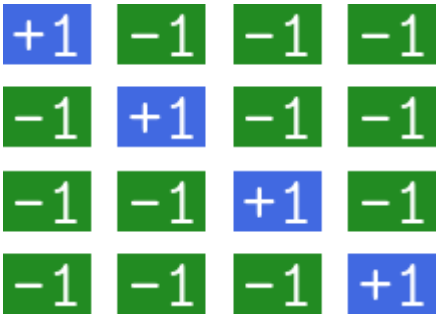
机器学习导论（7）–多分类学习之ECOC编码处理

Python admin 2年前 (2016-11-12) 1411次浏览 0个评论 扫描二维码

ECOC 是 Error-Correcting Output Codes 的缩写。中提到 ECOC 可以用来将 Multiclass Learning 问题转化为 Binary Classification 问题，本文中我们将对这个方法进行介绍。

要了解 ECOC，可以从 One-vs-Rest 的 Multiclass Learning 策略出发。回忆一下，对于一个 K 类的分类问题，One-vs-Rest 策略为每一个类 i 都训练一个 binary classifier，用于区分“类别 i”和“非类别 i”两类。对于这个策略，可以用下面这样一个图来表示（假设我们有四个类）：

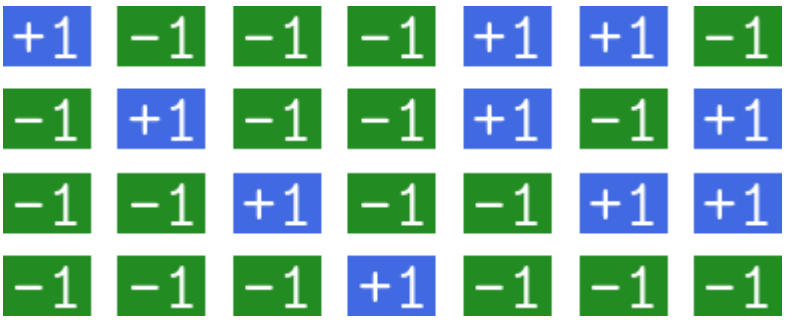


这个表中，每一行代表一个类，而每一列代表一个 binary classifier，其中表格内的元素表示该列所对应的 binary 问题中，该行所对应的类别的数据被当作正例还是负例。例如，第一列表示该 binary classifier 是通过把第一类当作正类 (+1)，剩余的第二、三、四类当作负类 (-1) 而训练出来的。如此类推。

现在我们将这个表格称作 ECOC codebook，而把表格的列数称作是 ECOC 的 code length L，并且不再要求 L和类别数 K 相等。在 codebook 中，每个类别会对应一个长度为 L 的 code 。例如，在刚才的例子中，第一类所对应的 code 为 +1-1-1-1 ，第四类所对应的 code 为 -1-1-1+1 。

对于给定的 codebook，我们可以训练出 L 个二分类器，可以看到，如果使用上面的这个 codebook，那么这个训练过程其实就和普通的 One-vs-Rest 策略完全一样。在进行预测的时候，通过使用 L 个分类器，可以得到 L 个二分类结果，把这些结果串起来就组成了一个 code，将这个 code 与每个类别所在 codebook 里的 code 进行比较，距离最小的那个类别将作为最终分类类别。计算 code 之间的距离有很多种方法，最简单的就是 Hamming Distance，简单地来说就是数两个 code 不相同的位数，例如 +1-1-1-1 和 -1+1-1-1 的 Hamming Distance 就是 2 。

那么，ECOC 为什么称为“Error Correcting”呢？从上面这个这个最简单的 One-vs-Rest 的 codebook 上看不出这样的好处，不过下面让我们再把 code length 增大一点，看如下的 codebook：



可以看到，我们在原来的基础上增加了三个二分类器。这样一来，不同类别所对应的 code 之间的 Hamming Distance 也增大了，比如，第一类和第二类之间的 Hamming Distance 现在变成了 4 ，于是这个 codebook 也就具有了一定的错误修正的能力。比如说，属于第三类的一个数据点 x ，理想情况下，所有的二分类器都预测正确，将会得到 code -1-1+1-1-1+1+1 ，但是如果中途发生了错误，例如，第三个分类器预测错误，得到了结果 -1 ，那么 code 变成 -1-1-1-1-1+1+1 ，不过，计算该 code 和四个类别的 code 之间的 Hamming Distance ，会得到 3、3、1、3 ，选择距离最小的类别，仍然得到第三类。所以说，ECOC 当 code length 足够长的时候，能够容忍其中小部分 binary classifier 发生错误而不影响最终结果。这就是 Error Correcting 的意思。

