数字图像处理第二次大作业--Image Retrieval

第7组:余欣彤 周琳钧 梁俊邦 赵涵

摘要

本次大作业需要我们从一个图像数据库中提取图像和文字的特征并学习，然后对每个查询图片，按相关度大小降序排序，并计算MAP作为评价标准。 在NUS的数据集中，我们使用多层神经网络，可以得到0.74的MAP；而在wiki数据集中，我们使用CNN对图片和词袋模型对文档的预处理，然后实现了两种不同的学习方法：多层神经网络以及逻辑回归，MAP分别达到了0.908和0.910。

1 任务概述与初步分析

我们需要实现一个相似图片查询系统，对每个询问的图片，需要返回一些比较相近的图片给用户。

1.1 MAP

定量地， 我们对每个图片返回数据库所有图片的一个排序， 使用MAP（mean average precision）作为该系统的评价指标。MAP计算公式如下：

$$\mathrm{MAP}=\frac{\sum_{i\in T}\mathrm{AP}(i)}{|T|}$$

其中,T为测试集合。AP定义如下：

$$\mathrm{AP}(i)=\frac{\sum_j P(j,i)\cdot\delta_i(j)}{|\{j|j\in D,\delta_i(j)=1\}|}$$

其中$\delta_i(j)$为i和j是否有相同的类别，D为数据库集合。 简单来说AP(i)就是数据库中与i有相同类别的图片j的P(j,i)的均值。P(j,i)定义如下：

$$P(j,i)=\frac{|\{k|\mathrm{index}(k)\leqslant\mathrm{index}(j),k\in D,\delta_i(k)=1\}|}{|\{k|k\in D,\mathrm{index}(k)\leqslant\mathrm{index}(j)\}|}$$

其中index(j)为图片j的相关度排名。 简单来讲P(j,i)就是对询问i来说，不比j的排名后的那些图片中,真正与i有相同类别的图片的比例。

1.2 NUS数据库文件格式

NUS数据库中，没有原始的图片和文字，只有分好的500个feature和1000个tag。 对每个图片，我们知道它们拥有的各种feature的数量以及是否拥有各种tag，并且知道每个图片的类别。在这里一个图片可以有多个类别。一对图片只要有一个公共类别就算是相关。

数据库大小为100000，训练集大小30000，测试集大小2000。 使用训练集进行学习，然后对每个测试-数据库图片对进行计算。

1.3 wiki数据库文件格式

wiki数据库中只有原始的图片和文字信息。 这里每张图片有且只有一个分类。 一对图片相似当且仅当分类相同。

数据库就是训练集,大小为2173,测试集大小693。

1.4 任务统一与切分

可以看到，对于NUS数据集，我们直接就得到了处理后的数据，没有接触到源数据，而wiki数据库则只有源数据需要我们进行处理。

于是可以想到，我们对wiki数据库中的图片和文字信息提取对应的feature和tag，这样就和NUS数据库的问题一样了。 得到处理过后的数据之后，我们使用某种机器学习的方法进行学习，然后进行测试即可。其中学习方法以及相似度的计算方法都是可以变动的。

一开始我们打算在机器学习的阶段中用一个模型直接输出10种分类的可能性，后来发现这种方法效果非常差，我们改成了用10个相同的模型来训练，分别输出每一种分类的概率。 实践表明做10个分类器的效果比较好。

2 NUS-WIDE dataset解决方案

我们在这里尝试了很多种机器学习的方法，包括神经网络，SVM，LR，逻辑回归，决策树等。我们还实现了PCA降维做备用。以下分别介绍这几种做法。另外该数据集中的id存在着一个映射关系,我们先用一个程序把它们都映射到了0到20万的序号了。

2.1 多层神经网络

神经网络是一个很经典也很古老的机器学习算法，和svm相比，它不需要选择核函数，能够自适应地去学习并调整参数，是一个表现不错的算法。 对于这种只有数据没有图片而且数据之间没有明显的局部性的材料，比起卷积神经网络更加适合用多层神经网络来训练。 而多层神经网络的缺点为，它没有办法设计很多层，导致训练效果可能会很差；还有就是学习速度很慢，计算量大。 由于我们的神经网络是自己实现的，没有加多线程优化和GPU优化，速度比起其他算法低了一个数量级。

2.1.1 网络结构

第一层为输入层，第二、三层为隐层，分别有300和60个神经元，最后输出层有两个神经元，表示属于和不属于某一个分类的概率。隐层和输出层的激发函数均用的是sigmoid函数。

