lazyIris examples

Phil

2015-09-04

Contents

Overview

lazyIris is a small implementation of k-nearest neighbours applied to the famous iris dataset.

Installing

First, ensure that the devtools package is installed and then install directly from the package github repository.

```
# check for and install devtools.
if(!("devtools" %in% rownames(installed.packages())))
  install.packages("devtools")

# install and load.
if(!("lazyIris" %in% rownames(installed.packages())))
  devtools::install_github("phil8192/lazy-iris")
require(lazyIris)
```

Loading data

The package has preprocessed iris data attached.

```
attach(iris.data)
```

```
## The following objects are masked from iris.data (pos = 3):
##
##     petal.length, petal.width, sepal.length, sepal.width, species
```

Example data may be loaded from the inst/extdata directory by using the *loadData* function. In addition, the *checkData* function will perform any necessary data sanity checks.

```
iris.data <- checkData(loadData())
```

```
## Warning in checkData(loadData()): removed duplicated rows.
```

```
##   cleaned data...
##   sepal.length    sepal.width     petal.length   petal.width
##  Min.   :4.300   Min.   :2.000   Min.   :1.00   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.60   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.40   Median :1.300
##  Mean   :5.856   Mean   :3.056   Mean   :3.78   Mean   :1.209
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.10   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.90   Max.   :2.500
##             species
##  Iris-setosa    :48
##  Iris-versicolor:50
##  Iris-virginica :49
##
##
##
```

A quick look at the data

The dataset consists of 4 features and 3 possible classes. Some of the features are highly correlated:

