# UWEO StatR201 Lecture 6

## Predictive Modeling And Cross-Validation

Assaf Oron, February 2013

## Tonight’s Menu - and Looking Forward

First, happy Valentine’s Day if you celebrate it *(does one even have a choice nowadays?)*

Second: last week I told you we were finishing the regression half of class, and starting the “machine-learning” (or however you want to call it) part. **I lied.**

**Ok, ok, I didn’t lie.** We are definitely starting the second half. And last week **did** mark the end of the material typically covered in “classical” regression courses. I just double-checked: neither the 400-level nor the 500-level regression classes given here by the UW stat department, venture into the deep *(and fun!)* waters we will enter today.

But we *will* remain within the general domain of regression for the coming 2 weeks. We will just give it some innovative (and - again - fun) twists. Namely, we will see what needs (or doesn’t need) to change when **regression is viewed as a prediction problem** - and learn tools for model selection. Tonight the more basic and straightforward ones, starting with introducing standard **selection criteria**…

- Penalized likelihood (LRT, AIC, BIC);
- The all-important **Cross-Validation**;
- … all implemented via the standard and simple **model-building procedures**…
- All-, Forward-, and Backward-Subset selection.

…and next week more snazzy procedures, still using similar criteria. Following that: applying similar principles to **classification.**

But before everything else, we spend the next 2-3 slides in **“DE-CRAM 1.5”: a recap of the classical regression half of our course.**

# DE-CRAM 1.5: A Quick Cookbook for Generic Modeling Problems

**1. Initial exploration:**

- Data cleanliness, quality, missingness; identify covariates as continuous/categorical/mish-mash;
- Univariate summaries (`hist/density/qqnorm/table`);
- Bivariate ($x$ vs. the outcome $y$): pairs scatter (continuous-continuous), boxplots (continuous-categorical), 2-way tables (cat.-cat.).

**2. Using #1, get into active and investigative exploration:**

- Experiment with transformation of response/covariates;
- Multi-way plots and tables to suggest potential interactions;
- **Outlier triage**
- Covariate vulnerabilities: correlations/VIF, low information content, imbalance, influential points.

**3. Once you have a working model and you fitted it, you might want to:**

- Interpret the model to figure out what it really says;
- Analyze the residuals (outlier triage, nonlinearities, heteroskedasticity, etc.).
- If relevant, attempt to add terms and examine them via the LRT and/or Wald test.

**The process is by no means linear. You should not be surprised/frustrated by needing to revisit steps 1-2 after you think you're almost finished with #3. That's an integral part of research. In fact, if you've never gone back from #3 even once, it's a reason to be concerned.**

# DE-CRAM 1.5: Cookbook Footnotes

**Footnote A: Outlier triage.** the word “triage” as well as the 1/2/3 designation are my own terminology, but I find them useful. The most important principle, however, is **investigate before taking action.** That’s true even if you need to set up a high-volume system with automated rules: investigate the typical or expected properties before deciding on the rules. Reminder about the 1/2/3 designation:

1. **Data error.** Fix or remove.
2. **Belongs to a different (sub-)population.** Plausible solutions: add indicator, exclude, add sensitivity analysis, workaround by using a more robust method.
3. **An integral part of the main story.** Solutions are similar to #2, except the *“add indicator”* option.

Which option to choose in each case depends **first and foremost - like every other major model-related decision - upon the real-life context of your task. Science, rather than numerical/statistical trickery, should always be front and center.**

**Footnote B: Solution Look-Up.** for problems that might arise, here are some of the tools that might alleviate them.

- **Nonlinearities, skewness; outliers in $y$:** transform $y$ and/or $x$; use polynomials or splines; robust/quantile regression; more rarely, GLMs.
- **Imbalance and influential points in X *(worse than $y$ outliers)*:** transform the covariate, consider excluding covariate or isolated points.
- **Non-continuous Y data:** GLMs (surely for binary/categorical data; not a must for count data).
- **Hetero-Skedasticity** (haven’t talked about this much; generally, any “non-white-noise” aspect of residuals): transformation often solve it; at other times GLMs might do the trick. If not, mixed models (taught in Spring) or more simply, robust SEs / GLS (possibly later this class) should do it.

**Footnote C: No Thresholds are Holy.** Just like the 0.05 p-value is just a number reflecting some consensus common-sense, all other numbers flung around these weeks (VIF>5, correlations of 0.7 or 0.8, etc.) are not holy and you might find that different thresholds, or threshold-free solutions, work better for your needs.

