Code & Data Appendix

"KAE-Informer: A Knowledge Auto-Embedding Informer for Forecasting Long-Term Workloads of Microservices"

We really appreciate the reviewers for carefully reviewing this paper. In this appendix, we provide some additional supporting information about the further experimental results to illustrate the generality of KAE-Informer for the long-term forecasting of QPS in cloud applications. In addition, we provide three Code & Data Appendix to reproduce our experimental results: 1) the 3-week QPS sequences collected from 10 representative large-scale applications which running in a global leading e-commerce company; 2) our code as a demo; 3) a sample of constructed training/testing dataset for an application to run the demo.

1. Uploaded Code& Data Appendix

1.1. Code

1.1.1. Requiremenets

You need to run our experiments with "python 3.8" The packages you need to install are listed as follow:

- matplotlib==3.1.1
- numpy==1.23.1
- pandas==1.4.3
- scikit_learn==0.24.2
- torch==1.8.0
- prophet==1.1
- dtaidistance==2.3.9
- statsmodels==0.12.2
- scipy==1.8.1

please run the command to install the package as:

```
cd KAE-Informer
pip install -r requirements.txt
```

1.1.2. Running the Code

In our supplementary material, all the code are in the folder "KAE-Informer". Including the folder "model" to save the code of modeling the KAE-Informer; and the folder "util" and "exp" to save the code about some tool function and the code which are called to execute the experiments. The users should run two files to reproduce our experiments:

- "KAE-Informer/data_event_search_ts_extractor.py" is used to extract the predictable residual components, the trend & dominant seasonality components and the similar event from the historical data; and construct the training and testing dataset for the selected application.
- "KAE-Informer/main_informer.py" is used to run the experiments to get the prediction results and the evaluation results of metric $MAPE$, MSE and $sTPE$. These results can be found at:

Specifically, to run the experiments, firstly, you need to run the file:

"data_event_search_ts_extractor.py" to construct the train/test datasets as:

```
python data_event_search_ts_extractor.py --app=app8 --start_day=18 --end_day=-1 --test_len=1440 --input_len=1440 --predict_len=60 --event_num=7 --dtw_length=240 --freq=min --sample_rate=1440
```

This command can generate the train dataset with 2820 items and a test set with 1440 items, the meaning of each parameter is:

- --app: selected app
- --test_len: length of the total predicted items
- --start_day: the day start to construct the dataset
- --end_day: the day ends to process the dataset, -1 means to the end of the data (do not change it)

- --predict_len: the steps of Y_L which is predicted by the Informer-architecture at once
- --input_len: the steps of X_L which is inputed to the main Informer architecture
- --event_num: the number of similar events queried from the historical data
- --dtw_length: the length of the sequence to input to the dtw algorithm to query the similar event
- --freq: the freq of the raw data,default min for 1-minute sampling interval
- --sample_rate: the sampling rate,i.e., the number of the sampling points collected per day

Secondly, you can run the experiments by running the "main_informer.py" as:

```
python main_informer.py --gpu=0 --model=kaeinformer --app=app8-example --enc_in=8
--dec_in=8 --c_out=8 --d_model=512 --d_ff=2048 --event_d_model=384 --d_model=512 --d_ff=2048
--event_d_ff=1536 --data=qps --data_item_size=480 --data_type=0 --seq_len=1440 --label_len=240
--pred_len=60 --sample_type=0 --train_epochs=20 --batch_size=32 --patience=6 --freq=t --
loss=huber --loss_beta=0.2 --qvk_kernel_size=5 --event_qvk_kernel_size=3 --event_seq_len=7 -
-event_label_len=2 --event_pred_len=1 --e_layers=3 --d_layers=2 --event_e_layers=2 --event_d_layers=1
--attn=prob --embed=timeF --activation=gelu --learning_rate=0.0002 --loss=huber --event_loss=mse
--root_path=./data/data_train_test/ --checkpoints=KAE/checkpoints/ --result_dir=./results/
--do_predict
```

The parameters are listed as follow:

- --gpu: the id of the used gpu device
- --app: selected app
- --model: model type (default:kaeinformer)
- --enc_in: dimension of the encoderinput matix,i.e., the extracted events num + 1 (extracted events num is set at "--event_num" in data_event_search_ts_extractor.py)
- --dec_in: dimension of the decoder input matix,i.e., the extracted events num + 1 (extracted events num is set at "--event_num" in data_event_search_ts_extractor.py)
- --c_out: the dimension of the Y_{out} matrix for the output of the main Informer networks (equal to dec_in)

