STATISTICS 452/652: Statistical Learning and Prediction

October 27, 2020

# Lecture 17: Intro to Classification

**(Reading: Section 2.1.5, 2.2.3)**

## 1 Goals of lecture

- Regression is great for predicting means or numerical responses.
- Categorical responses are different
  - Groups have no magnitude and (usually) no ordering
  - Squared error loss is not relevant
- We have seen categorical measurements used as explanatory variables
- Need methods for predicting categorical *responses*
  - Some are similar to regression methods we have learned
  - Others are quite different

## 2 Terminology and notation for classification

- Response measurement is a *nominal variable*, $Y$
  - $Y$ takes on values from $K$ possible CLASSES
  - Classes are indexed by $k = 1, \ldots, K$
    - Confusingly, ISLR uses "$j$" to index classes;
    - I reserve $i$ for different observations, $j$ for different explanatory variables, and $k$ for different classes
  - $K$ is often 2.
- Our "universe" has the same principles as before, but with some different details

1. In regression:
   (a) There was a *continuous* distribution of $Y$ values across the universe, $Y = g(\mathbb{X}) + \delta$.
      i. Not all measurements taken at $\mathbb{X} = x_0$ will equal $g(x_0)$
   (b) True structure $g(\mathbb{X})$ represented the mean of the distribution of $Y$
   (c) "Irreducible error" $\delta$ represented a numerical deviation from $g(\mathbb{X})$ for a given observed value
2. In classification,
   (a) At a point $\mathbb{X} = x_0$ in the universe there is a *discrete* distribution across the $K$ possible $Y$ values,

$$\begin{aligned} p_1(x_0) &= P(Y = 1|\mathbb{X} = x_0) \\ p_2(x_0) &= P(Y = 2|\mathbb{X} = x_0) \\ &\vdots \\ p_K(x_0) &= P(Y = K|\mathbb{X} = x_0) \end{aligned}$$

      i. That is, each possible class has a certain probability of occurring.
      ii. This probability likely depends on values of explanatory variables (both measured and unmeasured)
   (b) The "true class" at any $\mathbb{X} = x_0$ point in the universe is whichever class has the highest probability there
      i. Sometimes called the Bayes Classifier, although it is usually a theoretical concept rather than an actual computable thing
   (c) The "irreducible error" is now represented by the fact that some class other than the most likely class could be observed instead.

- Objective is to learn a function or machine $f(X)$ that will return a the *most likely class* $\hat{y}$ at a given value of $X$
  - Predicting the most likely class gives us the best chance of being right.
  - And estimated $\hat{f}$ is called a classifier
- Squared error measures like $RSS$, $sMSE$, and $MSPE$ are no longer relevant, since both $y$ and $\hat{y}$ are class labels instead of numerical values
  - Don't be confused by our use of numbers $1, 2, \ldots, K$ to *represent* class labels.
  - Class 3 is not equal to class 1 + class 2.
- Instead we measure error most often by the misclassification rate
  - What fraction of predictions were the wrong class?
  - Compute $I(y_i \neq \hat{y}_i)$ for each observation
    - $= 0$ if class is correctly predicted, $= 1$ if class is incorrectly predicted

  - Compute average indicator across sample
    - Gives back proportion of incorrect predictions

- Also often produce CONFUSION MATRIX
  - Just a $K \times K$ table of $\hat{Y}$ vs. $Y$

- Fitting, comparing, selecting, and evaluating models is done using same principles as before.
  - Just substitute misclassification rate for mean squares
  - Can be computed on training, validation, or test sets with exactly the same purposes and interpretations

- Bias-variance tradeoff is still alive and well here!
  - Some models are more flexible and complex than others
  - May be better at guessing which class *occurred* in the training set, but not as good at generalizing to when a different class was *more likely to occur* in the population.

# 3 K-Nearest Neighbours: A Simple Classifier

Focus now on a sample of size $n$ from $X$ and $Y$. How do we find $\hat{f}$?

- Many possible models and algorithms do classification!! See Figure 1.

