Kaggle - M5 Forecasting - Accuracy

Changjun Zhu
Student ID: 20659766
Group name: ZZL
HKUST
MAFS 6010(U)
czhual@connect.ust.hk

Abstract

This is a course project for MAFS6010U at the Hong Kong University of Science and Technology. In this project, I chose to participate in the Kaggle competition. Thus, in this report, I will introduce the data processing, the process of model building and the final prediction results in detail.

1. Introduction

The link of presentation video and the link of codes collection are as follows:

https://github.com/Goonmdt/kaggle_M5

https://space.bilibili.com/204278913/video

1.1 Background of the problem

In this competition, I will use hierarchical sales data from Walmart, the world's largest company by revenue, to forecast daily sales for the next 28 days. The data, covers stores in three US States (California, Texas, and Wisconsin) and includes item level, department, product categories, and store details. In addition, I could use

explanatory variables such as price, promotions, day of the week, and special events to improve forecasting accuracy.

The available data are stored in *calendar.csv*, *sales_train_validation.csv*, *sell-prices.csv* respectively. Finally, we need to input the prediction results into a new *csv* file just like *sample-submission.csv*.

In this report, I will focus on the ideas and reason of code writing and the results of each step. The complete code and some comments can be seen in the *ipynb* file in the link at the beginning of the report.

1.2 Light GBM

GBDT (gradient boosting decision tree) is an ever-growing model in machine learning. Its main idea is to use the weak classifier (decision tree) iterative training to get the optimal model. The model has the advantages of good training effect and not easy to fit. GBDT is widely used in industry. It is usually used for click through rate prediction, search sorting and other tasks. GBDT is also a lethal weapon in all kinds of data mining competitions. According to statistics, more than half of the championship schemes in Kaggle are based on GBDT.

LightGBM (light gradient boosting machine) is a framework to implement GBDT algorithm, which supports efficient parallel training, and has the following advantages: faster training speed, lower memory consumption, better accuracy, distributed support, and more quickly process massive data.

Compared with XGBoost, LightGBM has made some optimizations: decision tree algorithm based on histogram, leaf wise growth strategy with depth limitation, histogram difference acceleration, direct support for category feature, cache hit rate optimization, sparse feature optimization based on square graph, multi-threaded optimization. Thus, in this problem, I choose LightGBM as my final model.

2 Data processing

Firstly, there is a very common function in the kaggle competition: reduce-mem-usage. It is very useful to reduce the memory consumption of data.

```
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.\
                      format(end_mem, 100 * (start_mem - end_mem) / start_mem))
    return df
```

2.1 Loading data

By using pd.read_csv() function and reduce-mem-usage() function, we read data and convert it to Dataframe format. The data is stored in calendar_df, sell_prices_df and sales_train_validation_df. The effect of the reduce-mem-usage() function is shown as follows:

```
Mem. usage decreased to 130.48 Mb (37.5% reduction)
Sell prices has 6841121 rows and 4 columns
Mem. usage decreased to  0.12 Mb (41.9% reduction)
Calendar has 1969 rows and 14 columns
```

Next, we need to reduce the amount of data. On the one hand, too much data is easy to lead to too long training time, which is not conducive to adjusting parameters. On the other hand, the economic environment and people's living habits will change with time, the use of data with a long time span will easily lead to excessive noise of historical behavior.

Thus, here, we only use the data from January 1, 2015.

```
sell_prices_df.drop(sell_prices_df[sell_prices_df.wm_yr_wk < 11448].index, inplace=True)
sales_train_validation_df.drop(sales_train_validation_df.ix[:,'d_1':'d_1433'].head(0).columns, axis=1, inplace=True)
calendar_df.drop(calendar_df[calendar_df.wm_yr_wk < 11448].index, inplace=True)
```

Then we need to process the notnull item in calendar_df, sales_train_validation_df and sell_prices_df. The columns which need to be processed are： "event_name_1", "event_type_1", "event_name_2", "event_type_2", "item_id", "dept_id", "cat_id", "store_id" and "state_id". Because the data is not pure classified data, we use LabelEncoder() function instead of OneHotEncoder() function.

