姓名：刘鑫杰 专业：计算机科学与技术

***The Fourth Week***

**前言：**

前几周对ZSL的整体发展和相关论文有了了解和阅读， ZSL最经典的算法DAP算法和上周阅读的SAE语义自编码器，对其算法和原理有了一定的理解。但还感觉很抽象。

所以本周在matlab上实现一些算法，同时也巩固上周学习的matlab的相关语法知识.

**大目录：**

**一、数据集说明**

**二、《Semantic Autoencoder for Zero-Shot Learning》代码复现及其理解**

**要求：**

1.使用给定的数据集

2.涉及到矩阵求导可以参考《The Matrix Cookbook》

3.环境:MATLAB

4.说明原理及其算法流程

5.函数说明

**一、数据集说明**

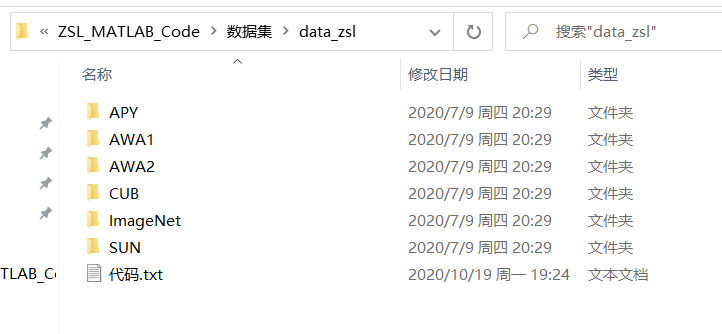
本数据集参考论文《Zero-Shot Learning—A Comprehensive Evaluation of the Good, the Bad and the Ugly》上的介绍（由于ZSL提出的新方法数量提升，文章提出统一评估协议和数据分割来定义一个新的基准。）

Zero-shot任务中训练和测试类是不相交的。而generalized zero-shot learning训练的类在测试的时候也出现了。从方法，数据集，评价协议三个方面对ZSL进行了基准和系统的评价。

1.方法：ZSL方法的关键是将可观察和不可观察的类通过一些辅助信息的形式连接起来，这些辅助信息能够将物体的视觉识别属性(visually distinguishing properties)进行编码。

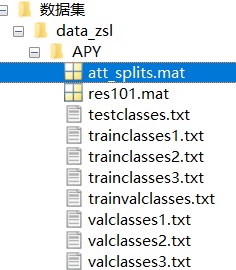
2.评估协议：强调了在与训练类分离的验证类上对方法超参数进行调优的必要性。同时也提出了不合理的地方：通过调优测试类的参数来提高ZSL性能违背了ZSL假设。通过深度神经网络DNN的在包含了Zero-shot测试类的大规模数据上的预训练来提取图片特征，这种图片特征提取是训练过程的一部分的设定也是违反了ZSL的定义的。

3.数据集：GZSL（generalized zero-shot learning）将一部分训练集中的样本(Seen)和测试集样本（Unseen）混合，用于测试阶段。🡪这个即所谓的标签嵌入（label embedding）

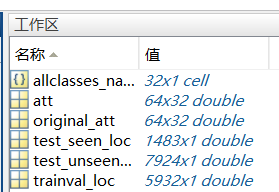


1.APY数据集说明(Attribute Pascal and Yahoo dataset)

共有32个类,其中20个类作为训练集(从其中随机选5个来进行验证),12个类作为测试集,类别语义为64维,共有15339张图片



(1)att\_splits.mat



allclasses\_names是32x1的元胞数组,里面的元素对应类别名称

att是64x32数组,每一列代表一个类,已经进行了归一化处理,范围(0,1)

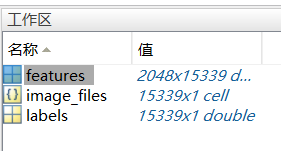
original\_att是att矩阵的原型

test\_seen\_loc 验证集可见类的位置

test\_unseen\_loc 测试集的位置

trainval\_loc训练集的位置

(2)res101.mat



features是2048x15339矩阵,每一列代表一张图片,共有2048维语义特征

image\_files是15339x1的元胞数组,每个元素是路径代表每张图片的路径

labels是15339x1的数组,每个元素是数字代表对应类别的数字标签

2.AWA1数据集说明(Animal with Attributes)

