MTH 3270 Notes 8

7 Statistical Learning and Predictive Analytics (Cont'd) (8)

7.3 Classifiers (Cont'd)

7.3.1 Naive Bayes

- The ***naive Bayes*** classifier predicts for an individual the class that the individual has the **highest probability** of belonging to, based on its values $X_1, X_2, \ldots, X_p$ of the explanatory variables.

  To estimate the (conditional) **probability** of each class, given the observed values of $X_1, X_2, \ldots, X_p$, it invokes ***Bayes' Rule*** (from MTH 3210 Probability and Statistics).

  For more details, see the textbook.

7.3.2 Artificial Neural Networks

- An ***artificial neural network*** can be thought of as a generalization of multiple regression for predicting a numerical response variable Y from explanatory variables $X_1, X_2, \ldots, X_p$.

  (The method can also be used for *classification*, i.e. for predicting a *categorical* response variable.)

  The idea is to **"derive" new** explanatory variables $\boldsymbol{H_1}, \boldsymbol{H_2}, \ldots, \boldsymbol{H_m}$ from the original X's via a (non-linear) function $\boldsymbol{g}$,

$$H_j \;=\; g(\hat{\alpha}_{j0} + \hat{\alpha}_{j1}X_1 + \cdots + \hat{\alpha}_{jp}X_p) \qquad \text{for } j = 1, 2, \ldots, m\,,$$

  then **predict** Y from the H's via another (non-linear) function $\boldsymbol{g_Y}$,

$$\hat{Y} \;=\; g_Y(\hat{\beta}_0 + \hat{\beta}_1 H_1 + \cdots + \hat{\beta}_m H_m)\,.$$

  The function g is called the ***activation function***, usually taken to be

$$g(z) \;=\; \frac{1}{1+e^{-z}},$$

  and recommendations also exist for the choice of g_Y, including for *classification*.

The coefficients, whose values are determined via the model fitting process, are called ***weights***:

$$\begin{array}{lll} \hat{\alpha}_{j0}, \hat{\alpha}_{j1}, \ldots, \hat{\alpha}_{jp} & \text{for } j = 1, 2, \ldots, m & (m(p+1) \text{ weights}) \\ \hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_m & \text{for } j = 1, 2, \ldots, m & (m+1 \text{ weights}) \end{array}$$

The "derived" explanatory variables, $H_1, H_2, \ldots, H_m$, are called ***hidden units***, and together comprise the so-called ***hidden layer*** of the neural network.

For more information, see the textbook.

7.4 Ensemble Methods

- ***Ensemble methods*** involve using **multiple** classification (or prediction) methods, and then classifying based on *majority vote* (or predicting based on *averaging* predictions).

  For example, we could perform *random forest*, *k nearest neighbor*, and *naive Bayes*, then classify an individual to whichever class got the most of the three predictions.

  For more information, see the textbook.

7.5 Evaluating Models

- Recall that a model ***overfits*** the **original data** if it predicts the *those* responses well, but *not* responses of **new** observations.

  Overfitting results when the ***complexity*** of the model is *too high.*

- For example, here again are the data on **lengths** and **weights** of **nine** snakes (from Class Notes 7):

```
Ln <- c(85.7, 64.5, 84.1, 82.5, 78.0, 81.3, 71.0, 86.7, 78.7)
Wt <- c(331.9, 121.5, 382.2, 287.3, 224.3, 245.2, 208.2, 393.4, 228.3)

snakes <- data.frame(Length = Ln, Weight = Wt)
```

```
g <- ggplot(snakes, aes(x = Length, y = Weight)) + geom_point()
g
```

We fit each of these ***polynomial regression models*** to the data:

$$\begin{array}{lrcl} \text{Model 0}: & Y & = & \hat{\beta}_0 \\ \text{Model 1}: & Y & = & \hat{\beta}_0 + \hat{\beta}_1 X \\ \text{Model 2}: & Y & = & \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 \\ \text{Model 3}: & Y & = & \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 \\ \text{Model 4}: & Y & = & \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 + \hat{\beta}_4 X^4 \\ \text{Model 5}: & Y & = & \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 + \hat{\beta}_4 X^4 + \hat{\beta}_5 X^5 \end{array}$$