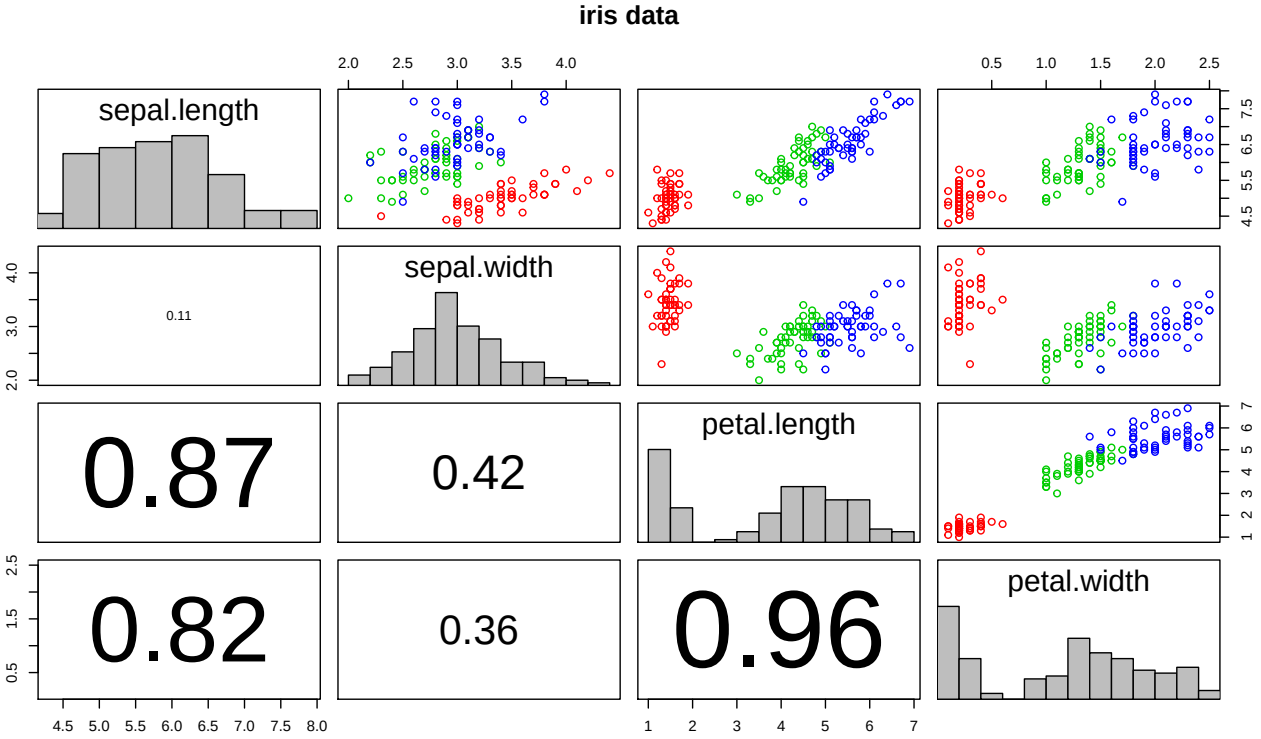
```
cor(iris.data[, 1:4])
```

	sepal.length	sepal.width	petal.length	petal.width
sepal.length	1.0000000	-0.1093208	0.8713046	0.8170583
sepal.width	-0.1093208	1.0000000	-0.4210574	-0.3563762
petal.length	0.8713046	-0.4210574	1.0000000	0.9618828
petal.width	0.8170583	-0.3563762	0.9618828	1.0000000

Iris data visualisation

The package provides a means to visualise the relationship between the 4 features and the corresponding class.

```
# plot all the data.
visualise(iris.data, class.name="species", main="iris data", plot.hist=TRUE,
    plot.cor=TRUE)
```



In the above visualisation, the colours correspond to the classification of the species of iris plant:

colour	species
red	Iris setosa
green	Iris versicolour
blue	Iris virginica

The lower left panels show the correlation between the 4 iris features, the diagonal panels contain a histogram of the distribution of each feature, and finally, the upper right panels contain scatter plots of each possible feature combination colour coded by species.

Querying the data

The *knn* function makes it possible to query the data for neighbouring instances given an arbitrary list of features.

10-Nearest neighbours

The following example obtains the *top 10* nearest neighbours to query:

```
# form the query.
# in this example, the feature values are actually the mean values in the
# dataset, thus the results may be interpreted as the top 10 "most average"
# instances.

query <- list(
    sepal.length=5.84,
    sepal.width=3.05,
    petal.length=3.76,
    petal.width=1.20)

# obtain the nearest-neighbours.
top.10 <- knn(query, iris.data, 10)
print(top.10, row.names=FALSE)
```

sepal.length	sepal.width	petal.length	petal.width	species	distance
5.6	2.9	3.6	1.3	Iris-versicolor	0.3401470
5.8	2.7	3.9	1.2	Iris-versicolor	0.3790778
5.6	3.0	4.1	1.3	Iris-versicolor	0.4309292
6.1	2.8	4.0	1.3	Iris-versicolor	0.4446347
5.7	2.8	4.1	1.3	Iris-versicolor	0.4557412
5.7	3.0	4.2	1.2	Iris-versicolor	0.4644351
5.7	2.9	4.2	1.3	Iris-versicolor	0.4956813
5.8	2.6	4.0	1.2	Iris-versicolor	0.5115662
5.8	2.7	4.1	1.0	Iris-versicolor	0.5288667
5.9	3.0	4.2	1.5	Iris-versicolor	0.5382379

Classification

In addition to the N-nearest neighours, the function also returns the distance from the query point. This distance can be used to predict the most likely class of the query point using the *classifier* function.

```
query <- list(sepal.length=5.84, sepal.width=3.05,
              petal.length=3.76, petal.width=1.20)
top.10 <- knn(query, iris.data, k=10)
prediction <- classifier(top.10$species, top.10$distance)
print(paste("prediction =", prediction$pred,
            "confidence =", prediction$conf))
```

```
## [1] "prediction = Iris-versicolor confidence = 1"
```

