Prediction problems 2: Classification

Data Science 101 Team

May 17, 2017
Learning Goals

- What is a classification problem?
- Defining a model
- Fitting a model
- Validating your model (does it work?)
Outline

- Examples and motivation
- Binary classification with linear predictions
  - Example: Zip Code prediction
  - A good enough prediction model
- Binary classification by nearest neighbors
  - One-nearest neighbor
  - $k$-nearest neighbor

Recall our learning goals: to formulate a model, fit it, and then evaluate it
The binary classification problem

- Data is in \((x, y)\) pairs, where \(x\) is *independent* variable (or feature, or covariate vector) and \(y\) is *target* to be predicted
- Goal in classification problems: find a good prediction function \(\text{Predict}\) so that
\[
\text{Predict}(x) \approx y
\]
for most data pairs \((x, y)\) in the future
Example: web advertising

Google/Bing are deciding whether to show you an advertisement or not

- Some covariates $x$:
  - What you actually searched for
  - What the advertisement is selling
  - How much money Google will make if you click on the ad

- Target $y$: whether you click on the ad or not!
Example two: Glioblastoma classification in image

![Glioblastoma images](image)

Figure 2: “Glioblastoma images, Figure 2.29 of Forghani and Schaefer, Clinical Applications of Diffusion, Functional Neuroradiology, pp. 13-52, 2011.”

Tumor classification from MRI (Magnetic Resonance Imaging). Given 2D images of lateral brain slices, can we identify tumors? Goal is to classify each pixel (voxel) as tumorous

- **Covariates $x$:** for each pixel,
  - Grayscale color of neighboring brain regions
  - Sometimes different “slices of brain”
- **Target $y$:** whether position in the brain is tumor
Binary classification

- Target variable \( y \) is either \( \{0, 1\} \)-valued or \( \{-1, +1\} \)-valued.
  - Call an example *positive* if its label \( y = 1 \) and *negative* otherwise
  - Example: label \( y \) is whether a patient is sick or not
  - Focus on the \( \{-1, 1\} \) case

- Independent variable (vector) \( x \in \mathbb{R}^p \)
- Making predictions given \( x \) is a bit different

\[
\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j
\]

- But usually we just take the sign of the above, i.e.

\[
\text{Predict}(x) = \begin{cases} 
1 & \text{if } \hat{y} > 0 \\
-1 & \text{if } \hat{y} \leq 0
\end{cases}
\]

- Sometimes we are more *confident* in answer if \( |\hat{y}| \gg 0 \)
Example: Digit Recognition

- Data vectors $x$ are in $[-1, 1]^{256}$
  - Each coordinate $j$ corresponds to grayscale level of pixel $j$ (of 256)
  - We predict digit (either 4 or 9) from pixels

```r
D = loadZips(4, 9, "data/");  # helper function for data
Xtest = D$Xtest;
ytest = D$ytest;
Xtrain = D$Xtrain;
ytrain = D$ytrain;
numeric.train.labels = ytrain;
numeric.train.labels[ytrain == 1] = 4;
numeric.train.labels[ytrain == -1] = 9;
```
Digit Recognition

```r
par(mfrow = c(1, 2))
displayImage(Xtrain[1,], numeric.train.labels[1])
displayImage(Xtrain[11,], numeric.train.labels[11])
```
Using linear regression for classification

- Target variable \( y \in \{-1, 1\} \)
- Goal: fit prediction model

\[
\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j = \beta_0 + \beta \cdot x
\]

(prediction is sign of \( \hat{y} \))

- Idea: minimize sum of squared errors for pairs \((x_i, y_i)\) with \( y_i \in \{-1, 1\} \)

\[
\min_{\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^p} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta \cdot x_i)^2
\]

- Remember our friend George Box

\textit{All models are wrong; some are useful}

(this one is \textit{really} wrong! But it is useful)
Fitting a classifier with squared error

- In linear regression case, used `lm`, and we can do this again
- Tell R to regress the vector of $y$ against the matrix of $X$ via $y \sim X$:

```r
linreg.fit <- lm(formula = ytrain ~ Xtrain)
```

- Checking if we did OK at least: see how we make predictions on our data

```r
linreg.fit <- lm(formula = ytrain ~ Xtrain)
train.pred = linreg.fit$fitted.values
num.mistakes = sum(sign(train.pred) != ytrain)
cat(paste("Number of mistakes on fitted data: ", num.mistakes, "\n"))
```

Number of mistakes on fitted data: 7 out of 1296
Evaluating a classifier

- But if we are actually making predictions, we should make predictions
- Simplest way to evaluate a classifier (more later): validation!
  - That is why we usually keep a test set of data we do not fit
- When performing classification, separate data into
  - training set and a
  - test or validation set
- Use the validation set to check if your fitted model and predictions are good: test data pairs

\[(x_{i}^{\text{test}}, y_{i}^{\text{test}}) \text{ for } i = 1, \ldots, n_{\text{test}}\]

For each test pair \((x, y)\), compare sign of prediction

\[\hat{y} = \beta_0 + \beta \cdot x\]

to true \(y\)
Evaluating a classifier on held-out data

- We stored some validation/test data in $X_{\text{test}}$ and $y_{\text{test}}$
- Pull out parameters of our model and make some predictions

\[ \hat{y} = \beta_0 + \beta \cdot x \]

