作业

作用目的:

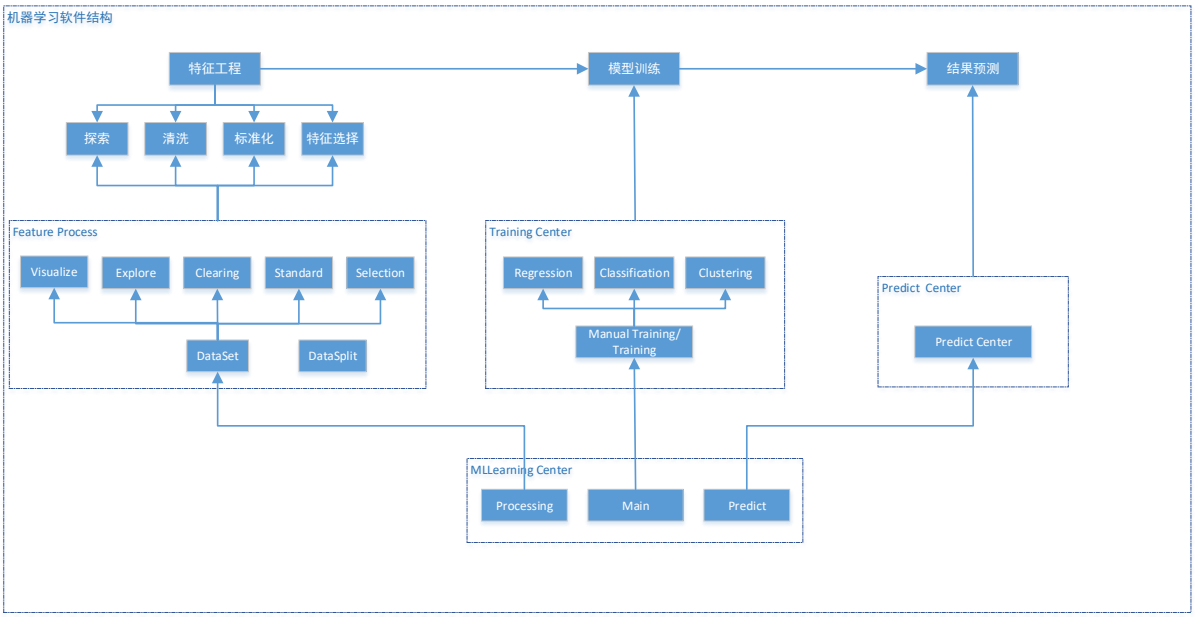
1) 搭建可扩展 AI 框架第二版，第一次修改
2) 使用搭建框架完成回归作业，观察学习曲线对比

第 1 部分

框架设计

根据机器学习的开发流程，将框架设计为三部分：特征工程、模型训练、结果预测；额外追加一个通用代码部分

1) 特征工程：此部分包括数据探索、数据清洗、数据标准化、特征选择等部分
2) 模型训练：模型训练包括手动训练和训练模块两部分，分别实现手写算法和 sklearn 等机器学习框架的算法
3) 结果预测：结果预测部分负责调用训练好的模型，来进行训练，本次实验不涉及此不部分，暂设定为 TODO 状态
4) 公用代码：包括通用的 onehot 设定、绘图等功能，本次扩展出简单图形绘制

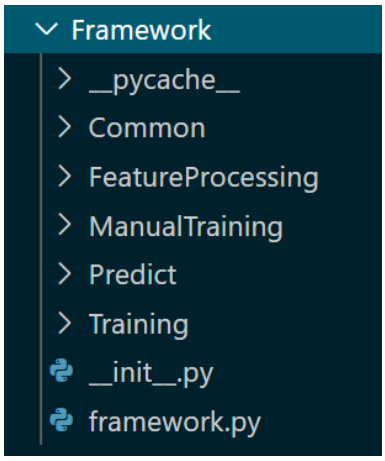


AI 框架设计图

框架的目的是集成机器学习的各类算法，并且方便每次的功能实现，单一对通用代码进行封装，并不能解决特征工程的复杂性，以及每次训练的循环操作，所以这里对代码结构进行调整，增加数据集操作模块 DataSet,以及工程端增加了 processing 模块，以处理复杂多变的源数据。