*Questions? (online/in-class)*

## Model Selection: the Problem

**Essentially, all models are wrong, but some are useful.** *(George E.P Box)*

The *teeny* task remaining for us, is **how to choose our (wrong but still useful) model.** If our dataset consists of only a handful of columns, then the tools summarized in DE-CRAM 1.5 Cookbook (+footnotes) might suffice.

Otherwise – meaning in most cases nowadays – **welcome to the biggest open problem in statistics.**

**A problem likely to remain open for a long while, because it is intimately related to the basic questions in the philosophy of science.**

This is especially true for the type of problems for which the above cookbook was designed…

- Explaining how $y$ depends on various covariates, in order to provide scientific insight;
- Informing policy decisions about changes in $x$ that might beneficially affect $y$;
- …in other words, **inference problems.** For those, model-selection is particulary challenging and treacherous.

## Why is Model Selection So Difficult?

1. The space of possible models for a given problem is infinite – and it's not a "small infinity" either.

- Think about the various types of models you already know, that can be applied to the same problem.
- Then, the various transformations available for $y$ and the $x$'s, including splines.
- **Note, that for choosing the right model type and transformations, we cannot use straightforward probability calculations or hypothesis tests – because these tests assume the model is already fixed!**

2. Even after we've settled on model and transformations, we might face a **combinatorial explosion**.

- With $q$ available covariates, there are $2^q$ potential combinations. And that's….
- …without considering **interactions**. For a model with 5 covariates, there 5*4/2=10 possible 2-way interaction terms - and $2^{10} = 1024$ possible combinations of these terms!

3. **Specifically for inference problems, model selection might "contaminate" our conclusions with multiple testing.**

**Let's keep this model-selection problem in mind as we play with a new dataset.**.

## Ok, Let’s Descend upon an Innocent New Dataset

Download the `"autoMPGtrain.csv"` file, containing some specifications of 298 cars sold in the US in the 1970s and early 1980s. **The outcome of interest is fuel consumption (`mpg`).**

We would like to know:

- Generic descriptives
- **Transformations**
- Suspected interactions
- **New covariates?** (e.g., hiding in the names?)

The idea is to reach a “regression ready” state…

…then break.

**Dictionary**

- name: model name
- cyl: # of cylinders
- volume: engine volume (cu. in.)
- hp: horsepower
- weight: in pounds
- accel: apparently a time of some sort (e.g., *zero to 60*)
- year: model year
- continent: 1-American, 2-European, 3-Asian
- mpg: calculated per EPA city/highway mix (?)

## Model Selection - and Prediction

**Prediction is very difficult, especially about the future.** *(Niels Bohr?)*

For statistical prediction models, this hillarious quote is (at least) **doubly wrong:**

1. Statistical *"Prediction"* is a generic term for **estimating the outcome's value at any combination of time/space/other-covariates,** at which an observation is not available - or an additional observation might be expected. **This might occur in the past, in the future, in the present, or without any time context.**
2. With statistical models, **switching from inference to prediction actually makes the model-selection problem much easier.** This is because concerns about interpretability, probabilistic uncertainty, explanatory power, scientific validity, etc., are now relegated to second place – trumped by **predictive performance.**

As we shall see very soon, there is still considerable overlap between model-selection methods for inference and prediction. But with the latter, the task becomes refreshingly simpler.

## `Prediction` in R: Simple Example

```r
challenger = read.csv("../Datasets/challenger.csv", as.is = TRUE)
chalmod = glm(cbind(Eroded, Intact) ~ I(tempF - 70), data = challenger, family = binomial)
predtemp = c(50, 40, 31)
predict(chalmod, newdata = data.frame(tempF = predtemp))
```

```
##       1       2       3
## -0.1577  1.6372  3.2527
```

Notes:

- If `newdata` is omitted, the fitted values for all observations are returned (a bit redundant since they already exist in the `glm` object). Otherwise, **`newdata` must be a data frame, including all covariates participating in the model** (other covariates not a must).
- The default `predict` returns the predictions on the model (`'link'`) scale. You can change that…

```r
predict(chalmod, type = "response", newdata = data.frame(tempF = predtemp))
```

```
##      1      2      3
## 0.4607 0.8372 0.9628
```

If you want to get halfway-decent confidence intervals for GLMs, however, you'll have to do it "by hand" via model-scale predictions.

