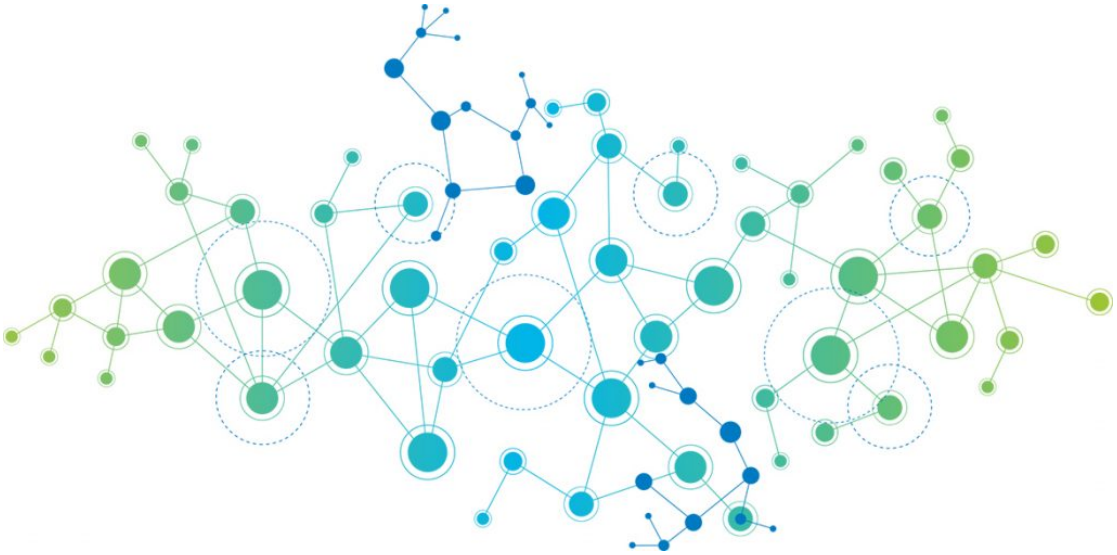
Bayesian reasoning to predict Spotify's hits

Pietro Fanti - pietro.fanti@studio.unibo.it

October 2021



Contents

1 Introduction

Music market has been in constant growing for the last years and this growth is of course driven by streaming platforms, of which Spotify is one of the main actors. With more than 60000 tracks loaded every day [1], Spotify's catalog is a huge database easily accessible thanks to Spotify's API [2], which allows to get detailed information for each track, linked both to physical proprieties of track's sound, both to perceptive features. Missing though public information about streams number (the API only provides a `popularity` value, but it is too much related with track's oldness to be reliable) we have used a Kaggle's database [3] to know if a track was an *hit* (is or has been popular) or not.

Then we tried to design or to learn a *bayesian network* to analyze which features make a song successful and which are instead irrelevant. The code is available on a public GitHub repository.

2 Data and preprocessing

The dataset is taken from *Kaggle - Hit song science - 34740 songs (+spotify features)* [3]. It contains 34740 entries representing songs. Each entry has the following fields:

- `artist_name`: the name of the artist;
- `danceability`: how suitable the track is for dancing based on a combination of musical elements. 0.0 is least danceable, while 1.0 is most danceable;
- `energy`: a perceptual measure of intensity and activity based on dynamic range, perceived loudness, timbre, onset rate and general entropy. 0.0 is least energetic, while 1.0 is most energetic;
- `key`: the estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C?/D?, 2 = D, and so on. If no key was detected, the value is -1.
- `loudness`: the overall loudness of the track in decibels;
- `mode`: indicate if the track is in minor (0) or major (1) modality;
- `speechiness`: the presence of spoken words in a track. 1.0 is a track made entirely of spoken words, while 0.0 is a track without spoken words;
- `acousticness`: a confidence measure of whether the track is acoustic. 0.0 is least confidence, 1.0 is most confidence;
- `instrumentalness`: a confidence measure of whether the track contains no vocal. 0.0 is least confidence, 1.0 is most confidence;
- `liveness`: a confidence measure of whether the track was performed live. 0.0 is least confidence, 1.0 is most confidence;
- `valence`: a perceptual measure of the musical positiveness of the track. 0.0 is the most negative (angry, sad) song, while 1.0 is the most positive (happy, cheerful) one;
- `tempo`: the overall estimated tempo of the track in beats per minute (BPM);
- `duration_ms`: the duration of the track in milliseconds;
- `On_chart`: 1 for hit (songs that have been at least one time on the 100 position charts), 0 for non-hit;

and also other features which are not used in this project because reputed irrelevant.

The database present oversampling problems, since the number of hits and non hits is almost the same, while in the reality there are more non-hit songs than hit songs. However this is not a big deal for this project, since even if we should have general higher probabilities for a song to be an hit, we are still be able to see which features are more important than others.

Before constructing the network we need to preprocess the data according to the following steps:

1. Remove all the useless features (i.e. the ones not listed above);
2. discretize continuous features, since `pmgpy` does not support continuous variables;
3. make the features range smaller, to reduce the RAM needed by `pgmpy` to generate the network.