当然 code length 是不能任意长的。一方面，L 越大，不管从训练还是预测来看，计算量都会增大；另一方面，由于类别数目的限制，可能的组合数目是有限的，（如果 binary classifier 所用的算法都是一样的并且没有额外的随即性的话），那么重复的 code 列是没有用的。另外需要注意的是，把一列里的 +1 和 -1 相互交换一下得到的列和原来的列也是完全重复的。

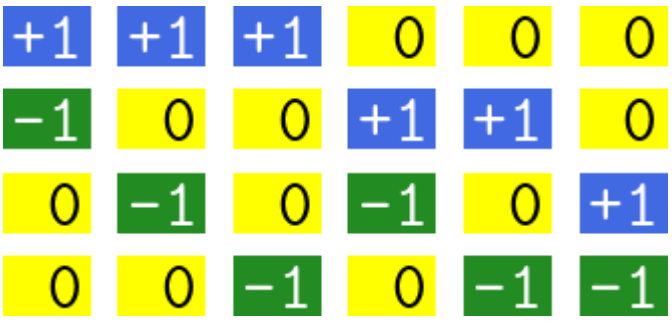
这里 Error Correcting 的原理其实很简单：如果不同类别的 code 之间的距离互相很大的话，那么某个类别对应的正确 code 即使在一些位上发生错误，从距离上来看仍然最接近原来的 code 的话（发生错误的位数小于类别间相互距离的一半），就仍然能够分类正确。于是设计 ECOC codebook 的原则也就显而易见了：尽量使得任意两个类别的 code 之间的距离的最小值达到最大。不过，给定一个 code length L ，我们也并不能通过暴力枚举所有码长度为 L的 codebook 选出最好的（最小距离最大的那个）码表，因为这是个 NP 问题，稍微大一点的L 就算不了了。而且，通常也没有必要得到最好的那个 codebook ，其实也并不是 codebook 本身的性质越好，最终的分类问题就处理得越好，其中还牵涉到很多其他因素，比如，不同类别之间的区分度不一样，不同的 code 列所得到的二分类问题的难度也会因此而相差很大，一个本身性质很好，但是对应的 binary classification 问题都很难（于是就很容易分错）的 codebook ，和另一个性质稍差一些，但是对应的二分类问题都相对简单的 codebook ，谁的效果更好也是不好说的。

除了暴力搜索之外，可以用一些代数方法来生成性质良好的 codebook ，例如 BCH Code 。不过，在分类问题中，似乎用得更多的是另一种更加简单易懂的 codebook 生成方式：随机。简单的来说，就是随机生成出很多很多个 codebook （例如 10000 个），然后从中挑一个最好的。另外在下面这篇论文中还给出了 margin-based classifier 作为基础的 binary classifier 的情况下的一些理论分析和实验结果，并给出了建议的 code 长度为 L=10logK：

E. Allwein, R. Schapire, and Y. Singer. *Reducing multiclass to binary: A unifying approach for margin classifiers*. Journal of Machine Learning Research, 1:113-141, 2002.

当然，在生成 codebook 的时候需要注意丢掉那些 invalid 的 codebook ，最明显的情况就是某一列只出现了 +1 或者 -1 ，这样并不能构成一个合法的二分类问题。此外，去除可能重复的列，可以避免不必要的计算量。

上面从 One-vs-Rest 出发简单介绍了一下 ECOC 处理 Multiclass 问题的方法，而 One-vs-Rest 可以看成是一个特殊的 ECOC codebook 。实际上，只要稍加扩展，也可以包含 One-vs-One 策略：这里我们在 codebook 中除了允许 +1 和 -1 之外，还加入 0 ，例如下面的 codebook：



