Spotify Song Recommendation Model

Alex Shen, Matt Robbins

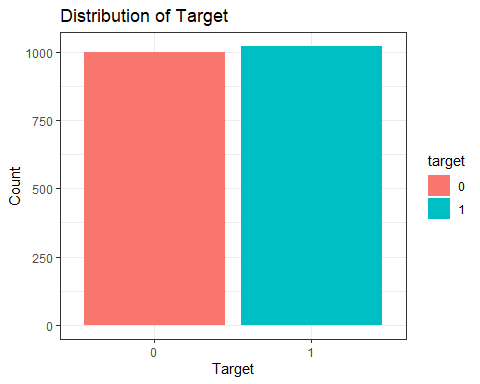
## Introduction

Spotify is a music streaming service with 356 million monthly active users and over 70 million tracks in their library. What drives their success? Aside from the size of the library, it is their ability to keep users engaged by providing a constant stream of relevant song and even podcast recommendations. In order to make these recommendations, there must be some data on songs as well as user preference data that can help decide which songs to recommend to a particular person. Some of Spotify’s competitors such as Pandora will tag songs with attributes manually, while Spotify has the advantage of using deep learning models to do this as well as synthesize this with artist, genre, and user preference information in order to better inform recommendations. The three main recommendation models used at Spotify include collaborative filtering (which analyzes similarity in behavior between users), natural language processing or NLP (which extracts information from words in song lyrics, song titles, and playlist titles), and audio models (which detect patterns in raw audio content). (ADD CITATION)

Our data set came from Kaggle, and was based on one person’s music preferences. Each observation in the data is a song on Spotify, and we are given information about each song, as well as whether or not the person liked the song. Continuous variables of interest that are between 0 and 1 include acousticness, danceability, energy, instrumentalness (amount of vocals in a song), liveness (how likely the song was recorded live), loudness, speechiness (amount of spoken words in a song), and valence (measure of how happy a song sounds). Other variables of interest include duration\_ms (continuous variable denoting duration of the song in ms), key (categorical variable denoting the key of the song from 0-11), mode (modality of the song expressed as 0 or 1, corresponding to major or minor), tempo (continuous variable in BPM), time\_signature (categorical variable from 1-5), and target (whether or not the person liked the song expressed as a 0 or 1). Most of these variables come from Spotify’s automated deep learning models that processes the raw audio of songs.

We wish to do a Bayesian logistic regression using the information about the song as predictors and target as the response variable. With such a model, the best use of it would be as a song recommender algorithm. We would be able to predict whether or not the person likes a song, and if we predict that they like the song, we would recommend it. One limitation of using this method to recommend a song is that it requires user input. Most users will not take the time to record whether or not they like or dislike songs that they come across, which is why Spotify’s recommendation model takes into account factors such as time a song is played for or if a person has the song in a playlist in order to determine if a person likes a song or not. Despite our proposed method not being useful as a recommendation algorithm for Spotify itself, we do have the upside of being able to definitively know whether or not a person likes a song, rather than having to infer on it as Spotify does. So, for people who are fine with compiling a list of songs that they explicitly like or dislike, our model should be able to make a more targeted recommendation.

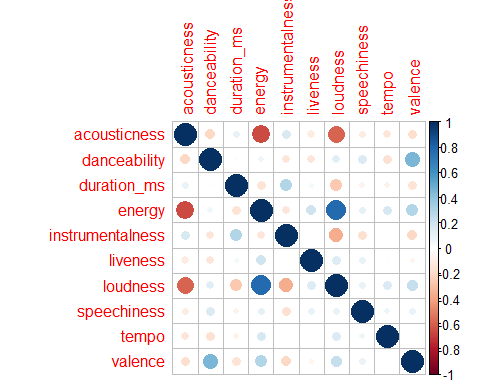
## Data



The data set has 2017 songs. We decided to randomly divide this data, with 75% of it becoming the data on which we will train the model on, and the remaining 25% becoming data we will test the model on after it is created. We also check the distribution of target, since we want to make sure there are enough liked and disliked songs in the data set to be able to properly train our model. For example, we do not want to see only a handful of liked songs and almost all disliked songs, as this could result in a model that predicts almost every song as disliked, and will end up with a inflated correct prediction rate. However, as we can see, the data set is split almost into half liked songs and half disliked songs, which is beneficial for training our model.

