**中国矿业大学计算机学院**

**2019 级本科生课程作业**

|  |  |
| --- | --- |
| 课程名称 | 机器学习及优化——实验二 |
| 报告时间 | 2023年4月24日 |
| 学生姓名 | 许万鹏 |
| 学 号 | 05191643 |
| 班 级 | 信息安全2019-01班 |
| 任课教师 | 李政伟 |

摘要

随着机器学习在医疗领域应用的不断加深，越来越多的通过机器学习改进医疗服务的例子逐渐涌现。乳腺癌是一种常见的恶性肿瘤疾病，为了能够更加精确地诊断是否患有乳腺癌的风险，我们可以使用机器学习技术。在本次实验中，本人首先使用机器学习库Scikit-Learn实现了一种简洁且有效的预测方式。为了更深入地理解各机器学习算法的原理，将理论知识工程化，本人基于目前最常用的深度学习框架PyTorch，从零实现了本次任务中用到的所有机器学习算法，如SGD、Adam等，并将它们组合成了一个完整的框架——XuTorch，最后调用该框架完成了乳腺癌疾病诊断任务。

目 录

[摘要 II](#_Toc133699128)

[1 概述 1](#_Toc133699129)

[1.1 数据集介绍 1](#_Toc133699130)

[1.2 数据探索 1](#_Toc133699131)

[2 基于SkLearn的简洁实现 3](#_Toc133699132)

[3 基于PyTorch的从零实现 4](#_Toc133699133)

[3.1 调用XuTorch的实现 4](#_Toc133699134)

[3.2 基于Pytorch实现XuTorch 9](#_Toc133699135)

[3.2.1 xutorch.dataset.prep\_dataloader 10](#_Toc133699136)

[3.2.2 xutorch.dataset.BreastCancer 11](#_Toc133699137)

[3.2.3 xutorch.misc.set\_seeds 13](#_Toc133699138)

[3.2.4 xutorch.misc.get\_device 13](#_Toc133699139)

[3.2.5 xutorch.model.EmulatedSVM 14](#_Toc133699140)

[3.2.6 xutorch.optim.SimpleSGD 15](#_Toc133699141)

[3.2.7 xutorch.loss.Hinge 16](#_Toc133699142)

[3.2.8 xutorch.plot.\* 17](#_Toc133699143)

# 概述

## 数据集介绍

该数据集由Dr. William H. Wolberg，W. Nick Street和Olvi L. Mangasarian创作，于美国加州大学欧文分校（UCI）机器学习仓库开放获取。其共有569条样本，每条样本由1列ID、1列诊断结果和30列实值特征组成，30列特征是通过计算10种原始特征的平均值（Mean）、标准差（Stand Error, SE）和最坏值（Worst）得到的，10种原始特征如下表。

|  |  |
| --- | --- |
| 变量 | 解释 |
| Radius | 半径（中心到周边点的距离平均值） |
| Texture | 纹理（灰度值的标准差） |
| Perimeter | 周长 |
| Area | 面积 |
| Smoothness | 光滑度（半径长度的局部变化） |
| Compactness | 紧凑度（） |
| Concavity | 凹度（轮廓的凹陷部分的严重程度） |
| Concave Points | 凹点（轮廓的凹陷部分的数量） |
| Symmetry | 对称性 |
| Fractal\_Dimension | 分形维数（"海岸线近似值" - 1） |

## 数据探索

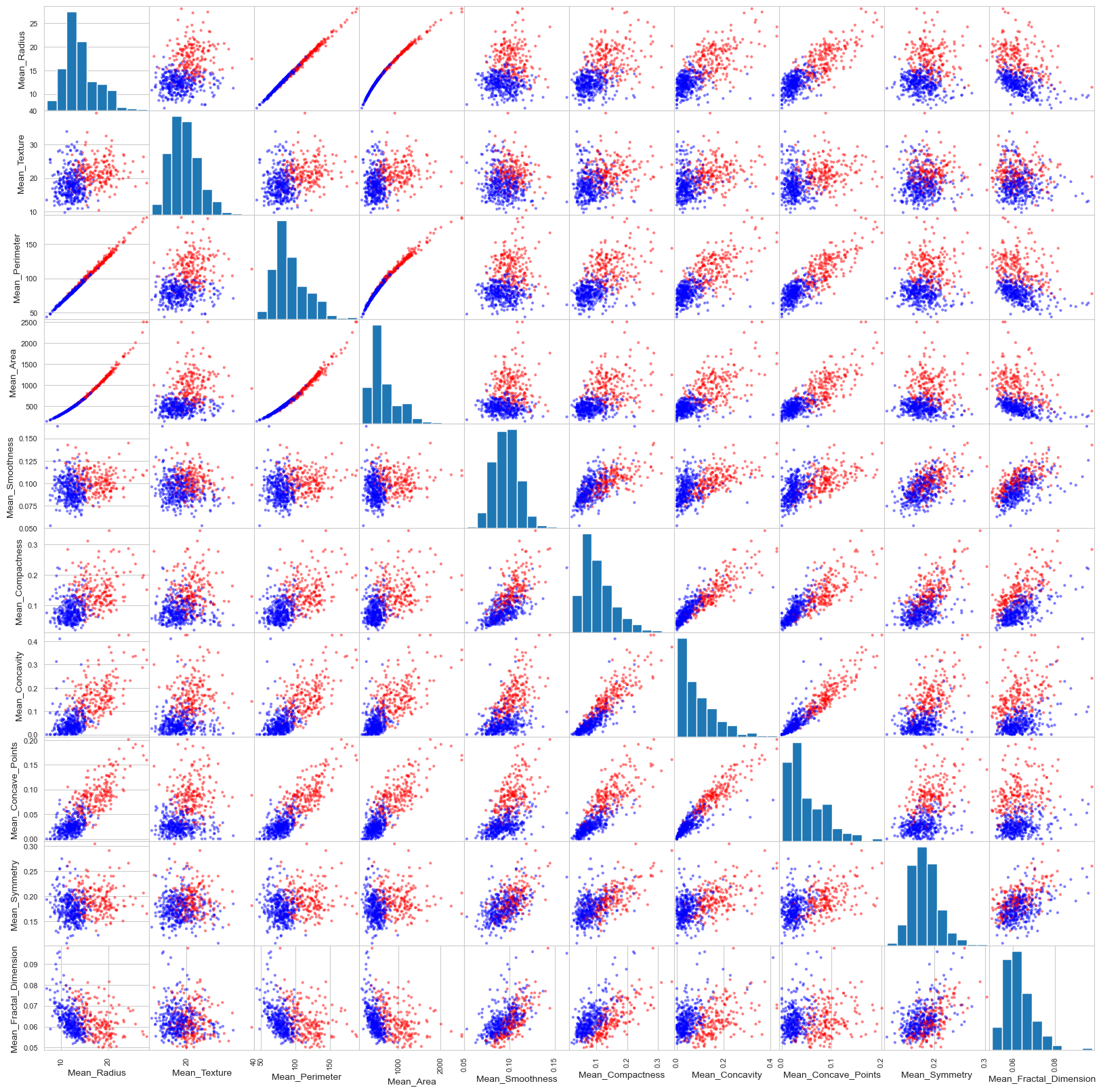
随着深度学习的流行，特征工程的重要性已逐渐减弱，因为大型的深度神经网络甚至可以学习到特征的重要性。因为波士顿房价数据集本身的特征少，只有13种，所以我的倾向是全部使用，让模型自己学习重要程度。这里可以进行一些简单的数据探索。

