Statistical Methods for High Dimensional Biology

Supervised learning II: Evaluation and Regularization

Keegan Korthauer

22 March 2021

with slide contributions from Sara Mostafavi

Announcements

- Analysis assignment is due one week from today (29 March)
- Wednesday's lecture (24 March) will be asynchronous - I'll be introducing GWAS
- Next week's lectures (29 and 31 March - our last! 😭) will both be **synchronous**
    - Guest lecturer **Dr. Yongjin Park** will be back with us to talk about causal inference in genomics on 29 March
    - Guest lecturer **Dr. Jessica Dennis** will be with us to talk about polygenic risk scores and phenome-wide association studies on 31 March

Learning objectives

- Explain the purpose of **supervised learning** and how it differs from **unsupervised learning**
- Understand the importance of evaluating supervised learning models on **test and/or validation sets**
- Explain the procedure of **cross-validation** and what it is used for
- List several metrics for **evaluating binary classification procedures**, and describe advantages and disadvantages of each
- Explain why **regularization** is useful when building supervised learning models from high-dimensional datasets
- Understand the main mathematical ideas behind the **ridge, lasso, and elastic net** regularization procedures

Recall: Supervised learning

A procedure or algorithm which uses a set of **inputs** (measured or preset variables) to predict the values of one or more **outputs** (variables which are influenced in some way by the inputs)

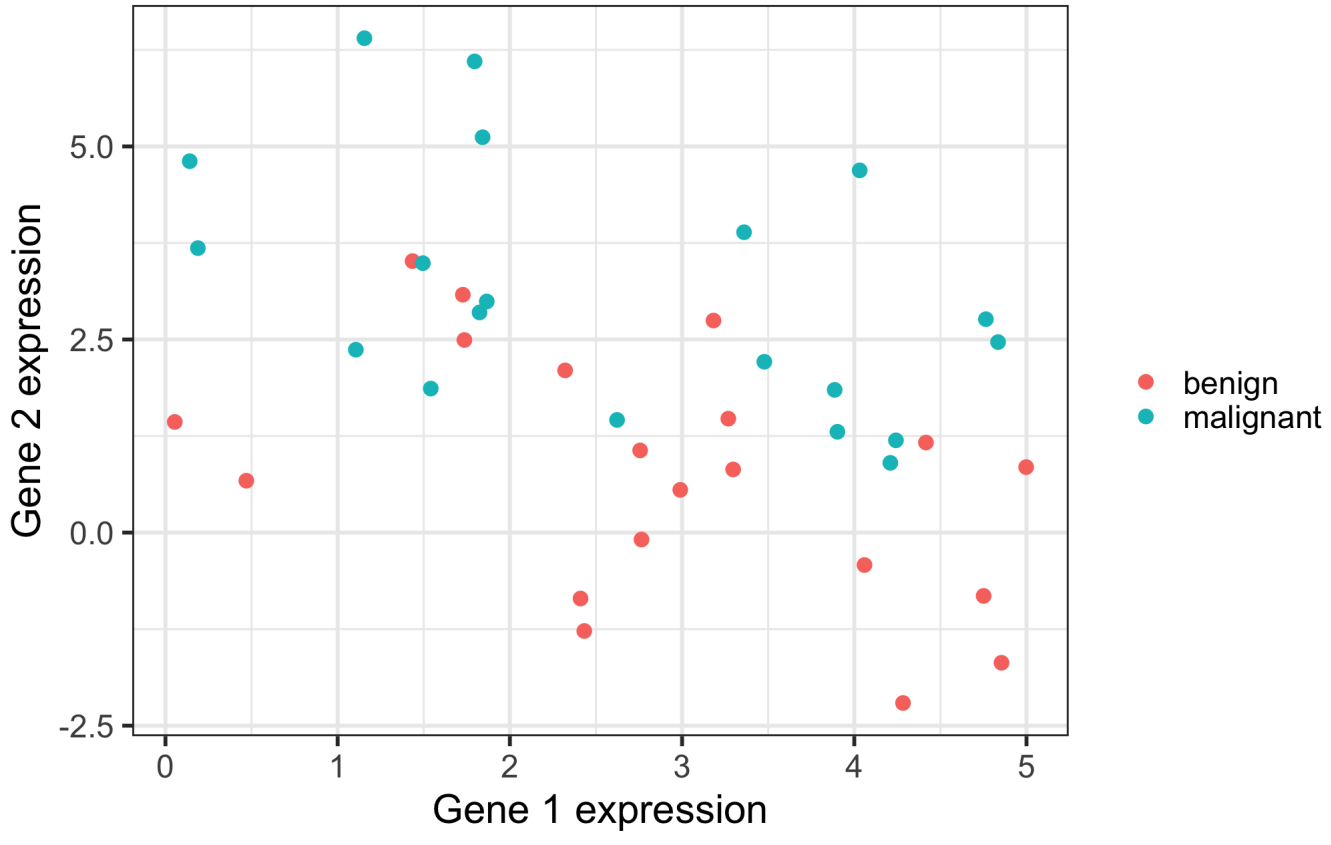
- We have a training data set of n examples in which we know both the inputs $(\boldsymbol{x}_i)$ and outputs (y_i), where $i = 1, \ldots, n$
- Based on this training data, we will produce a model (function) to predict y_i from $\boldsymbol{x}_i$
  - this model can be parametric or non-parametric
- Using the model, we can predict y on *new samples* of a **test set** in which we only know $\boldsymbol{x}$
- When y is continuous, this is **regression**; when y is discrete/categorical, this is **classification**

How do we find the **best** model?

