CS100: Studio 9

Classification

November 6 and 7, 2019

Instructions

During today’s studio, you’ll be building several binary classifiers in R to classify passengers on the Titanic as survivors or not. Specifically, you will be using $k$-nearest neighbors algorithm.

Upon completion of all tasks, a TA will give you credit for today’s studio. If you do not manage to complete all the assigned work during the studio period, do not worry. You can continue to work on this assignment until the following Wednesday at 6 PM. Come by TA hours any time before then to show us your completed work and get credit for today’s studio.

Objectives

To understand a supervised learning algorithm, namely $k$-nearest neighbors, and to use cross-validation to optimize the hyper-parameter of this algorithm.

Libraries

In this assignment, you’ll be relying on some special R libraries for building classifiers. Open RStudio, and then run the following commands in the console.

```
# Classification packages
install.packages("class") # kNN
install.packages("caret") # Cross-validation
```

Next, open a new R markdown file, and then enter and run the following setup code:

```
```

Note: You may have to install “prodlim”.

1
Data

The RMS Titanic was a luxury steamship that sank in 1912 off the coast of Newfoundland in the North Atlantic during its very first voyage. Of the 2,240 passengers and crew on board, more than 1,500 lost their lives. The people on board were of different ages, social classes, gender, and occupation. In this studio, you will be investigating the use of machine learning to predict whether a person would survive this disaster? (You can read more on the Titanic here).

The data describe various features of the people who were aboard the Titanic, such as their name, sex, age, etc. This file contains a detailed description of all the features.

Additionally, the data contain a binary variable (i.e., a label) indicating whether or not each passenger survived the sinking of the ship; 1 means the passenger survived, and 0 means they did not. In this assignment, your goal to build several classifiers to predict who survived the crash.

Use the following line of code to read in the data. Insert it towards the end of your setup chunk.

```r
titanic <- read.csv('https://cs.brown.edu/courses/cs100/studios/data/9/titanic.csv')
```

Spend a few minutes getting a feel for the data set. Explore it with the usual functions (`str`, `summary`), and try out `glimpse` as well. How many passengers survived? **Hint:** Try using the `table` function to answer this question: `table(titanic$survived)`.

Check the types of the features. You can use the functions `as.numeric`, `as.character`, etc. to convert any features whose types are not as you expect them to be. For example:

```r
# Convert the labels to a factor
titanic$survived <- as.factor(titanic$survived)
```

Finally, let’s see if the data are clean. How many observations are incomplete? **Hint:** Use `summary.` Some supervised learning algorithms (like k-NN) cannot handle missing data, so let’s also remove all the observations with missing values. (If we had time to conduct a more sophisticated analysis, we might try to fill in missing ages, for example, using regression.)

```r
titanic <- na.omit(titanic)
```

Explore the data again. How many passengers survived in this pruned data set? How many are female? How many are male?

**Feature Selection**

The first step in classification is to decide which features to include in your model. Spend some time discussing with your partner which features might best discriminate between a passenger’s chances of survival or not.
Generate some plots to help you choose appropriate features. For example, you can create a scatter plot depicting the relationship between any two numeric features (e.g., age vs. fare), and then color the points according to whether or not the passenger survived. You can do the same for one numeric and one categorical feature (e.g., age vs. sex).

# Sample plots
attach(titanic)
qplot(age, fare, col = survived)
qplot(age, sex, col = survived)

When plotting two categorical variables, you can adjust the size of the points to reflect the number of observations that fall in the various categories. Alternatively, you can jitter the data to see the individual data points.

# Sample plots
qplot(pclass, sex, col = survived, geom = "jitter")
qplot(pclass, age, col = survived, geom = "jitter", size = age)
qplot(pclass, age, col = survived, geom = "jitter", size = sibsp)
detach(titanic)

Note: qplot is a plotting function in the ggplot2 library. Here is a link to some more details about how to use qplot.

Choose two or three features for your analysis. You will use these features all throughout this studio to build your classifiers. We will refer to these features as feature_1, feature_2, and feature_3.

Hint: If you assign these features to variables, say feature_i, and do your analysis on these variables, you will find it easier to redo your analysis later on a different choice of features, because all you will have to do then is reassign your feature variables to new features, and the rest of your code should run more or less as is.

feature_1 <- ...
feature_2 <- ...
feature_3 <- ...

Note: This is not necessary! But may be helpful.

Model building

Now that we have selected our features, let’s build some classifiers! Specifically, let’s use R’s implementation of

\[ k \]

-NN to classify passengers on the Titanic. As you learned in lecture,

\[ k \]

-NN does not build a model, but instead classifies on the spot, the moment you call the function.
We’ll perform $k$-NN classification using the `knn` function in the `class` package. This function takes as input both training data and testing data, as well as ground truth: i.e., labels corresponding to the training data. Unsurprisingly, `knn` also takes as input $k$, the number of neighbors. You can view all the available additional/optional parameters by searching for `knn` in the Help window, or entering `?knn` in the console.

