

# Detection and prediction using DNA methylation, practical

Paul Yousefi, PhD, MPH

## Getting started

---

To start, everyone please make sure that:

- 🔥 your directory is correctly set
- 🔥 you only have the `dataset.rda` file loaded in active memory

If not, this should get you up to speed:

```
## for example:  
setwd("0:/teaching/advancedEpi_shortCourse")  
#rm(list=ls())
```

```
load("dataset.rda")
```

```
str(samples)
table(samples$smoking)
```

current	former	never
22	263	179

- ✖ Our current smoking variable has 3 categories, but we want to work with a binary outcome
- ✖ Let's do that by collapsing the **former** and **current** subjects into a single category of ever smokers

---

Let's add an never/ever smoking variable to our samples data frame:

```
samples$never.smoke <- sign(samples$smoking=="never")  
  
## be sure it matches the counts we'd expected  
table(samples$never.smoke)
```

```
>  
>    0    1  
> 285 179
```

✶ When I talk about predicting smoking going from now on I'll be referring to this variable

---

## Using a single CpG site as a predictor of smoking

---

## Single CpG predictor

✎ Let's start by seeing how well just the single top hit CpG does at predicting smoking

▶ cg05575921 in the *AHRR* gene

✎ Start by adding it as a new variable to our `samples` data frame

```
samples$cg05575921 <- meth["cg05575921", ]
```

---

## Single CpG predictor

✶ We'll use the pROC package to see how well cg05575921 does at predicting never . smoke

```
## load the pROC package
require("pROC")

## use the formula-based syntax of the package
roc.out <- roc(never.smoke ~ cg05575921, data = samples)
```



---

## Single CpG predictor

```
## load the pROC package  
require("pROC")
```

```
## use the formula-based syntax of the package  
roc.out <- roc(never.smoke ~ cg05575921, data = samples)
```

```
roc.out # man!!
```

```
>
```

```
> Call:
```

```
> roc.formula(formula = never.smoke ~ cg05575921, data = samples)
```

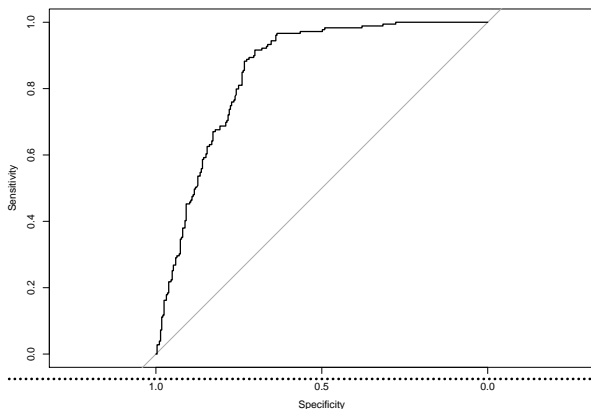
```
>
```

```
> Data: cg05575921 in 285 controls (never.smoke 0) < 179 cases
```

```
> Area under the curve: 0.851
```

## Single CpG predictor

```
plot.roc(roc.out)
```



Using published coefficients to make a prediction score

---

## Methylation score from published coefficients

- ✎ Let's compare how well our single site, cg05575921, performs at predicting smoking to a smoking score derived from the published coefficients of the largest blood EWAS meta-analysis to date by Joehanes et al. 2016
- ✎ The coefficients were reported in their supplemental material
  - ▶ I've conveniently read them into R for you so they're ready to use (you're welcome!)
  - ▶ Try loading them into R

```
load("joehanes2016_st2_bonf.rda")
```

## Methylation score from published coefficients

- ✦ The `joehanes` object has summary information on the 2617 CpGs that were significant at a Bonferroni p-value threshold in the original meta-analysis and that were available in our methylation dataset

```
str(joehanes)
```

```
> 'data.frame': 2617 obs. of 14 variables:
> $ probe.id : chr "cg16145216" "cg19406367" "cg
> $ infinium.design.type: chr "I" "II" "II" "II" ...
> $ chromosome : chr "1" "1" "1" "11" ...
> $ location..hg19. : chr "42,385,662" "66,999,929" "2,
> $ strand : chr "R" "R" "F" "R" ...
> $ gene.symbol : chr "HIVEP3" "SGIP1" "SKI" "CUGBP
> $ effect : num 0.0298 0.0175 -0.0122 -0.0124
> $ std_error : num 0.002 0.0013 0.0009 0.0009 0
```

