Models template

Nick Borg

September 23, 2016

Contents

Library imports **1**

Load your data. **1**

Clean your data **1**

Evaluating a classifier. **2**

Train test splits... **3**

Stuff for Kelly specifically **5**

This is a generic template for predicting behavioral repsonses from fMRI data. We assume that you import behavioral and brain data from the same .csv file, in the format

```
TR, Trial, Behavioral_1, ...., ROI_TR_1, ROI_TR_2, ... ROI2_TR_1, ...
```

Library imports

```
library(dplyr)
library(lme4)
library(e1071)
library(lmtest)
library(lubridate)
library(knitr)
```

Load your data.

```
# data_path <- '/the/path/to/csv/'
```

Clean your data

I recommend this extra preprocessing step if you end up using trial based data: You need to look at which columns contain brain data, here they are columns 19:113. the colnames function is useful for that. (Remember that R is 1 indexed.)

```
# #Remove Outliers with > 4 sd activations--non physiological (find reference? - cite Nik Sawe paper)
# max_act  =apply(as.matrix(df[19:113]), 1, function(x) { max(abs(x),na.rm=TRUE) } ) # this finds the l
# hist(max_act)
# df <- df[(max_act<4) & max_act > 0,]
# max_act  =apply(as.matrix(df[19:113]), 1, function(x) { max(abs(x),na.rm=TRUE) } )
# hist(max_act)
```

Evaluating a classifier.

In order to test a model on our data set, it would be nice if we could pass a single formula and get accuracy information from a classifier trained using that formula.

First of all, we assume that we have a single factor variable we want to predict that takes two values, so that as.numeric on your y yields (1, 2)

As an example, let's say that I want to know what predicts stock movement in Mirre's stock task. We'll want to write a function call that looks like: (kable is a function that prints nice tables from a dataframe)

```
formula <- as.formula('Result ~ Choice + (nacc8mmb_TR_4 + mpfcb_TR_4 + demartino_dmpfc8mmb_TR_4 + desai
ans <- evalClassifier(df, formula, type='glm', split.method = 'average')
kable(ans)
```

It would be nice if we could just run one function and get back a performance metric. The definition of evalClassifier is below. The different arguments are : * type (can be 'glm', 'svm', 'naive_bayes'), * split.method (how do you do a train/test split), * sep.subjects (do you train separate classifier within each subject)

Really, evalClassifier is just a high level function that does a tiny bit of data cleaning and calls a more basic function, .evalClassifier, which does the meat of the work. It lets us run more than one subject at a time, but otherwise it's sending the work off to .evalClassifier and then putting the results in a dataframe which prints things neatly.

```
evalClassifier <- function(df, formula, type='glm', split.method='8020', sep.subjects=FALSE, print.model
  # first get rid of any rows in which one of our variables is NA)
  formula.vars <- unlist(strsplit(gsub("[^[:alnum:]|'_' ]", "", as.character(formula)), c(' ')))
  for(v in formula.vars){
    if(v %in% colnames(df)) {
      df <- df[!is.na(df[v]), ]
    }
  }
  # make our results data frame.
  d.res <- data.frame()
  if(sep.subjects){
    #eval_model on each subjects
    for(subj in unique(df$Subject)){
      accuracies <- .evalClassifier(filter(df, Subject==subj), formula, type, split.method)
      d.res<- rbind(d.res, data.frame(Subject=subj, accuracies))
    }
    d.res <- d.res[order(-d.res$test.accuracy),] # order the subjects by their classifier accuracy
    # add the average of all the subjects
    d.res<- rbind(d.res, data.frame(Subject='Average', t(colMeans(d.res[,-1]))))
  } else {
    if(split.method=='loso'){
```

```
      #eval_model on each subjects
      for(subj in unique(df$Subject)){
        accuracies <- .evalClassifier(df, formula, type, split.method, test.sub=subj, print.model = pri
        d.res<- rbind(d.res, data.frame(Subject=subj, accuracies))
      }
      d.res <- d.res[order(-d.res$test.accuracy),] # order the subjects by their classifier accuracy
      # add the average of all the subjects
      d.res<- rbind(d.res, data.frame(Subject='Average', t(colMeans(d.res[,-1]))))
    } else{
      accuracies <- .evalClassifier(df, formula, type, split.method, print.model=TRUE)
      d.res <- rbind(d.res, data.frame(data='All', accuracies))
    }
  }
  d.res
}
```

.evalClassifier is defined like so:

```
.evalClassifier <- function(df, formula, type='glm', split.method='loso', test.sub = '',  print.model=F

  y <- all.vars(formula)[1] # get the y variable name to do train test splits
  train.test <<- trainTestSplit(df, split.method, y=y,test.sub=test.sub)
  m <- fitModel(df, train.test$train, formula, type, print.model)
  train.accuracy <- modelAccuracy(m, y, train.test$train, type)
  test.accuracy  <- modelAccuracy(m, y, train.test$test, type)
  list('train.accuracy'=train.accuracy, 'test.accuracy'=test.accuracy)
}
```

Train test splits...