略过数据清洗（df.isnull().sum()等类似工作），我们来分析一下数据相关性。

由于30个特征太多，我们选取10个原始特征的平均值（Mean）来进行分析。当然，其他统计特征的分析也同理。

import pandas as pd  
  
column\_names = ["ID", "Diagnosis",  
 "Mean\_Radius", "Mean\_Texture", "Mean\_Perimeter", "Mean\_Area", "Mean\_Smoothness", "Mean\_Compactness", "Mean\_Concavity", "Mean\_Concave\_Points", "Mean\_Symmetry", "Mean\_Fractal\_Dimension",  
 "SE\_Radius", "SE\_Texture", "SE\_Perimeter", "SE\_Area", "SE\_Smoothness", "SE\_Compactness", "SE\_Concavity", "SE\_Concave\_Points", "SE\_Symmetry", "SE\_Fractal\_Dimension",  
 "Worst\_Radius", "Worst\_Texture", "Worst\_Perimeter", "Worst\_Area", "Worst\_Smoothness", "Worst\_Compactness", "Worst\_Concavity", "Worst\_Concave\_Points", "Worst\_Symmetry", "Worst\_Fractal\_Dimension"  
]  
df = pd.read\_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data", names=column\_names).drop("ID", axis=1)  
df\_mean = df.iloc[:, [0] + list(range(1, 10 + 1))]  
colors = df["Diagnosis"].map(lambda x: {'M': "red", 'B': "blue"}.get(x))  
pd.plotting.scatter\_matrix(df\_mean, c=colors, alpha=0.5, figsize=(20, 20));

输出如下。



可以看出，大部分特征都是这样：随着其值的增加，红色样本点（恶性乳腺癌）的数量增加。因此，我们仍然选择所有特征，仅作归一化处理，让SVM自己判断其重要性。

# 基于SkLearn的简洁实现

首先，导入sklearn中已实现的函数。

from sklearn.svm import SVC  
from sklearn import datasets  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split

然后，按照机器学习的一般流程：准备数据、数据分析、特征工程、模型选择及训练、模型评估，逐个调用函数。

*# 加载UCI乳腺癌数据集*dataset = datasets.load\_breast\_cancer()  
  
*# 随机划分训练集和测试集*X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset.data, dataset.target, test\_size=0.3, random\_state=42)  
  
*# 特征工程：标准化数据，让数据符合正态分布，加速模型训练*transfer = StandardScaler()  
X\_train = transfer.fit\_transform(X\_train)  
X\_test = transfer.fit\_transform(X\_test)  
  
*# 创建用于分类的SVM模型——SVC*classifier = SVC(kernel='linear')  
*# classifier = SVC(kernel='rbf')  
# classifier = SVC(kernel='poly')  
  
# 训练*classifier.fit(X\_train, y\_train)  
  
*# 计算精确率Acc，评估SVM模型预测性能*score = classifier.score(X\_test, y\_test)  
print("精确率Acc:", score)

得到输出如下。

精确率Acc: 0.9766081871345029

可以看出，基于sklearn的代码可以很简洁地（不超过20行）得到较好的结果（精确率Acc 0.977）。

# 基于PyTorch的从零实现

## 调用XuTorch的实现

导入PyTroch和XuTroch库

import torch  
import xutorch

设置参数

args = {  
 'kernel\_func': 'linear',  
 'c': 0.01,  
 'lr': 0.01,  
 'num\_epochs': 1000,  
 'batch\_size': 12,  
 'early\_stop': 100,  
 'save\_path': 'weights/hw2\_model.pth'  
}

