Article from

The Modeling Platform

December 2016
Issue 106

Roll Your Own Cluster Model

By Bob Crompton

Model efficiency is an important area of model management, and model compression is one of the dimensions of such efficiency. Model compression improves efficiency by creating a significantly reduced number of model points compared to a seriatim model. Cluster modeling is a model compression methodology that has been successfully implemented for a number of years.

A good introduction to cluster modeling can be found in the article "Cluster Analysis: A Spatial Approach to Actuarial Modeling."[1] For simplicity, this article is referred to as "the Milliman article." Some actuarial software incorporates cluster modeling; if yours doesn't, this article is for you.

SOFTWARE USED

The clustering in this article is performed with the open source software R.[2] In addition to the software included in the standard installation, I used two additional packages—**xlsx** and **fpc**—to perform some of the tasks discussed in this article. To install these packages, use the R console commands:

```
install.packages("xlsx", dependencies = TRUE)

install.packages("fpc", dependencies = TRUE)
```

Since these packages are not part of the home library, they will need to be added. They can be manually loaded as follows:

```
library(xlsx)

library(fpc)
```

CREATING A CLUSTER MODEL

I obtained from a colleague an in-force file of universal life (UL) policies. There are five plan types in the file and 1,347 records.

The broad steps we need to create a cluster model are:

- Generate synthetic policy attributes.
- Apply weighting to the attributes as appropriate.
- Split data into segments.
- Import the file of in-force attributes to R.
- Apply the clustering algorithm.
- Export results.
- Reconfigure in-force files for the reduced number of cells and rerun the model.

These steps are considered in this article.

GENERATE SYNTHETIC POLICY ATTRIBUTES

One of the most interesting aspects of the Milliman article is the use of synthetic policy attributes—that is, policy attributes that are not found either in the in-force file or are not simple transformations of data found in the in-force file. Quinquennial ages are examples of simple transformations of in-force data.

The development of clusters uses attributes associated with projected cash flows in addition to the attributes found in the in-force file. For example, the life/health model used in the Milliman article includes the following synthetic attributes:

- Present value of proxy profits
- Present value of proxy profits through 10 projection years
- Present value of proxy profits through 20 projection years

There are three other attributes used in this model, of which two are synthetic. The only native attribute is beginning reserve.

It is instructive to review the synthetic attributes for the term life model included in the Milliman article:

- Beginning reserve
- Cumulative present value of proxy cash flows
- Present value of proxy cash flows
  - Years 1–5
  - Years 6–10
  - Years 11–15
  - Years 16–20
  - Years 21–25
  - Years 26–30
- Projected death benefits
  - Years 1–5
  - Years 6–10
  - Years 11–15
  - Years 16–20
  - Years 21–25
  - Years 26–30
- Projected premiums
  - Years 1–5
  - Years 6–10
  - Years 11–15
  - Years 16–20
  - Years 21–25
  - Years 26–30

This is 20 attributes, of which only one is native, with all the rest being synthetic. Why are there so many? The complexity of reserving for level term, combined with the mortality patterns during and after the level term period, mean extensive information is required if we want a well-fitting cluster model.

The attributes I used for my model were those used in the Milliman article for the traditional life/health model with one exception. I did not include the present value of total proxy profits because my original projections only went for 20 years. The information contained in the early years' proxy profits and the later years' proxy profits overlaps the information contained in total proxy profits.

APPLY WEIGHTS TO ATTRIBUTES

Weights adjust the relative importance of the various attributes used for clustering. Weighting affects both the selection of the representative cell for a cluster as well as the cells assigned to the cluster.

Consistent with the cluster attributes, the weights I used for my model were those used in the Milliman article for the traditional life/health model.

CREATE SEGMENTS

The in-force file needs to be split into segments. The Milliman article describes segments as follows:

> You divide the business into segments, which instructs the program not to map across segment boundaries. Segments might include plan code, issue year, GAAP [generally accepted accounting principles] era or any other dimension of interest.

So clustering is applied at the segment level. In my example, the entire file is treated as one segment. If I had used multiple segments, they would have been based on plan code.

IMPORTING THE DATA INTO R

For the import operation, I am demonstrating how to import from Excel because Excel is ubiquitous. I will demonstrate two possibilities because there is more than one way to skin a cat. (I don't actually know this from personal experience, but generations of folk wisdom attest to the truth of this statement. Who am I to question generations of folk wisdom?)

Importing Directly from Excel

For the first import operation, I use the `read.xlsx` function. The `read.xlsx` function is from the **xlsx** package.

Use the following console command:

```
MyInforce <- read.xlsx("c:/Inforce.xlsx", 1)

#  2nd parameter indicates which tab to import
```

MyInforce, the variable containing the output from the `read.xlsx` operation, is a data frame. Data frames are formatted matrix-like data structures. They are convenient since they can be used in many built-in functions that require matrix input.