In detail, `artist_name` is removed. Regarding the other features they are preprocessed as illustrated in table 1. `instrumentalness` and `liveness` have been mapped to Boolean values according to Spotify API's Documentation [2] which states that tracks with an `intrumentalness` value above 0.5 represent instrumental song, while tracks with a `liveness` value above 0.8 represent have a strong likelihood to be live.

Mapped from continuous range $[0, 1]$ to discrete range $[0, 4]$	Mapped to quintiles (5-quantiles)	Mapped to Boolean values
`energy`, `acousticness`, `danceability`, `speechiness`, `valence`	`loudness`, `tempo`, `duration_ms`	`instrumentalness`, `liveness`

Table 1: Preprocessing for each feature

3 The network

A Bayesian network is formally an acyclic directed graph where every nodes correspond to a random variable and edges correspond to conditional dependencies. Indeed an edge from node X to node Y means that the *parent* node X has a *direct influence* on Y.

Building a Bayesian network is a two-steps process. At first we have to define network's structure (i.e. the set of nodes and edges), then we must define a set of parameter for each node corresponding to the *conditional probability distribution* (CPD) quantifying the influences of the parents on that node.

Both steps can be done either manually or by an automatic process. In this project the structure has been both defined manually and learned by an algorithm, in order to see which approach gives the best results.

3.1 Defining the structure

At first we have defined the network according to what we know about the feature: `On_chart` is influenced by every other features. Also from Spotify API's documentation we know (or we can simply deduct) that `tempo` influences `danceability`; `speechiness` influences `danceability` and `instrumentalness`; `loudness` influences `energy` which influences `valence`.

We obtain the network in figure 1. The problem with this network is that is too complex to calculate the weights. According to `pgmpy` it should requires to allocate an array of 3200 TB, so we will try with other solutions.

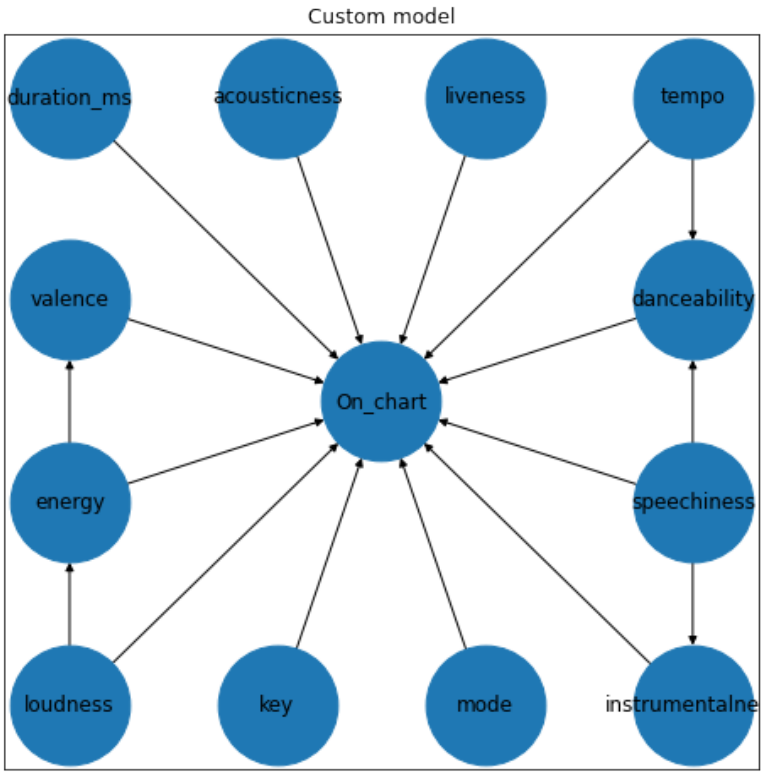


Figure 1

3.2 Learning the structure

The problem of learning the structure of a Bayesian Network is computationally very expensive, since number of possible graphs grow super-exponentially over the number of nodes n [4]. One of the earliest and easiest solution is Chow and Liu's tree-structured network [5].

Unfortunately, due to the nature of our problem, which makes difficult to choose a root, it does not return good results, as you can see in figure 2. Changing the root we obtain similar results, so we need more complex algorithms.

Modern algorithms usually belong to one of two different types: *score-based* and *constraint-based*. Algorithms of the former type solve the optimization problem

$$\arg\max_{G} score(G, D)$$

where $score(G, D)$ is a given scoring function which measure how much the graph G fits the database D. We have choose one of the most common scoring functions, which is is the *Bayesian Dirichlet equivalent uniform* (BDeu), that, assuming a uniform prior probability over all possible direct acyclic graphs, computes the posterior probability of a certain graph given the data [6]. Beside to the score function, we also need an algorithm to perform the search the (local) maximum. We chose the *Hill Climb Search*, a local-search algorithm.

