Applying Machine Learning in Advertising Technology

Vangie Shue

December 14, 2014

Abstract

This project investigates the application of machine learning in advertising technology, a rapidly growing field. In order to utilize machine learning in advertising technology however, a balance must be found between accuracy and efficiency. As such, this project will provide a preliminary comparison of several algorithms on the basis of how long they take to fit a large amount of data and how accurate their resulting model is. The algorithms will be used to tackle a binary classification problem, specifically: determining hispanic users from the audience in order to target them for advertising campaigns? For our study, we limited our analysis to the following algorithms: Naive Bayes, Gradient Boosting, Logistic Regression, and Support Vector Machines–all of which have an associated library available in R. From our results, we found =====RESULTS=====

Keywords–analytics, online advertising, classification, machine learning

Introduction

Online advertising has undergone sweeping development thanks to fairly recent but rapid innovation and advancement of tools allowing advertisers to automate the management, evaluation, and analysis of their display advertising campaigns. Services like Nielsen Online Campaign Ratings for example, provide demand-side clients, the advertisers, with the ability to determine whether their ads are reaching particular user populations that are more likely to be interested in their products. On the supply-side, companies like Mediamath offer management applications that allow the publishers more control over how their ads are distributed such as by setting daily impression caps and black- or white-listing particular websites. With these improvements to help streamline campaign management, advertisers can now focus on solving more abstract questions and develop more fine-tuned targeting strategies, thus improving the return on their investments (12).

There is a great opportunity to utilize machine learning in identifying pertinent users so that more relevant ads can be targeted to them. Publishers can track how their users typically navigate their sites through tracking pixels and cookies (3). With services like Demdex, a data management platform, they can create traits to define their users. Multiple traits can be further clustered into segments. When publishers receive their impression reports, they are able to see which traits/segments a particular cookie (or "user") satisfies. In machine learning, traits and segments would be analogous to features, while each cookie can act as a data point. As such, a medium-sized company, with nearly four million cookies accessing their sites per day, quickly aggregates a tremendous amount of data.

CafeMom is an online community for moms to come together to exchange advice and support on topics like pregnancy, health, etc. One of their flagship services is MamasLatinas, a site geared towards providing a more relevant community for Hispanic moms. Many advertisers commission CafeMom to help bring their products to Hispanic moms, leading CafeMom to need to develop a reliable model to identify Hispanic women so that they can target these users in both on- and off-site advertisements. But even a simple problem like binary classification can be modeled in many ways in machine learning. This project therefore seeks to provide a preliminary evaluation of the typical learning algorithms that could solve this problem: Naive Bayes, Gradient Boosting, Logistic Regression, and Support Vector Machines. Advertising data is big in scale not only because there are so many online users, but also because it contains a tremendous number of features. As such, the study will need to compare these algorithms both in terms of how quickly they can model the data and how accurately their model predicts on test data.

Classifying Rare Events

While CafeMom may be able to obtain a significant amount of information on Hispanic users through MamasLatinas, they are interested in seeing how these users navigate on other sites in order for them to be able to identify Hispanic moms on affiliated sites. In addition, when sampling, we must be careful not to select on X differently for the two samples (1). Therefore, our project only guarantees a Hispanic user based on their User Agent language. The complement is only considered to be users who use English for their User Agent. As such, in the grand schemes of all their users, Hispanic moms actually constitute only around 2% of their known population of data. Identifying a Hispanic mom is therefore actually a fairly rare event.

According to King and Zeng, rare events tend to be difficult to explain and predict (1). Either logistic regression algorithms will sharply underestimate these events or data collection strategies are grossly inefficient (1). In addition, the authors identified that rare events are actually more statistically informative than "zeros" (1). King and Zeng determined that they could achieve better results by sampling all available events and only a tiny fraction of non-events. This method would also have the added benefit of being more efficient since you would not have to model over [as] tremendous amounts of data in order to get a sufficient number of events while maintaining the event probability.

On the other hand, Paul Allison responded with some critique to the study (2). He believed that the problem was not actually the "rarity" of the events but the actual *number* of events was too small, and therefore led to small-sample bias in logistic models. By example, Allison believed that as long as the sample of rare events was large enough (ex. 2000

events), you could still model on the original sample size (ex. 100,000 cases). In addition, Paul Allison suggested the use of penalized likelihood, or the Firth method, to model data with rare events (2). However, since modeling with enough data to reduce the small-sample bias would drastically increase the execution time, this project will utilize the method offered by King and Zeng. However, we will do a brief comparison of the weighted event method with a non-weighted sample to provide a glimpse as to whether the results would be significantly different.