代码

Framework 代码描述

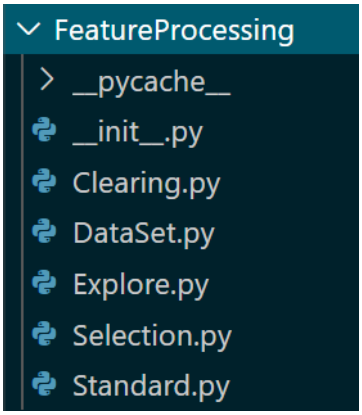


代码结构图

Framework 设计为 python 包的形式，在包代码不成熟阶段，暂时放在项目内直接通过相对路径引用。Framework 包括五个子包，分别为：

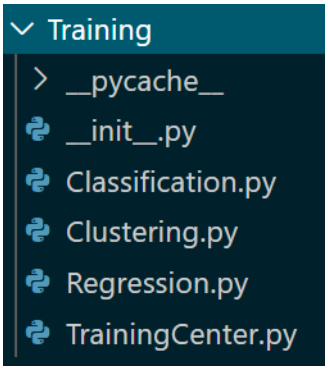
1） Common:通用代码包：主要包括绘图、基础数据处理等通用代码

2） FeatureProcesing:特征工程相关代码，主要是对 sklearn 代码调用的封装



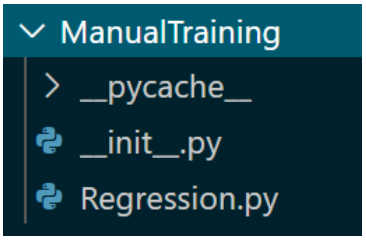
FeatureProcesing 包结构图

3） Training:根据算法分类，目前设定分类、聚类、回归三个子包



Training 包结构图

4） ManualTraining:暂时封装本次作业的 BGD 和 SGD 两个算法



ManualTraining 包结构图

代码展示：

```
import random

class Regression():
    def training_sgd_fit(self, x, y, alpha,theta_array):
        diff = [0, 0]
        error1 = 0
        m = len(x)
        #init the parameters to zero
        theta0 = theta_array[0]
        theta1 = theta_array[1]
        theta2 = theta_array[2]
        epoch = 0
        error_array = []
        epoch_array = []
        #calculate the parameters
        # 线性回归: h(x) = theta0  + theta1 * x[i][0] + theta2 * x[i][1]
        # 损失函数: 累和 (1/2) *  (y - h(x)) ^ 2
        # theta0 = theta0 - (   -alpha * (y - h(x))* 1 )
        # theta1 = theta1 - (   -alpha * (y - h(x))* x[i][0] )
        # theta2 = theta2 - (   -alpha * (y - h(x))* x[i][1] )
        # 1. 随机梯度下降算法在迭代的时候，每迭代一个新的样本，就会更新一次所有的
theta 参数。
        i = random.randint(0, m - 1)

        # (y - h(x))
        diff[0] = y[i] - (theta0 * 1 + theta1 * x[i][0] + theta2 * x[i][1])
        # - (y - h(x))x
        gradient0 = -diff[0] * 1
        gradient1 = -diff[0] * x[i][0]
        gradient2 = -diff[0] * x[i][1]
        # theta = theta - (  - alpha * (y - h(x))x )
        theta0 = theta0 - alpha * gradient0
        theta1 = theta1 - alpha * gradient1
        theta2 = theta2 - alpha * gradient2
        #theta3
```

```
        #calculate the cost function
        error1 = 0
        # 此处error为一个相对的Error值。
        for i in range(m):
            error1 += (y[i] - (theta0 * 1 + theta1 * x[i][0] + theta2 * x[i][1
]))**2

        error1 = error1 / m
        return error1,(theta0,theta1,theta2)

    def training_bgd_fit(self, x, y, alpha,theta_array):
        diff = [0,0]
        error1 = 0
        error0 =0
        m = len(x)

        #init the parameters to zero
        theta0 = theta_array[0]
        theta1 = theta_array[1]
        theta2 = theta_array[2]
        sum0 = 0
        sum1 = 0
        sum2 = 0

        epoch = 0
        error_array = []
        epoch_array = []
        #calculate the parameters
        # 线性回归: hi(x) = theta0 + theta1 * x[i][1] + theta2 * x[i][2]
        # 损失函数: (1/2) 累加 * (y - h(x)) ^ 2
        # theta = theta - 累和(  - alpha * (y - h(x))x )
        # 1. 随机梯度下降算法在迭代的时候，每迭代一个新的样本，就会更新一次所有的
theta参数。
        #calculate the parameters
        # 2. 批梯度下降算法在迭代的时候，是完成所有样本的迭代后才会去更新一次theta
参数
        for i in range(m):
            #begin batch gradient descent
            diff[0] = y[i]-( theta0 + theta1 * x[i][0] + theta2 * x[i][1] )
            sum0 = sum0 - ( -alpha * diff[0]* 1)
            sum1 = sum1 - ( -alpha * diff[0]* x[i][0])
            sum2 = sum2 - ( -alpha * diff[0]* x[i][1])
            #end  batch gradient descent
```

```
        theta0 = theta0 + sum0 / m;
        theta1 = theta1 + sum1 / m;
        theta2 = theta2 + sum2 / m;

        sum0 = 0
        sum1 = 0
        sum2 = 0
        #calculate the cost function
        error1 = 0
        for i in range(m):
            error1 += ( y[i]-
( theta0 + theta1 * x[i][0] + theta2 * x[i][1] ) )**2
        error1 = error1 / m
        return error1,(theta0,theta1,theta2)
```