创建数据加载器

train\_loader, train\_dataset = xutorch.dataset.prep\_dataloader(  
 dataset\_name='BreastCancer',  
 download=True,  
 train=True,  
 batch\_size=args['batch\_size'],  
 test\_ratio=0.25,  
 transform=True  
)  
val\_loader, val\_dataset = xutorch.dataset.prep\_dataloader(  
 dataset\_name='BreastCancer',  
 download=True,  
 train=False,  
 batch\_size=args['batch\_size'],  
 test\_ratio=0.25,  
 transform=True  
)

输出如下。

Finished reading the train set of BreastCancer Dataset (444 samples found, each dim = 30)

Finished reading the val set of BreastCancer Dataset (125 samples found, each dim = 30)

创建模型、优化器、损失函数

xutorch.misc.set\_seeds(42)  
  
device = xutorch.misc.get\_device()  
  
model = xutorch.model.EmulatedSVM(kernel\_func=args['kernel\_func'], input\_dim=train\_dataset.dim, output\_dim=1).to(device)  
  
*# 创建一个Hinge损失函数*criterion = xutorch.loss.Hinge(args['c'])  
  
optimizer = xutorch.optim.SimpleSGD(model.parameters(), lr=args['lr'])

训练模型

loss\_record = {  
 'train': [],   
 'val': []  
}  
  
best\_val\_acc = 0.0  
best\_val\_acc\_epoch = 0.0  
  
early\_stop\_cnt = 0  
  
for epoch in range(args['num\_epochs']):  
 train\_acc = 0.0  
 train\_loss = 0.0  
  
 model.train()  
 for batch\_idx, (inputs, labels) in enumerate(train\_loader):  
 inputs, labels = inputs.to(device), labels.to(device)  
   
 outputs = model(inputs)  
 train\_pred = torch.sign(outputs)  
  
 weight = model.fc.weight.squeeze()  
 loss = criterion(outputs, labels, weight)  
  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 train\_acc += (train\_pred.cpu() == labels.cpu()).sum().item()  
 train\_loss += loss.item()  
 loss\_record['train'].append(train\_loss / len(train\_dataset))  
  
 val\_acc = 0.0  
 val\_loss = 0.0  
  
 model.eval()  
 for batch\_idx, (inputs, labels) in enumerate(val\_loader):  
 inputs, labels = inputs.to(device), labels.to(device)  
 with torch.no\_grad():  
 outputs = model(inputs)  
 val\_pred = torch.sign(outputs)  
  
 weight = model.fc.weight.squeeze()  
 loss = criterion(outputs, labels, weight)  
   
 val\_acc += (val\_pred.cpu() == labels.cpu()).sum().item()  
 val\_loss += loss.item()  
 loss\_record['val'].append(val\_loss / len(val\_dataset))  
  
 if epoch == 0 or val\_acc > best\_val\_acc:  
 best\_val\_acc = val\_acc  
 best\_val\_acc\_epoch = epoch  
 print(f'Saving model (epoch = {epoch + 1 : 4d}, acc = {best\_val\_acc / len(val\_dataset) : .4f})')  
 torch.save(model.state\_dict(), args['save\_path'])  
 early\_stop\_cnt = 0  
 else:  
 early\_stop\_cnt += 1  
   
 if early\_stop\_cnt > args['early\_stop']:  
 print('EARLY STOP')  
 break  
  
 if epoch % 10 == 9:  
 print('[{:03d}/{:03d}] Train Acc: {:3.6f} Loss: {:3.6f} | Val Acc: {:3.6f} loss: {:3.6f}'.format(  
 epoch + 1, args['num\_epochs'],   
 train\_acc / len(train\_dataset), train\_loss / len(train\_dataset),   
 val\_acc / len(val\_dataset), val\_loss / len(val\_dataset)))

输出如下。

Saving model (epoch = 1, acc = 0.9040)

Saving model (epoch = 2, acc = 0.9360)

Saving model (epoch = 3, acc = 0.9440)

Saving model (epoch = 4, acc = 0.9600)

Saving model (epoch = 6, acc = 0.9680)

[010/1000] Train Acc: 0.977477 Loss: 0.007894 | Val Acc: 0.968000 loss: 0.010868

[020/1000] Train Acc: 0.981982 Loss: 0.006601 | Val Acc: 0.968000 loss: 0.009654

[030/1000] Train Acc: 0.981982 Loss: 0.006119 | Val Acc: 0.968000 loss: 0.009449

[040/1000] Train Acc: 0.984234 Loss: 0.005829 | Val Acc: 0.968000 loss: 0.009279

[050/1000] Train Acc: 0.981982 Loss: 0.005613 | Val Acc: 0.968000 loss: 0.009149

[060/1000] Train Acc: 0.981982 Loss: 0.005473 | Val Acc: 0.968000 loss: 0.008954

Saving model (epoch = 64, acc = 0.9760)

