Introduction to programming in R: classification

Erik Spence

SciNet HPC Consortium

29 March 2023

Today's slides

To find today's slides, go to the "Introduction to programming in R" page, under Lectures, "Classification".

https://scinet.courses/1277

Today's class

SciNet

Today we will visit the following topics:

- Classification algorithms, in general.
- Decision trees.
- Logistic regression.
- ROC curves.

Classification

SciNet

Classification is similar to regression, in a sense:

- You fit a model to data with known answers ($y = f(x_1, x_2, x_3, ...)$).
- You use the model to make predictions about new data.

But what do you do if the labels (y) are discrete? How do you deal with that?

- Data point y is either in category 1 or 2.
- You don't get points for putting y in category 1.5.

Classification algorithms are used to create models for separating data into known categories.

Classification problems

SciNet

Some classic classification problems:

- Bioinformatics - classifying proteins according to function.
- Medical diagnosis.
- Image processing:
  - what objects exist in an image?
  - hand-written text analysis.
- Text categorization:
  - Spam filtering
  - Sentiment analysis: is this tweet positive or negative?
- Language recognition.
- Fraud detection.

Input variables can be continuous, discrete, or both.

Classification approaches

SciNet

There are lots of classification approaches which one might use.

- Decision trees: analyze the features of the data and make 'decisions' about how to 'split' the data into uniform groups.
- Logistic regression: like linear regression, but now we fit a "yes/no" function to the data.
- Naive Bayes: a type of probabilistic analysis.
- kNN: k Nearest Neighbours; use the k nearest neighbours to a data point to predict the category of a new data point.
- Support Vector Machines: essentially a linear model of the data, used to separate groups.
- Neural networks: a weird algorithmic approach to using functions to categorize data.

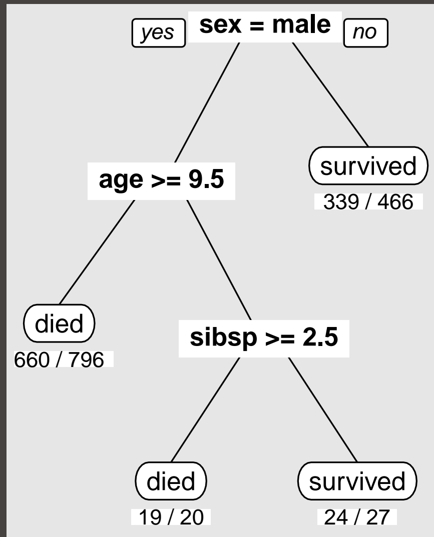
There isn't time to cover all of these. Today we'll cover Decision Trees and logistic regression.

Decision Trees

A Decision Tree is a structure which classifies an input based on a number of binary decisions.

It splits the data set based on one of the p "features" of the data.

"Features" are the independent variables associated with the data $(x_1, x_2, ..., x_p)$.



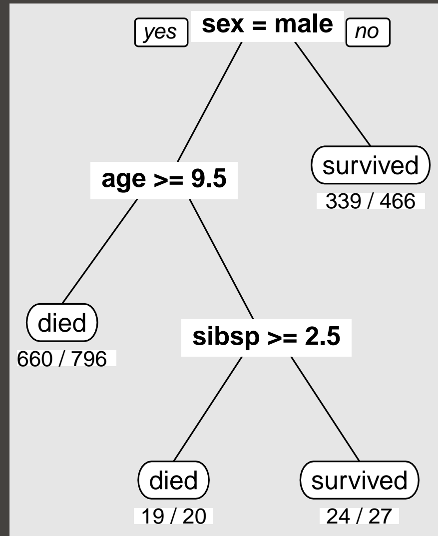
Decision Trees, continued

Data can be split based on

- discrete data ("if category == A") or,
- continuous data ("if height < 1.5m")

The goal of developing a decision tree is to determine when and where and how to split the data, so as to maximize the 'purity' of the resulting sub-data set.

A good decision tree will have "leaves" (ends of the tree) which are as pure as possible.



Splitting algorithms, continued

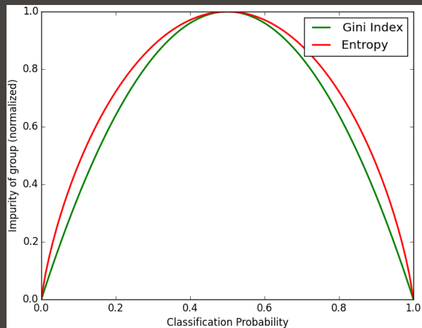
Algorithms which split the data rank possible splits based on increasing 'purity' of the two subgroups it generates.

