cat senza transRev
dev_X = dev_df[ common_feats ]
#val_x contiene colonne numeriche e cat senza transRev
val_X = val_df[ common_feats ]
#test è ciò che dobbiamo trovare
test_X = test_df[ common_feats ]
```

In [33]:
```
def write(tipo):
    parameters["final_score"] = final_score
    try:
        if len(pd.read_csv("./tests.csv").columns) != len(parameters.keys())+1:
            print("il file tests.csv contiene meno colonne del necessario, verrà sost
            !rm "./tests.csv"
    except:
        print("il file tests.csv verrà creato ora perchè non esistente")
    with open("./tests.csv",'a') as ff:
        if os.fstat(ff.fileno()).st_size == 0:
            for k in parameters.keys():
                print(k+',',file = ff,sep='',end='' )
            print("type",file = ff)
        for v in parameters.values():
            print(str(v)+',',file = ff,sep='',end='' )
        print(tipo,file = ff )
```

0.1 Linear Regression

In [34]:
```
from regression import lin

        if test_also_lin_reg == 1:
            pred_test = lin(dev_X,dev_y,test_X)
            pred_val = lin(dev_X,dev_y,val_X)

            val_pred_df = pd.DataFrame({"fullVisitorId":val_df["fullVisitorId"].values})
            val_pred_df["totals.totalTransactionRevenue"] = val_df["totals.totalTransactionRev
            val_pred_df["PredictedRevenue"] = np.expm1(pred_val)
            val_pred_df = val_pred_df.groupby("fullVisitorId")["totals.totalTransactionRevenue
            val_pred_df[val_pred_df["PredictedRevenue"]>10^20]=0

            final_score = np.sqrt(metrics.mean_squared_error(np.log1p(val_pred_df["totals.tota
                                                             , np.log1p(val_pred_df["Predicted
            print(final_score)
            write("lin_reg")
```

```
1.0062071651087179
```

0.2 LightGBM single-tree

In [35]:
```
# custom function to run light-gbm model
        def lgbm(train_X, train_y, val_X, val_y, test_X):

            params = {
                "objective" : "regression",
                "metric" : "rmse",
                "num_leaves" : n_leaves,
                "feature_fraction" : feature_fraction,
                "bagging_fraction" : bagging_fraction,
                "bagging_freq":bagging_freq,
                "learning_rate" : learn_rate,
                "verbosity" : -1
            }

            lgtrain = lgb.Dataset(train_X, label=train_y)
            lgval = lgb.Dataset(val_X, label=val_y)
            model = lgb.train(params, lgtrain, 10000, valid_sets=[lgval], early_stopping_round

            pred_test_y = model.predict(test_X, num_iteration=model.best_iteration)
            pred_val_y = model.predict(val_X, num_iteration=model.best_iteration)
            return pred_test_y, model, pred_val_y
```

In [36]:
```
#%time pred_test, model, pred_val = lgbm(dev_X, dev_y, val_X, val_y, test_X)
```

In [37]:
```
def score():
            pred_val[pred_val<0] = 0
```

```
        val_pred_df = pd.DataFrame({"fullVisitorId":val_df["fullVisitorId"].values})
        val_pred_df["totals.totalTransactionRevenue"] = val_df["totals.totalTransactionRev
        val_pred_df["PredictedRevenue"] = np.expm1(pred_val)
        val_pred_df = val_pred_df.groupby("fullVisitorId")["totals.totalTransactionRevenue
        final_score = np.sqrt(metrics.mean_squared_error(np.log1p(val_pred_df["totals.tota
        print(final_score)

        sub_df = pd.DataFrame({"fullVisitorId":test_id})
        pred_test[pred_test<0] = 0
        sub_df["PredictedLogRevenue"] = np.expm1(pred_test)
        sub_df = sub_df.groupby("fullVisitorId")["PredictedLogRevenue"].sum().reset_index
        sub_df.columns = ["fullVisitorId", "PredictedLogRevenue"]
        sub_df["PredictedLogRevenue"] = np.log1p(sub_df["PredictedLogRevenue"])


        write("LightGBM")

        if commit:
            !kaggle competitions submit -c ga-customer-revenue-prediction -f {my_submissio


        return final_score,sub_df
```

```
In [38]: def write_df(sub_df):
             sub_df.to_csv(path_or_buf=my_submission_file, header=True, mode='w',index=False)
             !wc -l {my_submission_file}
```

```
In [39]: #final_score,sub_df = score()
```

```
In [40]: #write_df(sub_df)
```

```
In [41]: def plot_imp(model):
             fig, ax = plt.subplots(figsize=(12,18))
             lgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax)
             ax.grid(False)
             plt.title("LightGBM - Feature Importance", fontsize=15)
             plt.show()
```