共50个类,其中40个类作为训练集（随机选取13个用于验证）,10个类作为测试集,类别语义为85维,共有30475张图片

3.AWA2数据集说明(Animal with Attributes2)

共50个类,其中40个类作为训练集（随机选取13个用于验证）,10个类作为测试集,类别语义为85维,共有37322张图片

4.CUB数据集说明(Caltech-UCSD-Birds-200-2011)

共200个类,其中150个类作为训练集（从中随机选取50个作为验证集）,50个类作为测试集,类别语义为312维,共有11788张图片

5.ImageNet数据集说明(ILSVRC2012/ILSVRC2010)

利用ImageNet做成的数据集，由ILSVRC2012的1000个类作为训练集（200个类验证），ILSVRC2010的360个类作为测试集，有254000张图片。它由4.6M的Wikipedia数据集训练而得到，共1000维。

6.SUN数据集说明(Sun database)

共717个类,其中645个类作为训练集（验证集随机选择65个类）,72个类作为测试集,类别语义为102维,每个类别20张图片,共有717\*20张图片

上述数据集中（1)–(4）(6)都是较小型(small-scale）的数据集，(5）是大型(large-

scale）数据集。虽然(1)–(4）(6)已经提供了人工定义的类别语义，但是有些作者也会从维基

语料库中自动提取出类别的语义表示，来检测自己的模型。

图像和类嵌入（class embeddings）：

我们为SUN、CUB、AWA和ImageNet从整个图像中提取图像特征，不进行图像预处理。对于aPY，正如[10]中提出的，我们从边界框（bounding box）中提取图像特征。我们的图像嵌入是101层ResNet[16]的2048-dim顶层池单元，因为我们发现它比GoogleNet[35]的1024 -dim顶层池单元（top layer pooling units）表现更好。ResNet是在ImageNet 1K上预训练的，没有经过微调。除了ResNet特性外，我们还根据已发布的图像特性对方法进行评估。作为类嵌入，对于aPY、AWA、CUB和SUN，我们使用每个类的属性。对于ImageNet，我们使用由[7]提供的Word2Vec[25]，因为它不包含所有类的属性注释。

数据集分割（dataset splits）：

在SS（Standard Split）和PS（Proposed Split）中，测试类中的图片不会出现在训练阶段，在测试阶段SS不包括任何一个来自训练集的图片，然而PS是包含的。我们这样设计PS是因为在测试阶段同时测试训练类和测试类对于展示方法的广义性(generlization)是非常重要的

**二、《Semantic Autoencoder for Zero-Shot Learning》代码复现及其理解**

**目录：**

**1.对原始数据集的处理**

**2.SAE代码(MATLAB)**

**3.算法流程及其相应函数的说明**

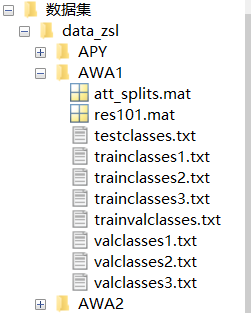
**4.错误调试过程**

**5.参数lambda调整过程**

**6.总结**

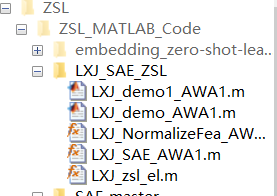
**1.对原始数据集的处理**

*Animal with Attributes(AWA1)*: 共50个类,其中40个类作为训练集,10个类作为测试集,类别语义为85维,共有30475张图片



将训练集和验证集合并为训练集,

**2.SAE代码(MATLAB)**



***LXJ\_demo\_AWA1.m***

***说明:demo,展示主程序***

%Demo on AWA for ZSL with SAE

%复现零样本学习语义自编码器

% <Semantic Autoencoder for Zero-shot Learning>

% 在训练集19832+4958=24790 测试集5685的情况下

% 将会得到下列结果

% [1] AwA ZSL accuracy [V >>> S]: 45.3%

% [2] AwA ZSL accuracy [S >>> V]: 50.0%

tic;

clc

clear all

close all

%%%%% 将数据集加载进来

load('数据集/data\_zsl/AWA1/att\_splits.mat');

load('数据集/data\_zsl/AWA1/res101.mat');