Figure 3 shows the learned network, that actually has some problems:

- `On_chart` can not directly influence anyone of the other features, since at first a track is made, with all its features, and only then it becomes an hit or not. Not being clear if the best solution is to remove these edges or to invert their direction, we have tried both ways further on;
- the edge `tempo` $\longrightarrow$ `danceability`, which we know for sure to be true, is missing.

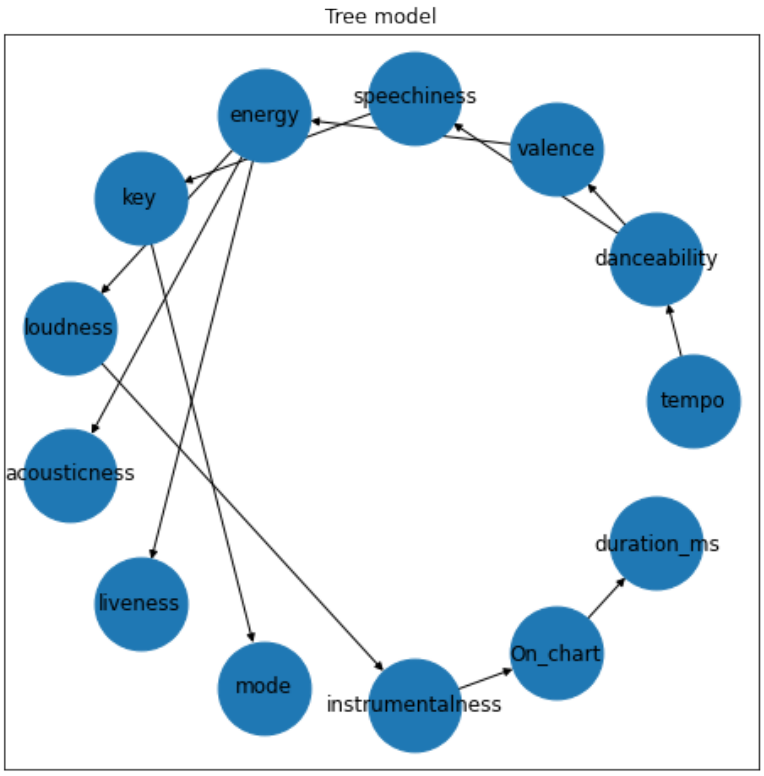


Figure 2

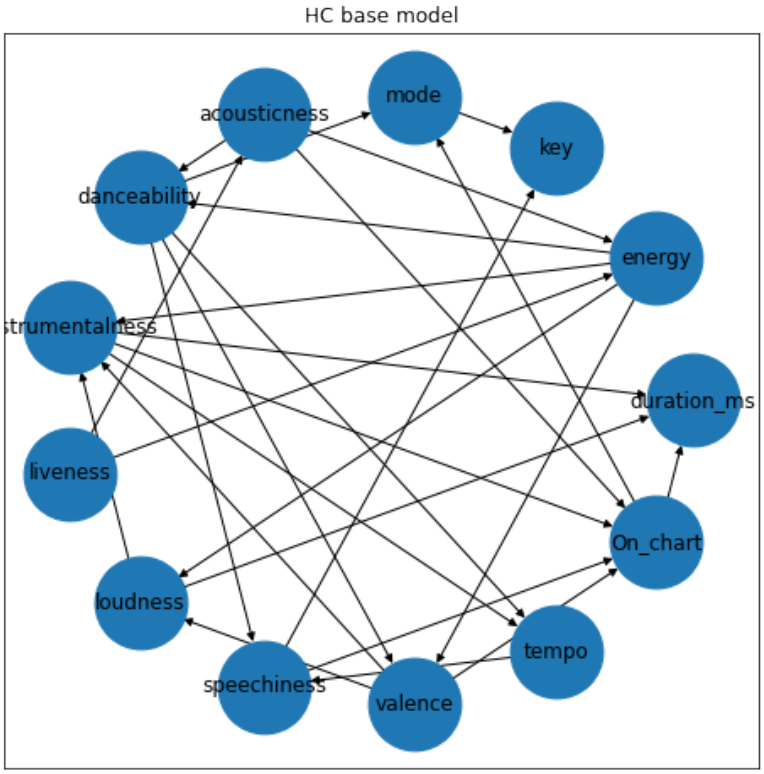


Figure 3

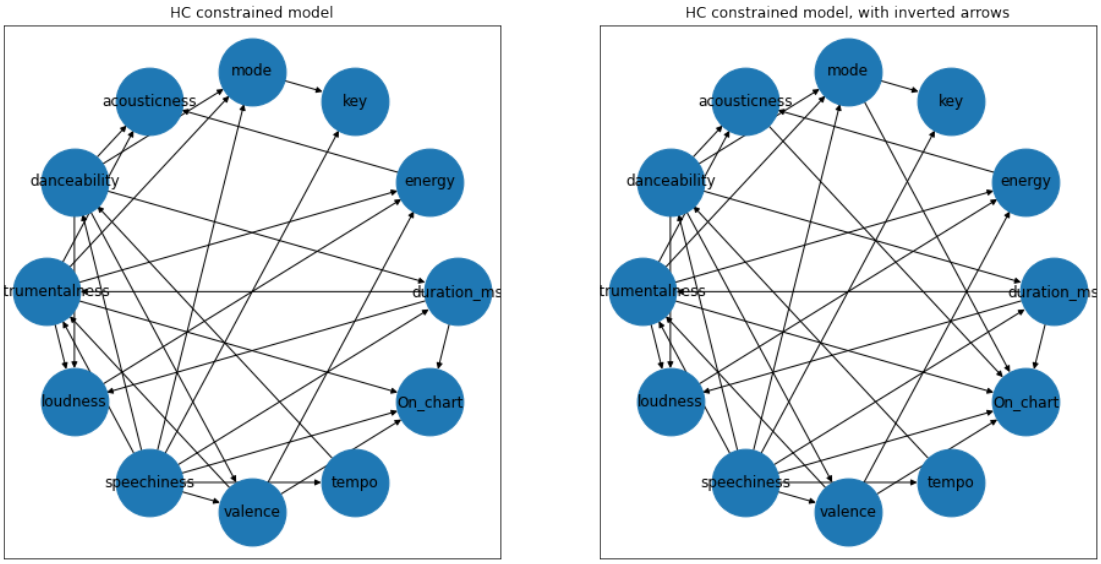


Figure 4

Fortunately, `pgmpy` provides a way to inject some prior knowledge in the learning process, so these problems can be easily solved. However this operation require too much RAM and since variables ranges have already been widely reduced, we drop a feature to reduce computational complexity. The chosen feature is the one that, watching the network, looks to have less influence on `On_chart`: `liveness`.

So, we obtain networks in figure 4. The first graph is obtained removing impossible edges, while the latter inverting their direction, as discussed above.

4 Learning the parameters