5） Predict:本次暂不开发

DataSet 数据集代码

说明:

1） __init__:数据集初始化，需要传入训练批次以及单次训练大小
2） DataInit:通过反射，调用实现端的 Processing 模块，完成初始数据源的获取，并在 Processing 模块中实现每个批次数据的特殊处理扩展
3） VisualizeSourceData:对数据源数据进行简单的绘图观察，暂时支持散点图、线形图以及对应的两个 3D 图
4） Fetch_next_batch:获取批次训练数据
5） Get_step:获取训练批次

```
import os
import sys
from ..Common.BasicVisualize import digram_show

class DataSet():
    def __init__(self, batch_size, epoch):
        print("DataSet init")
        self.batch_size = batch_size
        self.epoch = epoch
        self.DataInit()
    # 初始化函数
    def DataInit(self):
        path = os.path.join(sys.path[0])
        print("test path:" + path)
        # 反射，调用processer 模块，获取数据方法 get_data
```

```
        moduleName = 'Processing'  # 要引入的模块
        className = "Processor"  # 要使用的方法
        model = __import__(moduleName, globals=path)  # 导入模块
        self.processClass = getattr(model, className)  # 找到模块中的属性

        self.X, self.y = self.processClass().get_data()
        print('DataSet init data')
    # 训练初始数据可视化（简要）
    def VisualizeSourceData(self, show_type):
        X_data, y_data = self.X, self.y
        if show_type == 'plot':
            digram_show.show_plot(X_data, y_data)
        elif show_type == 'scatter':
            digram_show.show_scatter(X_data, y_data)
        elif show_type == 'plot_3d':
            print(type(X_data))
            digram_show.show_plot_3d([x[0] for x in X_data], y_data,
                                     [x[1] for x in X_data])
        elif show_type == 'scatter_3d':
            digram_show.show_scatter_3d([x[0] for x in X_data], y_data,
                                        [x[1] for x in X_data])
    # 获取一批训练数据 TODO：根据训练批次获取数据
    def fetch_next_batch(self):
        x_data = self.processClass().input_x(self.X)
        y_data = self.processClass().input_y(self.y)
        if self.batch_size > len(self.X):
            return x_data, y_data
        else:
            return x_data[0:self.batch_size-1], y_data[0:self.batch_size-1]
    # 获取当前数据执行批次
    def get_step(self):
        return self.epoch
```

调用端代码描述

调用端代码主要包括三个部分：main、processing、predict(本次暂不实现)

