Learning algorithms and cross-validation

Michael C Sachs

SMARTbiomed
Pioneer Centre

Biomarker signature/risk score/...

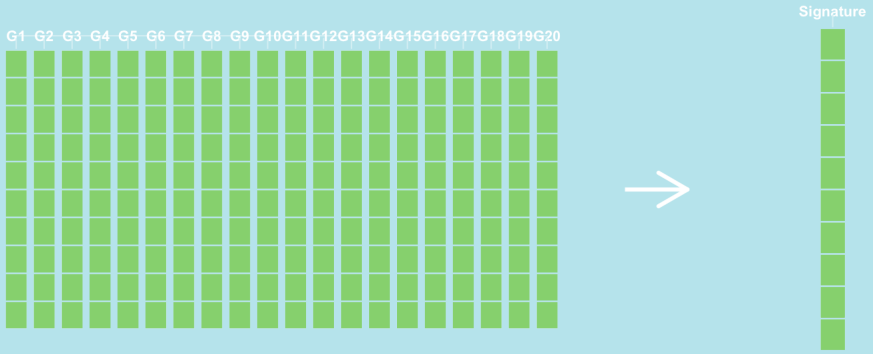


Figure: A set of 20 predictors measured on a number of subjects is translated into a one-dimensional prediction.

A risk prediction model

> *A **risk prediction model** is a transformation of multiple individual features to a one-dimensional space that coincides with the outcome space.*

- A model that *reliably predicts* an outcome may be useful for treatment selection or prognosis.
- A model that *discriminates* between groups that would be treated differently may be clinically useful.
- A signature may be continuous, binary, or take multiple discrete values.

The performance of the signature is evaluated with a variety of measures, but ...

Key point

The development of a model must be cleanly separated from its evaluation

Outline

Issues in development and evaluation of models for prediction

- Modeling methods
- SuperLearner
- Methods for evaluation/validation
- Pitfalls in evaluation/validation

Decisions, Decisions

- Do I include all features or a subset?
  - Which subset?
- How do I combine the features?
  - Transformations?
  - Weight and combine with regression coefficients?
  - What coefficients?
  - Thresholds/Cutoffs before or after combining?

Signature Development

Let X denote the set of p features. The signature is an unknown function

$$f(X) : \mathbb{R}^p \mapsto \mathbb{R}^1$$

- Development Phase Goals:
  - Estimate f based on some *training data*
    - Based on association with outcome Y
  - Provide a valid estimate of performance of the method in which f is estimated
    - Depends on the true signal in the data
    - and the manner in which f is estimated
  - Provide a specification of f for others to use (optional)

Methods for signature development

Broad classes of methods

1. Filter + combine
2. Regularization
3. Other

Dataset includes X_{ij} and Y_i, A_i for $i = 1, \ldots, n$, $j = 1, \ldots, p$
The $p > n$ problem

Filter + combine

General idea:

- Number of features, p, is too large to incorporate all of the X_j values into a multivariable prognostic model
- Select only those genes with a promising univariate association with Y.
- First step, for each $j = 1, \ldots, p$, calculate the two-sample t-statistic **comparing each X_j to Y**
- Then, select only the top 5% of t-statistics in absolute value.
- Second step, those top 5% are then included in a multivariable regression model to predict Y

Variations:

- Statistic used for comparison
- Threshold for inclusion
- Regression model used to combine

Regularization

Least squares:

$$\sum_{i=1}^{n}(Y_i - X_i^T\beta)^2.$$

Penalized least squares:

$$\sum_{i=1}^{n}(Y_i - X_i^T\beta)^2 + h(\beta, \lambda),$$

where h is a penalty function depending on fixed penalty parameters λ.

Bias-variance tradeoff

- A bit of bias can improve predictions

Variable selection

- Different h have different nice properties
  - Lasso: $h(\beta, \lambda) = \lambda \sum_{j=1}^{p} |\beta_j|$
  - Ridge: $h(\beta, \lambda) = \lambda \sum_{j=1}^{p} (\beta_j)^2$
  - Elastic net: $\lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} (\beta_j)^2$

Works for many regression models

- Logistic
- Cox
- Can add/subtract a penalty term to any loss function to be optimized

Other

- Clustering/Principal components analysis
- Regression trees
- Other regression models
- **Stacking**/ensemble methods

Stacking/ensemble methods

- Unless you generated the data, it is impossible to predict which prediction development algorithm will work best
- The principle of clean training-validation separation makes it hard to choose one method a priori
- instead, use them all and average the results!

Define a *library* of algorithms, each of which gives you a way to estimate a predictive model.

Superlearner

On a V-fold cross-validation partition of the data,

1. Fit each algorithm separately, on the training portion. Get V model fits for each algorithm
2. Obtain predictions for Y on the validation set for each fold, get predictions for each subject and algorithm
3. Compute the cross-validated performance for each algorithm
4. Regress the outcome Y on the set of cross validated predictions to obtain optimal weights
5. Fit each algorithm on the full data, combine with the optimal weights estimated in step 4.

