3 PQHS 471 Notes Week 3

3.1 Week 3 Day 1

3.1.1 HOML 2 review

3.1.2 ISLR 4.1–4.2: Overview of classification

ISLR Chapter 4 covers some traditional statistical approaches to classification problems: logistic regression and discriminant analysis (linear or quadratic). Modern machine learning approaches will be covered in future lectures.

3.1.3 ISLR 4.3: Logistic regression models

Let p be the probability of an event. The **odds** of the event is $\frac{p}{1-p}$, and the **log-odds** of the event is $\log(\frac{p}{1-p})$. The function $\log(\frac{p}{1-p})$ is called the **logit** function. The **odds-ratio** (OR) for the event between Condition 1 and Condition 2 is $\frac{p_1/(1-p_1)}{p_2/(1-p_2)}$, where p_k is the probability of the event under Condition k. An OR is always *between two conditions*. If the two conditions are defined by a binary variable, say sex, with its coding well specified (say 0=F, 1=M), one may say "the OR for sex". If the coding is unclear, "the OR for sex" is ambiguous because it could be the OR for man vs. woman or the OR for woman vs man; if the former is 2 the latter is 0.5. For a variable with more than two categories (say, age), a phrase like "the OR for age" does not make sense (unless there is a context, say, linear effect is assumed).

In **logistic regression**, we model the outcome probability (not the outcome itself) as a function of predictors:

$$p = \sigma(\beta_0 + \beta_1 X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}, \tag{4.1}$$

$$\log(\frac{p}{1-p}) = \beta_0 + \beta_1 X. \tag{4.2}$$

Here $\sigma(t) = \frac{e^t}{1+e^t}$ is called the *sigmoid function* in neural networks. It is often used as an activation function. The model is fit with the **maximum likelihood** approach to obtain the MLEs for the coefficients. Logistic regression models are not classification models per se, although they can be used to derive classifiers.

Interpretation of coefficients: In (4.2), β_1 is the log-OR for every unit increase in X, and e^{β_1} is the OR for every unit increase in X. The CI for β_1 is often symmetric, while the CI for the OR, e^{β_1}, is often asymmetric.

```
library(ISLR)
mod = glm(default ~ ., data=Default, family=binomial)
options(scipen=999, digits=3)  ## set output options
cbind(mod$coef, confint(mod))  ## log-OR scale
cbind(exp(mod$coef), exp(confint(mod)))  ## OR scale
```

Consider **an additive model** between two binary predictors, sex (0=F, 1=M) and smoking status (0=NS, 1=S),

$$\begin{aligned}\log(\frac{p}{1-p}) &= \beta_0 + \beta_1 \cdot \text{sex} + \beta_2 \cdot \text{smoking} \\ &= \begin{cases} \beta_0 & \text{if female non-smoker} \\ \beta_0 + \beta_1 & \text{if male non-smoker} \\ \beta_0 + \beta_2 & \text{female smoker} \\ \beta_0 + \beta_1 + \beta_2 & \text{male smoker} \end{cases}\end{aligned}$$

- e^{β_0} is the odds for female non-smokers;
- e^{β_1} is the OR for sex (because it is the OR for male vs. female among non-smokers and also among smokers);
- e^{β_2} is the OR for smoking (because it is the OR for smoking vs. non-smoking among females and also among males).

