Useful notebooks:

- Preprocessing : https://www.kaggle.com/code/motono0223/js24-preprocessing-create-lags
- Training (XGB) : https://www.kaggle.com/code/motono0223/js24-train-gbdt-model-with-lags-singlemodel
  - trained XGB model : https://www.kaggle.com/datasets/motono0223/js24-trained-gbdt-model
- Training (NN): https://www.kaggle.com/code/voix97/jane-street-rmf-training-nn
  - trained NN model : https://www.kaggle.com/datasets/voix97/js-xs-nn-trained-model
- Inference of NN : https://www.kaggle.com/code/voix97/jane-street-rmf-nn-with-pytorch-lightning
- Inference of NN+XGB: **this notebook** https://www.kaggle.com/code/voix97/jane-street-rmf-nn-xgb
- EDA(1) : https://www.kaggle.com/code/motono0223/eda-jane-street-real-time-market-data-forecasting
- EDA(2) : https://www.kaggle.com/code/motono0223/eda-v2-jane-street-real-time-market-forecasting

```
import pandas as pd
import polars as pl
import numpy as np
import os, gc
from tqdm.auto import tqdm
from matplotlib import pyplot as plt
import pickle
import seaborn

import torch
import torch.nn as nn
import torch.nn.functional as F
from pytorch_lightning import (LightningDataModule, LightningModule,
Trainer)
from pytorch_lightning.callbacks import EarlyStopping,
ModelCheckpoint, Timer

import pandas as pd
import numpy as np
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader


from sklearn.metrics import r2_score
from lightgbm import LGBMRegressor
import lightgbm as lgb
```

```
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.ensemble import VotingRegressor

import warnings
warnings.filterwarnings('ignore')
pd.options.display.max_columns = None

import kaggle_evaluation.jane_street_inference_server
```

NN + XGB inference

Configurations

```
class CONFIG:
    seed = 42
    target_col = "responder_6"
    # feature_cols = ["symbol_id", "time_id"] + [f"feature_{idx:02d}"
for idx in range(79)]+ [f"responder_{idx}_lag_1" for idx in range(9)]
    feature_cols = [f"feature_{idx:02d}" for idx in range(79)]+
[f"responder_{idx}_lag_1" for idx in range(9)]

    model_paths = [

#"/kaggle/input/js24-train-gbdt-model-with-lags-singlemodel/result.pkl
",
        #"/kaggle/input/js24-trained-gbdt-model/result.pkl",
        "/kaggle/input/js-xs-nn-trained-model",
        "/kaggle/input/js-with-lags-trained-xgb/result.pkl",
    ]
```

Load preprocessed data (to calculate CV)

```
valid = pl.scan_parquet(

f"/kaggle/input/js24-preprocessing-create-lags/validation.parquet/"
).collect().to_pandas()
```

