Summary

背景

1.比赛介绍

- 预测金融市场的投资回报率
- 是一个有监督学习的回归问题
- 数据特征和标签举办方已经给出，其中特征是匿名特征
- 评估标准是 皮尔逊相关系数[1]

2.比赛要求

- 代码以 Notebooks 格式提交
- 代码在举办方指定的环境下运行
  - GPU 免费使用 36 小时
  - RAM 为 13G
  - 代码运行时是断网状态
- 代码运行时间不得超过 9 小时

数据集

1.下载地址：

- https://www.kaggle.com/competitions/ubiquant-market-prediction/data

2.数据结构

表格类型数据（3141410 row * 304 $columns$）

row_id	time_id	investment_id	target	f_0	f_1	f_2	f_3
3141410 unique values	0 1219	0 3773	-9.42 12	-17.7 7.85	-6.58 8.71	-8.64 8.01	-18 47.1
0_1	0	1	-0.30087492	0.9325727820396423	0.11369125545024872	-0.4022061228752136	0.3783864378929138
0_2	0	2	-0.23104009	0.8108017444610596	-0.5141154527664185	0.7423681616783142	-0.6166727542877197
0_6	0	6	0.56880707	0.39397355914115906	0.6159366369247437	0.5678060054779053	-0.6079627275466919
0_7	0	7	-1.0647804	-2.3435354232788086	-0.01187007874250412	1.8746063709259033	-0.6063464283943176
0_8	0	8	-0.53194016	0.8420571684837341	-0.2629927098751068	2.3300297260284424	-0.5834221839904785
0_9	0	9	1.5059036	0.6088548302650452	1.3693046569824219	-0.7615153193473816	0.8658595681190491
0_10	0	10	-0.26073086	-1.8637969493865967	0.1136912852525711	1.5738638639450073	-0.5984325408935547
0_12	0	12	-0.469207	0.40895360708236694	-0.7652381062507629	0.26142969727516174	-0.5918948650360107
0_13	0	13	0.09452465	0.8611865043640137	2.373795509338379	-1.1489765644073486	0.7522054314613342
0_14	0	14	-0.25112018	-2.476555109024048	0.23925258219242096	2.222353458404541	-0.5822759866714478

其中：

row_id：每行数据的唯一标识符

time_id：有序的时间点，但间隔不是恒定的，共 1211 个唯一值

investment_id：一支投资（股票）的标识符，共 3579 个唯一值

f_0 - f_299：共 300 个匿名特征

target：最终需要预测的目标，即投资回报率

3.数据分析

参考：EDA- target analysis[2]

方法

一共使用了四种模型去做预测

- LinearRegression Model
- XGBoost Model
- DNN Model
- TabNet Model

1.LinearRegression Model

利用 300 个特征做简单的多元线性回归预测，熟悉一下回归任务的流程

- 核心代码

```
# 这里只展示部分核心代码

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from scipy.stats import pearsonr


kf  = KFold(n_splits=5)
models = []
scores = []
for i,(train_index,val_index) in enumerate(kf.split(x_data)):
    print('-'*50)
    print(f'round{i}')

    x_train,y_train = x_data.iloc[train_index],y_data.iloc[train_index]
    x_val,y_val = x_data.iloc[val_index],y_data.iloc[val_index]

    model = LinearRegression()
    model.fit(x_train,y_train)
    models.append(model)
    joblib.dump(model,f'round_{i}.pkl')

    y_pred = model.predict(x_val)
    rmse = np.sqrt(mean_squared_error(y_pred,y_val))
    corr = pearsonr(y_pred,y_val)[0]

    print(f'RMSE: {rmse},\t Pearson correlation score: {corr}')

print(f'相关系数的均值: {np.mean(scores, axis=0)}')
```

- 效果

```
rmse: 0.9293    相关系数: 0.1102    得分: 0.108    排名: 2498/2893
```

2.XGBoost Model

使用经典的处理表格数据的树模型，看看其效果如何

- 核心代码

```python
from sklearn.model_selection import train_test_split
import xgboost as xgb

X_train,X_test,y_train,y_test = train_test_split(X_train, y_train, test_size=0.2, random_state=1

model1 = xgb.XGBRegressor(
    n_estimators=500,
    learning_rate=0.05,
    max_depth=12,
    subsample=0.9,
    colsample_bytree=0.7,
    #colsample_bylevel=0.75,
    missing=-999,
    random_state=1111,
    tree_method='gpu_hist'
    )

model1.fit(X_train, y_train, early_stopping_rounds=10, eval_set=[(X_test, y_test)], verbose=1)
```