Now consider **an interaction model** between sex and smoking,

$$\log(\frac{p}{1-p}) = \beta_0 + \beta_1 \cdot \text{sex} + \beta_2 \cdot \text{smoking} + \beta_3 \cdot \text{sex} \cdot \text{smoking}$$
$$= \begin{cases} \beta_0 & \text{if female non-smoker} \\ \beta_0 + \beta_1 & \text{if male non-smoker} \\ \beta_0 + \beta_2 & \text{if female smoker} \\ \beta_0 + \beta_1 + \beta_2 + \beta_3 & \text{if male smoker} \end{cases}$$

- e^{β_0} is the odds for female non-smokers.
- e^{β_1} is the OR for male non-smokers vs. female non-smokers (i.e., the OR for sex among non-smokers). The OR for sex among smokers is $e^{\beta_1+\beta_3}$.
- e^{β_2} is the OR for female smokers vs. female non-smokers (i.e., the OR for smoking among females). The OR for smoking among males is $e^{\beta_2+\beta_3}$.
- e^{β_3} is a ratio of ORs: (OR for sex among smokers)/(OR for sex among non-smokers), or equivalently, (OR for smoking among males)/(OR for smoking among females).

In this interaction model, (1) the "main-effects" β_1 and β_2 do not reflect the overall effects of sex and smoking; (2) β_0, β_1, and β_3 are not comparable because they are not the same concepts. So, it is meaningless to make statements like $\beta_1 = \beta_3$ or "the interaction effect is stronger than the main effect".

Collinearity of the predictors can make coefficients change direction. That is, the effect of x_1 becomes opposite (and still significant!) after x_2 is added to the model. An example is below:

```
library(ISLR)
with(Default, {  ## display data, Fig 4.1 left panel
    plot(balance, income)
    points(balance[default=="Yes"], income[default=="Yes"], col=2, pch=19)
    })

summary(glm(default ~ balance, family=binomial, data=Default))$coef
summary(glm(default ~ student, family=binomial, data=Default))$coef  ## student has coef >0
summary(glm(default ~ balance + student, family=binomial, data=Default))$coef  ## student has coef <0
```

This is similar to **Simpson's paradox**, where X and Y may appear to be associated but the association is gone once Z is taken into account; that is, the "effect" of X on Y becomes zero after Z is added to the model. One example is the Berkeley admission data in 1973, in which every department admitted approximately equal fractions of male and female applicants, but the overall tally across departments clearly showed a significantly higher admission rate for males than for females. The reason is that the departments with a high admission rate often had a higher proportion of male applicants than females, and the departments with a low admission rate often had a higher proportion of female applicants. Here, X is the sex of an applicant, Y is the person's admission status, and Z is the department the person applied for. Another example is in genetics: two genetic markers can be in linkage equilibrium (i.e., no association) in all subpopulations but the overall tally can appear to show linkage disequilibrium (i.e., association). Example 3: One could sample people from San Francisco and show that certain genetic markers are associated with using chopsticks.

3.1.4 Assignment

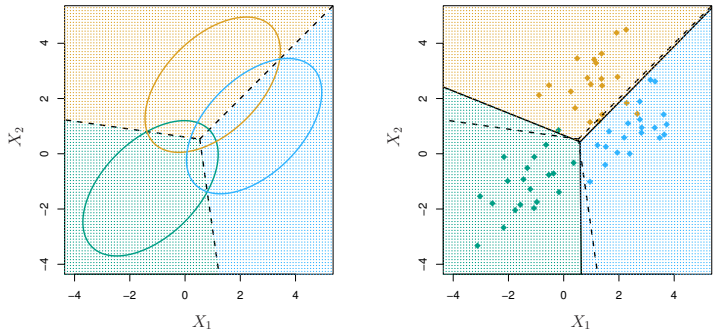
1. Reading for next lecture: ISLR 4.4–4.5; HOML Chapter 3.
2. ISLR Chapter 4 R Labs

3.2 Week 3 Day 2

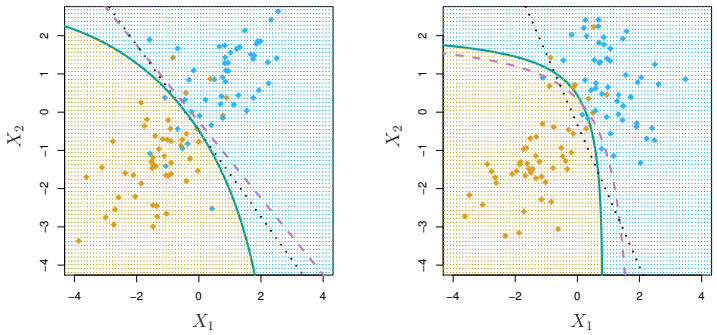
3.2.1 ISLR 4.4

This section explains LDA/QDA, the confusion matrix, and ROC. My notes are in separate files: LDA-QDA.pdf and ConfusionMatrix-ROC.pdf.