%将训练集的数据放在X\_tr\_temp中, 大小为2048 x (19832+4958)-->2048 x 24790

%将trainval\_loc和test\_seen\_loc合并为一个,作为训练矩阵trainval\_testseen\_loc

trainval\_testseen\_loc=[trainval\_loc;test\_seen\_loc];

X\_tr\_temp=features(:,trainval\_testseen\_loc');

%将X\_tr\_temp训练集数据使用NormalizeFea()函数归一化

% X\_tr的大小为24790 x 2048

X\_tr = LXJ\_NormalizeFea\_AWA1(X\_tr\_temp)';

% 将矩阵X\_te提取出来

%将测试集的数据放在X\_te\_temp中, 大小为2048 x 5685

X\_te\_temp=features(:,test\_unseen\_loc');

%将X\_te\_temp训练集数据使用NormalizeFea()函数归一化

% X\_te的大小为5685 x 2048

X\_te = LXJ\_NormalizeFea\_AWA1(X\_te\_temp)';

%%%%% 训练

% SAE

lambda = 18;

% 将语义矩阵S\_tr

% 下面要得到X\_tr\_temp对应的语义矩阵,首先基于一个前提,X\_tr\_temp是基于trainval\_testseen\_loc得到的,语义矩阵也要与此对应

% 我们先分析一个单过程是如何得到的:

% 首先:将trainval\_testseen\_loc中的给出的每个元素在features中的位置,通过这个位置信息在label中遍历找到该位置对应的标签信息

% 得到标签数字后,在通过查original\_att找到对应的语义信息,将这列添加到S\_tr中. 后面依次迭代进行,直到得出完整的训练集语义空间

% 训练集语义矩阵

S\_tr=[];

% trainval\_testseen\_loc中每个元素对应的数字标签(通过查表label)存储在trainval\_loc\_label[]中

train\_size=size(trainval\_testseen\_loc,1);

trainval\_loc\_label=zeros(1,train\_size);

% original\_att的转置矩阵为original\_att\_Transpose

original\_att\_Transpose=original\_att';

for i=1:train\_size

trainval\_loc\_label(1,i)=labels(trainval\_testseen\_loc(i,1),1);

S\_tr=[S\_tr original\_att\_Transpose(trainval\_loc\_label(1,i),:)'];

end

% 此时的S\_tr就是85 x 24790的semantic语义矩阵了

W=LXJ\_SAE\_AWA1(X\_tr\_temp,S\_tr,lambda);

% 下面考虑生成一个结构体param

param.HITK=1;

% testclasses\_id是测试类别对应的数字标签,可以利用allclasses\_names和testclasses.txt来求出测试类别的数字标签

filename='F:/MATLAB\_files/ZSL/ZSL\_MATLAB\_Code/数据集/data\_zsl/AWA1/testclasses.txt';

% 利用textread函数从testclasses.txt文件中提取对应的字符串类型的字符标签

temp\_strlabel=textread(filename,'%s');

% 将值赋给变量testclasses\_name\_str

testclasses\_name\_str=temp\_strlabel;

% 初始化testclasses\_id用于存储测试类的数字标签

param.testclasses\_id=zeros(10,1);

% 利用两个for循环遍历数组testclasses\_name\_str数组,对比字符串,将数字标签赋给testclasses\_id

for i=1:10

for j=1:50

if strcmp(testclasses\_name\_str(i,1),allclasses\_names(j,1))

param.testclasses\_id(i,1)=j;

break;

end

end

end

% 测试类别对应的数字标签,大小10 x 1

param.testclasses\_id=param.testclasses\_id;

% 测试集对应的数字标签,大小 5685 x 1

test\_unseen\_size=size(test\_unseen\_loc,1);

param.test\_labels=zeros(test\_unseen\_size,1);

% 利用for循环依次将labels test\_labels test\_unseen\_loc联系起来

for i=1:test\_unseen\_size

param.test\_labels(i,1)=labels(test\_unseen\_loc(i,1),1);

end

param.test\_labels=param.test\_labels;

%%%%% 测试

%[F --> S], projecting data from feature space to semantic sapce

S\_est = X\_te \* NormalizeFea(W)';