1) Main：代码调用主逻辑，负责初始化数据集，调用 framework 中的各模块获取算法等，代码展示如下

```
from Framework.FeatureProcessing.DataSet import DataSet
from Framework.ManualTraining.Regression import Regression
from Framework.Common.BasicVisualize import digram_show
```

```
batch_size = 100
epoch = 500
dataSet = DataSet(batch_size, epoch)
# 查看数据空间分布
# dataSet.VisualizeSourceData('scatter_3d');

# 初始化线性模型库
regression = Regression()

error_sgd_array = []
epoch_sgd_array = []
theta_sgd_array=[0,0,0]
error_bgd_array = []
epoch_bgd_array = []
theta_bgd_array=[0,0,0]
alpha=0.001
for step in range(dataSet.get_step()):
    # 获取训练数据
    X_train, y_train = dataSet.fetch_next_batch()

    # SGD 训练
    error, theta_sgd_array = regression.training_sgd_fit(X_train, y_train, alp
ha,theta_sgd_array)
    error_sgd_array.append(error)
    epoch_sgd_array.append(step)
    print(' theta0 : %f, theta1 : %f, theta2 : %f, sgd error1 : %f, epoch : %
f'
        % (theta_sgd_array[0], theta_sgd_array[1], theta_sgd_array[2], error,
step))

    # BGD 训练
    error_bgd, theta_bgd_array = regression.training_bgd_fit(X_train, y_train,
 alpha,theta_bgd_array)
    error_bgd_array.append(error_bgd)
    epoch_bgd_array.append(step)
    print(' theta0 : %f, theta1 : %f, theta2 : %f, sgd error1 : %f, epoch : %
f'
        % (theta_bgd_array[0], theta_bgd_array[1], theta_bgd_array[2], error_b
gd, step))

digram_show.show_plot_diff(epoch_sgd_array, error_sgd_array, epoch_bgd_array,
error_bgd_array)
```

2） Processing:负责初始化数据传递给 dataset 数据集，对每一批次数据的特殊化处理，代码展示如下

```
class Processor():

    # 初始数据获取
    def get_data(self):
        # data init
        X = [(0., 3), (1., 3), (2., 3), (3., 2), (4., 4), (0., 3), (1., 3.1),
             (2., 3.5), (3., 2.1), (4., 4.2)]
        y = [
            95.364, 97.217205, 75.195834, 60.105519, 49.342380, 100.364,
            100.217205, 100.195834, 100.105519, 12.342380
        ]
        return X,y

    # 特征数据加工
    def input_x(self, data_X):
        # 特征数据加工处理
        return data_X

    # 标签数据加工
    def input_y(self, label):
        # 标签数据加工处理
        return label
```

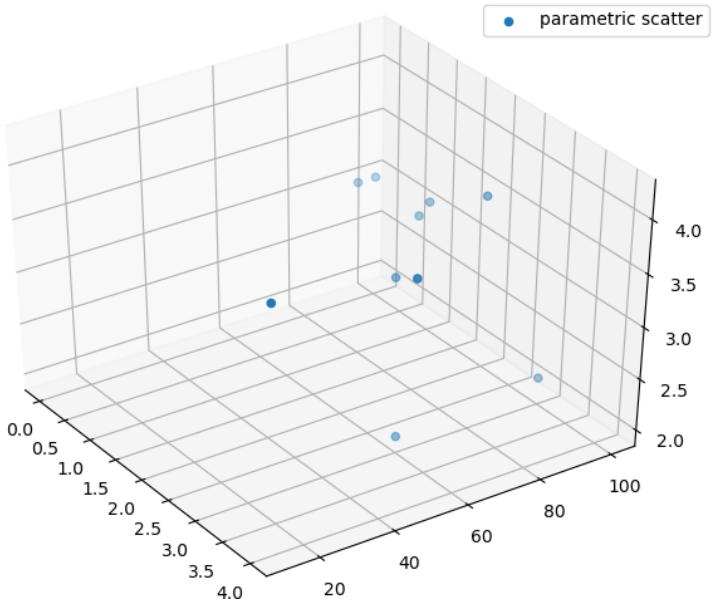
第 2 部分

数据源可视化

执行代码在 main 函数中，

```
dataSet = DataSet(batch_size, epoch)
# 查看数据空间分布
dataSet.VisualizeSourceData('scatter_3d');
```