LDA: There are K discriminant functions that are linear in x. (4.13) (4.17) for 1-dimensional X, (4.19) for multi-dimensional X. These functions determine linear boundaries.



QDA: The discriminant functions (4.23) are quadratic in x.



Both LDA and QDA assume multivariate normality, a very strong assumption. LDA also assumes equal variance.

LDA has a similar setup as (equal-variance) t-test/Hotelling's T^2-test, but with a different purpose and with the roles of X and Y switched. Similarly, QDA is similar to unequal-variance t-test/Hotelling's test.

Confusion matrix: Tables 4.6, 4.7. https://en.wikipedia.org/wiki/Sensitivity_and_specificity

The `confusionMatrix()` function in the R `caret` package displays columns as truth and rows as predicted values, same as the layout in the above wikipedia page. The `sklearn.metrics.confusion_matrix()` in Python is the same as `table()` in R; their display depends on the order of the two input variables.

```
library(MASS)  ## for lda(), qda()
library(ISLR)  ## for Default dataset
library(caret) ## for confusionMatrix()

## LDA
lda1 = lda(default ~ balance + income, data=Default)
names(lda1)
## by default the prior is the class distribution
lda1$prior; table(Default$default)
## compare observed with predicted
table(Default$default, predict(lda1)$class)
## confusion matrix
confusionMatrix(predict(lda1)$class, Default$default)  ## wrong
confusionMatrix(predict(lda1)$class, Default$default, positive="Yes")
confusionMatrix(predict(lda1)$class, Default$default, positive="Yes", mode="everything")

## QDA
qda1 = qda(default ~ balance + income, data=Default)
```

```
qda1$prior; table(Default$default)
confusionMatrix(predict(qda1)$class, Default$default, positive="Yes", mode="everything")
```

Example: QDA boundary with a single predictor:

```
mu1=0; s1=2; pi1=0.5  ## true distribution for class 1 (red)
mu2=3; s2=1; pi2=0.5  ## true distribution for class 2 (green)
curve(pi1*dnorm(x, mu1, s1) +pi2*dnorm(x, mu2, s2), -4, 7, ylim=c(0,0.3))
curve(pi1*dnorm(x, mu1, s1), col=2, add=T)
curve(pi2*dnorm(x, mu2, s2), col=3, add=T)

## There are 2 threshold values for the QDA !?
pred1 = function(x) ((x-mu1)/s1)^2-2*log(pi1)+log(s1^2) <((x-mu2)/s2)^2-2*log(pi2)+log(s2^2)
curve(0.3*pred1(x), n=1001, lty=2, add=T)

## Enlarge the far right portion to see why.  Red crosses above green there!
curve(pi1*dnorm(x, mu1, s1) +pi2*dnorm(x, mu2, s2), 6, 8)
curve(pi1*dnorm(x, mu1, s1), col=2, add=T)
curve(pi2*dnorm(x, mu2, s2), col=3, add=T)
```

Example: QDA boundary with two predictors:

```
mu1=c(0,0); s1=1.2; pi1=0.5  ## true distribution for class 1
mu2=c(3,2); s2=1; pi2=0.5    ## true distribution for class 2
grid=seq(-3, 6, .01)
x1grid = rep(grid, length(grid))
x2grid = rep(grid, each=length(grid))
d1 = pi1 * dnorm(x1grid, mu1[1], s1) * dnorm(x2grid, mu1[2], s1)
d2 = pi2 * dnorm(x1grid, mu2[1], s2) * dnorm(x2grid, mu2[2], s2)
library(rgl)
plot3d(x1grid, x2grid, d1, col='lightgrey')  ## class 1 distribution
plot3d(x1grid, x2grid, d2, col='grey', add=T) ## class 2 distribution
```