- 效果

```
rmse: 0.89553    相关系数: 0.1534    得分: 0.138    排名: 1800/2893
```

3.DNN Model

使用最简单的神经网络DNN去尝试处理表格数据

- 核心代码

```python
def pythonash_model():
    inputs_ = tf.keras.Input(shape=[df_x.shape[1]])
    x = tf.keras.layers.Dense(64, kernel_initializer='he_normal',activation='relu')(inputs_)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)

    x = tf.keras.layers.Dense(128, kernel_initializer='he_normal',activation='relu')(x)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)

    x = tf.keras.layers.Dense(256, kernel_initializer='he_normal',activation='relu')(x)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)

    x = tf.keras.layers.Dense(512, kernel_initializer='he_normal',activation='relu')(x)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)

    x = tf.keras.layers.Dense(256, kernel_initializer='he_normal',activation='relu')(x)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)
    drop = tf.keras.layers.Dropout(0.4)(x)

    x = tf.keras.layers.Dense(128, kernel_initializer='he_normal',activation='relu')(x)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)

    x = tf.keras.layers.Dense(8, kernel_initializer='he_normal',activation='relu')(x)
    batch = tf.keras.layers.BatchNormalization()(x)
    leaky = tf.keras.layers.LeakyReLU(0.1)(batch)
    drop = tf.keras.layers.Dropout(0.4)(X)

    outputs_ = tf.keras.layers.Dense(1)(x)

    model = tf.keras.Model(inputs=inputs_, outputs=outputs_)

    rmse = tf.keras.metrics.RootMeanSquaredError()

    # learning_sch = tf.keras.optimizers.schedules.ExponentialDecay(
    #     initial_learning_rate=0.003,
    #     decay_steps=9700,
    #     decay_rate=0.98)
    # adam = tf.keras.optimizers.Adam(learning_rate=learning_sch)

    model.compile(loss='mse', metrics=rmse, optimizer=tf.optimizers.Adam(0.001))
    return model


pythonash_model().summary()
from tensorflow.keras.utils import  plot_model
```

```
plot_model(pythonash_model(),to_file = '/workspace/xxl/modle.png',show_shapes=True,expand_nestec

kfold_generator = KFold(n_splits =5, shuffle=True, random_state = 2022)
print(kfold_generator)
callbacks = tf.keras.callbacks.ModelCheckpoint('pythonash_model.h5', save_best_only=True)
for train_index, val_index in kfold_generator.split(df_x, df_y):
    train_x, train_y = df_x.iloc[train_index], df_y.iloc[train_index]
    val_x, val_y = df_x.iloc[val_index], df_y.iloc[val_index]
    tf_train = tf.data.Dataset.from_tensor_slices((train_x, train_y)).shuffle(2022).batch(1024,c
        1)
    tf_val = tf.data.Dataset.from_tensor_slices((val_x, val_y)).batch(1024,drop_remainder=False)
        1)
    model = pythonash_model()
    model.fit(tf_train, callbacks=callbacks, epochs=5,  #### change the epochs into more numbers
              validation_data=(tf_val))
    corr = pearsonr(model.predict(tf_val).ravel(),val_y.values.ravel())
    print(corr)
```

- 效果

```
rmse: 0.9164    相关系数: 0.1265    得分: 0.124    排名: 2356/2893
```

4.TabNet Model[3]

将 NN 和 DT 优势结合起来,一种专门用来处理表格数据的网络模型

- 核心代码

```python
# 完整代码链接：https://www.kaggle.com/code/cylykryatsl/tabnet4

from pytorch_tabnet.tab_model import TabNetRegressor
from pytorch_tabnet.metrics import Metric

clf = TabNetRegressor(cat_emb_dim=1, cat_idxs= [i for i, f in enumerate(features) if f in ['inve
n_a=16,n_d=16,gamma =1.4690246460970766,optimizer_fn = Adam,scheduler_params = dict(T_0=200, T_m
        scheduler_fn = CosineAnnealingWarmRestarts)

class PearsonCorrelation(Metric):
    def __init__(self):
        self._name = 'pearson_corr'
        self._maximize = True

    def __call__(self, x, y):
        x = x.squeeze()
        y = y.squeeze()
        x_diff = x - np.mean(x)
        y_diff = y - np.mean(y)
        return np.dot(x_diff, y_diff)/(np.sqrt(sum(x_diff**2))*np.sqrt(sum(y_diff**2)))

clf.fit(
    X_train=X_train, y_train=y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    eval_name=['train', 'test'],
    eval_metric=['rmse','pearson_corr'],
    max_epochs=30,
    patience=50,
    batch_size=1024, virtual_batch_size=256,
    num_workers=0,
    drop_last=False,
)
```

- 效果

这里做了几组对比实验，如下所示：

类别特征	Batch_size	virtual_batch_size	Epochs	得分
无	1024	128	20	0.1304
investment_id	1024	128	30	0.1310
investment_id、time_id	1024	128	30	0.1215
investment_id、time_id	1024	128	20	0.1464
investment_id、time_id	1024	128	60	0.1171
investment_id、time_id	2048	128	30	0.1289
investment_id、time_id	2048	128	20	0.1290
investment_id、time_id	2048	256	30	0.1211
investment_id、time_id	2048	256	20	0.1328
investment_id、time_id	1024	256	30	0.1370
investment_id、time_id	1024	256	20	0.1213

```
# 效果最好的一组实验结果如下：
rmse: 0.90070    相关系数: 0.1738    得分: 0.1464    排名: 1465/2893
```

心得

虽然名次不好看，但是知道了怎么去查资料解决自己的问题，同时也意识到了提前规划项目进程的重要性。总之，收获很多。

1. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient ↩
2. https://www.kaggle.com/code/lucamassaron/eda-target-analysis ↩
3. TabNet: Attentive Interpretable Tabular Learning ↩