Consider the probability p that a member of one of the labels is in a given target category. Two common measures for the 'impurity' of the generated groups are given by

Gini index: $\sum p(1-p)$

Entropy: $-\sum[p\ln p + (1-p)\ln(1-p)]$

Where the sum is over all labels and possible values in the given target category. A perfect Gini index is an impurity of 0, or a probability of 0 or 1.



Splitting algorithms, continued more

So how do these algorithms proceed?

- While every data point is not in a pure sub-tree:
  - For each feature which we haven't yet split upon, for the data remaining in the sub-tree, consider a split:
    - If the feature is categorical, consider all values, split by value and measure the impurity of the resulting subgroups.
    - If the feature is continuous, use line optimization to choose the best point at which to split, keeping track of the impurity at that point.
  - Choose the split which maximizes the change in the impurity (smallest impurity value), and split the data.

Let's use an R package to build a decision tree. We'll use the Iris data set.

- The data consists as four measurements of 150 wild irises of 3 species.
- It's a classic classification problem.
- It's one of the data sets which comes with R.
- We first randomly split the iris data set, 70/30, into training and test data sets.

```
>
> str(iris)
'data.frame':  150 obs.  of 5 variables:
 $Sepal.Length: num 5.1 4.9 4.7 4.6 5 ...
 $Sepal.Width : num 3.5 3 3.2 3.1 3.6 ...
 $Petal.Length: num 1.4 1.4 1.3 1.5 1.4 ...
 $Petal.Width : num 0.2 0.2 0.2 0.2 0.2 ...
 $Species     : Factor w/ 3 levels ...
>
> ind <- sample(c(T,F), nrow(iris),
+   replace = T, prob = c(0.7, 0.3))
> train.d <- iris[ind,]
> test.d <- iris[!ind,]
>
```

http://en.wikipedia.org/wiki/Iris_flower_data_set

Training versus Testing

In general, we get our data, and that's it. We don't have the luxury of generating more data on a whim.

We would like to do out-of-sample testing of whatever model we generate, to see how it does against new data. But we don't have any new data.

The solution is to hold out some of the original data. Most of the data is used for training the model, the rest is used for testing it. These data should be chosen randomly, as in the previous slide.

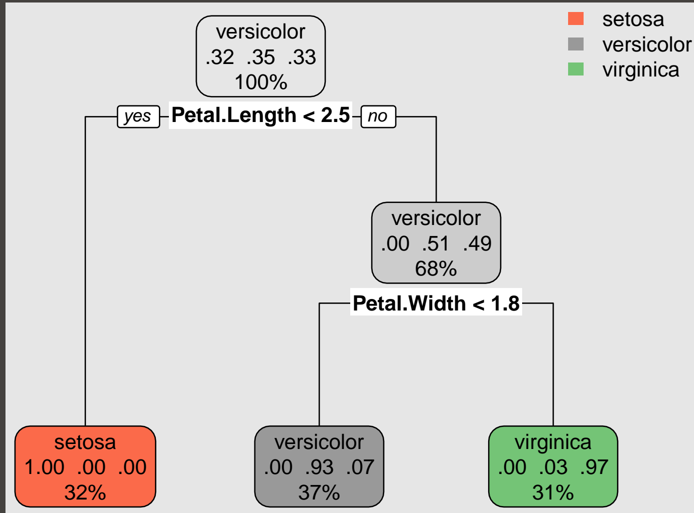
R analysis, an example, continued

Now that the data's split up, we're ready to generate the tree.

- Load the 'rpart' and 'rpart.plot' (non-standard) libraries.
- Create our formula (Note the simpler syntax which you can use).
- Generate the decision tree.
- Plot the result.

```
>
> library(rpart)
> library(rpart.plot)
>
> f <- Species ~ Sepal.Length + Sepal.Width +
+   Petal.Length + Petal.Width
>
> f <- Species ~ .    # same as above
>
> iris.tree <- rpart(f, data = train.d)
>
> rpart.plot(iris.tree)
>
```

Our decision tree



Confusion matrix

SciNet

How do you determine the effectiveness of a classifier? You can count the number incorrectly classified, but this doesn't give you much information you can use to improve the result.

