

**模式识别与机器学习**

**实验报告**

**班级：10051901**

**姓名：牛远卓**

**学号：2019302973**

1. **实验目的**

通过跑SVM模型，了解它的构造、特点以及思想。

1. **实验原理**

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM还包括核技巧，这使它成为实质上的非线性分类器。SVM的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM的的学习算法就是求解凸二次规划的最优化算法。

1. **实验步骤和程序流程**

数据导入

实验先将（200，15，28，28）的训练数据和（200，5，28，28）的测试数据分别通过flatten导入到训练和测试的一维数组中。再将他们转换成numpy.array的形式，这样就可以直接放入svm中了。

SVM分类

我用的svm分类是用了Pytorch自带的包：sklearn.svm。然后调了些不同的参数，比如kernal,C,gamma等，来找出测试正确率最高的svm模型。

整体的项目流程如下图。

图示

描述已自动生成

后来发现只用svm跑出来效果太差，于是我根据Make SVM great againg这篇文章的code又跑了一下。由于它的code将数据集随机分成了训练、验证和测试集。其中，测试集是没有标签的。于是，我将我处理的老师给的数据分别给了项目的训练和验证集。测试就不测试了。简而言之，我将老师给的测试集作为项目的验证集“喂给”模型。详情请见后面的代码.程序流程如下图：

图示

描述已自动生成

其中，SiameseSVMNet模型具体构造可以看看原论文的介绍（在参考文献3中）。介绍的是相当详细清晰。

1. **实验结果**

**我主要测了kernal为linear，poly与rbf的情况，发现:**

**kernal=linear时，模型对其他参数都不敏感，毕竟是线性模型，能有啥超参数呢。正确率0.386。**

电脑萤幕

低可信度描述已自动生成

**kernal=poly时，模型degree敏感，毕竟是多项式模型。degree=3时，正确率最高，为0.402。**

文本

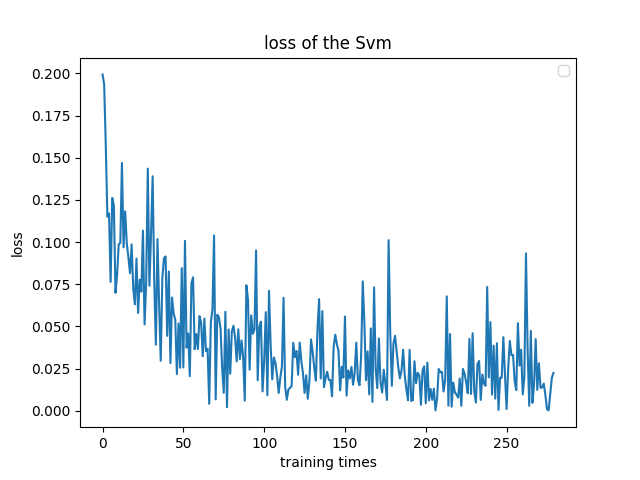
描述已自动生成

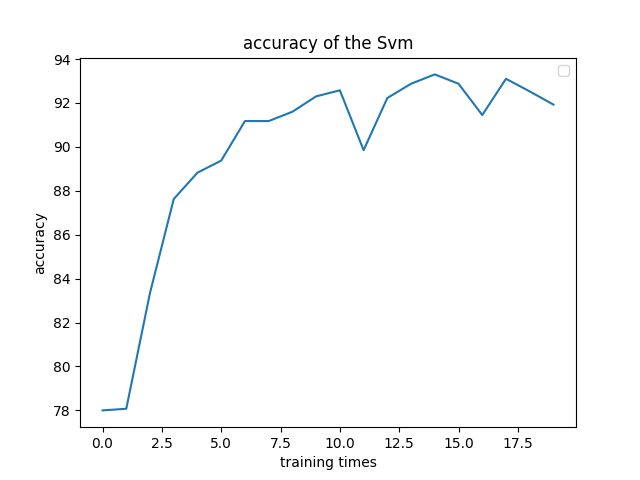
**kernal=rbf时，模型对C敏感，C=3.0或3.1时，正确率最高，为0.426。**

