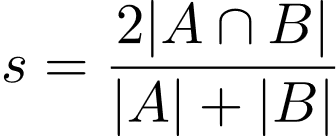
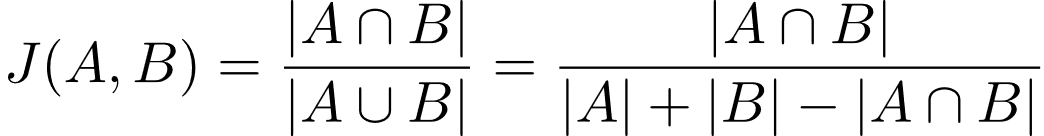
关于图像的分析：

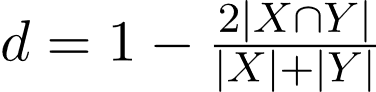
Dice距离用于度量两个集合的相似性，因为可以把字符串理解为一种集合，因此Dice距离也会用于度量字符串的相似性。此外，Dice系数的一个非常著名的使用即实验性能评测的F1值。Dice系数定义如下：



其中分子是A与B的交集数量的两倍，分母为X和Y的长度之和，所以他的范围也在0到1之间。从公式看，Dice系数和Jaccard非常的类似。Jaccard是在分子和分母上都减去了|A∩B|。

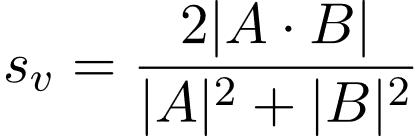


与Jaccard不同的是，相应的差异函数



不是一个合适的距离度量措施，因为它没有三角形不等性的性质。例如给定 {a}, {b}, 和 {a,b}, 前两个集合的距离为1， 而第三个集合和其他任意两个集合的距离为三分之一。

与Jaccard类似, 集合操作可以用两个向量A和B的操作来表示:



利用Python，实现代码为：

def dice\_coefficient(a, b):

"""dice coefficient 2nt/na + nb."""

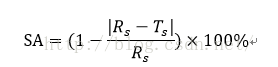
a\_bigrams = set(a)

b\_bigrams = set(b)

overlap = len(a\_bigrams & b\_bigrams)

return overlap \* 2.0/(len(a\_bigrams) + len(b\_bigrams))

关于分割精度：



其中Rs表示专家手工勾画出的分割图像的参考面积，Ts表示算法分割得到的图像的真实面积，|Rs-Ts|表示错误分割的像素点个数。

分割在图像参考面积之外的像素点的比率即为过分割率：

https://img-blog.csdn.net/20161101002459046?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

含义同上，Os表示本不应该包含在分割结果中的像素点个数，实际却在分割结果中的像素点个数。换句话讲，Os中的像素点出现在实际分割图像中，但不出现在理论分割图像Rs中。

分割在GT图像参考面积之中欠缺的像素点的比率即为欠分割率：

https://img-blog.csdn.net/20161101002503838?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

含义同上，Us表示本应该包含在分割结果中的像素点个数，实际却不在分割结果中的像素点个数。换句话讲，Us中的像素点出现在理论分割图像中，但不出现在实际分割图像中。

关于DICOM格式的图像：

医学CT成像设备使用X射线（一种电磁波）得到人体的结构和功能信息。CT影像是基于X射线吸收剖面的重构图像，由于不同物质和组织吸收X射线能力不同，因此X射线可用于诊断。CT成像作为当前多类疾病实体诊断的金标准，广泛应用于大脑、肝脏、胸部、腹部、骨盆、脊柱等身体部位以及CT血管造影的早期诊断筛查。但是与MR图像相比较，CT图像敏感性和特异性相对较差。

