Classification I

Introduction to Computational BioStatistics with R

Alexey Fedoseev

November 11, 2025

Classification in Machine Learning

Classification is a central topic in machine learning. Classification is similar to regression, but deals with discrete target labels. Examples of these labels are:

- Yes or No
- Male or Female
- Salary brackets: below \$40k, \$40k-\$60k, \$60k-\$80k, \$80k-\$100k, \$100k+

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

Classification Problems

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

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 kNN.

Decision Trees in R

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, 2/3 and 1/3, 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(TRUE, FALSE), nrow(iris),
+   replace = T, prob = c(0.66, 0.33))
> 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.

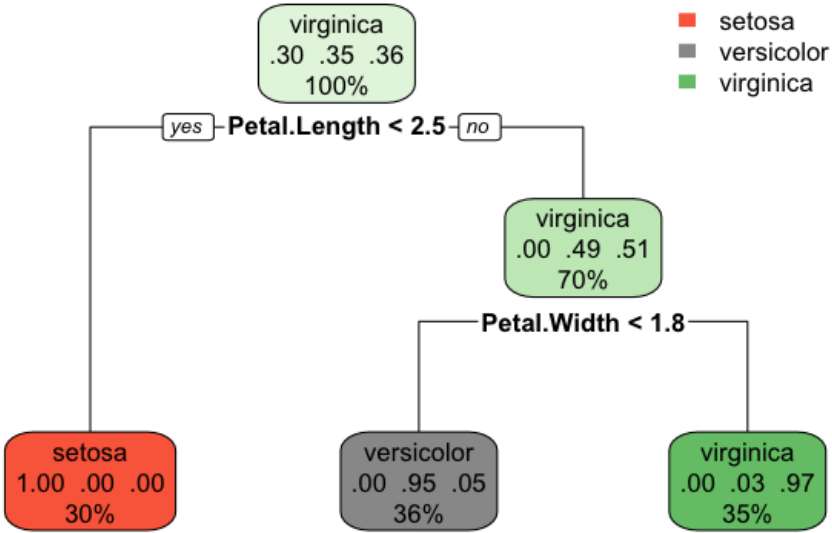
Fitting the Decision Tree Model

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)
```

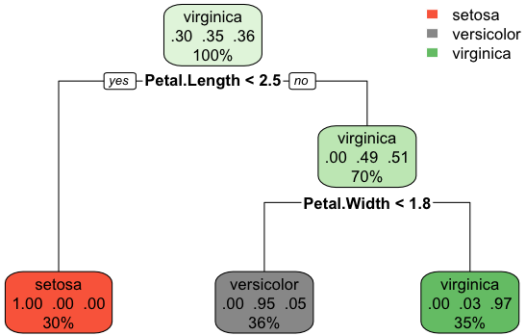
Visualizing the Iris Decision Tree



Traversing the Decision Tree

You found an iris flower and you want to classify it. Start at the root node (depth 0).

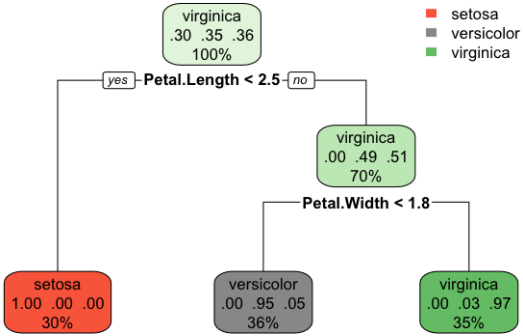
- Measure the petal length of flower
- If it is smaller than 2.5 cm you move down to the left node (depth 1)
- The left node of the Decision Tree predicts that your flower is Iris-Setosa.



Traversing the Decision Tree

Let's say you have another flower you want to classify. You start at the root node (depth 0).

- Measure the petal length of flower
- If it is greater than 2.5 cm you move down to the right node (depth 1)
- Is the petal width smaller than 1.8 cm?
- If it is, then your flower is most likely an Iris-Versicolor (depth 2, left)
- If not, it is likely an Iris-Virginica (depth 2, right).



Confusion Matrix

How to determine the effectiveness of a classifier?

A good way to evaluate the performance of a classifier is to look at the confusion matrix. The general idea is to count the number of times instances of class A are classified as class B.

```
> pred <- predict(iris.tree, type = 'class')
> sum(train.d$Species == pred) / nrow(train.d)
[1] 0.9719626
> table(train.d$Species, pred)
```

	setosa (predicted)	versicolor (predicted)	virginica (predicted)
setosa (actual)	32	0	0
versicolor (actual)	0	36	1
virginica (actual)	0	2	36

Out-of-Sample Evaluation

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$Species == testPred)/nrow(test.d)
[1] 0.9302326
>
> table(test.d$Species, testPred)
             setosa versicolor virginica
  setosa         18          0         0
  versicolor      0         13         0
  virginica       0          3         9
```

Over-fitting in Decision Trees

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

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.