After having defined the graph structure, the next step is to learn network's parameters. Since our network have complete data (there are no uncertain or missing entries in any feature) we can rely on *Maximum Likelihood Estimation* (MLE). Given a bayesian network, its parameters $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ (with θ_i the conditional probability distribution of network's variable X_i) and a *global likelihood function* $\mathcal{L}(\theta|\mathcal{D})$ defined as

$$\mathcal{L}(\theta|\mathcal{D}) = \prod_i \ell_i(\theta_i|\mathcal{D})$$

where $\ell_i(\theta_i|\mathcal{D})$ is the *local conditional likelihood* associated with variable X_i and defined as

$$\ell_i(\theta_i|\mathcal{D}) = \prod_m \mathbf{P}(x_i = m|Parents(X_i) = m, \theta_i)$$

The goal of maximum likelihood estimation is to solve the optimization problem

$$\hat{\theta} = \arg\max_\theta \mathcal{L}(\theta|\mathcal{D})$$

Since, as explained in section 3.2, the custom model requires to much RAM to have parameters estimated, it has been omitted in this part. Also the tree model has been omitted, due to its inaccurate looking structure.

So, parameters has been estimated for the three Hill Climb models: the base one, the constrained one and the constrained with inverted arrows one.

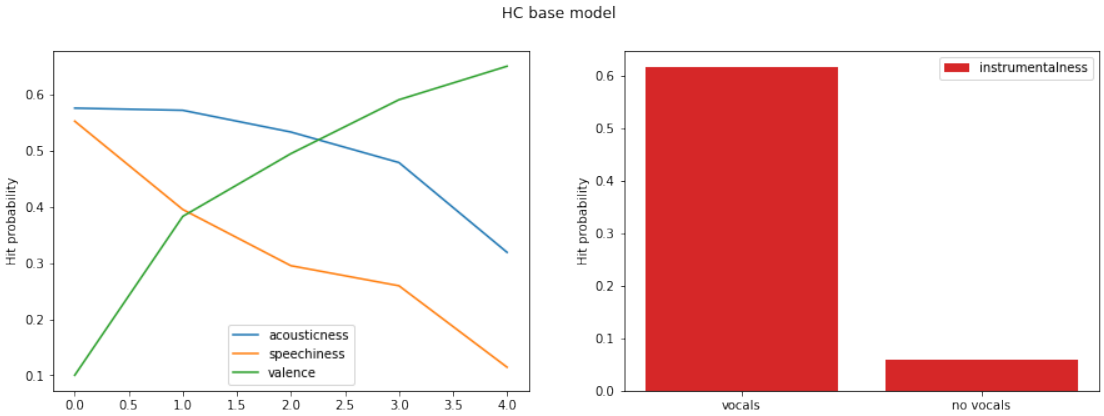


Figure 5

5 Probabilistic reasoning

Now that we have networks and their parameters, we can test them performing some inferences. Since we have three different model we have tested them separately.

