Lecture 7: Statistical modeling

An introduction to statistical modeling

Yujin Jeong

STATS 195

Statistical Modeling

Introduction to models

In 1976, a British statistician named George Box wrote the famous line,

"All models are wrong, some are useful."

- The goal of a model is to provide a simple low-dimensional summary of a dataset. Ideally, the model will capture true "signals" (i.e. patterns generated by the phenomenon of interest), and ignore "noise" (i.e. random variation that you're not interested in).

Hypothesis generation vs. confirmation

Traditionally, the focus of modelling is on inference, or for confirming that an hypothesis is true. There is a pair of ideas that you must understand in order to do inference correctly:

- Each observation can either be used for exploration or confirmation, not both.
- You can use an observation as many times as you like for exploration, but you can only use it once for confirmation.

This is necessary because to confirm a hypothesis you must use data independent of the data that you used to generate the hypothesis. Otherwise you will be over optimistic.

Confirmatory analysis

If you plan to do an confirmatory analysis, one approach is to split your data into three pieces before you begin the analysis:

- Training set: the bulk (e.g. 50%) of the data set that can be used to do anything: visualizing, fitting multiple models.
- Validation set: a smaller subset (e.g. 20%) of the data set that is used to compare different models and choose the best one.
- Test set: a held back data set used only ONCE to test and assess your final model.

Linear Regression

Linear Regression

Regression is a supervised learning method, whose goal is inferring the relationship between input data, x, and a **continuous** response varaible, y. Linear regression is a type of regression where y is modeled as a linear function of x.

- Given a data set $\{y_i,\ x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ of n statistical units, the model takes the form:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i, \quad i = 1, \ldots, n$$

- Fitting a linear model to a given data set usually requires estimating the regression coefficients β such that the residual sum of squares (RSS) below is minimized.

$$||\epsilon||_2^2 = \sum_i (y_i - x_i^T \hat{\beta})^2$$

Linear Regression

Here is a toy function that generates random normal data (X) and then creates Y which is a function of X plus some noise.

```
genData<-function(n,p,Sig=diag(p),beta,sig=1){
  X<-MASS::mvrnorm(n,mu=rep(0,p),Sigma=Sig) #generating X
  if(length(beta)<p){ #padding beta with zeros if given fewer coefficients than p
    d<-p-length(beta)
    beta<-c(beta,rep(0,d))
  }
  eps<-rnorm(n,0,sig) #noise
  Y<-X%*%beta+eps #creating Y from X, beta, and noise
  res<-data.frame(cbind(X,Y)) #combining X and Y
  names(res)[(p+1)]<-"Y" #renaming Y
  return(as_tibble(res)) #returning as a tibble
}
```

- n: the number of data points we want to generate
- p: the number of features
- beta: true beta in a linear model
- sig: the standard deviation of independent noise (ϵ)

Linear Regression

We can generate some data to play with:

```
set.seed(810)
tb1<-genData(100,3,beta=c(2,3),sig=2)
```

In this case our data should look like:

$$Y = 2X_1 + 3X_2 + 0X_3 + \epsilon$$

with X_1, X_2, X_3 independent standard normals and $\epsilon \sim N(0, 2^2)$.

Linear Regression

In R, the linear models can be fit with a `lm()` function.

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Should be `NULL` or a numeric vector. If non-NULL, weighted least squares is used with weights `weights` (that is, minimizing `sum(w*e^2)`); otherwise ordinary least squares is used. See also 'Details',

Linear Regression

The input for `formula` is an object of class `formula`. To learn more about formulas, this reference sheet is the best resource. Let's practice the syntax:

```
lm1 <- lm(Y~., data=tb1)
lm2 <- lm(Y~X1+X2+X3, data=tb1)
lm3 <- lm(Y~.-X3, data=tb1)
lm4 <- lm(Y~X1+X2, data=tb1)
lm5<-lm(Y~X1,data=tb1)
```

```
> lm1

Call:
lm(formula = Y ~ ., data = tb1)

Coefficients:
(Intercept)           X1           X2           X3
    0.27676      2.11415      3.46535     -0.05145
```

Linear Regression

Let's check the details on the fitted model:

```
summary(lm1)
```

```
Call:
lm(formula = Y ~ ., data = tb1)

Residuals:
    Min      1Q  Median      3Q     Max
-4.1764 -1.2844  0.1808  1.2439  5.0091

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.27676    0.20777   1.332    0.186
X1           2.11415    0.20092  10.522   <2e-16 ***
X2           3.46535    0.23013  15.058   <2e-16 ***
X3          -0.05145    0.19626  -0.262    0.794
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.057 on 96 degrees of freedom
Multiple R-squared:  0.7873,  Adjusted R-squared:  0.7806
F-statistic: 118.4 on 3 and 96 DF,  p-value: < 2.2e-16
```

Linear Regression

We can compute the fitted values $\hat{y}$ a.k.a. the predicted y values for existing observations.

```
predict(lm1) #fitted values
```

Linear Regression

We can also use the predict function to make predictions for new data that wasn't used to train the model

```
set.seed(90)
testdata <- genData(30, 3, beta=c(2,3), sig=2)
yhat1 <- predict(lm1, newdata=select(testdata,-Y))
yhat3 <- predict(lm3, newdata=select(testdata,-Y))
yhat5 <- predict(lm5, newdata=select(testdata,-Y))
```

We can further quantify the error. One commonly used standard is the mean square error (MSE) or root mean square error (RMSE). Below is the MSE for our three models.

```
mean(sum((yhat1-testdata$Y)^2))
mean(sum((yhat3-testdata$Y)^2))
mean(sum((yhat5-testdata$Y)^2))
```

```
## [1] 155.2934
## [1] 156.6276
## [1] 418.329
```

Lasso Regression

Lasso Regression

Lasso regression is a regression with L_1 penalty. That is, it solves the problem:

$$\hat{\beta} = \arg\min_{\beta} \sum_i (y_i - x_i^T \beta)^2 + \lambda ||\beta||_1.$$

- It turns out that the penalization on the L_1 norm $||\beta||_1$ promotes sparsity - only a handful of $\hat{\beta}_j$ will actually be non-zero.
- The number of non-zero coefficients depends on the choice of the tuning parameter λ. The higher the λ, the fewer non-zero coefficients.