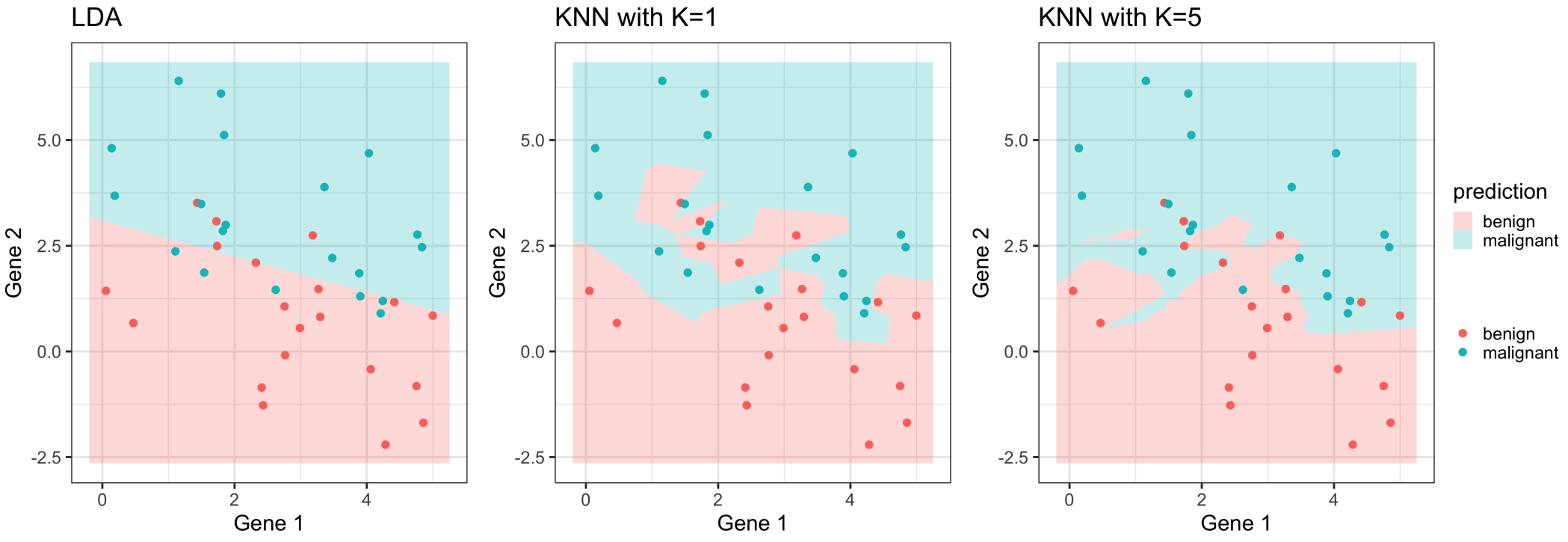
How do we compare models?

Example classification task

Train a classifier to predit tumor type from gene expression data



Which classifier is better?



What makes a good classifer?

A good prediction model should predict the class labels of the samples in the **test set** accurately

In other words, the model should **generalize**

Overfitting

If we allow very complicated predictors, we could **overfit** training data

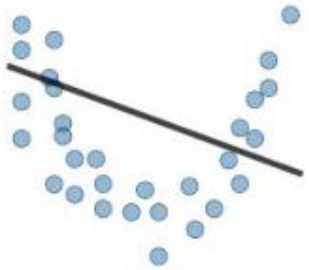
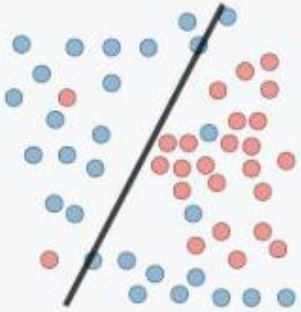
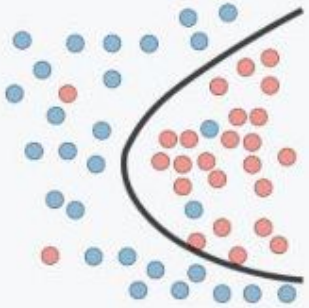
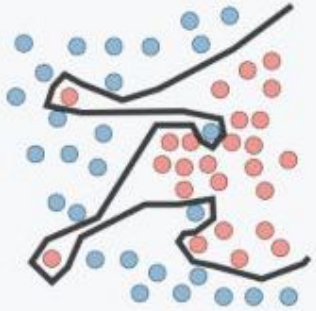
	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			

image source

Bias vs variance tradeoff in supervised learning

- **Bias**: error in assumptions of the learning algorithm
  - results in missing the relevant relations between features and target outputs (underfitting)
- **Variance**: error from sensitivity to small fluctuations in the training set
  - results in modeling the random noise in the training data, rather than the intended outputs (overfitting)

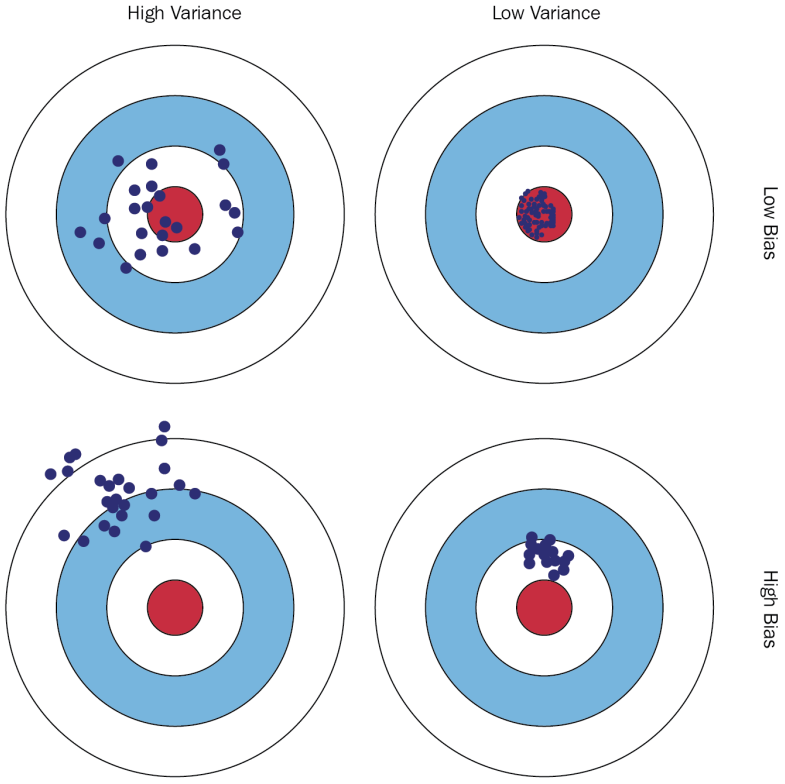


image source: Hands-On Transfer Learning with Python by Sarkar et al.