After selecting the variables relevant to our analysis, we also factorized the categorical variables so that R would not accidentally treat them as continuous.

For our purposes, since we are trying to model and predict whether or not this person likes a song, the response variable target takes the value of 0 or 1. That means our sampling model is a Bernoulli distribution with probability p.



Because our continuous predictors are on different scales, we decided to normalize them. We then plot a correlation plot between each of the continuous predictors. As we can see, several variables have a large of correlation. Energy and acousticness as well as loudness and acousticness are negatively correlated, while energy and loudness are positively correlated. We are trying to avoid multicollinearity in our model, and since high correlation often is associated with multicollinearity, we will keep an eye on the pairs of variables with high magnitudes of correlation.

## Model Creation

We were not able to find another logistic regression on a data set of songs on Spotify in order to predict whether or not someone likes a song. As a result, we did not have much information about what the coefficients might be, and decided to choose weakly informative priors. In this project, we consider three different prediction models: a logistic regression model which uses Cauchy(0,2.5) priors for its coefficients, a logistic regression model which uses regularized horseshoe priors for its coefficients, and a “null” model which produces 1 (a “yes” for liking the song), regardless of input variables. We choose a Cauchy(0,2.5) prior for our first model because it is an empirically successful choice for weakly informative priors (<https://projecteuclid.org/journals/annals-of-applied-statistics/volume-2/issue-4/A-weakly-informative-default-prior-distribution/10.1214/08-AOAS191.full>). Regularized horseshoe priors are typically meant for models in which the number of predictive variables is large in comparison to the number of samples. Although this is not the case in our experiment (, ), we were still interested to see whether this model could outperform our standard model.

We decided to backwards select our models using Bayes factors, which are useful in selecting Bayesian models. In our full model, we included all predictors with target as the response variable. We also decided to add some interaction effects. We thought that the mode variable (modality of a song, major or minor) would affect the danceability of a song as well as the valence (measurement of how happy a song sounds). We also thought that the time signature of a song might affect its tempo. As a result, we added interactions between mode and danceability, mode and valence, and time signature and tempo.

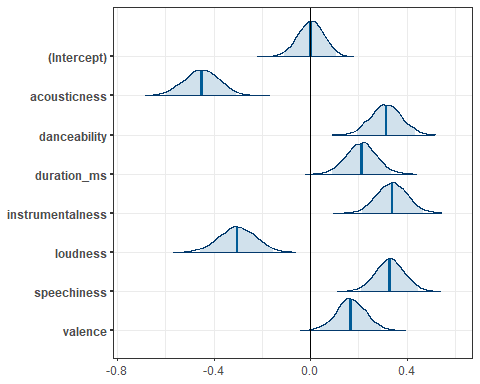
We calculated the Bayes factor of each coefficient compared to a point-null, which is the Savage-Dickey Ratio (<https://easystats.github.io/bayestestR/articles/bayes_factors.html>). We also checked the p value of a hypothesis test for each coefficient equaling 0. We dropped the term with the highest p value and lowest Bayes factor (these were always in agreement). We stopped when each term had a Bayes factor that can be interpreted as strong (or better) support for the coefficient not equaling zero. (<https://www.statisticshowto.com/bayes-factor-definition/>)

### Model with Cauchy Prior

## Sampling priors, please wait...

## Bayes Factor (Savage-Dickey density ratio)  
##   
## Parameter | BF  
## -------------------------  
## (Intercept) | 0.017  
## acousticness | > 1000  
## danceability | 85.47  
## duration\_ms | 4.90  
## instrumentalness | 204.79  
## loudness | 54.16  
## speechiness | > 1000  
## valence | 0.738  
##   
## \* Evidence Against The Null: 0