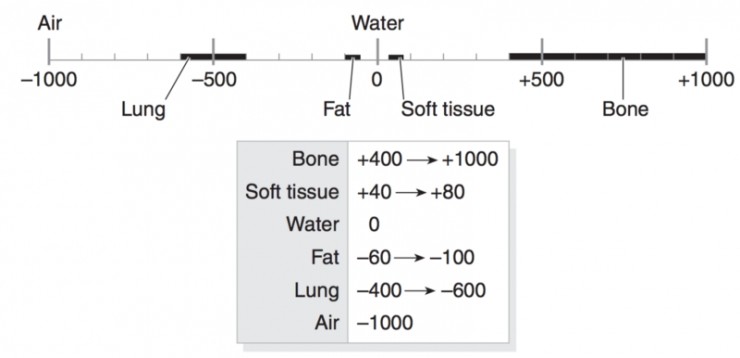
CT成像中的伪影包括：部分容积效应、条形伪影、运动伪影、束硬化伪影、环状伪影、金属伪影等。由于这些伪影的存在给CT图像分割带来了一定的难度，不同组织部位分割精度也不一样。

DICOM通讯协议是用来在档案中搜索影像研究，并将影像研究还原显示的。所有连接了医院网络的医学成像应用程序都会使用DICOM协议交换信息，这些信息中的大部分是DICOM图像，不过还包括了一些患者信息和治疗方案。还有一些网络要求是用于控制和跟踪手术、安排手术日程、报告状态，以及分担医生和成像设备之间的工作量的。

CT扫描测量的单元是“胡斯菲尔德单元”（HU），这个单元测量的是放射性密度。为了得到精确的测量结果，CT扫描仪经过了严格的校准。下面是关于CT扫描测量的细节内容：

每个像素都会被分配一个数值（CT数），这个数值是相应的voxel内所有衰减值的平均值。这个数字是与水的衰减值相比较得出的，而且是以任意单元的规模显示的，这个任意单元叫做“胡斯菲尔德单元”（HU），是以Godfrey Hounsfield先生的名字命名的。

这个任意单元的规模将水的衰减值定为零。CT数字的范围是2000HU，尽管有一些现代扫描仪的HU范围达到了4000。每个数值都代表了一种灰色阴影，在光谱两端有+1000白色和-1000黑色。



有些扫描仪是有柱状扫描边界的，但是其输出的图像确实方形的。在扫描边界之外的像素将被赋予-2000的定值

医学图像分割是医学图像处理与分析领域的复杂而关键的步骤，其目的是将医学图像中具有某些特殊含义的部分分割出来，并提取相关特征，为临床诊疗和病理学研究提供可靠的依据，辅助医生做出更为准确的诊断。图像分割过程是把图像分割成多个区域，这些区域内部有类似的性质，如灰度、颜色、纹理、亮度、对比度等。医学图像分割的目标是(以放射治疗为例)：(1)研究解剖结构；(2)识别感兴趣区域(即定位肿瘤、病变和其他异常组织)；(3)测量组织体积；(4)观察肿瘤生长或治疗中肿瘤体积的减少，为治疗前的计划和治疗中提供帮助；(5)辐射剂量计算。

从医学图像中自动分割出目标是个艰巨的任务，因为医学图像具有较高的复杂性且缺少简单的线性特征；此外分割结果的准确率还受到部分容积效应、灰度不均匀性、伪影、不同软组织间灰度的接近性等因素的影响。针对通常采用的校正技术来说，可以将MR和CT图像中的伪影分类为：(1)需要适当的滤波算法处理的伪影，如噪声伪影、敏感性伪影、存在非清晰边缘的伪影；(2)需要适当图像修复算法的伪影，如运动伪影；(3)需要特定算法的伪影，如部分容积和灰度不均匀性。图像处理领域尽管在已存在很多算法处理上述问题，但是医学图像分割仍然是个复杂和具有挑战性的问题。从医学图像处理过程的角度来看，基于灰度和基于纹理特征技术的分类是常规的分类方式。此外，用机器学习的工具去优化这些图像分割算法是当前较受关注的技术．

CT和MR图像的分割主要涉及3个相关问题：变化的噪声、像素灰度分类的不确定性及灰度的非均衡性。图像中单一组织的灰度水平一般是逐渐变化的，且其概率密度服从特定的分布函数，该组织对应的图像区域包含有限的像素(或体素)且满足部分容积平均，然而该区域中的单个像素(或体素)的灰度不与任何一类一致，往往被看作混合组织类。