Create a training set `knn_train` using `select` to build a data frame that consists of only your preferred features: e.g., age and fare. Important: As

$k$-nearest neighbors depends on a distance function, it works best on quantitative features, not categorical ones (though it does work for categorical data that are encoded quantitatively: e.g., `gender` as 0, 1, 2, etc.). Be sure to “clean” your preferred features, so they are all numerical, not categorical.

```
# Training data
knn_train <- titanic %>% select(feature_1, feature_2)
```

For now, let’s use our training data as test data as well:

```
# Test data
knn_test <- knn_train
```

Next, create a vector of labels. Note that this vector must be of type `factor`.

```
# Training labels
knn_label <- titanic$survived
```

You are now ready to use `knn` to classify. Try $k = 3$:

```
# k-NN
knn3_class <- knn(knn_train, knn_test, knn_label, k = 3)
```

The output of `knn` is a vector of predictions for all data in the test set. Summarize `knn3_class` to find out how many of the passengers were classified as survivors, and how many were predicted to perish.

```
summary(knn3_class)
```

Repeat this exercise for $k = 1$, $k = 5$, and $k = 15$, creating `knn_class1`, `knn_class5`, and `knn_class15`, respectively.

These summaries give you a rough idea of whether your classifiers are predicting a number of survivors that is in the ballpark of the true number, but we need to take a deeper dive to compare the accuracy of these various classifiers.
Evaluating model accuracy

By now, you have now built several \( k \) classifiers. Congratulations! The next step is to try to gauge the accuracy of each classifier, so we can pick the best one. To do that, we’ll need to do two things: 1. Generate predictions from each classifier, and 2. Evaluate the accuracy of those predictions.

Generate Predictions

Generating predictions is trivial with \texttt{knn}, as its output is precisely a vector of predictions. Other classifiers do require a bit more work to extract predictions.

Training Accuracy

Training accuracy is the accuracy of a classifier on the data on which it was trained. Training accuracy is very rarely 100%, but it does tend to be much higher than the alternative, namely testing accuracy, which gauges how a classifier might perform “in the wild”.

Here is an \texttt{accuracy} function that evaluates the accuracy of predictions:

\begin{verbatim}
accuracy <- function(predictions, ground_truth) {
    mean(predictions == ground_truth)
}
\end{verbatim}

Given a vector of predicted labels \texttt{predictions} and a vector of actual labels \texttt{ground_truth}, this \texttt{accuracy} function calculates the proportion of values in \texttt{predictions} that match those in \texttt{ground_truth}. Add this function to your code base, and use it to compute the accuracy of your classifiers, like this:

\begin{verbatim}
accuracy(knn1_class, titanic$survived)
accuracy(knn3_class, titanic$survived)
...
\end{verbatim}

Your accuracies should be decent (> 0.65) Can you do better? The answer is yes. Before proceeding, spend a few minutes brainstorming with your partner about a strategy for improving your predictions.

\textit{Note:} Time permitting (e.g., after studio), you should write a loop that iterates over (odd) values of \( k \) in the range of, say, 1 to 25, and plots their accuracies. You can use this plot to determine the best value of \( k \).
Scaling

The $k$-NN algorithm relies on a distance metric to find neighbors. The usual distance metric is Euclidean distance. The trouble with this (or any numeric) metric is that it downweights features whose range of measurements is smaller than others. For example, age dominates travel class (i.e., first, second, or third), so that classifications that take into account the former are barely influenced by the latter. The data must be comparable, which can be achieved either through normalization or standardization.

Write a normalize function that takes as input a vector $x$, and then subtracts the minimum value of $x$ from all entries and divides by the difference between the maximum and the minimum values. Or, write a standardize function that takes as input a vector $x$, and then subtracts the mean value of $x$ from all entries and divides by the standard deviation. Preprocessing your features before learning using either function should improve your accuracies. Confirm that it does.

Training vs. Testing data

The next issue we face is that these accuracies do not tell us what we actually want to know, namely how well our classifiers will perform “in the wild”. Why might that be? Think back to our discussions in lecture, and discuss the issue with your partner.

In order to gauge the potential performance of a classifier “in the wild”, it is recommended practice to separate your data set into two—a training set and a testing set. Let’s separate the titanic data set into training and test sets.

Test Sets

Before partitioning your data into training and test sets, its best to first shuffle them, to avoid any potential biases in their ordering. The dplyr library provides the sample_n function that allows you to sample $n$ rows from a data frame. Conveniently, you can also use this function to shuffle your data. In particular, if you run sample_n(table, nrow(table)), you will select all rows from a data frame in random order. Do that now on titanic..

shuffled <- sample_n(titanic, nrow(titanic))

Next, let’s partition shuffled into training and test sets. What’s the best way to go about this? A good rule of thumb is to set aside 20% of the data for testing, and to use the remaining 80% for training. Using this rule of thumb, let’s calculate our split sizes:

split <- 0.8 * nrow(shuffled)
We can now partition into training and testing sets:

```r
training <- shuffled[1 : split, ]
```

```r
test <- shuffled[(split + 1):nrow(shuffled), ]
```