[070/1000] Train Acc: 0.981982 Loss: 0.005379 | Val Acc: 0.968000 loss: 0.008810

[080/1000] Train Acc: 0.981982 Loss: 0.005323 | Val Acc: 0.968000 loss: 0.008780

[090/1000] Train Acc: 0.979730 Loss: 0.005286 | Val Acc: 0.976000 loss: 0.008758

[100/1000] Train Acc: 0.981982 Loss: 0.005231 | Val Acc: 0.976000 loss: 0.008752

[110/1000] Train Acc: 0.981982 Loss: 0.005219 | Val Acc: 0.976000 loss: 0.008662

[120/1000] Train Acc: 0.981982 Loss: 0.005172 | Val Acc: 0.976000 loss: 0.008685

[130/1000] Train Acc: 0.981982 Loss: 0.005162 | Val Acc: 0.976000 loss: 0.008661

[140/1000] Train Acc: 0.981982 Loss: 0.005139 | Val Acc: 0.976000 loss: 0.008613

[150/1000] Train Acc: 0.981982 Loss: 0.005135 | Val Acc: 0.976000 loss: 0.008664

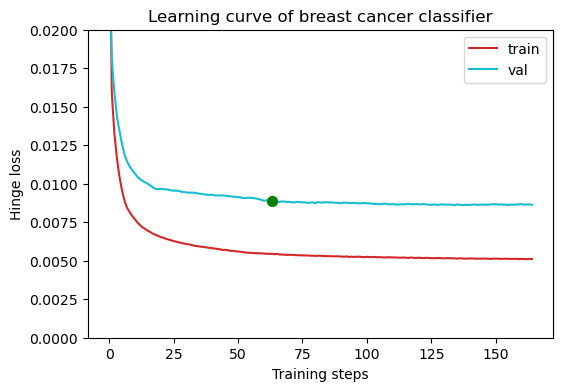
[160/1000] Train Acc: 0.984234 Loss: 0.005107 | Val Acc: 0.976000 loss: 0.008670

\*\*EARLY STOP\*\*

通过最后一轮的Acc可以看出，其略低于同随机种子下的基于sklearn的实现。因为我实现的SVM并不是基于SMO算法求闭式解（性能要求，不易实现），而是基于SGD求最优化解，因此会略低0.01左右。

绘制学习曲线

xutorch.plot.plot\_learning\_curve(loss\_record, title='breast cancer classifier', loss\_name='Hinge', bottom=0.0, top=0.02, min\_loss\_x=best\_val\_acc\_epoch, min\_loss\_y=loss\_record['val'][best\_val\_acc\_epoch])



分析权重系数，可以发现Worst\_Texture特征的系数较大，Mean\_Fractal\_Dimension特征的系数较小。这说明在乳腺癌诊断种，纹理越差，患乳腺癌概率越大；分形维数越高，患乳腺癌概率越小。

当然，在不同的核方法下，最具影响的特征会发生变化，但它们都是诊断乳腺癌的重要因素。

params = model.state\_dict()  
  
for name, param in params.items():  
 print(name, param)  
  
*# 对乳腺癌影响最显著的两个feature*feature\_names = ["Mean\_Radius", "Mean\_Texture", "Mean\_Perimeter", "Mean\_Area", "Mean\_Smoothness", "Mean\_Compactness", "Mean\_Concavity", "Mean\_Concave\_Points", "Mean\_Symmetry", "Mean\_Fractal\_Dimension", "SE\_Radius", "SE\_Texture", "SE\_Perimeter", "SE\_Area", "SE\_Smoothness", "SE\_Compactness", "SE\_Concavity", "SE\_Concave\_Points", "SE\_Symmetry", "SE\_Fractal\_Dimension", "Worst\_Radius", "Worst\_Texture", "Worst\_Perimeter", "Worst\_Area", "Worst\_Smoothness", "Worst\_Compactness", "Worst\_Concavity", "Worst\_Concave\_Points", "Worst\_Symmetry", "Worst\_Fractal\_Dimension"]  
feature\_names[torch.max(model.fc.weight, 1)[1].item()], feature\_names[torch.min(model.fc.weight, 1)[1].item()]

输出如下。

fc.weight tensor([[ 0.2650, 0.2481, 0.1537, 0.3058, 0.1708, -0.1461, 0.2963, 0.3841, -0.0040, -0.3619, 0.5232, -0.2604, 0.3518, 0.3636, 0.1659, -0.3340, 0.0677, 0.0669, -0.1391, -0.1785, 0.3508, 0.5606, 0.3077, 0.4505, 0.4458, -0.0076, 0.3460, 0.3579, 0.4218, 0.1251]], device='cuda:0')

fc.bias tensor([-0.1809], device='cuda:0')

('Worst\_Texture', 'Mean\_Fractal\_Dimension')

最后，尝试各种核方法，可以得到结果：

linear: 0.9760

poly: 0.9440

rbf: 0.8320

sigmoid: 0.9680

gaussian: 0.8320

可以看出，在乳腺癌诊断问题中，采用SVM配合sigmoid核效果较好，略优于线性核。

## 基于Pytorch实现XuTorch

这里介绍**本次任务所用到的**XuTorch中的机器学习算法的实现过程。

XuTorch

│ \_\_init\_\_.py

│

├─dataset

│ │ boston.py

│ │ **breast\_cancer.py**

│ │ **prep\_dataloder.py**

│ │ sentiment.py

│ │ \_\_init\_\_.py

│ │

│ └─data

│ boston.txt

│ huangguo\_mountain.csv

│ **wdbc.data**

│

├─loss

│ cross\_entropy.py

│ **hinge.py**

│ mean\_squared\_error.py

│ \_\_init\_\_.py

│

├─misc

│ **get\_device.py**

│ **r2\_score.py**

│ **set\_seeds.py**

│ \_\_init\_\_.py

│

├─model

│ embedding\_mlp.py

│ linear.py

│ mlp.py

│ **svm.py**

│ \_\_init\_\_.py

│

├─optim

│ adam.py

│ **sgd.py**

│ \_\_init\_\_.py

│

└─plot

**plot.py**

\_\_init\_\_.py

### xutorch.dataset.prep\_dataloader

这里定义了一个数据加载函数prep\_dataloader，其目的是根据数据集名称生成相应数据集对象并将其放入数据加载器中。

from xutorch.dataset.boston import BOSTON  
from xutorch.dataset.breast\_cancer import BreastCancer  
from xutorch.dataset.sentiment import Sentiment  
from torch.utils.data import DataLoader  
  