文本

描述已自动生成

**Make SVM great again:**

****

****

**可以看到，acc最高到了92.5%。比什么都不做就给svm分类效果要好得多。但是，效果没到原文中的99%。我想这很大一部分原因是我的数据集要比论文使用的小得多。**

1. **评价分析**

在选择kernal这一任务上，我参考了吴恩达老师的看法。

1.当 n 很大，m相对于n较小时。（如 n =10000，m = 10-1000）

选用模型：逻辑回归、线性SVM（其实就是普通不带核函数的SVM，视频中也叫线性核函数。）

个人理解：当特征维度过高，每个样本提供的信息已足够训练（如在已知房子的大小、装修情况、交通等等特征后，很容易退出房子单价。），这样的模型很容易过拟合。如果还采用带核函数的SVM，会使得模型更加容易过拟合，所以采用线性模型就够了。

解决方法：降维、增大数据量、正则等等

PS.听过一句话，高维线性模型等于低维非线性模型，结合核函数很好理解。

2.n小，m一般大时。（n = 1-1000 , m = 10-10000/20000）

选用模型：非线性SVM（如高斯核SVM）

个人理解：此时样本数相比于特征数可能差别不是很大，样本特征提供的信息可能不足，为了防止模型欠拟合，所以吴恩达建议使用非线性SVM。

3.n小，m很大时。（n = 1-1000 , m > 50000）

选用模型：非线性SVM 或 逻辑回归、线性SVM。

个人理解：先说吴恩达提到这种大批量训练集情况下使用非线性SVM也不是不可以，只不过核函数的存在会拖累运行速度。使用他建议 增加/创建 新的特征，然后再使用逻辑回归、线性SVM。这个我的理解是，由于特征少，训练得来的模型可能欠拟合，此时需要增加新的特征，获取更多特征信息。（欲向第一种情况靠拢）

这次学习了svm的主要架构以及各项参数的物理意义。

C : float, default=1.0

正则化参数. 正则化的强度与C的大小成反比. 但必须大于0. 用的是l2正 则化。

kernel :

核函数{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'

degree : int, default=3

多项式的度数

gamma : {'scale', 'auto'} or float, default='scale'

'rbf', 'poly' and 'sigmoid'中的核系数

如果gamma是scale,那么gamma=1/(n\_features \* X.var())

如果gamma是auto,那么gamma=1/n\_features

coef0 : float, default=0.0

在核函数中的独立变量

只在poly和sigmoid有用。

至于Make SVM great again主要使用度量学习在SVM分类前提取了图像的特征。具体来说，就是用一个特征提取卷积网络将同类的图像提取到特征空间的相近的地方；异类的到相远的地方。这显著的提高了SVM分类的准确率。

图示

描述已自动生成

**附1:参考文献**

1. [**https://zhuanlan.zhihu.com/p/31886934**](https://zhuanlan.zhihu.com/p/31886934)
2. [**https://blog.csdn.net/qq\_43703185/article/details/107845141**](https://blog.csdn.net/qq_43703185/article/details/107845141)
3. **https://openreview.net/pdf?id=B1EVwkqTW**

**附2：代码**

train.py:

import scipy.io as io

import matplotlib.pyplot as plt

from sklearn.preprocessing import scale

import torchvision

import torchvision.transforms as transforms

import torch

from sklearn import svm

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split as ts

import numpy as np

if \_\_name\_\_ == '\_\_main\_\_':

# import cv2

# 导入数据

    data=io.loadmat('./data2/Dataset.mat')

    # transform = transforms.Compose(

    #     [transforms.Resize([227,227]),

    #     transforms.ToTensor(),

    #     transforms.RandomCrop(224,padding = 2 ,pad\_if\_needed = True,fill = 0,padding\_mode ='constant'),

    #     transforms.RandomHorizontalFlip(p=0.5), # 表示进行左右的翻转

    #     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    # trainset = torchvision.datasets.ImageFolder(root='./dataset\_train',transform=transform)

    # trainloader = torch.utils.data.DataLoader(trainset, batch\_size=32,

                                            #   shuffle=True, num\_workers=2)

    # for data in enumerate(trainloader, 0):

    #     inputs, labels = data

        # print(inputs)

        # print("-----------------------")

        # print(labels)

    print(data.keys())

    print(data['train'].shape)

    print(data['test'].shape)

