摘要

本文定义了用于比较机器学习数据集的集合差异的度量，提出数据集之间的差异是关于组合覆盖率的函数，并举例阐明了这种方法在估计和评测机器学习模型的性能上的效用。识别和测量数据集之间的差异在机器学习领域是一个非常重要的问题，模型的准确性很大程度上取决于训练数据能否充分代表应用程序中遇到的数据。该方法可应用于迁移学习，即预测一个模型是否能在其他数据集上也能有良好的预测结果的问题。

问题描述

在软硬件系统测试中，绝大多数的错误都来源于各组件系统之间的交互。一般来说，充分测试4~6个组件之间的交互就可以检测软件系统中所有能考虑的故障。因为机器学习(ML,Machine Learning)的某些特性比如大规模输入、白盒测试工作以及集成或系统级出现的紧急行为，传统的软件测试方法对嵌入式机器学习系统的测试具有一定的困难。为了解决组件系统集成过程中意外交互所导致的故障，我们需要对机器学习中的软硬件系统集成引进一个新的测试方案及度量。

解决思路

组合覆盖(CC,Combinatorial Testing)法是一种有效减少测试用例个数的测试用例设计方法。组合测试(CT,Combinatorial Testing)使用伪详尽测试来解决大输入域，已经被广泛应用于自动驾驶汽车系统中嵌入ML组件的测试用例生成、测试神经网络的内状态空间、特征选择和可解释的ML。在原有CT的基础之上，我们定义一种新的组合覆盖度量来比较ML数据集。

指标的定义

在特征应用于CT之前，首先将连续值离散化。$t-way$值组合是指将特定的值赋给t个因子或t元组。组合覆盖(CC,Combinatorial Coverage)也称为总$t-way$覆盖。给定一个数据集D是U的子集，定义一个k因素的体系和他们各自的等级，U是所有可能的数据点的集合，U_t是所有可行的$t-way$值组合的集合。如果某些集合不可能，那它会被阻止并从U_t中删除。 给一个数据集D是U的子集，定义D_t是D中出现的$t-way$组合的集合。D数据集的$t-way$覆盖率为

$$CC_t(D)=\frac{D_t}{U_t}$$

$$\begin{cases}\text{我们定义}T_t/S_t\text{是属于}T_t\text{但不属于}S_t\text{的集合组合}\\ \text{定义}SDCC_t\text{（}Set\ Difference\ Combination\ Coverage\text{）为}t\text{集合的集合差异组合覆盖（}SDCC_t\in[0,1]\text{）}\\ \text{则}t-way\text{集合差异组合覆盖}SDCC_t(T/S)=\frac{|T_t/S_t|}{|T_t|}\end{cases}$$

根据集合论会有以下关系：

$$\begin{cases}1)S_t\subset T_t\Rightarrow 0<SDCC_t(T/S)<1,\\ 2)S_t=T_t\Rightarrow SDCC_t(T/S)=0,\\ 3)(S_t\not\subset T_t)\cap(T_t\not\subset S_t)\cap(S_t\cap T_t\neq\emptyset)\Rightarrow 0<SDCC_t(T/S)<1\\ 4)T_t\subset S_t\Rightarrow SDCC_t(T/S)=0\\ 5)S_t\cap T_t=\emptyset\Rightarrow SDCC_t(T/S)=1\end{cases}$$

注意$SDCC_t(T/S)$和$SDCC_t(S/T)$不一定相等，它们的差值也是度量之一。

应用层面

a.故障定位

$t-way$值组合的集差法可以用于故障定位。通常运行测试集会有大量的通过测试和少数的失败测试，而只有失败测试的一小部分值组合会导致失败。我们令

$$P_t = 通过测试中的值组合，\ F_t = 失败测试中的值组合，\ C_t = 故障触发的值组合$$

识别故障触发值组合的第一步是一个基本的消除规则：计算$\frac{F_t}{P_t}$，即不存在于任何通过测试集合中且在失败测试集合中的值组合。因为这对于确定性系统而言,这些值组合必定包含触发故障的值组合C_t，还可以使用基本的集合操作进一步减少故障中可能涉及的值组合。

b.可解释性分类

从某种角度来看，ML中的分类问题于CT中的故障定位问题本质上是相同的。通过计算属于集合C但不属于集合N的$t-way$值组合的集合C_t/N_t来对ML分类进行解释或证明，其含义为这些值组合相比起N集，与C集合的关系更密切。

XAI即可解释性人工智能，是一种可以使AI算法更加透明可靠,特别是在医疗，金融和军事领域能够提供可解释性的方法。XAI方法根据提取信息的时间分为intrinsic和post-hoc。 intrinsic模型是能够在训练过程中自解释的，例如决策树模型。 post-hoc模型则是在训练之后可以用一个简化的模型来解释。 post-hoc解释模型可以用来解释intrinsic模型，但是反过来则不一定。 post-hoc模型对模型的预测更准，但是解释性就不如intrinsic模型。

