Bayesian Network on Spotify top50 ranking

Marco Aspromonte ,Francesco Romito

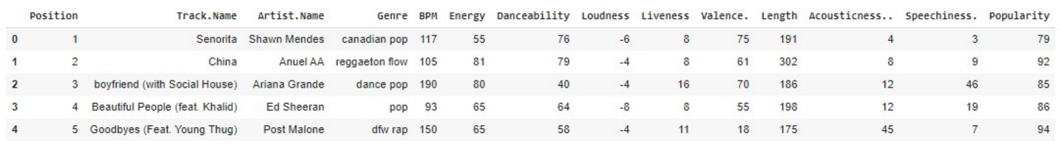
June 28, 2021

Abstract

1 The Datasets

1.1 A brief description

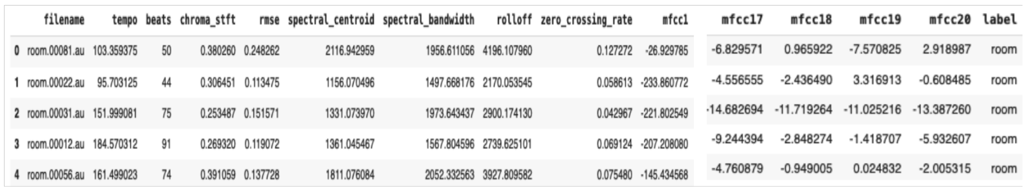
The chosen datasets have been downloaded from Kaggle.The first dataset, called 'top50.txt', contains the data of the top 50 songs of Spotify ranking, grouped by the position. The other attributes are more general properties of the song as a music product, so here we have a tabular representation of it:



	Position	Track.Name	Artist.Name	Genre	BPM	Energy	Danceability	Loudness	Liveness	Valence.	Length	Acousticness..	Speechiness.	Popularity
0	1	Senorita	Shawn Mendes	canadian pop	117	55	76	-6	8	75	191	4	3	79
1	2	China	Anuel AA	reggaeton flow	105	81	79	-4	8	61	302	8	9	92
2	3	boyfriend (with Social House)	Ariana Grande	dance pop	190	80	40	-4	16	70	186	12	46	85
3	4	Beautiful People (feat. Khalid)	Ed Sheeran	pop	93	65	64	-8	8	55	198	12	19	86
4	5	Goodbyes (Feat. Young Thug)	Post Malone	dfw rap	150	65	58	-4	11	18	175	45	7	94

Figure 1: top50 dataset

While the second dataset called 'genres.csv' consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks.



	filename	tempo	beats	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rate	mfcc1	mfcc17	mfcc18	mfcc19	mfcc20	label
0	room.00081.au	103.359375	50	0.380260	0.248262	2116.942959	1956.611056	4196.107960	0.127272	-26.929785	-6.829571	0.965922	-7.570825	2.918987	room
1	room.00022.au	95.703125	44	0.306451	0.113475	1156.070496	1497.668176	2170.053545	0.058613	-233.860772	-4.556555	-2.436490	3.316913	-0.608485	room
2	room.00031.au	151.999081	75	0.253487	0.151571	1331.073970	1973.643437	2900.174130	0.042967	-221.802549	-14.682694	-11.719264	-11.025216	-13.387260	room
3	room.00012.au	184.570312	91	0.269320	0.119072	1361.045467	1567.804596	2739.625101	0.069124	-207.208080	-9.244394	-2.848274	-1.418707	-5.932607	room
4	room.00056.au	161.499023	74	0.391059	0.137728	1811.076084	2052.332563	3927.809582	0.075480	-145.434568	-4.760879	-0.949005	0.024832	-2.005315	room

Figure 2: sound properties dataset

Since we have two datasets and we want to mashup them together, we can continue with a preprocessing phase.

1.2 Preprocessing

After importing the two datasets mentioned above, in order to manage them in a convenient way we convert them into pandas dataframe. For the first, given that it has a `txt` extension, we first import it as a numpy array, then cast it into a pandas dataframe:

```
X = np.loadtxt('top50.txt', delimiter=',',dtype=str)
df = pd.DataFrame(X, columns=["Position","Track.Name","Artist.Name","Genre",
                              "BPM","Energy","Danceability","Loudness",
                              "Liveness","Valence.","Length","Acousticness..",
                              "Speechiness.","Popularity"])
```

After that we drop the NaN and the values that we do not use for the mashup :

```
df = df.drop(['Track.Name'],axis=1)
df = df.drop(['Artist.Name'],axis=1)
df = df.drop(['Danceability'],axis=1)
df = df.drop(['Liveness'],axis=1)
df = df.drop(['Valence.'],axis=1)
df = df.drop(['Acousticness..'],axis=1)
df = df.drop(['Speechiness.'],axis=1)
```

Since that in the top50 dataset there are many specific categories, we decide to merge them in some macro-ones, if they are musically compatible. Such procedure is explained in the following piece of code:

```
for i in range(0,50):
  if 'pop' in df['Genre'][i] or 'brostep' in df['Genre'][i]:
    df['Genre'][i]='pop'
  elif 'rap' in df['Genre'][i] or 'trap' in df['Genre'][i]:
    df['Genre'][i]='rap'
  elif 'edm' in df['Genre'][i] or 'room' in df['Genre'][i]:
    df['Genre'][i]='room'
 ...
```

Now that we have introduced the main pre-processing phase of the top50 dataset, let's discuss about the second one that is already in `csv` format, so we import it directly as a pandas dataframe. Now, in order to consider only the column we need, i.e. the spectral bandwidth attribute, we drop all the others and the NaN values:

```
for i in range(1,21):
  df_gen = df_gen.drop(['mfcc{}'.format(i)],axis=1)

df_gen = pd.read_csv('genres.csv')
...

df = df.dropna()
df_gen = df_gen.dropna()
df = df.replace({'"':''}, regex=True)
```