# plt.imshow(data['train'][0][0])

# 200类，每类15张训练集，5张测试集，图片size为28\*28

# 把28\*28 的矩阵resize为 784的一维向量作为SVM的输入

    data\_train=data['train']

    X\_train=[]

    for i in data\_train:

        for j in i:

        # print(j)

            j=j.flatten()

        # print(j)

            X\_train.append(j)

    y\_train=[]

    for i in range(0,200):

        for j in range(0,15):

            y\_train.append(i)

    data\_test=data['test']

    X\_test=[]

    for i in data\_test:

        for j in i:

        # print(j.shape)

            j=j.flatten()

        # print(j.shape)

            X\_test.append(j)

    y\_test=[]

    for i in range(0,200):

        for j in range(0,5):

            y\_test.append(i)

    X\_train=np.array(X\_train)

    X\_test=np.array(X\_test)

    y\_train=np.array(y\_train)

    y\_test=np.array(y\_test)

# trainset = transform(X\_train)

    print(X\_train.shape,X\_test.shape,y\_train.shape,y\_test.shape)

# kernel = 'rbf'

    # list1 = np.arange(0.0001,0.001,0.0001) #C\_best=3.1 0.426

    list2 = np.arange(2.5,3.5,0.1) #C\_best=3.1 0.426

    # for gamma1 in list1:

    for Ci in list2:

        clf\_rbf = svm.SVC(kernel='rbf', C=Ci)

        clf\_rbf.fit(X\_train,y\_train)

        score\_rbf = clf\_rbf.score(X\_test,y\_test)

        print("The score of C= %f rbf  is : %f"%(Ci,score\_rbf))

# # kernel = 'linear'

#     list2 = np.arange(2.5,3.5,0.1)

#     for Ci in list2:

#         clf\_linear = svm.SVC(kernel='linear',C=Ci)

#         clf\_linear.fit(X\_train,y\_train)

#         score\_linear = clf\_linear.score(X\_test,y\_test)

#         print("The score of linear is : %f"%score\_linear)

# kernel = 'poly'

    # list1 = np.arange(0,10,1) #C\_best=3.1 0.426

    # list2 = np.arange(2.5,3.5,0.1) #C\_best=3.1 0.426

    # for c1 in list1:#The score of degree= 3.000000 C=3.400000 poly is : 0.402000

    #     # for Ci in list2:

    #     clf\_poly = svm.SVC(kernel='poly',C=3,coef0=c1)

    #     clf\_poly.fit(X\_train,y\_train)

    #     score\_poly = clf\_poly.score(X\_test,y\_test)

    #     print("The score of coef0= %f C=3 poly is : %f"%(c1,score\_poly))

# # kernel = 'sigmoid'

#     clf\_rbf = svm.SVC(kernel='sigmoid')

#     clf\_rbf.fit(X\_train,y\_train)

#     score\_rbf = clf\_rbf.score(X\_test,y\_test)

#     print("The score of sigmoid is : %f"%score\_rbf)

# # kernel = 'precomputed'

# clf\_rbf = svm.SVC(kernel='precomputed')

# clf\_rbf.fit(X\_train,y\_train)

# score\_rbf = clf\_rbf.score(X\_test,y\_test)

# print("The score of precomputed is : %f"%score\_rbf)

dataload.py:

# import numpy as np

# import h5py

# mat = h5py.File('./data2/Dataset.mat')

# print(mat['images'].shape)#查看mat文件中images的格式

#   #(2284, 3, 640, 480)

# images = np.transpose(mat['images'])

#  #转置，images是numpy.ndarray格式的

# print(images)#控制台输出数据

# print(images.shape)#输出数据格式

#  #(480, 640, 3, 2284)

# np.save('./images', images)#保存数据，会生成一个images.npy文件

import matplotlib.pyplot as plt

import scipy.io as sio

import numpy as np

import  torchvision.transforms as transforms

from PIL import Image

import os

yFile = './data2/Dataset.mat'    #相对路径

datay=sio.loadmat(yFile)

plt.imshow(datay['train'][1][0])