Means: arithmetic, geometric, harmonic

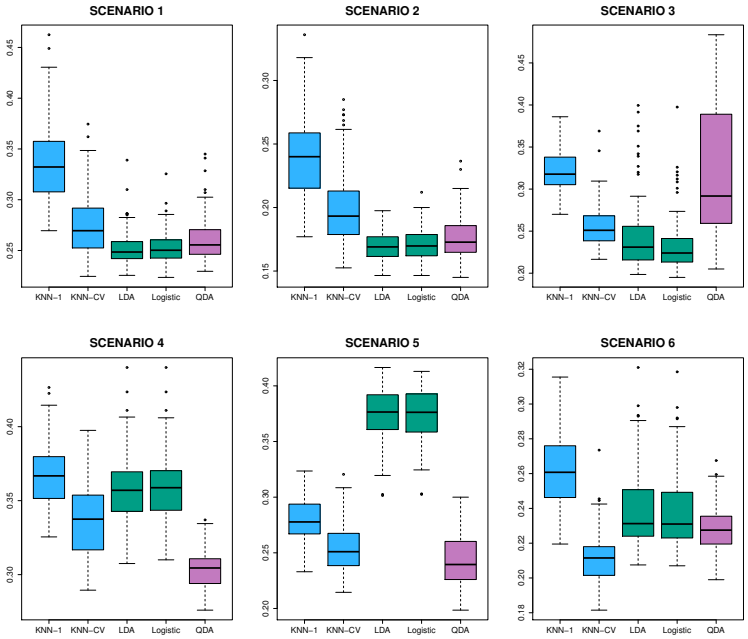
```
grid = seq(0.1, 10, .1)
x=rep(grid, length(grid))
y=rep(grid, each=length(grid))
am = 0.5*(x+y)
gm = sqrt(x*y)
hm = 2*x*y/(x+y)
library(rgl)
plot3d(x,y,am)
plot3d(x,y,gm, col=2, add=T)
plot3d(x,y,hm, col=3, add=T)
```

3.2.2 ISLR 4.5 Comparisons of LDA/QDA/logistic/KNN

For the KNN methods, two versions were evaluated: 1-NN and k-NN with k chosen from cross-validation. Consider a binary outcome with $\mu_1 \neq \mu_2$, and 2 predictors. Consider 6 scenarios, each with 100 replications:

1. $\binom{x_1}{x_2} \sim N\left(\mu_k, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)\right)$; 20 obs in each class. LDA is the best, followed closedly by logistic regression.
2. $\binom{x_1}{x_2} \sim N\left(\mu_k, \left(\begin{smallmatrix}1 & -0.5\\ -0.5 & 1\end{smallmatrix}\right)\right)$; 20 obs in each class. LDA is the best, followed closedly by logistic regression.
3. $x_1 \sim t_d$ and $x_2 \sim t_d$ are independent; 50 obs in each class. The set-up violated the assumptions of LDA. Logistic regression is the best, followed closedly by LDA. The QDA results deteriorated considerably as a consequence of non-normality.