## Prediction in R: Simple Example

```
chalpred = predict(chalmod, se.fit = TRUE, newdata = data.frame(tempF = predtemp))
expit = function(x) exp(x)/(1 + exp(x))
data.frame(Temp = predtemp, Mean = expit(chalpred$fit), CI95.lower = expit(chalpred$fit -
    2 * chalpred$se.fit), CI95.upper = expit(chalpred$fit + 2 * chalpred$se.fit))
```

```
##   Temp   Mean CI95.lower CI95.upper
## 1   50 0.4607     0.1497     0.8056
## 2   40 0.8372     0.2680     0.9863
## 3   31 0.9628     0.4015     0.9990
```

Note that **these confidence-intervals are for the expected response value (=mean) at the specified covariate values. The intervals for an individual future data point are much wider, because they must also incorporate the basic single-observation uncertainty.** This comment is more relevant for linear regression than for binary outcomes (when all you can get is *yes* or *no*), but keep it in mind.

*Questions? (online/in-class)* Try it a bit.

## Generalizability vs. Over-Fitting

Whether done for inference or prediction, we typically see our regression dataset **as coming from a broader population.**

Therefore, it is crucial that **our model will successfully generalize to that broader population.**

As seen in HW2, one can keep increasing the likelihood by adding terms. Similarly, one can keep reducing the least-squares sum. In fact, if $y$ has $m$ unique values ($m \leq n$), then **a model with $m - 1$ terms can fit it perfectly.** ^

But overfitting our model to the nuances of the dataset under our nose, will clearly **not help achieve the goal of generalizability** (to put it mildly).

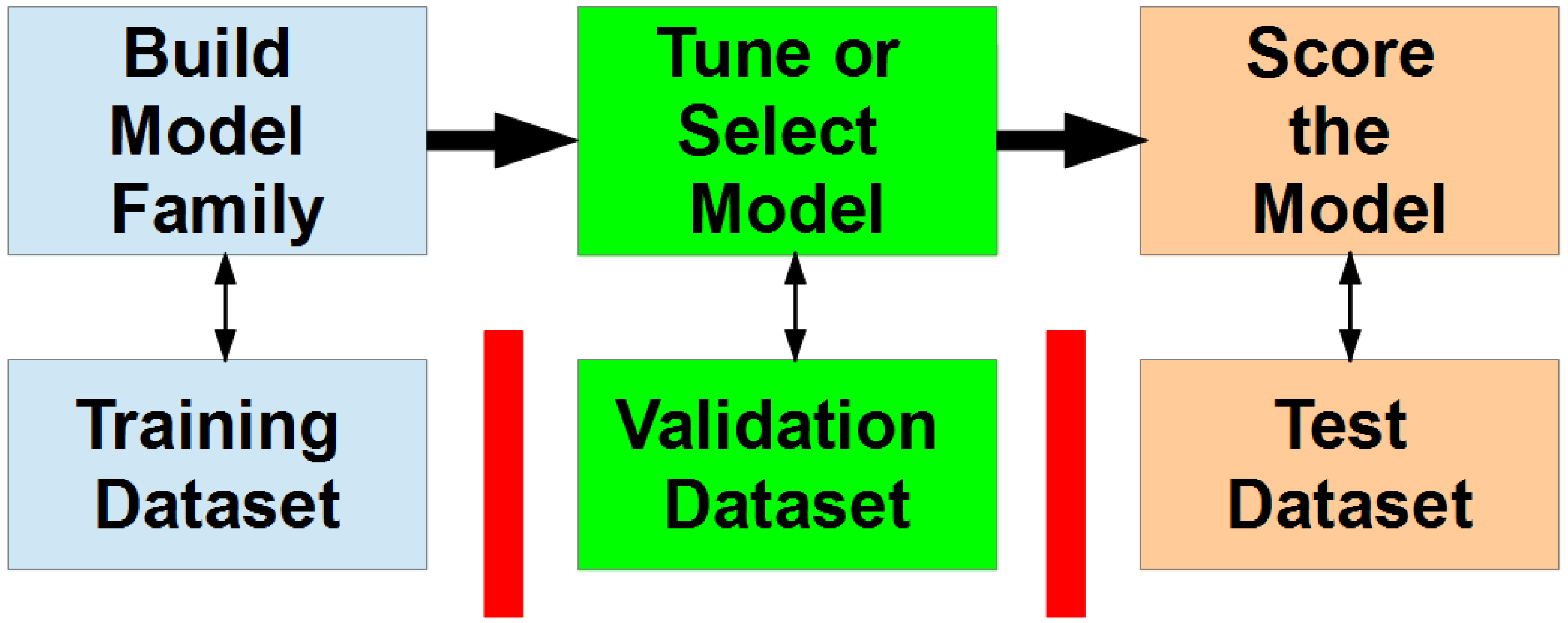
This is the rationale and common-sense underlying the rather lengthy discussions and derivations of Hastie et al.'s Chapter 7. **The Bias-Variance Tradeoff**, a rather standard term which this chapter repeatedly invokes, could be confusing for some. For example, a linear regression *per se* is not biased, regardless of its complexity. Perhaps a better term (not mathematically, but explanation-wise) is **model inaccuracy**.