2.1.2 多种尝试

即便是神经网络，其内部结构也需要仔细地设计。 刚开始做的时候曾经设计过几种不同的结构如下。

由于最后是要输出两个图片的相似度，我们尝试了用对应特征差的平方来做输入，然后在训练集中随机挑取若干对来训练，最后直接用输出结果来排序。 可是由于这样训练的话测试的时候需要跑上亿次网络，时间开销太大，而且前几千次的训练中也没有看到有什么效果，故放弃了这种模型。

后来我们改成最新的模型，但是训练数据为原始的数据。 这里有一个问题就是训练数据中负例占了绝对多数，如果直接拿来训练的话最后得到的神经网络肯定会偏向判断为负例的结果。 所以这里把训练数据改成每次随机从训练集里面找,而且正负例间隔地训练。

但是这个时候还是没有效果，与期望输出的距离总是在0.5附近。 于是我们找了一个很简单的xor模型来训练，找到了网络里面的bug：每一层中我们没有加常数输出的神经元，导致学习效果很差。加上常数神经元之后,网络终于可以正常学习了,而且可以有不错的效果。

而除了网络模型之外，计算相似度的函数也对最终MAP有很大的影响。 后面的实验结果中有详细说明。

2.2 其他尝试

2.2.1 概述

为了和神经网络模型进行对比，我们利用传统的机器学习算法对数据集进行了学习和预测。本阶段我们共尝试的方法有：朴素贝叶斯算法、SVM、线性回归、Logistic回归、决策树和KNN算法，这里线性回归和KNN效果比较差，我们重点关注其他4种算法的预测精度。 实验结果在稍后会展开讲解。

2.2.2 环境

python 2.7,算法包scikit-learn

2.2.3 实验过程

NUS-WIDE数算法对feature进行降维。所以我们共进行了4组实验,训练用特征分别为：

1. Feature+tag(1500维)
2. Tag(1000维)
3. 降维后的Feature+Tag(50维)
4. 降维后的Feature+Tag(5维)

为了比较快捷方便的评判算法的精度，本阶段我们没有用MAP作为评价指标（MAP计算量十分巨大，我们要调整好多参数，时间上不允许），而是将30000数据集划分成了80%的训练集和20%的验证集， 然后对每一个Concept进行一个二分类的学习， 并引入三个评价指标：Correct（总正确率）， Tp（true-positive率）和Fn（false-negative率）， 因为数据是有偏的（偏差最小的为Concept 7，大约正例1:3负例，偏差最大的为Concept 1，大约正例1:15负例），所以单纯的评价总正确率不足够信服。 实验表明，正负例比例越悬殊，训练的数据越不精确，因此我们采用Concept 7做报告,作为一个训练精度的上界。

2.2.4 训练参数

经过调整参数,以下参数达到了各算法的最优值：
Naïve Bayes:默认
SVM:rbf核,C=2048,class_weight=auto,不归一化
DecisionTree:entropy模型,maxdepth=10
Logistic回归:默认参数

3 wikipedia dataset解决方案

由于我们提取了feature和tag之后可以使用NUS数据集的解决方法，所以这里主要介绍提取这两种数据的方法。 我们的术语中，feature指的是从图片中收集的特征，而tag指的是从文字中收集的特征。 我们使用CNN卷积神经网络作为feature的提取方法，而使用词袋模型作为tag的提取方法。

由于除了神经网络外其他机器学习方法实现略有不同,在这里稍微提及一下。

3.1 传统机器学习算法实现

3.1.1 概述

Wiki数据集有一个好处就是每张图片只有1个分类，因此从一定程度上带来了简便，本数据集上由于不再是二分类，因此DecisionTree算法不再适用，因此我们只讨论余下3种算法。 同样实验结果稍后介绍。

3.1.2 实验过程

特征采用赵涵给出的两个tag集：tag2k和tag5k，分别有约2000个tag和5000个tag。 本阶段我们采用MAP作为指标评判。 具体做法是对每个测试样例给出其分类，然后把样本集中和它分类一致的排在前面，不一致的排在后面。因此，提高MAP的关键就是在于提高分类的准确程度。

3.1.3 实验参数

经过调整参数，以下参数达到了各算法的最优值：
Naïve Bayes:默认
SVM:rbf核，C=2048，class_weight=auto，归一化
Logistic回归:C=0.01