Bias-variance connection to prediction error

One metric for prediction error in regression is **Mean Squared Error (MSE)**:

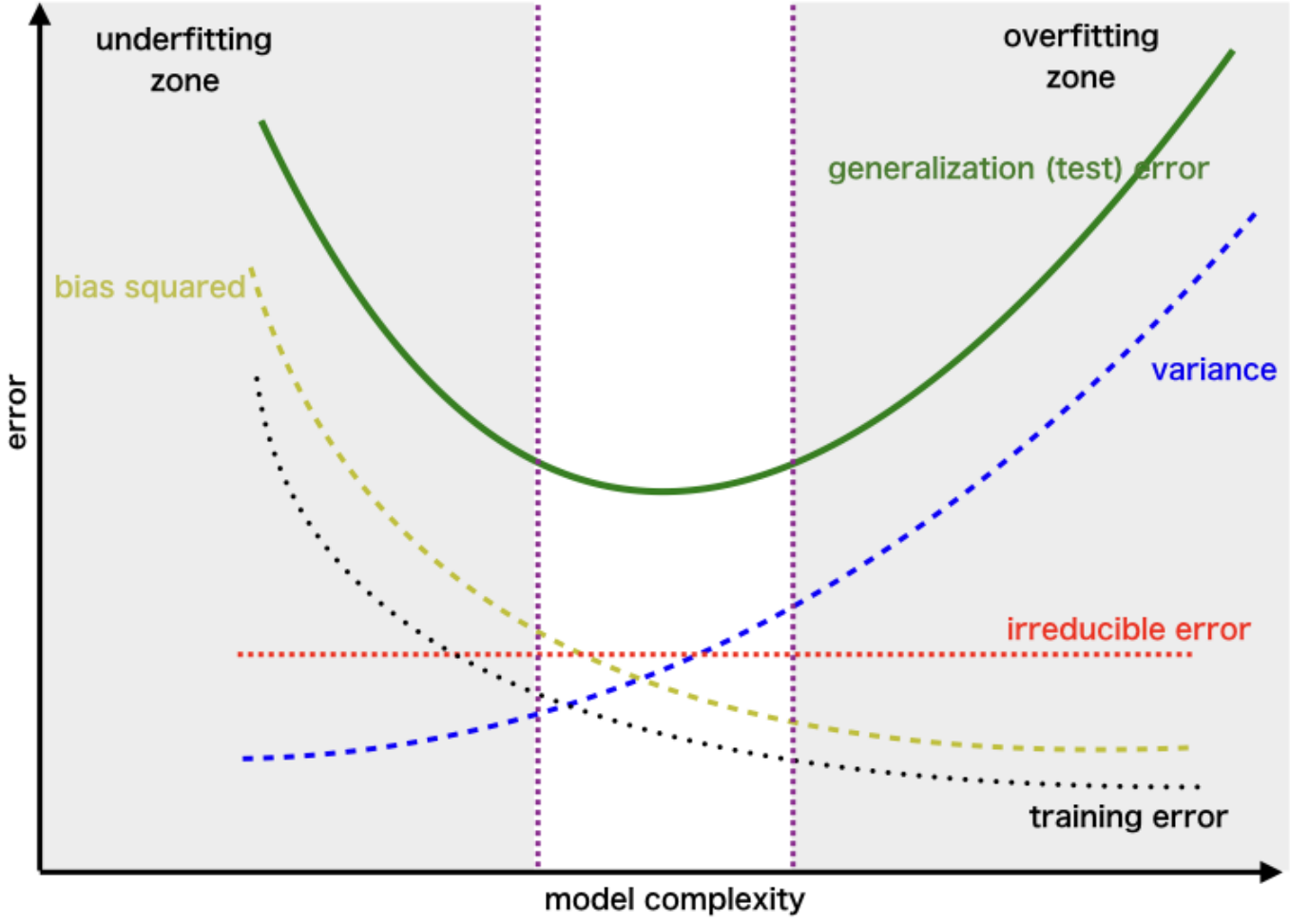
$$MSE(\hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

It can be shown that

$$MSE(\hat{\boldsymbol{y}}) = Var(\hat{\boldsymbol{y}}) + (Bias(\hat{\boldsymbol{y}}))^2$$

Error and model complexity

image source



Model evaluation

What to use for our test set?

How do we know what the truth is in our test set samples?

Golden rule: the model does not touch the test set data *in any way*

- Train the model on one set of samples (**training set**) and use a completely separate of samples (**test set**) to evaluate
- If we evaluate *more than one model* on the test set, then ideally we have yet another completely separate and independent set of samples (**validation set**) for final evaluation
- If an independent dataset for testing is not available, we need to create one using part of our training data - how?