Load model

```
xgb_model = None
model_path = CONFIG.model_paths[1]
```

```
with open( model_path, "rb") as fp:
    result = pickle.load(fp)
    xgb_model = result["model"]

xgb_feature_cols = ["symbol_id", "time_id"] + CONFIG.feature_cols

# Show model
display(xgb_model)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device='cuda',
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05,
max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=200, n_gpus=2,
n_jobs=None,
             num_parallel_tree=None, ...)
```

```
# Custom R2 metric for validation
def r2_val(y_true, y_pred, sample_weight):
    r2 = 1 - np.average((y_pred - y_true) ** 2, weights=sample_weight)
/ (np.average((y_true) ** 2, weights=sample_weight) + 1e-38)
    return r2


class NN(LightningModule):
    def __init__(self, input_dim, hidden_dims, dropouts, lr,
weight_decay):
        super().__init__()
        self.save_hyperparameters()
        layers = []
        in_dim = input_dim
        for i, hidden_dim in enumerate(hidden_dims):
            layers.append(nn.BatchNorm1d(in_dim))
            if i > 0:
                layers.append(nn.SiLU())
            if i < len(dropouts):
                layers.append(nn.Dropout(dropouts[i]))
            layers.append(nn.Linear(in_dim, hidden_dim))
            # layers.append(nn.ReLU())
            in_dim = hidden_dim
        layers.append(nn.Linear(in_dim, 1))  # 输出层
```

```
        layers.append(nn.Tanh())
        self.model = nn.Sequential(*layers)
        self.lr = lr
        self.weight_decay = weight_decay
        self.validation_step_outputs = []

    def forward(self, x):
        return 5 * self.model(x).squeeze(-1)  # 输出为一维张量

    def training_step(self, batch):
        x, y, w = batch
        y_hat = self(x)
        loss = F.mse_loss(y_hat, y, reduction='none') * w  # 考虑样本权
重
        loss = loss.mean()
        self.log('train_loss', loss, on_step=False, on_epoch=True,
batch_size=x.size(0))
        return loss

    def validation_step(self, batch):
        x, y, w = batch
        y_hat = self(x)
        loss = F.mse_loss(y_hat, y, reduction='none') * w
        loss = loss.mean()
        self.log('val_loss', loss, on_step=False, on_epoch=True,
batch_size=x.size(0))
        self.validation_step_outputs.append((y_hat, y, w))
        return loss

    def on_validation_epoch_end(self):
        """Calculate validation WRMSE at the end of the epoch."""
        y = torch.cat([x[1] for x in
self.validation_step_outputs]).cpu().numpy()
        if self.trainer.sanity_checking:
            prob = torch.cat([x[0] for x in
self.validation_step_outputs]).cpu().numpy()
        else:
            prob = torch.cat([x[0] for x in
self.validation_step_outputs]).cpu().numpy()
            weights = torch.cat([x[2] for x in
self.validation_step_outputs]).cpu().numpy()
            # r2_val
            val_r_square = r2_val(y, prob, weights)
            self.log("val_r_square", val_r_square, prog_bar=True,
on_step=False, on_epoch=True)
        self.validation_step_outputs.clear()

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.lr,
weight_decay=self.weight_decay)
```

```
        scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
factor=0.5, patience=5,

verbose=True)
        return {
            'optimizer': optimizer,
            'lr_scheduler': {
                'scheduler': scheduler,
                'monitor': 'val_loss',
            }
        }

    def on_train_epoch_end(self):
        if self.trainer.sanity_checking:
            return
        epoch = self.trainer.current_epoch
        metrics = {k: v.item() if isinstance(v, torch.Tensor) else v
for k, v in self.trainer.logged_metrics.items()}
        formatted_metrics = {k: f"{v:.5f}" for k, v in
metrics.items()}
        print(f"Epoch {epoch}: {formatted_metrics}")

N_folds = 5
# 加载最佳模型
models = []
for fold in range(N_folds):
    checkpoint_path = f"{CONFIG.model_paths[0]}/nn_{fold}.model"
    model = NN.load_from_checkpoint(checkpoint_path)
    models.append(model.to("cuda:0"))
```

CV Score

```
X_valid = valid[ xgb_feature_cols ]
y_valid = valid[ CONFIG.target_col ]
w_valid = valid[ "weight" ]
y_pred_valid_xgb = xgb_model.predict(X_valid)
valid_score = r2_score( y_valid, y_pred_valid_xgb,
sample_weight=w_valid )
valid_score
```

```
0.011719618025724632
```

```
X_valid = valid[ CONFIG.feature_cols ]
y_valid = valid[ CONFIG.target_col ]
w_valid = valid[ "weight" ]
X_valid = X_valid.fillna(method = 'ffill').fillna(0)
X_valid.shape, y_valid.shape, w_valid.shape
```

```
((1082224, 88), (1082224,), (1082224,))
```

```
y_pred_valid_nn = np.zeros(y_valid.shape)
with torch.no_grad():
    for model in models:
        model.eval()
        y_pred_valid_nn +=
model(torch.FloatTensor(X_valid.values).to("cuda:0")).cpu().numpy() /
len(models)
valid_score = r2_score( y_valid, y_pred_valid_nn,
sample_weight=w_valid )
valid_score
```

```
0.010941769867835682
```

```
y_pred_valid_ensemble = 0.5 * (y_pred_valid_xgb + y_pred_valid_nn)
valid_score = r2_score( y_valid, y_pred_valid_ensemble,
sample_weight=w_valid )
valid_score
```

```
0.012029819687069843
```

```
del valid, X_valid, y_valid, w_valid
gc.collect()
```

```
273
```