和以前一样，每一列对应一个二分类器，+1 对应正类，-1 对应负类，而 0 在这里表示忽略该类。例如，第一列就表示训练一个用于区分第一类和第二类的二分类器，而忽略第三类和第四类。其他的类推，实际上很容易就看出来，这个扩展的 codebook 实际上就是 ECOC 版的 One-vs-One：训练了任选两类的所有组合的二分类器。允许了 0 存在于 codebook 中之后，同样可以使用随机方法来生成除了 One-vs-One 之外的其他 codebook，因为 0 表示忽略该类别，所以这样的 codebook 称作是 sparse codebook，一般会令 $L=15\log K$。随机的时候设定三个概率 $P(0)+P(+1)+P(-1)=1$，不过 sparse codebook 在随机生成的时候比 dense codebook 更容易得到 invalid 的 codebook：特别是在 K 比较小的时候，很容易出现没有 +1 或者没有 -1 的列。一个 workaround 就是，在生成每一列的时候，首先直接随机选定两个位置，赋值为 +1 和 -1 （或 -1 和 +1），然后剩下的位置在使用正常的方法采样 0、+1、-1 进行赋值，这样就能保证每一列总是至少同时有 +1 和 -1 出现了。不过这样做会使得实际的三个概率值发生变化，如果记变化后的概率为 Q 的话，那么我们可以得到下面的关系：

$$Q(0)Q(+1)Q(-1)===K-1KP(0)1K+K-2KP(+1)1K+K-2KP(-1)$$

有兴趣的小朋友可以自己算一下。不过一般生成随机 codebook 的时候几个概率值似乎有没有说一定要取多少，所以发生一些变化大概也没有太大的影响。

除了这些生成 codebook 的方法之外，还有一类叫做 data dependent 的方法，根据具体问题的数据来学习出特定的 codebook。例如在 Shogun 中也有从 ecoclib 移植过来的用于学习一个树形的 codebook 的 data dependent Encoder：`ECOCDiscriminantEncoder`，它试图最大化二分类器对应问题中两类数据的可区分性。

各种花哨的 codebook 生成的方法的介绍就先暂时告一段落吧，其实 ECOC 作为一种看待问题的角度来说的话，是很有意思的，它把之前的许多多类到二类问题的转化策略统一起来。事实上，每一个类对应一个 code，其实就相当于降维了，把数据降到 $\{+1,-1\}^L$ 或者 $\{+1,0,-1\}^L$ 这个离散空间中，啊，这种目标是离散编码的降维似乎更多地被机器学习领域称作 Hashing，当然 hashing 问题和分类问题还是各自有各自的侧重点的，所以也不能一概而论，但是 ECOC code 在 hashing 中也确实有应用，比如小鱼童鞋就曾经做过这方面的工作。

