**Stanford University Stats 202 Kaggle Competition**

**December, 2014**

**Team:**

**TowerProperty**

**Members:**

**Kartik Trasi (ktrasi@gmail.com)**

**Michal Lewowski (milewows@stanford.edu)**

**Matthew Schumwinger (mjs13@stanford.edu)**

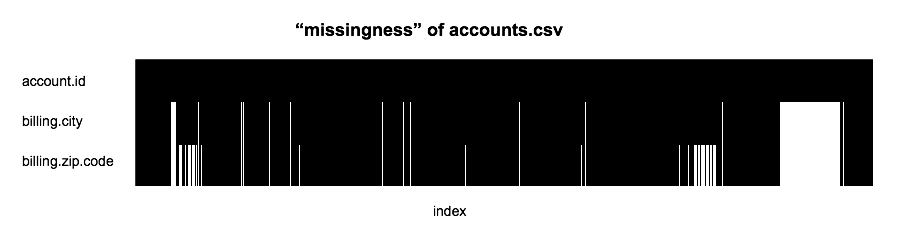
**Features**

From the very beginning, our top priority was to develop useful features. Knowing that we would learn more powerful statistical learning methods as the course progressed, we made sure that we had the features ready so we would be able to evaluate them quickly and easily. Apart from some obvious features (that we created as a part of the homework), we spent substantial time deriving many other useful features. These can be divided into the following categories:

* **Preferences**
  + We parsed the past concerts table for the artists being played in a given year and compared that with the number of subscriptions bought that year by account .
  + We wanted to see if we could classify accounts as Bach, Haydn or Vivaldi -lovers and use that for predictions (perhaps Bach lovers will be more likely to buy subscriptions this year as well because there is another Bach concert in the performance schedule).
  + We also computed the negative preferences – if a certain account did not buy a subscription for a given year, what concerts were performed that year? Perhaps, if similar concert is played this year we could detect a pattern.
* **Spatial**
  + We parsed and backfilled the account.csv file to compute geodesic distances from the thee primary venue locations (SF Jazz Center, Bing Concert Hall, and the First Congregational Church in Berkely). We did this by assigning latitude and longitude coordinates to each account according to the centroid of its billing zip code. In general, we observed a pattern that accounts from outside California rarely bought new subscriptions and accounts outside the Bay Area rarely bought more than one subscription (this effect can be observed by exploring our web visualization [here](https://a.tiles.mapbox.com/v4/milwaukeestat.k3a6h78o/page.html?access_token=pk.eyJ1IjoibWlsd2F1a2Vlc3RhdCIsImEiOiJtSjZjOGdjIn0.tjwaUCz_8PMqq9ePiwB5nw" \l "8/37.697/-121.542)). Therefore, we expected that spatial predictors like State, City or distance-to-venue would be very useful (it actually turned out that they behaved strangely, which we describe below).
* **Past subscriptions**
  + We noticed that submitting predictions that mimic the prior year was , by itself, resulting in a good score, so it was obvious that number of subscriptions bought over each year for a few last years would be a useful predictor.
* **Tickets, prices, sections, packages**
  + Using dcast() and melt(), we also prepared a lot of predictors based on price levels, number of tickets bought by that account during that year, etc.
* **Account.num**
  + This was a surprising feature to even consider – could the simple line number (its index) in the accounts.csv for given account could be useful? It shouldn’t be useful if the account order was random. But, as found out, it actually was quite useful.

Once we noticed that we were likely to get best results with GBM and decision trees, we analyzed the accounts that were buying subscriptions and changing the level of subscriptions over the years, as well as those that bought last year. We noticed that the distribution of these in the accounts.csv file was far from random. Those accounts were much more frequent at the beginning and end of that file. That made us think that maybe this file was not really randomized and it represents some useful (maybe chronological?) order (as opposed to the subscriptions.csv and tickets.csv files).

The non-random nature of the accounts.csv order is also illustrated in the “missingness” of spatial features, as shown in this plot:



We also felt that decision trees have a natural property of finding the right splits, which can be useful to separate some accounts from the others. So, based on that, we started including account.num as a predictor and… it discovered that it did in fact make our cross-validation scores better. Actually, in the winning model it turned out to be the fourth most important predictor!

* **Derivative features**
  + We also smoothed some features to avoid over-fitting. Usually, we were using predictors from a the last few years, but in the winning model we actually used only the last two years and average values over the last five years. We also added some derived predictors expressing the variance of number of subscriptions bought (so that the decision tree would be more easily able to separate “stable” accounts from “unstable” ones).

In total, we created 277 features, which we applied in different combinations. Surprisingly, our final model used only 18 of them.

**Approaches**

As suggested in the lecture, we used GitHub to host the code and collaborate. Our code was written entirely in R (although we sometimes used Excel to derive new predictors).

Originally, we started with multiple linear regression models. These models were used primarily to help us become familiar with the data as we focused most of our efforts on preparing features that we could easily apply to any new method we would learn as the course progressed.

From the very beginning, we also decided to use 10-fold cross validation error as the metric to optimize against. We also used it to compare different methods.