Similarly, we continue to adopt reduce_ mem_ Use function to reduce memory usage.

```
def encode_categorical(df, cols):
    for col in cols:
        # Leave NaN as it is.
        le = preprocessing.LabelEncoder()
        not_null = df[col][df[col].notnull()]
        df[col] = pd.Series(le.fit_transform(not_null), index=not_null.index)

    return df
```

```
Mem. usage decreased to  0.03 Mb (32.9% reduction)
Mem. usage decreased to 21.46 Mb (81.0% reduction)
Mem. usage decreased to 33.81 Mb (46.4% reduction)
```

2.2 Merging dataframes

Firstly, we need to use pd.melt() function. Columns ['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_id'] do not need to be modified. And we change other columns to 'day' and 'demand'. The results are as follows:

```
    sales_train_validation = pd.melt(sales_train_validation, id_vars =
                                     ['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_id'],
                                     var_name = 'day', value_name = 'demand')
```

```
sales_train_validation.shape: (14635200, 8)
----------------------sales_train_validation--------------------------------
                            id  item_id  dept_id  cat_id  store_id  \
0  HOBBIES_1_001_CA_1_validation     1437        3       1         0
1  HOBBIES_1_002_CA_1_validation     1438        3       1         0
2  HOBBIES_1_003_CA_1_validation     1439        3       1         0
3  HOBBIES_1_004_CA_1_validation     1440        3       1         0
4  HOBBIES_1_005_CA_1_validation     1441        3       1         0

   state_id     day  demand
0         0  d_1434       0
1         0  d_1434       0
2         0  d_1434       0
3         0  d_1434       2
4         0  d_1434       3
```

Next, we need to separate test dataframes into test1 and teat2. Then, we could get product table which is the original sales_train_validation_df without the d_ columns.

After merging with product table, we could get test1and test2 as follows:

```
---------------------------------test1-----------------------------------------
                            id  item_id  dept_id  cat_id  store_id  \
0  HOBBIES_1_001_CA_1_validation     1437        3       1         0
1  HOBBIES_1_002_CA_1_validation     1438        3       1         0
2  HOBBIES_1_003_CA_1_validation     1439        3       1         0
3  HOBBIES_1_004_CA_1_validation     1440        3       1         0
4  HOBBIES_1_005_CA_1_validation     1441        3       1         0

   state_id     day  demand   part
0         0  d_1914       0  test1
1         0  d_1914       0  test1
2         0  d_1914       0  test1
3         0  d_1914       0  test1
4         0  d_1914       0  test1
```

```
---------------------------------test2-----------------------------------------
                            id  item_id  dept_id  cat_id  store_id  \
0  HOBBIES_1_001_CA_1_evaluation     1437        3       1         0
1  HOBBIES_1_002_CA_1_evaluation     1438        3       1         0
2  HOBBIES_1_003_CA_1_evaluation     1439        3       1         0
3  HOBBIES_1_004_CA_1_evaluation     1440        3       1         0
4  HOBBIES_1_005_CA_1_evaluation     1441        3       1         0

   state_id     day  demand   part
0         0  d_1942       0  test2
1         0  d_1942       0  test2
2         0  d_1942       0  test2
3         0  d_1942       0  test2
4         0  d_1942       0  test2
```

Next, we could get data as follows:

```
    data = pd.concat([sales_train_validation, test1, test2], axis = 0)
```

After dropping some features ('weekday', 'wday', 'month', 'year') in calendar. We merge sell_prices, calendar, data together, and get a new data as follows:

(More details about data processing could be seen in the link at the beginning of the article.)

```
--------------------------------data-------------------------
                              id  item_id  dept_id  cat_id  store_id  \
0  HOBBIES_1_001_CA_1_validation     1437        3       1         0
1  HOBBIES_1_002_CA_1_validation     1438        3       1         0
2  HOBBIES_1_003_CA_1_validation     1439        3       1         0
3  HOBBIES_1_004_CA_1_validation     1440        3       1         0
4  HOBBIES_1_005_CA_1_validation     1441        3       1         0

   state_id  demand   part        date  wm_yr_wk  event_name_1  event_type_1  \
0         0       0  train  2015-01-01     11448          18.0           1.0
1         0       0  train  2015-01-01     11448          18.0           1.0
2         0       0  train  2015-01-01     11448          18.0           1.0
3         0       2  train  2015-01-01     11448          18.0           1.0
4         0       3  train  2015-01-01     11448          18.0           1.0

   event_name_2  event_type_2  snap_CA  snap_TX  snap_WI  sell_price
0           NaN           NaN        1        1        0    8.257812
1           NaN           NaN        1        1        0    3.970703
2           NaN           NaN        1        1        0    2.970703
3           NaN           NaN        1        1        0    4.640625
4           NaN           NaN        1        1        0    2.880859
Our final dataset to train has 15488920 rows and 18 columns
```