% 下面还差几个数据集待制作

param.train\_labels=trainval\_loc\_label';%train\_labels是训练集24790 x 1的数字标签矩阵

% 测试集语义矩阵S\_te\_gt和S\_te\_pro的制作,大小10 x 85

S\_te\_gt=[];

for i=1:size(param.testclasses\_id,1)

S\_te\_gt=[S\_te\_gt;original\_att\_Transpose(param.testclasses\_id(i,1),:)];

end

[zsl\_accuracy, Y\_hit5] = LXJ\_zsl\_el((S\_est), S\_te\_gt, param);

fprintf('[1] LXJ\_SAE\_ZSL\_AWA1 从特征空间到语义空间的预测准确率[F-->S]: %.1f%%\n', zsl\_accuracy\*100);

%[S --> F], projecting from semantic to visual space

S\_te\_pro=S\_te\_gt;

X\_te\_pro = NormalizeFea( S\_te\_pro')' \* NormalizeFea(W);

[zsl\_accuracy]= LXJ\_zsl\_el(X\_te, X\_te\_pro, param);

fprintf('[2] LXJ\_SAE\_ZSL\_AWA1 从语义空间到视觉空间的预测准确率[S-->F]: %.1f%%\n', zsl\_accuracy\*100);

toc;

***LXJ\_NormalizeFea\_AWA1.m***

***说明:特征归一化函数***

***LXJ\_SAE\_AWA1.m***

***说明:语义自编码函数***

function [W]=LXJ\_SAE\_AWA1(X,S,lambda)

%语义自编码函数

%输入:

% X: d x N的数据矩阵

% S: k x N的语义矩阵

% lambda:超参数,拉格朗日式中的lambda

%返回:一个映射W,即 k x d的矩阵

A=S\*S';

B=lambda\*X\*X';

C=(1+lambda)\*S\*X';

W=sylvester(A,B,C);

end

***LXJ\_zsl\_el.m***

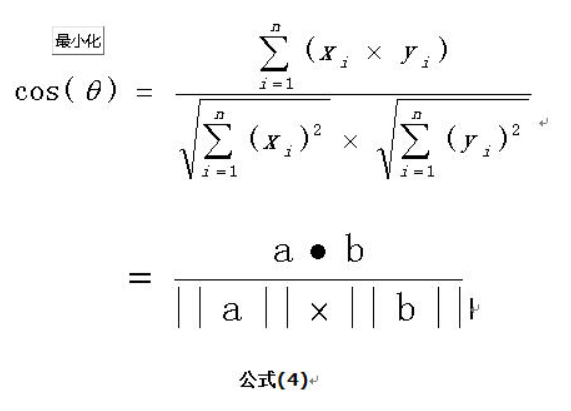
***说明:计算准确率的函数***

**3.算法流程及其相应函数的说明**

1.余弦举例的应用—cosine distance

参考网址: https://blog.csdn.net/m0\_38024592/article/details/100075613

应用场景：分析两个特征向量之间的相似性时，常使用余弦相似度来表示，其取值范围为[ -1, 1 ]。



PS：如果希望得到类似于距离的表示，余弦距离 = 1 - 余弦相似度，其取值范围为[ 0， 2 ]，即相同的两个向量余弦距离为0。

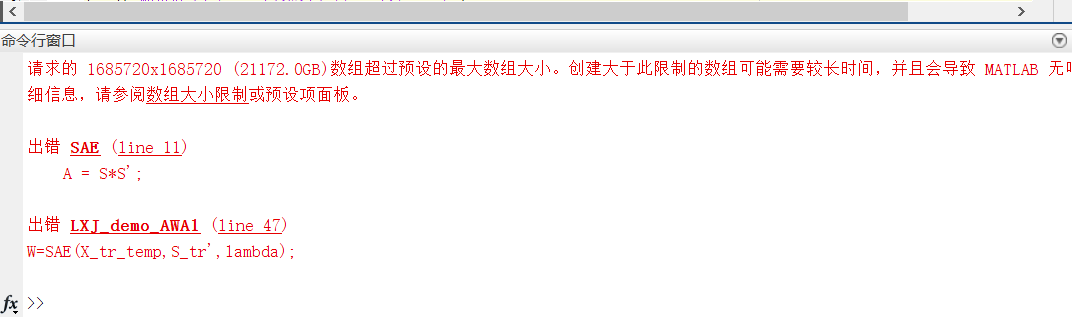
当一对文本相似度的长度差距很大、但内容相近时，如果使用词频或者词向量作为特征，它们在特征空间中的欧氏距离通常很大；而如果使用余弦相似度的话，它们之间的夹角可能很小，因而相似度高。

