Non-universal Default Methods for AutoML

AutoML—KDD CUP 2019, Team B-KDD

Yike Guo[1], Ming Ding[2], Jingqing Zhang[1], Kele Xu[3], Zhifeng Gao[4], Xiaohan Yi[4], Shichao Kan[5], Bo Li[6], Chenyue Lu[7], Yanbo J. Wang[8]

[1] Imperial College London, {y.guo, jingqing.zhang15}@imperial.ac.uk
[2] Beijing Emay Softcom Technology Ltd., 409786373@qq.com
[3] National University of Defense Technology, kelele.xu@gmail.com
[4] Peking University, {gao_zhifeng, chlxyd}@pku.edu.cn
[5] Beijing Jiaotong University, 16112062@bjtu.edu.cn
[6] Beijing University of Posts and Telecommunications, deepblue.lb@gmail.com
[7] Bangkejiaruan Information Technology Ltd., luchenue@gmail.com
[8] University of International Business and Economics, wang_yb@uibe.edu.cn

AutoML

Typical Pipeline of Machine Learning

Feature Engineering: *cat: categorical* feature, an integer

- *cat* in the main table：main_k, k1, k2, k3, k4, c1, c2, c3, c4, c5
- Group 1：key = main_k, c = [k1, k2, k3, k4]
- Group 2：key =[k1, k2, k3, k4, c1, c2, c3, c4, c5], c = [main_k]
- Feature crosses: groupby(key).c.map(unique)
- Naïve feature: count encoding for all *cat* columns
- Retain raw features

c_1	c_2	c_3	c_02	c_01
1547156292460673948 8	1477140115124674786 9	1491713516984025063 6	4450187851156539593	1441813205520435181 6
1547156292460673948 8	1477140115124674786 9	1491713516984025063 6	4450187851156539593	1441813205520435181 6
1547156292460673948 8	1477140115124674786 9	2518459674421623809	4450187851156539593	1471682470271360914 6
1547156292460673948 8	1477140115124674786 9	nan	4450187851156539593	7944328449160122873
1547156292460673948 8	1477140115124674786 9	nan	4450187851156539593	7944328449160122873
1547156292460673948 8	1477140115124674786 9	1074274011086202650 3	4450187851156539593	1432477070190016298 6
1547156292460673948 8	1477140115124674786 9	3245770646492378216	4450187851156539593	1039323220572613749 7
1547156292460673948 8	1477140115124674786 9	3245770646492378216	4450187851156539593	1039323220572613749 7
1547156292460673948 8	1477140115124674786 9	1576405665030144817	4450187851156539593	1314205385433465361 5

Feature Engineering: *multi-cat: multi-value categorical* Feature: a set of integers, split by the comma.

- Each column has various number of value, missing values are common
- Naïve features：first, count, max, min
- TopK features：value_counts and keep top5
- Feature transformation：Sorting and indexing for all possible values to reduce memory cost
- Others：matrix decomposition (no clear improvement)

m_5	m_6
1424658006609...	1762300949447501459, 1086209129492879775, 1775772373012578060, 1210810826568567952, 1401382722452338574, 1808660062599197866, 1621752711412666890, 1424658066096349545 0, 1401382722...
1440355466044...	1112643148638127990 8, 7576661479484112078, 1631934657315116022, 6762034980918878821, 8900826406680265835, 6184859593055600466, 4524039208298693280, 3135959926243692765, 5176824202665...
1440355466044...	1440355466046114082 6, 8687959905840099307, 8687959905840099307, 7975891937206711011, 1167601845856010839 2, 1845103123481925 41, 4977485048243745539, 2002046427857346320, 6374907413338 3...
1600162513822...	1600162513822504595, 1736988417921902152 7, 9732311525218367813, 3469760874165103855, 5352179907982106683, 1638231242292087605 4, 1099545968286217356 2
6536810263819...	6536810263819855580, 1076869169931381100 34, 5514145401570336318, 3948504310464497728, 3308258733780402358, 7044331531145705012, 5380662013524298092, 8834292916447907018, 7044331531145705...
1682871280221...	6999291990201313288, 1024995562692534295 8, 1551495932781294489 5, 8585241690577846319, 2966453060669992507 5, 1750939790978863754 2, 9902812066728847071, 1109736744403935628 6, 6040633434744...
1440355466044...	1233072459856008989, 1440355466046114082 6, 1026371210959921369 0
nan	1469648921758975087 0, 1097551641200794819, 3336034437468264982, 5182117108453591448, 1095114195111716618 9, 1415205184456110974, 3570804945235115170, 1614701278262970502, 1680107675724...
nan	8173846417840837776
1745156978600...	1745156786001754931 1
nan	nan
5032090355072...	5032090355072484807
1010507131596...	6448042231042864979, 4455914309992404777, 4265083610283588639, 2962633538197748938, 1010507131599364736 2, 4502044243959595604, 1354167056304662286, 2916840126641875520, 1658549794832...