Cross-validation

- Split the training data into subsets/partitions, and use some to train the model and others to test the model's prediction
- This is a general approach for estimating the error of the model in an unbiased way
- **K-fold cross-validation**: multiple rounds of cross-validation are performed using different partitions, and the evaluation results are combined to give an estimate of the model's predictive performance and its variability

K-fold cross-validation

1. Divide input data into K approximately equal-sized partitions (folds, F_i), indexed by $i = 1, \ldots, K$, each with n_i samples

2. Set aside one of the partitions (folds) for the test set

3. Train the model on all input data *except* the held-out fold

4. Measure cross-validation (CV) error using data from the held-out (test set) fold i: $Error_i = \frac{1}{n_i} \sum_{j \in F_i} L(y_j, \hat{y}_j^{(-i)})$

5. Repeat, holding out a different fold each time until CV error is computed for all folds $i = 1, \ldots, K$, and average: $Mean\ Error = \frac{1}{k} \sum_{i=1}^{k} Error_i$

Example: 3-fold cross-validation

- Randomly divide $n = 12$ samples into $K = 3$ equally sized folds
- Train 3 different models
  - Model 1: train on folds 2 + 3, test on fold 1 (get CV error for fold 1)
  - Model 2: train on folds 1 + 3, test on fold 2 (get CV error for fold 2)
  - Model 3: train on folds 1 + 2, test on fold 3 (get CV error for fold 3)
- Average the CV error across the three models

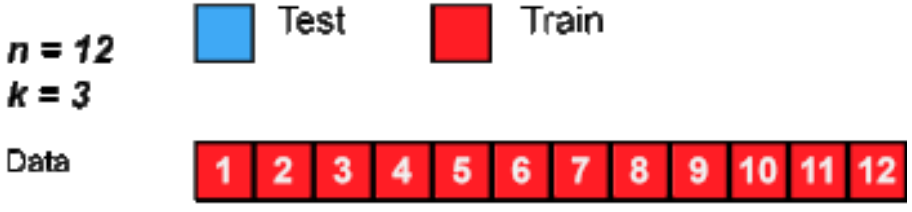


image source

Choice of K

Assume we have n samples:

- What is the *smallest* K we can choose?
- What is the *largest* K we can choose?

Larger K:

- variance of CV error smaller
- computational time larger

Smaller K:

- variance of CV error larger
- computational time lower

Extreme case: $K = n$

If we perform K-fold cross-validation with $K = n$, this is a special case called **leave-one-out cross-validation**

image source

In general, smaller values of K are usually used, unless the sample size n is relatively small

How to measure error on test/validation set?

Examples for a continuous response

- Squared error loss

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Mean Squared Error (MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Absolute error loss

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Evaluating classifier performance on test set

Categorical response:

- Misclassification rate: $\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{\{c_i \neq \hat{c}_i\}}$

A *buffet* of metrics for a **binary** response:

	Actual Positive	**Actual Negative**
Test Positive	True Positive (TP)	False Positive (FP)
Test Negative	False Negative (FN)	True Negative (TN)

- Accuracy (1-error rate): $\frac{1}{n}(\mathrm{TP}+\mathrm{TN})$
- Sensitivity / True Positive Rate (TPR) / Power / Recall: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
- Specificity / True Negative Rate (TNR): $\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$
- Precision (1-FDR): $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$

Evaluating classifier performance on test set

Chef's tasting menu (combine buffet items)

- F_1 score: harmonic mean of precision and recall:

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

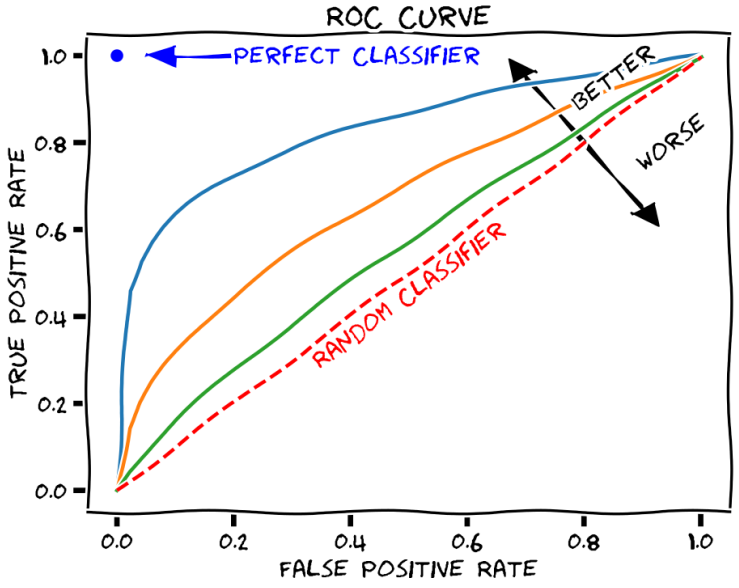
- Matthew's correlation coefficient:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Evaluating classifier performance on test set

Chef's tasting menu (combine buffet items)

- **AUC/AUROC**: Area under the Receiver Operating Characteristic (ROC) curve



FPR and TPR computed over a range of decision boundaries

image source

Note on unbalanced data

- Several metrics can be misleading for highly **unbalanced** data (i.e. where the vast majority of samples are from one class)
- For example, consider the case where 95% of samples in the training and test sets are positives. If we pick a classifier that only predicts positive, then the accuracy $\left(\frac{1}{n}(\mathrm{TP} + \mathrm{TN})\right)$ of our classifier in the test set is 0.95
- AUROC has the same problem
- MCC and F_1 are more balanced metrics in terms of considering the precision in both classes even when data is unbalanced

Summary so far

Supervised learning

1. Gather/measure data (features & response)
2. Formulate a model
3. Fit/estimate model parameters to minimize expected loss
4. Apply model to held out test set & evaluate

Beware of overfitting!

If error on training set << error on test set

Cross-validation

- Good first-pass solution for tuning model parameters
- Assumes training and test sets are independent (this might not be true, especially due to systematic artifacts in genomics, which could lead to inflated accuracy estimates)

What if we have a high-dimensional feature set?