CT和MR图像分割常用的一些方法有：基于阈值、基于区域、基于形变模型、基于模糊及基于神经网络。

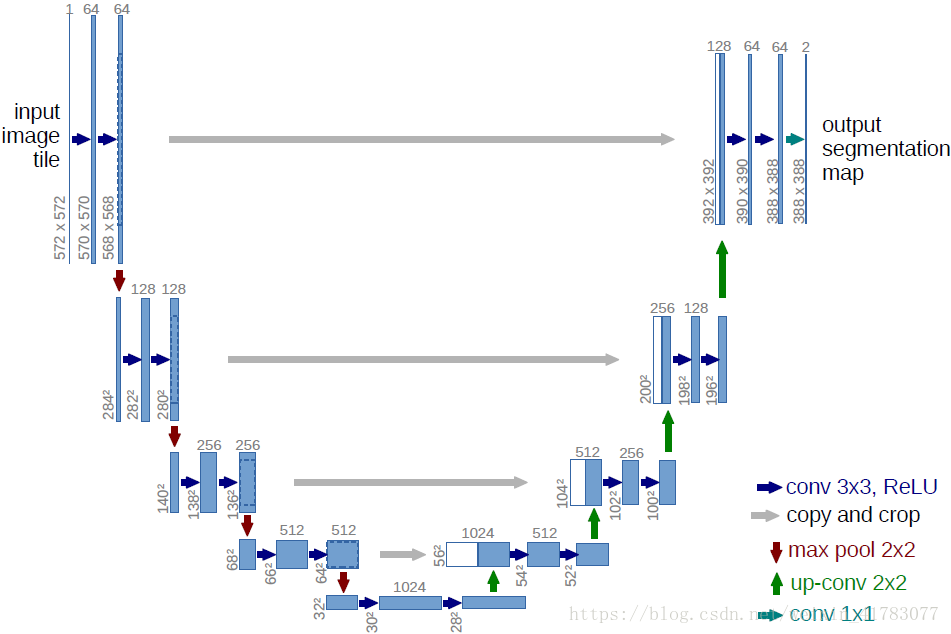
传统的图像处理技术主要包括特征提取和分类器两部分，特征提取算法的设计复杂性与应用局限性、稳定性，以及特定的特征提取算法与特定的分类器相结合的多样性限制着图像处理技术的发展。神经网络的出现，使端到端的图像处理成为可能，当网络的隐藏层发展到多层时便称之为深度学习，但同时需要用逐层初始化技术解决深层次网络训练难度大的问题，之后深度学习便成为时代的主角。而卷积神经网络(Convolutional Neural Network，CNN)便是深度学习与图像处理技术相结合所产生的经典模型，实现该模型的网络实例在特定的图像问题处理上都卓有成效。

传统的基于卷积神经网络的分割方法的做法通常是：为了对一个像素分类，使用该像素周围的一个图像块作为卷积神经网络的输入用于训练和预测。这种方法有几个缺点：一是存储开销很大。例如对每个像素使用的图像块的大小为15x15，则所需的存储空间为原来图像的225倍。二是计算效率低下。相邻的像素块基本上是重复的，针对每个像素块逐个计算卷积，这种计算也有很大程度上的重复。三是像素块大小的限制了感知区域的大小。通常像素块的大小比整幅图像的大小小很多，只能提取一些局部的特征，从而导致分类的性能受到限制。

2015年，OlafRonneberger等人提出了U-net网络结构，U-net是基于FCN的一种语义分割网络，适用于做医学图像的分割。

U-net网络结构与FCN网络结构相似，也是分为下采样阶段和上采样阶段，网络结构中只有卷积层和池化层，没有全连接层，网络中较浅的高分辨率层用来解决像素定位的问题，较深的层用来解决像素分类的问题，从而可以实现图像语义级别的分割。与FCN网络不同的是，U-net的上采样阶段与下采样阶段采用了相同数量层次的卷积操作，且使用skip connection结构将下采样层与上采样层相连，使得下采样层提取到的特征可以直接传递到上采样层，这使得U-net网络的像素定位更加准确，分割精度更高。此外，在训练过程中，U-net只需要一次训练，FCN为达到较精确的FCN-8s结构需要三次训练，故U-net网络的训练效率也高于FCN网络。