The 'Confusion Matrix', tells you which misclassifications happened. Traditionally, 'true' classifications are on the rows, and predictions are on the columns.

```
> pred <- predict(iris.tree, type = 'class')
>
> sum(train.d$Species == pred) / nrow(train.d)
[1] 0.9611650
>
> table(train.d$Species, pred)
            setosa versicolor virginica
 setosa         36          0         0
 versicolor      0         35         1
 virginica       0          3        28
```

R analysis, an example, continued more

Ok, but how does the decision tree do on the test data?

- Test the built tree against the test data.
- Print out the table of results.
- Not bad!

```
>
> testPred <- predict(iris.tree,
+   newdata = test.d, type = 'class')
>
> sum(test.d$Species == testPred) / nrow(test.d)
[1] 0.957447
>
> table(test.d$Species, testPred)
             setosa versicolor virginica
 setosa          14          0         0
 versicolor       0         14         0
 virginica        0          2        17
>
```

Trees and over-fitting

SciNet

As with polynomials and regression, we can easily produce overly-complex decision trees which do great on the training data, but don't generalize.

This happens when the number of free (trainable) parameters in the model is similar to the number of data.

In fact, this is guaranteed to happen with decision trees, since given enough splits, it will always perfectly classify the data.

How do we deal with this? The usual approach is to prune the tree at some level, where the results are "good enough", and the model is not "too complex".

Random forests

SciNet

You may have heard of "random forests". What are those?

- Random forests fall under the category of "ensemble methods". This means an averaging over several machine-learning models.
- In this case, a random forest is an average over a collection of decision trees.
- To do this,
  - bootstrapping is applied to the data set in question, and decision trees are fit to each sample;
  - however, during the training of the trees, at each split only a subset of possible features are chosen as split candidates;
  - predictions on out-of-sample data are then generated, and an average over all trees is made.
- This results in a lowering of the overfitting, which is inherently large with decision trees.

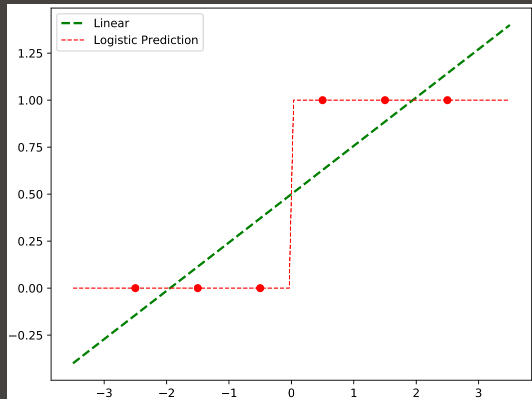
If you end up using decision trees in your research, random forests are worth considering.

Logistic regression

One way to consider binary classification is to go back to regression, and fit a linear regression to an integer 0/1 variable for classification: over 0.5, True, else False.

This requires a linear separation between the classes to be effective.

However, naive application of linear regression can lead to a number of problems, which grow with the number of dimensions. These are mostly related to the unbounded nature of the function.



Logistic regression, continued

SciNet

A whole infrastructure exists for "generalized linear models", where the function being fit is not

$$y = \beta_0 + x_1\beta_1 + x_2\beta_2 + \cdots = \mathbf{x} \cdot \vec{\beta}$$

but rather some power or exponential of $\mathbf{x} \cdot \vec{\beta}$.

Consider instead fitting, not the probability p, but rather the log of the odds ratio,

$$\mu = \ln\left(\frac{p}{1-p}\right) = \mathbf{x} \cdot \vec{\beta}$$

We can fit this log-odds equation, and derive

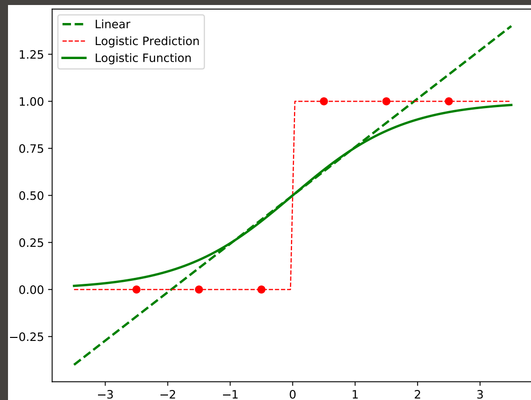
$$p = \frac{e^{\mathbf{x}\cdot\vec{\beta}}}{1 + e^{\mathbf{x}\cdot\vec{\beta}}} = \frac{1}{1 + e^{-\mathbf{x}\cdot\vec{\beta}}}$$

Logistic regression, continued more

$$p = \frac{1}{1 + e^{-\mathbf{x}\cdot\vec{\beta}}}$$

This approach has a number of very nice properties:

- We have a nice, bounded, well-behaved function.
- We can directly calculate the inferred probability of category membership.
- We're essentially fitting a Bernoulli process.

