使用CPM构建预测模型

Alex / 2022-10-22 / free_learner@163.com / AlexBrain.cn

更新于2023-09-23，主要是文字排版上的更新，内容基本保持不变。

介绍使用Connectome-based Predictive Modeling（CPM）方法构建预测模型的基本步骤。

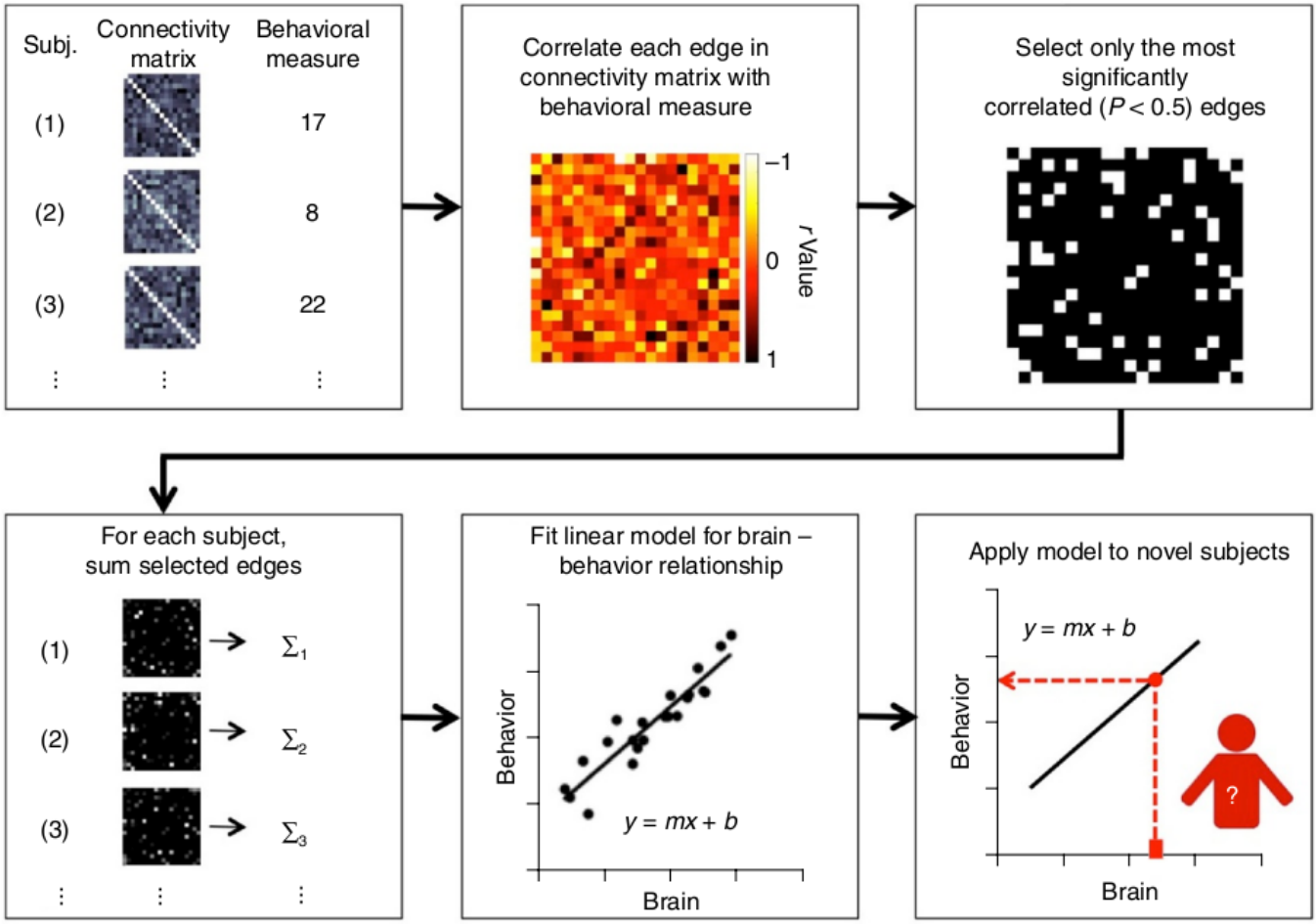
一、背景和原理

如果你想使用脑影像指标（比如，功能连接）预测行为表现（比如，智力）或者临床症状（比如，MoCA），那么可以考虑使用CPM的方法。相比于其他预测模型（比如，SVR），CPM在原理上特别简单。如下图所示，假设每个被试有一个连接矩阵和一个行为分数，CPM：