- --d_model: dimension of hidden layers in model
- --d_ff: dimension of fcn
- --event_d_model: dimension of hidden layers in the ER-embedding architectures
- --event_d_ff: dimension of the fcn in ER-embedding layers
- --data: the domain of this data, default: qps (do not change)
- --data_item_size: the size of the dataset item (default: if you use the app8-example to run this demo, set it as 480, if you run "data_event_search_ts_extractor.py" to generate the dataset, set it as the item number of the train dataset,i.e., the shape 0 of the "*_train_ts.npy" in "./data/data_train_test/*")
- --data_type: the type of the components input to the main Informer and ER-Embedding layers, i.e. 0 (default), which means train the main Informer and ER-embedding layers with the input of the predictable residual components; 2: train the main Informer and ER-embedding layers with the input of raw data involving all the components.
- --seq_len: input sequence length of Informer encoder, equal to "--input_len" in data_event_search_ts_extractor.py
- --label_len: start token length of Informer decoder, equal to "--dtw_length" in data_event_search_ts_extractor.py
- --pred_len: prediction sequence length, equal to "--predict_len" in data_event_search_ts_extractor.py
- --freq: the sampling rate, default: 't' means freq='min' in data_event_search_ts_extractor.py
- --sample_type: the sample type means if we consider the extracted events in our model, default: 0 means consider it
- --event_seq_len: the extracted events num (extracted events num is set at "--event_num" in data_event_search_ts_extractor.py)
- --event_label_len: the token event num used in the ER-Embedding layers
- --event_pred_len: 1, the predicted event num

- --e_layers: the number of the encoder layers of main Informer
- --d_layers: the number of the decoder layers of main Informer
- --event_e_layers: the number of the encoder layers of ER-embedding layers
- --event_d_layers: the number of the decoder layers of ER-embedding layers
- --attn: attention used in encoder of main Informer, options:[prob, full]
- --embed: time features encoding, options:[timeF, fixed, learned], default: timeF
- --activation: activation function, default: gelu
- --learning_rate: learning rate
- --train_epochs: number of the training epochs, default: 20
- --patience: early stopping patience, default: 6
- --batch_size: the mini batch size, default: 32
- --loss: the loss function for the main Informer architecture, default: huber
- --event_loss: the loss function for the ER-Embedding layers
- --qvk_kernel_size: kernel size for qvk projection in convolution ProbSparse self-attention in the main Informer architecture, default 5
- --event_qvk_kernel_size: kernel size for qvk projection in convolution ProbSparse self-attention in the ER-Embedding layers, default 3
- --root_path=./data/data_train_test/ : the data path of the train/test dataset
- --checkpoints=KAE/checkpoints/ : the path to save the model
- --result_dir=./results/: the path to save the metric and prediction results
- --do_predict: do prediction for this model

Specifically, a user can simply run this file as:

python main_informer.py --gpu=0 --model=kaeinformer --data=qps --app=app8-example --data_item_size=480 --data_type=0 --seq_len=1440 --label_len=240 --train_epochs=20

--sample_type=0 --root_path=./data/data_train_test/
--checkpoints=KAE/checkpoints/ --result_dir=./results/ --do_predict

Please set the selected parameter of "app" and "data_item_size" if you want to choose the dataset you construct by self through running the "data_event_search_ts_extractor.py".

Finally, if you set "--do_predict" when running "main_informer.py", you can generate the prediction results and the metric for the selected app, which are saved at:

It shoule be noted that the "app8-example" dataset only can be used to show the training and predicting process, since the size of this dataset is too small, the performance of the trained model show the extremely obvious performance degradation. For more details, please read the "Readme.md" file in the "KAE-Informer" folder. It shoule be noted that our example"app8-example" dataset only can be used to show the training and predicting process. Since the size of this dataset is too small, the performance of the trained model show the extremely obvious performance degradation.

1.2. Data

1.3. Raw Data

All the 10 raw ".csv" file of the QPS sequence collected from 10 cloud application for 3 weeks are upload in the folder "KAE-Informer/data/data_submit". For each of 10 applications (app0 $\sim$ app9), 30240 items are collected with the sampling interval of 1 minute.

1.3.1. Train and Test Dataset

Users can run the file "data_event_search_ts_extractor.py" to generate the train & test datasets for the selected application. For convenience, we give a example of app8 (a front-end application) called "app8-example". This example involves 1) a train dataset consisting of 480 items for 8 hours and 2) a corresponding test dataset consisting of 60 items for 1 hours. All the examples can be found at: "KAE-Informer/data/data_train_test/app8-example/". Including the training or test ".npy" files for:

- "*_train/test_prophet_ts.npy": trend & dominant seasonality components in the train/test dataset
- "*_train/test_ts.npy": predictable residual components to train/test the main Informer architecture

- "*_train/test_event.npy": predictable residual components to train/test the ER-embedding Informer architectures
- "*_train/test_base_ts.npy": the ground-truth of the raw data for train/test datasets