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

1 概述

1.1 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（ $perimeter^2 / area - 1.0$ ）
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）

1.2 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。

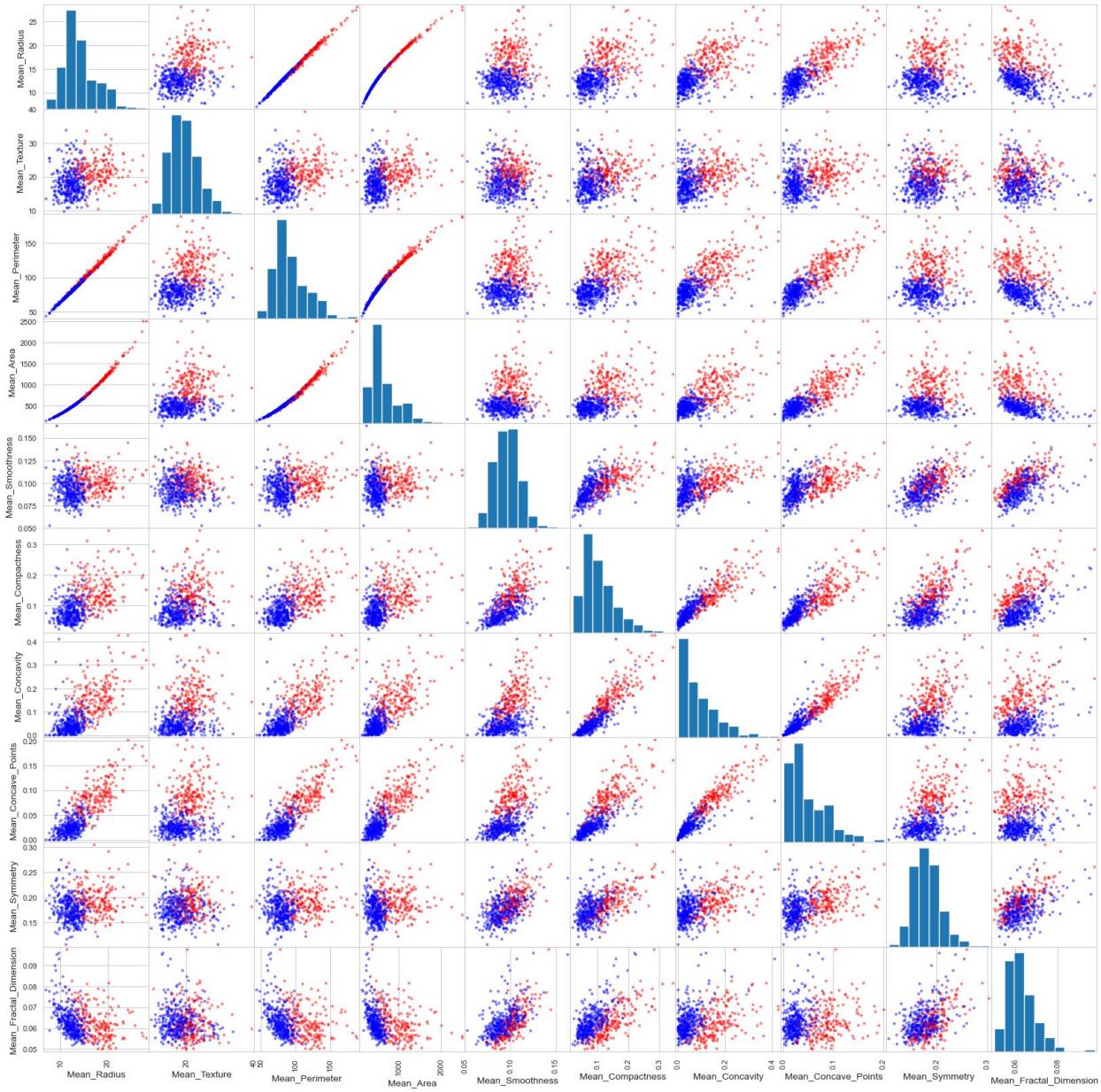
由于 30 个特征太多，我们选取 10 个原始特征的平均值（Mean）来进行分析。当然，其他统计特征的分析也同理。

```
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。

2 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）。

3 基于 PyTorch 的从零实现

3.1 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)
```

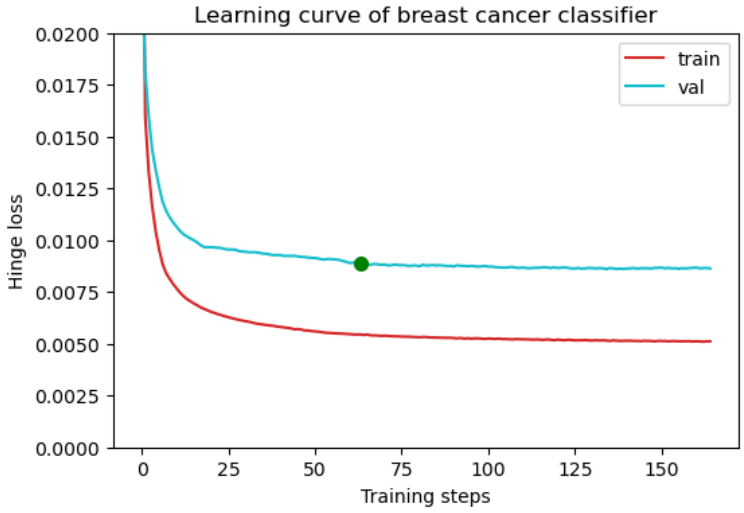
```
Saving model (epoch =    6, acc =  0.9680)
[010/1000] Train Acc: 0.977477 Loss: 0.007894 | Val Acc: 0.968000 loss:
0.010868
[020/1000] Train Acc: 0.981982 Loss: 0.006601 | Val Acc: 0.968000 loss:
0.009654
[030/1000] Train Acc: 0.981982 Loss: 0.006119 | Val Acc: 0.968000 loss:
0.009449
[040/1000] 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。

3.2 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
```

3.2.1 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
```

3.2.2 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]
```

3.2.3 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
```

3.2.4 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')
```

3.2.5 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)
```

3.2.6 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**。

(2)动量法(Momentum)

■ **使用动量的随机梯度下降(SGD)**

Require: 学习速率η， 动量因子μ

Require: 初始参数$\boldsymbol{\theta}_0$，初始速度$\boldsymbol{m_0}$

while 没有达到停止准则 do

从训练数据中抽取m个样本$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}_{t-1}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_{t-1}\right), \boldsymbol{y}^{(i)}\right)$

动量更新：$\boldsymbol{m}_t \leftarrow \gamma \boldsymbol{m}_{t-1} + \eta * \boldsymbol{g}_t$

应用更新：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t$

end while

在实践中，γ的一般取值为0.9 ~0.99。

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
```

3.2.7 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 $y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i+b\right)\geq 1$ 的样本尽可能少。

参考自课程 PPT 中的 Chap6 P32 合页损失函数。

6.2.4 合页损失函数

基本思路：最大化间隔的同时，让不满足约束 $y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1$ 的样本尽可能少。

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m} \ell_{0/1}(y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) - 1)$$

其中$\ell_{0/1}$是 0/1损失函数 (0/1 loss function):

$$\ell_{0/1} = \begin{cases} 1, & z < 0 \\ 0, & z \geq 0 \end{cases}$$

障碍：0/1损失函数非凸、不连续，不易优化!

hinge损失：$\ell_{hinge}(z) = \max(0, 1 - z)$

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
```

3.2.8 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()
```