## Call:  
## BF.glm(x = model)  
##   
## Bayesian hypothesis test  
## Type: exploratory  
## Object: stanreg  
## Parameter: general parameters  
## Method: adjusted fractional Bayes factors using Gaussian approximations  
##   
## Posterior probabilities:  
## Pr(=0) Pr(<0) Pr(>0)  
## (Intercept) 0.951 0.023 0.025  
## acousticness 0.000 1.000 0.000  
## danceability 0.000 0.000 1.000  
## duration\_ms 0.053 0.000 0.947  
## instrumentalness 0.000 0.000 1.000  
## loudness 0.007 0.993 0.000  
## speechiness 0.000 0.000 1.000  
## valence 0.347 0.002 0.650



##   
## Model Info:  
## function: stan\_glm  
## family: binomial [logit]  
## formula: target ~ acousticness + danceability + duration\_ms + instrumentalness +   
## loudness + speechiness + valence  
## algorithm: sampling  
## sample: 4000 (posterior sample size)  
## priors: see help('prior\_summary')  
## observations: 1513  
## predictors: 8  
##   
## Estimates:  
## mean sd 10% 50% 90%  
## (Intercept) 0.0 0.1 -0.1 0.0 0.1   
## acousticness -0.5 0.1 -0.5 -0.5 -0.4   
## danceability 0.3 0.1 0.2 0.3 0.4   
## duration\_ms 0.2 0.1 0.1 0.2 0.3   
## instrumentalness 0.3 0.1 0.3 0.3 0.4   
## loudness -0.3 0.1 -0.4 -0.3 -0.2   
## speechiness 0.3 0.1 0.3 0.3 0.4   
## valence 0.2 0.1 0.1 0.2 0.2   
##   
## Fit Diagnostics:  
## mean sd 10% 50% 90%  
## mean\_PPD 0.5 0.0 0.5 0.5 0.5   
##   
## The mean\_ppd is the sample average posterior predictive distribution of the outcome variable (for details see help('summary.stanreg')).  
##   
## MCMC diagnostics  
## mcse Rhat n\_eff  
## (Intercept) 0.0 1.0 4647   
## acousticness 0.0 1.0 5269   
## danceability 0.0 1.0 5148   
## duration\_ms 0.0 1.0 5717   
## instrumentalness 0.0 1.0 4501   
## loudness 0.0 1.0 5189   
## speechiness 0.0 1.0 4791   
## valence 0.0 1.0 4946   
## mean\_PPD 0.0 1.0 4277   
## log-posterior 0.0 1.0 2069   
##   
## For each parameter, mcse is Monte Carlo standard error, n\_eff is a crude measure of effective sample size, and Rhat is the potential scale reduction factor on split chains (at convergence Rhat=1).

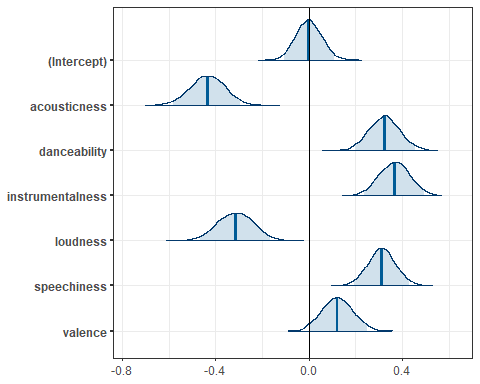
|  |  |  |
| --- | --- | --- |
| term | estimate | std.error |
| (Intercept) | 0.003 | 0.054 |
| acousticness | -0.451 | 0.073 |
| danceability | 0.314 | 0.063 |
| duration\_ms | 0.212 | 0.061 |
| instrumentalness | 0.339 | 0.064 |
| loudness | -0.303 | 0.077 |
| speechiness | 0.331 | 0.059 |
| valence | 0.168 | 0.061 |