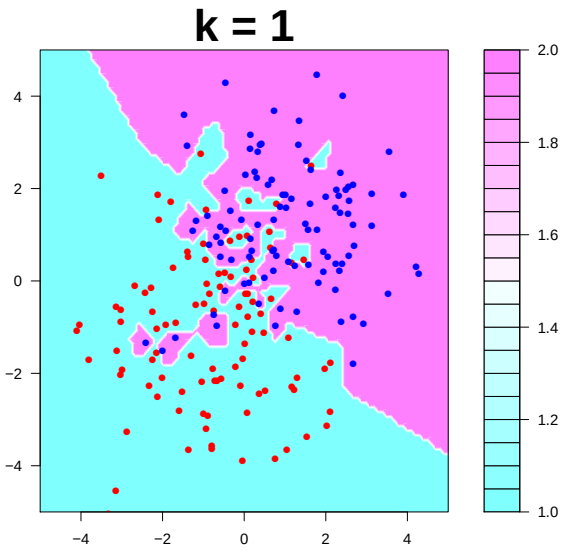
What is kNN?

Consider a more-geometric approach to classification: given an input data point, find the nearest point in the training set, and choose that classification for your input data point.

This is a type of regression.

A generalization is to choose the k Nearest Neighbours (kNN), and choose the classification that the majority of those k points has.

This case: two 2D Gaussians, centred on (-1,-1) (red), and (1,1) (blue) with $\sigma = 1.5$, $k = 1$.



Nearest Neighbours - kNN, Continued

```
library(class); n <- 100

# Generate Gaussian data.
x1 <- rnorm(n, -1, 1.5); y1 <- rnorm(n, -1, 1.5)
x2 <- rnorm(n, 1, 1.5); y2 <- rnorm(n, 1, 1.5)

# Generate mesh for plotting.
x.range <- seq(-5, 5, length = n)
x3 <- rep(x.range, n); y3 <- rep(x.range, each = n)

# Put the data in data frames.
my.data <- data.frame(x = c(x1, x2), y = c(y1, y2))
grid.data <- data.frame(x = x3, y = y3)

# Create labels for the data.
labels <- rep(c(1, 2), each = n)
```

Nearest Neighbours - kNN, Continued More

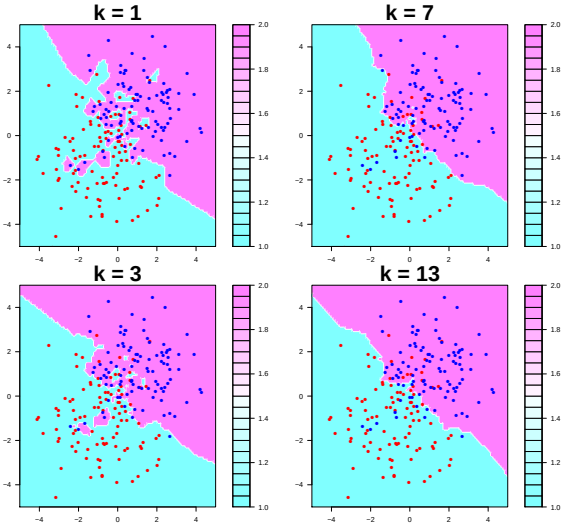
```
# Make predictions, based on the training data.
p <- as.integer(knn(train = my.data,
                    test = grid.data, cl = labels))
# Make the result 2D.
dim(p) <- c(n, n)
# Make the plot.
filled.contour(x.range, x.range, p,
               plot.axes = { axis(1); axis(2);
                 points(x1, y1, pch = 16,
                        col = "red", xlim = c(-5, 5),
                        ylim = c(-5, 5), ann = F);
                 points(x2, y2, pch = 16,
                        col = "blue", xlim = c(-5, 5),
                        ylim = c(-5, 5), ann = F)
               })
```

Bias-Variance in kNN

There's a bias-variance-like trade-off in kNN, as can be seen by varying k on the same data.

At low k, the variance is very large. The model is trying to fit to every single point.

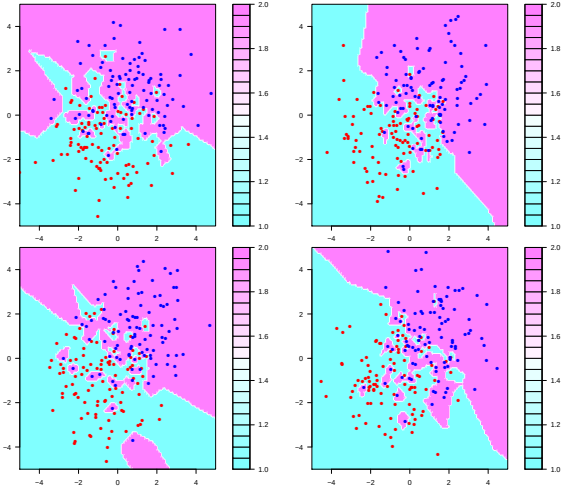
At higher k, we average over a large area, and we start to lose features.