A model lacking in detail does not even describe the dataset very accurately – and of course, cannot be expected to accurately predict data it hasn't even seen. However, a too-detailed model will indeed have a high variance, because every change in the dataset would substantially change the model.

In general – as the book says – **our true prediction error out in the real world, will likely be larger than the one we see on our sample.** Chapter 7 documents some ways in which theorists have been able to quantify the evils of **overfitting** (*model optimism'*, etc.), and the value of **parsimony** or priniciples such as **Occam's Razor.**

^ *(Footnote: for logistic regression, $m$ will be the number of "events", i.e., 'yes' observations. A common rule-of-thumb to avoid overfitting in logistic regression, is to use no more terms than 1/5th the number of events)*

# Prediction: The Training-Test Process (idealized)

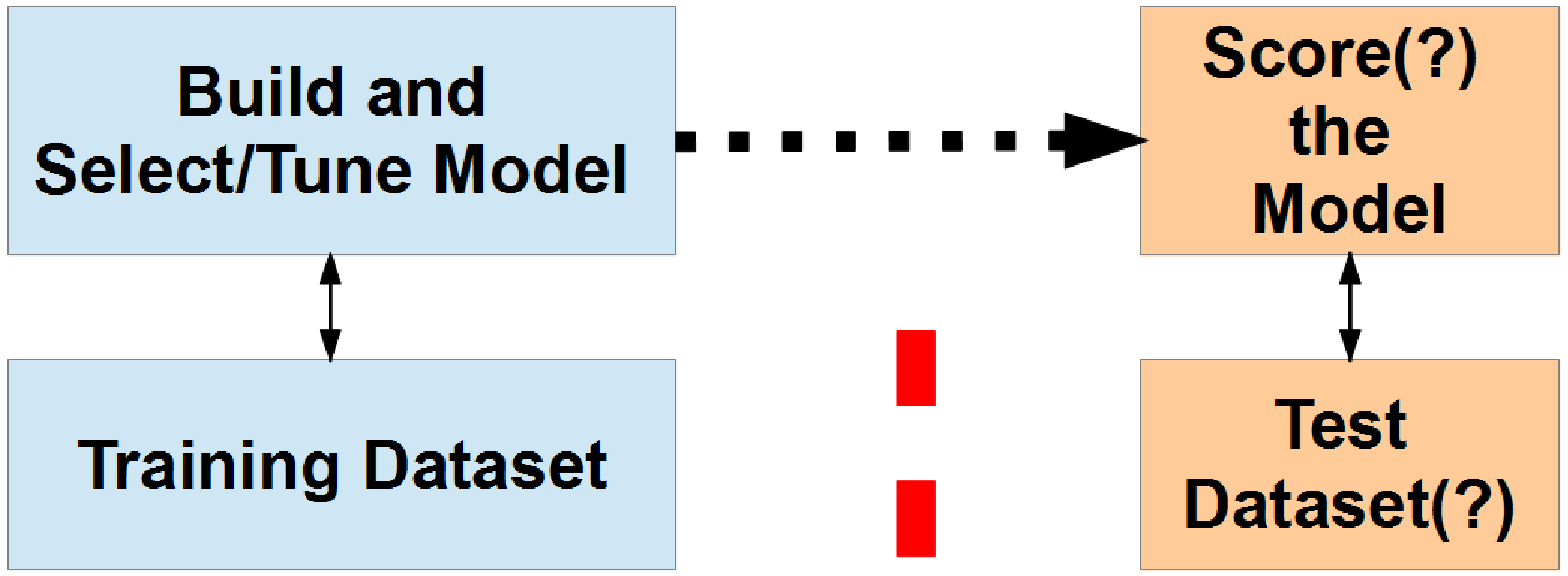


Ideally,

- There will be a **firewall** between the 3 (sub-)datasets. The **test set** provides an idea of the final model's **out-of-sample predictive abilities.** Hastie *et al.* suggest a 50:25:25 split between the sub-datasets.
- The process will be as **automated** as possible (this might also save work in the long run).