U-net网络结构如图2所示, 蓝色箭头代表卷积和激活函数, 灰色箭头代表复制剪切操作, 红色箭头代表下采样, 绿色箭头代表反卷积,conv 1X1代表卷积核为1X1的卷积操作。从图中可以看出，U-net网络没有全连接层，只有卷积和下采样。U-net可以对图像进行端到端的分割，即输入是一幅图像, 输出也是一幅图像。



因此，对于直肠癌淋巴结CT图像的分析我们可以做以下训练：

结节的位置相对于由CT扫描仪定义的以毫米为单位的坐标系。图像数据数由不同长度的大小为512×512的array构成。为了将体素位置转换为世界坐标系，我们需要知道[0,0,0]体素的真实坐标和体素的间距（mm）。

为了找到一个结节的体素坐标，根据其真实的位置，我们使用itk图像对象的GetOrigin（）和GetSpacing（）方法：

itk\_img = sitk.ReadImage(img\_file)

img\_array = sitk.GetArrayFromImage(itk\_img) *# indexes are z,y,x (notice the ordering)*

center = np.array([node\_x,node\_y,node\_z]) *# nodule center*

origin = np.array(itk\_img.GetOrigin()) *# x,y,z Origin in world coordinates (mm)*

spacing = np.array(itk\_img.GetSpacing()) *# spacing of voxels in world coor. (mm)*

v\_center =np.rint((center-origin)/spacing) *# nodule center in voxel space (still x,y,z ordering)*

结节的中心位于img\_array的v\_center[2]切片中。我们将节点信息传递给make\_mask（）函数，并复制生成的掩码给v\_center[2]切片图像以及其上方和下方的切片图像。

i = 0

for i\_z in range(int(v\_center[2])-1,int(v\_center[2])+2):

mask = make\_mask(center,diam,i\_z\*spacing[2]+origin[2],width,height,spacing,origin)

masks[i] = mask

imgs[i] = matrix2int16(img\_array[i\_z])

i+=1

np.save(output\_path+"images\_%d.npy" % (fcount) ,imgs)

np.save(output\_path+"masks\_%d.npy" % (fcount) ,masks)

值得注意的是，在make\_mask（）函数中，掩码坐标必须与array的坐标的顺序相匹配。x和y排序被翻转。请参阅下面代码中的最后一行的下一个代码：

def make\_mask(center,diam,z,width,height,spacing,origin):

for v\_x in v\_xrange:

for v\_y in v\_yrange:

p\_x = spacing[0]\*v\_x + origin[0]

p\_y = spacing[1]\*v\_y + origin[1]

if np.linalg.norm(center-np.array([p\_x,p\_y,z]))<=diam:

mask[int((p\_y-origin[1])/spacing[1]),int((p\_x-origin[0])/spacing[0])] = 1.0

return(mask)

由于结节位置是根据球体定义的，并且结节是不规则形状的，球体边缘附近的切片可能不含结节组织。使用这样的切片可能产生假阳性导致污染训练集。对于这个分割任务，可能存在一个最优的包含的切片的数量。但是为了简单起见，我们只取3张切片，只选择最大的靠近结节中心的切片。

检查以确保结节mask如我们所预期：

import matplotlib.pyplot as plt

imgs = np.load(output\_path+'images\_0.npy')

masks = np.load(output\_path+'masks\_0.npy')

for i in range(len(imgs)):

print "image %d" % i

fig,ax = plt.subplots(2,2,figsize=[8,8])

ax[0,0].imshow(imgs[i],cmap='gray')

ax[0,1].imshow(masks[i],cmap='gray')