4. For class 1, $\binom{x_1}{x_2} \sim N\left(\mu_1, \left(\begin{smallmatrix} 1 & 0.5 \\ 0.5 & 1 \end{smallmatrix}\right)\right)$; for class 2, $\binom{x_1}{x_2} \sim N\left(\mu_2, \left(\begin{smallmatrix} 1 & -0.5 \\ -0.5 & 1 \end{smallmatrix}\right)\right)$. QDA is the best, followed by kNN-CV.
5. $\binom{x_1}{x_2} \sim N\left(\mu_k, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)\right)$. Then $p(y = 1|x_1, x_2) = \frac{e^{\beta_1 x_1^2 + \beta_2 x_2^2 + \beta_3 x_1 x_2}}{1 + e^{\beta_1 x_1^2 + \beta_2 x_2^2 + \beta_3 x_1 x_2}}$. QDA is the best, followed closedly by kNN-CV. The linear methods had very poor performance.
6. Same as 5, but with the responses sampled from a more complicated nonlinear function. kNN-CV wins, but 1-NN gave the worst results. This shows that "even when the data exhibits a complex nonlinear relationship, a nonparametric method such as KNN can still give poor results if the level of smoothness is not chosen correctly."



3.2.3 HOML 3

Some good points:

- Precision–recall tradeoff.
- Sensitivity–specificity tradeoff.
- Misclassification error rate can be a misleading criterion when there is strong **class imbalance**.

The class `sklearn.linear_model.SGDClassifier` is a linear model with SGD learning (i.e., fitted with the SGD algorithm). The default is hinge loss with l_2 penalty, which means SVM. Default 5 epochs and mini-batch size 1?

3.2.3.1 The MNIST data

The MNIST dataset has been extensively used as a test bed for machine learning methods. Because the same test set has been used, new methods claiming better performance is probably in the territory of overfitting. Nonetheless, it is still a very good starting point for trying machine learning methods.

The MNIST dataset contains 70,000 handwritten digits in 28×28 resolution in greyscale. See http://yann.lecun.com/exdb/mnist/ for the history and performance of various methods. There is some imbalance in the outcome categories. The digit "1" is the most frequent, 7877; "7" is the 2nd most frequent, 7293. The digit "5" is least frequent, 6313; "4" and "8" are 2nd/3rd least frequent, 6824/6825.

The data have been packaged in a few ways, either as a single dataset, as training/test, or as training/validation/test. The files are often python friendly. HOML uses the Matlab version from mldata.org: http://mldata.org/repository/data/download/matlab/mnist-original/. Scikit-Learn provides a function `fetch_mldata` to download and save it in $HOME/scikit_learn_data/mldata/. This "MNIST original" version, mnist-original.mat, comes as a single dataset

with no pre-splitting. Its `data` part is a 2-D array with dimensions 784×70000. Note that $784 = 28 \times 28$. The 784 pixels are ordered so that the first 28 pixels are the first row (from left to right), the next 28 pixels are the second row, etc. This is called *row-major order*, which is used by Python's numpy (also by C). R uses *column-major order* (same as Fortran, Matlab, Julia).

The pixel intensity is an integer between 0 and 255. `reshape(28,28)` makes it ready to view with `imshow`. Scikit-Learn's `fetch_mldata` by default transposes data array and its output is 70000×28.

```
import numpy as np
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
mnist   ## like a python dictionary (a Perl hash table, or an R list)
X, y = mnist["data"], mnist["target"]  ## X and y are numpy arrays
print(X.shape)  ## (70000, 784)
print(y.shape)  ## (70000,)
print(np.unique(y, return_counts=True))  ## tally the counts

some_digit_image = X[36000].reshape(28, 28)  ## reshape(nrow, ncol)
import matplotlib
import matplotlib.pyplot as plt
plt.imshow(some_digit_image, cmap = matplotlib.cm.binary, interpolation="none")
plt.axis("off")
```

To load the data into R, one can use `R.matlab`, which does not require python installed in your computer, or `reticulate`, which runs python behind R. My code using `R.matlab`:

```
library(R.matlab)
aa = readMat("/home/lic3/scikit_learn_data/mldata/mnist-original.mat")
str(aa)
names(aa); attributes(aa)
X = aa$data
y = aa$label
class(X); dim(X)  ## (784, 70000)
class(y); dim(y)  ## (1, 70000)
```