3.2 feature

3.2.1 提取手段

对训练集和测试集的所有图片，我们使用CNN提取特征作为机器学习的输入。

3.2.2 CNN模型

首先按照训练集列表中的文件名找到图片文件，由于每张图片大小不同，无法直接作为CNN的输入，因此统一resize成100*100大小的图片。 然后使用DeepLearnToolbox的CNN函数，网络结构为默认的6c-2s-12c-2s结构，训练过程迭代10次。 其中c为卷积层，s为pooling层。 前面的数为该层的单元数量。如下图所示。

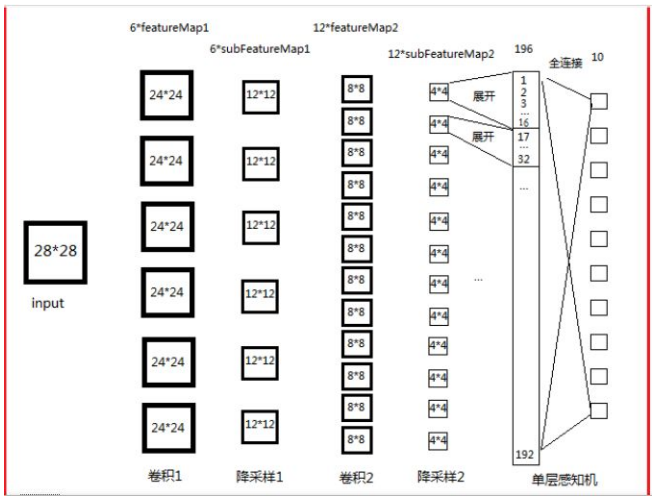

图 1.

我们曾经尝试只使用一个CNN神经网络，输出10个值表示该图片被分类为某个类别的可能性。网络设定为每张图片的输入为10000维的向量，输出为10维的向量，分别对应10个类别。 训练过程中每次迭代时间约为70s。 对测试集，错误率为86.15%，且输出的10维向量均为极小的浮点数，大小为1e-19至1e-58不等，无法归一化也无法直接作为图片的特征向量供MLP使用。 将迭代次数增加到15次，结果没有变化，说明CNN网络已经收敛。

后来我们使用10个神经网络，每个对一个单独的类别进行分类。 网络设定为对每一类单独训练一个CNN，将输出的10个结果作为图片的特征向量。 训练过程中每次迭代时间约为70s。 输出的结果仍然是极小的浮点数，但量级均为1e-78，归一化后的结果十分位不为0。 以0.5为分类阈值，可得训练集对每一个类别的正确率如下。

类别序号	true-positive	false-negative	总正确率
1	0.1594	0.8211	0.7791
2	0.3713	0.6512	0.6162
3	0.3607	0.5682	0.5449
4	0.3548	0.7429	0.6986
5	0.3861	0.6854	0.6576
6	0.1067	0.7158	0.6659
7	0.1559	0.7564	0.8050
8	0.2083	0.8034	0.7639
9	0.1121	0.7907	0.7239
10	0.5274	0.6364	0.6190

表格 1.

其中，true-positive为正例中的正确率，false-negative为负例中的正确率，下同。 由于每一类的true和false样本数量不同，因此分类器倾向于把图片分为negative的，导致f-n较高。

3.3 tag

在这里处理wikiset的文字部分，得到一系列的tag，并且对每一篇文章，得到对应tag的值。我们主要收集单词的tf-idf值然后筛选值比较大的单词作为tag。

3.3.1 tf-idf算法

tf-idf算法全称为term frequency－inverse document frequency，是一种用于资讯检索与资讯探勘的常用加权技术。 我们利用这种方法可以有效的提取出具有特征的关键词。 计算公式如下：

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$$\mathrm{idf}_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

tf指的是词频，也就是一个词在某一篇文章中出现的频率；idf指的是逆向文档频率，指的是在所有文档中，出现该单词文档频率的逆向频率（取log，分母实际处理中要加一）。 当一个单词在某一篇文章中出现很多，而在其他文章中出现较少时，tf-idf会比较大，可以认为是该文章的特征单词。

3.3.2 具体做法

我们将所有文章按照类别合并成十篇长文章， 这样使得相同类别的特征单词聚在一个文章中，便于使用tf-idf进行处理。 一开始我们没有考虑到将所有文档合并成10篇长文档，因此虽然我得到了不少tf-idf较大的值，但是这些词基本都仅仅属于某一篇文章，对于这一类文章没什么帮助。 后来我们发现每篇文章都比较短而且相同单词出现的频率比较少，把同类的文章合并起来有利于我们找到真正出现频率高的单词。