  
def prep\_dataloader(dataset\_name, download, train, batch\_size, num\_workers=0, test\_ratio=0.25, \*\*kwargs):  
 dataset = None  
 if dataset\_name == 'BOSTON':  
 dataset = BOSTON(download=download, train=train, test\_ratio=test\_ratio, \*\*kwargs)  
 elif dataset\_name == 'BreastCancer':  
 dataset = BreastCancer(download=download, train=train, test\_ratio=test\_ratio, \*\*kwargs)  
 elif dataset\_name == 'Sentiment':  
 dataset = Sentiment(download=download, train=train, test\_ratio=test\_ratio, \*\*kwargs)  
  
 dataloader = DataLoader(dataset, batch\_size, shuffle=train, drop\_last=False, num\_workers=num\_workers, pin\_memory=True)  
 return dataloader, dataset

### xutorch.dataset.BreastCancer

该代码实现了一个BreastCancer数据集的PyTorch数据集类，提供了威斯康星州乳腺癌数据集的读取和预处理方法。其中包含以下属性和方法：

\_\_init\_\_的参数：

download: bool型，默认值为True，指示是否下载数据集，如果设为False，则使用本地路径进行读取。

path: str型，数据集文件路径，默认为./data/wdbc.data，如果download设为True，则此路径将被忽略。

train: bool型，默认值为True，指示是否加载训练集数据。如果为False，则加载测试集数据。

transform: bool型，默认值为True，指示是否对数据进行标准化处理。

test\_ratio: float型，默认值为0.25，表示测试集占比。

类的属性：

data: torch.FloatTensor类型，数据集的特征向量，已经过处理，为一个张量。

target: torch.FloatTensor类型，数据集的标签向量，已经过处理，为一个张量。

dim: int型，数据集的特征向量的维度。

类的方法：

len(self): 返回数据集的样本数。

getitem(self, index): 根据索引返回相应的数据和标签。

在\_\_init\_\_方法中，首先会读取数据集并将其分割为特征数据和目标数据两个部分。其中，特征数据包含30个维度的特征，而目标数据仅包含一个维度的标签（且要将字符型转换为-1和1以符合SVM要求）。然后根据train参数将数据集分为训练集和测试集，并且将数据集进行了归一化处理。最后输出数据集的大小信息。

import os  
import torch  
from torch.utils.data import Dataset  
import pandas as pd  
  
  
class BreastCancer(Dataset):  
 data\_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data"  
 current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
 def \_\_init\_\_(self, download=True, path=os.path.join(current\_dir, 'data', 'wdbc.data'), train=True, transform=True, test\_ratio=0.25):  
 if download:  
 path = self.data\_url  
  
 column\_names = ["ID", "Diagnosis", "Mean\_Radius", "Mean\_Texture", "Mean\_Perimeter", "Mean\_Area",  
 "Mean\_Smoothness", "Mean\_Compactness", "Mean\_Concavity", "Mean\_Concave\_Points", "Mean\_Symmetry",  
 "Mean\_Fractal\_Dimension", "SE\_Radius", "SE\_Texture", "SE\_Perimeter", "SE\_Area", "SE\_Smoothness",  
 "SE\_Compactness", "SE\_Concavity", "SE\_Concave\_Points", "SE\_Symmetry", "SE\_Fractal\_Dimension",  
 "Worst\_Radius", "Worst\_Texture", "Worst\_Perimeter", "Worst\_Area", "Worst\_Smoothness",  
 "Worst\_Compactness", "Worst\_Concavity", "Worst\_Concave\_Points", "Worst\_Symmetry",  
 "Worst\_Fractal\_Dimension"]  
 df = pd.read\_csv(path, names=column\_names)  
 df = df.drop("ID", axis=1)  
  
 data = df.drop("Diagnosis", axis=1)  
 targets = df["Diagnosis"].replace({'M': 1, 'B': -1}) *# -1和1符合SVM表述* pivot = int(100 \* (1 - test\_ratio))  
 if train:  
 indices = [i for i in range(len(data)) if i % 100 < pivot]  
 else:  
 indices = [i for i in range(len(data)) if i % 100 >= pivot]  
  
 self.data = torch.FloatTensor(data.iloc[indices].values)  
 self.target = torch.FloatTensor(targets.iloc[indices].values)  
  
 if transform:  
 *# 对train或dev集合分别进行归一化* self.data = (self.data - self.data.mean(dim=0, keepdim=True)) / self.data.std(dim=0, keepdim=True)  
  
 self.dim = self.data.shape[1]  
  
 print('Finished reading the {} set of BreastCancer Dataset ({} samples found, each dim = {})'.format(  
 'train' if train else 'val', len(self.data), self.dim))  
  
 def \_\_len\_\_(self):  
 return len(self.data)  
  
 def \_\_getitem\_\_(self, index):  
 return self.data[index], self.target[index]

