Data Mining Algorithms In R/Dimensionality Reduction/Feature Selection

Contents

Feature Selection in R with the FSelector Package

Introduction

In Data Mining, Feature Selection is the task where we intend to reduce the dataset dimension by analyzing and understanding the impact of its features on a model. Consider for example a predictive model $C_1A_1 + C_2A_2 + C_3A_3 = S$, where C_i are constants, A_i are features and S is the predictor output. It is interesting to understand how important are the used features (A_1, A_2 and A_3) and what are their relevance to the model and their correlation with S. Such analysis allow us to select a subset of the original features, reducing the dimension and complexity of future steps on the Data Mining process. During a subset selection, we try to identify and remove as much of the irrelevant and redundant information as possible.

Techniques for Feature Selection can be divided in two approaches: **feature ranking** and **subset selection**. In the first approach, features are ranked by some criteria and then features above a defined threshold are selected. In the second approach, one searches a space of feature subsets for the optimal subset. Moreover, the second apprach can be split in three parts:

1. **Filter approaches:** you select the features first, then you use this subset to execute a classification algorithm;
2. **Embedded approaches** the feature selection occurs as part a classification algorithm;
3. **Wrapper approaches** an algorithm for classification is applied over the dataset in order to identify the best features.

The FSelector Package for R offers algorithms for filtering attributes (e.g. cfs, chi-squared, information gain, linear correlation) and algorithms for wrapping classifiers and search attribute subset space (e.g. best-first search, back-ward search, forward search, hill climbing search). The package also makes it possible to choose subsets of features based on attributes' weights by performing different ways of cutoff.

The FSelector Package was created by Piotr Romanski and this article is based on the version 0.16, released in April 11, 2009.

The Feature Ranking Approach

As we explained, in the ranking approach, features are ranked by some criteria and those which are above a defined threshold are selected. A general algorithm can be considered for such approach where you just need to decide which one if the best ranking criteria to be used. In the F-Selector Package, such criteria is represented by a set of functions that calculate weights to your features according to a model. All of this elements will be explained in this text.

Feature Ranking Algorithm

The general algorithm for the Feature Ranking Approach is:

```
for each feature F_i
   wf_i = getFeatureWeight(F_i)
   add wf_i to weight_list
sort weight_list
```

```
choose top-k features
```

We can translate such algorithm idea to R language by these commands:

```
#load a dataset and use it as the main source of data
library(mlbench)
data(HouseVotes84)

#calculate weights for each atribute using some function
weights <- SOME_FUNCTION(Class~., HouseVotes84)
print(weights)

#select a subset of 5 features with the lowest weight
subset <- cutoff.k(weights, 5)

#print the results
f <- as.simple.formula(subset, "Class")
print(f)
```

On the first part of the code above, we use the function *library* to load the package *mlbench* which contains a collection of artificial and real-world machine learning benchmark problems, including, e.g., several data sets from the UCI repository.(http://cran.r-project.org/web/packages/mlbench/index.html). After that, we use the *mlbench* dataset *HouseVotes84* (United States Congressional Voting Records 1984) as the working data source for the later steps. Instead of using the *HouseVotes84*, you can also define your own dataset and provide it as the input for the algorithm.

The *HouseVotes84* data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA contains 16 variables, and consider ninve different types of votes represented by three classes: yea (voted for, paired for, announced for), nay (voted against, paired against, announced against) and unknown (voted present, voted present to avoid conflict of interest, did not vote or otherwise make a position known).

On the second part of the code above, we calculate weights for each attribute using some function. This function must be selected by the user and they are listed later on this text. The output of those functions will be a list of features and its weights according to the chosen technique, and that will be available in the *weights* array. You can print the weights like we do using the command *print*.

On the third part of the code, we define a cut-off of the top-5 features of the *weights* array. By using the function *cutoff.k*, we inform the array name and the *k* value, that is 5 in our case. The result will be stored in another array, called *subset*, cointaining the 5 features with the highest rank weight.