对于刚刚处理好的10篇长文章，我们先对其进行转成大写的操作，然后进行分词。 分词策略比较简单，即认为非字母的部分都是分隔符。 对于每个单词，统计在这10篇文档中出现的文档个数并且记录，之后便可求出每个单词的idf。 然后统计每个单词在10篇文档中分别出现的频率，得到tf。之后用tf和idf计算出每个单词对每篇文档的tf-idf。

刚刚我们得到了每个单词对每篇文档的tf-idf的值，然后我们可以设定threshold = 5e-5（经不断调整后得到）。 对于所有单词，如果该单词在某一篇文章中得tf-idf，大于threshold，则认为该单词是特征单词，保留为tag。 我们曾经把阈值设置为1e-4，得到了两千个左右的单词，现在是五千个左右。稍后我们会看到单词数量的不同对学习效果的影响。

重新遍历所有文档，然后遍历所有tag，计算每篇文档中所有tag的出现频率。 由于频率实在很小（1e-3量级）为了保证数据范围便于操作， 我们将频率扩大了1000倍， 这样结果保持在1~10左右,有利于之后处理。

3.3.3 处理结果

我们用上述方法总共处理出了5427个tag。 经观察发现，大多数tag都主要聚集在10个分类中的某一类，且确实具有实际意义，说明处理效果很好。 处理后的文件在附件中可以找得到，稍后会详细提及。

4 实验效果

4.1 神经网络

在NUS数据集上，网络输入为1500个feature和tag，训练一轮需要5分钟左右，测试一轮需要15分钟，所以对全部的类别都学习一遍的话需要3个半小时，MAP最高可以到0.73。 改成训练3轮后,MAP稍微提升到了0.74。由于学习时间太长而且提升甚微,没有再改成更多的训练轮数。

在wiki数据集上， 如果输入为2000个tag和10个feature的话， 训练测试一共需要30分钟左右， MAP最高可以到0.83； 把tag的数量改成5000个的话， 耗时在2个小时以内， MAP最高可以到0.908。

值得一提的是，即便网络模型相同，不同的计算相似度的函数对最终MAP的影响非常大。 刚开始我们的函数设计如下：如果两者在某一维的输出均大于0.6，则加上1，否则加上它们的乘积，以这10个数的和作为相似度来排序。 这样的函数在NUS中MAP只有0.48左右，导致一度以为神经网络不可行。

后来我们改成了如下函数：把10维向量当做是该图片被分为某类的概率，计算两张图片拥有至少一个相同类别的概率作为相似度。使用这个函数可以让MAP达到0.73,wiki也提高到了0.82。

然后我们观察了wiki中AP比较低的一些数据， 发现其输出很异常：有多个类别的输出均有0.9以上，有些甚至为1，这样导致和本来不相关的一些图片相似度也很高。 我们观察到虽然输出有几个是很高， 但是最高的仍然是对的分类， 所以我们加上了一条： 如果最高分类不相同，那么相似度要乘上一个惩罚系数0.2。 加上这条之后wiki的MAP提高了0.84。 可是由于NUS中可以有多个类别,这种方法无法应用。

接着我们看到有些图片被完全分到了其他类别去了。 我们仔细研究了其中一个，第682个测试图片。 它的文章中只有3个tag。 其中一个还是armor，这就不难解释为什么它会被分到warfare类别中了。 它本来的类别是history，历史的文章中总是有很多人名和地名，而它们又不是经常用到的，所以在2000个tag中基本不会出现。 所以我们想到可以增加tag的数量。 增加到5000个之后果然分类效果有了明显的提高,可以到0.88了。

针对上面的多输出很大的情况，还有一种方法解决，就是归一化向量。 把它们的1范数都归一化成1,这样的输出就是完全的概率分布了。所以MAP也相应提高到了0.908。

4.2 其他算法

4.2.1 NUS

结果如下图。

	all_data			tag_data			ft50			ft5		
	Correct	Tp	Fn	Correct	Tp	Fn	Correct	Tp	Fn	Correct	Tp	Fn
SVM	0.6776	0.1326	0.8747	0.5498	0.4818	0.5744	0.7239	0.0226	0.9775	0.6319	0.2267	0.7785
NB	0.4918	0.6421	0.4397	0.4985	0.6337	0.4516	0.438	0.6227	0.374	0.7425	0	1
LR	0.7257	0.0673	0.954	0.7367	0.0511	0.9744	0.7423	0	1	0.7425	0	1
Dtree	0.616	0.257	0.7405	0.6403	0.2673	0.7697	0.6058	0.2887	0.7158	0.6153	0.2641	0.7371

图 2.