5.1 Hill Climb base model

In this model the features that directly influence `On_chart` are:

- `acousticness`, range $\{0, ..., 4\}$;
- `instrumentalness`, range $\{0, 1\}$;
- `speechiness`, range $\{0, ..., 4\}$;
- `valence`, range $\{0, ..., 4\}$.

To see how they influence the track probability to be an hit we execute a set of queries of the form

$$\mathbf{P}(On\_chart = 1 | acousticness = a)$$

$$\mathbf{P}(On\_chart = 1 | instrumentalness = i)$$

$$\mathbf{P}(On\_chart = 1 | speechiness = s)$$

$$\mathbf{P}(On\_chart = 1 | valence = v)$$

with $a, s, v \in \{0, \ldots, 4\}$ and $i \in \{0, 1\}$. Results are shown in figure 5.

A first consideration on probabilities values must be done. As already discussed in section 2, this dataset is oversampled, this leads to the unrealistically high probability of ~ 0.6 for any song with vocals, for example, to be an hit. Obviously the real probabilities are way lower, because in real world only very few tracks become hits. However this not prevent us from analyzing which feature influence the popularity of a song and in which way.

Made this necessary premise, we observe the following:

- Instrumentalness seems to have a huge impact: tracks with no vocals have less than 10% probability of be hits;
- speechiness has also an important role, and this was quite expected, since according to Spotify documentation a song should have a speechiness value between 0 and 0.33 (0 and 2 after preproccessing), with the exceptions of rap songs which span between 0.33 and 0.66 (2 and 3). Tracks above 0.66 are that are probably made entirely of spoken words, like podcasts;

- non-acoustic tracks have higher probabilities, but it seems to be the less impacting factor.
- valence is very relevant, with negative songs having almost no chance.

We want to see now what happens to the other 3 features when we fix `instrumentalness=0` (tracks with vocals). This can be done with the set of queries

$$\mathbf{P}(On\_chart = 1 | instrumentalness = 0, acousticness = a, speechiness = s, valence = v)$$

Results, shown in figure 6, present a curious behaviour: sad or angry songs (valence 0) have higher probabilities when the speechiness is 3, which should corresponds to rap songs, and the acousticness is 2.

5.2 Hill Climb constrained model

In this model the features that directly influence `On_chart` are:

- `duration_ms`, range $\{0, ..., 4\}$;
- `instrumentalness`, range $\{0, 1\}$;
- `speechiness`, range $\{0, ..., 4\}$;
- `valence`, range $\{0, ..., 4\}$.

To see how they influence the track probability to be an hit we execute a set of queries of the form

$$\mathbf{P}(On\_chart = 1 | duration\_ms = d)$$
$$\mathbf{P}(On\_chart = 1 | instrumentalness = i)$$
$$\mathbf{P}(On\_chart = 1 | speechiness = s)$$
$$\mathbf{P}(On\_chart = 1 | valence = v)$$

with $d, s, v \in \{0, \dots, 4\}$ and $i \in \{0, 1\}$. Results are shown in figure 7.

Speechiness, valence and instrumentalness have a behavior similar to the one they have in the base model, while long tracks seem to have more success.

Also for this model, we want to see now what happens to the other 3 features when we fix `instrumentalness=0` (tracks with vocals). This can be done with the set of queries

$$\mathbf{P}(On\_chart = 1 | instrumentalness = 0, duration\_ms = d, speechiness = s, valence = v)$$

Results are shown in figure 8.

5.3 Hill Climb constrained model, with inverted arrows

The last model we test is the one obtained imposing the edges $X \longrightarrow On\_chart$ for each X such that $On\_chart \longrightarrow X$ is an edge present in the base model. As already explained, this is done because `On_chart` can not directly influence any of the other features, but it can be only influenced from them.

In the model obtained the features that directly influence `On_chart` are:

- `duration_ms`, range $\{0, ..., 4\}$;
- `acousticness`, range $\{0, ..., 4\}$;
- `instrumentalness`, range $\{0, 1\}$;
- `speechiness`, range $\{0, ..., 4\}$;
- `valence`, range $\{0, ..., 4\}$;
- `mode`, range $\{0, 1\}$.