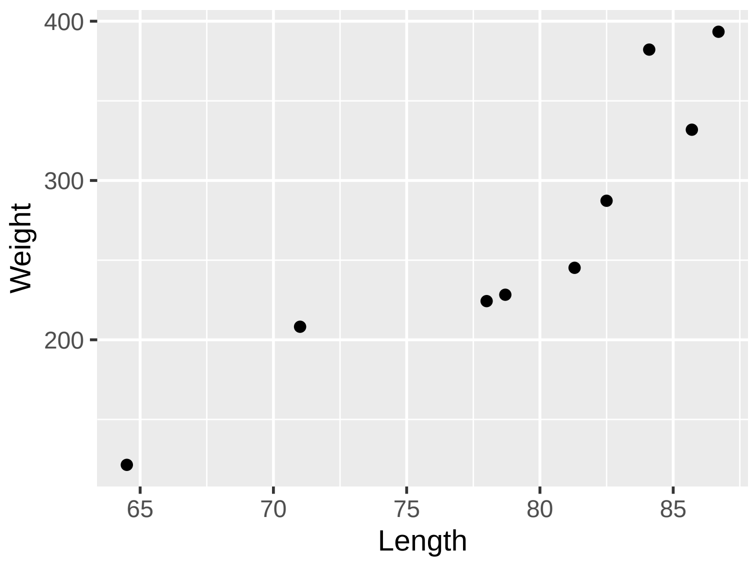

Figure 1

where Y is the **weight** and X the **length** of a snake.

```r
g + stat_smooth(method = "lm", formula = y ~ 1, se = F) +
ggtitle(label = "Model 0")
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 1), se = F) +
ggtitle(label = "Model 1")
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 2), se = F) +
ggtitle(label = "Model 2")
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 3), se = F) +
ggtitle(label = "Model 3")
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 4), se = F) +
ggtitle(label = "Model 4")
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 5), se = F) +
ggtitle(label = "Model 5")
```

The models fit the **original** data progressively **better** as the model **complexity** gets **higher**, but they **don't necessarily** predict **new** observations better.

For example, here are five **new** snakes:

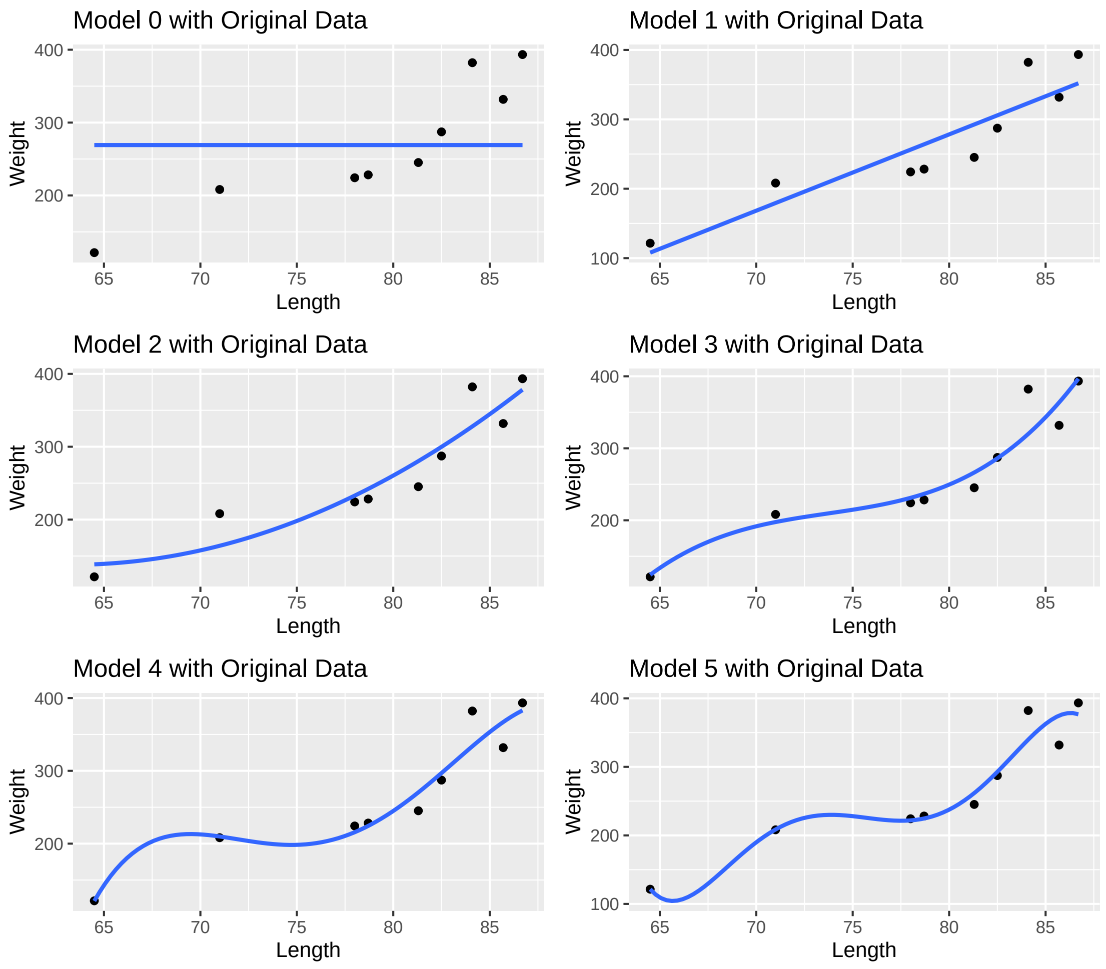

Figure 2

```
newSnakes <- data.frame(Length = c(67, 72, 77, 81, 86),
                        Weight = c(127.9, 153.7, 204.7, 300.6, 291.4))
newSnakes

##   Length Weight
## 1     67  127.9
## 2     72  153.7
## 3     77  204.7
## 4     81  300.6
## 5     86  291.4
```

The **fifth-degree** polynomial performs well in ***in-sample testing*** (i.e. it fits the **original** data well), but not so well in ***out-of-sample testing*** (i.e. it doesn't predict **new** observations very well).

The the **linear** model does better in ***out-of-sample testing***.

```r
g <- ggplot(snakes, aes(x = Length, y = Weight)) + geom_point(alpha = 0.05)
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 1), se = F) +
ggtitle(label = "Model 5 with New Snakes") +
geom_point(data = newSnakes)
```

```r
g + stat_smooth(method = "lm", formula = y ~ poly(x, 5), se = F) +
ggtitle(label = "Model 5 with New Snakes") +
geom_point(data = newSnakes)
```

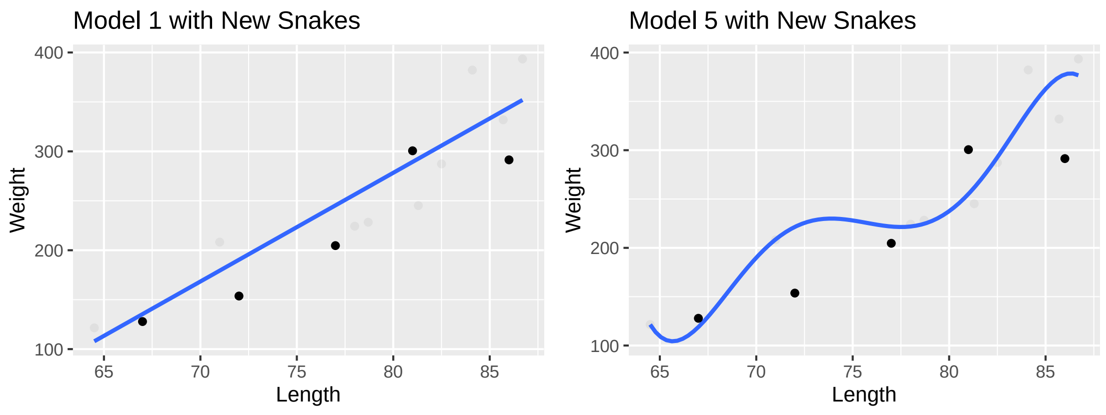

Figure 3

The **degree** of the polynomial is a kind of **tuning parameter** that controls how "flexible" the model is. Higher-degree polynomials can "flex" to conform to the data better. Lower-degree ones are more "rigid".

We can use the five **new** observations to ***validate*** a given choice for the polynomial **degree**: The degree that leads to the **smallest prediction errors** for the **new** observations is preferred.

7.5.1 Cross-Validation

- In the absence of **new** observations by which to **validate** a model, we can divide the **original** data set into *two parts*:
  - ***Training set***: Used to build (fit) the model.
  - ***Test set***: Used to test (or validate) the model.

For example, **80%** of the (original) data set might be used as the **training set** to build the model and the other **20%** as the **test set** to test (or validate) the model.

- Another method is ***cross-validation***. Here's how to perform (***two-fold***) cross-validation:

  1. Randomly separate the (original) data into two sets with the same number of observations. Call them `my.data1` and `my.data2`.
  2. Build (fit) the model using the data in `my.data1`, then run the data in `my.data2` through the model and measure the model's (**out-of sample**) **prediction error**.
  3. Now reverse the roles of `my.data1` and `my.data2` (i.e. use `my.data2` to build the model and `my.data1` to test it).
  4. If your model **overfits** the data, it will likely have large **prediction errors** on the second (**out-of-sample**) data set.

- ***k-fold*** cross-validation is similar, but the (original) data set is separated into ***k*** smaller sets that take turns serving as the **test set**.

7.5.2 Measuring Prediction Error: Numerical Response

- When the response (Y) is **numerical**, here are some ways to measure a model's **prediction error**:

  – ***RMSE***: Root mean squared error:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}\,.$$

  (where Y_i is an *observed* response and $\hat{Y}_i$ is it's *predicted* value.)