To get coefficients $\beta_0$ and $\beta$, use `coefficients` (the first entry is the intercept $\beta_0$)

```r
beta.0 = linreg.fit$coefficients[1]
beta = linreg.fit$coefficients[-1]  # 2:ncol(Xtrain)
```

To make predictions, use the $n_{\text{test}} \times p$ matrix of test examples, $X_{\text{test}}$

```r
test.pred = Xtest %*% beta + beta.0;  # Get the test predictions
```
Find mistakes

mistakes = which(sign(test.pred) != ytest)
cat(paste("Number of mistakes on held out data:\n", length(mistakes), " of 
length(ytest), ", round(100 * length(mistakes)/length(ytest), 1), "] error rate")
sep = ""))

Number of mistakes on held out data:
11 of 377 (2.9% error rate)
Visualizing the mistakes

- Take a look at some of the mistaken examples

```r
par(mfrow = c(1, 2))
## Show some of the mistaken images
displayImage(Xtest[mistakes[5], ], numeric.test.labels[mistakes[5]])
displayImage(Xtest[mistakes[9], ], numeric.test.labels[mistakes[9]])
```

True label 9  True label 9
Summary for linear prediction rules

- Make predictions by taking sign of
  \[ \hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j = \beta_0 + \beta \cdot x \]

- Must validate (check on test data)
- Many other classification schemes possible with linear models
  - Logistic regression
  - Probit regression
Nearest neighbor classification

Another way to perform classification: do what your friends do

- The *norm* \( \|x\| \) is the length of the vector \( x \)
- In one dimension, the norm is just absolute value, \( \|x\| = |x| \)
- In two dimensions,

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \|x\| = \sqrt{x_1^2 + x_2^2}
\]

Figure 3:
Nearest neighbor classification

- In higher dimensions (i.e. with $p$ parameters)

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \in \mathbb{R}^p, \|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}$$
How to carry out nearest neighbor classification

- Have *training pairs* $x_i, y_i$, where $y_i$ are classification labels
  - Can have any type of classification label
  - Could have $y_i \in \{-1, 1\}$, or $y_i \in \{1, \ldots, L\}$ where $L$ is number of labels

- Given a new example $x$, to make a prediction
  1. Find the training example $x_i$ closest to $x$, i.e.

     $$\|x_i - x\| \leq \|x_j - x\| \text{ for all } j = 1, \ldots, n$$

  2. Set $\hat{y} = y_i$ (the label of example $i$)
Nearest Neighbor Classification

Figure 4:
Nearest neighbor classification in R

Method `knn` (*k*-nearest neighbor) in the library `class` (classification)

- Usage:

```r
library(class)
test.predictions = knn(train = Xtrain, test = Xtest,
                       cl = ytrain, k = 1)
```

- Note that the method `knn` assumes it is given a test dataset already
Computing the error rate of our nearest-neighbor classifier

```r
num.errors = sum(ytest != test.predictions)
cat(paste("Number of mistakes on held out data:\n", num.errors, " of ", length(ytest), " (", round(100 * num.errors/length(ytest), 1), ")", "% error rate)\n
Number of mistakes on held out data:
17 of 377 (4.5% error rate)
```
To find all entries of a vector satisfying a condition, use `which`. We use this to find the first mistaken example:

```r
mistakes = which(test.predictions != ytest)
x = Xtest[mistakes[1], ]
distances.squared = rowSums((Xtrain - ones(nrow(Xtrain), 1)%*%x)^2)
closest.ind = which.min(distances.squared)
nearest.x = Xtrain[closest.ind, ]
```
Neighbor Mistakes . . .

And now we display the mistaken example \( x \) (left) and its nearest neighbor (right)

\[
\text{par(mfrow = c(1, 2))}
\]
\[
\text{displayImage}(x, \text{ label } = \text{ numeric.test.labels}[\text{mistakes}[1]])
\]
\[
\text{displayImage}(\text{nearest}.x, \text{ label } = \text{ numeric.train.labels}[\text{closest.ind}])
\]

True label  9  

True label  4
k-nearest neighbor classification

Figure 5:

- Can also average over multiple neighbors
- Have training pairs \( x_i, y_i \), where \( y_i \) are classification labels
- Given a new vector of covariates \( x \), to make a prediction:
  1. Find the \( k \) training examples in the dataset, call them \( x(1), \ldots, x(k) \) closest to \( x \)
  2. Take a majority vote of their labels \( y(1), \ldots, y(k) \)
  3. Set \( \hat{y} = \text{Vote}(y(1), \ldots, y(k)) \) breaking ties arbitrarily
In R, use `knn` with the value `k`:

```r
test.predictions = knn(train = Xtrain, test = Xtest, cl = ytrain, k = 2);
num.errors = sum(ytest != test.predictions)
```

Fraction of mistakes for 2 neighbors: 0.05
Fraction of mistakes for 3 neighbors: 0.037
Fraction of mistakes for 4 neighbors: 0.04
Check out the errors we make

Find the three images closest to the second of our mistakes

test.predictions = knn(train = Xtrain, test = Xtest, 
    cl = ytrain, k = 3);

mistakes = which(test.predictions != ytest);
x = Xtest[mistakes[2], ];
distances.squared = rowSums((Xtrain - ones(nrow(Xtrain), 1) %*% x)^2);
distance.ranks = rank(distances.squared);
near.inds = which(distance.ranks <= 3);
Check out the errors we make

The test image is in the upper left, nearest neighbors are the remaining three.