不过至于 ECOC 方法到底效果有多好，或者不同的更加花哨或者更加复杂的 codebook 究竟对实际性能有多大提高呢？我也不清楚了，因为我也没有在实际问题中尝试过这样的方法。下面用 Shogun 中目前有的各种 ECOC 编码来做一个简单的实验。代码如下：

```
from scipy.io import loadmat

mat = loadmat('uci-20070111-optdigits.mat')['int0'].astype(float)
X = mat[:-1,:]
Y = mat[-1,:]
isplit = X.shape[1]/2
traindat = X[:,:isplit]
label_traindat = Y[:isplit]
testdat = X[:, isplit:]
label_testdat = Y[isplit:]


import shogun.Classifier as Classifier
from Classifier import LibLinear, L2R_L2LOSS_SVC,
from Classifier import ECOCStrategy, LinearMulticlassMachine
from shogun.Features import RealFeatures, MulticlassLabels
from shogun.Kernel import GaussianKernel
from shogun.Evaluation import MulticlassAccuracy

def nonabstract_class(name):
    try:
        getattr(Classifier, name)()
    except TypeError:
```

```
        return False
    return True

import re
encoders = [x for x in dir(Classifier)
        if re.match(r'ECOC.+Encoder', x) and nonabstract_class(x)]
decoders = [x for x in dir(Classifier)
        if re.match(r'ECOC.+Decoder', x) and nonabstract_class(x)]

fea_train = RealFeatures(traindat)
fea_test  = RealFeatures(testdat)
gnd_train = MulticlassLabels(label_traindat)
if label_testdat is None:
    gnd_test = None
else:
    gnd_test = MulticlassLabels(label_testdat)

base_classifier = LibLinear(L2R_L2LOSS_SVC)

print('Testing with %d encoders and %d decoders' % (len(encoders), len(decoders)))
print('-' * 70)
format_str = '%%15s + %%-10s  %%-10%s %%-10%s %%-10%s'
print((format_str % ('s', 's', 's')) %
                ('encoder', 'decoder', 'codelen', 'time', 'accuracy'))

def run_ecoc(ier, idr):
    encoder = getattr(Classifier, encoders[ier])()
    decoder = getattr(Classifier, decoders[idr])()

    # whether encoder is data dependent
    if hasattr(encoder, 'set_labels'):
        encoder.set_labels(gnd_train)
        encoder.set_features(fea_train)

    strategy = ECOCStrategy(encoder, decoder)
    classifier = LinearMulticlassMachine(strategy, fea_train,
                        base_classifier, gnd_train)
    classifier.train()
    label_pred = classifier.apply(fea_test)
    if gnd_test is not None:
        evaluator = MulticlassAccuracy()
        acc = evaluator.evaluate(label_pred, gnd_test)
    else:
        acc = None

    return (classifier.get_num_machines(), acc)


import time
for ier in range(len(encoders)):
    for idr in range(len(decoders)):
        t_begin = time.clock()
        (codelen, acc) = run_ecoc(ier, idr)
        if acc is None:
            acc_fmt = 's'
            acc = 'N/A'
        else:
            acc_fmt = '.4f'

        t_elapse = time.clock() - t_begin
        print((format_str % ('d', '.3f', acc_fmt)) %
                (encoders[ier][4:-7], decoders[idr][4:-7],
                                    codelen, t_elapse, acc))
```

这里使用了 UCI 20070111 optdigits 数据作为测试。注意这只是一个简单的 code demonstration 主要目的是为了示范一下用 Shogun 的 Python modular interface 还是比较方便

的。不过在实验结果的 Accuracy 上的参考价值就非常小，因为我这里偷懒直接挑了速度相对较快的 LibLinear 作为基础的二分类器，并且完全没有调整任何参数。结果如下：

```
Testing with 6 encoders and 5 decoders
----------------------------------------------------------------------
        encoder + decoder     codelen    time       accuracy
   Discriminant + AED         9          2.830      0.0875
   Discriminant + ED          9          2.960      0.1000
   Discriminant + HD          9          2.820      0.0993
   Discriminant + IHD         9          2.770      0.1050
   Discriminant + LLB         9          2.780      0.1007
         Forest + AED         27         9.380      0.0982
         Forest + ED          27         9.290      0.1000
         Forest + HD          27         8.610      0.0975
         Forest + IHD         27         8.680      0.1046
         Forest + LLB         27         8.730      0.1004
            OVO + AED         45         2.920      0.0975
            OVO + ED          45         2.920      0.0975
            OVO + HD          45         2.860      0.0961
            OVO + IHD         45         2.770      0.0961
            OVO + LLB         45         2.690      0.0975
            OVR + AED         10         1.030      0.0954
            OVR + ED          10         1.000      0.0954
            OVR + HD          10         1.000      0.0925
            OVR + IHD         10         0.970      0.0947
            OVR + LLB         10         0.990      0.0954
    RandomDense + AED         23         12.370     0.0947
    RandomDense + ED          23         13.230     0.0975
    RandomDense + HD          23         13.580     0.0982
    RandomDense + IHD         23         14.390     0.0979
    RandomDense + LLB         23         15.270     0.0968
  RandomSparse + AED          35         6.330      0.1032
  RandomSparse + ED           35         8.200      0.1004
  RandomSparse + HD           35         7.780      0.0957
  RandomSparse + IHD          35         6.240      0.0972
  RandomSparse + LLB          35         7.780      0.0975
```

上面的结果中其实大家都半斤八两的样子…… =.=bb 嗯，也许用 LibLinear 来做这个数据本身就不合适，好吧，我承认其实这个例子其实只是从 Shogun 的功能测试的 test case 里抽取出来的，不过毕竟这一块也是我在做的嘛，也不能算是“盗用代码”哈哈……不过这些功能目前都只在 git 的最新版里才有，大概等到今年的 GSoC 结束以后会发布一个新版本吧。