Feature Engineering

```
@timeit
def transform_multi_hash(df,config,tname,y = pd.Series()):
    maintype = list(config['tables'][CONSTANT.MAIN_TABLE_NAME]['type'].keys())
    maintype = [i for i in maintype if i.startswith(CONSTANT.MULTI_CAT_PREFIX)]
    muldf = df[[c for c in df if c.startswith(CONSTANT.MULTI_CAT_PREFIX) and (c.find('nan') < 1)]]
    df.drop(muldf.columns,axis=1,inplace=True)
    for c in muldf:
        dfstack = muldf[c].apply(lambda x: ','.join(str(x).split(',')[:40])).str.split(',', expand = True)
        df['%s_stack_first'%c] =  dfstack[0].astype(np.float32)
        df['%s_stack_count'%c] = dfstack.count(axis=1).fillna(0).astype(np.int16)
        df['%s_stack_max'%c] = dfstack.astype(np.float32).max(axis=1)
        df['%s_stack_min'%c] = dfstack.astype(np.float32).min(axis=1)
        shape1 = 5
        dfstack = dfstack.stack()
        dfstack = pd.DataFrame(dfstack)
        dfstack.columns = ['mul']
        multisort = sorted(list(dfstack['mul'].value_counts().sort_values(ascending=False)[:(shape1)].index))
        labeldict = dict(enumerate(list(sorted(dfstack['mul'].astype(np.float32).unique()))))
        labeldict = dict(zip(labeldict.values(),labeldict.keys()))
        df['%s_stack_first'%c] = df['%s_stack_first'%c].map(labeldict).fillna(0).astype(np.int32)
        df['%s_stack_max'%c] = df['%s_stack_max'%c].map(labeldict).fillna(0).astype(np.int32)
        df['%s_stack_min'%c] = df['%s_stack_min'%c].map(labeldict).fillna(0).astype(np.int32)
        dfstack = dfstack[dfstack['mul'].isin(multisort)]
        dfstack.index = [i[0] for i in dfstack.index]
        dfstack['col'] = dfstack.index
        dfstack['count'] = 1
        dfstack = dfstack.groupby(['col','mul'])['count'].count().unstack().reset_index()
        dfstack.drop('col',axis=1,inplace=True)
        multisort = [i for i in multisort if i in dfstack.columns]
        dfstack = dfstack[multisort].add_prefix('%s_stack_'%c)
        for cols in dfstack.columns:
            try:
                df[cols] = dfstack[cols].fillna(0).astype(np.int16)
            except:
                pass
```

Feature Engineering: *time* feature, an integer that indicates the timestamp.

- Before feature extraction: sorting by time, handling missing values, formatting by pandas, filling nan by preceding or succeeding values, recording time range
- Naive feature: duration of time difference (seconds)
- Grouping by: over 1 year, over 1 month, over 1 week, over 1 day
- For columns with minute-level features, no second-level features will be extracted.
- If hour-, minuate-, second-level features are missing, then microsecond-level features will be extracted.
- Feature crosses：by main_key, calculating time difference within a group

t_1	t_2	t_01
2017-11-14 20:41:24.432000	2017-08-09 21:12:26.352000	2017-12-05 00:00:00.767000
2017-11-11 19:49:04.312000	2017-04-21 17:23:07.354000	2017-12-07 00:00:00.763000
2017-11-15 00:34:41.335000	2017-09-29 22:41:43.306000	2017-12-07 00:00:00.558000
NaT	2017-10-28 23:27:25.136000	2017-12-07 00:00:00.547000
2017-11-15 21:19:15.215000	2017-09-13 14:06:20.110000	2017-12-11 00:00:00.648000

