PAKDD 2014 - ASUS Malfunctional Components Prediction

Heikki Uuksulainen (Helsinki, Finland)
heikki.uuksulainen@gmail.com

April 17, 2014

1 Introduction

The goal of this competition was to predict future malfunctional components of ASUS notebooks from historical data. The number of repairs was heavenly dependent not just from the particular module-component pair but sales dates as well. This lead to the model where prediction were made for each triple module-component-sale date separately and then aggregated for final predictions.

The data was modelled using binomial regression model. The chosen approach could be thought as a distribution free method in terms of that underlying hazard function was not assumed to follow any particular distribution (like Weibull, Gompertz, log-Normal, etc). Some key aspects of the model were combining information from different levels of hierarchical structures of the data and assuming certain kind of constraints for some of the parameters of the model.

2 Data

For the beginning a couple of notes about missing data. Firstly, sales information was missing for the module $M7$ and month 2006/5. As it was only this one module and one month I didn't apply any complex imputation schemes but instead just added hand picked value based on the previous monthly sales figures. Given the model parametrization described in the next section this should have only quite limited impact on the predictions. Secondly, if repair information was missing for any given month it was assumed that it indicated zero repairs and this information was added to the original dataset.

The following list of variables were derived for all unique module-component-sale time combinations and for all of their possible repair months

Name	**Description**
component	Unique integer ID value for each component
moduleComponent	Unique integer ID value for each Module-Component pair
moduleComponentTime	Unique integer ID value for each Module-Component-Sale time triple
saleCount	How many components were sold
repairCount	How many components were repaired
age	Time between sale and repair in months
repairMonth	Month of the repair, 1 =January, 12 =December

In addition to these, in the code there is a couple of helper data objects like maps between different ID values (e.g. mapping from moduleComponent ID to component ID) and length variables (e.g. how many different components exists).

3 Model

The problem we would like to solve can be stated as following: given that we know how many components were sold, how many of those will be repaired in any given month? One possible way to approach this problem is binomial regression model. The problem can be written as

$$repairCount \sim \text{binomial}(saleCount, p)$$

, where p is the probability of repair. As the probability p has to be in the interval $[0, 1]$ it's often modelled using logistic link function, ie.

$$p = \text{inv.logit}(\theta)$$

, where θ is now unrestricted parameter. The parameter θ was then modelled as a linear combination of three major components:

- $\beta_{mct-scale}$, scale parameters for each moduleComponentTime
- β_{mc-age}, time dependent parameters (one for each possible age) shared by all moduleComponentTime with same moduleComponent. Ie. it's assumed that sale time of the moduleComponent doesn't have an effect to the "time distribution" of the repairs.
- $\beta_{all-month}$, parameters for different repair months (January, February, etc) shared by all

To take the hierarchical structure of the data in consideration Bayesian modelling techniques (though, not full Bayesian inference) were used and the parameters were given hierarchical prior distributions. As an example of such hierarchical structure is the relationship between component and moduleComponent. Let's say that the component which we are interested is a battery. It does seem quite reasonable to assume that different batteries do behave more similarly to each other (in terms of repair counts) than say batteries and processors. A bit different kind of hierarchy would be in terms of time. E.g. module sold at time T should behave more similarly to the same module sold at time $T+1$ than $T+2$, if there is any difference at all.

Parameters β_{mc-age} were modelled in three parts as following for all moduleComponents separately (in total 80 parameters for each moduleComponent):

1. $age \in [0, 26]$, there is training data for all of the moduleComponents
2. $age \in [27, 59]$, training data for just some of the moduleComponents and age (but at least some amount of pooled data for components)
3. $age \in [59, 80]$, no training data

For the first set of parameters the prior distribution was selected to be multivariate normal where the mean vector is given by the component and the variance matrix is linear combination of squared exponential and Gaussian noise variances and is the same for all components. (see for example [3], also in the code mv-normal is calculated using Cholesky decomposition, details: [2]) For the second set of parameters there is additional monotonically decreasing constraint. This constraint was included in the following way for all $age \in [27, 59]$:

$$\beta_{mc-age}[age] = \beta_{mc-age}[age-1] - \exp(decay_{mc-age}[age-26])$$

, where the vector $decay_{mc-age}$ follows same kind of multivariate normal prior as in the first set of parameters. And finally for the remaining parameters, the decay term was just assumed to be the same as the last value in the second set of parameters.