How your train test splits are defined will depend on your project, so you'll probably need to add cases to the following function.Right now there's just a basic function to call for

```
trainTestSplit <- function(df, split.method='8020', y='result', test.sub="", downsample=FALSE){
  set.seed(0)
  # Assume that the two responses are levels 1, 2 of a factor
  ind1 <- which(as.numeric(df[,y])==2)
  ind0 <- which(as.numeric(df[,y])==1)

  # downsample results. to upsample, use max and replace = TRUE
  # uncomment section below for one_subj
  if(downsample){
    sampsize <- min(length(ind1), length(ind0))
    sampind1 <- sample(ind1, sampsize, replace = FALSE)
    sampind0 <- sample(ind0, sampsize, replace = FALSE)
    sampind <- sample(c(sampind1,sampind0)) # the outside call to sample should randomize
  } else{
    sampind <- sample(1:nrow(df))
  }

  balanced.df <- df[sampind,]
```

```
  #Change response levels to 0,1 from 1, 2
  balanced.df[,y] <- factor(as.numeric(balanced.df[,y])-1, levels=c(0,1))

  #return last day
  if(split.method=='8020'){
    nrows <- nrow(balanced.df)
    train.ind <- sample(1:nrows, size=floor(.8 * nrows))
    test.ind <- c(1:nrows)[!c(1:nrows) %in% train.ind]
    d.train = balanced.df[train.ind,]
    d.test  =balanced.df[test.ind,]
    l <- list("train" = d.train, "test" = d.test)
  } else if(split.method=='loso'){
    train.ind <- balanced.df$Subject!= test.sub
    test.ind <- balanced.df$Subject==test.sub
    d.train <<- balanced.df[train.ind,]
    d.test  <<- balanced.df[test.ind,]
    l <- list("train" = d.train, "test" = d.test)
  } else {
    print('trainTestSplit called with split.method != 8020 or loso')
  }
  l
}
```

Lastly the fit model and predict accuracy functions are below:

```
modelAccuracy <- function(m, y, d.test, type='glm', predictions=FALSE){

  if(type=='glm'){
    pred = predict(m, newdata = d.test, type = 'response', allow.new.levels = TRUE)
    pred.bin = factor(ifelse(pred >= 0.5,1,0), levels = c(0,1))
  }

  if(type=='svm'){
    pred = predict(m, d.test)
    pred.bin <- factor(as.numeric(pred), levels=c(1,2), labels=c(0,1))
  }

  if(type=='naivebayes'){
    pred = predict(m, d.test, type='raw')[,2]
    pred.bin = factor(ifelse(pred >= 0.5,1,0), levels = c(0,1))
  }

  class.tab = table(real=d.test[,y], model=pred.bin)
  #print(class.tab)
  acc <- (class.tab[1,1])/sum(class.tab)
  try(acc <- acc +  class.tab[2,2]/sum(class.tab))

  acc
}

fitModel <- function(df, d.train, formula, type='glm', print.model=TRUE){
  # glimpse(d.train)
  needs.multilevel <- '|' %in% unlist(strsplit(gsub("[^[:alnum:]|'_' ]", "", as.character(formula)), c(
```

```
  if(type=='glm'){
    if(needs.multilevel){
      null_formula <- as.formula(paste(as.character(formula)[2], '~', '(1 | Subject)'))
      model <- glmer(formula, data = df, family = binomial(link = logit), control=glmerControl(optimize
      m2 <- glmer(null_formula, data = df, family = binomial(link = logit), control=glmerControl(optimi

      if(print.model){
        print(summary(model), correlation=T)
        if(needs.multilevel){
          print(rsquared.glmm(model))
        }
        print(lrtest(model, m2))
        print(anova(model))
        #print(confint(model))
      }
      model <- glmer(formula, data = d.train, family = binomial(link = logit), control=glmerControl(opt

    } else{
      null_formula <- as.formula(paste(as.character(formula)[2], '~', '1'))
      model <- glm(formula, data=df, family="binomial")
      m2 <- glm(null_formula, data=df, family="binomial")

      if(print.model){
        print(summary(model))
        #print(PseudoR2(model))
        print(lrtest(model, m2))
        print(anova(model))
      }
      #print(summary(d.train))
      model <- glm(formula, data=d.train, family="binomial")
    }


  }
  if(type=='svm'){
     model <- svm(formula, data=d.train)
  }
  if(type=='naivebayes'){
    if(needs.multilevel){
      print('naiveBayes cannot handle interaction terms')
    } else{
      model <-  naiveBayes(formula, data=d.train)
    }
  }
  model
}
```