Why LASSO? Sparse data

Let's generate a following sparse data set where X_1, X_2 are related to Y but $X_3, \ldots, X_{50}$ are irrelevant.

```
set.seed(813)
spar.train<-genData(1000,50,beta=c(2,3),sig=5)
spar.test<-genData(100,50,beta=c(2,3),sig=5)
```

Let's fit a big linear model and the true model and see how we do:

```
lm.spar1 <- lm(Y~.,data=spar.train)
lm.spar2 <- lm(Y~X1+X2,data=spar.train)
summary(lm.spar1)
summary(lm.spar2)
```

Why LASSO? Sparse data

How do our different models do on this new data?

```
yhat.spar1 <- predict(lm.spar1,newdata=select(spar.test,-Y))
yhat.spar2 <- predict(lm.spar2,newdata=select(spar.test,-Y))
```

```
mean(sum((yhat.spar1-spar.test$Y)^2))
mean(sum((yhat.spar2-spar.test$Y)^2))
```

```
## [1] 2792.527
## [1] 2543.195
```

In the real world, we wouldn't know which features are relevant. There are many models that work better than a naive linear regression to help with feature selection.

One of the best (especially when the true model is sparse) is called LASSO.

Lasso Regression

Lasso regression is implemented in a R package `glmnet`. You will notice the syntax is different from `lm`.

```
#install.packages("glmnet")
library(glmnet)
lasso.spar <- cv.glmnet(x=as.matrix(select(spar.train,-Y)), y=spar.train$Y, alpha=1) #alpha=1 gives us LASSO
coef(lasso.spar)
```

```
51 x 1 sparse Matrix of class "dgCMatrix"
                    s1
(Intercept) 0.06758913
X1          1.60383916
X2          2.33118203
X3          .
X4          .
X5          .
X6          .
X7          .
X8          .
X9          .
X10         .
X11         .
...
```

Lasso Regression

We can see that our LASSO model figured out that only X1 and X2 were important on its own! You can also see that it actually got slightly different coefficients than our smaller model did (it shrunk the coefficients). In this case, the LASSO model is actually closer to the truth, and our test error is actually slightly better:

```
yhat.spar3 <- predict(lasso.spar,newx=as.matrix(select(spar.test,-Y))) #newx instead of newdata
mean(sum((yhat.spar1-spar.test$Y)^2))
mean(sum((yhat.spar2-spar.test$Y)^2))
mean(sum((yhat.spar3-spar.test$Y)^2))
```

```
## [1] 2792.527
## [1] 2543.195
## [1] 2423.058
```

Credit Data Example

Credit data example

A credit data in ISLR package is a simulated data set containing information on ten thousand customers. The aim here is to predict which customers will default on their credit card debt.

```
library(ISLR)
lmcredit <- lm(Balance~., data=Credit)
```

Let's first divide the credit data set into a training set, a validation set, and a test set.

```
n = nrow(Credit)
s = sample(1:n, size = n, replace = FALSE)
credit_train = Credit[s %% 3 == 0, ]
credit_val = Credit[s %% 3 == 1, ]
credit_test = Credit[s %% 3 == 2, ]
```

STEP 1: Fit models on the training set

Let’s pick 3-4 models.

```
model1 <- # FILL IN
model2 <- # FILL IN
model3 <- # FILL IN
model4 <- # FILL IN
```

STEP 2: Choose the best model using the validation set

Let's compare different models on the validation data set.

```
yhat1 <- # FILL IN
yhat2 <- # FILL IN
yhat3 <- # FILL IN
yhat4 <- # FILL IN
```

```
mean(sum((yhat1-credit_val$Balance)^2))
mean(sum((yhat2-credit_val$Balance)^2))
mean(sum((yhat3-credit_val$Balance)^2))
mean(sum((yhat4-credit_val$Balance)^2))
```

Which model gives the smallest MSE?

STEP 3: Evaluate the final model on the test set

Report the final model and evaluate it on the test data set.

For example,

```
model <- lm(Balance ~ Income + Limit + Student, data = rbind(credit_train, credit_val))
model
```

```
Call:
lm(formula = Balance ~ Income + Limit + Student, data = rbind(credit_train,
    credit_val))

Coefficients:
(Intercept)       Income        Limit   StudentYes
  -440.3858      -7.9513       0.2689     425.2858
```

```
yhat_test <- predict(model, newdata=select(credit_test,-Balance))
mean(sum((yhat_test-credit_test$Balance)^2))
```

```
## [1] 1497515
```

More on models

Other methods

Go take stats 191, 202, 216 or equivalent to learn:

- stats::glm() #generlaized linear regression
- MASS::lda() #LDA and QDA
- e1071::svm() #support vector machines
- e1071::gknn() #k nearest neighbors
- e1071::naiveBayes() #naive bayes
- kernlab::ksvm() #support vector machines
- randomForest::randomForest() #randomforest
- xgboost::xgboost() #xgboost
- nnet, neuralnet, RSNNS, h2o #neural networks (you should use python)