Each moduleComponent has a varying number of sales months, denote by N, and therefore different number of $\beta_{mct-scale}$ parameters, usually around 10-15. The first selling month was given a distribution

$$\beta_{mct-scale}[1] \sim normal(\mu, \sigma)$$

and then for the rest $T \in [2, N]$

$$\beta_{mct-scale}[T] \sim normal(\beta_{mct-scale}[T-1], \sigma_{change})$$

Finally, one of 12 $\beta_{all-month}$ parameters was fixed to be zero and the rest followed normal distributions.

For inference, a maximum a posteriori probability (MAP) estimates of the parameters were learned using BFGS-optimization algorithm. After learning the parameters then the prediction were simply calculated as means of binomial distributions, ie

$$y = saleCount \times p$$

These are predictions for all possible repair months and moduleComponentTimes. So to get the asked repair counts for the moduleComponents these predictions were then aggregated over different sale times.

Additional Comments and Observations

The model presented in this document could be improved in many ways. For example, the following list of ideas were considered, but not implemented due the time, computing power, memory etc. limits during the competition.

- Let time dependent parameters to vary for each moduleComponentTime, not just for moduleComponent. For example, this could be done using same kind of hierarchical structure of the parameters as in the original model and by just adding one more layer. Though, it would probably be a better idea to find a way to reduce the number of parameters before doing so.

- Repair counts of components are correlated. So it would make sense to use multivariate model instead of modelling them independently as was done here.
- Full Bayesian treatment instead of point estimates using MCMC (or variational etc) methods. Better parametrization should make this easier, now the chain in the MCMC sampling didn't seem to mix that well (or in the test run the number of warm up/burn-in iterations wasn't high enough).
- Monotonically decreasing constraint might be a bit too strict (at least one module might have > 24 months warranty period?).
- Improving the prior distribution structures / selection of hyperparameters. E.g. in the current implementation the covariance matrices for the time dependent parameters are not that great (too much noise etc).

4 Code

Dependencies

Data processing and some parts of the prediction process were done with programming language R (version 3.0.1). Actual model specification and parameter learning on the other hand were carried out with probabilistic programming language Stan [1] through RStan (version 2.2) interface. Detailed installation instructions for RStan are given in the website: `https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started`. In addition R package rjson was used to load json files to R.

How To Generate the Solution

Training the model

Run train.R script, which will

1. Load dataset custom_train.csv, if that file exists using functions from data_generation.R. Otherwise first generates the dataset (using original datasets from Kaggle and functions from data_generation.R) and then save to the file.

2. Load stan model code (text specification of the model following somewhat similar syntax as in BUGS/JAGS) and compile the model using stan_model() function from RStan library
3. Learn model parameters using optimizing() function from RStan library
4. Save parameters to the file

Making predictions

Run predict.R script, which will

1. Load dataset custom_test.csv following similar logic as in the 1. training
2. Load learned parameters from the disk
3. Calculate predictions for module-component-sale times
4. Aggregate predictions for module-components
5. Merge predictions with Output_TargetID_Mapping.csv to make submission file
6. Save submission to the disk

References

[1] Stan Development Team. Stan: A c++ library for probability and sampling,version 2.2. `http://mc-stan.org/`, 2014.

[2] Stan Development Team. *Stan Modeling Language Users Guide and Reference Manual, Version 2.2*, 2014.

[3] Wikipedia. Gaussian process. `http://en.wikipedia.org/wiki/Gaussian_process`, April 16, 2014.