### Model with Horseshoe Prior

## Sampling priors, please wait...

## Bayes Factor (Savage-Dickey density ratio)  
##   
## Parameter | BF  
## -------------------------  
## (Intercept) | 0.019  
## acousticness | > 1000  
## danceability | > 1000  
## instrumentalness | > 1000  
## loudness | > 1000  
## speechiness | > 1000  
## valence | 41.25  
##   
## \* Evidence Against The Null: 0

## Call:  
## BF.glm(x = hs\_model)  
##   
## Bayesian hypothesis test  
## Type: exploratory  
## Object: stanreg  
## Parameter: general parameters  
## Method: adjusted fractional Bayes factors using Gaussian approximations  
##   
## Posterior probabilities:  
## Pr(=0) Pr(<0) Pr(>0)  
## (Intercept) 0.951 0.025 0.024  
## acousticness 0.000 1.000 0.000  
## danceability 0.000 0.000 1.000  
## instrumentalness 0.000 0.000 1.000  
## loudness 0.004 0.996 0.000  
## speechiness 0.000 0.000 1.000  
## valence 0.770 0.007 0.224

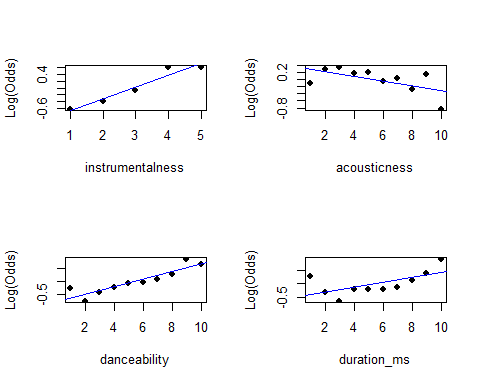


##   
## Model Info:  
## function: stan\_glm  
## family: binomial [logit]  
## formula: target ~ acousticness + danceability + instrumentalness + loudness +   
## speechiness + valence  
## algorithm: sampling  
## sample: 4000 (posterior sample size)  
## priors: see help('prior\_summary')  
## observations: 1513  
## predictors: 7  
##   
## Estimates:  
## mean sd 10% 50% 90%  
## (Intercept) 0.0 0.1 -0.1 0.0 0.1   
## acousticness -0.4 0.1 -0.5 -0.4 -0.3   
## danceability 0.3 0.1 0.2 0.3 0.4   
## instrumentalness 0.4 0.1 0.3 0.4 0.4   
## loudness -0.3 0.1 -0.4 -0.3 -0.2   
## speechiness 0.3 0.1 0.2 0.3 0.4   
## valence 0.1 0.1 0.0 0.1 0.2   
##   
## Fit Diagnostics:  
## mean sd 10% 50% 90%  
## mean\_PPD 0.5 0.0 0.5 0.5 0.5   
##   
## The mean\_ppd is the sample average posterior predictive distribution of the outcome variable (for details see help('summary.stanreg')).  
##   
## MCMC diagnostics  
## mcse Rhat n\_eff  
## (Intercept) 0.0 1.0 5651   
## acousticness 0.0 1.0 3883   
## danceability 0.0 1.0 3585   
## instrumentalness 0.0 1.0 3775   
## loudness 0.0 1.0 3939   
## speechiness 0.0 1.0 3981   
## valence 0.0 1.0 2892   
## mean\_PPD 0.0 1.0 4715   
## log-posterior 0.1 1.0 1022   
##   
## For each parameter, mcse is Monte Carlo standard error, n\_eff is a crude measure of effective sample size, and Rhat is the potential scale reduction factor on split chains (at convergence Rhat=1).

|  |  |  |
| --- | --- | --- |
| term | estimate | std.error |
| (Intercept) | -0.002 | 0.056 |
| acousticness | -0.434 | 0.074 |
| danceability | 0.324 | 0.065 |
| instrumentalness | 0.367 | 0.066 |
| loudness | -0.314 | 0.080 |
| speechiness | 0.311 | 0.058 |
| valence | 0.122 | 0.066 |

Using a Rhat convergence diagnostic for each variable in each model, we find that all variables produced a Rhat value less than 1.05, which is within the valid range for convergence of the MCMC process (<https://mc-stan.org/rstan/reference/Rhat.html>).