  – ***MAE***: Mean absolute error:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|\,.$$

  – ***Correlation***: The correlation between the Y_i's and the $\hat{Y}_i$'s.

  – $\boldsymbol{R^2}$: The R^2 value (or the *adjusted* R^2 value).

7.5.3 Categorical Response: The Confusion Matrix and ROC Curves

- When the response (Y) is **categorical**, **prediction** is called **classification**. The **prediction error** is measured by the **confusion matrix**.

  When the response has *two categories* (**positive** and **negative**, say), we can compute the ***true positive*** and ***true negative*** rates (sometimes called ***sensitivity*** and ***specificity***). For example, consider this **confusion matrix**:

		Prediction		
		Positive	Negative	
Actual	Positive	25	15	
	Negative	13	37	

The **true positive** and **true negative rates** are:

$$\begin{aligned}\textbf{True Positive Rate} &= \frac{25}{25+15} = \mathbf{0.625}\\ \textbf{True Negative Rate} &= \frac{37}{13+37} = \mathbf{0.74}\end{aligned}$$

We can also compute the ***false positive rate*** (which is one minus the *true negative rate*):

$$\textbf{False Positive Rate} = \frac{13}{13+37} = \mathbf{0.26}\,.$$

- Classification procedures usually classify individuals as **positive** if the **probability** of the individual belonging to that category is greater than the ***threshold*** value **0.5**.

  But a different threshold value could be used. Using a value *smaller* than 0.5 will result in **more true positives** but also **more false positives**. Using a value *larger* than 0.5 will result in **fewer false positives** but also **fewer true positives**.

  We can assess a classifier's performance across the range of **threshold** values from **0.0** to **1.0** via a ***receiver operating characteristic*** (or ***ROC***) ***curve***.

  An **ROC curve** is a plot of the **true positive rate** (y-axis) versus the **false positive rate** (x-axis) for candidate threshold values ranging between **0.0** and **1.0**.

  For more details, see the text book.

Section 7.5 Exercises

Exercise 1 We'll use `snakes` as the **training set** to *build* models and `newSnakes` as the **test set** to *test* the models and *validate* one of them.

Here are the (**original**) data (to be used as the **training set**):

```
Ln <- c(85.7, 64.5, 84.1, 82.5, 78.0, 81.3, 71.0, 86.7, 78.7)
Wt <- c(331.9, 121.5, 382.2, 287.3, 224.3, 245.2, 208.2, 393.4, 228.3)

snakes <- data.frame(Length = Ln, Weight = Wt)
```

and here are the five **new** snakes (to be used as the **test set**): :

```
newSnakes <- data.frame(Length = c(67, 72, 77, 81, 86),
                        Weight = c(127.9, 153.7, 204.7, 300.6, 291.4))
```

Fit these **six *polynomial regression models*** (using the "as is" function `I()`):

```
mod0 <- lm(Weight ~ 1, data = snakes)
mod1 <- lm(Weight ~ Length, data = snakes)
mod2 <- lm(Weight ~ Length + I(Length^2), data = snakes)
mod3 <- lm(Weight ~ Length + I(Length^2) + I(Length^3), data = snakes)
mod4 <- lm(Weight ~ Length + I(Length^2) + I(Length^3) + I(Length^4),
           data = snakes)
mod5 <- lm(Weight ~ Length + I(Length^2) + I(Length^3) + I(Length^4) +
           I(Length^5), data = snakes)
```

Obtain the **predicted weights** for the five **new** snakes:

```
pred0 <- predict(mod0, newdata = newSnakes)
pred1 <- predict(mod1, newdata = newSnakes)
pred2 <- predict(mod2, newdata = newSnakes)
pred3 <- predict(mod3, newdata = newSnakes)
pred4 <- predict(mod4, newdata = newSnakes)
pred5 <- predict(mod5, newdata = newSnakes)
```

Obtain the **prediction errors** associated with each model (in predicting **weights** of the five **new** snakes):

```
pe0 <- newSnakes$Weight - pred0
pe1 <- newSnakes$Weight - pred1
pe2 <- newSnakes$Weight - pred2
pe3 <- newSnakes$Weight - pred3
pe4 <- newSnakes$Weight - pred4
pe5 <- newSnakes$Weight - pred5
```

Obtain the **RMSE** for each model (based on **prediction errors** for the five **new** snakes):

```
rmse0 <- sqrt(mean(pe0^2))
rmse1 <- sqrt(mean(pe1^2))
rmse2 <- sqrt(mean(pe2^2))
rmse3 <- sqrt(mean(pe3^2))
rmse4 <- sqrt(mean(pe4^2))
rmse5 <- sqrt(mean(pe5^2))
```

Which of the six **polynomial regression models** is best according to the (*out-of-sample*) **RMSE**?