以前的模型归纳方法现在可以从ML输出出发通过逆向工程获得一个解释或者模型。使用统计方法来识别与类最密切相关的特征。而组合XAI方法通过产生用于解释的特征组合来扩展了这个方法。这种区别是非常重要的，因为密切相关的单一因素在识别值组合中不一定存在。基于规则的专家系统通常被认为很容易解释，但通常不如神经网络等更不透明的方法那么方便。而XAI组合方法提供了一种自然映射到清晰可理解的规则。

c.模型包络

计算机视觉(CV,Computer Vision)包括检测或分类图像中的物品等任务。领域的复杂性取决于所有影响图像生成的变量，导致交互效应的可能性很高。ML模型的操作包络描述了期望它在其中正确执行的环境；部署到外部环境可能会导致意想不到的结果。以一个ML模型学习它所训练的例子，因此要在每个环境下按预期执行，训练数据集必须包含足够的代表性例子。

计算机视觉模型的操作包络的一个维度是通过描述模型训练为数据集中存在的特征值组合覆盖的环境来定义的。这些特征可能直接来自图像数据，但使用元数据有两个好处。①元数据更容易被操作员理解;②当图像数据不存在时，元数据可能是可用的。

当类标签可用时，我们有一种计算值组合的特殊方法。标签中心主义(Label centrism)迫使所有值组合涵盖一个标签；以标签中心主义的值组合包括其标签和t-1个其他特征。这类标签中心主义方法可以描述物体出现的环境。

测试集设计

在设计测试集的过程中，我们将数据集划分为训练S集、验证T集和测试T集。当数据集较大并且采用随机选择时，由于它们来自相同的总体，测试集在训练集中也具有代表性。计算$SDCC_t(S/T)$以及$SDCC_t(T/S)$可以在糟糕的随机抽取中提供保障。用普通的随机算法划分几个随机区域并且维护一个最低SDCC值的区域,这就等同于在模型的操作包络中进行测试。

在实践中，由于数据集不够大，很少有人从头开始训练网络。常见的做法是使用预训练的网络（例如在ImageNet上训练的分类1000类的网络）来重新fine-tuning（也叫微调），或者当做特征提取器。这种方法称为Fne-tuning方法。

训练数据集的代表性声明通常依赖于随机选择或者按对象类型计数。组合覆盖(CC)计算训练集提供了一个测量的上下文模型训练通过值组合的可调参数t。在迁移学习中，在一个环境训练的模型部署到一个新的环境，可能没有再训练或fne调优，这会导致在更大的部署环境下模型无法正常表现。

实验内容

Planesnet是一个标记的数据集，由从行星卫星图像中提取的图像芯片组成。(https://www.kaggle.com/rhammell/planesnet/version/2)在此数据集记录了检测行星中间分辨率遥感图像中飞机的位置，我们可以创建网络模型（这里采用VGG16）来辨别图像中是否飞机有飞机的存在，这是一个很典型的二分类问题。

准备工作

①数据集：将其随机划分为训练S集、验证T集和测试T集，并随机组合。在本实验中，一组数据共20张图片，大约75%去到训练集，25%去到验证集。

②模型准备：使用keras实现VGG16模型，构建模型的基本框架。

实验过程

①我们迭代时对数据进行交叉检验，损失函数使用'categorical_crossentropy'(交叉熵损失函数)，这样可以加快参数的更新，并且贴近组合测试的量标。

②训练时，模型加入交叉检验。交叉检验增加了验证的可靠性以及模型的鲁棒性，并且集合差异组合覆盖SDCC能得到很好的表现，进而可以作为参数去调整学习率。如此一来，收敛过程会加快，基本上第一批数据训练完即可得到很好的效果。