Enter: regularization

- In high-dimensional biology, we are typically faced with thousands to millions of features / covariates ($p >> n$ problem)
- **Problem**: it usually won't work very well (or at all) to input all ~20K genes in a gene expression experiment (or ~1M SNPs in a genotyping array dataset) to build a classifier of the response (e.g. phenotypic trait, disease, or other outcome)
- A main analysis goal is to **identify** features / covariates that are **important** in predicting the response (feature selection)
- **A Solution**: we can use **regularization** as a tool to perform feature selection

Feature selection options

- **The "filter" approach**:
  - without considering the labels/response: filter features based on variation (for example, only consider the top X genes with the highest variance or coefficient of variation)
  - considering the labels/response: only keep genes significantly correlated with the response (*must be careful here to do this step after holding out test set!*)
- **The "wrapper" approach**:
  - identify features that lead to good performance by the specific classifier, e.g. forward/backward selection using cross-validation
  - computationally intensive for many features
- **The "embedded" approach**:
  - Modify objective function to include a penalty to favor models with fewer features
  - main idea: weights/coefficients of less important features are 'shrunk' to zero

Feature selection in regression

- **Best subset regression**:
  - find subset of size k with smallest error (e.g. mean squared error)
  - unfeasible for omics studies, because $\binom{p}{k}$ is *huge*
- **Forward stepwise regression**:
  - sequentially add the feature that most improves the fit (e.g. minimizes prediction error)
- **Regularization methods (shrinkage/penalization)**:
  - modify objective function to explicitly penalize magnitude of coefficients

Regularized regression framework

Minimize the following objective function:

$$argmin_{\boldsymbol{\beta}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda f_p(\beta)$$

- $(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$ is the sum of squared errors
- $f_p(\beta)$ is a penalization function
- $\lambda \geq 0$ is a tuning parameter to balance squared error vs penalty
  - what happens if $\lambda = 0$?
  - what happens if $\lambda \to \infty$?

Ridge regression

Minimize the following objective function:

$$argmin_{\boldsymbol{\beta}} \, (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda f_p(\boldsymbol{\beta})$$

where the penalty function is the **L2 norm**:

$$f_p(\beta) = ||\boldsymbol{\beta}||_2^2 = \sum_{j=1}^{p} \beta_j^2$$

This leads to coefficient estimates

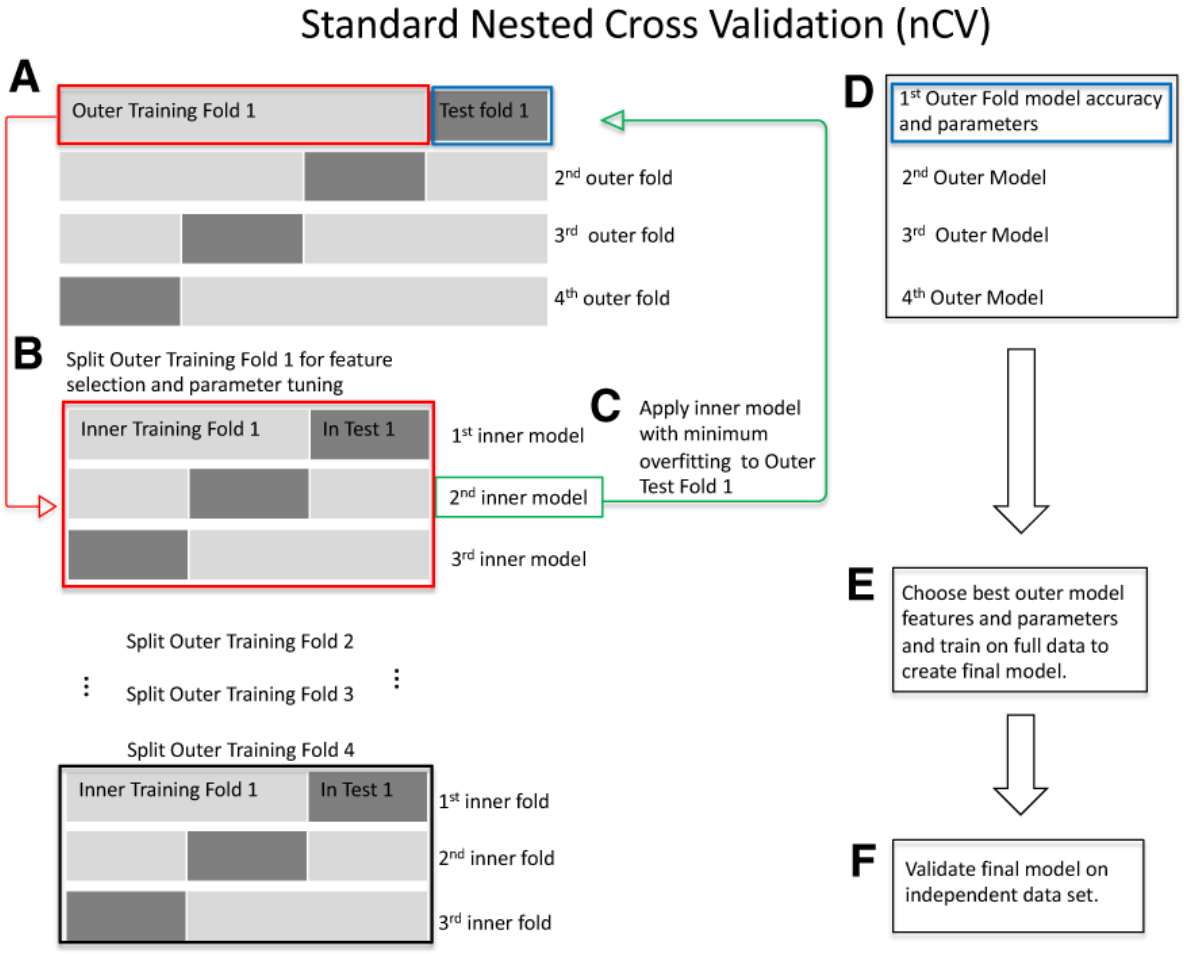
$$\hat{\beta}_{\lambda}^{ridge} = (\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