Importing From a Comma-Separated File

For this import operation, I use the `read.table` function:

```
MyInforce <- read.table("c:/Inforce.csv", header
= TRUE, sep = ",")
```

The `read.table` operation yields a data frame, the same as the `read.xlsx` operation.

Comments on Excel vs. CSV Files

The `read.xlsx` function has the obvious benefit of convenience. You are working directly from Excel so you don't have to reformat. In addition, if you put each segment in a separate tab, it is easy to loop through the tabs with the `read.xlsx` function.

DIY clustering is an easy and straightforward process using existing code.

However, there are some serious drawbacks to using the `read.xlsx` approach:

- It's s-l-o-w! An Excel file with 10,000 records took about 35 minutes to load using `read.xlsx`.
- It's memory intensive. My computer was unable to load a file of 25,000 records because the Java back-end to **xlsx** ran out of memory.

Unless all of your segments are less than 2,000 or so records, reading directly from Excel may not be in the cards. Note that the R manual on data import/export has the following warning:

> The most common R data import/export question seems to be "how do I read an Excel spreadsheet."
>
> … The first piece of advice is to avoid doing so if possible! If you have access to Excel, export the data you want from Excel in tab-delimited or comma-separated form, and use `read.delim` or `read.csv` to import it into R.[3]

However, if you are committed to using Excel as your data source, the manual contains a description of a few other R packages that provide Excel import capability. Experiment with some or all of these to see if they work any better than the **xlsx** package.

CSV files don't have either of these problems. I was able to load a file of 100,000 records in less than 5 seconds using the `read.table` function. The only issue I have noted with CSV files is that if you forget to reformat comma-separated numeric values, chaos and darkness will result.

APPLY CLUSTERING ALGORITHM

Once the data is in R, it is simple to create clusters. There are a number of cluster functions available. I used the `pam` function because the output contains the information we need to create the clusters. `Pam` is based on a version of the K-means approach to clustering. The following code is for the cluster model with 13 cells. Note that the data is standardized by setting one of the function parameters shown. Standardization often gives better results when using clustering algorithms.

```
fit <- pam(MyInforce, 13, stand = TRUE)
```

The output variable `fit` is a list containing, among other things, the index number of the representative cells for each cluster in the component `id.med`, and the cluster assignment of each of the in-force records in the component `clustering`. So the component `id.med` is a vector with length equal to the number of clusters (in this instance, 13), and the component `clustering` is a vector with length equal to the number of in-force records.

Table 1
Comparison of Policy Attributes ($1,000s)

Attribute	Original	Clustered	Ratio
Initial reserve	522,352.9	523,339.7	100.2%
Projected first-year premiums	77,247.2	79,570.8	103.0%
Projected first-year benefits	38,000.4	40,133.5	105.6%
Present value proxy profits, years 1–10	125,481.3	128,489.8	102.4%
Present value proxy profits, years 11–20	(104,775.4)	(109,964.9)	105.0%

We pass these back from R using the following export code:

```
write.table(fit$id.med, "c:/Cluster_index.csv",
sep = ",")

write.table(fit$id.clustering, "c:/Cluster_
assignment.csv", sep = ",")
```

It is possible to use the `write.xlsx` approach to write directly to Excel files; however, the same caveats regarding speed and memory mentioned in the discussion of importing the data will apply to exporting the data as well.

RECONFIGURING THE IN-FORCE FILE

Once the cluster assignments and the cluster representative cells are determined, reconfiguring the in-force file is straightforward. Static items such as issue age, sex, plan code and similar items are set equal to these items from the representative cell.

Dynamic items such as initial reserve, initial fund, amount of insurance and similar items are summed over all the in-force records belonging to the cluster.

RESULTS OF THE TEST FILES

The results from the test clustering are shown in Table 1. I compare the clustered vs. unclustered results for the total net cash flows and the totals of the synthetic attributes.

Observations on Cluster Model Fit

Given that this model has a 99 percent compression ratio (13 cells compared to 1,347 in the original model), the fit is reasonable.

Model projections for premiums, benefits and distributable earnings are shown in Figure 1.

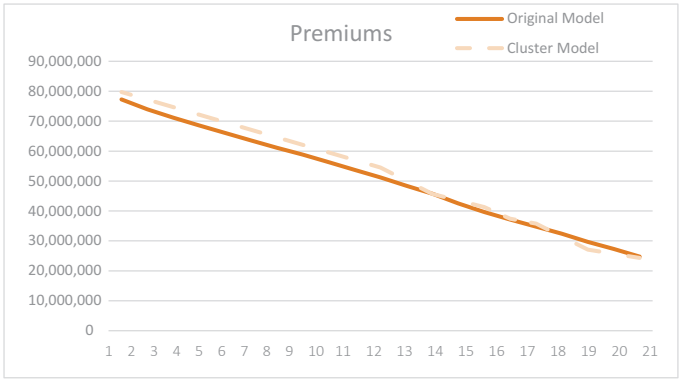
Although the purpose of this article is merely to show how to create cluster models, a brief discussion of how to improve the fit might be helpful. There are two obvious possibilities. The first is to adjust the weighting factors. For example, the weightings for both early and late proxy profits could be increased. This would likely improve the fit for first-year benefits as well as for proxy profits.