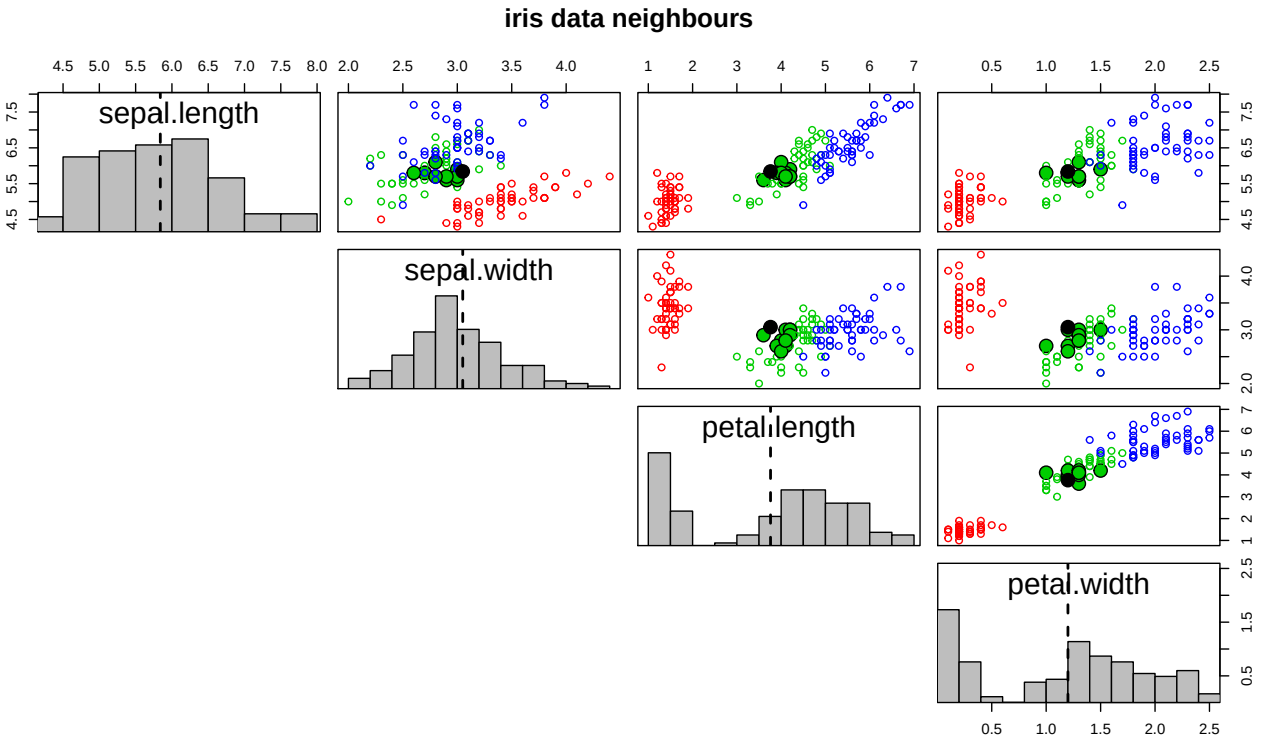
Visualising a query result

Given a list of nearest neighbours returned from the *knn* function, it is possible to visualise the query point and it's nearest neighbours over all dimensions in the feature space by using the *visualise* function.

```
# form the query and obtain the nearest neighbours.
q <- list(sepal.length=5.84, sepal.width=3.05,
          petal.length=3.76, petal.width=1.20)
```

```
top.10 <- knn(query, iris.data, k=10)

# visualise the result.
visualise(iris.data, class.name="species", query=q, neighbours=top.10,
    main="iris data neighbours", plot.hist=TRUE, plot.cor=FALSE)
```



In the above plot, the query point is shown as a black point. The resulting neighbours from the *knn* query are highlighted (opaque) circles. In addition, the query point with respect to the distribution of features has been highlighted with a black dashed vertical line over the corresponding feature histograms.