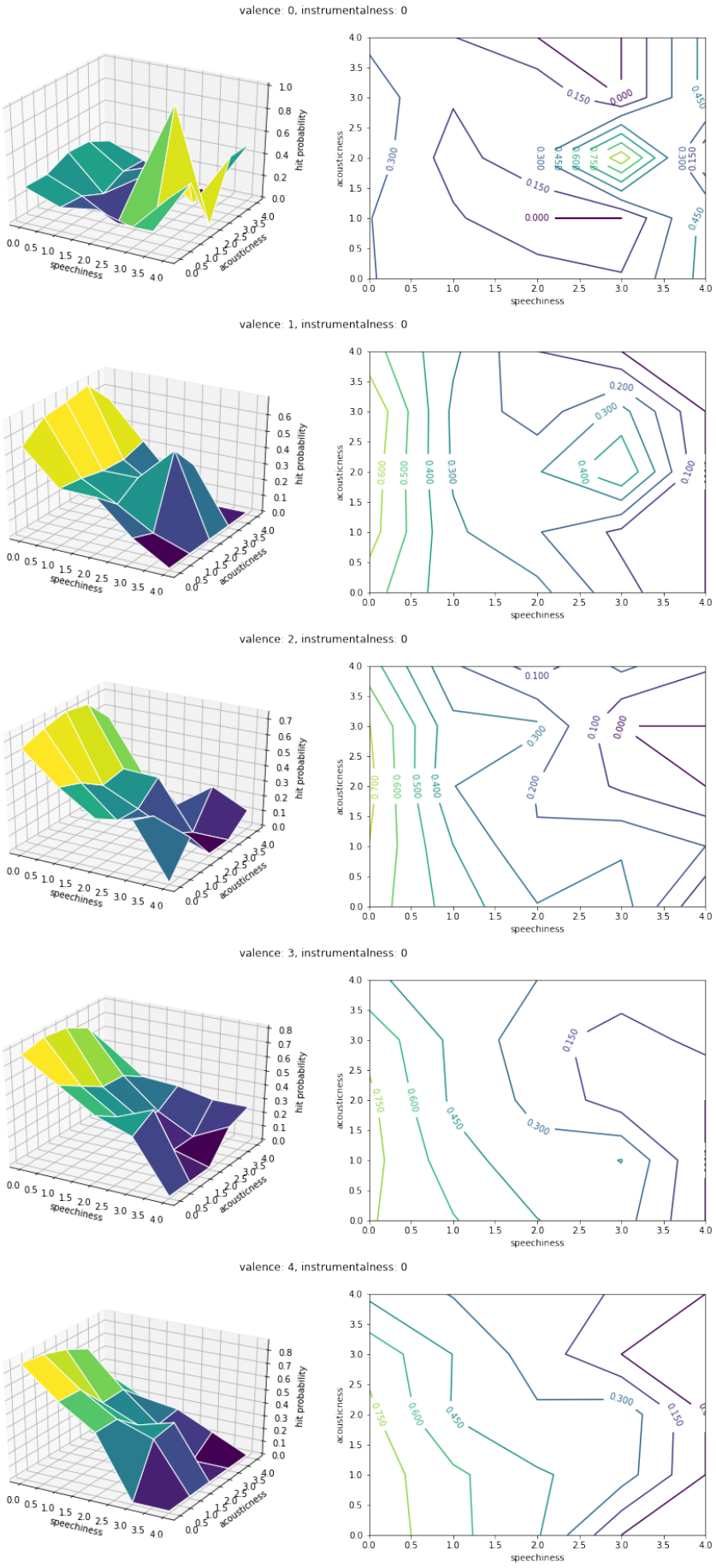

valence: 0, instrumentalness: 0
valence: 1, instrumentalness: 0
valence: 2, instrumentalness: 0
valence: 3, instrumentalness: 0
valence: 4, instrumentalness: 0
speechiness
acousticness
hit probability

Figure 6

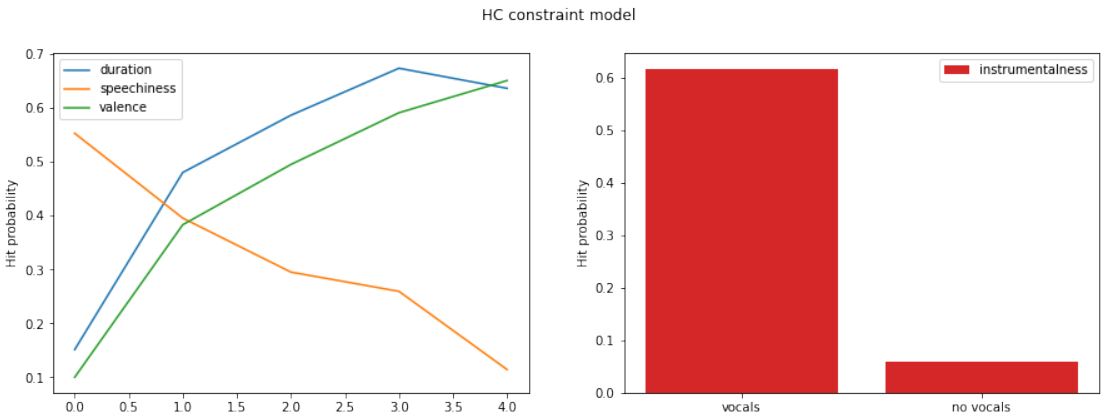


Figure 7

To see how they influence tracks' probabilities to be an hit we execute a set of queries of the form

$$\mathbf{P}(On\_chart = 1|duration\_ms = d)$$

$$\mathbf{P}(On\_chart = 1|acousticness = a)$$

$$\mathbf{P}(On\_chart = 1|instrumentalness = i)$$

$$\mathbf{P}(On\_chart = 1|speechiness = s)$$

$$\mathbf{P}(On\_chart = 1|valence = v)$$

$$\mathbf{P}(On\_chart = 1|mode = m)$$

with $d, a, s, v \in \{0, \ldots, 4\}$ and $i, m \in \{0, 1\}$.