We applied and tested the following methods:

* Multiple linear regression (lm),
* SVM (ksvm),
* BART (because it was mentioned in passing during one of the lectures),
* KNN (knn), and
* Boosted decision trees (gbm)

We didn’t have much luck with ksvm, bart and knn. Maybe we did not put enough effort into that, but since we already had very good results from using GBM, the bar was already quite high. Our biggest effort soon turned to tuning GBM parameters.

As mentioned, we were always using 10-fold cross validation error to guide us through any decisions. Originally we were tuning mostly:

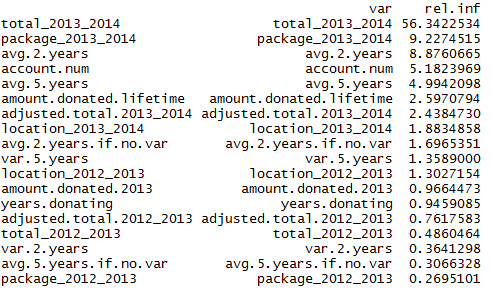
* number of trees
* bagfrac
* shrinkage
* depth

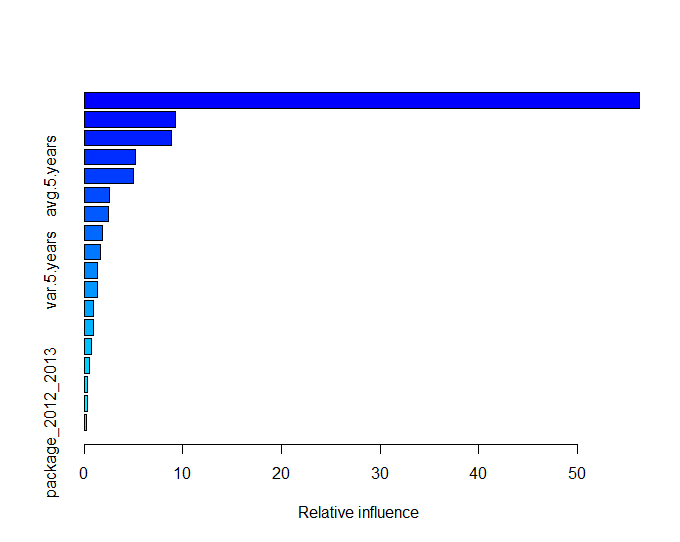
In the later phase, we realized that we were probably using too many predictors and that there was one more, very important parameter to tune: which is minobsinnode. This parameter actually helped a lot, which probably makes sense as the default value is 10, which feels too big for the data. We saw significant improvements when setting the parameter to 4 and 5 – we ended up using 5.

The tuning process was both manual and automated (We wrote some scripts that we executed overnight, which worked by randomly changing the parameters and set of predictors as well as computing 10-fold cross-validation error. But these scripts were usually used only as a guide for values and approaches that might be worth to be further investigated manually).

To our initial surprise, we discovered that it was often beneficial to drop a predictor that had high importance in the splits! We noticed this phenomenon mostly with spatial predictors like City and distance-from- venue. They were always getting chosen (in each out of 10 splits) as very important … and yet they were consistently increasing the final error. Once we detected this phenomenon, we grew more and more suspicious of any predictor that we were using. It turned out that simplifying the model (by removing predictors) actually showed improvements. For example: we were in general never using data from before 2005 (we knew from PCA in our homework that the years 2002-2004 were weird) and the older data didn’t seem to be too useful, but we were usually using all data for years greater than 2009. It turned out that the more aggressive we were in eliminating older predictors, the smaller the error. In the final model, we used only 2013-2014 and 2012-2013 seasons and values averaged over last 5 years.

Out of all predictors, the model that gave us the lowest 10-fold CV error and won the competition, was surprisingly simple and small:





Some variables are hopefully quite straightforward, the others we will try to explain:

* adjusted.total\_2013\_2014 – the total number of subscriptions adjusted with number of tickets bought by that account that year.
* var.5.years – the variance of number of subscriptions over 5 years.
* avg.5.years.if.no.var – the average value of subscriptions from the last 5 years, assuming someone was always buying same number of subscriptions; and 0 otherwise.
* years.donating – the number of years from that account’s first donation

The final cross-validated parameters that we chose were:

* trees = 3375
* bagfrac = 0.5
* shrinkage = 0.001
* depth = 9
* minobsinnode = 5

This model resulted in a 10-fold cross validation error of 0.0908453290892268, which was quite similar to both our score in the public and private Kaggle dashboards.

The rule we used to choose our final two submissions was simple: select the one that produced the lowest public Kaggle score, and the one that gave us the lowest 10-fold cross validation error. This actually turned out to be somewhat of a tough decision because we had other submissions with better public Kaggle scores. Nevertheless, we still decided to trust our 10-fold cross-validation and it turned out to be the right choice. (It was also not that easy of a decision to choose the model that was not really using all of these additional predictors that we had spent time creating!) In the end, the more advanced model –the one that resulted in our best public score– was also beaten by the simpler one.