On the fourth part of the code, you can print the final model as a simple formula, composed by the features subset present in the *subset* array, and the predictable feature names *Class*.

Available Feature Ranking Techniques in FSelector Package

All the listed techniques below can be used to generate rank weights for features. The first parameter, *formula*, is a symbolic description of a model (e.g. Class~. for a model where all the features are used do predict the feature *Class*). The second parameter, *data*, is the target data to be considered in the model.

Chi-squared Filter

Usage:

```
chi.squared(formula, data)
```

Correlation Filter

Usage for Pearson's correlation:

```
linear.correlation(formula, data)
```

Usage for Spearman's correlation:

```
rank.correlation(formula, data)
```

Entropy-Based Filter

Usage for Information Gain:

```
information.gain(formula, data)
```

Usage for Gain Ratio:

```
gain.ratio(formula, data)
```

Usage for Symmetrical Uncertainty:

```
symmetrical.uncertainty(formula, data)
```

OneR Algorithm

Usage:

```
oneR(formula, data)
```

Random Forest Filter

Usage:

```
random.forest.importance(formula, data, importance.type = 1)
```

Where the *importance.type* parameter specify the type of importance measure (1=mean decrease in accuracy, 2=mean decrease in node impurity).

The Feature Subset Selection Approach

In the feature subset selection approach, one searches a space of feature subsets for the optimal subset. Such approach is present on the FSelector package by wrappers techniques (e.g. best-first search, back-ward search, forward search, hill climbing search). Those techniques works by informing a function that takes a subset and generate an evaluation value for that subset. A search is performed in the subsets space until the best solution can be found.

Feature Subset Selection Algorithm

The general algorithm for the Feature Subset Selection approach is:

```
S = all subsets
```

```
for each subset s in S
   evaluate(s)

return (the best subset)
```

We can translate the algorithm idea in R by using these commands:

```
#load a dataset and use it as the main source of data
library(mlbench)
data(HouseVotes84)

#define the evaluation function
evaluator <- function(subset) {
  #here you must define a function that returns a double value to evaluate the given subset
  #consider high values for good evaluation and low values for bad evaluation.

  return(something)
}

#perform the best subset search
subset <- MY_FUNCTION(data, evaluator)
```

```
#prints the result
f <- as.simple.formula(subset, "Class")
print(f)
```

As in the first example on this text, we use again the HouseVotes84 dataset from the mlbench library. We must define a evaluation function that wil do some calculus over a given subset and return a high value for a good subset. Later, all you have to do is choose a search strategy and you can also print the selection.

Feature Subset Search Techniques Available in FSelector Package

The FSelector Package offers some functions to search for the best subset selection. In most of them, the *attributes* parameters is a character vector of all attributes to search (the set of features that will be parted in subsets during the search), and the *eval.fun* parameter is a function taking as first parameter a character vector of all attributes and returning a numeric indicating how important a given subset is. The CFS and the Consistency techniques encapsulate such process by using the best first approach, so, you just have to inform the symbolic model and the data, like in the ranking approach.

Best First Search

Usage:

```
best.first.search(attributes, eval.fun)
```

Exhaustive Search

Usage:

```
exhaustive.search(attributes, eval.fun)
```

Greedy Search

Usage for forward greedy search:

```
forward.search(attributes, eval.fun)
```

Usage for backward greedy search:

```
backward.search(attributes, eval.fun)
```

Hill Climbing Search

Usage:

```
hill.climbing.search(attributes, eval.fun)
```

CFS Filter

Usage:

```
cfs(formula, data)
```

Consistency Based Filter

Usage:

```
consistency(formula, data)
```

Retrieved from "http://en.wikibooks.org/w/index.php?title=Data_Mining_Algorithms_In_R/Dimensionality_Reduction/Feature_Selection&oldid=2719378"