- A very simple idea that does not require a model is to classify any point $x_0$ according to the majority of points in its "neighbourhood"
  - Whichever class is most all around $x_0$ is a good guess for the best class at $x_0$

1. K-Nearest-Neighbours (KNN) Algorithm:[1]
   (a) Choose $m$, the chosen size of the "neighbourhood," in terms of the number of points that will get to "vote" on the class
   (b) Select a point $x_0$ in the $p$-dimensional space of $X$.
   (c) Identify the $m$ closest points to $x_0$ the neighbourhood and call this collection $\mathcal{N}_m(x_0)$.
   (d) Count the number of points of each class within $\mathcal{N}_m(x_0)$:
   (e) $\hat{Y}(x_0)$ is class with the highest count

[1]Note: People usually use "$K$" to index the number of neighbours to be used in the prediction. The procedure is then abbreviated as "KNN". We are already using $K$ to be the number of classes. So to avoid confusion, so I will use the ubiquitous "$m$" to index number of neighbours. But I will still abbreviate it as "KNN" to emphasize the universal term.

Figure 1: Simulated data for a classification problem, and three classifiers: Top left: Linear; top right: 15 nearest neighbour; bottom left: 1 nearest neighbour; bottom right: Bayes classifier based on known model




**FIGURE 2.1.** *A classification example in two dimensions. The classes are coded as a binary variable (*BLUE $= 0$, ORANGE $= 1$*), and then fit by linear regression. The line is the decision boundary defined by* $x^T\hat{\beta} = 0.5$*. The orange shaded region denotes that part of input space classified as* ORANGE*, while the blue region is classified as* BLUE*.*



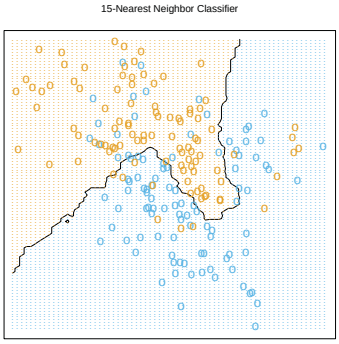



**FIGURE 2.2.** *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (*BLUE $= 0,$ ORANGE $= 1$*) and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*



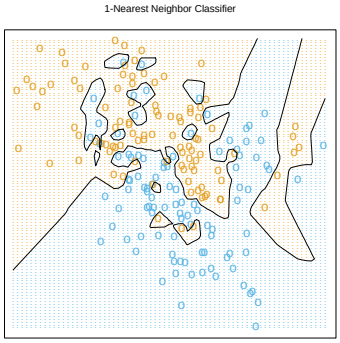



**FIGURE 2.3.** *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (*BLUE $= 0,$ ORANGE $= 1$*), and then predicted by* 1*-nearest-neighbor classification.*



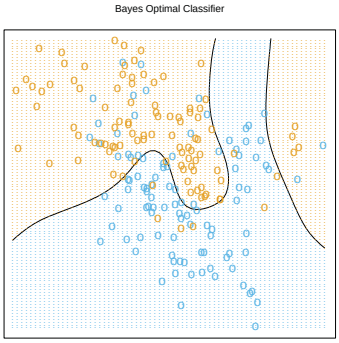



**FIGURE 2.5.** *The optimal Bayes decision boundary for the simulation example of Figures 2.1, 2.2 and 2.3. Since the generating density is known for each class, this boundary can be calculated exactly (Exercise 2.2).*

2. See Figure 1 for examples with $m = 1$ and $m = 15$
3. This produces very irregular shapes as boundaries between classes.
4. $m$ is a tuning parameter:
   (a) Matters A LOT!
   (b) Bias/Variance tradeoff
      i. Larger $m$: low variance and higher bias
      ii. Smaller $m$: higher variance and low bias
      iii. See Figure 2

Figure 2: Training and test error rates for KNN using various values of $m$ from a simulated example. $X$-axis is $1/m$, so model complexity increases from left to right. (From ISLR.)