Classification Algorithms

In an attempt to cover the breadth of learning algorithms for binary classification, we have chosen to evaluate the following algorithms: Naive Bayes, Support Vector Machine, Gradient Boosted Model, and Logistic Regression. All have implementations available in R, which was therefore our language of choice for this study. We will use prediction rate as a measure of goodness of fit, but also look at the rate of false-postives and false-negatives to get an even clearer idea of how the model predicts on data.

Naive Bayes

Naive Bayes is a supervised learning algorithm based on applying the Bayes' theorem with the "naive" assumption of independence between every pair of features. As such, we hypothesize that the modeling time would be on the shorter end of the spectrum, but could suffer in terms of prediction quality. The decision rule for Bernoulli Naive Bayes is based on the following (15):

$$P(xi|y) = P(i|y)xi + (1 - P(i|y))(1 - xi)$$

It explicitly penalizes the non-occurrence of a feature (15). This study will use the `naiveBayes()` in **e1071** as is is the more popular of the two available `R` libraries that support Naive Bayes modeling (22).

Random Forest

Random Forest was initially considered for this study as it is a distinctive algorithm for classification. Random forest tends to perform well, but unfortunately, they suffer heavily with sparse data where bagging and splitting is wasted on non-events. In addition, they are very memory intensive due to the process of growing many classification trees (19). Even with **bigrf**, a random forest library in R geared towards big matrices (20), a test on the "smallest" sample of training data caused the machine to abort, so random forest was decidedly not used for this research.

Support Vector Machines

Support Vector Machines are among the most popular and efficient classification methods used. Fundamentally, SVMs try to find the hyperplane with the maximum margin, or the optimal canonical hyperplane, by identifying the weight vector and bias that yield the maximum margin of all possible separating hyperplanes (23). Currently, four `R` packages offer SVM modeling. A paper by Karatzoglou, Meyer, and Hornik compared these packages and concluded that `ksvm()` in the **kernlab** was the most flexible SVM implementation as it allowed for user-defined kernels. On the other hand, `svm()` in the **e1071** package is a 'robust interface to the award-winning **libsvm** SVM library (21). As such, we chose to use **e1071**'s `svm()` for this study as it was meant to cover more breadth than depth and therefore would not be exploring tuning parameters too intimately.

$$K(xi,xj) = (xi\hat{T})xj + 1$$

Gradient Boosted Model

Gradient boosting model involves the application of boosting to regression trees. A gradient boosted model is initially composed of very simple trees and successive trees are added to predict the residuals of the preceding tree (27). The gradient boosted model is initialized as a constant below, and for m from 1 to M, the pseudo-residuals are computed and a base learner is trained with it. The multiplier is solved and the model updated for the next iteration of m (28):

$$F(x) = y$$
$$F(x) = \arg\min\{f(x)\} \text{ SUM}(y, f(x))$$

Gradient boosted model, or generalized boosted model as it is called in the **gbm** package available in R, is considered to have several advantages over logistic regression: robustness to outliers, predicting on missing data, handling of unequal class sizes and unbalanced predictor variables, as well as tending to have greater predictive ability (7). Of course, a model is not without drawbacks, and may suffer from overfitting especially if the number of ending nodes is too small or number of trees is too large (7). Thus, cross validation is highly recommended if gradient boosted model is the algorithm of choice. However, this study will observe how well the modeled works without the additional tuning via cross validation.

= = = = = = = = = = =

LOGISTIC REGRESSION

(4) glmnet fits a generalized linear model via penalized maximum likelihood. (6) it can deal with all shapes of data, including very large sparse data matrices.