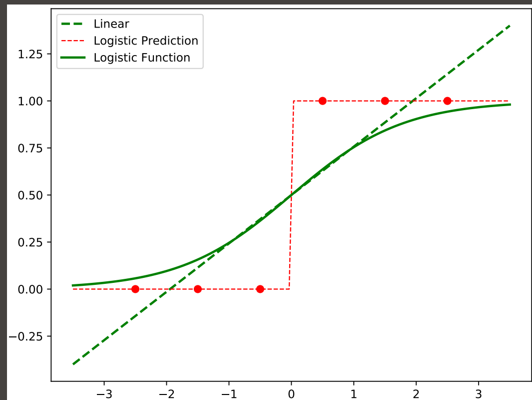


Logistic regression, continued some more

One has to use somewhat different numerical algorithms to fit these curves; typical curve-fitting algorithms deal very poorly with exponentials.

Techniques like expectation maximization (EM) or other well-conditioned iterative methods are often used.

That's fine; they're all hidden beneath whatever logistic or GLM packages you might want to use.



Logistic regression, example

Logistic regression in R is usually performed using a glm.

- You will need to install the 'mlbench' package to get this data.
- The 'complete.cases' function returns a boolean vector, indicating which rows have complete data.
- As always, we split into training and testing data.
- We specify "family = 'binomial'" when we want to perform logistic regression.

```
>
> data(BreastCancer, package = 'mlbench')
>
> bc <- BreastCancer[complete.cases(BreastCancer),]
>
> ind <- sample(c(TRUE, FALSE), nrow(bc),
+               replace = TRUE, prob = c(0.7, 0.3))
>
> train.d <- bc[ind,]
> test.d <- bc[!ind,]
>
> model <- glm(Class ~ Cl.thickness + Cell.size +
+      Cell.shape, family = 'binomial',
+      data = train.d)
>
```

Logistic regression, example, continued

Predictions using logistic regression need some postprocessing.

- Use the "type = 'response'" flag when using predict with logistic regression.
- The returned values are probabilities of 'success'. These must be converted to a classification.
- The ifelse function applies a conditional to each element, and returns a the first argument for a TRUE value, the second for FALSE.

```
> pred <- predict(model, newdata = test.d,
+                   type = 'response')
>
> new.pred <- ifelse(pred > 0.5,
+                      'malignant', 'benign')
>
> conf <- table(test.d$Class, as.factor(new.pred))
>
> conf
          benign malignant
benign       122         4
malignant      4        60
>
> sum(diag(conf)) / sum(conf)
[1] 0.9578947
>
```

Evaluating binary classifiers

SciNet

Binary classification is a common and important enough special case that its confusion matrix elements have special names, and various quality measures are defined.

Note that "Positive" here refers to "category 1" and "Negative" means "category 0".

	Classified Positive (CP)	Classified Negative (CN)
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

One can always get exactly one of FN or FP to be zero (for example, just classify everything positive, then there will never be any false negatives).

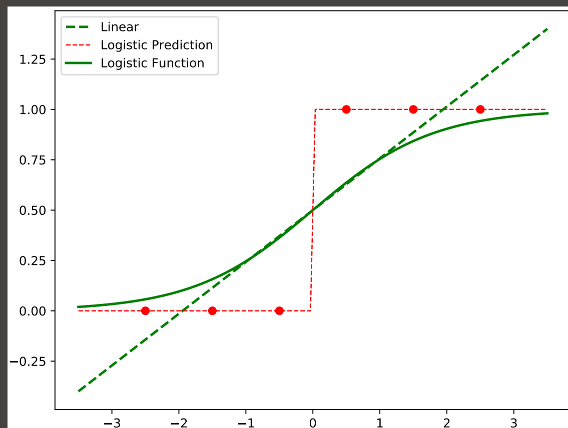
But there is usually a tradeoff between false positives and false negatives.