# images\_train = np.transpose(datay['train'])

# images\_test = np.transpose(datay['test'])

# print (datay)

# print (datay['train'],datay['train'].shape)

# # (200, 15, 28, 28)

# print (datay['test'],datay['test'].shape)

# # (200, 5, 28, 28)

# print (datay['images'],datay['images'].shape)

# np.save('C:\\Users\\Jerry\\Desktop\\junior2\\PRML\\Project\\images\_train\\images\_train.npy', images\_train)

# np.save('C:\\Users\\Jerry\\Desktop\\junior2\\PRML\\Project\\images\_test\\images\_test.npy', images\_test)

# print(list(datay))

# print(datay['test'])

# print(datay['train'])

# dataset\_train = np.load('./images\_train/images\_train.npy')

# dataset\_test = np.load('./images\_test/images\_test.npy')

# print(dataset\_train.shape[3]) #200

# print(dataset\_test.shape[3]) #200

# print(dataset\_train.shape[2]) #15

# print(dataset\_test.shape[2]) #5

# print(dataset\_train.shape[1]) #28

# print(dataset\_test.shape[1]) #28

# print(dataset\_train.shape[0]) #28

# print(dataset\_test.shape[0]) #28

# print(dataset\_test)

for i in range(datay['train'].shape[0]): #200

    imgs\_tensor = datay['train'][i, :, :, :]

    # print(imgs\_tensor.shape)#(15, 28, 28)

    for j in range(imgs\_tensor.shape[0]): #15

        img\_tensor = imgs\_tensor[j,:,:]

        # print(img\_tensor.shape)#(28, 28)

        img = transforms.ToPILImage()(img\_tensor)#转成图片

        # print(img.size)#(28, 28)

        # print(img)

        # print(img.mode)

        # if img.mode != 'L':

        #     img = img.convert('L')

        # for i in range(dataset\_train.shape[3]): #200创建文件夹分类

        #     isExists = os.path.exists('./dataset\_train/%d' % i )

        #     if not isExists:                      #判断如果文件不存在,则创建

        #         os.makedirs('./dataset\_train/%d' % i)

        #         # print("%s 目录创建成功"%i)

        #     else:

        #         # print("%s 目录已经存在"%i)

        #         continue          #如果文件不存在,则继续上述操作,直到循环结束

        # plt.savefig('./dataset\_train/%d/%d.jpg' %(i,j))

        plt.imsave('./dataset\_train/%d/%d.png' %(i,j),img)

        if(os.path.isfile('./dataset\_train/%d/%d.jpg' %(i,j))):

            os.remove('./dataset\_train/%d/%d.jpg' %(i,j))

        #img.show()

#添加图片

# for i in range(dataset\_train.shape[3]): #200

#     for j in range(dataset\_train.shape[2]): #15

#         I = Image.open('./dataset\_train/%d/%d.jpg'%(i,j))

#         I.save('./dataset\_train/%d/%d.jpg' %(i,j))

# for i in range(dataset\_test.shape[3]): #200

#     imgs\_tensor = dataset\_test[i, :, :, :]

#     # print(imgs\_tensor.shape)#(28, 28, 15)

#     for j in range(imgs\_tensor.shape[2]): #15

#         img\_tensor = imgs\_tensor[j,:,:]

#         # print(img\_tensor.shape)#(28, 28)

#         img = transforms.ToPILImage()(img\_tensor)#转成图片

#         # print(img.size)#(28, 28)

#         # print(img)

#         for i in range(dataset\_test.shape[3]): #200创建文件夹分类

#             isExists = os.path.exists('./dataset\_test/%d' % i )

#             if not isExists:                      #判断如果文件不存在,则创建

#                 os.makedirs('./dataset\_test/%d' % i)

#                 # print("%s 目录创建成功"%i)

#             else:

#                 # print("%s 目录已经存在"%i)

#                 continue          #如果文件不存在,则继续上述操作,直到循环结束

#         plt.savefig('./dataset\_test/%d/%d.jpg' %(i,j))

for i in range(datay['test'].shape[0]): #200

    imgs\_tensor = datay['test'][i, :, :, :]

    # print(imgs\_tensor.shape)#(15, 28, 28)

    for j in range(imgs\_tensor.shape[0]): #15

        img\_tensor = imgs\_tensor[j,:,:]