---

## Methylation score from published coefficients

- ✦ Let's restrict our big methylation data object, `meth`, to just the CpGs that are in the `joehanes` list
- ✦ This keeps the CpGs that we expect to be most related to smoking behavior while reducing the size of the data were working with

```
meth <- meth[joehanes$probe.id, ]
```

---

## Methylation score from published coefficients

Transpose the methylation matrix so that the CpGs are the columns of the object, like they would be if they were normal variables

```
X <- t(meth)
```

- ✦ This puts our methylation covariates in the matrix-notation that is consistent with most mathematical notation of linear models

---

## Methylation score from published coefficients

That is, while you may be used to thinking of linear regression in this format:

$$\star E(Y|X) = \beta_0 + \beta_1 X_1 \dots \beta_j X_j$$

You can also write an equation for the  $\beta$  coefficients in matrix notation as below:

$$\star \hat{\beta} = (X^T X)^{-1} X^T y$$

This is a handy way of describing our coefficients when we're thinking about how to use them to generate fitted/predicted values:

$$\star \hat{y} = X \hat{\beta}$$



---

## Methylation score from published coefficients

Let's also go ahead and make a nice named vector of the `joehanes` coefficients that we can apply to our observed methylation and generate our smoking score ( $\hat{y}$ ):

```
coefs <- joehanes$effect  
names(coefs) <- joehanes$probe.id
```

---

## QUESTION TIME!

1. Apply the `joeHanes` coefficients to our observed methylation values
2. Add the score to your existing `samples` data frame
3. Calculate the AUC for this predictor
4. Draw the ROC curve
5. Does the `joeHanes` score predict never/ever smoking better than just `cg05575921` alone?

---

*#1. Apply the joehanes coefficients to our observed  
# methylation values*

```
y.hat <- X %*% coefs
```

*#2. Add the score to your existing `samples`  
# data frame*

```
samples$y.hat <- as.vector(y.hat)
```

*#3. Calculate the AUC for this predictor*

```
roc.out.again <- roc(never.smoke ~ y.hat, data = samples)  
roc.out.again$auc
```

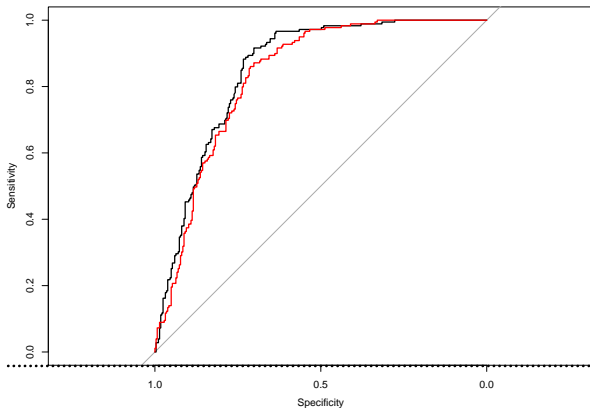
```
> Area under the curve: 0.832
```

---

---

#### #4. Draw the ROC curve

```
plot.roc(roc.out)  
lines.roc(roc.out.again, col="red")
```



---

```
#5. Does the joehanes score predict never/ever  
#   smoking better than just cg05575921 alone?  
### No! Not according to the AUCs:  
roc.out$auc
```

```
> Area under the curve: 0.851
```

```
roc.out.again$auc
```

```
> Area under the curve: 0.832
```

---

Train and test a new predictor with cross validation

---

## Making folds

Let's chop our dataset up into several subsets so that we can train and test a prediction model on different observations in our dataset

- ✂ We could do this using the base function, `sample()`, but if we want to make more than a couple folds this gets messy

