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" : "mean",
            #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 23.2 s, sys: 594 ms, total: 23.8 s
Wall time: 23.9 s
```

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

```
Loaded test_v2.csv. Shape: (500, 76)
CPU times: user 2.72 s, sys: 47.6 ms, total: 2.77 s
Wall time: 2.78 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 4.67 s, sys: 2.04 s, total: 6.72 s
Wall time: 6.73 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 455 ms, sys: 84.7 ms, total: 539 ms
Wall time: 537 ms


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

[******]CPU times: user 268 ms, sys: 21.4 ms, total: 289 ms
Wall time: 285 ms


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.totalTransactionRe
             val_pred_df["PredictedRevenue"] = np.expm1(pred_val)
             val_pred_df = val_pred_df.groupby("fullVisitorId")["totals.totalTransactionRevenu
             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["Predicte
             print(final_score)
             write("lin_reg")
```

```
1.0230158552172006
```

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_roun

             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.totalTransactionRevenu
        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 [48]: 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("---------------")

testing for  16
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.63636
[600]        valid_0's rmse: 1.63987
Early stopping, best iteration is:
[380]        valid_0's rmse: 1.62873
CPU times: user 6.27 s, sys: 255 ms, total: 6.53 s
Wall time: 1.93 s
```

```
1.6287219856869415
---------------
testing for  32
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63159
[600]       valid_0's rmse: 1.63221
Early stopping, best iteration is:
[400]       valid_0's rmse: 1.62434
CPU times: user 13.5 s, sys: 1.42 s, total: 15 s
Wall time: 5.16 s
1.624220219230191
---------------
testing for  64
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63217
[600]       valid_0's rmse: 1.63511
Early stopping, best iteration is:
[397]       valid_0's rmse: 1.62528
CPU times: user 15.3 s, sys: 471 ms, total: 15.8 s
Wall time: 4.34 s
1.625158693119286
---------------
testing for  128
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63217
[600]       valid_0's rmse: 1.63426
Early stopping, best iteration is:
[397]       valid_0's rmse: 1.62528
CPU times: user 21.8 s, sys: 187 ms, total: 22 s
Wall time: 5.59 s
1.6251567686026698
---------------
testing for  256
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63217
[600]       valid_0's rmse: 1.63426
Early stopping, best iteration is:
[397]       valid_0's rmse: 1.62528
CPU times: user 41.5 s, sys: 1.98 s, total: 43.5 s
Wall time: 13.1 s
1.6251567663041573
---------------
testing for  512
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63217
[600]       valid_0's rmse: 1.63426
Early stopping, best iteration is:
[397]       valid_0's rmse: 1.62528
```

```
CPU times: user 39.9 s, sys: 1.25 s, total: 41.1 s
Wall time: 11.4 s
1.625156766379118
---------------
```

```
In [51]: 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("---------------")
```

```
testing for  16
[500]        valid_0's rmse: 1.73791
[1000]        valid_0's rmse: 1.74423
[1500]        valid_0's rmse: 1.74733
[2000]        valid_0's rmse: 1.75378
[2500]        valid_0's rmse: 1.75313
[3000]        valid_0's rmse: 1.75144
CPU times: user 26 s, sys: 1.01 s, total: 27 s
Wall time: 7.7 s
1.751441067818203
---------------
testing for  32
[500]        valid_0's rmse: 1.81967
[1000]        valid_0's rmse: 1.83387
[1500]        valid_0's rmse: 1.8389
[2000]        valid_0's rmse: 1.847
[2500]        valid_0's rmse: 1.84379
[3000]        valid_0's rmse: 1.84142
CPU times: user 31.7 s, sys: 346 ms, total: 32.1 s
Wall time: 8.14 s
1.8414151694689804
---------------
testing for  64
[500]        valid_0's rmse: 1.82008
[1000]        valid_0's rmse: 1.8346
[1500]        valid_0's rmse: 1.83987
[2000]        valid_0's rmse: 1.84797
[2500]        valid_0's rmse: 1.84474
[3000]        valid_0's rmse: 1.84241
CPU times: user 39.9 s, sys: 1.55 s, total: 41.5 s
Wall time: 11.5 s
1.842406748104294
---------------
testing for  128
```

```
[500]        valid_0's rmse: 1.82008
[1000]        valid_0's rmse: 1.8346
[1500]        valid_0's rmse: 1.83987
[2000]        valid_0's rmse: 1.84797
[2500]        valid_0's rmse: 1.84474
[3000]        valid_0's rmse: 1.84241
CPU times: user 42.5 s, sys: 2.03 s, total: 44.5 s
Wall time: 12.8 s
1.842406748104294
---------------
testing for  256
[500]        valid_0's rmse: 1.82008
[1000]        valid_0's rmse: 1.8346
[1500]        valid_0's rmse: 1.83987
[2000]        valid_0's rmse: 1.84797
[2500]        valid_0's rmse: 1.84474
[3000]        valid_0's rmse: 1.84241
CPU times: user 42.7 s, sys: 1.87 s, total: 44.6 s
Wall time: 12.9 s
1.842406748104294
---------------
testing for  512
[500]        valid_0's rmse: 1.82008
[1000]        valid_0's rmse: 1.8346
[1500]        valid_0's rmse: 1.83987
[2000]        valid_0's rmse: 1.84797