        # print(img\_tensor.shape)#(28, 28)

        img = transforms.ToPILImage()(img\_tensor)#转成图片

        # print(img.size)#(28, 28)

        # print(img)

        # for i in range(dataset\_train.shape[3]): #200创建文件夹分类

        #     isExists = os.path.exists('./dataset\_train/%d' % i )

        #     if not isExists:                      #判断如果文件不存在,则创建

        #         os.makedirs('./dataset\_train/%d' % i)

        #         # print("%s 目录创建成功"%i)

        #     else:

        #         # print("%s 目录已经存在"%i)

        #         continue          #如果文件不存在,则继续上述操作,直到循环结束

        # plt.savefig('./dataset\_train/%d/%d.jpg' %(i,j))

        plt.imsave('./dataset\_test/%d/%d.png' %(i,j),img)

        if(os.path.isfile('./dataset\_test/%d/%d.jpg'  %(i,j))):

            os.remove('./dataset\_test/%d/%d.jpg' %(i,j))

        #img.show()

Make SVM great again:

训练集和测试集：

class OmniglotPair(object):

    def \_\_init\_\_(self, path, choosen\_classes=None, transform=None, train=True):

        random.seed(42)

        self.path = path

        self.data = []

        self.transform = transform

        self.train = train

        class\_num = 0

        folders = [f for f in os.listdir(path) if not f[0] == '.']

        for directory in folders:

            folders2 = [f for f in os.listdir(

                os.path.join(path, directory)) if not f[0] == '.']

            for subdirectory in folders2:

                # character is in the choosen classes

                if (choosen\_classes is None) or class\_num in choosen\_classes:

                    folders3 = [f for f in os.listdir(os.path.join(

                        path, directory, subdirectory)) if not f[0] == '.']

                    temp\_array = []

                    for file in folders3:

                        temp\_array.append(os.path.join(

                            path, directory, subdirectory, file))

                    self.data.append(temp\_array)

                class\_num += 1

        self.sub\_index = []

        if(self.train):

            for i in range(14):

                for j in range(i, 15):

                    self.sub\_index.append((i, j))

        else:

            for i in range(4):

                for j in range(i, 5):

                    self.sub\_index.append((i, j))

    def \_\_getitem\_\_(self, index):

        # if(index >= 14 \* 15 \* 200):

        #     return

        if(self.train):

            class\_num = index // 210

            img\_number = index % 210

            if (img\_number % 2 == 0):

                img1 = self.load\_img(

                    self.data[class\_num][self.sub\_index[img\_number // 2][0]])

                img2 = self.load\_img(

                    self.data[class\_num][self.sub\_index[img\_number // 2][1]])

                return img1, img2, torch.from\_numpy(np.array([1])).float()

            else:

                img1 = self.load\_img(

                self.data[class\_num][self.sub\_index[img\_number // 2][0]])

                inc = random.randrange(1, len(self.data))

                dn = (class\_num + inc) % len(self.data)

                di = random.randrange(0, len(self.data[dn]))

                img2 = self.load\_img(self.data[dn][di])

                return img1, img2, torch.from\_numpy(np.array([-1])).float()

        else:

            class\_num = index // 20

            img\_number = index % 20

            if (img\_number % 2 == 0):

                img1 = self.load\_img(

                    self.data[class\_num][self.sub\_index[img\_number // 2][0]])

                img2 = self.load\_img(

                    self.data[class\_num][self.sub\_index[img\_number // 2][1]])

                return img1, img2, torch.from\_numpy(np.array([1])).float()

            else:

                img1 = self.load\_img(

                self.data[class\_num][self.sub\_index[img\_number // 2][0]])

                inc = random.randrange(1, len(self.data))

                dn = (class\_num + inc) % len(self.data)

                di = random.randrange(0, len(self.data[dn]))

                img2 = self.load\_img(self.data[dn][di])

                return img1, img2, torch.from\_numpy(np.array([-1])).float()

    def load\_img(self, path):

        img = Image.open(path)

        img = img.convert('L')

        if self.transform is not None:

            img = self.transform(img)

        img\_array = np.array(img)

        img\_array = img\_array / 255.0

        img\_array = img\_array.reshape(

            (1, img\_array.shape[0], img\_array.shape[1]))

        return torch.from\_numpy(img\_array).float()

    def \_\_len\_\_(self):

        if(self.train):

            return 15 \* 14 \* len(self.data)

        else:

            return 5 \* 4 \* len(self.data)

def get\_random\_partition(seed=42):

    digit\_indices = [x for x in range(200)]

    random.seed(seed)

    random.shuffle(digit\_indices)

    train = digit\_indices[:]

    validate = digit\_indices[:]

    test = digit\_indices[:]

    return train, validate, test

def get\_loaders(args):

    train, validate, test = get\_random\_partition(args.seed)

    transform1 = transforms.Resize(105)

