TabNet的使用

安装

pip命令安装

```
pip install pytorch-tabnet
# pip install tensorflow-tabnet
```

github源码安装

```
'''
源码下载地址：https://github.com/dreamquark-ai/tabnet
使用源码安装，加入了一些自定义模块
Step1：将网站中的 pytorch_tanbet 文件夹下载到本地
Step2：创建虚拟环境
Step3：将pytorch_tabnet 文件夹复制到虚拟环境的软件包目录
'''
```

- **创建虚拟环境**

```
# 在linux服务器依次运行：

conda create -n tabnet python=3.7
conda env list

# 显示结果如下：
base                  *    /workspace/miniconda3
tabnet                   /workspace/miniconda3/envs/tabnet

# /workspace/miniconda3/envs/tabnet 是虚拟环境所在的位置
```

- **安装 Pytorch 和 TabNet**

```
# 激活虚拟环境
conda activate tabnet

# 安装pytroch
conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch

# 将 pytorch_tabnet 库移动到虚拟环境软件包目录
sudo cp  */pytorch_tabnet  /workspace/miniconda3/envs/tabnet/lib/python3.7/site-packages/pytorch

#  * 是你下载的 pytorch_tabnet 源码所在目录
```

使用

Import Library

```
from pytorch_tabnet.tab_model import TabNetClassifier

import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
np.random.seed(0)


import os
import wget
from pathlib import Path
import shutil
import gzip

from matplotlib import pyplot as plt
```

Data Processing

- **划分数据集**

```
# 切分Train、Val、Test集

if "Set" not in train.columns:
    train["Set"] = np.random.choice(["train", "valid", "test"], p =[.8, .1, .1], size=(train.sha

# np.random.choice() 函数可参考：https://numpy.org/doc/stable/reference/random/generated/numpy.ra


# 返回Train、Val、Test集各自对应的索引

train_indices = train[train.Set=="train"].index
valid_indices = train[train.Set=="valid"].index
test_indices = train[train.Set=="test"].index

# 通过索引返回Train、Val、Test集

X_train = train[features].values[train_indices]
y_train = train[target].values[train_indices].reshape(-1, 1)

X_valid = train[features].values[valid_indices]
y_valid = train[target].values[valid_indices].reshape(-1, 1)

X_test = train[features].values[test_indices]
y_test = train[target].values[test_indices].reshape(-1, 1)
```

- **离散（类别）数据得到编码**

```
# 数据编码，将离散型数据(如类别数据)进行整数编码

categorical_columns = []
categorical_dims =  {}

for col in train.columns[train.dtypes == object]:
    print(col, train[col].nunique())
    l_enc = LabelEncoder()
    train[col] = train[col].fillna("VV_likely")
    train[col] = l_enc.fit_transform(train[col].values)

# LabelEncoder() 函数可参考：https://scikit-learn.org/stable/modules/generated/sklearn.preprocess

    categorical_columns.append(col)
    categorical_dims[col] = len(l_enc.classes_)
```

- **预设置模型参数（后面传入参数需要用到）**

```
# 预设置参数

unused_feat = ['Set']

# 返回的是特征列表
features = [ col for col in train.columns if col not in unused_feat+[target]]

# 返回的是类别特征的索引列表
cat_idxs = [ i for i, f in enumerate(features) if f in categorical_columns]

# 返回的是 类别特征的唯一值得数量 列表
cat_dims = [ categorical_dims[f] for i, f in enumerate(features) if f in categorical_columns]
```

TabNetClassifier、TabNetRegressor

- **调用分类（回归）模块**

```
# 调用 TabNet 的分类（回归）模块
from pytorch_tabnet.tab_model import TabNetClassifier
# from pytorch_tabnet.tab_model import TabNetRegressor
```

- **设置模型参数**

```
# 设定模型参数
clf = TabNetClassifier(
    n_d=64, n_a=64, n_steps=5,
    gamma=1.5, n_independent=2, n_shared=2,
    cat_idxs=cat_idxs,
    cat_dims=cat_dims,
    cat_emb_dim=1,
    lambda_sparse=1e-4, momentum=0.3, clip_value=2.,
    optimizer_fn=torch.optim.Adam,
    optimizer_params=dict(lr=2e-2),
    scheduler_params = {"gamma": 0.95,
                     "step_size": 20},
    scheduler_fn=torch.optim.lr_scheduler.StepLR, epsilon=1e-15
)

# 参数的解释：https://github.com/dreamquark-ai/tabnet#model-parameters
```

- **传入数据集**

```
# 传入数据
clf.fit(
    X_train=X_train, y_train=y_train,
    eval_set=[(X_train, y_train), (X_valid, y_valid)],
    eval_name=['train', 'valid'],
    eval_metric=['mae', 'rmse', 'mse'],
    max_epochs=max_epochs,
    patience=50,
    batch_size=1024, virtual_batch_size=128,
    num_workers=0,
    drop_last=False,
    augmentations=aug, #aug
)

# 参数的解释：https://github.com/dreamquark-ai/tabnet#fit-parameters
```

- **预测**

```
# 预测时，可以直接调用 predict 函数
preds = clf.predict(X_test)

y_true = y_test
test_score = mean_squared_error(y_pred=preds, y_true=y_true)

print(f"BEST VALID SCORE FOR  : {clf.best_cost}")
print(f"FINAL TEST SCORE FOR  : {test_score}")
```

- **保存和加载模型**

```
# 保存 tabnet 模型
saving_path_name = "./tabnet_model_test_1"
saved_filepath = clf.save_model(saving_path_name)

# 加载保存的模型
loaded_clf = TabNetRegressor()
loaded_clf.load_model(saved_filepath)

# 用模型去做预测
loaded_preds = loaded_clf.predict(X_test)
loaded_test_mse = mean_squared_error(loaded_preds, y_test)
```

问题

- **没有理解的参数**

```
[1] n_independent : int (default=2)

# Number of independent Gated Linear Units layers at each step. Usual values range from 1 to 5.

[2] n_shared : int (default=2)

# Number of shared Gated Linear Units at each step Usual values range from 1 to 5.

[3] clip_value : float (default None)

# If a float is given this will clip the gradient at clip_value.

[4] scheduler_fn : torch.optim.lr_scheduler (default=None)

# Pytorch Scheduler to change learning rates during training.

[5] scheduler_params : dict

# Dictionnary of parameters to apply to the scheduler_fn. Ex : {"gamma": 0.95, "step_size": 10}
```

- **SMOTE数据增强**

```
# 使用 SMOTE 进行数据增强，还没有详细了解SMOTE原理
from pytorch_tabnet.augmentations import RegressionSMOTE
aug = RegressionSMOTE(p=0.2)
augmentations=aug
```

代码链接

Forest-Cover-Type