We'll use the `caret` package to automatically partition our data into 10-folds:

```
require(caret)

set.seed(20180314)
Y <- samples$never.smoke
flds <- createFolds(Y, k = 10, list = TRUE,
  .....returnTrain = FALSE).....
```

---

## Making folds

```
str(flds)
```

```
> List of 10
> $ Fold01: int [1:46] 3 13 20 22 27 36 96 98 111 119 ...
> $ Fold02: int [1:47] 5 16 17 45 63 77 83 87 92 93 ...
> $ Fold03: int [1:46] 2 11 15 18 26 32 42 46 65 72 ...
> $ Fold04: int [1:46] 10 34 47 53 57 61 81 82 85 94 ...
> $ Fold05: int [1:47] 7 31 40 50 51 54 64 69 70 90 ...
> $ Fold06: int [1:46] 6 8 9 21 24 33 48 113 121 123 ...
> $ Fold07: int [1:47] 4 37 39 41 44 67 80 84 89 107 ...
> $ Fold08: int [1:46] 12 23 25 29 35 49 52 56 59 62 ...
> $ Fold09: int [1:46] 14 19 28 68 71 79 88 101 106 117 ...
> $ Fold10: int [1:47] 1 30 38 43 55 58 60 73 75 78
```



- ✿ Use the first fold we generated to be our testing dataset
- ✿ The rest we'll use for training

```
test <- flds$Fold01  
train <- (-test)
```

---

# Training

- ✦ Let's train our own predictor using a lasso model, like we learned about in the lecture earlier
  - ▶ E.g. the Liu et al. 2016 DNAm alcohol score used a lasso model

---

## Training

Load the `glmnet` package which has the lasso model function and try fitting on our training data

```
require(glmnet)
set.seed(420)
fit.lasso <- cv.glmnet(X[train,], Y[train],
  family='binomial', alpha=1, standardize=TRUE,
  type.measure='auc', nfolds = 10)
```

🔥 Remember: lasso picks the shrinkage factor,  $\lambda$ , by internal cross-validation

---

---

## Lasso coefficients

```
coef.lasso <- as.matrix(coef(fit.lasso, s = "lambda.min"))
coef.lasso <- data.frame(betas=coef.lasso[,1],
                        stringsAsFactors = F)
coef.lasso <- subset(coef.lasso, betas!=0)

str(coef.lasso)
```

```
> 'data.frame': 19 obs. of 1 variable:
> $ betas: num -11.964 5.546 1.306 0.378 3.963 ...
```

---

## Lasso coefficients

```
coef.lasso
```

```
>                                betas
> (Intercept) -11.96407920
> cg05951221   5.54642609
> cg05575921   1.30629345
> cg10919522   0.37799224
> cg21566642   3.96254550
> cg25189904   0.94875906
> cg03636183   0.29443121
> cg19859270   7.46925604
> cg06126421   3.62530479
> cg00138101  -3.55277076
> cg24154132  -4.65644918
> cg18956562  -2.14032920
```

---

## Predicting in new data

We can see how well the model we just fit performs in new data by predicting on the observations from the testing set:

```
pred.lasso <- as.vector(predict(fit.lasso,  
  newx = X[test,], type = "response", s = "lambda.min"))
```

---

## ROC

```
roc.out.1 <- roc(Y[test], pred.lasso)
roc.out.1$auc
```

> Area under the curve: 0.8481

✶ Seems like our performance is quite similar to the other methods we've tried so far