与神经网络模型相比，传统的机器学习算法要相对差一点，SVM、LR、Dtree对于负例划分的很好，但正例比较差，而Naïve Bayes则对正例划分较好，负例较差。 从整体角度来看分类的效果并不尽如人意，这与后面的Wiki数据集形成了较大的反差。 主要原因是：首先，Tag并不十分精确，一个是tag数比较少，第二个是并没有用tf-idf算法估计每个tag的权重，只是简单的给出了tag在文本是否存在，其次，feature对图片预测的效果本身并不好，准确率很低。 另外，将feature和tag拼在一起后，feature给出的信息要多于tag，因此PCA降维之后绝大部分的信息都偏向于feature，可以说一定程度上等同于只用feature降维，因此这些都给最终的预测造成了一定的困难。

训练时间：除了SVM都很快，毕竟SVM复杂度在O(n^2*Feature)和O(n^3*Feature)之间，这样，随着数据集的增长时间增长的非常快。SVM在All Feature中大概运行20分钟左右。其他算法在All Feature中速度不超过1分钟。

4.2.2 wiki

结果如下图。

	Correct	MAP
SVM(2k)	0.772	0.7916
SVM(5k)	0.8716	0.8825
SVM(norm, 2k)	0.7994	
SVM(norm, 5k)	0.8571	
LR(2k)	0.8773	0.8876
LR(5k)	0.9019	0.9103
NB(2k)	0.8355	0.8501
NB(5k)	0.8615	0.8735

图 3.

可以看出我们提取的feature训练出来的结果非常好，其中Logictic Regression（5k）的MAP可以达到0.9103,略优于神经网络模型,SVM和Naïve Bayes也能够达到88%和87%和MAP。

训练时间:依然是除了SVM都很快,SVM大概要1分钟,其他算法基本上几秒钟出结果。

5 实验结果、代码、和数据

我们在github上有一个工作目录:https://github.com/zhoulinjun1994/Image-Retrieval

5.1 NUS dataset

neural_network中，new_*id.txt为映射后的图片序号。 mlp.cpp和neuron.h为神经网络的源代码。 check.cpp为计算MAP的程序。 id_change.cpp为映射序号的源程序。 可以以以下顺序运行，无需其他所有的预处理文件：id_change--mlp--check。 运行神经网络时需要把new_*id.txt以及可执行程序都放到NUS文件夹的根目录下。train1和train3为训练1轮和3轮的输出。con0~9.txt为对应concept的数据库与测试集的输出。 前面100000行为数据库对应的输出，后2000行为测试集的。运行check时需要把con0~9.txt也放到NUS文件夹的根目录下。 log.txt为控制台输出。 mlp最后会顺带计算MAP。

others中为其他机器学习方法的汇总。

实验运行方法：

python trainready.py ： 准备数据

python train.py ： 训练，并给出在验证集下的精度，程序中注释掉的几行为可选机器学习算法

/Data中保存着一些供训练用的数据,其中：

ft50.txt为降为50维后的数据

ft5.txt为降为5维后的数据

trdata_all.txt为全部feature+tag

trdata.txt全部tag

trdata_feature.txt为全部feature

5.2 wiki dataset

tag中为对文章预处理的程序和文件。 tag.cpp为生成5000个tag的程序。 TE.cpp根据5000个tag生成每张图片拥有的tag的数量。 tag.cpp后面注释的部分根据5000个tag生成一张表， 每行一个tag， 有10个数表示每一类中出现这个tag的频率。 tags2_stats_2.txt就是这张表，tags2.txt和*set_tag.txt都是这几个程序的输出，但是只有2000个tag。 5000个tag的数据在机器学习算法对应的文件夹中有。运行这些程序时需要把它们放在wiki文件夹的根目录下。

cnn中为我们所使用的cnn的代码和文件。main.m里放了使用的指令。

neural_network中为神经网络的源程序和输出结果。 分为2ktag和5ktag两个文件夹，里面的文件意义与NUS的相同。根目录的三个程序意义和用法也与NUS的相同。

others中存放着其他机器学习算法的数据和代码。
python train.py训练,并生成预测的结果文件result.txt。
数据：

testset_tag2k.txt 2k tag的测试集
testset_tag5k.txt 5k tag的测试集
trainset_tag2k.txt 2k tag的训练集
trainset_tag5k.txt 5k tag的测试集

6 引用文献

R.B.Palm, Prediction as a candidate for learning deep direrarchical models of data , 2012.