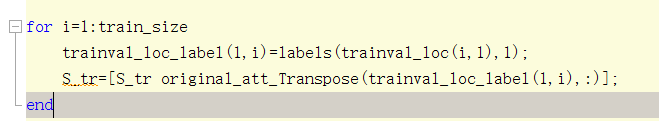
此外，在文本、图像、视频等领域，研究的对象的特征维度往往很高，余弦相似度在高维情况下依然保持“相同时为1，正交时为0，相反时为-1”的性质，而欧氏距离的数值则受维度的影响，范围不固定，并且含义比较模糊。

总体来说，欧氏距离体现数值上的**绝对差异**，而余弦距离体现方向上的**相对差异（余弦距离不是严格定义上的距离）**。

**4.错误调试过程**

1.错误分析: 创建的S\_tr矩阵过大1 x 1685720, 实际上本意是创建, 19832 x 85的semantic语义矩阵.





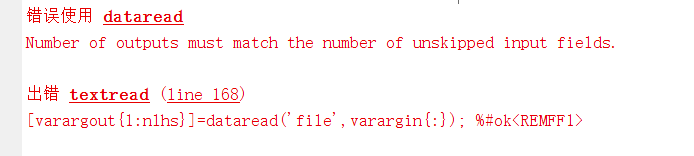
上面使用MATLAB语法使用错误:

S\_tr=[S\_tr original\_att\_Transpose(trainval\_loc\_label(1,i),:)];这行代码不能将向空矩阵中添加列向量

解决办法:

original\_att\_Transpose(trainval\_loc\_label(1,i),:)这行代码取出的是一个行向量,将其转置为列向量即可添加到改空矩阵中.

2.



依然是MATLAB语法使用错误:

[label1,label2,label3,label4,label5,label6,label7,label8,label9,label10]=textread(filename,'%s');左右两边的输入与输出不匹配,修改如下

temp\_strlabel=textread(filename,'%s');

**5.参数lambda调整过程**

1.lambda = 5; 🡪结果40.0% 46.7%

2.lambda = 2; 🡪结果34.7% 42.5%

3.lambda = 20; 🡪结果45.5% 49.8%

4.lambda = 200; 🡪结果43.7% 43.5%

5.lambda = 100; 🡪结果44.0% 46.4%

6.lambda = 20; 🡪结果45.5% 49.8%

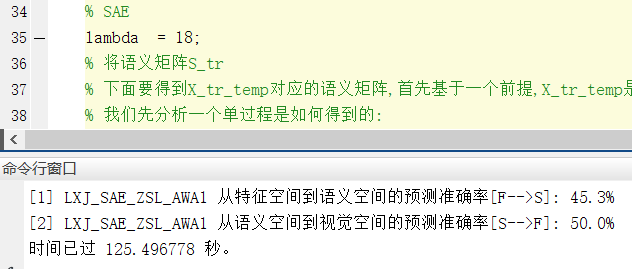
7.lambda = 50; 🡪结果44.8% 48.4%

8.lambda = 25; 🡪结果45.4% 49.5%

9.lambda = 17; 🡪结果45.2% 49.9%

10.lambda = 22; 🡪结果45.4% 49.7%

11.lambda = 18;



**5.总结**

通过不断调整lambda的值,当lambda=18时,准确率达到最大值45.3% 50.0%,与原论文的准确率(84.0% 84.7%)相差很多.

原因: 1.可能是用的数据集不一样; 2.数据集分割方式不同 3.测试数据有限, 参数lambda可能达到了局部最优,并未达到全局最优

通过这次代码复现, 对算法的数学表达有了更多的理解. 同时也对数据集在MATLAB上的处理有了一定练习. 同时也对MATLAB的使用更熟练了一些.