1. 计算连接矩阵的每个元素（也称为边）与行为分数的相关系数，得到一个相关矩阵；
2. 根据相关系数的P值筛选出显著的边；
3. 将每个被试的连接矩阵中的这些边求和（为方便描述，称为显著边总和）；
4. 将行为分数作为因变量，将显著边总和作为自变量，拟合线性回归模型（或一般线性模型），这样就构建好了预测模型；
5. 对新的被试计算显著边总和，根据预测模型计算出预测的行为分数，预测行为分数和真实行为分数的相关系数用来度量预测模型准确性。在大多数情况下，我们只有一个数据集可以使用，这个时候需要采用交叉验证的方法（cross validation）。以留一交叉验证为例（Leave-One-Out Cross Validation, LOOCV），将一个被试的数据作为测试数据，用剩下的数据构建模型，然后对每一个被试重复这个过程；
6. 为了检验预测模型准确性的显著性（也就是这个准确性是否高于随机水平），需要使用置换检验，在每次置换中将行为分数随机排列，重复上述（1-5）获得模型准确性的零分布，根据模型准确性在零分布中的位置计算P值。

更多信息请参考：

Shen, Xilin et al. “Using connectome-based predictive modeling to predict individual behavior from brain connectivity.” *Nature protocols* vol. 12,3 (2017): 506-518. doi:10.1038/nprot.2016.178



在上述步骤中存在一些可选项：在第1步中，可以使用偏相关来控制混淆因素、用斯皮尔曼相关来应对非高斯分布或者离群值、用稳健回归来避免离群值的影响等；在第2步中，可以设置不同的显著性水平，比如P=0.05或者P=0.01等；在第3步中，可以只考虑正相关的边或负相关的边，也可以把正相关的显著边总和与负相关的显著边总和都作为自变量纳入预测模型中。

二、代码实现

关于CPM的代码实现，如果是Matlab，可以参考：https://www.nitrc.org/projects/bioimagesuite/，这是前面文献里（Shen et al., 2017）给出的代码。如果是R，我查到NetworkToolbox可以实现。由于CPM原理很简单，以下给出我自己写的简单的R代码：

```r
## custom functions used in CPM
## calculate correlation matrix
calcCORR <- function(x, y){
  NSUB <- nrow(x)
  R <- cor(y, x)
  Tvalue <- R/sqrt((1-R^2)/(NSUB-2))
  Pvalue <- 2*pt(abs(Tvalue), NSUB-2, lower.tail=FALSE)
  CORRmat <- rbind(R, Pvalue)
  return(CORRmat)
}
## create feature selection mask
createMask <- function(CORRmat, pthr){
  R <- CORRmat[1,]
  Pvalue <- CORRmat[2,]
  posmask <- (R > 0) * (Pvalue < pthr)
  negmask <- (R < 0) * (Pvalue < pthr)
  mask <- rbind(posmask,negmask)
  return(mask)
}
## summarize features by summing the all features
sumFeature <- function(x, mask){
  posmask <- mask[1,]
  negmask <- mask[2,]
  posSum <- rowSums(x[,posmask == 1, drop=FALSE])
  negSum <- rowSums(x[,negmask == 1, drop=FALSE])
  AllSum <- data.frame(Pos=posSum, Neg=negSum)
  return(AllSum)
}
## K-fold CV CPM
CPM_CV <- function(x, y, pthr, K){
  ## x is a subject by brain-feature dataframe
  ## y is a vector representing behavior data
  ## pthr is the significance level to select features
  ## K means the K-fold CV
  NSUB <- nrow(x)
  y_predict <- numeric(length = NSUB)
  ## randomly split the data into K folds
  rand_idx <- sample(NSUB)
  folds <- cut(c(1:NSUB), breaks=K, labels=FALSE)
  for (fold_idx in c(1:K)){
    sub_idx <- which(folds == fold_idx, arr.ind=TRUE)
    x_train <- x[-rand_idx[sub_idx],]
    y_train <- y[-rand_idx[sub_idx]]
    x_test <- x[rand_idx[sub_idx],,drop=FALSE]
    ## calculate correlation between x and y
    corr_train <- calcCORR(x_train, y_train)
    ## create feature selection mask
```

```
    mask_train <- createMask(corr_train, pthr)
    ## summarise
    sum_train <- sumFeature(x_train, mask_train)
    ## fit linear model
    model_train <- lm(y_train ~ Pos + Neg, data = sum_train)
    ## predict
    sum_test <- sumFeature(x_test, mask_train)
    y_predict[rand_idx[sub_idx]] <- predict(model_train, sum_test)
  }
  ## evaluate model performance
  output <- cor(y, y_predict)
  return(output)
}
```

```
## invoke custom functions defined above and do permutation test
## significance level for feature selection
pthr <- 0.05
## 10-fold CV
K <- 10
## number of permutations
nperm <- 1000
## the correlation between real and predicated data
acc_real <- CPM_CV(brain_dat, behavior_dat, pthr, K)
## permutation procedure to obtain the null distribution
NSUB <- nrow(brain_dat)
acc_null <- numeric(length=nperm)
for (perm_idx in c(1:nperm)){
  acc_null[perm_idx] <- CPM_CV(brain_dat, behavior_dat[sample(NSUB)], pthr, K)
}
acc_all <- c(acc_real, acc_null)
Pvalue <- sum( acc_all >= acc_all[1])/length(acc_all)
output <- c(acc_real, Pvalue)
```