    # transform2 = transforms.Compose(

    #     [transforms.Resize(105),

    #      transforms.RandomCrop(105,padding = 2 ,pad\_if\_needed = True,fill = 0,padding\_mode ='constant'),

    #      transforms.RandomHorizontalFlip(p=0.5) # 表示进行左右的翻转

    #      ])

    dset\_train = OmniglotPair("data\_train\_val", train, transform=transform1, train=True)

    if args.cuda:

        train\_loader = DataLoader(dset\_train,

                                  batch\_size=args.batch\_size,

                                  shuffle=True,

                                  num\_workers=1,

                                  pin\_memory=True)

    else:

        train\_loader = DataLoader(dset\_train,

                                  batch\_size=args.batch\_size,

                                  shuffle=True,

                                  num\_workers=4,

                                  pin\_memory=False)

    dset\_validate = OmniglotPair("data\_test", validate, transform=transform1, train=False)

    if args.cuda:

        validate\_loader = DataLoader(dset\_validate,

                                     batch\_size=args.test\_batch\_size,

                                     shuffle=True,

                                     num\_workers=1,

                                     pin\_memory=True)

    else:

        validate\_loader = DataLoader(dset\_validate,

                                     batch\_size=args.test\_batch\_size,

                                     shuffle=True,

                                     num\_workers=4,

                                     pin\_memory=False)

    dset\_test = Omniglot("data\_test", test, transform=transform1)

    return train\_loader, validate\_loader, dset\_test

train.py

def main():

    args = get\_args()

    model = SiameseSVMNet()

    if args.cuda:

        model = model.cuda()

    criterion = SVMLoss(args.C)

    optimizer = torch.optim.Adam(model.parameters())

    train\_loader, validate\_loader, test\_data = get\_loaders(args)

    def training(epoch):

        print('Epoch', epoch + 1)

        model.train()

        for batch\_idx, (x0, x1, label) in enumerate(train\_loader):

            if args.cuda:

                x0, x1, label = x0.cuda(), x1.cuda(), label.cuda()

            x0, x1, label = Variable(x0), Variable(x1), Variable(label)

            optimizer.zero\_grad()

            output = model(x0, x1)

            loss = criterion(output, label)

            loss.backward()

            optimizer.step()

            if batch\_idx % 100 == 0:

                loss1.append(loss.item())

                print("\n Batch:  ", batch\_idx, " / ",

                      len(train\_loader), " --- Loss: ", loss.item())

    def validate():

        model.eval()

        acc = 0

        for batch\_idx, (x0, x1, label) in enumerate(validate\_loader):

            if args.cuda:

                x0, x1, label = x0.cuda(), x1.cuda(), label.cuda()

            x0, x1, label = Variable(x0), Variable(x1), Variable(label)

            output = model(x0, x1)

            acc += compute\_accuracy(output, label)

        acc = 100.0 \* acc / len(validate\_loader.dataset)

        print('\nValidation set: Accuracy: {}%\n'.format(acc))

        acc1.append(acc)

        return acc

svmloss.py:

class SVMLoss(nn.Module):

    def \_\_init\_\_(self, C):

        super(SVMLoss, self).\_\_init\_\_()

        self.C = C

    def forward(self, output, y):

        # loss calculation

        temp1 = (output \* y).view(-1)

        ones = temp1 / temp1

        temp2 = ones - temp1

        zeros = temp2 - temp2

        elementwiseloss = torch.max(temp2, zeros)

        elementwiseloss = elementwiseloss \* elementwiseloss

        return self.C \* torch.mean(elementwiseloss)

def compute\_accuracy(predictions, labels):

    '''Compute classification accuracy'''

    predictions = torch.sign(predictions)

    correct = predictions.eq(labels)

    result = 0

    for i  in range(correct.shape[0]):

        if correct[i]:

            result += 1

    return result