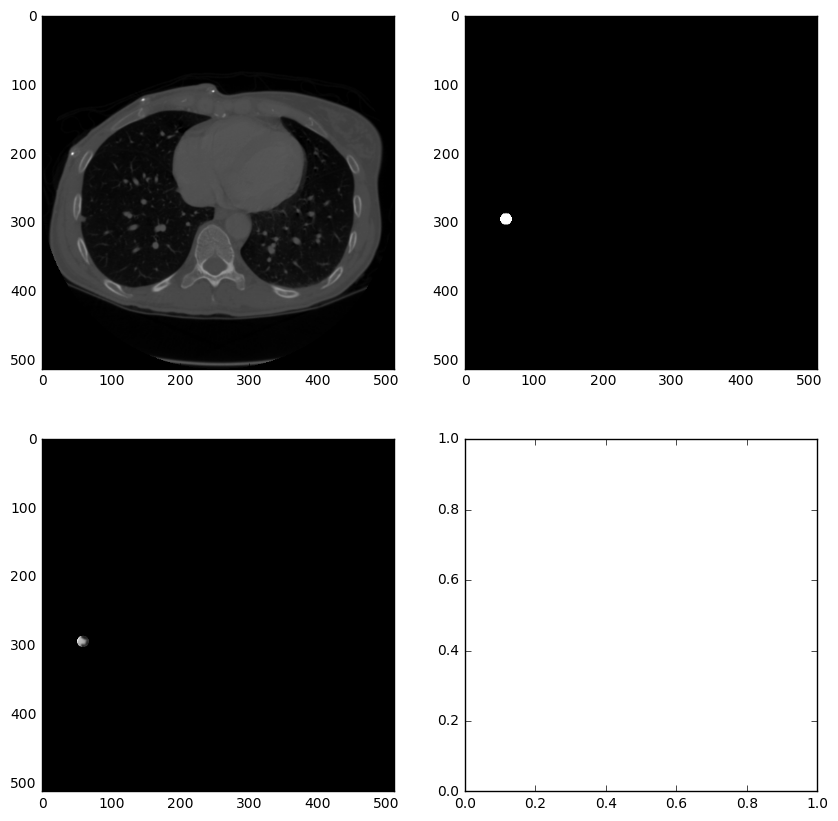
ax[1,0].imshow(imgs[i]\*masks[i],cmap='gray')

plt.show()

raw\_input("hit enter to cont : ")

结节的中心位于img\_array的v\_center[2]切片中。我们将节点信息传递给make\_mask（）函数，并复制生成的掩码给v\_center[2]切片图像以及其上方和下方的切片图像。

左上角的图像是扫描切片。 右上角的图像是节点掩码（mask）。左下角的图像是掩码切片，突出显示了结节。



Dice系数作为分割的成本函数

我们将要使用的网络是教程开端提到的U-net，使用的是keras框架来构建。 损失函数是Dice系数，链接点击打开链接

比较了预测和实际的节点掩膜。

损失函数如下：

smooth = 1.

*# Tensorflow version for the model*

def dice\_coef(y\_true, y\_pred):

y\_true\_f = K.flatten(y\_true)

y\_pred\_f = K.flatten(y\_pred)

intersection = K.sum(y\_true\_f \* y\_pred\_f)

return (2. \* intersection + smooth) / (K.sum(y\_true\_f) + K.sum(y\_pred\_f) + smooth)

def dice\_coef\_loss(y\_true, y\_pred):

return -dice\_coef(y\_true, y\_pred)

该损失函数类似于用于评估该网络最初编写的超声神经分割挑战的度量（再次参见本教程开头的链接）。

载入分割器

smooth = 1.

*# Tensorflow version for the model*

def dice\_coef(y\_true, y\_pred):

y\_true\_f = K.flatten(y\_true)

y\_pred\_f = K.flatten(y\_pred)

intersection = K.sum(y\_true\_f \* y\_pred\_f)

return (2. \* intersection + smooth) / (K.sum(y\_true\_f) + K.sum(y\_pred\_f) + smooth)

def dice\_coef\_loss(y\_true, y\_pred):

return -dice\_coef(y\_true, y\_pred)

函数调用命令

model = get\_unet()

model\_checkpoint = ModelCheckpoint('unet.hdf5', monitor='loss', save\_best\_only=True)

将编译并返回模型，并告诉keras在checkpoints保存模型的权重。 如果您想从以前的训练结果加载最佳权重，或使用本教程回购中包含的权重，请使用下面这行命令加载权重文件

model.load\_weights('unet.hdf5')

从命令行调用LUNA\_train\_unet.py系统将尝试从当前目录加载一个unet.hdf5文件，并在脚本中根据下面这行命令设置的参数进行训练并测试。

model.fit(imgs\_train, imgs\_mask\_train, batch\_size=2, nb\_epoch=20,

verbose=1, shuffle=True,callbacks=[model\_checkpoint])