Outperforms any individual algorithm in the library

Polley, Eric C. and van der Laan, Mark J., "Super Learner In Prediction" (May 2010). U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 266. `https://biostats.bepress.com/ucbbiostat/paper266`

Considerations for incorporating treatment

Looking for a strong and qualitative interaction with treatment indicator A

- Expand design matrix X to include all treatment by feature interactions $X_{ij} \times A_i$
- Grouped lasso: include interaction terms and force in the main effects

Presence of interaction does not guarantee clinical utility

You need to be able to compute the predictions for people who haven't yet received treatment!

Performance?

Examples of ϕ:

- Mean squared error: $E\{\hat{Y} - Y\}$, `mean(Y.hat - Y)`
- Classification accuracy, sensitivity, specificity
- ROC curve, AUC
- Hazard ratio, absolute difference in survival
- Odds ratio
- P-value from logrank test?

Calibration

Accurately predicts clinical outcome of interest

Discrimination

Separates into groups that clearly distinguish the outcome

Valid estimates of performance

Let S denote the development dataset, includes X and possibly Y, A and other variables, a sample of size n from distribution $\mathcal{P}$ with domain $\mathcal{X}$.
Let $\mathcal{F} : \mathcal{X} \mapsto \mathcal{D}$ denote the process or algorithm that generates a particular f, i.e. $f \in \mathcal{F}$
Let $\phi_f : \mathcal{X} \mapsto \mathbb{R}$ denote the performance evaluation function, e.g., accuracy of predictive model, magnitude of hazard ratio.
We are interested in estimating $E_{\mathcal{P}}[\phi_f(S)]$, the *generalization error* for a fixed f or maybe for a class $\mathcal{F}$.
An estimate of that is the in-sample empirical estimate: $\hat{E}[\phi_f(S)] = \frac{1}{n}\sum_{i=1}^{n} \phi_f(s_i)$.
However, if the analyst **interacts with** S in the definition of ϕ_f, the estimate will be biased (overfit). I.e.,

$$|E_{\mathcal{P}}[\phi_f(S)] - \hat{E}[\phi_f(S)]|$$

will be large.

Examples of "interacts with S"

1. Working on a genomic classifier for binary Y:
   - Test it out on S, and take a look at the individual-level $\phi(s_i)$ of a classifier $f(x_i)$
   - For i where $\phi(s_i)$ is poor, manually change the value of y_i.
2. Developing a predictor for binary Y:
   - Test the association of each X_j with Y using t-test on S.
   - Select the 50 most significant
   - Put them all in a regression model and test on S
3. Developing a classfier
   - Select 50 most significant genes using S, Split S into S_h and S_t
   - Put them in a regression model estimated using S_t, Test it on S_h
4. Developing predictive signature
   - Split into S_t and S_h, Build clustering model on S_t, Test performance on S_h
   - Performance isn't as good as I expected
   - Go back to S_t and try again using a different approach

Which ones give valid estimates?

An analogy

Let's say I'm conducting a randomized clinical trial testing a new treatment versus a placebo.

- I measure OS, EFS, pCR, DFS, and some PK biomarker
- At the end, test them all and report the most significant difference
- Evaluating ϕ using S while also using S to define ϕ
- "Data-adaptive data analysis"

Obviously not OK. In clinical trials we have pre-registration.

- With signatures, often we use S to develop f, and thus ϕ_f.
- Adapting to the features in S, not necessarily the distribution that generated S

Overfitting

Remedies

1. Split-sample
2. Cross-validation
3. Bootstrapping
4. Pre-validation

Split sample

- Partition S into S_t and S_h with sample sizes n_t and n_h
- Hide S_h from yourself
- Generate an f_t using S_t only
- Estimate is $E[\phi_{f_t}(S_h)]$
- *Error for the fixed* f_t

```r
holdoutest <- function(trratio = .5, data){

  npart.tr <- floor(trratio * n)

  train.dex <- sample(1:n, npart.tr)
  hold.dex <- setdiff(1:n, train.dex)

  train <- data[train.dex, ]
  hold <- data[hold.dex, ]

  selecttr <- selectvars(train)
  fit <- fitclassifier(train, selecttr)
  estimate_performance(fit, hold)
}
```

Cross validation

- Leave out sample i and estimate f using S_{-i}
- Get single estimate $\phi_f(s_i)$
- Repeat a number of times
- Average the single estimates
- *Generalization error for the class* $\mathcal{F}$

```r
cross_validate <- function(K = 50, data){

  over <- partition_index(data, K = K)

  cvests <- sapply(over, function(index){

    leaveout <- data[index, ]
    estin <- data[setdiff(1:n, oot.dex), ]

    selectin <- selectvars(estin)
    fit <- fitclassifier(estin, selectin)

    estimate_performance(fit, leavout)

  })
  rowMeans(cvests, na.rm = TRUE)

}
```