Bias-Variance in kNN, Continued

On the right we see 4 different datasets drawn from the same population as before. The model has been built with $k = 1$ for all 4.

It's clear that the decision boundary varies widely from one run to the next. This is another symptom of overfitting.



Scaling continuous features

In the iris data set, petal length varies over a much greater range than sepal width. If we just use Euclidean distance for kNN, sepal width will provide very little information: all points are close to each other in that dimension.

We want the information in all variables to contribute to the solution. To this end, we should scale the variables to that they all get to play. A common technique is to centre the variables by subtracting off their means, and then scaling them by their standard deviations.

$$x' = \frac{x - \mu}{\sigma_x}$$

Many libraries will do this for you, for methods where it matters. But not all will; check the documentation!

The Caret Package

The caret package has many many Machine Learning algorithms built into it, including kNN. It has **cross-validation** built-in for model evaluation.

It can be used to determine the most-important features of the data.

It will also select the best value for k.

The 'tuneLength' is the number of values of k caret will consider.

```
> library(caret)
>
> ind <- sample(c(T,F), nrow(iris),
+   replace = T, prob = c(0.7, 0.3))
> train.d <- iris[ind,]
> test.d <- iris[!ind,]
>
> fit.control <- trainControl(method = 'cv'
+   number = 10)
>
> knnFit <- train(Species ~ .,
    data = train.d,
    method = 'knn',
    preProcess = c('center', 'scale'),
    trControl = fit.control,
    tuneLength = 20)
```

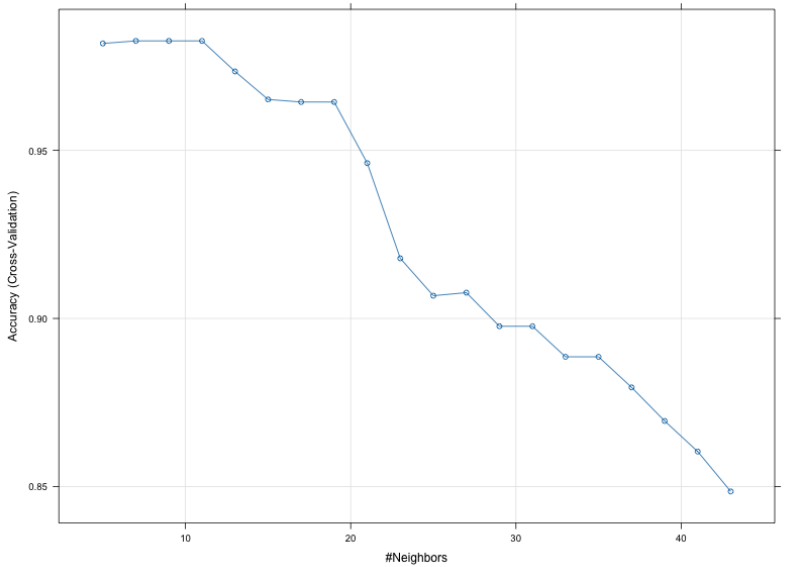
The Caret Package, Continued

```
> knnFit
  ...
Pre-processing: centered (4), scaled (4)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 96, 95, 94, 95, 96, 95, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.9818182  0.9723259
  ...
  43  0.8485354  0.7711114

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 11.
> plot(knnFit)
```

The Caret Analysis



The Caret Package, Continued

```
> knnPredict <- predict(knnFit, newdata = test.d)
> table(knnPredict, test.d$Species)
knnPredict   setosa versicolor virginica
  setosa         15          0         0
  versicolor      0         12         2
  virginica       0          2        13
>
> varImp(knnFit)
ROC curve variable importance

  variables are sorted by maximum importance across the classes
             setosa versicolor virginica
Petal.Width  100.00     100.00    100.00
Petal.Length 100.00     100.00    100.00
Sepal.Length  98.25      58.21     98.25
Sepal.Width   40.30      40.30      0.00
```

Summary

Things to remember from today:

- Decision tree strength: can sensibly deal with categorical data.
- Decision tree strength: perform implicit feature selection.
- Decision tree strength: easy to understand (and explain) the results.
- Decision tree weakness: prone to over-fitting.
- kNN strength: completely non-parametric (data can take any form).
- kNN strength: works in as many dimensions as you like.
- kNN weakness: slow if there are too many data points.
- kNN weakness: doesn't handle categorical data.

Note that there are other classification algorithms out there: logistic regression, naive Bayes, support vector machines, *etc.*