测试代码如下

num\_test = len(imgs\_test)

imgs\_mask\_test = np.ndarray([num\_test,1,512,512],dtype=np.float32)

for i in range(num\_test):

imgs\_mask\_test[i] = model.predict([imgs\_test[i:i+1]], verbose=0)[0]

np.save('masksTestPredicted.npy', imgs\_mask\_test)

mean = 0.0

for i in range(num\_test):

mean+=dice\_coef\_np(imgs\_mask\_test\_true[i,0], imgs\_mask\_test[i,0])

mean/=num\_test

print("Mean Dice Coeff : ",mean)

model.predict（）函数操作一次可以测试多个样本，但因为我们可以快速重载GPU，我们循环测试各个样本。

最终结果是通过GPU机器，使用TitanX训练得到的。 对于家用GPU计算基准测试，个人设置了GTX970，我们可以在大约一个小时内运行20个epoch，训练集大小为320，批量大小为2。 我们开始约3个小时的训练后，我们就开始得到相对合理的结节掩膜预测结果，损失值大约0.3 。

现在，我们已经准备好开始使用我们上一节中的图像分割来训练分类器以生成特征。  
The Data Science Bowl训练数据集必须被放到分割器进行分割，这可以通过重用用于处理LUNA数据的代码来完成。 但是存在两点区别：  
首先，DSB数据采用dicom格式，可以使用pydicom模块进行读取

import dicom

dc = dicom.read\_file(filename)

img = dc.pixel\_array

其次，为了在扫描中定位节点，扫描的每一层都必须通过分割器运行，因此每个层也必须进行图像处理以对ROI进行掩膜运算。 这是一个非常耗时的过程。

我们首先描述结节图的一些特征，并将它们放入特征向量中，这些特征向量可以用于分类。 我们选取特征并不详尽，它的目的在于说明如何发现可用于识别结节所在区域的度量。

def getRegionMetricRow(fname = "nodules.npy"):

seg = np.load(fname)

nslices = seg.shape[0]

*#metrics*

totalArea = 0.

avgArea = 0.

maxArea = 0.

avgEcc = 0.

avgEquivlentDiameter = 0.

stdEquivlentDiameter = 0.

weightedX = 0.

weightedY = 0.

numNodes = 0.

numNodesperSlice = 0.

*# do not allow any nodes to be larger than 10% of the pixels to eliminate background regions*

maxAllowedArea = 0.10 \* 512 \* 512

areas = []

eqDiameters = []

for slicen in range(nslices):

regions = getRegionFromMap(seg[slicen,0,:,:])

for region in regions:

if region.area > maxAllowedArea:

continue

totalArea += region.area

areas.append(region.area)

avgEcc += region.eccentricity

avgEquivlentDiameter += region.equivalent\_diameter

eqDiameters.append(region.equivalent\_diameter)

weightedX += region.centroid[0]\*region.area

weightedY += region.centroid[1]\*region.area

numNodes += 1

weightedX = weightedX / totalArea

weightedY = weightedY / totalArea

avgArea = totalArea / numNodes

avgEcc = avgEcc / numNodes

avgEquivlentDiameter = avgEquivlentDiameter / numNodes

stdEquivlentDiameter = np.std(eqDiameters)

maxArea = max(areas)

numNodesperSlice = numNodes\*1. / nslices

return np.array([avgArea,maxArea,avgEcc,avgEquivlentDiameter,\

stdEquivlentDiameter, weightedX, weightedY, numNodes, numNodesperSlice])

def getRegionFromMap(slice\_npy):

thr = np.where(slice\_npy > np.mean(slice\_npy),0.,1.0)

label\_image = label(thr)

labels = label\_image.astype(int)

regions = regionprops(labels)

return regions

import pickle

def createFeatureDataset(nodfiles=None):

if nodfiles == None:

noddir = "/home/jmulholland/NLST\_nodules/"

nodfiles = glob(noddir +"\*npy")