可视化结果：

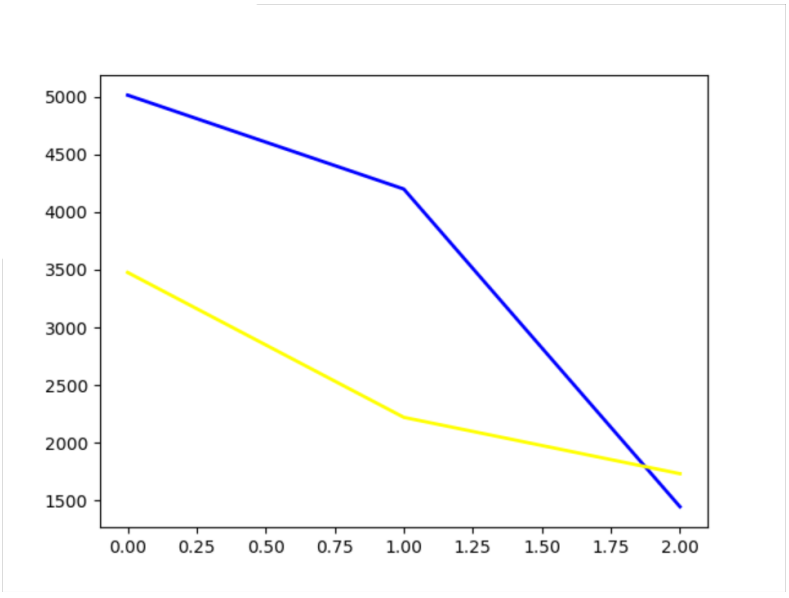


数据源可视化

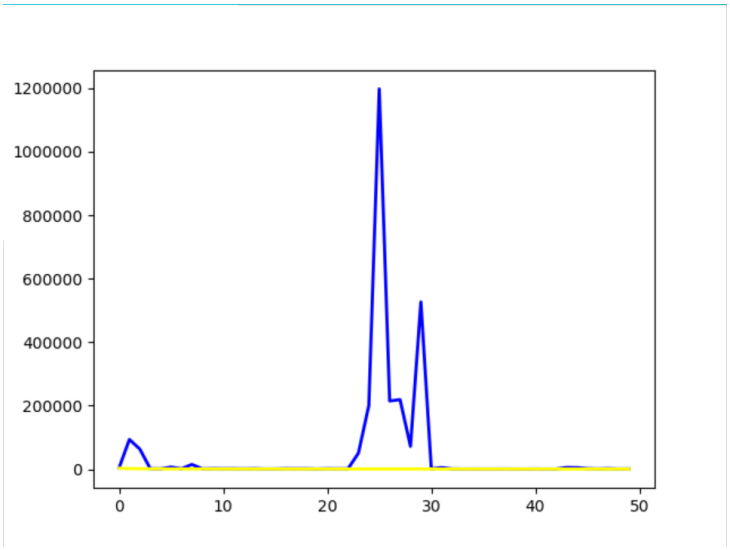
训练记录

说明：由于数据源一直，所以获取数据部分合并，统一通过 dataset 获取下一批次训练数据。每次训练结果获取后，统一绘制在一张图上，BGD 和 SGD 分别用黄色和蓝色来标注。分辨观察在相同训练数据和相同的迭代步长设置下，对 Loss 值的影响曲线

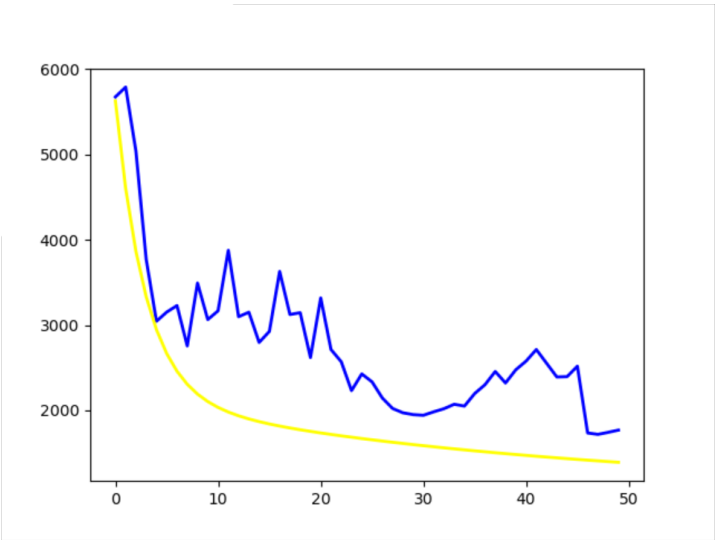
1) 实验 1：（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=3,alpha=0.1