Feature Engineering: *time* feature, an integer that indicates the timestamp.

```
    days = config['timecoltotal'] / (3600*24)
if days >= 180 or (tname != CONSTANT.MAIN_TABLE_NAME and truec != config['time_col']):
    df['%s_year'%c] = df[c].dt.year.astype(np.int16)
    df['%s_dayofweek'%c] = df[c].dt.dayofweek.astype(np.int16)
    df['%s_daysinmonth'%c] = df[c].dt.daysinmonth.astype(np.int16)
    df['%s_weekofyear'%c] = df[c].dt.weekofyear.astype(np.int16)
    df['%s_hour'%c] = df[c].dt.hour.astype(np.int16)
    df['%s_minute'%c] = df[c].dt.minute.astype(np.int16)
    df['%s_month'%c] = df[c].dt.month.astype(np.int16)
elif days >=25:
    df['%s_month'%c] = df[c].dt.month.astype(np.int16)
    df['%s_dayofweek'%c] = df[c].dt.dayofweek.astype(np.int16)
    df['%s_hour'%c] = df[c].dt.hour.astype(np.int16)
    df['%s_minute'%c] = df[c].dt.minute.astype(np.int16)
elif days >=15:
    df['%s_dayofweek'%c] = df[c].dt.dayofweek.astype(np.int16)
    df['%s_hour'%c] = df[c].dt.hour.astype(np.int16)
    df['%s_minute'%c] = df[c].dt.minute.astype(np.int16)
elif days >=5:
    df['%s_dayofweek'%c] = df[c].dt.dayofweek.astype(np.int16)
    df['%s_hour'%c] = df[c].dt.hour.astype(np.int16)
    df['%s_minute'%c] = df[c].dt.minute.astype(np.int16)
elif days >=1:
    df['%s_hour'%c] = df[c].dt.hour.astype(np.int16)
    df['%s_minute'%c] = df[c].dt.minute.astype(np.int16)
else:
    df['%s_minute'%c] = df[c].dt.minute.astype(np.int16)
if len(df['%s_minute'%c].unique()) <20 :
    df['%s_microsecond'%c] = df[c].dt.microsecond.astype(np.int32)
```

Feature Crosses

- Feature crosses by combining two or multiple columns in raw data
- Regular：groupby key and time
- Auto：based on leaf nodes of XGBoost
- Advanced feature crosses: nonlinear relations between features

Feature Crosses: Regular

- groupby(keys)[cols].agg(ops)
- ops including last, first, mode, min, max, mean, std, sum, median, skur, skew, nunique, count, rank, etc.

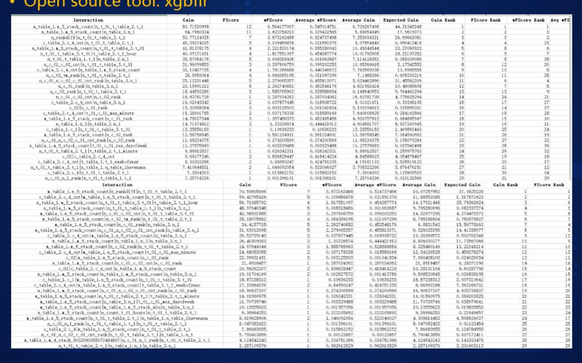
- Given main_key k1, k2, k3, k4, c1, c2, c3, c4, c
- main_key is uid　k1 is ID　k2 is Phone Model　k3 is Location　k4 is some other important key

Temporal features

- If hour-level information is available e.g. 2017-08-09 21:12:26, the extracted value will be 2017080921
- If only day-level information is available e.g. 2017-12-05 00:00:00, the extracted value will be 20171205
- The extracted values will be used with main_key for ranking and generating new features.

Feature Crosses: Auto by Leaf nodes of XGBoost

- Open source tool: xgbfir



Feature Crosses: Auto by Leaf nodes of XGBoost

- Excluding groups with temporal information
- Extracting Top10 groups
- New features by adding and dividing
- Without memory and time limitation, the whote dataset can be used in XGBoost
- With memory and time limitation, grouping by main_key, using 1/5 main_key, 20 trees, default parameter settings.