2.3 Feature engineering

Firstly, rolling demand features:

```
    for val in [28, 29, 30]:
        data[f"shift_t{val}"] = data.groupby(["id"])["demand"].transform(lambda x: x.shift(val))
    for val in [7, 30, 60, 90, 180]:
        data[f"rolling_std_t{val}"] = data.groupby(["id"])["demand"].transform(lambda x: x.shift(28).rolling(val).std())
    for val in [7, 30, 60, 90, 180]:
        data[f"rolling_mean_t{val}"] = data.groupby(["id"])["demand"].transform(lambda x: x.shift(28).rolling(val).mean())

    data["rolling_skew_t30"] = data.groupby(["id"])["demand"].transform( lambda x: x.shift(28).rolling(30).skew())
    data["rolling_kurt_t30"] = data.groupby(["id"])["demand"].transform(lambda x: x.shift(28).rolling(30).kurt())
```

Secondly, we choose some price features:

```
    data['lag_price_t1'] = data.groupby(['id'])['sell_price'].transform(lambda x: x.shift(1))
    data['price_change_t1'] = (data['lag_price_t1'] - data['sell_price']) / (data['lag_price_t1'])
    data['rolling_price_max_t365'] = data.groupby(['id'])['sell_price'].transform(lambda x: x.shift(1).rolling(365).max())
    data['price_change_t365'] = (data['rolling_price_max_t365'] - data['sell_price']) / (data['rolling_price_max_t365'])
    data['rolling_price_std_t7'] = data.groupby(['id'])['sell_price'].transform(lambda x: x.rolling(7).std())
    data['rolling_price_std_t30'] = data.groupby(['id'])['sell_price'].transform(lambda x: x.rolling(30).std())
    data.drop(['rolling_price_max_t365', 'lag_price_t1'], inplace = True, axis = 1)
```

Thirdly, we choose some time features:

```
    data['date'] = pd.to_datetime(data['date'])
    attrs = ["year", "quarter", "month", "week", "day", "dayofweek", "is_year_end", "is_year_start", "is_quarter_end", \
        "is_quarter_start", "is_month_end","is_month_start",
    ]

    for attr in attrs:
        dtype = np.int16 if attr == "year" else np.int8
        data[attr] = getattr(data['date'].dt, attr).astype(dtype)
    data["is_weekend"] = data["dayofweek"].isin([5, 6]).astype(np.int8)
```

3 Model building, training and testing

3.1 Model building and training

As we said before, in this report we decided to choose LGBM as final model.

First of all, we need to separate training dataset and test dataset:

```
    x_train = data[data['date'] <= '2016-03-27']
    y_train = x_train['demand']
    x_val = data[(data['date'] > '2016-03-27') & (data['date'] <= '2016-04-24')]
    y_val = x_val['demand']
    test = data[(data['date'] > '2016-04-24')]
    del data
    gc.collect()
```

Next, we need to set some parameters:

```
    params = {
#         'boosting_type': 'gbdt',
        'metric': 'rmse',
        'objective': 'poisson',
        'n_jobs': -1,
        'seed': 20,
        'learning_rate': 0.1,
        'alpha': 0.1,
        'lambda': 0.1,
        'bagging_fraction': 0.66,
        'bagging_freq': 2,
        'colsample_bytree': 0.77}

    train_set = lgb.Dataset(x_train[features], y_train)
    val_set = lgb.Dataset(x_val[features], y_val)

    del x_train, y_train
```

Finally, we could build the model, train it and save it:

```
    model = lgb.train(params, train_set, num_boost_round = 2000, early_stopping_rounds = 200,
                      valid_sets = [train_set, val_set], verbose_eval = 100)
    joblib.dump(model, 'lgbm_0.sav')
```

num_boost_round is the number of boosting iterations, or the number of residual trees.

early_stopping_rounds = 200, which means if the error iteration of the verification set can't be further reduced within 200 times, the iteration will be stopped.

verbose_eval = 100, which means the evaluation results of the elements in evals will be output in the results, once every five iterations.