*# dict with mapping between truth and*

truthdata = pickle.load(open("/home/sander/truthdict.pkl",'r'))

numfeatures = 9

feature\_array = np.zeros((len(nodfiles),numfeatures))

truth\_metric = np.zeros((len(nodfiles)))

for i,nodfile in enumerate(nodfiles):

patID = nodfile.split("\_")[2]

truth\_metric[i] = truthdata[int(patID)]

feature\_array[i] = getRegionMetricRow(nodfile)

np.save("dataY.npy", truth\_metric)

np.save("dataX.npy", feature\_array)

一旦我们创建了特征向量，我们将它们加载到一些简单的分类模型中，看看实验结果。 我们选择随机森林和XGBoost基于我们的特征工程创建一些模型。

from sklearn import cross\_validation

from sklearn.cross\_validation import StratifiedKFold as KFold

from sklearn.metrics import classification\_report

from sklearn.ensemble import RandomForestClassifier as RF

import xgboost as xgb

X = np.load("dataX.npy")

Y = np.load("dataY.npy")

kf = KFold(Y, n\_folds=3)

y\_pred = Y \* 0

for train, test in kf:

X\_train, X\_test, y\_train, y\_test = X[train,:], X[test,:], Y[train], Y[test]

clf = RF(n\_estimators=100, n\_jobs=3)

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

y\_pred[test] = clf.predict(X\_test)

print classification\_report(Y, y\_pred, target\_names=["No Cancer", "Cancer"])

print("logloss",logloss(Y, y\_pred))

*# All Cancer*

print "Predicting all positive"

y\_pred = np.ones(Y.shape)

print classification\_report(Y, y\_pred, target\_names=["No Cancer", "Cancer"])

print("logloss",logloss(Y, y\_pred))

*# No Cancer*

print "Predicting all negative"

y\_pred = Y\*0

print classification\_report(Y, y\_pred, target\_names=["No Cancer", "Cancer"])

print("logloss",logloss(Y, y\_pred))

*# try XGBoost*

print ("XGBoost")

kf = KFold(Y, n\_folds=3)

y\_pred = Y \* 0

for train, test in kf:

X\_train, X\_test, y\_train, y\_test = X[train,:], X[test,:], Y[train], Y[test]

clf = xgb.XGBClassifier(objective="binary:logistic")

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

y\_pred[test] = clf.predict(X\_test)

print classification\_report(Y, y\_pred, target\_names=["No Cancer", "Cancer"])

print("logloss",logloss(Y, y\_pred))

结果显示了建模的效果

Random Forest precision recall f1-score support

No Cancer 0.77 0.95 0.85 219 Cancer 0.08 0.02 0.03 64

avg / total 0.61 0.73 0.66 283

('logloss', 9.1534198755740697) Predicting all positive precision recall f1-score support

No Cancer 0.00 0.00 0.00 219 Cancer 0.23 1.00 0.37 64

avg / total 0.05 0.23 0.08 283

('logloss', 26.728505803260379) Predicting all negative precision recall f1-score support

No Cancer 0.77 1.00 0.87 219 Cancer 0.00 0.00 0.00 64

avg / total 0.60 0.77 0.68 283

('logloss', 7.8108893613932242) XGBoost precision recall f1-score support

No Cancer 0.77 0.92 0.84 219 Cancer 0.18 0.06 0.09 64

avg / total 0.64 0.72 0.67 283

('logloss', 9.5195722669850706)

我们比较随机森林，XGBoost的结果，以及预测患有癌症和预测不患有癌症的两个模型的结果。

Random Forest

precision recall f1-score support

No Cancer 0.81 0.97 0.88 463

Cancer 0.13 0.02 0.03 107

avg / total 0.68 0.79 0.72 570

('logloss', 7.150150893624649)

XGBoost

precision recall f1-score support

No Cancer 0.83 0.86 0.84 463

Cancer 0.27 0.21 0.24 107

avg / total 0.72 0.74 0.73 570

('logloss', 8.9074570257718957)

Predicting all positive

precision recall f1-score support

No Cancer 0.00 0.00 0.00 463

Cancer 0.19 1.00 0.32 107

avg / total 0.04 0.19 0.06 570

('logloss', 28.055831025357818)

Predicting all negative

precision recall f1-score support

No Cancer 0.81 1.00 0.90 463

Cancer 0.00 0.00 0.00 107

avg / total 0.66 0.81 0.73 570

('logloss', 6.4835948671148085)