If the diagram reminds you of **insect metamorphosis…** then you are onto something. And just like with insects, in practice it is quite common to encounter "partial-metamorphosis" beasts, with only 2 stages instead of 3.

## Prediction: The Training-Test Process (2-stage, “naive”)



Why would people skip the validation stage and dataset (apart from plain ignorance or laziness)?

- **Overall dataset not large enough.** You’d be surprised how sub-dataset peculiarities and differences persist, and how easily model systems can train/tune themselves to **overfit** these peculiarities. As a rule of thumb (my own :), if $n < 10^4$ it is probably not a good idea to separate out a validation dataset, unless your method (or bosses) *absolutely* requires it.
- **Method incompatible with validation-dataset tuning.** Some of the simplest selection methods don’t really know what to do with a separate validation-only dataset.
- **Cost-Effect isn’t there.** Perhaps the most common reason: not all methods benefit from a 3-way split.

# Criteria for Naive Model Selection: Penalized Likelihood

What I call here *“naive”* model selection, performs the selection **in-sample**, on the same data used to train/fit the model. We are looking for is some way to make larger models “pay” for their tendency to over-fit. Something, that will set the bar higher with more parameters.

**Wait! We’ve already got one. It is called Likelihood-Ratio Tests.** *(or, u could use Wald tests :)*

We require the Deviance (or fit) of the larger model to be better (=smaller), by an amount that beats the significance threshold of the associated Null distribution – a distribution that plays Devil’s advocate by assuming that the smaller model is better. Symbolically for the LRT, if

$D_0 - D_1 >$ `qchisq(1-alpha,df=p_1-p_0)`

Then we conclude “Model 1” is better than “Model 0”. (Reminder: $D = -2\lambda > 0$, with $\lambda$ the log-likelihood. So less Deviance is better. And `p_0,p_1` are the degrees of freedom “consumed” by each model.)

**Can you think of 2 obvious ways in which LRTs are rather limited as model-selection criteria?**

## Penalized Likelihood: AIC and BIC

Here are 2 problems with using LRTs.

1. **They are only relevant for nested models.**
2. The thresholds are not invariant to the choice of reference, and therefore the criterion can become ambiguous.

```
round(qchisq(0.95, df = 1:5), 1)
```

```
## [1]  3.8  6.0  7.8  9.5 11.1
```

The threshold for choosing between models changes nonlinearly as a function of the gap in d.f. between them – and also as a function of what model is the test's reference Null.

Two alternative and commonly used, but still Deviance-based, criteria are the **Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC)**. As the book details (in some length), their motivation is different, but they boil down to 2 very similar linear formulae:

$$AIC = D + 2p, \quad BIC = D + p\log n.$$

The AIC and BIC can both be calculated as invariant "scores", and compared directly across all models (nested and non-nested) for the same dataset and $y$. You choose the model with lowest AIC (or BIC).

- **AIC is more aggressive** (easier to add terms, more geared towards prediction), while
- **BIC is more conservative** (and indeed it is motivated by an inference perspective).

## Implementing In R: Stepwise with AIC/BIC

The book describes these well in Section 3.3. The idea is simple:

1. **Stepwise forward/backward:** add/remove one term at a time, based on its contribution to the existing model (gauged via AIC/BIC or a similar in-sample criterion).
2. **All-subset or best-subset:** apply the same principle, but try to explore a larger region of the combinatorial model space - or all of it.

The former works better with large $p$, the latter can be attempted with smaller $p$. **We will later see a middle way which requires a bit of work – but is very cost effective.**

There is a wealth of options for forward/backward/subset model selection in R with AIC-type criteria:

- `step` in the `stats` package does forward, backward, and bidirectional stepwise;
- `MASS::stepAIC` has similar functionality and supposedly more options;
- The `leap` package has the `regsubset` which allowed all the above, plus all-subset and *“leaps and bounds”* best-subset exploration (that’s the one that stuck my laptop:).
- The `bestglm` package (inspired by the Hastie *et al.* book!) has an omnibus function bearing the same name, promising multiple options for any GLM. But…it cannot handle multi-level categorical variables.
- Others probably exist…

Incidentally, `AIC` and `BIC` are both basic R functions (there is also `MASS::extractAIC`, somewhat more sophisticated).