Design

Data Processing

The data used in this study was provided by CafeMom. The raw data is provided via hourly logs from Adobe Demdex. The logs are tab-delimited with a field for unique user ID (uuid) and a comma-delimited array of IDs referring to the traits that that user has satisfied. The data is read as a hive table and partitioned into hispanic and non-hispanic based on the user's User Agent trait. A hispanic user is assumed to have either trait 780664 or 920203 (e.g. their User Agent language is Spanish) while a non-hispanic user is assumed to have either trait 783538 or 920206 (their User Agent language is English). This is to ensure that our data is complementary and does not accidentally bias towards either group.

```
insert overwrite table segment_hispanic
partition(class)
select uuid, max(case when datatype = 4
and (array_contains(varints,780664)
or array_contains(varints,920203))
then 1 else 0 end ) class
from user_attribs2
where (datatype = 4 and
(array_contains(varints,780664)
or array_contains(varints,783538)
or array_contains(varints,920203)
or array_contains(varints,920206)))
group by uuid;
```

In total, the data had 80,324,851 total uuids with 1,698,878 uuids attributed to hispanic users. The percent of hispanic users in our population is therefore approximately 5%. In accordance to the King and Zeng paper, which advised that the non-events be no more than 2- to 5-times the events, we weighted the hispanic users by selecting all the hispanic uuids and only 10% of the non-events.

```
insert overwrite table segment_hispanic2
partition(class)
select * from (
select * from segment_hispanic
where class = 1
union all
select * from segment_hispanic
tablesample(bucket 1 out of 10)
) s;
```

The table segment_hispanic2 consequently had 9,733,751 total uuids with around 19% being hispanic uuids. The data was then processed with R such that the array of traits over all the UUIDs were exploded into separate columns, constituting the feature space. The matrix had 2,105,480 features. After sampling 1.5% rows from the matrix so that we could limit how long the modeling would take, we removed all features that were positive for fewer than 50 UUIDs. This data subset now had 142,433 UUIDs (rows) and 10,128 features (columns). The data was then further partitioned such to set aside approximately 1% of the data to use as test/validation data.

In order to compare the algorithms on their model-fitting speed, we fitted the model using the original data, 0.5% (approx. 70,000 rows), and 0.25% (approx. 35,000 rows) of the data. We then used the first model (based on "all" the data) to determine how long it took to predict on 100, 250, 500, and 1000 rows of test data. Finally, we compared how much the accuracy improved by testing the three models on five samples ot 500 rows of test data. If time permitted, we tested several tuning parameters to see if it would significantly improve the performance of the algorithm in either speed or accuracy.

Naive Bayes

Fitting and predicting data with `naiveBayes` in R is very simple (16). By default, laplace smoothing is set to 0 (25). Increasing the smoothing would likely decrease the modeling time but at the risk of underfitting. Since our data is already very sparse, we used the default laplace smoothing. Perhaps for the same reason, varying the threshold for the prediction did not vary the outcome.

```
library(e1071)
system.time(fitnb<-naiveBayes(
                x=datatrain
                , y=as.factor(ytrain)
                , laplace=0
                ))
system.time(prednb<-predict(
            object=fitnb
            , newdata=datatest
            , type="class", threshold=0.001))
```

Support Vector Machines

The package **e1071** also contains `svm()`. Its implementation is similar to Naive Bayes (24). We use C-classification, also known as Classification SVM Type 1 (26). We chose to use the linear kernel as it is the preferred kernel when dealing with large sparse data vectors (23). The parameters cost, class.weights, tolerance, epsilon, and shrinkage were kept at their default values (24) to minimize over-tuning.

```
library(e1071)
system.time(fitsvm<-svm(
                x=datatrain
                , y=as.factor(ytrain)
                , scale=FALSE
                , type="C-classification"
                , kernel="linear"
                , probability=TRUE
                ))
system.time(predsvm<-predict(
              object=fitsvm
              , newdata=datatest
              , probability=TRUE
            ))
```