```
In [42]: #plot_imp(model)
```

0.3 LightGBM con rf

```
In [43]: parameters["n_leaves"] = 400
         parameters["bagging_fraction"] = 0.99
         parameters["feature_fraction"] = 0.99
         parameters["bagging_freq"] = 20
         parameters["min_child_samples"] = 10
         locals().update(parameters)
```

```
In [44]: # custom function to run light-gbm model
         def lgbm_rf(train_X, train_y, val_X, val_y, test_X):

             params = {
                 "objective" : "regression",
                 "metric" : "rmse",
                 "num_leaves" : n_leaves,
                 "learning_rate" : learn_rate,
                 "bagging_fraction" : bagging_fraction,
                 "feature_fraction" : feature_fraction,
                 "bagging_freq":bagging_freq,
                 'max_depth':-1,
                 "min_child_samples" : min_child_samples,
                 "boosting":"rf"
             }

             lgtrain = lgb.Dataset(train_X, label=train_y)
             lgval = lgb.Dataset(val_X, label=val_y)
             model = lgb.train(params, lgtrain, 3000, valid_sets=[lgval], verbose_eval=500,kee

             pred_test_y = model.predict(test_X, num_iteration=model.best_iteration)
             pred_val_y = model.predict(val_X, num_iteration=model.best_iteration)
             return pred_test_y, model, pred_val_y

In [45]: #%time pred_test, model, pred_val = lgbm_rf(dev_X, dev_y, val_X, val_y, test_X)

In [46]: #final_score,sub_df = score()

In [47]: #plot_imp(model)
```

0.4 Iterative Testing

```
In [51]: def test_leaves_lgbm():
             for x in [x for x in range(4,10)]:
                 x = 2**x
                 print("testing for ",x)
                 parameters["n_leaves"] = x
                 locals().update(parameters)
                 %time pred_test, model, pred_val = lgbm(dev_X, dev_y, val_X, val_y, test_X)
                 final_score,sub_df = score()
                 print("---------------")

In [52]: %time test_leaves_lgbm()

testing for  16
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
```

```
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.3 s, sys: 1.05 s, total: 31.3 s
Wall time: 8.71 s
1.0101261973148616
---------------
testing for  32
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 33 s, sys: 1.69 s, total: 34.7 s
Wall time: 10.7 s
1.0101261973148616
---------------
testing for  64
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 28.2 s, sys: 605 ms, total: 28.8 s
Wall time: 7.72 s
1.0101261973148616
---------------
testing for  128
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
```

```
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 32.1 s, sys: 1.37 s, total: 33.5 s
Wall time: 9.6 s
1.0101261973148616
---------------
testing for  256
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 28.1 s, sys: 447 ms, total: 28.5 s
Wall time: 7.45 s
1.0101261973148616
---------------
testing for  512
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 32.8 s, sys: 1.51 s, total: 34.3 s
Wall time: 10 s
1.0101261973148616
---------------
CPU times: user 3min 5s, sys: 6.72 s, total: 3min 11s
Wall time: 54.3 s
```

```
In [53]: def test_leaves_lgbm_rf():
             for x in [2**x for x in range(4,10)]:
                 print("testing for ",x)
                 parameters["n_leaves"] = x
                 locals().update(parameters)
                 %time pred_test, model, pred_val = lgbm_rf(dev_X, dev_y, val_X, val_y, test_X
                 final_score,sub_df = score()
```

```
                print("---------------")
```

```
In [54]: %time test_leaves_lgbm_rf()
```

```
testing for  16
[500]        valid_0's rmse: 1.05773
[1000]        valid_0's rmse: 1.05219
[1500]        valid_0's rmse: 1.04915
[2000]        valid_0's rmse: 1.05332
[2500]        valid_0's rmse: 1.0528
[3000]        valid_0's rmse: 1.04959
CPU times: user 23.1 s, sys: 1.08 s, total: 24.1 s
Wall time: 7.19 s
1.0101261973148616
---------------
testing for  32
[500]        valid_0's rmse: 1.05773
[1000]        valid_0's rmse: 1.05219
[1500]        valid_0's rmse: 1.04915
[2000]        valid_0's rmse: 1.05332
[2500]        valid_0's rmse: 1.0528
[3000]        valid_0's rmse: 1.04959
CPU times: user 24.3 s, sys: 1.35 s, total: 25.6 s
Wall time: 7.54 s
1.0101261973148616
---------------
testing for  64
[500]        valid_0's rmse: 1.05773
[1000]        valid_0's rmse: 1.05219
[1500]        valid_0's rmse: 1.04915
[2000]        valid_0's rmse: 1.05332
[2500]        valid_0's rmse: 1.0528
[3000]        valid_0's rmse: 1.04959
CPU times: user 22 s, sys: 802 ms, total: 22.8 s
Wall time: 6.4 s
1.0101261973148616
---------------
testing for  128
[500]        valid_0's rmse: 1.05773
[1000]        valid_0's rmse: 1.05219
[1500]        valid_0's rmse: 1.04915
[2000]        valid_0's rmse: 1.05332
[2500]        valid_0's rmse: 1.0528
[3000]        valid_0's rmse: 1.04959
CPU times: user 25 s, sys: 1.44 s, total: 26.4 s
Wall time: 7.81 s
1.0101261973148616
---------------
```