Stuff for Kelly specifically

```
df <- read.csv("~/Desktop/cue_relapse_data.csv")
glimpse(df)
```

```
## Observations: 20
## Variables: 46
## $ subjects         (fctr) ag151024, si151120, tf151127, wr151127, ja15...
## $ age              (int) 34, 54, 53, 58, 32, 53, 26, 33, 49, 30, 28, 5...
## $ smoke            (dbl) NaN, NaN, NaN, NaN, NaN, 1, 0, 1, 1, 0, 1, 1,...
## $ pa_drugs         (dbl) -1.97400643, -0.11785113, -0.75621142, -0.088...
## $ pref_drugs       (dbl) -2.8888889, -1.1111111, -3.0000000, 0.6666667...
## $ nacc_alcohol_TR3 (dbl) -0.0353788600, -0.0265012800, 0.0625966300, 0...
## $ nacc_alcohol_TR4 (dbl) 0.022002340, 0.025237430, 0.047870950, -0.109...
## $ nacc_alcohol_TR5 (dbl) 0.117623100, 0.028540590, 0.071004580, 0.0237...
## $ nacc_alcohol_TR6 (dbl) 0.175055600, 0.038801550, -0.013699150, 0.144...
## $ nacc_alcohol_TR7 (dbl) 0.103936300, -0.048429630, 0.108606000, 0.196...
## $ nacc_drugs_TR3   (dbl) 0.027899910, 0.004846273, 0.212291000, 0.0224...
## $ nacc_drugs_TR4   (dbl) -0.11162570, 0.16857990, -0.09103268, 0.01342...
## $ nacc_drugs_TR5   (dbl) -0.024931880, 0.234433900, -0.066006870, -0.1...
## $ nacc_drugs_TR6   (dbl) 0.127327800, 0.164772500, -0.135243700, -0.10...
## $ nacc_drugs_TR7   (dbl) 0.06677589, 0.05952367, -0.02081618, 0.187937...
## $ nacc_food_TR3    (dbl) 0.056292880, -0.113395200, 0.143180100, -0.06...
## $ nacc_food_TR4    (dbl) -0.0486885600, -0.1359679000, -0.0580079000, ...
## $ nacc_food_TR5    (dbl) -0.0826579300, -0.0439078500, 0.0108328100, -...
## $ nacc_food_TR6    (dbl) 0.03577925, 0.03935857, -0.23631250, -0.10878...
## $ nacc_food_TR7    (dbl) 0.16986320, 0.00115464, 0.32648360, 0.0802244...
## $ nacc_neutral_TR3 (dbl) -0.128708400, -0.076840610, 0.042672870, 0.09...
## $ nacc_neutral_TR4 (dbl) -0.16483450, -0.09502515, 0.08344576, 0.02355...
## $ nacc_neutral_TR5 (dbl) -0.175577300, 0.063001400, -0.166032300, -0.2...
## $ nacc_neutral_TR6 (dbl) -0.179871300, 0.004997686, -0.226226000, -0.1...
## $ nacc_neutral_TR7 (dbl) -0.121671800, -0.058948050, -0.178806200, -0....
## $ mpfc_alcohol_TR3 (dbl) -0.054790950, 0.004460754, -0.004834496, 0.12...
## $ mpfc_alcohol_TR4 (dbl) 0.1424564000, -0.0001813521, 0.0603463600, 0....
## $ mpfc_alcohol_TR5 (dbl) 0.169669600, 0.004832847, -0.213841300, 0.183...
## $ mpfc_alcohol_TR6 (dbl) 0.176519300, -0.154635600, -0.294992600, 0.27...
## $ mpfc_alcohol_TR7 (dbl) -0.036894960, -0.006763289, -0.036225850, 0.0...
## $ mpfc_drugs_TR3   (dbl) -0.0272860600, 0.3635549000, 0.1632500000, 0....
## $ mpfc_drugs_TR4   (dbl) -0.08196586, 0.06701886, 0.20888570, -0.28265...
## $ mpfc_drugs_TR5   (dbl) 0.104480700, 0.200758800, -0.246268900, 0.166...
## $ mpfc_drugs_TR6   (dbl) 0.090766380, 0.119605000, -0.131730500, 0.321...
## $ mpfc_drugs_TR7   (dbl) -0.093706150, 0.015718160, 0.093034220, 0.327...
## $ mpfc_food_TR3    (dbl) 0.27409260, -0.05622110, 0.03876768, -0.24119...
## $ mpfc_food_TR4    (dbl) 0.10719830, 0.08550984, 0.26011600, 0.0383412...
## $ mpfc_food_TR5    (dbl) 0.145928000, -0.150911300, 0.004680386, -0.15...
## $ mpfc_food_TR6    (dbl) 0.20628170, -0.14154610, -0.24856490, -0.0787...
## $ mpfc_food_TR7    (dbl) -0.084343050, -0.090938740, 0.288201000, -0.0...
## $ mpfc_neutral_TR3 (dbl) 0.22088110, -0.02599651, 0.17353030, 0.036459...
## $ mpfc_neutral_TR4 (dbl) -0.14295960, -0.24020720, 0.17777800, -0.0383...
## $ mpfc_neutral_TR5 (dbl) -0.320026400, -0.033608000, -0.336698200, 0.0...
## $ mpfc_neutral_TR6 (dbl) -0.191710200, -0.054294420, -0.421371300, 0.0...
## $ mpfc_neutral_TR7 (dbl) -0.303880100, -0.008238650, -0.202308400, 0.1...
## $ relapse          (int) 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, ...
```

```
df$relapse = as.factor(df$relapse)
df$Subject <- df$subjects
```

```
formula <- as.formula("relapse ~ nacc_drugs_TR5  + mpfc_drugs_TR6 + pref_drugs ")
res <- evalClassifier(df, formula, type='glm', split.method='loso')
kable(res)
```