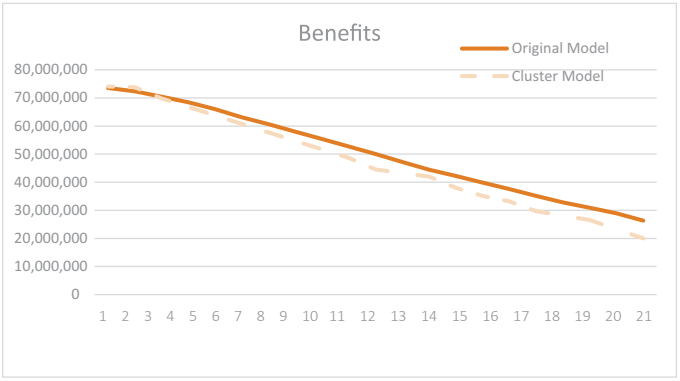
The second obvious adjustment is to split the model into three or four segments, rather than just one segment. Three segments with four cells each might perform better at fitting the original data since the different UL plans have differing patterns of cash flows and profit emergence.

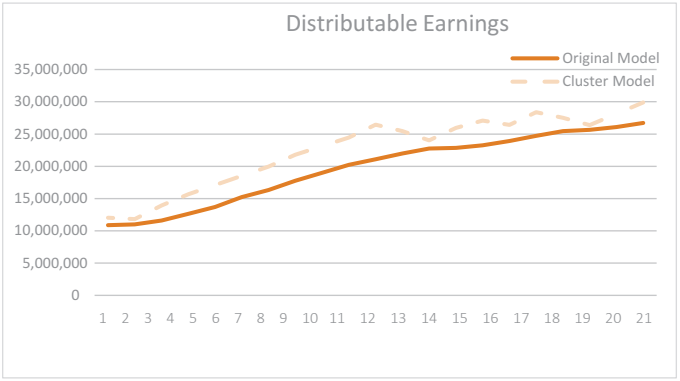
OTHER THINGS TO CONSIDER

In addition to simply generating and identifying clusters, there are several other things we can consider as we create cluster models.

Figure 1
Model Projections for Premiums, Benefits and Distributable Earnings







Optimal Number of Clusters

What is the optimal number of clusters? If we don't care much about model fit, and are mainly concerned about processing time, then obviously one cluster is optimal. In that case, we simply define the cluster based on the average policy number. (This is technically known as "humor." It is not intended to be taken seriously.)

The function `pamk` in the package ***fpc*** estimates the optimal number of clusters based on "optimum average silhouette width." A cluster silhouette is a measure of how close each point in one cluster is to points in the neighboring clusters. The further away the points are, the better.

Interestingly, using the `pamk` function to investigate the optimal number of clusters in the range of two to 100 results in an optimal cluster count of three. That seems to explain how the 99 percent compression model performed as well as it did. Of course, silhouette optimization is not what actuaries are really interested in, so this result does not mean we should automatically choose three clusters.

Testing for Sensitivity to Order of Attributes

Many K-means algorithms used for clustering seem to be based on the expectation-maximization algorithm. There are some anecdotes that these algorithms are sensitive to the order in which the attributes are presented. The documentation for the `pam` function claims it is "a more robust version of K-means."[4] It is not clear if this means it does not exhibit sensitivity to the order of attributes. I did not bother tracing back to the source reference for the algorithm, but it is something users can easily test.

I tested for this sensitivity by running `pam` with two additional input files that differed only in the column order of the data. Both additional files produced clusters identical to the original in-force file.

CONCLUSION

As presented here, DIY clustering is an easy and straightforward process using existing code. As my father-in-law used to tell me, "It's easy once you know how." In R, once you know how, many things are insanely easy. The difficult part of R is that it is so extensive, finding the right package to do just what you want is a time-consuming task. ■

Bob Crompton, FSA, MAAA, is a vice president of Actuarial Resources Corporation of Georgia, located in Alpharetta, Ga. He can be reached at *bob.crompton@arcga.com.*

ENDNOTES

1 Avi Freedman and Craig Reynolds, "Cluster Analysis: A Spatial Approach to Actuarial Modeling," Milliman Research Report, August 2008, *http://www.milliman.com/uploadedFiles/insight/research/life-rr/cluster-analysis-a-spatial-rr08-01-08.pdf.*

2 R Core Team, R: A Language and Environment for Statistical Computing (Vienna: R Foundation for Statistical Computing, 2016), *https://www.R-project.org/.*

3 This can be accessed at *https://cran.r-project.org/.*

4 The pam function is contained in the package cluster: Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K. (2015). cluster: Cluster Analysis Basics and Extensions. R package version 2.0.3. The quickest way to access the documentation for pam is to enter "??pam" at the command prompt in R. You can also access the documentation at *https://cran.r-project.org/web/packages/cluster/index.html.*