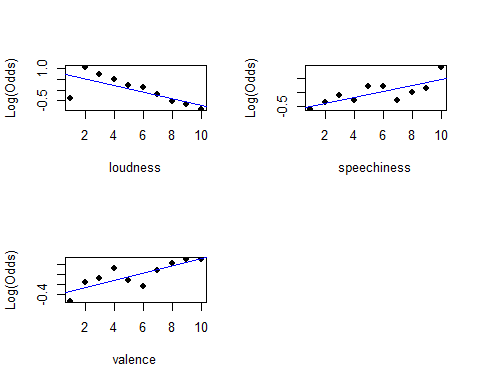
## Model Conditions



## Registered S3 methods overwritten by 'car':  
## method from  
## influence.merMod lme4  
## cooks.distance.influence.merMod lme4  
## dfbeta.influence.merMod lme4  
## dfbetas.influence.merMod lme4

## # A tibble: 7 x 2  
## Variables VIF  
## <chr> <dbl>  
## 1 acousticness 1.71  
## 2 danceability 1.30  
## 3 duration\_ms 1.11  
## 4 instrumentalness 1.22  
## 5 loudness 1.89  
## 6 speechiness 1.04  
## 7 valence 1.36

## # A tibble: 6 x 2  
## Variables VIF  
## <chr> <dbl>  
## 1 acousticness 1.68  
## 2 danceability 1.32  
## 3 instrumentalness 1.25  
## 4 loudness 1.91  
## 5 speechiness 1.04  
## 6 valence 1.36



Now that our models are created, we check our model conditions for logistic regression. The first condition is linearity. As is shown by the graphs above, each of the continuous predictors in our models appear to have a more or less linear relationship with the log odds of the person liking a song. As a result, the linearity condition is satisfied.

The next condition is the randomness condition. Because the samples are not stated to be truly randomly taken we must consider whether or not the observations differ systemically from our population of interest. The population of interest is songs that the person who made the data set listens to and rates as a like or a dislike, and so there is no reason to believe that the sample is not representative of the population. The randomness condition is satisfied.

The next condition is the independence condition. We have no reason to believe that each song is not independent from one another, and so the independence condition is satisfied.

Lastly, we check multicollinearity by checking the variance inflation factors of each of the variables. As seen in the output for both cauchy and horseshoe, since all VIF values are under 10, multicollinearity is not a problem in our model.

## Model Comparison and Results

##   
## Computed from 4000 by 1513 log-likelihood matrix  
##   
## Estimate SE  
## elpd\_loo -960.5 13.4  
## p\_loo 8.6 0.4  
## looic 1921.0 26.8  
## ------  
## Monte Carlo SE of elpd\_loo is 0.0.  
##   
## All Pareto k estimates are good (k < 0.5).  
## See help('pareto-k-diagnostic') for details.

##   
## Computed from 4000 by 1513 log-likelihood matrix  
##   
## Estimate SE  
## elpd\_loo -966.0 12.4  
## p\_loo 7.7 0.3  
## looic 1932.0 24.7  
## ------  
## Monte Carlo SE of elpd\_loo is 0.0.  
##   
## All Pareto k estimates are good (k < 0.5).  
## See help('pareto-k-diagnostic') for details.