It is worth mentioning that we choose learning_rate = 0.1, which is actually a relatively high value. Because of the large amount of data, this choice is to speed up the convergence speed and facilitate us to adjust other parameters.

In terms of parameter adjustment, we introduce gridsearchcv() function in sklearn, and also draw on some settings of other models in the discussion of kaggle competition..

3.2 Model testing

```
    val_pred = model.predict(x_val[features], num_iteration=model.best_iteration)
    val_score = np.sqrt(metrics.mean_squared_error(val_pred, y_val))
    print(f'Our val rmse score is {val_score}')
    y_pred = model.predict(test[features], num_iteration=model.best_iteration)
    test['demand'] = y_pred
```

The results are as follows:

```
Training until validation scores don't improve for 200 rounds
[100]   training's rmse: 2.42896        valid_1's rmse: 2.21782
[200]   training's rmse: 2.35653        valid_1's rmse: 2.18368
[300]   training's rmse: 2.3143 valid_1's rmse: 2.17147
[400]   training's rmse: 2.28636        valid_1's rmse: 2.16097
[500]   training's rmse: 2.2597 valid_1's rmse: 2.1516
[600]   training's rmse: 2.23905        valid_1's rmse: 2.14726
[700]   training's rmse: 2.2219 valid_1's rmse: 2.14589
[800]   training's rmse: 2.20611        valid_1's rmse: 2.14254
[900]   training's rmse: 2.19157        valid_1's rmse: 2.14044
[1000]  training's rmse: 2.17944        valid_1's rmse: 2.13748
[1100]  training's rmse: 2.16892        valid_1's rmse: 2.13646
[1200]  training's rmse: 2.15798        valid_1's rmse: 2.13592
[1300]  training's rmse: 2.14835        valid_1's rmse: 2.13259
[1400]  training's rmse: 2.13901        valid_1's rmse: 2.12953
[1500]  training's rmse: 2.13099        valid_1's rmse: 2.1294
[1600]  training's rmse: 2.12305        valid_1's rmse: 2.12641
[1700]  training's rmse: 2.11457        valid_1's rmse: 2.12655
[1800]  training's rmse: 2.10898        valid_1's rmse: 2.12614
[1900]  training's rmse: 2.10208        valid_1's rmse: 2.12587
Early stopping, best iteration is:
[1752]  training's rmse: 2.11167        valid_1's rmse: 2.12562
Our val rmse score is 2.1256213751985094
```

In general, the results predicted by the model are quite good, which shows that LGBM can be used in this problem. Later, we can further analyze the data, further extract

some important features, appropriately reduce the learning rate, and increase the amount of data to improve the results, which will not be shown here.

3.3 Save results

At last, we need to import the predicted results into a new CSV file: *submission.csv*, in the format of *sample_submission.csv*. Note that here we only import the validation part. As for the evaluation part, we can wait for the official result of 28 days later to import.

```
def predict(test, submission):
    predictions = test[['id', 'date', 'demand']]
    predictions = pd.pivot_table(predictions, index = 'id', columns = 'date', values = 'demand').reset_index()
    predictions.columns = ['id'] + ['F' + str(i + 1) for i in range(28)]

    evaluation_rows = [row for row in submission['id'] if 'evaluation' in row]
    evaluation = submission[submission['id'].isin(evaluation_rows)]

    validation = submission[['id']].merge(predictions, on = 'id')
    final = pd.concat([validation, evaluation])
    final.to_csv('submission.csv', index = False)
```

4 Conclusion

100%: Zhu Changjun (Student ID: 20659766)

In this kaggle competition, I really learned a lot of practical skills. And I also realized a lot of my own shortcomings in the process of communicating with others. I'll put all the files of this contest in the GitHub connection at the beginning of the article. The presentation video is in the second connection.