最后我再来简单地提一下 decoding 的问题。前面我们说了，最简单了 decoding 方法就是计算 Hamming Distance 。不过从上面这个例子中也可以看到，decoding 的方法应该还不少，在 ecoclib 中还有更多的 decoder ，不过我觉得没有必要并且容易把用户搞晕掉就只实现了几个比较有代表性的。

其实细心的小朋友应该发现，我虽然一直在说 ECOC 的 One-vs-Rest codebook 和传统的 One-vs-Rest 策略的等价性，但是他们其实也并不是完全等价的：或者更准确地来说，如果使用 Hamming Distance decoding 的话，他们的结果还是会有所差别的。因为传统的 OvR 策略直接采取二分类器中 decision value 最大的那个所对应的类别，这里的 decision value 具体是什么取决于底下用的是什么二分类器，比如 SVM 可以给出数据点到分隔面的距离，Logistic Regression 可以给出概率值等等。而 Hamming Decoding 则先把二分类器们的结果转化为一个 binary code ，然后通过距离的方法来分类。第一步 binary 化的时候实际上损失了挺多信息；而且即使第一步不进行 binary 化直接计算距离，也并不一定总是会得到和“最大 decision value 分类”同样的结果。

不过好在有另一种所谓的 loss-based decoding 方法，可以做到和传统的 OvR 完全等价。考虑一个 loss function ℓ ，并记 codebook 为 B ，其中 Bij 为 B 的第 i 行第 j 列所对应的元素。记数据 x 的分类结果为

$f(x)=(f1(x),...,fL(x))$

注意这里我们保留原始的 decision value ， 并没有做二值化。那么我们定义它与 codebook 的第 i 行 Bi 的距离为

$d\ell(Bi,f(x))=\sum j=1L\ell(Mij,fj(x))$

啊，其实简单地说就是把 L 个 loss 全部加起来了。然后我们这里用一个简单的 loss

$\ell(y,f)=-yf$

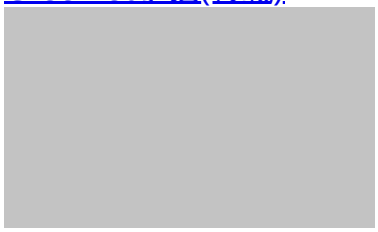
就可以达到完全和 OvR 等价了。例如一个预测结果 f 中最大的那个元素是 fk ， 那么传统的 OvR 会将分类结果定为 k ， 那么用 ECOC 的 loss-based decoding 加上上面这个 loss 呢？ 首先注意到 OvR 的 ECOC codebook 每一行都只有一个 +1 ， 其他的全是 -1 。由于 fk 是所有$fj,j=1,...,L$ 中最大的，所以我们有

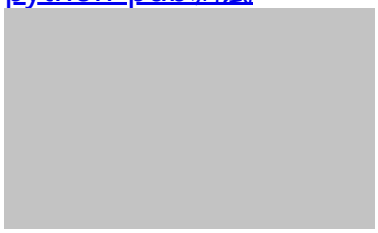
$fk-\sum j\neq kfj>fk'-\sum j\neq k'fj,k'\neq k$

于是， $d\ell(Bk,f(x))$ 肯定就是最小的啦，于是两个结果就会一样了。

Deeplearn, 版权所有丨如未注明，均为原创丨本网站采用BY-NC-SA协议进行授权，转载请注明机器学习导论（7）–多分类学习之 ECOC 编码处理!
喜欢 (1)
Python
win64下安装机器学习库问题 机器学习导论（8）–随机森林

关于作者：admin
作者主页

- 
Greenlet小记(转载)
- 
python设计模式–命令模式

- 
  python-pdb调试
- 
  python设计模式–职责链模式

- python设计模式–代理模式
- python设计模式–mvc控制模式
- python设计模式–享元模式
- python设计模式–装饰器模式
- python设计模式-适配器模式
- scipy稀疏矩阵模块
- python设计模式–原型模式
- python-sort函数key解析

您必须 登录 才能发表评论！