## Forward/Backward: Let’s Do This

```r
autos = read.csv("../Datasets/autoMPGtrain.csv", as.is = T)
autos$gp100m = 100/autos$mpg
autos$continent = factor(autos$continent)
autos$diesel = grepl("diesel", autos$name)
# reference 'full models'
fullmod = lm(gp100m ~ ., data = autos[, -c(1, 9)])
fullWinteract = lm(gp100m ~ .^2, data = autos[, -c(1, 9)])
emptymod = lm(gp100m ~ 1, data = autos[, -c(1, 9)])

## I won't run this here, but note the syntax step(fullmod) # backward AIC
## step(fullmod,k=log(298)) # backward BIC Now forward BIC, allowing
## interactions:
## step(emptymod,list(upper=fullWinteract),direction='forward',k=log(298))
```

Play with this… try the all-subset function too… and questions???

## Introducing Cross-Validation

While AIC/BIC are more workable penalized-likelihood criteria than LRT, **they still originate in the inference world.** Besides, generally speaking the likelihood is wrong (cf. *“all models are wrong…”*).

**It makes more sense, if prediction is the goal - to use a prediction-related metric for model selection.**

We would like to know how well the models **trained** by the available observations, will predict **other** observations (a.k.a., **out-of-sample prediction**).

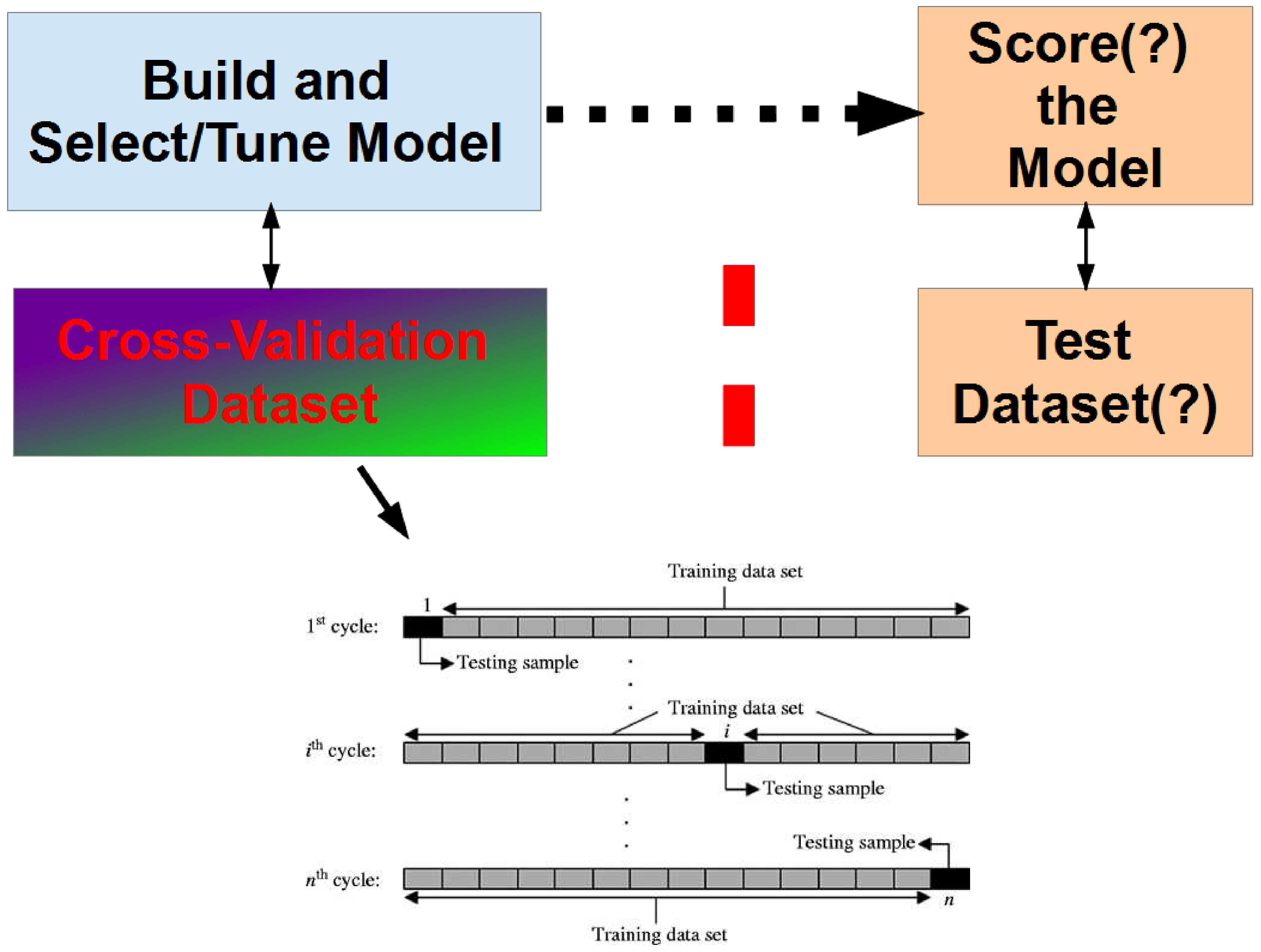
Cross-Validation (CV) separates our dataset into $K \leq n$ parts, then cycles through it $K$ times using the same model. At each cycle, $K - 1$ parts are used to train (=fit) the model, then the $K$-th part is predicted.