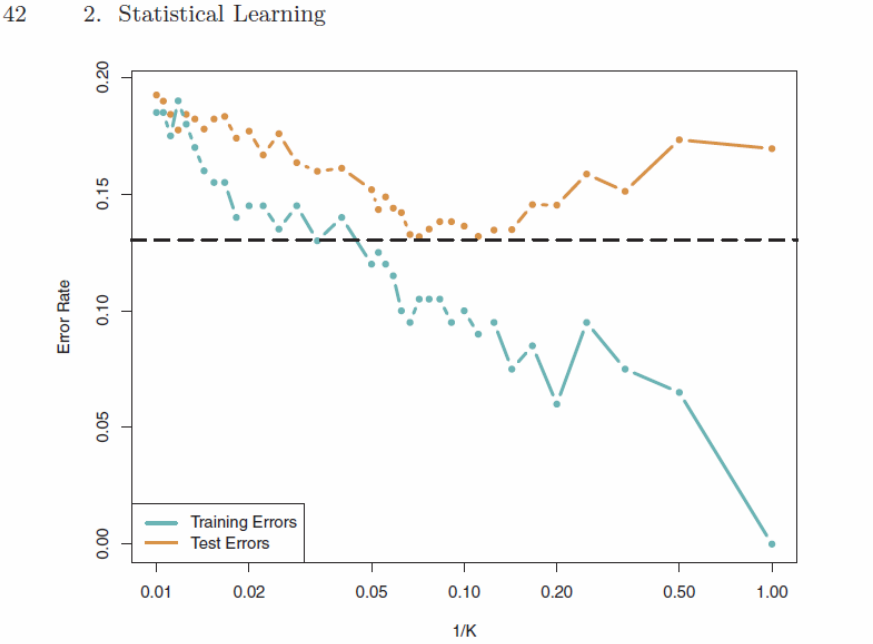



**FIGURE 2.17.** *The KNN training error rate (blue, 200 observations) and test error rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of flexibility (assessed using $1/K$) increases, or equivalently as the number of neighbors $K$ decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.*

# 4 Notes

KNN is not one of my favourite tools, but some people really like it

1. KNN struggles in high dimensions

(a) Larger $m$ may create big neighbourhoods that span regions with very different class probability distributions

(b) Problem gets worse as $p$ gets larger because points less dense, have to look farther for $m$ neighbours.

(c) Adds potential bias and variance from distribution and variability of $X$

2. Complicated computation, slow for large $n, p$

3. Need to *standardize* the features (sd=1) before using

   (a) Want "distance" to have same meaning in all directions: relative to spread of data

   (b) Categorical features have no "distance" metric

      i. Need to be turned into contrasts (e.g., indicators) and standardized

4. *Asymptotically* the error of the 1-NN is twice the optimal (Bayes) error

   (a) Application of this knowledge: fit a 1-NN and measure test error to estimate target error for other methods as half that much

   (b) Need large $n : p$ ratio to resemble asymptotics

5. Somewhat sensitive to presence of irrelevant features

   (a) Helps considerably if you can do something to remove worthless variables.

6. Performance *can* be surprisingly good.

   (a) Insensitive to outliers if $m$ not too close to 1

**Example: KNN on Wheat Kernel Data (L17 - Nearest Neighbour Wheat.R)**

Here we introduce a new data set, the Wheat data (Original source: Martin, Herrman, Loughin, and Oentong (1998), *Cereal Chemistry*. Description amended from Bilder and Loughin (2014) *Analysis of Categorical Data with R):*
The presence of sprouted or diseased kernels in wheat can reduce the value of a wheat producer's entire crop. It is important to identify these kernels after harvested but prior to sale. To facilitate this identification process, automated systems have been developed to separate healthy kernels from the rest. Improving these systems requires better understanding of the measurable ways in which healthy kernels differ from kernels that have sprouted prematurely or are infected with a fungus ("Scab"). To this end, an observational study measures numerous physical properties of kernels—density, hardness, size, weight, and moisture content—on a sample of wheat kernels from two different classes of wheat, hard red winter ("hrw") and soft red winter ("srw"). Each kernel's condition was also classified as "Healthy," "Sprout," or "Scab" by human visual inspection. In the data provided by the authors of this paper, we have measurements from 275 wheat kernels.

