RecLab: A place for recommendation algorithms test

RecLab is a toolkit designed to streamline the development of recommendation system demos and experiments. It aims to provide a user-friendly environment for quickly building, evaluating, and experimenting with various recommendation algorithms, datasets, and feature engineering workflows.

Key Features

- Easy Recommendation Demo Setup
- Built-in Datasets with Lazy and Chunked Loading
- Feature Engineering Tools
- Model Building and Classic Algorithms

Getting Started

Installation

```
pip install reclab
```

Loading a Dataset

```
from reclab.datasets import BLOG_REC  # Example dataset loader

# Initialize the dataset; no data is loaded yet
test_ds = BLOG_REC()
# Show the table list in this dataset
test_ds.list_tables()
```

Data Full Read

Data full read will load all the table to storage. This works good for small tables.

```
full_data_author = test_ds.get_table_data('Author Data.csv')
```

Data Streaming Read

Data streaming will read data from a iterable file stream object, which only load the data when the data is used. This load strategy works good for big tables.

Besides, the dataset iter object is adapt to DataLoader in torch.utils.data, so you can definitely use this module just as a dataset reader and convert it to DataLoader without changing your original code.

```
from torch.utils.data import DataLoader
author_loader = test_ds.iter_loader('Author Data.csv')
author_loader_slice = test_ds.iter_loader('Author Data.csv', start = 1, end =
3)
author_loader_chunk = test_ds.iter_loader('Author Data.csv', chunk_size = 3)
loader = DataLoader(author_loader_slice, batch_size=None)
```

Streaming Feature Engineering

FeatureStreaming is a streaming data feature engineering class, where you should assign a DataLoader, the table header and the batch_size. The fts object will create a window computing unit, which can be accessed as a pandas.dataframe. You can do your data engineering here.

However, if you want to access the full table before processing the features, kindly use the *FeatureStreaming.process_all* method in the class to handle all the data in the DataLoader.

As you can see, this class have good adaption to our dataset sections.

```
from reclab.data.test_autoFE import FeatureStreaming
bk = BOOK()
loader = DataLoader(bk.iter_loader('Books.csv', start = 1, end = 201))
header = bk.get_table_header('Books.csv')
fts = FeatureStreaming(loader, 100, header)
while True:
    batch_df = fts.process()
    if batch_df.empty:
        break
    print(batch_df)
```

Feature Selection

Here I implement the core function of package of *nni* by Microsoft, the core algorithm used is proposed in paper Feature Gradients: Scalable Feature Selection via Discrete Relaxation. This is an efficient search algorithm based on gradient.

```
from reclab.data.gradientSelector import FeatureGradientSelector
...
selector = FeatureGradientSelector(n_features=10)
selector.fit(X, y)
selected_features = selector.get_features(indices=True)
X_train_selected = X_train.iloc[:, selected_features]
...
```

I tested the performances on the breast cancer dataset, which is a classic medium size dataset, the reduced features still made great classification acc(perhaps it's just the SVM...)

```
The Original feature names are:
['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean
smoothness', 'mean compactness', 'mean concavity', 'mean concave points', 'mean
symmetry', 'mean fractal dimension', 'radius error', 'texture error',
'perimeter error', 'area error', 'smoothness error', 'compactness error',
'concavity error', 'concave points error', 'symmetry error', 'fractal dimension
error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area',
'worst smoothness', 'worst compactness', 'worst concavity', 'worst concave
points', 'worst symmetry', 'worst fractal dimension']

The Selected feature names are:
['mean perimeter', 'radius error', 'worst smoothness', 'texture error', 'mean
radius', 'worst radius', 'worst concave points', 'mean fractal dimension',
'compactness error', 'mean concavity']

The acc trained by original features on SVM: 0.9521276595744681
The acc trained by selected features on SVM: 0.9202127659574468
```

Train and Evaluatin

Version

0.0.1
still working for a released version!...