Bootstrapping

- Randomly sample S_b from S, with replacement
- Derive f using S_b
- Estimate using samples not selected S_{-b}
- Repeat and average
- *Generalization error for the class* $\mathcal{F}$

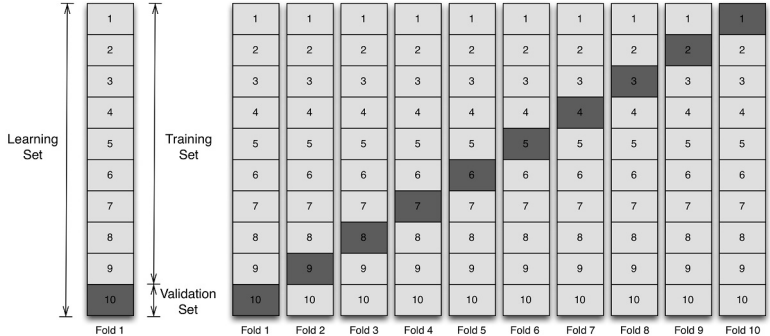
```r
bootstrap <- function(B = 50, data){

  bootests <- replicate(B, {

    boot.dex <- sample(1:n, n, replace = TRUE)
    bin <- data[boot.dex, ]
    notbin <- setdiff(1:n, unique(boot.dex))
    leaveout <- data[notbin, ]

    selectin <- selectvars(bin)
    fit <- fitclassifier(bin, selectin)

    estimate_performance(fit, leaveout)

  })

  rowMeans(bootests)

}
```

Pre-validation

- Leave out sample i and estimate f using S_{-i}: $\hat{f}_{-i}$
- Get single fitted value $\hat{y}_i = \hat{f}_{-i}(s_i)$
- Repeat for all i
- Compare $\hat{y}_i$ with y_i
- *Generalization error for the class* $\mathcal{F}$

```r
y.hats <- lapply(over, function(oot.dex){

    leaveout <- data[oot.dex, ]
    estin <- data[setdiff(1:n, oot.dex), ]

    selectin <- selectvars(estin)
    fit <- fitclassifier(estin, selectin)

    preval <- predict(fit, newdata = leaveout, type = "response")
    preval

  })
```

X-Validation image



Rose (2010, 2016)

Figure: Cross-validation

Common errors

1. Incomplete/partial validation
   - Feature selection performed on full data
   - Then only regression model validated
2. Resubstitution
   - Model built using training sample
   - Performance estimated using full data
3. Resubstitution (2)
   - Model build using full data
   - Performance estimated using holdout sample

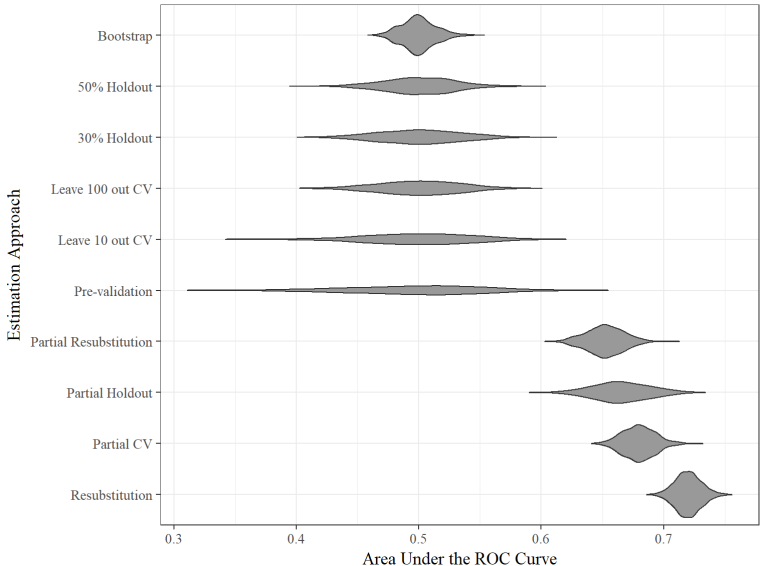
Numerical experiment

- Data were generated with n samples, each with a binary outcome Y with prevalence 0.3
- p features sampled from the standard normal distribution.
- This is the null case where no features are associated with Y.

Signature estimation

- Each feature is regressed against Y in a univariate logistic regression model.
- The 25 features with the smallest p-values are selected
- Logistic regression model defines our final signature.

Results: $\phi = AUC$



Common source of trouble

Example 4: I decided on a statistical model in advance (e.g., lasso), did training/validation split.

Performance wasn't as good as I expected on the validation set, so I went back and used random forest.

Don't do this! Use SuperLearner instead.

Summary

- Omics-based signatures are particularly vulnerable to overfitting due to high dimensional data
- Potential for model complexity is high
- Select relevant features with true associations
- Avoid fitting to random noise in the observations
- Use one of the many approaches to get valid performance estimates