[2500]        valid_0's rmse: 1.84474
[3000]        valid_0's rmse: 1.84241
CPU times: user 40.4 s, sys: 1.02 s, total: 41.4 s
Wall time: 10.9 s
1.842406748104294
---------------
```

```
In [53]: 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("---------------")
```

```
testing for  16
Training until validation scores don't improve for 300 rounds.
[300]        valid_0's rmse: 1.63636
[600]        valid_0's rmse: 1.63987
Early stopping, best iteration is:
[380]        valid_0's rmse: 1.62873
```

```
CPU times: user 6.6 s, sys: 298 ms, total: 6.89 s
Wall time: 2.06 s
1.6287219856869415
---------------
testing for  17
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63779
[600]       valid_0's rmse: 1.64154
Early stopping, best iteration is:
[399]       valid_0's rmse: 1.62945
CPU times: user 9.05 s, sys: 748 ms, total: 9.8 s
Wall time: 3.29 s
1.6294353614099684
---------------
testing for  18
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63331
[600]       valid_0's rmse: 1.6355
Early stopping, best iteration is:
[400]       valid_0's rmse: 1.62482
CPU times: user 7.47 s, sys: 315 ms, total: 7.78 s
Wall time: 2.25 s
1.6248081114912876
---------------
testing for  19
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63129
[600]       valid_0's rmse: 1.634
Early stopping, best iteration is:
[400]       valid_0's rmse: 1.62301
CPU times: user 7.62 s, sys: 250 ms, total: 7.87 s
Wall time: 2.17 s
1.6229723334486241
---------------
testing for  20
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63121
[600]       valid_0's rmse: 1.63453
Early stopping, best iteration is:
[400]       valid_0's rmse: 1.62302
CPU times: user 8.19 s, sys: 370 ms, total: 8.56 s
Wall time: 2.48 s
1.622974978937571
---------------
testing for  21
Training until validation scores don't improve for 300 rounds.
[300]       valid_0's rmse: 1.63222
[600]       valid_0's rmse: 1.6333
```

```
Early stopping, best iteration is:
[401]	valid_0's rmse: 1.62429
CPU times: user 7.88 s, sys: 236 ms, total: 8.11 s
Wall time: 2.2 s
1.6242393262244306
---------------
testing for  22
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63242
[600]	valid_0's rmse: 1.63471
Early stopping, best iteration is:
[403]	valid_0's rmse: 1.62476
CPU times: user 8.04 s, sys: 223 ms, total: 8.26 s
Wall time: 2.22 s
1.6246811229180314
---------------
testing for  23
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63093
[600]	valid_0's rmse: 1.63254
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62409
CPU times: user 9.74 s, sys: 710 ms, total: 10.4 s
Wall time: 3.47 s
1.6239978885849027
---------------
testing for  24
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63154
[600]	valid_0's rmse: 1.63518
Early stopping, best iteration is:
[405]	valid_0's rmse: 1.62478
CPU times: user 10.1 s, sys: 709 ms, total: 10.8 s
Wall time: 3.57 s
1.6246903455290411
---------------
testing for  25
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63293
[600]	valid_0's rmse: 1.63101
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62506
CPU times: user 10.6 s, sys: 746 ms, total: 11.3 s
Wall time: 3.54 s
1.6249800387555666
---------------
testing for  26
Training until validation scores don't improve for 300 rounds.
```

```
[300]	valid_0's rmse: 1.6318
[600]	valid_0's rmse: 1.62999
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62331
CPU times: user 10.4 s, sys: 695 ms, total: 11.1 s
Wall time: 3.49 s
1.6231936069407145
---------------
testing for  27
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63265
[600]	valid_0's rmse: 1.63167
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62442
CPU times: user 8.72 s, sys: 247 ms, total: 8.97 s
Wall time: 2.43 s
1.6243252429876853
---------------
testing for  28
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63268
[600]	valid_0's rmse: 1.63329
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62427
CPU times: user 11.2 s, sys: 673 ms, total: 11.9 s
Wall time: 3.49 s
1.6241598282203922
---------------
testing for  29
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63265
[600]	valid_0's rmse: 1.63488
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62493
CPU times: user 10.1 s, sys: 454 ms, total: 10.6 s
Wall time: 3.03 s
1.6248069142765094
---------------
testing for  30
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63151
[600]	valid_0's rmse: 1.63329
Early stopping, best iteration is:
[397]	valid_0's rmse: 1.62449
CPU times: user 10.2 s, sys: 436 ms, total: 10.6 s
Wall time: 3.04 s
1.6243706858432774
---------------
```

```
testing for  31
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63242
[600]	valid_0's rmse: 1.6333
Early stopping, best iteration is:
[396]	valid_0's rmse: 1.62495
CPU times: user 10.3 s, sys: 400 ms, total: 10.7 s
Wall time: 3.03 s
1.624837110639381
---------------
testing for  32
Training until validation scores don't improve for 300 rounds.
[300]	valid_0's rmse: 1.63159
[600]	valid_0's rmse: 1.63221
Early stopping, best iteration is:
[400]	valid_0's rmse: 1.62434
CPU times: user 9.41 s, sys: 146 ms, total: 9.56 s
Wall time: 2.45 s
1.624220219230191
---------------
```