Important details

- Features/covariates should be **standardized** so they are on the same *scale*
  - if not, then they will have different contributions to the penalty (undesirable)
  - since features have mean 0, no intercept is needed in the model
- Coefficient estimates depend on the value of the tuning parameter λ
  - can use **nested cross-validation** to find (tune) a value of λ that yields coefficient estimates that minimize the test error/loss function

Nested cross-validation

Fig 1, Parvandeh et al., 2020



Probabilistic (Bayesian) interpretation of ridge regression

Recall our objective function:

$$argmin_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \boldsymbol{x_i^T}\boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

The probabilistic (Bayesian) formulation places a prior distribution on the parameters $\boldsymbol{\beta}$:

$$p(\boldsymbol{\beta}) \sim MVN(0,\, \nu \boldsymbol{I_p})$$

Where ν governs the spread of the distribution on higher magnitude coefficient weights - akin to the λ tuning parameter (i.e. if ν is small, coefficients will be more concentrated around zero)

Then the parameter estimates $\hat{\boldsymbol{\theta}} = (\hat{\boldsymbol{\beta}}, \hat{\sigma}, \hat{\nu})$ are obtained via the mean of the posterior:

$$p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{\theta}) = N(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{\beta}, \boldsymbol{\sigma})N(\boldsymbol{\beta}|0, \nu)$$

Another flavour of regularization: lasso

- **Lasso: least absolute selection and shrinkage operator**
- Proposed by Tibshirani (1996)
- Minimize the following objective function:

$$argmin_{\boldsymbol{\beta}} \, (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda f_p(\boldsymbol{\beta})$$

where the penalty function is the **L1 norm**:

$$f_p(\beta) = ||\boldsymbol{\beta}||_1 = \sum_{j=1}^{p} |\beta_j|$$

- The only difference between ridge and lasso regularization is L2 vs L1 norm penalty - but solutions behave *very* differently

Another flavour of regularization: lasso

$$argmin_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \boldsymbol{x_i^T \beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- As before, tuning parameter λ controls the strength of the penalty and is selected with nested CV
  - What happens at $\lambda = 0$ and $\lambda \to \infty$?
- For $0 < \lambda < \infty$, we are balancing two ideas: (1) minimize the squared error, and (2) penalize the magnitude of the coefficients
- Unlike with ridge, the L1 penalty causes some coefficients to be set to **zero exactly**
  - this means lasso is performing *feature selection*
- No closed form solution; requires numerical optimization (e.g. LARS algorithm)

Lasso: limitations and considerations

- If $p > n$, lasso can select at most n features (rest of the coefficients will be set to zero)
- If multiple features are highly correlated, lasso will choose only one among them (arbitrarily) -- **interpret feature selection with caution**

Lasso vs ridge

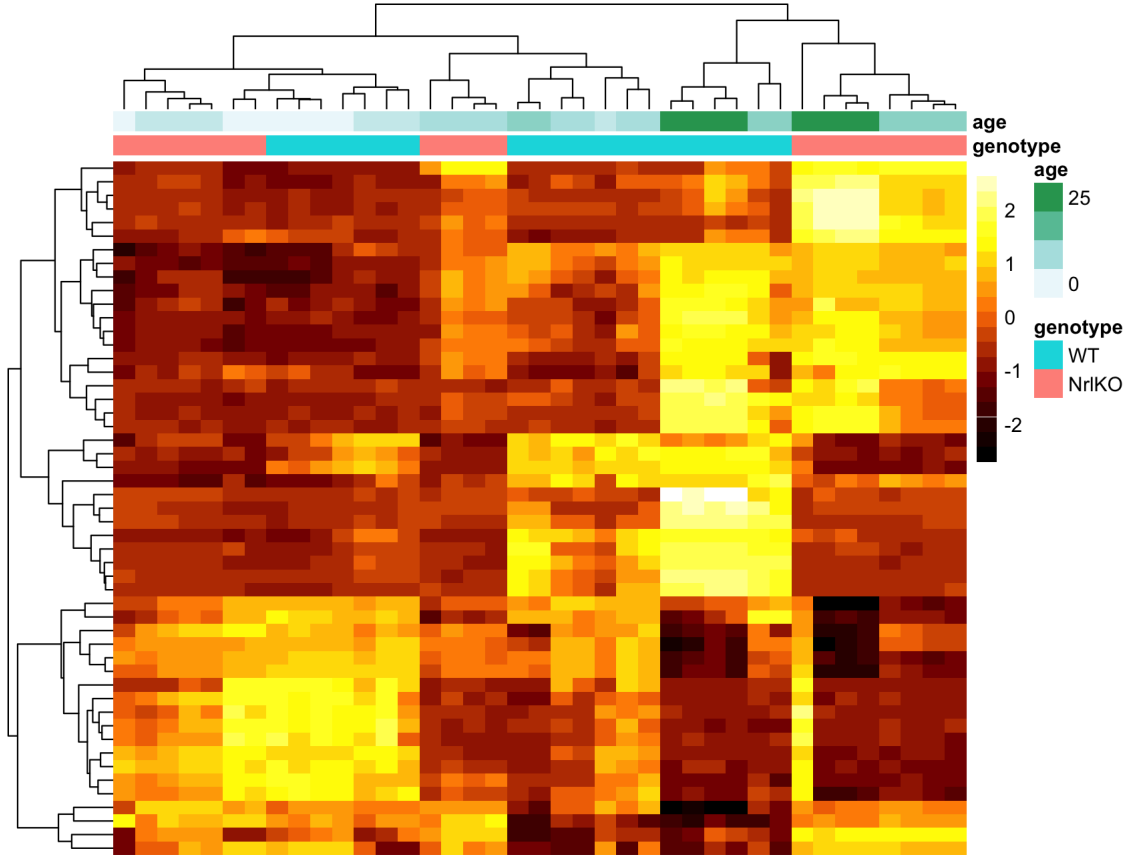
- Both lead to shrinkage of coefficient estimates, but ridge shrinks magnitude whereas lasso shrinks some to zero
- **Predictive performance**: typically comparable
- **Interpretation**: lasso leads to variable selection