The data are available in the spreadsheet **wheat.csv**. This is an "easy" problem in the sense that the number of explanatories is relatively small (6). The data have a few quirks, and I supply the code to smooth them out and create two sets. a summary of the full data is below.

```
> summary(wheat)
 class         density          hardness
 hrw:143   Min.   :0.7352   Min.   :-44.080
 srw:132   1st Qu.:1.1358   1st Qu.:  0.689
           Median :1.2126   Median : 24.465
           Mean   :1.1885   Mean   : 25.564
           3rd Qu.:1.2687   3rd Qu.: 45.606
           Max.   :1.6454   Max.   :111.934
      size             weight          moisture
 Min.   :0.5973   Min.   : 8.532   Min.   : 6.486
 1st Qu.:1.8900   1st Qu.:21.982   1st Qu.: 9.540
 Median :2.2303   Median :27.610   Median :11.909
 Mean   :2.2047   Mean   :27.501   Mean   :11.192
 3rd Qu.:2.5125   3rd Qu.:32.882   3rd Qu.:12.538
 Max.   :4.3100   Max.   :46.334   Max.   :14.514
      type        classnum
 Healthy:96   Min.   :1.00
 Scab   :83   1st Qu.:1.00
 Sprout :96   Median :1.00
              Mean   :1.48
              3rd Qu.:2.00
              Max.   :2.00
```

**KNN** There are lots of functions that do KNN. There is a `knn()` function in the `class` package that is loaded automatically into `R`. There is also a `knn()` function in the "fast nearest neighbor" (`FNN`) package. I'll use the latter, but I have not seen a comparison of them. One can also tune using `knn.cv()` in both packages, using leave-one-out, or using the `train()` function in `caret` for the first function. I wrote my own tuning script and follow-up plot.

The program first scales the explanatory variables by subtracting off the mean of the training set and dividing by the standard deviation of the training set. Then all variables contribute equally to distance measures, and the test data have the same transformation applied as the training data.