Classification thresholds

In most binary classifiers, there's some equivalent of a threshold you can set. This threshold determines when a given data point moves from one categorization to the other.

For the case of logistic regression, the default threshold is 0.5.

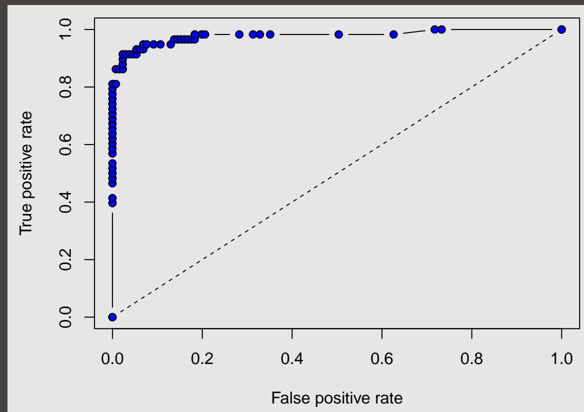
- Set it lower (allow more true, but also false, positives).
- Set it higher (allow more true, but also false, negatives).



ROC curve

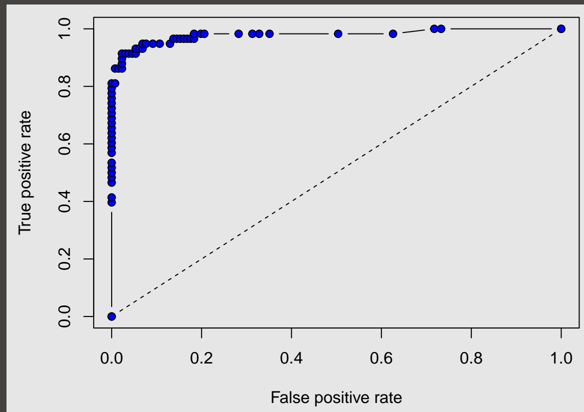
By varying the classification threshold, from 1 to 0, we can get a collection of points for the TPR and FPR. Plotting the two measures on either axis gives a ROC (Receiver Operating Characteristic) curve.

- The diagonal line represents random chance.
- We want our curve to be as high above the diagonal as possible.



ROC curve, continued

```
>
> library(ROCR)
>
> ROC.pred <- prediction(pred, test.d$Class)
> ROC.perf <- performance(ROC.pred,
+                         measure = 'tpr',
+                         x.measure = 'fpr')
>
> plot(ROC.perf, type = 'b', pch = 21,
+      bg = 'blue')
>
> lines(c(0, 1), c(0, 1), lty = 2)
>
```



Note that your curve will look different from this one, due to randomness.

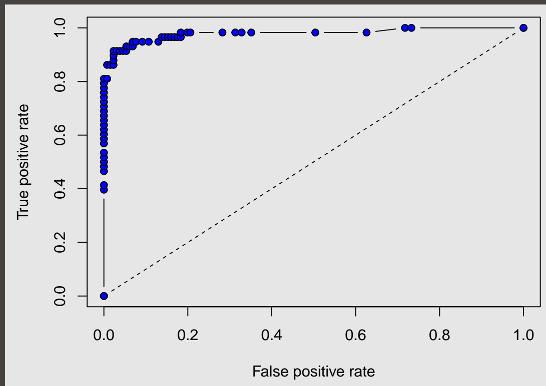
ROC curve, continued more

The quality of a classifier is given by the ROC curve's AUC (area under the curve).

- The worst classifiers will have an AUC near 0.5.
- Good classifiers have an AUC near 1.0.

For the non-binary classification situation, you create "one versus all" ROC curves, with one ROC curve for each category.

```
>
> ROC.AUC <- performance(ROC.pred,
+                        measure = 'auc')
> ROC.AUC@y.values
[[1]]
[1] 0.9786706
```



Summary

SciNet

Things to remember from today:

- Decision tree strength: can sensibly deal with categorical data.
- Decision tree strength: preform implicit feature selection.
- Decision tree strength: easy to understand (and explain) the results.
- Decision tree weakness: prone to over-fitting.
- Logistic regression:
  - not prone to over-fitting.
  - can work well with noisy data.
  - assumes (requires) there is a single smooth boundary between categories.
- Note that there are other classification algorithms out there: kNN, naive Bayes, support vector machines, *etc.*