Results, shown in figure 9, seem to sum up what emerge from the other two models and they also tell us that major tracks have a bit more chances to become hits, but the difference seems to be negligible.

We have only analyzed features that directly influence `On_chart` (its parents), but the generated models tell us also about other features, with a lot of possible combinations. For example we could want to know if there is a best tempo to choose if we want to public on Spotify an angry (valence=0) metal (energy=4, loudness=4, acousticness=0, speechiness=0) song with vocals (instrumentalness=0) and we want it to become an hit. This can be solved by running the set of queries of the form

$$\mathbf{P}(On\_chart = 1|tempo = t, valence = 0, energy = 4, loudness = 4, \\ acousticness = 0, speechiness = 0, instrumentalness = 0)$$

with $t \in \{0, ;4\}$. What if we want it to be danceable? We need to run the queries

$$\mathbf{P}(On\_danceable = d|tempo = t, valence = 0, energy = 4, loudness = 4, \\ acousticness = 0, speechiness = 0, instrumentalness = 0)$$

with $d, t \in \{0, ;4\}$.

The answer, shown in figure 10, is that there is not a best possible tempo to make the song a hit, since it is irrelevant (indeed, but if we choose a tempo in the second quintile at least we have the highest probability to make it danceable.

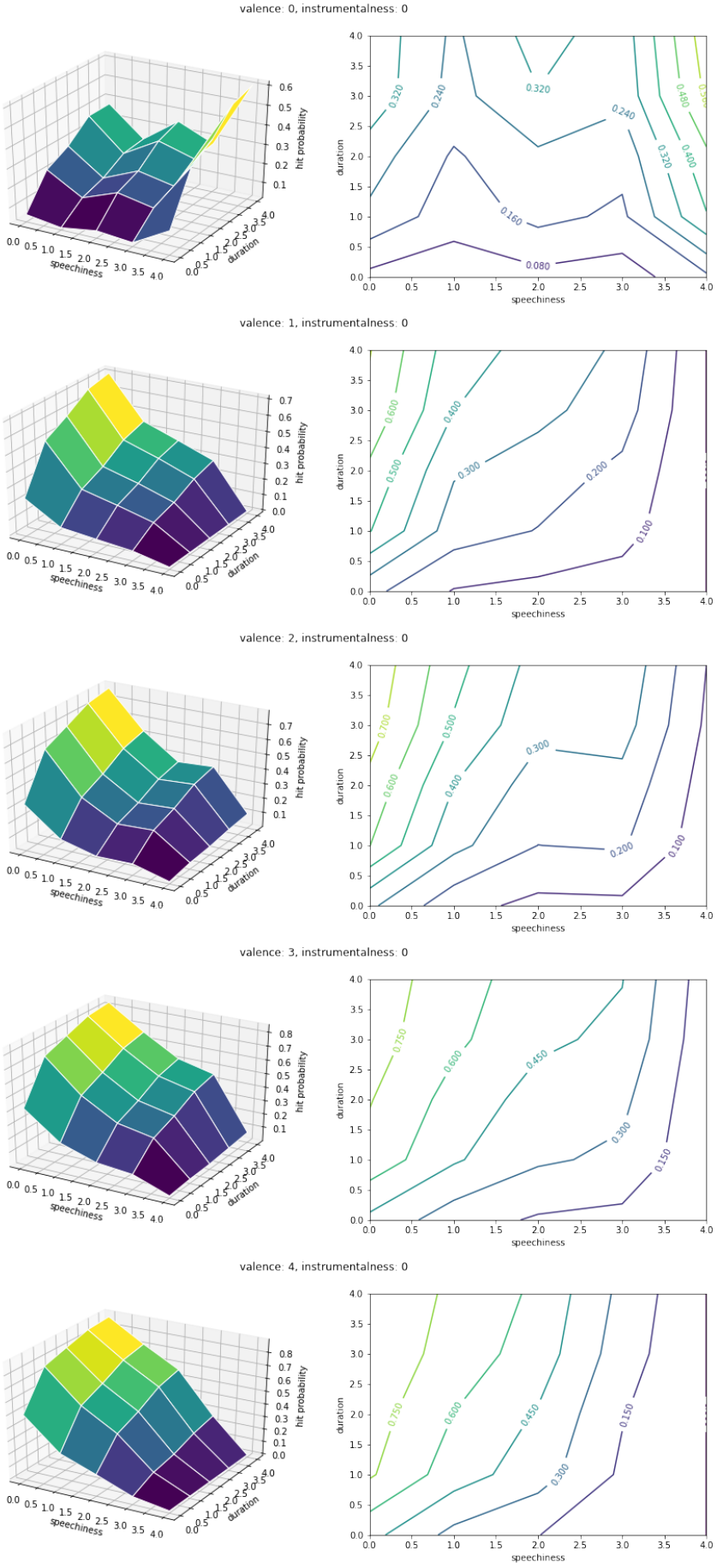

valence: 0, instrumentalness: 0
valence: 1, instrumentalness: 0
valence: 2, instrumentalness: 0
valence: 3, instrumentalness: 0
valence: 4, instrumentalness: 0
speechiness
duration
hit probability