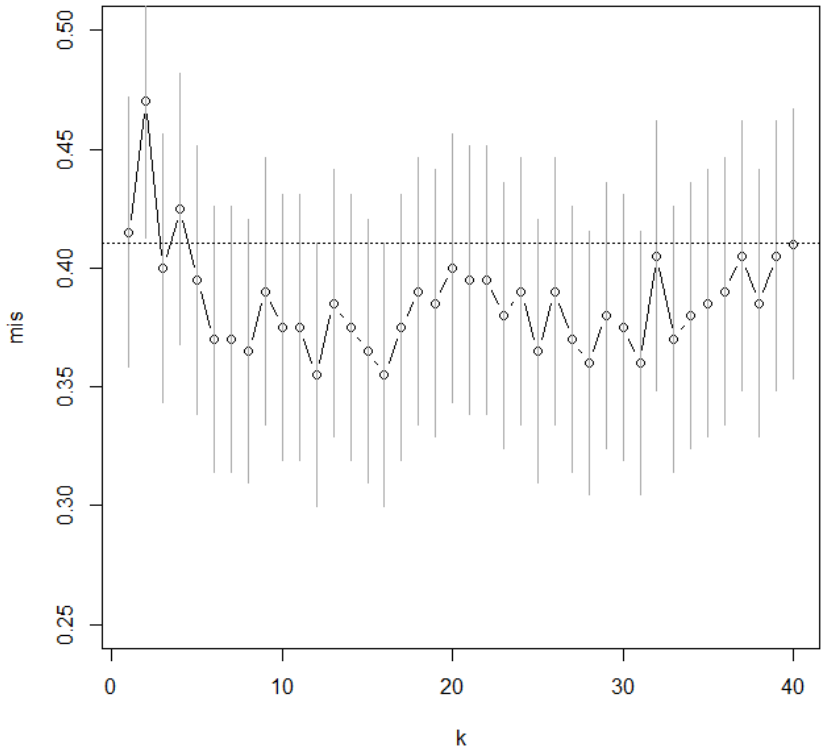
Then I fit a 1-NN model ($m = 1$) to show how the basic code works. The `knn()` function uses the matrix or data frame of training data in `train=` to produce predicted values for a matrix or data frame in `test=`. They can be the same data if training error is desired. The response factor-class object is listed as `cl=`. The number of NNs is `k=` . I use the code to compute training and test misclassification rates for 1-NN (we know that training error should be 0), as well as confusion matrices. The results are below.

```
> # Fit the 1-NN function using set 1 to train AND test
> #   (compute training error)
> knnfit.1.1 <- knn(train=x.1, test=x.1, cl=set1[,6], k=1)
>
> # Create Confusion Matrix and misclass rate
> table(knnfit.1.1, set1[,6],  dnn=c("Predicted","Observed"))
         Observed
Predicted Healthy Scab Sprout
  Healthy      71    0      0
  Scab          0   61      0
  Sprout        0    0     68
> (misclass.knn1.1 <-
+          mean(ifelse(knnfit.1.1 == set1[,6], yes=0, no=1)))
[1] 0
>
> # Fit the 1-NN function using set 1 to train and set2 to test
> #   (compute test error)
> knnfit.1.2 <- knn(train=x.1, test=x.2, cl=set1[,6], k=1)
> # Create Confusion Matrix and misclass rate
> table(knnfit.1.2, set2[,6],  dnn=c("Predicted","Observed"))
         Observed
Predicted Healthy Scab Sprout
  Healthy      13    4     12
  Scab          2   12      1
  Sprout       10    6     15
> (misclass.knn1.2 <-
+          mean(ifelse(knnfit.1.2 == set2[,6], yes=0, no=1)))
[1] 0.4666667
```

We see that the 1-NN provides perfect training prediction, which happens because each response observation is the predicted value for every $x_0$ it is closest to, including itself. The test misclassification rate is a miserable 47%, largely because the classifier does not distinguish well between `Healthy` and `Sprout` classes.

Next I ran the `knn.cv()` function to do use leave-one-out CV to compute the error rate for $m = 1, 2, \ldots, 40$. I also computed an approximate standard error for these error rates, assuming that the prediction errors are independent and using the standard error for the binomial proportion. The result is in Figure 3

Figure 3: CV error rates for KNN on wheat training data using $m = 1, 2, \ldots, 40$.




The misclassification rate is minimized at $m = 12$, whereas applying a "1-se rule" and using the least complex model with error rate within 1SE of the minimum suggests $m = 40$. The test set error rates for these values are also high: 0.49 and 0.51. This may just be a data set where there are too many unnecessary variables that influence the measurement of distance between observations but do not improve predictions

---

# 5 What to take away from this

1. Classification is a new problem where we try to predict a categorical variable
2. The "true structure" is a probability distribution across all possible classes at each location in the universe.
   (a) The class with highest probability it the best class for that location.
3. There are lots of methods for doing classification
4. We introduced one simple one, $K$-nearest neighbours, that uses observations closest to a point in space to determine the prediction at that point.

5. It's not a great technique but can be very flexible and adaptable to any true structure

# 6 Exercises

## Concepts

1. *Establishing a "baseline" error rate.* Suppose that we have a classification problem with $K$ classes, and suppose that the proportions of observations in each class are $p_1, p_2, \ldots, p_K$. Suppose that class Q has the largest proportion, so that $p_Q > p_m$ for all other $m \neq Q$.

   If you had no explanatory variables and still had to do prediction, you would use a NAIVE CLASSIFIER that always assigns most common class to all predictions. In our problem, **what would be the misclassification rate for the naive classifier?**

   This is sometimes called the BASELINE ERROR RATE for the problem, and represents a guess at the worst error rate you expect and "real" classifier to have, assuming that future samples have the same distribution of classes as this one.