Gradient Boosted Model

The **gbm** package in R allows control over several parameters. We set the distribution to "bernoulli" since this is a binary classification problem. The number of trees to fit dictates the number of iterations and basis functions in the additive expansion (29). Best practice would be to set the number of trees to a large number and prune it back afterwards, but to limit the modelling time, we will fit with 100 trees. The shrinkage is the step size or learning rate, which we chose to be 0.1. A smaller step would take longer to model although it could yield better performance. The interaction depth limits the maximum depth of variable interactions, where 1 would imply an additive model and 2 would imply up to two-way interaction, and so forth. Cross-validation can be used to determine the interaction depth. While increasing the interaction depth may produce a closer model, it will also increase the fitting time substantially. The `n.minobsinnode` parameter sets the minimum number of observations for a terminal node. Decreasing this value increases in-sample fit but therefore risks overfitting.

```
library(gbm)
system.time(fitgbm<-gbm.fit(
            x=datatrain[train0,]
            , y=ytrain[train0]
            , distribution="bernoulli"
            , n.trees = 100
            , shrinkage = 1
            , interaction.depth = 2
            , n.minobsinnode = 10
            , keep.data = FALSE
            , verbose = FALSE))
system.time(newtrees<-gbm.perf(
            fitgbm, method="cv"))
system.time(predgbm<-predict.gbm(
            object=fitgbm
            , newdata=datatest
            , n.trees=newtrees
            , type="response"))
```

= = = = = = = = = =

LOGISTIC REGRESSION

We used `glmnet` (Generalized Linear Models) to generate logistic regression model. Then `predict.glm` is used to predict on the validation sample. (3) We then determine the how low of a predicted probability is needed to accurately classify the hispanic division.

Comparison

–Graph: times to model + times to predict (overlayed) –Graph: accuracy

Results

RARE EVENTS=====

Naive Bayes

Figure 1 depicts how the modelling time increased almost linear with restpect to the size of the training data. This is expected since Naive Bayes does not consider feature interactions, so it can fit the classifier in `O(m*n)` time, where `m` is the number of training examples and `n` is the number of features (17). Figure 2 shows the nearly linear relationship between the size of the test data and how long it took to produce to the predictions. Finally, Figure 3 demonstrates the correlation between the accuracy of the predictions and the amount of data used to train the model. In addition, the blue line indicates the change in percent of false-positives, while the red line indicates the percent change in false-negatives. The accuracy appears to increase almost linearly and mostly due to the decrease in false-negatives. The average accuracy of the Naive Bayes model was 80.6%.

```
> m<-c(35000, 70000, 142433)
> ftimes<-c(82.248, 167.030, 294.883)/60
> plot(x=m,y=ftimes,type="b"
+      ,xlab="# training examples"
+      ,ylab="time (min)")
> #cor(m, ftimes) #0.9966318
> #lm(ftimes ~ m) #m: 3.245e-5
```

=====[[NB1_TIME]]=====
Figure 1. Fitting time for Naive Bayes

```
> ptimes<-c(33.053, 82.095, 168.915, 335.145)/60
> sample<-c(100,250,500,1000)
> plot(x=sample,y=ptimes,type="b"
+      ,xlab="# test cases"
+      ,ylab="time (min)")
> #cor(sample, ptimes) #0.9999622
> #lm(ptimes ~ sample) #m: 0.005606
```

=====[[NB2_TIME]]=====
Figure 2. Predicting time for Naive Bayes

```
> m<-c(35000, 70000, 142433)
> nb_acc<-c(0.7992, 0.8036, 0.8148)-0.7992
> #ave(nb_acc): 0.8058667
> nb_fp<-c(0.0056,0.0036,0.0052)-0.0056
> nb_fn<-c(0.1952,0.1928,0.1800)-0.1952
> yax<-c(-0.02,0.02)
> plot(x=m,y=nb_acc,type="b"
+      ,xlab="# training examples"
+      ,ylab="d/dx accuracy (%)"
+      ,ylim=yax
+      )
> par(new=TRUE)
> plot(x=m,y=nb_fp,type="b",axes=FALSE
+      ,xlab="",ylab="",col="blue"
+      ,ylim=yax
+      )
> par(new=TRUE)
> plot(x=m,y=nb_fn,type="b",axes=FALSE
+      ,xlab="",ylab="",col="red"
+      ,ylim=yax
+      )
> #cor(m, nb_acc) #0.9988469
> #lm(nb_acc ~ m) #m: 1.466e-7
```

=====[[NB3_TIME]]=====
Figure 3. Accuracy of Naive Bayes

The Naive Bayes function has little opportunity for tuning. A brief test on modifying the laplace smoothing saw a decrease in fitting time but the accuracy also deteriorated.