Regularization in action

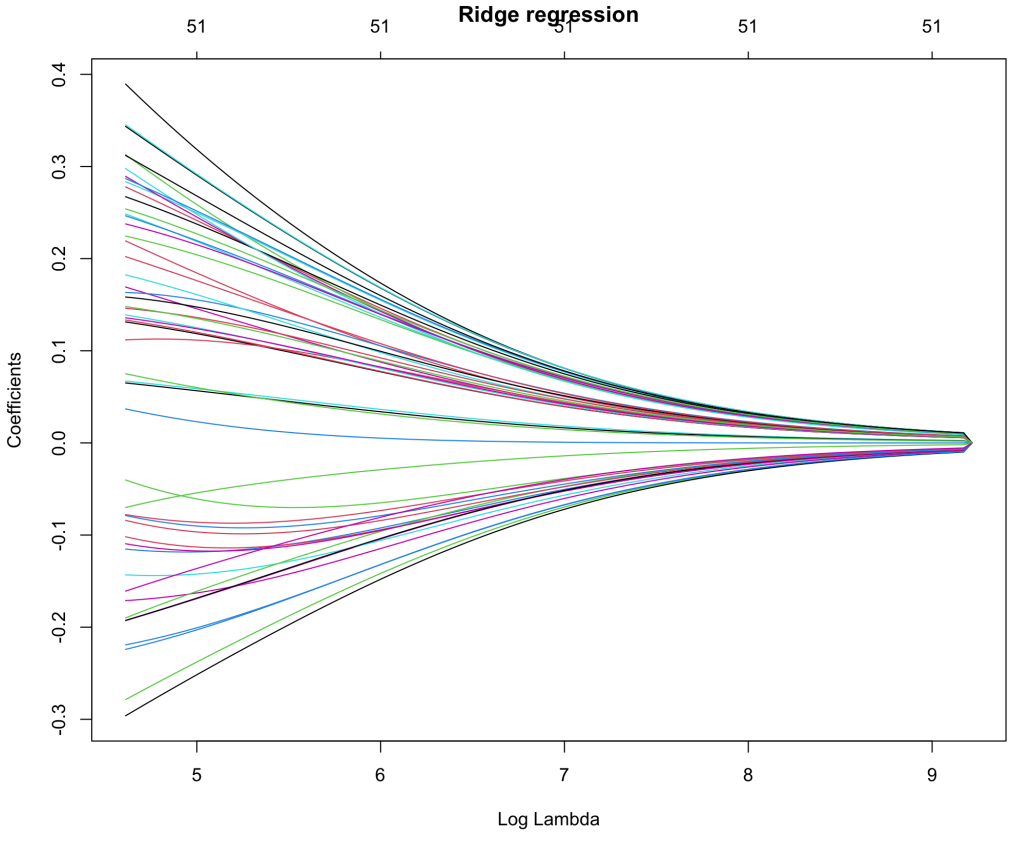
Let's revisit the photoreceptor dataset

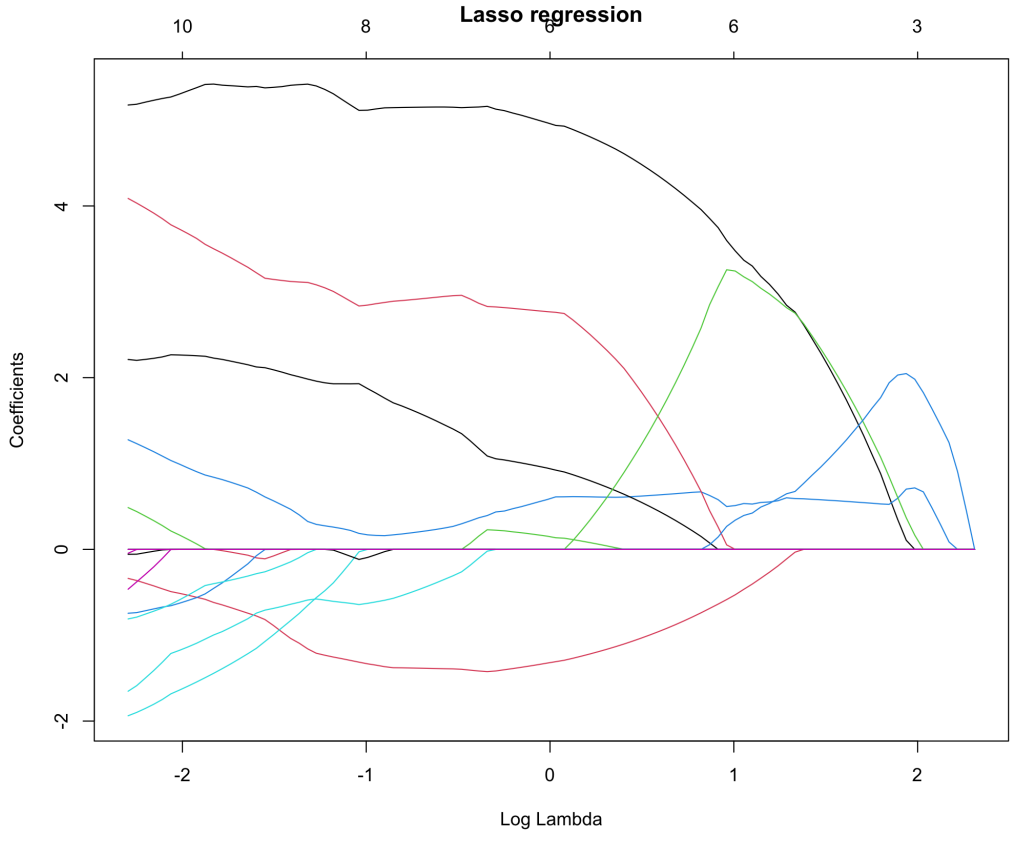
Here's a heatmap of the top 50 most variable genes