2. *Difficulties with classifying unbalanced responses.* Suppose you have a classification problem with $K = 2$, and that 95% of the responses are class 1. **What is the baseline error rate for this problem?**

   It is often the case that the baseline error rate is hard to beat with a "real" classifier, because correctly classifying a portion of the class-2 data often causes an even larger number of class-1 data to be misclassified. For example, if the ratio of class 1 to class 2 is 95:5, then correctly classifying even one or two class-2 observations may cause 5 or 10 class-1 responses to be misclassified. For this reason, we may choose to use other measures besides total misclassifications to judge a classifier. We will talk about these more later.

## Application

DATA: Vehicle Image Recognition[2], **vehicle.csv**

This data set is described in Section 8.7 of Izenman, with a longer description here: http://archive.ics.uci.edu/ml/datasets/Statlog+%28Vehicle+Silhouettes%29 . Please read through this description briefly to get a general idea of what these data are about. Details aren't important.

The basic purpose of the study was to improve on image analysis to recognize different shapes. Four model cars—a 2-door car (OPEL, coded as `1`), a 4-door car (SAAB, coded as `2`), a double-decker bus (BUS, coded as `3`), and a minivan (VAN, coded as `4`)—were viewed in silhouette from different angles and 18 different measurements were taken of these images. The goal is to use these 18 measurements to identify which type of car made the picture. So this is a classification problem. Remember the coding of the response variable, because it will help with later interpretations.

[2]This dataset comes from the Turing Institute, Glasgow, Scotland.

The data set can be read using
`read.csv("<file location>\\vehicle.csv")`
The explanatory variables have long names, and the response is `class`, with coding 1, 2, 3, 4 as described above.

1. Get to know the data

   (a) Read the data and **print a summary**.

   (b) Notice that `class` has been read as numeric. Convert it to a factor using
   `vehdata$class = factor(vehdata$class, labels=c("2D", "4D", "BUS", "VAN"))`
   Show a summary of this new version of `class`.

   (c) **Print the correlation matrix for the explanatory variables. Comment on any strong correlations (maybe beyond $\pm 0.7$ and note especially any beyond $\pm 0.9$)**

2. Split the data using the code below, where `set1` will be the training set for future analyses and `set2` the test set:

```
set.seed(46685326, kind="Mersenne-Twister")
perm <- sample(x=nrow(vehdata))
set1 <- vehdata[which(perm <= 3*nrow(vehdata)/4), ]
set2 <- vehdata[which(perm > 3*nrow(vehdata)/4), ]
```

   **Print the top 6 observations from each data set**.

3. Run a KNN analysis on the training data using $m = 1$.

   (a) **Show the confusion matrix for the test data. Comment on how well separated the four classes are. In particular, are there classes that are easier/harder to separate?**

   (b) Compute and **report the test misclassification rate and approximate standard error.**

4. Reset the seed to `set.seed(9910314, kind="Mersenne-Twister")`. Tune the KNN using CV error with `knn.cv()`. Use a grid from $m = 1, \ldots, 40$. This may take a few seconds or minutes.

   (a) Plot the validation error with standard errors against the number of neighbours. **Show the plot and comment: is there a clear best $m$ or is there a broad range of similar values, according to the SE?**

   (b) Report the value of $m$ with lowest error, as well as the one selected by the 1SE rule.

(c) Compute the test misclassification rate with both of these parameter values. **Report both error rates, using only one more digit than the first digit in their standard errors.** (For example, if the SE is 0.00375, the first digit in the SE 3 after the decimal, so report error to 4 after the decimal.)

Keep these error rates handy. You will use them for comparing all classification methods.

It would be better to use a more rigourous method like multiple reps of CV for computing error rates to make an "arena" for comparing methods. Classification is no different from regression in this way. However, this process is time consuming, and you have already practiced doing this on regression. I want to keep the classification assignments lighter, because they will accumulate rather quickly.