### xutorch.misc.set\_seeds

固定随机种子以复现相同结果。

import os  
import torch  
import random  
import numpy as np  
  
  
def set\_seeds(seed):  
 torch.manual\_seed(seed)  
 os.environ['PYTHONHASHSEED'] = str(seed)  
 random.seed(seed)  
 np.random.seed(seed)  
 if torch.cuda.is\_available():  
 torch.cuda.manual\_seed(seed)  
 torch.cuda.manual\_seed\_all(seed)  
 *# if torch.backends.cudnn.is\_available():  
 # torch.backends.cudnn.benchmark = False  
 # torch.backends.cudnn.deterministic = True*

### xutorch.misc.get\_device

检查计算设备：是否有可用的GPU，如果有则使用GPU，否则使用CPU。

import torch  
  
  
def get\_device():  
 return torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')

### xutorch.model.EmulatedSVM

这里定义了一个SVM类，说是SVM类但由于没有办法实现SMO方法，所以姑且可以当作一个内置了核方法的线性模型来实现，毕竟SVM目前最常见的用法就是配合核方法，重要的是核方法部分。因此这个类叫做仿制的（Emulated）SVM。

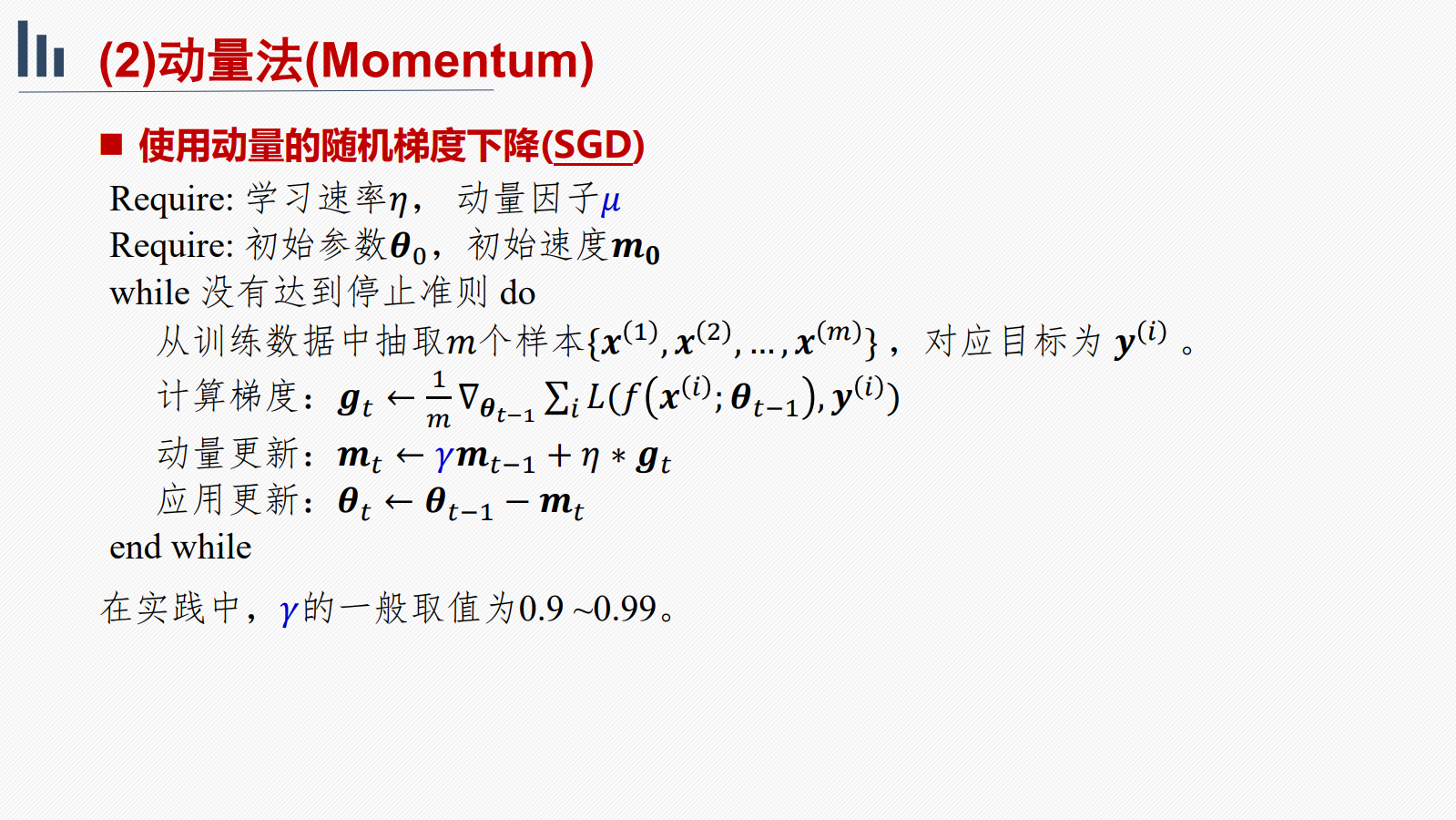
import torch  
import torch.nn as nn  
  
  
class EmulatedSVM(nn.Module):  
 def \_\_init\_\_(self, kernel\_func, input\_dim=30, output\_dim=1):  
 super(EmulatedSVM, self).\_\_init\_\_()  
 self.kernel\_func = None  
 self.kernel\_funcs = {  
 'linear': self.linear\_kernel,  
 'poly': self.polynomial\_kernel,  
 'rbf': self.rbf\_kernel,  
 'sigmoid': self.sigmoid\_kernel,  
 'gaussian': self.gaussian\_kernel  
 }  
 if isinstance(kernel\_func, str):  
 *# 根据简称获取核函数* if kernel\_func in self.kernel\_funcs:  
 self.kernel\_func = self.kernel\_funcs[kernel\_func]  
 *# 根据全称获取核函数* elif hasattr(self, kernel\_func):  
 self.kernel\_func = getattr(self, kernel\_func)  
 else:  
 raise ValueError("Invalid kernel function: {}".format(kernel\_func))  
 elif callable(kernel\_func):  
 self.kernel\_func = kernel\_func  
  
 self.fc = nn.Linear(input\_dim, output\_dim)  
  
 def forward(self, x):  
 x = self.kernel\_func(x)  
 x = self.fc(x)  
 return x.squeeze(1)  
  