Support Vector Machines

From Figure 1, we see that the modelling time begins to increase exponentially as the number of training examples increases. At 140,000 plus data points, the modelling took more than 1.5 hours. In Figure 2, predicting time was nearly linear with respect to the number of test cases and fairly quick. Figure 3 shows little change in the accuracy, a small decrease in the false-positive percent, and a vague increase in the false-negatives. The accuracy of the SVM predictions averaged at 89.1%.

```
> m<-c(35000, 70000, 142433)
> ftimes<-c(436.036, 1531.416, 6470.774)/60
> plot(x=m,y=ftimes,type="b"
+      ,xlab="# training examples"
+      ,ylab="time (min)")
> #cor(m, ftimes) #0.9881824
> #lm(ftimes ~ m) #m: 9.664e-4
```

=====[[NB1_TIME]]=====
Figure 1. Fitting time for SVMs

```
> ptimes<-c(6.115, 7.448, 9.808, 14.865)/60
> samples<-c(100,250,500,1000)
> plot(x=sample,y=ptimes,type="b"
+      ,xlab="# test cases"
+      ,ylab="time (min)")
> #cor(sample, ptimes) #0.9996913
> #lm(ptimes ~ sample) #m: 0.0001627
```

=====[[NB2_TIME]]=====
Figure 2. Predicting time for SVMs

```
> m<-c(35000, 70000, 142433)
> acc<-c(0.8908,0.8916,0.8916)-0.8908
> #ave(nb_acc): 0.8058667
> fp<-c(0.0196,0.0188,0.0176)-0.0196
> fn<-c(0.0896,0.0896,0.0908)-0.0896
> yax<-c(-0.002,0.002)
> plot(x=m,y=acc,type="b"
+      ,xlab="# training examples"
+      ,ylab="d/dx accuracy (%)"
+      ,ylim=yax
+      )
> par(new=TRUE)
> plot(x=m,y=fp,type="b",axes=FALSE
+      ,xlab="",ylab="",col="blue"
+      ,ylim=yax
+      )
> par(new=TRUE)
> plot(x=m,y=fn,type="b",axes=FALSE
+      ,xlab="",ylab="",col="red"
+      ,ylim=yax
+      )
> #cor(m, acc) #0.9988469
> #lm(acc ~ m) #m: 1.466e-7
```

=====[[NB3_TIME]]=====
Figure 3. Accuracy of SVMs

Gradient Boosted Model

Gradient boosted models also take a significant amount of time to generate, but at least their times appears to only increase linearly as opposed to exponentially. Fitting slightly less than 150,000 data points took almost 40 minutes. However, as a trade-off, the predicting times are extremely fast–only our largest test sample, it finished predicting in less than 1 second. Unlike SVMs and Naive Bayes, the accuracy did not appear to improve significantly when the model was fitted on more data. The accuracy averaged at around 89.4%.

```
> m<-c(35000, 70000, 142433)
> ftimes<-c(569.411, 1145.877, 2349.197)/60
> plot(x=m,y=ftimes,type="b"
+      ,xlab="# training examples"
+      ,ylab="time (min)")
> #cor(m, ftimes) #0.9999978
> #lm(ftimes ~ m) #m: 0.0002762
```

=====[[NB1_TIME]]=====

Figure 1. Fitting time for GBM

```
> ptimes<-c(0.078, 0.133, 0.254, 0.464)/60
> samples<-c(100,250,500,1000)
> plot(x=sample,y=ptimes,type="b"
+      ,xlab="# test cases"
+      ,ylab="time (min)")
> #cor(sample, ptimes) #0.9994819
> #lm(ptimes ~ sample) #m: 7.225e-6
```

=====[[NB2_TIME]]=====

Figure 2. Predicting time for GBM

```
> m<-c(35000, 70000, 142433)
> acc<-c(0.8944,0.8932,0.8944)-0.8944
> #ave(nb_acc): 0.8058667
> fp<-c(0.0188,0.0200,0.0184)-0.0188
> fn<-c(0.0868,0.0868,0.0872)-0.0868
> yax<-c(-0.002,0.002)
> plot(x=m,y=acc,type="b"
+      ,xlab="# training examples"
+      ,ylab="d/dx Accuracy (%)"
+      ,ylim=yax
+      )
> par(new=TRUE)
> plot(x=m,y=fp,type="b",axes=FALSE
+      ,xlab="",ylab="",col="blue"
+      ,ylim=yax
+      )
> par(new=TRUE)
> plot(x=m,y=fn,type="b",axes=FALSE
+      ,xlab="",ylab="",col="red"
+      ,ylim=yax
+      )
> #cor(m, acc) #0.1972159
```

=====[[NB3_TIME]]=====

Figure 3. Accuracy of GBM

= = = = = = = = = = =

LOGISTIC REGRESSION COMPARISON

Overlay fitting graphs. Overlay predicting graphs.