My code using `reticulate`:

```
library(reticulate)
#use_condaenv("r-tensorflow")  ## if you have installed keras in a conda environment
scipy <- import("scipy")
aa = scipy$io$loadmat('/home/lic3/scikit_learn_data/mldata/mnist-original.mat')
str(aa)

X = aa$data
y = aa$label
class(X); dim(X)  ## 784, 70000
class(y); dim(y)  ## 1, 70000
```

Another version, http://deeplearning.net/data/mnist/mnist.pkl.gz, contains three subsets for training/validation/test. Nielsen used this version in his online book. In this version, the pixel intensity values are rescaled values between 0 and 1. To load into Python:

```
import pickle, gzip
with gzip.open('mnist.pkl.gz', 'rb') as f:
    (x_train, y_train), (x_valid, y_valid), (x_test, y_test) = pickle.load(f, encoding="latin1")
```

We can use `reticulate` to load it into R:

```
library(reticulate)
library(zeallot)
```

```
my_py_load_object <- function(filename) {
  pickle <- import("pickle")
  gzip <- import("gzip")
  handle <- gzip$open(filename, "rb")
  on.exit(handle$close(), add = TRUE)
  pickle$load(handle, encoding="latin1")  ## to read cPickle format
}

mypklfile = '/home/lic3/Work/Teaching/Nielsen/neural-networks-and-deep-learning/data/mnist.pkl.gz'
aa = my_py_load_object(mypklfile)
str(aa)
c(c(x_train2, y_train2), c(x_valid2, y_valid2), c(x_test2, y_test2)) %<-% aa
```

There is a third version, https://s3.amazonaws.com/img-datasets/mnist.npz. The R `keras` package provides `dataset_mnist()` to download and save it to $HOME/.keras/datasets/mnist.npz. This version contains two subsets: `train` and `test`. Their `x` parts are 3-D arrays with dimensions $60000 \times 28 \times 28$ and $10000 \times 28 \times 28$.

3.2.3.2 Beyond a single binary outcome

The Scikit-Learn document http://scikit-learn.org/stable/modules/multiclass.html shows all the `sklearn` methods for various classification scenarios.

Multiclass/multinomial classification with $K > 2$ classes:

1. Some "**inherently multiclass**" methods are applicable for any number of classes (e.g., random forests, naive Bayes, KNN)
2. Binary classification methods can be employed in the following ways:
   - **One-versus-all**: Build K binary prediction models, one for each class versus the rest, and select the class with the highest decision score (assuming the scores are comparable across the K models).
   - **One-versus-one**: Build $K(K-1)/2$ binary prediction models, one for a pair of classes, and select the most frequently predicted class.

In sklearn, when `SGDClassifier` is used for multiclass classification, one-versus-all is performed. To perform one-versus-one with `SGDClassifier`, one needs `sklearn.multiclass.OneVsOneClassifier`.

Scaling features can often improve the performance of `SGDClassifier`.

Multilabel classification with multiple *binary* outcome variables. The example in HOML Chapter 3 has two binary outcomes, $(I(y \geq 7), y \bmod 2)$. The outcomes are put together as an $n \times 2$ array, which is then provided to a KNN model as the `y` in `.fit(X,y)`. The KNN model has class `sklearn.neighbors.KNeighborsClassifier`; by default it is a 5-NN. Internally a separate KNN model is fit for each outcome variable.

Multioutput classification with multiple *non-binary* outcome variables. The example in HOML Chapter 3 has 784 outcomes each with value range $[0, 255]$. 5-NN `KNeighborsClassifier` is used to build the model; internally 784 KNN models are fit. I wonder if `KNeighborsRegressor` followed by rounding would be better because the outcomes are quantitative by nature.

3.2.4 Assignment

1. **Homework**: ISLR Chapter 4 Exercises 13
2. Reading for next lecture: ISLR 5.
3. ISLR Chapter 5 R Labs