 *# 核函数方法* def linear\_kernel(self, x):  
 return x  
  
 def polynomial\_kernel(self, x, degree=2):  
 return torch.pow(x + 1, degree)  
  
 def gaussian\_kernel(self, x, sigma=1.0):  
 return torch.exp(-torch.pow(x, 2) / (2 \* sigma \*\* 2))  
  
 def rbf\_kernel(self, x, gamma=1.0):  
 return torch.exp(-gamma \* torch.pow(x, 2))  
  
 def sigmoid\_kernel(self, x):  
 return torch.sigmoid(x)

### xutorch.optim.SimpleSGD

这里实现了简单的随机梯度下降（SGD）算法。为了省略一些不重要的操作（如zero\_grad和另存参数），我令该类继承自PyTorch的Optimizer类。

构造函数\_\_init\_\_中，接收三个参数：params表示要更新的参数组成的列表，lr是学习率（默认为0.01），momentum是动量（默认为0）。

类中的step方法是该优化器的核心方法，用于更新参数，**参考自课程PPT中的Chap3 P62 动量法**。



在这个方法中，首先遍历参数组group，然后遍历每个参数p，如果p没有梯度，则跳过。如果p有梯度，则获取该参数的状态state，以及学习率lr和动量momentum。如果状态中没有动量缓存，则创建一个全零的tensor作为缓存。然后将该参数的梯度与动量缓存进行累积，根据学习率和动量的大小对梯度进行加权和，更新参数p的数值。注意，如果不在step内部清空梯度（p.data=None），则一定要在代码外部显式地书写optimizer.zero\_grad()

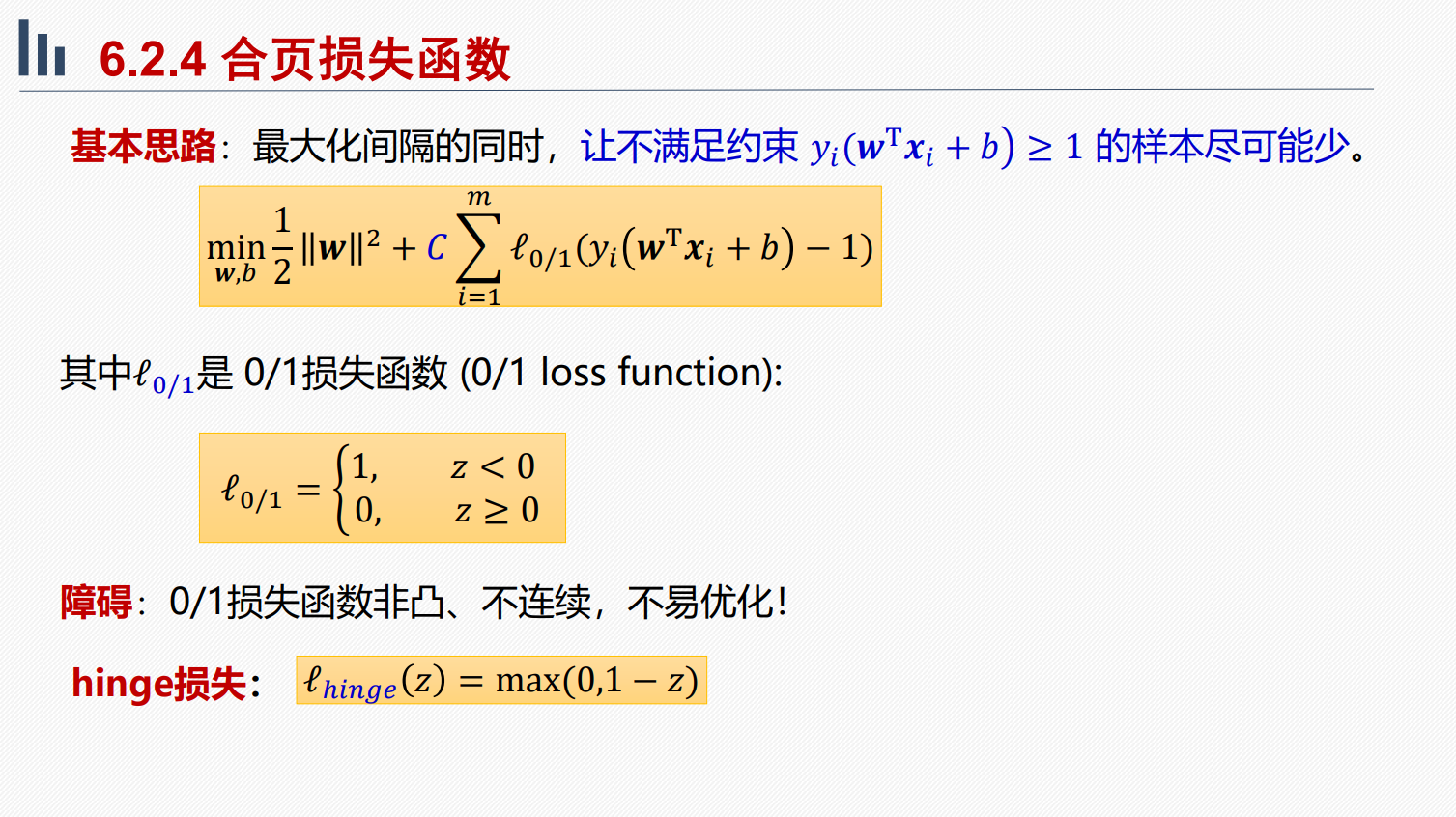
需要注意的是，对于原地方法buf.mul\_和p.data.add\_中的alpha参数，它们的含义分别是乘数和加数的系数，用于对乘积或加和进行缩放。