Feature Crosses: Leaf nodes of XGBoost

```
@timeit
def xgbgain(df,label,config):
    import xgboost as xgb
    import xgbfir
    if len(set(label)) == 2:
        xgbmodel = xgb.XGBClassifier(tree_method = 'hist',n_estimators=20,seed=2019,n_jobs=4).fit(df,label)
    else:
        xgbmodel = xgb.XGBRegressor(tree_method = 'hist',n_estimators=20,seed=2019,n_jobs=4).fit(df,label)
    filename = datetime.now().strftime("%MnHmS")
    xgbfir.saveXgbFI(xgbmodel,feature_names=df.columns,OutputXlsxFile='%s.xlsx'%filename)
    try:
        gaindf = pd.read_excel('%s.xlsx'%filename,sheet_name='Interaction Depth 0')
        catecol = gaindf[gaindf['Average Tree Depth'] <2]['Interaction'].to_list()
        catecol = [i for i in catecol if i.count(CONSTANT.CATEGORY_PREFIX) >=1]
        config['catecol'] = catecol
        ranklist = [i for i in gaindf.columns if i.count('Rank') >=1 ]
        gainlist = []
        for rank in ranklist:
            gainlist.extend(gaindf[gaindf[rank] <=10]['Interaction'].to_list())
        gainlist = list(set(gainlist))
        gainlist = [i for i in gainlist if i.count(CONSTANT.TIME_PREFIX) < 1]
        config['depthzero'] = gainlist
        zero_num = []
        for zeronum in [i for i in gainlist if i.count(CONSTANT.NUMERICAL_PREFIX) >=1]:
            cutlist = [float('-inf')]
            cutlist.append(float('inf'))
            zero_num.append([zeronum,cutlist])
        config['zero_num_cat'] = zero_num
    except:
        config['catecol'] = []
        config['depthzero'] = []
        config['zero_num_cat'] = []
    try:
        gaindf = pd.read_excel('%s.xlsx'%filename,sheet_name='Interaction Depth 1')
        ranklist = [i for i in gaindf.columns if i.count('Rank') >=1 ]
        gainlist = []
        for rank in ranklist:
            gainlist.extend(gaindf[gaindf[rank] <=10]['Interaction'].to_list())
        gainlist = list(set(gainlist))
        gainlist = [i for i in gainlist if i.count(CONSTANT.TIME_PREFIX) < 1]
        gainlist = [sorted(i.split('|')) for i in gainlist]
        gainlist = [i for i in gainlist if i[0] != i[1]]
        config['depthone'] = gainlist
    except:
        config['depthone'] = []
```

Feature Crosses: Auto Leaf nodes of XGBoost

```
@timeit
def transform_depthone(df, config):
    for firstcol,secondcol in config['depthone']:
        df['add_%s_%s'%(firstcol,secondcol)] = (df[firstcol] + df[secondcol]).fillna(0).astype(np.float32)
        if df[firstcol].max() > df[secondcol].max():
            df['divede_%s_%s'%(firstcol,secondcol)] = ((df[firstcol] + 1)/(df[secondcol] + 1)).fillna(0).astype(np.float32)
        else:
            df['divede_%s_%s'%(firstcol,secondcol)] = ((df[secondcol] + 1)/(df[firstcol] + 1)).fillna(0).astype(np.float32)
```

Feature Engineering: *numerical* Feature, a real value.

- 1、Retain raw values
- 2、Ranking with *cat* columns
- 3、Featuring crossing with main_key by mean, sum, last
- 4、Based on XGBoost, BINs for top numerical features.

```
@timeit
def transform_zero_cat(df, config,y):
    for zeronum,cutlist in  config['zero_num_cat']:
        if not y.empty:
            cutlist = [float('-inf')]
            for qu in range(1,10):
                cutlist.append(df[zeronum].quantile(qu * 0.1))
            cutlist.append(float('inf'))
            config[f'{zeronum}_cut'] = cutlist
        else:
            cutlist = config[f'{zeronum}_cut']
        df[zeronum + '_zerocat'] = pd.cut(df[zeronum],
          list(sorted(set(cutlist))),
          labels = range(len(set(cutlist)) -1)).fillna(0).astype(np.int16)
```

Handling Missing Values

- Excluding some columns
- 1、Excluding columns that contain supplementary information.
- 2、Excluding the *cat* and *num* that have only two columns.
- 3、Excluding columns of which more than 50% values are NaN.
- 4、If a value has >= 2 unique digits but more than 50% digits are identical (e.g. 111112), then the whole column will be excluded.