*Note:* A simpler way of splitting the data into training and test data is to use the `createDataPartition` function in the `caret` package. We walked you the process manually just now so that you would understand what is going on behind the scenes, but feel free to use `createDataPartition` going forward. Here is an example; go to R’s Help window for explanation of the parameters.

```r
split <- 0.80
index <- createDataPartition(knn_label, p = split, list = FALSE)
training <- titanic[index, ]
```

```r
test <- titanic[-index, ]
```

And now that you have partitioned the data (possibly twice!) into training and test sets, you can retrain all of your classifiers on `training` and test their accuracy on `test`. For example, to build a new

\[ \text{kNN} \]

-NN classifier, you should proceed as follows:

```r
knn_train <- training %>% select(feature_1, feature_2, feature_3)
knn_test <- test %>% select(feature_1, feature_2, feature_3)
knn_label <- training$survived
```

*Hint:* Be sure your data are normalized or standardized!

Build

\[ \text{kNN} \]

-NN classifiers on this training and test set for a few values of

\[ k \]

But be careful when testing their accuracy; be sure you do so against `test$survived`!

Note the accuracies of your models now. Are they higher or lower than before? Do these new accuracies cause you re-assess how well your various models work? Which classifier experiences the greatest decrease in accuracy?

**Cross-validation**

Beyond partitioning our data *once* into training and test sets, on which we might happen to see unusually high or unusually low accuracies, we can better estimate the accuracy of our models by partitioning our data into many training and test sets. This process, whereby we carry out the above procedure (train on training data; test on testing data) multiple times for multiple partitions of the data, and then average the accuracy across all partitions, is called *cross-validation*. 
Next, we will estimate the accuracies of our classifiers using cross-validation.

For the \( \{k\}\)-NN models, we can use \texttt{knn.cv} function for cross-validation. For example:

```r
knn3_cv <- knn.cv(knn_train, knn_label, k = 3)
accuracy(knn3_cv, knn_label)
```

Repeat this for \( k = 1 \), \( k = 5 \), and \( k = 15 \). Which value of \( k \) is most accurate? Try a few more values of \( k \).

Around what value of \( k \) does accuracy peak?

Talk with your partner about how the accuracy of your models has changed compared to using only a single test set. Was there a large change? Does this make intuitive sense? What does this make you think about the usefulness of cross validation?

**caret and train**

Before you complete this studio, we want to draw your attention to a very powerful function within the \texttt{caret} library, namely \texttt{train}. Enter \texttt{?train} into the console to read a little bit about \texttt{train}.

The \texttt{train} function is an interface for calling all sorts of learning algorithms. That is, we need not call \texttt{lm} or \texttt{knn} specifically. Instead, we can call \texttt{train}, and pass it our preferred learning algorithm as an argument. For example:

```r
knn_model <- train(survived ~ age + fare + sex, data = training, method = "knn", trControl = trainControl(method = "cv"), preProcess = c("center","scale"), tuneLength = 20)
```

This call to \texttt{train} invokes \texttt{knn} on the training data using the features specified. Moreover, it carries out a cross validation (\texttt{trainControl(method = "cv"})) and it preprocesses the data by standardizing it (i.e., “center” subtracts the mean of a predictor from the predictor’s values, and “scale” divides the results by the standard deviation). The parameter \texttt{tuneLength} specifies how many values of the tuning parameters (in this case, \( k \)) to try.

The results of the \texttt{knn_model} are stored in \texttt{knn_model$results}. Enter \texttt{knn_model$results} to see them. Next, you can plot the standard deviation of the accuracy vs. value of \( k \).
qplot(knnFit$results[[1]][[1]], knnFit$results[[4]][[1]])

Do you observe a tradeoff between bias and variance? Where is the bias greatest, and where is the variance greatest? Retrain your knn_model a few times and replot the results, if the graph is not exactly expected, to look for a pattern.

**Analysis Questions**

Discuss the following questions with your partner, and jot down your answers (in your R markdown file, if you like!) so that you can have a brief discussion about them with one of the TAs.

- Which of the models best fit the data?
- Which variables seem to have the most influence? (If you don’t know the answer to this question offhand, just discuss with your partner how you might find out.)
- Should we ever evaluate classifier performance on the same data it was trained on? What if we don’t have very many data; then, what might we do instead?
- How does the Titanic sinking compare to modern natural disasters such as Hurricane Katrina. Briefly look at this study published by the American Medical Association on fatality rates across different demographics during Hurricane Katrina. Which groups suffered the highest fatality rates?

Here is another paper on Hurricane Katrina from the Journal of Social Science Research. The data analysis section starts on page 302.

- Are there similarities between the factors that affected Titanic fatality rates and Hurricane Katrina fatality rates? What factors are different between the two disasters?

**End of Studio**

When you are done please call over a TA to review your work, and check you off for this studio. If you do not finish within the two hour studio period, remember to come to TA office hours to get checked off.