```
testing for  256
[500]        valid_0's rmse: 1.05773
[1000]        valid_0's rmse: 1.05219
[1500]        valid_0's rmse: 1.04915
[2000]        valid_0's rmse: 1.05332
[2500]        valid_0's rmse: 1.0528
[3000]        valid_0's rmse: 1.04959
CPU times: user 21.8 s, sys: 670 ms, total: 22.5 s
Wall time: 6.11 s
1.0101261973148616
---------------
testing for  512
[500]        valid_0's rmse: 1.05773
[1000]        valid_0's rmse: 1.05219
[1500]        valid_0's rmse: 1.04915
[2000]        valid_0's rmse: 1.05332
[2500]        valid_0's rmse: 1.0528
[3000]        valid_0's rmse: 1.04959
CPU times: user 22.1 s, sys: 759 ms, total: 22.9 s
Wall time: 6.4 s
1.0101261973148616
---------------
CPU times: user 2min 18s, sys: 6.16 s, total: 2min 24s
Wall time: 41.6 s
```

```
In [55]: def test_leaves_lgbm_best_value():
             for x in [x for x in range(16,33)]:
                 print("testing for ",x)
                 parameters["n_leaves"] = x
                 locals().update(parameters)
                 %time pred_test, model, pred_val = lgbm(dev_X, dev_y, val_X, val_y, test_X)
                 final_score,sub_df = score()
                 print("---------------")

In [56]: %time test_leaves_lgbm_best_value()

testing for  16
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
```

```
CPU times: user 32 s, sys: 1.48 s, total: 33.5 s
Wall time: 9.96 s
1.0101261973148616
---------------
testing for  17
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.1 s, sys: 944 ms, total: 31.1 s
Wall time: 8.62 s
1.0101261973148616
---------------
testing for  18
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.1 s, sys: 1.07 s, total: 31.2 s
Wall time: 8.75 s
1.0101261973148616
---------------
testing for  19
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.2 s, sys: 953 ms, total: 31.1 s
Wall time: 8.57 s
1.0101261973148616
```

```
---------------
testing for  20
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 28 s, sys: 409 ms, total: 28.4 s
Wall time: 7.38 s
1.0101261973148616
---------------
testing for  21
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.7 s, sys: 1.07 s, total: 31.7 s
Wall time: 8.84 s
1.0101261973148616
---------------
testing for  22
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.4 s, sys: 1.07 s, total: 31.5 s
Wall time: 8.91 s
1.0101261973148616
---------------
testing for  23
Training until validation scores don't improve for 300 rounds.
```

```
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 32.4 s, sys: 1.52 s, total: 33.9 s
Wall time: 9.96 s
1.0101261973148616
---------------
testing for  24
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 31.3 s, sys: 1.21 s, total: 32.5 s
Wall time: 9.42 s
1.0101261973148616
---------------
testing for  25
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 32.3 s, sys: 1.53 s, total: 33.8 s
Wall time: 10.2 s
1.0101261973148616
---------------
testing for  26
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
```

```
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.9 s, sys: 1.07 s, total: 31.9 s
Wall time: 9.02 s
1.0101261973148616
---------------
testing for  27
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 33.8 s, sys: 1.88 s, total: 35.7 s
Wall time: 11 s
1.0101261973148616
---------------
testing for  28
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 33.2 s, sys: 1.62 s, total: 34.8 s
Wall time: 10.6 s
1.0101261973148616
---------------
testing for  29
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
```

```
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 28.1 s, sys: 577 ms, total: 28.7 s
Wall time: 7.71 s
1.0101261973148616
---------------
testing for  30
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 32 s, sys: 1.43 s, total: 33.5 s
Wall time: 10.2 s
1.0101261973148616
---------------
testing for  31
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
CPU times: user 30.7 s, sys: 1.12 s, total: 31.9 s
Wall time: 8.98 s
1.0101261973148616
---------------
testing for  32
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.16704
[600]        valid_0's rmse: 1.06321
[900]        valid_0's rmse: 1.04217
[1200]        valid_0's rmse: 1.03181
[1500]        valid_0's rmse: 1.02217
[1800]        valid_0's rmse: 1.01757
[2100]        valid_0's rmse: 1.01564
Early stopping, best iteration is:
[2042]        valid_0's rmse: 1.01446
```

```
CPU times: user 29.4 s, sys: 757 ms, total: 30.2 s
Wall time: 8.23 s
1.0101261973148616
---------------
CPU times: user 8min 47s, sys: 19.8 s, total: 9min 6s
Wall time: 2min 36s
```