- The case of CV with $K = n$ is known as **leave-one-out (LOOCV).** It might sound over-the-top, but in many scenarios it is a very good option. In fact, with linear regression there is a closed-form solution based on the full-data regression, so there’s no need to actually run $n$ models (this solution is similar to the formulae used in `influence.measures`).
- More commonly, people use $K$ values between 5 and 10.
- If you dataset is grouped (e.g., by location), **the CV groups should respect this grouping.**

We collect all the prediction errors (according to some **loss function**, e.g., square-errors), and the average over the entire dataset is the model’s CV score.

**We choose the level of complexity whose model has the best CV score – or a somewhat less-complex model that has scored “close enough” to the top.**

## Prediction: The Training-Test Process (2-stage, with Cross-Validation)



# Common Cross-Validation Metrics

The same metrics we've encountered in estimation are used to score CV performance: MSE/RMSE, bias, etc.

- A robust alternative to RMSE is MAE, the **mean absolute error.**
- Even more robust: **Quantile Absolute Error (QAE)**. Symbolically, `quantile(abs(yhat-y),quantile=q)`. I like using the 75th percentile.

But interestingly, one of the most commonly quoted metric is **Prediction R-Squared.** Recall, in regression $R^2$ is the proportion of $y$'s variance that is accounted for by the model. It also happens to be the square correlation between predictions and observations:

$$\operatorname{cor}(\hat{y}, y)^2.$$

With CV and out-of-sample prediction, this identity is broken, so one needs to choose between the two definitions. (*Very*) sadly, almost everyone chooses the former, which now becomes

$$R^2_{pred.} = \max\left(0, 1 - \frac{MSE(\hat{y})}{Var[y]}\right).$$

It needs to be artificially baselined at zero, to match the scale of the in-sample $R^2$ (which cannot be negative). Note that **this metric provides almost no additional information beyond the MSE** – and is also woefully nonlinear and non-robust. Which is why I strongly prefer the correlation-based measure.

The correlation-based measure tells us **how well the $\hat{y}$s' relative order and distribution mimic those of the $y$s - in a scale indepenedent manner** (because correlation is not affected by bias).

There are other measures you might commonly find, e.g., mean/median *relative* error, etc.

## Implementing In R: All-Subset with CV

For many advanced model-selection and machine-learning methods (e.g., Lasso and most classification tools), CV is an integral part of the package. For others, you often have to do more work “by hand.” Specifically for regression, the lack of availability is quite surprising.

The `boot` (bootstrap) package provides CV for GLMs. **However, it does not enable us to keep the groups fixed between models - which is a must** (unless you do LOOCV). The `bestglm` package offers CV options - but as said, only if you don’t have categorical covariates *(or are willing to split them into individual terms; bad idea)*.