Conclusion

We have demonstrated the application of machine learning in advertising technology, in particular for a problem with rare events.

Rare Events: In addition, it allows us to model with less data and is therefore more efficient for this application.

Our results should not be interpreted as a standard comparison of these algorithms. Since most algorithms, and their implementations in R, allow for the tuning of parameters (such as making learning steps smaller and testing various lambda values), without a doubt, the performance of these algorithms could be improved and therefore yield different results from our own research. This project is better served as an example or reference of how machine learning algorithms can be compared.

While the results are encouraging, further exploration of machine learning tools is needed especially for processing large amounts of data with a similarly immense feature space. Apache Mahout is a very promising option as the project is aimed towards scalable machine learning and may allow us to test additional algorithms like random forest (18). Alternatively, using SparkR would allow for integration of R's diverse offering of machine learning tools and the parallel processing power of Spark (8). However, these technologies are still very new and undergoing significant amounts of development. Furthermore, they are also limited in their available documentation and resources. Nonetheless, moving forward we expect to see much advancement in machine learning tools to solve problems in advertising technology.

Acknowledgements

I would like to thank Patrick McCann at Cafemom for providing guidance for this project as well as allowing the use of Cafemom's advertising data to perform this research. In addition, I would like to thank Professor Mohri, who has imparted invaluable knowledge on machine learning algorithms in Foundations of Machine Learning (CSCI-GA. 2566-001).

References

- 1 http://gking.harvard.edu/files/gking/files/0s.pdf (rare events)
- 2 http://www.statisticalhorizons.com/logistic-regression-for-rare-events (rare events)
- 4 http://cran.r-project.org/web/packages/glmnet/glmnet.pdf
- 3 http://stats.stackexchange.com/questions/25389/obtaining-predicted-values-y-1-or-0-from-a-logistic-regression-model-fit
- 5 http://machinelearningmastery.com/an-introduction-to-feature-selection/ (feature selection tips)
- best algorithms: http://www.researchgate.net/post/What_is_the_best_algorithm_for_classification_task
- 23 http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/SVM (SVM)

- 22 http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/Na%C3%AFve_Bayes
- 24 http://www.inside-r.org/node/57517
- 25 http://www.inside-r.org/packages/cran/e1071/docs/naiveBayes
- 26 http://www.statsoft.com/textbook/support-vector-machines
- http://www.jstatsoft.org/v61/i10/paper (random ferns)
- http://www.statmethods.net/advstats/cart.html (rpart and random forest)
- 6 http://www.inside-r.org/packages/cran/glmnet/docs/glmnet
- 7 http://vimeo.com/71992876 (gbm)
- 8 http://datasaucer.blogspot.com/ (R for big data)
- 9 http://www.inside-r.org/packages/cran/gbm/docs/gbm
- 10 http://jwijffels.github.io/RMOA/doc/pdf/Manual.pdf (MOA manual)
- 11 https://github.com/jwijffels/RMOA (streaming RMOA)
- 12 http://wallblog.co.uk/2014/02/12/what-is-next-for-the-world-of-advertising-technology/
- 13 http://www.grovo.com/display-advertising-fundamentals/the-key-players
- 14 http://www.cafemom.com/
- 15 http://scikit-learn.org/stable/modules/naive_bayes.html
- 16 http://www.inside-r.org/packages/cran/e1071/docs/naiveBayes
- 17 https://www.biostars.org/p/8727/
- 18 http://mahout.apache.org/
- 19 https://www.stat.berkeley.edu/ breiman/RandomForests/cc_home.htm
- 20 http://cran.r-project.org/web/packages/bigrf/bigrf.pdf
- 21 http://www.jstatsoft.org/v15/i09/paper
- 27 http://www.statsoft.com/Textbook/Boosting-Trees-Regression-Classification
- 28 http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/
- 29 http://www.inside-r.org/packages/cran/gbm/docs/gbm