Let's try to predict age from gene expression (of these 50 genes)



Ridge vs Lasso: λ trace plots





Ridge vs Lasso: coefficient estimates at the "best" λ

Ridge

```
## 52 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)   8.71794872
## 1416041_at    0.28991957
## 1416306_at    0.16964241
## 1417155_at   -0.22039024
## 1417457_at   -0.11693743
## 1419025_at    0.20476129
## 1419740_at    0.13525811
## 1420484_a_at  0.26797289
## 1421061_at    0.22492372
## 1421084_at    0.19350408
## 1421346_a_at  0.14890543
## 1423631_at    0.05501770
## 1423851_a_at -0.15801095
## 1425171_at    0.11209590
## 1425172_at    0.11340570
## 1425232_x_at  0.23652972
## 1426288_at    0.01831489
## 1429372_at   -0.13912463
## 1431225_at   -0.15693200
## 1433575_at   -0.23270455
## 1434437_x_at -0.11398803
## 1437086_at   -0.06601051
## 1437502_x_at -0.19055391
## 1440256_at    0.23388056
## 1441144_at    0.22628867
## 1441330_at    0.22320704
## 1445574_at    0.10898392
## 1445710_x_at -0.14913461
## 1448182_a_at -0.19202531
```

Lasso

```
## 52 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)   8.7179487
## 1416041_at    5.3573510
## 1416306_at    .
## 1417155_at   -1.2331726
## 1417457_at    .
## 1419025_at    .
## 1419740_at    .
## 1420484_a_at  0.2744969
## 1421061_at    .
## 1421084_at    .
## 1421346_a_at -0.4821727
## 1423631_at    .
## 1423851_a_at  .
## 1425171_at    .
## 1425172_at    .
## 1425232_x_at  1.9402775
## 1426288_at    .
## 1429372_at    .
## 1431225_at    .
## 1433575_at    .
## 1434437_x_at  .
## 1437086_at    .
## 1437502_x_at  .
## 1440256_at    .
## 1441144_at    .
## 1441330_at    .
## 1445574_at    .
## 1445710_x_at  .
## 1448182_a_at  .
```

Hybrid approach: elastic net

$$argmin_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \boldsymbol{x_i^T \beta})^2 + \lambda \Big(\alpha \sum_{j=1}^{p} \beta_j^2 + (1-\alpha) \sum_{j=1}^{p} |\beta_j| \Big)$$

- A compromise between ridge and lasso regression: penalty is a convex combination of the L1 and L2 norm
- Addresses the shortcoming of the lasso in the presence of correlated features
- Additional 'tuning' parameter $\alpha \in [0, 1]$ which weights L1 vs L2 penalties - need to set upfront, or perform additional CV to tune
  - what happens if $\alpha = 0$?
  - what happens if $\alpha = 1$?

Regularized models in R

- Highly recommend the `glmnet` package
- Implements regularized linear regression (lasso, ridge, elastic net)
- Also includes extensions to generalized linear models (e.g. logistic, poisson, multinomial, etc) and survival analysis (Cox regression)
- Includes functions for performing nested CV to select 'best' λ
  - `lambda.min`: λ that minimizes error
  - `lambda.1se`: largest λ that is still within 1 standard error of `lambda.min`

`glmnet` in practice

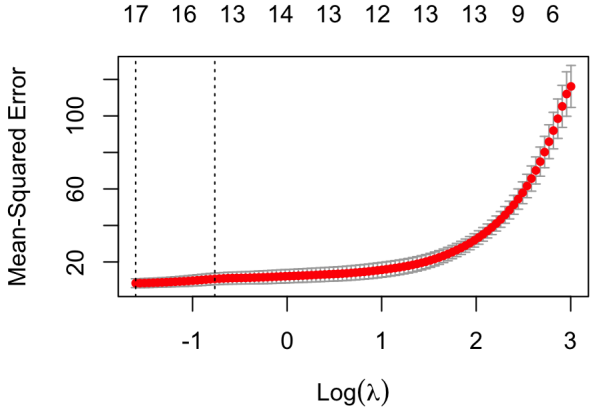
5-fold nested CV for elastic net regression ($\alpha = 0.5$):

- Change `alpha` to 1 for lasso (default), and 0 for ridge
- Default loss is MSE for linear models (change with `type.measure` parameter)
- For other types of regression, change `family` parameter (e.g. `binomial` for logistic regression)

```
en_fit = cv.glmnet(t(x), pData(eset)$age,
                   alpha = 0.5, nfolds = 5
en_fit$lambda.1se
```

```
## [1] 0.4654681
```

```
plot(en_fit)
```



Discussion

Designing variable penalization functions is an active area of research (e.g. fused lasso, group lasso, etc)

Remember the golden rule: the test set should not influence the model/classifier *in any way*

Additional Resources

- Conceptual overview + R implementation: Chapter 12.4-12.6 of Modern Statistics for Modern Biology by Holmes and Huber
- More detailed conceptual overview: Chapters 29 and 33.9 of Intro to Data Science by Irizarry
- Mathematical framework: Chapters 3.4, 7.2-7.4, and 7.10 in Elements of Statistical Learning by Hastie, Tibshirani and Friedman