Figure 8

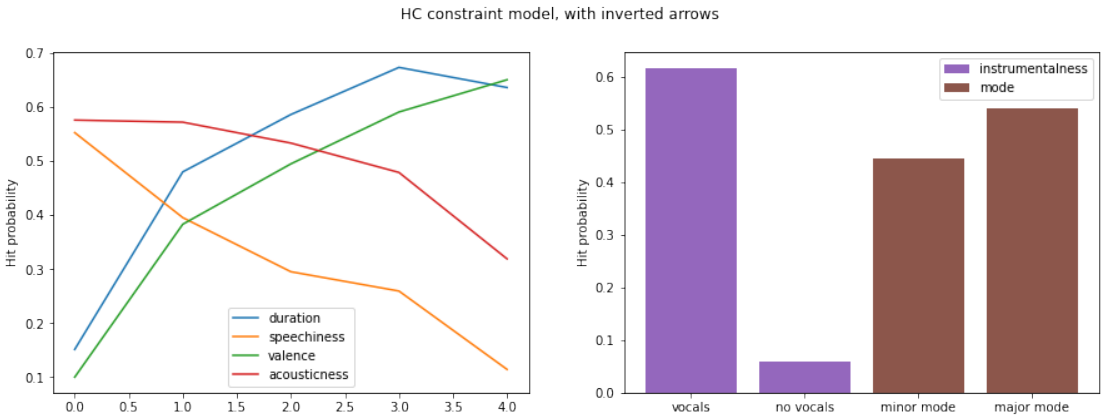


Figure 9

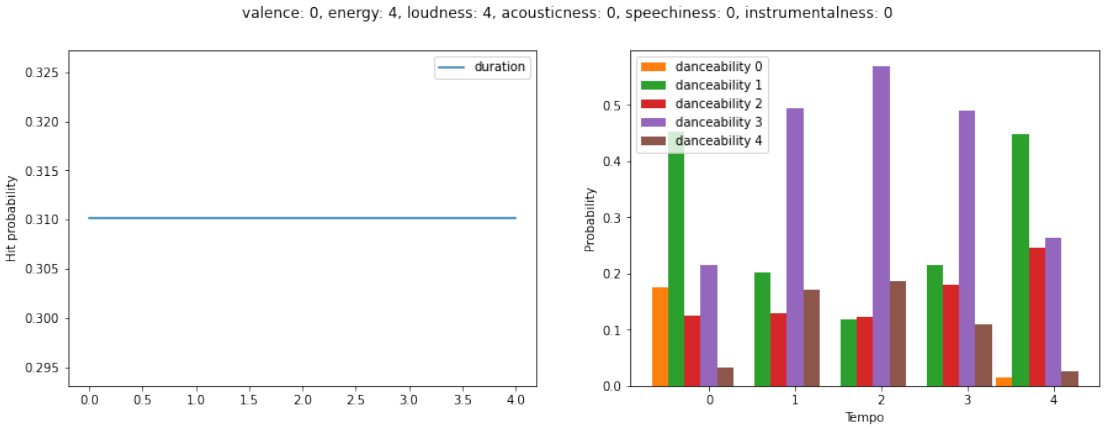


Figure 10

6 Conclusions

Bayesian network is a powerful tool that allows to represent complex probabilistic relationships and dependencies, however the heuristics currently used to overcome the huge computational complexity needed from this model lack in quality, in particular in graph's structure generating process. Parameters' learning looks more reliable, but require too many resources for some customize models, making it unusable sometimes.

In this project we have shown that bayesian networks are proper tools to analyze songs' features and to try to predict which will be hits, but a not oversampled dataset is of course needed, in order to improve predictions' quality.

References

[1] Spotify, "Spotify stream on." [Online]. Available: https://www.youtube.com/watch?v=Vvo-2MrSgFE

[2] ——, "Web api reference." [Online]. Available: https://developer.spotify.com/documentation/web-api/reference/

[3] S. theodoropoulos, "Hit song science - 34740 songs (+spotify features)," 2021. [Online]. Available: https://www.kaggle.com/multispiros/34740-hit-and-nonhit-songs-spotify-features

[4] M. Scanagatta, A. Salmerón, and F. Stella, "A survey on bayesian network structure learning from data," *Progress in Artificial Intelligence*, vol. 8, no. 4, pp. 425–439, December 2019.

[5] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 462–467, May 1968.

[6] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning bayesian networks: The combination of knowledge and statistical data," *Machine Learning*, vol. 20, no. 3, pp. 197–243, 1995.