- Handling Missing Values
- 1、numerical --> -1
- 2、cat --> 0s with length same as mode[0], type str
- 3、multi same as cat
- 4、time --> preceding or succeeding values

Deduplication based on Correlation

- Duplication: if the correlation between two columns is higher than 0.98
- Deduplication: excluding one of the columns

```
@timeit
def drop_corr(df,config):
    keys = []
    for i in config['relations']:
        keys.extend(i['key'])
    threshold = 0.98
    dropcol = []
    for ty in [CONSTANT.CATEGORY_PREFIX,CONSTANT.NUMERICAL_PREFIX,CONSTANT.MULTI_CAT_PREFIX]:
        corr_matrix = df[[c for c in df if c.startswith(ty) and c not in keys]].corr()
        select_use_features = corr_matrix.columns.values.tolist()
        for idx,feature in enumerate(corr_matrix.columns):
            if feature in select_use_features:
                ##print('deal feature: ',feature)
                tmp = corr_matrix[feature].values[idx+1:]
                for i,v in enumerate(tmp):
                    tmp_idx = idx + i
                    kv = corr_matrix.columns.values[tmp_idx]
                    if kv in select_use_features and v > threshold:
                        select_use_features.remove(kv)
                        print('***remove feauture: ',kv)
                        if kv not in keys:
                            dropcol.append(kv)
    df.drop(dropcol,axis=1,inplace=True)
```

AutoML Time Estimation

- AutoML: lightGBM
- 1、Customised time evaluation function
- 2、Training 3 trees and evaluating time efficiency by the customised function
- 3、Estimating avg / min time needed for each fold
- 4、Dividing by the remaining time limitation by 3 as the model needs to be trained on at least 3 folds.
- 5、Estimating max number of rounds for one fold based on the time needed for the 3 trees
- 6、Evaluating time efficiency more accurately by the time needed for the one fold and the number of rounds
- Allowing extra 10 seconds to avoid timeout

```
def time_auc(preds, train_data):
    score = time.time()
    return 'time_auc',score,True
```

```
lgbmodel = lgb.train({**params, **best_hyperparams}, train_X, 3
                     , valid_data,feval=time_auc,early_stopping_rounds=False,
                     verbose_eval=False,evals_result=evals_result)
timeys = [datetime.fromtimestamp(i) for i in evals_result['valid_0']['time_auc']]
time2  = np.min([i.total_seconds() for i in np.diff(timeys)])
```

Hyper-parameters Search

- Smaller searching space and fewer searching iterations due to time limitation
- 4 fixed values for each parameters and the searching space is decided by the 7 datasets.
- Numround: 12, searching iteration: 7 , early-stopping is consistent.

```
@timeit
def hyperopt_lightgbm_data(train_data, valid_data, params: Dict, config: Config,max_evals:int):
    space = {
        "learning_rate": hp.choice("learning_rate",np.linspace(0.18, 0.21, 4, dtype=float)),
        "num_leaves": hp.choice('num_leaves',[31,63]),
        "feature_fraction": hp.choice("feature_fraction",[0.9,1.0]),
        "bagging_freq": hp.choice("bagging_freq", [0]),
        "reg_alpha": hp.choice("reg_alpha", np.linspace(0, 0.5, 4, dtype=float)[1:4]),
        "reg_lambda": hp.choice("reg_lambda", np.linspace(1, 30, 4, dtype=float)[0:3]),
        "min_child_weight": hp.choice('min_child_weight', np.linspace(0, 1, 4, dtype=float)[0:3])
    }
    def objective(hyperparams):
        evals_result = {}
        model = lgb.train({**params, **hyperparams}, train_data, 10, valid_data,
                          early_stopping_rounds=10, verbose_eval=False,evals_result=evals_result)
        score = model.best_score["valid_0"][params["metric"]]
        if config['is_classification']:
            score = -score
        return {'loss': score, 'status': STATUS_OK}
    trials = Trials()
    algo = partial(tpe.suggest, n_startup_jobs=5)
    best = fmin(fn=objective, space=space, trials=trials, algo=algo, max_evals=max_evals, verbose=1,
                rstate=np.random.RandomState(1))
    hyperparams = space_eval(space, best)
    log("{:0.4f} {}".format(trials.best_trial['result']['loss'], hyperparams))
    return hyperparams,trials.best_trial['result']['loss']
```