---

## QUESTION TIME!

1. Use another fold of our data to train at least **two** new prediction models for never/ever smoking
    - ▶ Any flavor of logistic regression (lm, glm, poisson, interactions, quadratics, any combination of CpGs)
    - ▶ Forward/backward stepwise regression
    - ▶ Mean value of Y
    - ▶ K-nearest neighbors (`class::knn`)
    - ▶ Bayes regression (`arm::bayesglm`)
    - ▶ Or any other!
  2. For each method, predict the fitted value  $\hat{y}$  in a separate fold
  3. Calculate the AUCs for each
  4. Plot their performance on a ROC curve
-



```
X <- as.data.frame(X)
# new folds
test <- flds$Fold02
train <- (-test)

## Model 1: glm using a random subset of cpgs
set.seed(86)
cpgs <- sample(colnames(X), 20) # sample 20 cpgs

fit.glm <- glm(Y[train] ~. ,
               data=X[train,cpgs], family='binomial')

pred.glm <- predict(fit.glm,
                   newx = X[test,], type = "response")
```

```
## Model 2: stepwise glm choosing from 20 random cpgs  
fit.step <- step(fit.glm, direction = "backward",  
  trace = 1, k = 2)
```

```
pred.step <- predict(fit.step, newdata = X[test,],  
  type = "response")
```

```
## Model 2: stepwise glm choosing from 20 random cpgs  
fit.step <- step(fit.glm, direction = "backward",  
  trace = 1, k = 2)
```

```
> Start: AIC=492.06
```

```
> Y[train] ~ cg00315394 + cg07229212 + cg04017131 + cg2605595
```

```
> .....cg00756943 + cg16554099 + cg10310310 + cg03222009 + cg2
```

```
> cg21121843 + cg03935116 + cg02363630 + cg04761746 + cg1
```

```
> cg19142553 + cg08202836 + cg17823346 + cg04641860 + cg0
```

```
roc.step <- pROC::roc(Y[test], pred.step)
roc.step$auc
```

```
> Area under the curve: 0.6868
```

```
## Model 3: knn
fit.knn <- class::knn(train = X[train,], test = X[test,],
  k = 10, cl = Y[train], prob = TRUE)
pred.knn <- (as.numeric(fit.knn) - 1) * attr(fit.knn, "prob")
  (1 - (as.numeric(fit.knn) - 1)) * (1 - attr(fit.knn,
    "prob"))

roc.knn <- pROC::roc(Y[test], pred.knn)
roc.knn$auc
```

> Area under the curve: 0.8516

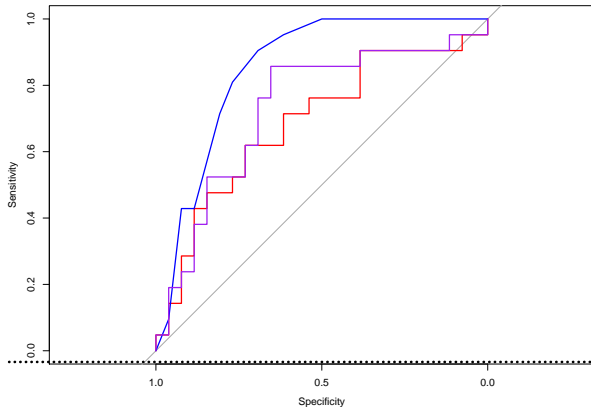
```
## Model 3: bayes regression
fit.bayes <- arm::bayesglm(Y[train] ~., data = X[train, cpgs])
pred.bayes <- predict(fit.bayes, newdata = X[test,],
                      type = "response")

roc.bayes <- pROC::roc(Y[test], pred.bayes)
roc.bayes$auc
```

> Area under the curve: 0.7308

---

```
plot.roc(roc.step, col="red")  
lines.roc(roc.knn, col="blue")  
lines.roc(roc.bayes, col="purple")
```



---

## Next steps

- ✦ The SuperLearner package can automate the cross validation using a library of candidate prediction models
  - ▶ <https://cran.r-project.org/web/packages/SuperLearner/index.html>

```
##
## Call:
## CV.SuperLearner(Y = Y_train, X = X_train, V = 3, family = binomial(),
##   SL.library = list("SL.mean", "SL.glmnet", c("SL.glmnet", "screen.corp")),
##   method = "method.AUC")
##
## Risk is based on: Area under ROC curve (AUC)
##
## All risk estimates are based on V = 3
##
##           Algorithm      Ave se      Min      Max
##           Super Learner 0.89558 NA 0.88320 0.91186
##           Discrete SL 0.89675 NA 0.87520 0.91506
##           SL.mean_All 0.50000 NA 0.50000 0.50000
##           SL.glmnet_All 0.89764 NA 0.88667 0.91506
##           SL.glmnet_screen.corp 0.89558 NA 0.87520 0.91186
```