Anyway… it so happens that at my former UW job we really needed that. So we created these functions, modified versions of which are on the class website.

```
source("../Code/cvEngine.r")
source("../Code/cvConstrainedSubsets.r")
cvid = sample(1:5, size = 298, replace = TRUE)  ## K=5 CV groups

# Not run here:
# firstcut=cvConSubsets(autos,'gp100m',cov.list=names(autos)[c(2:8,11)],cv.group=cvid)
```

# Implementing In R: All-Subset with CV

We already **know** `weight` and `year` are a shoo-in for the best model. **So let's `forced` them in, and also explore their interaction:** (note: *not* a good idea to add an interaction term, unless both individual covariates are `forced`)

```
mynames = c(names(autos)[c(2:8, 11)], "weight:year")
#
# firstcut=cvConSubsets(autos,'gp100m',cov.list=mynames,cv.group=cvid,forced=c(4,6))
```

A different tweak: maybe `year` is nonlinear? Try a spline. But we don't want to have both the linear `year` and its spline term together in the model. **Solution: we can make them `avoid` each other.**

```
spnames = c(mynames[-9], "ns(year,df=3)")
#
# firstcut=cvConSubsets(autos,'gp100m',cov.list=spnames,cv.group=cvid,forced=c(4,6))
```

Finally: I transformed `mpg` to `gp100m=100/mpg`, because consumption is more linear in the predictors. But don't we want to calculate and evaluate prediction errors on the **original** scale?

```
spnames = c(mynames[-9], "ns(year,df=3)")
#
# firstcut=cvConSubsets(autos,'gp100m',cov.list=spnames,cv.group=cvid,forced=c(4,6),transfun=function(x)
# 100/x,transform.list=list(mean=0,sd=1))
```

## R Cool Trick: `grep`, `grepl`, and a Function from `cvConSubsets`...

Did you notice this?

```
grep("diesel", autos$name)
```

```
## [1] 182 240 241 269 270 288
```

```
table(grepl("diesel", autos$name))
```

```
##
## FALSE  TRUE
##   292     6
```

Now, try the function `combn` used in `cvConSubsets`. What does it do?

```
combn(names(autos)[2:8], 2)
```

```
##      [,1]     [,2]  [,3]     [,4]    [,5]   [,6]        [,7]     [,8]
## [1,] "cyl"    "cyl" "cyl"    "cyl"   "cyl"  "cyl"       "volume" "volume"
## [2,] "volume" "hp"  "weight" "accel" "year" "continent" "hp"     "weight"
##      [,9]     [,10]    [,11]       [,12]    [,13]   [,14]  [,15]
## [1,] "volume" "volume" "volume"    "hp"     "hp"    "hp"   "hp"
## [2,] "accel"  "year"   "continent" "weight" "accel" "year" "continent"
##      [,16]    [,17]    [,18]       [,19]   [,20]       [,21]
## [1,] "weight" "weight" "weight"    "accel" "accel"     "year"
## [2,] "accel"  "year"   "continent" "year"  "continent" "continent"
```

# R Annoyance: Matrices Perceived as Vectors

Let us follow step-by-step, the way the `forced` argument works, for two-covariate models.

```
zz = combn(names(autos)[2:8], 2)
zz = zz[, apply(zz, 2, function(x, b) b %in% x, b = "weight")]
zz = zz[, apply(zz, 2, function(x, b) b %in% x, b = "year")]
zz
```

```
## [1] "weight" "year"
```

```
dim(zz)
```

```
## NULL
```

**Once a matrix reduces to 1 row or 1 column, R usually auto-converts it to a vector. In the best case, an error will be generated and you will understand where it came from. In the worst case, you won't notice it and something bad will happen down the road.**

The workaround is straightforward: **handle the exception** by identifying it and recasting the vector as a 1-column or 1-row matrix, as intended. (look up the code of `cvConSubsets.r` to see how it was done in this case).

## R Cool Trick: Some Stuff I Learned This Week

Finally, suppose you have a time series of categorical/integer outcomes, and you want to know the distribution of **streaks** or ‘runs’ of the same value repeated.

Is there some obscure function in an exotic R package that does it?

```
tosses = sample(c("Heads", "Tails"), size = 20, replace = TRUE)
tosses
```

```
##  [1] "Tails" "Heads" "Heads" "Heads" "Heads" "Heads" "Tails" "Tails"
##  [9] "Tails" "Heads" "Tails" "Tails" "Heads" "Heads" "Tails" "Tails"
## [17] "Tails" "Heads" "Tails" "Heads"
```

```
runs = rle(tosses)
runs$values
```

```
##  [1] "Tails" "Heads" "Tails" "Heads" "Tails" "Heads" "Tails" "Heads"
##  [9] "Tails" "Heads"
```

```
runs$lengths
```

```
##  [1] 1 5 3 1 2 2 3 1 1 1
```