There seems to be bug in official code, can only submit polars dataframe

```
lags_ : pl.DataFrame | None = None

def predict(test: pl.DataFrame, lags: pl.DataFrame | None) ->
pl.DataFrame | pd.DataFrame:
    global lags_
    if lags is not None:
        lags_ = lags

    predictions = test.select(
        'row_id',
        pl.lit(0.0).alias('responder_6'),
    )
    symbol_ids = test.select('symbol_id').to_numpy()[:, 0]

    if not lags is None:
        lags = lags.group_by(["date_id", "symbol_id"],
maintain_order=True).last() # pick up last record of previous date
        test = test.join(lags, on=["date_id", "symbol_id"],
how="left")
    else:
        test = test.with_columns(
            ( pl.lit(0.0).alias(f'responder_{idx}_lag_1') for idx in
```

```
range(9) )
        )

    preds = np.zeros((test.shape[0],))
    preds += xgb_model.predict(test[xgb_feature_cols].to_pandas()) / 2
    test_input = test[CONFIG.feature_cols].to_pandas()
    test_input = test_input.fillna(method = 'ffill').fillna(0)
    test_input = torch.FloatTensor(test_input.values).to("cuda:0")
    with torch.no_grad():
        for i, nn_model in enumerate(tqdm(models)):
            nn_model.eval()
            preds += nn_model(test_input).cpu().numpy() / 10
    print(f"predict> preds.shape =", preds.shape)

    predictions = \
    test.select('row_id').\
    with_columns(
        pl.Series(
            name   = 'responder_6',
            values = np.clip(preds, a_min = -5, a_max = 5),
            dtype  = pl.Float64,
        )
    )

    # The predict function must return a DataFrame
    assert isinstance(predictions, pl.DataFrame | pd.DataFrame)
    # with columns 'row_id', 'responer_6'
    assert list(predictions.columns) == ['row_id', 'responder_6']
    # and as many rows as the test data.
    assert len(predictions) == len(test)

    return predictions

def train_epoch(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    predictions = []
    actuals = []

    for batch_idx, (data, target) in enumerate(tqdm(train_loader)):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()

        output = model(data)
        loss = criterion(output, target)

        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```

```
        predictions.extend(output.cpu().detach().numpy())
        actuals.extend(target.cpu().numpy())

    epoch_loss = running_loss / len(train_loader)
    return epoch_loss, predictions, actuals

def plot_training_results(train_losses, val_losses, predictions,
actuals, save_path=None):
    plt.style.use('seaborn')
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,
12))

    # Loss curves
    epochs = range(1, len(train_losses) + 1)
    ax1.plot(epochs, train_losses, 'b-', label='Training Loss')
    ax1.plot(epochs, val_losses, 'r-', label='Validation Loss')
    ax1.set_title('Training and Validation Loss')
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Loss')
    ax1.legend()

    # Prediction vs Actual scatter plot
    ax2.scatter(actuals, predictions, alpha=0.5)
    ax2.plot([min(actuals), max(actuals)], [min(actuals),
max(actuals)], 'r--')
    ax2.set_title('Predictions vs Actuals')
    ax2.set_xlabel('Actual Values')
    ax2.set_ylabel('Predicted Values')

    # Prediction distribution
    sns.histplot(predictions, ax=ax3, bins=50)
    ax3.set_title('Prediction Distribution')
    ax3.set_xlabel('Predicted Values')
    ax3.set_ylabel('Count')

    # Error distribution
    errors = np.array(predictions) - np.array(actuals)
    sns.histplot(errors, ax=ax4, bins=50)
    ax4.set_title('Error Distribution')
    ax4.set_xlabel('Prediction Error')
    ax4.set_ylabel('Count')

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path)

    return fig
```

When your notebook is run on the hidden test set, inference_server.serve must be called within 15 minutes of the notebook starting or the gateway will throw an error. If you need more than 15 minutes to load your model you can do so during the very first `predict` call, which does not have the usual 10 minute response deadline.

inference_server = kaggle_evaluation.jane_street_inference_server.JSInferenceServer(predict)

if os.getenv('KAGGLE_IS_COMPETITION_RERUN'): inference_server.serve() else: inference_server.run_local_gateway(('/kaggle/input/jane-street-realtime-marketdata-forecasting/test.parquet', '/kaggle/input/jane-street-realtime-marketdata-forecasting/lags.parquet',))