import torch  
from torch.optim import Optimizer  
  
  
class SimpleSGD(Optimizer):  
 def \_\_init\_\_(self, params, lr=0.01, momentum=0):  
 defaults = dict(lr=lr, momentum=momentum)  
 super(SimpleSGD, self).\_\_init\_\_(params, defaults)  
  
 def step(self):  
 for group in self.param\_groups:  
 for p in group['params']:  
 if p.grad is None:  
 continue  
 grad = p.grad.data  
 state = self.state[p]  
  
 lr = group['lr']  
 momentum = group['momentum']  
 if 'momentum\_buffer' not in state:  
 buf = state['momentum\_buffer'] = torch.zeros\_like(p.data)  
 else:  
 buf = state['momentum\_buffer']  
 *# buf = momentum \* buf + grad* buf.mul\_(momentum).add\_(grad)  
  
 *# p.data += -self.lr \* buf* p.data.add\_(buf, alpha=-lr) *# new*

### xutorch.loss.Hinge

该类用于计算合页（Hinge）损失函数。该损失函数通常用于SVM优化，其目的是最大化间隔的同时，让不满足约束的样本尽可能少。

**参考自课程PPT中的Chap6 P32 合页损失函数。**



forward方法实现了接受两个tensor作为输入参数outputs和targets，该损失函数由两个部分组成，第一部分是对于每个样本的损失的平均值，第二部分是对于权重的正则项，两部分的损失相加就是最终的损失。

import torch  
import torch.nn as nn  
  
  
class Hinge(nn.Module):  
 def \_\_init\_\_(self, c=0.01):  
 super(Hinge, self).\_\_init\_\_()  
 self.c = c  
  
 def forward(self, outputs, labels, weight):  
 *# loss = torch.mean(torch.clamp(1 - labels \* outputs, min=0)) + 0.01 \* (weight.t() @ weight) / 2.0* loss\_part1 = torch.mean(torch.clamp(1 - labels \* outputs, min=0))  
 loss\_part2 = self.c \* torch.norm(weight, p=2) \*\* 2 / 2.0  
 loss = loss\_part1 + loss\_part2  
  
 return loss

### xutorch.plot.\*

这里包含了三个用于绘图的函数，其中：

plot\_gt\_and\_pred 用于绘制预测值与真实值的散点图（预测-真实值散点图）；

plot\_gt\_vs\_pred 用于绘制真实值与预测值的曲线图（拟合曲线）；

plot\_learning\_curve 用于绘制训练曲线。

import torch  
import matplotlib.pyplot as plt  
from matplotlib.pyplot import figure  
  
  
def plot\_gt\_and\_pred(targets=None, preds=None, bottom=0.0, top=35.0):  
 import os  
 os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"  
  
 preds = torch.cat(preds, dim=0).numpy()  
 targets = torch.cat(targets, dim=0).numpy()  
  
 figure(figsize=(5, 5))  
 plt.scatter(targets, preds, c='r', alpha=0.5)  
 plt.plot([bottom, top], [bottom, top], c='b')  
 plt.xlim(bottom, top)  
 plt.ylim(bottom, top)  
 plt.xlabel('ground truth value')  
 plt.ylabel('predicted value')  
 plt.title('Ground Truth v.s. Prediction')  
 plt.show()  
  
  
def plot\_gt\_vs\_pred(y\_true, y\_pred):  
 y\_pred = torch.cat(y\_pred, dim=0).numpy()  
 y\_true = torch.cat(y\_true, dim=0).numpy()  
 plt.plot(y\_true, label="gt")  
 plt.plot(y\_pred, label="predict")  
 plt.xlabel("index")  
 plt.ylabel("price")  
 plt.legend(loc="best")  
 plt.title("gt v.s. predict")  
 plt.show()  
  
  
def plot\_learning\_curve(loss\_record, loss\_name='', title='', bottom=0.0, top=100.0, min\_loss\_x=None, min\_loss\_y=None):  
 x\_1 = range(len(loss\_record['train']))  
 x\_2 = x\_1[::len(loss\_record['train']) // len(loss\_record['val'])]  
 figure(figsize=(6, 4))  
 plt.plot(x\_1, loss\_record['train'], c='tab:red', label='train')  
 plt.plot(x\_2, loss\_record['val'], c='tab:cyan', label='val')  
 if min\_loss\_x and min\_loss\_y:  
 plt.scatter(min\_loss\_x, min\_loss\_y, c='g', alpha=1, marker='o', s=50, zorder=10)  
 plt.ylim(bottom, top)  
 plt.xlabel('Training steps')  
 plt.ylabel('{} loss'.format(loss\_name))  
 plt.title('Learning curve of {}'.format(title))  
 plt.legend()  
 plt.show()