实现代码

```
import os
import json
import glob
import shutil
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mimg
from os import listdir, makedirs, getcwd, remove
from os.path import isfile, join, abspath, exists, isdir, expanduser
from PIL import Image
from pathlib import Path
from keras.models import Sequential, Model
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import ImageDataGenerator,load_img, img_to_array
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input, Flatten
from keras.layers import GlobalMaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.merge import Concatenate
from keras.models import Model
from keras.optimizers import Adam, SGD
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle

%matplotlib inline
#magic关键字，将 matplotlib 设置为以交互方式工作
```

```
import os
print(os.listdir("../input"))
```

```
input_path = Path('../input/planesnet/planesnet/planesnet') #返回数据集地址
planes_path = input_path
```

```
planes = []

all_planes = os.listdir(planes_path) # 作为列表读入
for ac in all_planes:
    planes.append((ac[0],str(planes_path)+"/"+str(ac)))
planes = pd.DataFrame(data=planes, columns=['label','image_path'], index=None) #数据框架，记录标签以及文件编号和路径
planes.sample(5) #打表测试
```

```
print("Total number of planes images in the dataset: ", len(planes))
ac_count = planes['label'].value_counts()
plt.figure(figsize=(12,8))
sns.barplot(x=ac_count.index, y=ac_count.values)
plt.title("Images count for each category", fontsize=16)
plt.xlabel("Label", fontsize=14)
plt.ylabel("Count", fontsize=14)
plt.show() #显示目前数据集中的结果
```

```
random_samples = []
num_0=0
num_1=0
for item in planes.sample(20).iterrows():
    random_samples.append((item[1].label, item[1].image_path)) #随机放入数据集

f, ax = plt.subplots(5,4, figsize=(20,20))
for i,sample in enumerate(random_samples):
    ax[i//4, i%4].imshow(mimg.imread(random_samples[i][1]))
    ax[i//4, i%4].set_title(random_samples[i][0])
    if(random_samples[i][0]=='0'):
        num_0+=1
    else:
        num_1+=1
    ax[i//4, i%4].axis('off')
print(num_0,num_1,(num_1)/(num_0+num_1))
plt.show()    #显示当前组
```

```
f = open('../input/planesnet/planesnet.json')
planesnet = json.load(f)
f.close()
#加载数据，并处理标签
X = np.array(planesnet['data']) / 255.
X = X.reshape([-1,3,20,20]).transpose([0,2,3,1])
Y = np.array(planesnet['labels'])
Y = to_categorical(Y, 2)
X,Y = shuffle(X,Y,random_state=42)
X_train = X[0:25000]
Y_train = Y[0:25000]
X_test = X[25000:]
Y_test = Y[25000:]
```

```
print("Input shape : {0}".format(X.shape))
print("Training shape : {0}".format(X_train.shape))
print("Testing shape : {0}".format(X_test.shape)) #这是目前的训练集、测试集和验证集
```

```
#如果没有目录则创建
cache_dir = expanduser(join('~', '.keras'))
if not exists(cache_dir):
    makedirs(cache_dir)

# make the models sub-directory
models_dir = join(cache_dir, 'models')
if not exists(models_dir):
    makedirs(models_dir)
```

```
!cp ../input/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 ~/.keras/models/
```

```
def get_vgg_16(input_shape=48):
    base_model = VGG16(include_top=False, input_shape=(input_shape,input_shape,3))
    x = Flatten()(base_model.output)
    x = Dense(2, activation='softmax', name='fc2')(x)
    model = Model(inputs=base_model.input, outputs=x)
    return model

def show_train_history(train_history,train,validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train History')
    plt.ylabel(train)
    plt.xlabel('Epoch')
    plt.legend(['train', 'validation'], loc='best')
    plt.show()
```

下面是VGG16的模型主要内容

```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping,
ReduceLROnPlateau

weight_path="{}_weights.best.hdf5".format('seg_model')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
save_best_only=True, mode='min', save_weights_only=True)


reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                                   patience=1, verbose=1, mode='min',
                                   min_delta=0.0001, cooldown=2, min_lr=1e-7)

early = EarlyStopping(monitor="val_loss", mode="min", verbose=2,
                      patience=5) # probably needs to be more patient, but kaggle
time is limited

callbacks_list = [checkpoint, early, reduceLROnPlat]
```

```
model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
```

```
# this applies 32 convolution filters of size 3x3 each.
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(20, 20, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

train_history  = model.fit(X_train, Y_train, batch_size=100, epochs=50,
callbacks=callbacks_list, validation_split=0.27)
score = model.evaluate(X_test, Y_test, batch_size=100)
```

```
show_train_history(train_history,'acc','val_acc')
#测试模型的准确率
score = model.evaluate(x=X_test,y=Y_test,batch_size=200)
score
print('Score Accuracy : {:.2f}%'.format(score[1]*100))
#输出最终的训练结果
```

实验总结

总的来说，VGG16是一个很好的模型，在识别飞机的任务中，即使不加入组合测试优化也可以跑到很高的准确率；在速度上并不优异。通过定义组合测试量标并对损失函数、交叉检验概率以及学习率等参数进行调整，可以有效地让函数达到收敛，再结合模拟退火来加快模型训练速度。训练完成后准确率小幅度下跌，训练速度得到大幅提升。

小结

本文定义了新的组合测试指标用作在机器学习领域为分类输出提供解可解释性并且在ML模型的操作包络中大获成功。并且结合了XAI方法以及模型包络解决了某些CV上的困难。通过实验，我们发现设立改组合测试指标可以达到快速检验的效果，从而进一步融入机器学习模型的损失函数、交叉检验率、学习率等对模型进行调整和优化。