KFOLD: Data Splitting

- Stratification in main_key

```
keytrain = config['keytrain']
mainkey = config['mainkey']
kfkey = keytrain.groupby(mainkey,as_index=False).sum()
kfkey['label'] = kfkey.label.map(kfkey.label.value_counts())
uselabel = []
labellist = sorted(kfkey.label.unique(),reverse=True)
for i in labellist:
    uselabel.append(i)
    if sum(uselabel) > sum(kfkey.label.unique()) * 0.99:break
if len(labellist) > 3 and len(uselabel) < 3:
    uselabel = labellist[:3]
kfkey = kfkey[kfkey.label.isin(uselabel)]
kfkey = kfkey.sample(frac=1,random_state=2019).reset_index(drop=True)
skf = StratifiedKFold(n_splits=5, random_state=2019, shuffle=True)
for train_index, test_index in skf.split(kfkey,kfkey.label):
    test_index = keytrain.loc[keytrain[mainkey].isin(kfkey.loc[test_index,mainkey])].index.tolist()
```

KFOLD: Stratification

- Stratification in main_key.
- StratifiedKFold guarantees the partition of labels is unbiased in random data splitting
- 2 strategies as follows

```
skf = StratifiedKFold(n_splits=5, random_state=2019, shuffle=True)
skflist0 = []
for train_index, test_index in skf.split(keytrain,keytrain.label):
    skflist0.append([train_index, test_index])
skflist1 = []
for train_index, test_index in skf.split(kfkey,kfkey.label):
    test_index = keytrain.loc[keytrain[mainkey].isin(kfkey.loc[test_index,mainkey])].index.tolist()
    train_index = keytrain.loc[~keytrain.index.isin(test_index)].index.tolist()
    skflist1.append([train_index, test_index])
skflist = []
for i in range(5):
    skflist.append(skflist0[i])
    skflist.append(skflist1[i])
```

KFOLD: Stratification

- As the dataset was randomly split into folds, the results of each fold could be unstable. When the hyperparameter searching was finished, for each fold, we ran two random seeds and used the fold with highest AUC as the fold in StratifiedKFold.

```
for i in range(2):
    i = i+50
    skf = StratifiedKFold(n_splits=5, random_state=2019+i, shuffle=True)
    skflist2 = []
    for train_index, test_index in skf.split(keytrain,keytrain.label):
        skflist2.append([train_index, test_index])
    gc.collect()
    train_X = pickle.load(open(trainfilename,'rb'))
    train_index, test_index = skflist2[0]
    valid_data = train_X[test_index]
    valid_data = valid_data
    train_X = train_X[train_index]
    train_X = lgb.Dataset(train_X, label=train_y[train_index])
    valid_data = lgb.Dataset(valid_data, label=train_y[test_index])
    model = lgb.train({**params, **best_hyperparams}, train_X, 12, valid_data,
                      early_stopping_rounds=12, verbose_eval=False)
    score = model.best_score["valid_0"][params["metric"]]
    if score > scorebest:
        skflist0 = skflist2
    del train_X,valid_data
```

Others

- Memory optimization
- 1、float64 requires more memory so astype() is used when feasible. Avoid logical errors when fillna() is used.
- 2、Using pkl for temporary storage. The memory needed by pkl is 50% of that by numpy and the dumping/loading of pkl is quick.
- 3、Using sparse matrix as inputs of lgb. The memory needed by this can be 2-3 times of numpy but the memory is running smoothly during the prediction phase. And the prediction phase is quick.
- 4、Using astype(float32) to transform the data before transforming it to sparse matrix. Checking exception during the transformation so that the data can be fed into the model for training and data in other types (e.g. float64) can processed as well.
- Training techniques：
- 1、Different random seeds to split datasets and merging folds.
- 2、Merging by avg is sufficient enough in practice.