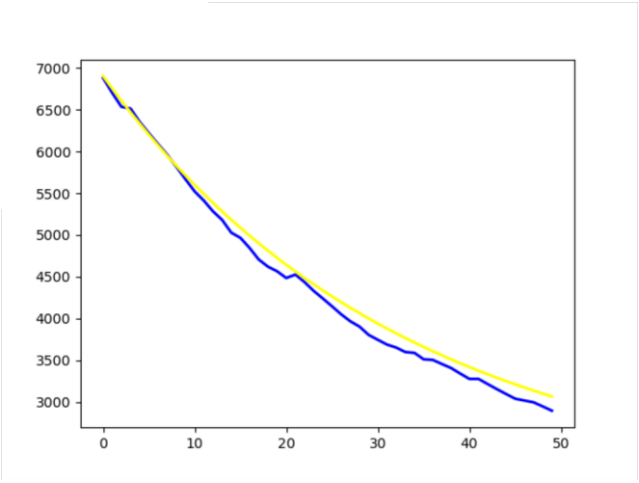
2) 实验 2：（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=50,alpha=0.1



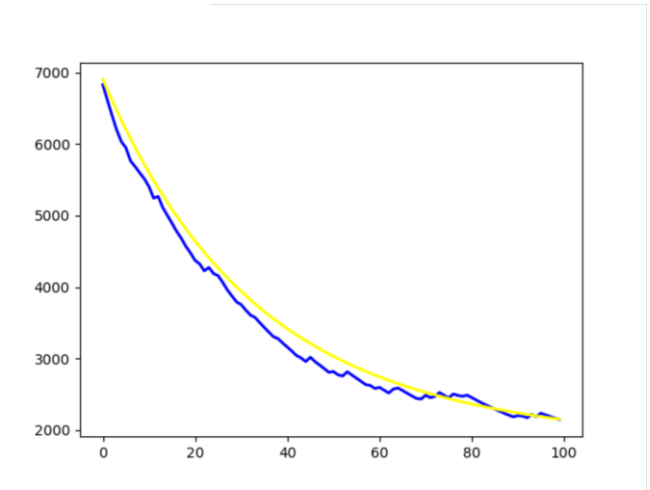
3) 实验 3:（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=50,alpha=0.01



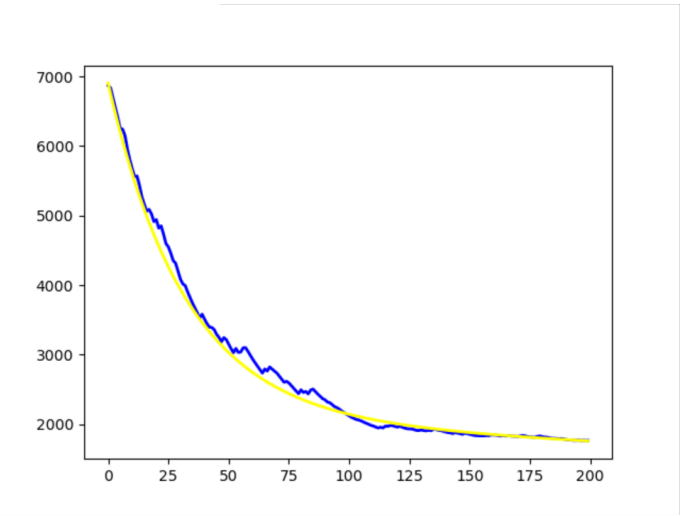
4) 实验 4:（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=50,alpha=0.001



5) 实验 5:（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=100,alpha=0.001



6) 实验 6:（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=200,alpha=0.001



7) 实验 7:（BGD 和 SGD 分别用黄色和蓝色来标注）epoch=2000,alpha=0.001