	Subject	train.accuracy	test.accuracy
2	si151120	0.8421053	1.00
4	wr151127	0.7894737	1.00
5	ja151218	0.7894737	1.00
6	wh160130	0.8421053	1.00
8	as160317	0.8421053	1.00
9	rv160413	0.8421053	1.00
10	ja160416	0.8421053	1.00
11	cm160510	0.7894737	1.00
12	tj160529	0.8421053	1.00
13	at160601	0.8421053	1.00
14	zm160627	0.8421053	1.00
15	jf160703	0.8421053	1.00
17	rs160730	0.8421053	1.00
18	nc160905	0.8421053	1.00
20	lm160914	0.8421053	1.00
1	ag151024	0.8421053	0.00
3	tf151127	0.8421053	0.00
7	nb160221	0.8947368	0.00
16	cg160715	0.8421053	0.00
19	gm160909	0.8947368	0.00
21	Average	0.8394737	0.75

```
formula <- as.formula("relapse ~ nacc_drugs_TR5  + mpfc_drugs_TR6 + pref_drugs ")
res <- evalClassifier(df, formula, type='svm', split.method='loso')
kable(res)
```

	Subject	train.accuracy	test.accuracy
1	ag151024	0.8421053	1.00
5	ja151218	0.8421053	1.00
6	wh160130	0.8421053	1.00
8	as160317	0.8421053	1.00
9	rv160413	0.8421053	1.00
11	cm160510	0.8421053	1.00
12	tj160529	0.8421053	1.00
13	at160601	0.8421053	1.00
14	zm160627	0.8421053	1.00
15	jf160703	0.8421053	1.00
17	rs160730	0.8421053	1.00
18	nc160905	0.8421053	1.00
20	lm160914	0.8421053	1.00
2	si151120	0.8421053	0.00
3	tf151127	0.8947368	0.00
4	wr151127	0.8421053	0.00
7	nb160221	0.8947368	0.00
10	ja160416	0.8947368	0.00

	Subject	train.accuracy	test.accuracy
16	cg160715	0.8421053	0.00
19	gm160909	0.8947368	0.00
21	Average	0.8526316	0.65

```
formula <- as.formula("relapse ~ nacc_drugs_TR5  + mpfc_drugs_TR6 + pref_drugs ")
res <- evalClassifier(df, formula, type='naivebayes', split.method='loso', print.model=TRUE)
kable(res)
```

	Subject	train.accuracy	test.accuracy
1	ag151024	0.7894737	1.0
2	si151120	0.7894737	1.0
4	wr151127	0.8421053	1.0
5	ja151218	0.8421053	1.0
6	wh160130	0.7894737	1.0
8	as160317	0.7894737	1.0
9	rv160413	0.7894737	1.0
11	cm160510	0.8421053	1.0
12	tj160529	0.7894737	1.0
13	at160601	0.8421053	1.0
14	zm160627	0.7894737	1.0
15	jf160703	0.7894737	1.0
17	rs160730	0.7894737	1.0
20	lm160914	0.7894737	1.0
3	tf151127	0.8421053	0.0
7	nb160221	0.8421053	0.0
10	ja160416	0.8947368	0.0
16	cg160715	0.8421053	0.0
18	nc160905	0.7894737	0.0
19	gm160909	0.8947368	0.0
21	Average	0.8184211	0.7