##   
## Computed from 4000 by 1513 log-likelihood matrix  
##   
## Estimate SE  
## elpd\_loo -1049.7 0.1  
## p\_loo 1.0 0.0  
## looic 2099.4 0.3  
## ------  
## Monte Carlo SE of elpd\_loo is 0.0.  
##   
## All Pareto k estimates are good (k < 0.5).  
## See help('pareto-k-diagnostic') for details.

To compare the predictive performance of our three models, we use leave-one-out cross-validation (LOO). In particular, we use the LOO Information Criterion (LOOIC) as a metric by which to measure the predictive performance of our models. LOOIC measures the predictive accuracy of a given model by leaving a single sample out of the training set and training the model on the all other samples in the dataset. It then tests its accuracy on this single left-out sample and repeats this process for all other samples in the training set. LOOIC reports a metric based on the accuracy of these LOO predictions, where a lower LOOIC score corresponds to a better model (<http://mc-stan.org/rstanarm/reference/loo.stanreg.html>). The LOOIC values for all three models are shown below. Of the three models, we see that the cauchy prior model performed the best.

|  |  |  |
| --- | --- | --- |
| Model | HS | Null |
| 1921.02 | 1932.0 | 4 2099.45 |

With our chosen model, we now want to be able to use it on song data we may find in the wild. However, our current model was fit on a normalized dataset, so its parameters are only trained to classify songs by characteristics with mean 0 and standard deviation 1. To retrieve the parameters for a model that works on unnormalized data () instead of normalized data , we adjust our coefficients as follows:

With our unnormalized model, we first check to see how well it predicts the ‘target’ value of the samples on which it was trained. Using a probability threshold of 0.5, we find that our cauchy prior predictor categorizes 0.658 of the songs in the training set. For comparison, the horseshoe prior model has a prediction accuracy of 0.657, and the “null” model has a prediction accuracy of 0.502 on the training set. Of course, here we are checking performance on songs which the models have already trained on. What we really want to measure is the ability of our models to make predictions on songs they have not yet seen. For this, we check the accuracy of our models on the test set. Since our models have not seen this data yet, we would expect their performance on this dataset to be slightly worse than on the training set. This is exactly what we see for the cauchy and horseshoe prior models. Again, using a probability threshold of 0.5, we find that our cauchy prior predictor has accuracy 0.6825397 on songs in the test set. For comparison, the horseshoe prior model has prediction accuracy hs\_test\_acc, and the “null” model has prediction accuracy of one\_test\_acc on the test set. Thus, we see that under this metric, the cauchy prior predictor again performs the best.

## Conclusion

Our final model with a weakly informative cauchy prior had a predictive accuracy of 0.6825397 on the testing data. This accuracy is noticeably better than our null model that guesses 1 every time and ends up with an average prediction accuracy of a little over 50% because the data is comprised of a little over 50% liked songs.

This prediction/recommendation accuracy would be acceptable to us as Spotify users. From anecdotal evidence, while Spotify does recommend plenty of songs that we enjoy, it also recommends many songs that we do not like as well. From being active Spotify users, we would estimate the accuracy of Spotify’s recommendations to be in the same ballpark as our predictive accuracy. Spotify also has the added bonus of having extra variables to create their recommendation that we were not able to use. For example, Spotify is able to take into account similar user’s activities as well as information about artists and genres that we were not able to use. However, we do have the advantage of explicitly knowing whether or not a song is liked or disliked, while Spotify must infer that information for each user. Because our model was built given this information, it does not have the capability to be used as a replacement to Spotify’s algorithm because users will not explicitly record whether or not they like each song.

For further exploration on this topic, if we had access to data regarding listening times, whether or not a song is on a user’s playlist, and other such data that might clue us in on if a user enjoys a song, we could create an algorithm with this data to figure out if a user likes a song, and then create recommendations with our current logistic models using the values of ‘target’ that we predicted. This would probably drop our accuracy rate, but it would then have the potential for real world use within the Spotify app that Spotify’s algorithms have. Until then, our algorithm can be used for people like the Kaggle user who compiled information about songs he or she did or didn’t like.