Since in the top50 dataset we do not have all the samples that belong to the genres dataset , we decided to remove some of them, if they are for example strongly different from the others:

```
df_gen = df_gen[~df_gen['label'].astype('str').str.contains('classical')]
df_gen = df_gen[~df_gen['label'].astype('str').str.contains('country')]
df_gen = df_gen[~df_gen['label'].astype('str').str.contains('disco')]
```

Considering the genres, what is convenient to do is just to classify them trough the spectral bandwidth, this is done by computing the mean of the SB of the samples :

```
def mean_gen(label_str):
  count, counter = 0,0
  for i in range(0,df_gen.shape[0]-1):
    if df_gen['label'].iloc[i] == label_str:
      count = count + 1
      counter = counter + df_gen['spectral_bandwidth'].iloc[i]
  mean = counter / count
  return mean
```

In this way the SB becomes a key attribute useful to identify the relationship between each song sample and its corresponding genre. Such correspondence trough SB is defined in a chain of conditional operators in which each song sample of a particular genre acquires the mean value of the SB plus a random bias that make it different from the other samples of its own category and introduce a bit of variance in the model. Now we can proceed with the mashup phase of the two datasets:

```
df['band_width'] = pd.Series(np.random.randn(len(df['Genre'])), index=df.index)
```

However it was convenient to get for each song sample a binarized value for each attribute, by considering a treshold operator that split each attribute in two main classes of high' and 'low' if they are respectively higher or lower than the mean of such attribute that is easily computed with the mean() function, that also here becomes a treshold operator:

```
...
for i in range(0,50):
  if (df['BPM'][i] > bpm_mean):
    df['BPM'][i] = 'high'
  else:
    df['BPM'][i] = 'low'

  if (df['Energy'][i]>  energy_mean):
    df['Energy'][i] = 'high'
  else:
    df['Energy'][i] = 'low'
......
```

While considering the Position field, it was more appropriate to split the values also in a 'medium' range, as follows:

```
    if (df['Position'][i]< pos_len/3 ):
    df['Position'][i] = 'high'
  elif (df['Position'][i]> 2* pos_len/3 ):
    df['Position'][i] = 'low'
  else:
    df['Position'][i] = 'medium'
```

Now that we have a cleaned and get a usable dataset, we can proceed with the Bayesian Network construction.

2 Bayesian Network

2.1 The model

In order to represent the set of our variables and their conditional dependencies and indipendencies trough a Bayesian Model, first we consider the `pgmpy` library, from where we obtain the BayesianModel function, which takes as parameters our variables to build the acyclic graph.

```
model = BayesianModel([('band_width', 'Genre'),('BPM', 'Genre'),
                       ('Genre', 'Position'),('Energy', 'Position'),
                       ('Loudness', 'Position'),('Length', 'Position'),
                       ('Position', 'Popularity')])
```

In order to fit the model to our dataset, we call the `fit` function taking as parameters our dataframe and the estimator class. Then in order to check the validity of the model, a `check_model` function is called. We can now perform a graphical representation of the model in order to check the `parents` and the `children`:

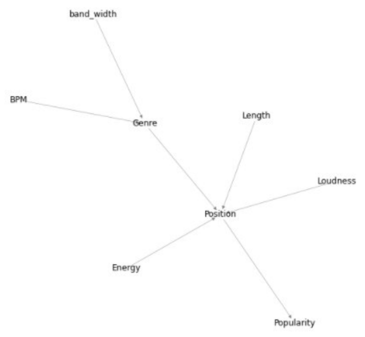


Figure 3: Bayesian Model graph

2.2 The estimator

Now that we have defined the structure of the dependencies, we have to associate to each node of the graph its probability distribution by an estimator. Since we have to construct the CPTs table for each node of the graph, instead of define them by hand, we use the `BayesianEstimator()` in the fitting function, taking as estimator parameter the Bayesian one.

```
model.fit(df, estimator=BayesianEstimator)
```

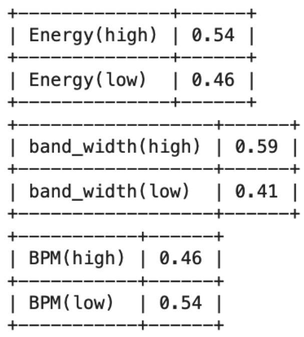
Even if the tables are already created, in order to check the values of conditional probability by an analysis on the resulting tables, we use the function `cpd_estimator` and print all the CPTs for each node:

```
b_est = BayesianEstimator(model, df)

def cdp_estimator(property):
  return b_est.estimate_cpd(property, prior_type='BDeu', equivalent_sample_size=50)

cdp_band = cdp_estimator('band_width')
cpd_bpm = cdp_estimator('BPM')
cpd_energy = cdp_estimator('Energy')
cpd_loud = cdp_estimator('Loudness')
cpd_length = cdp_estimator('Length')
```

Then check if the sum of the probabilities in the tables is equal to 1:



```
+--------------+------+
| Energy(high) | 0.54 |
+--------------+------+
| Energy(low)  | 0.46 |
+--------------+------+
+-----------------+------+
| band_width(high) | 0.59 |
+-----------------+------+
| band_width(low)  | 0.41 |
+-----------------+------+
+-----------+------+
| BPM(high) | 0.46 |
+-----------+------+
| BPM(low)  | 0.54 |
+-----------+------+
```

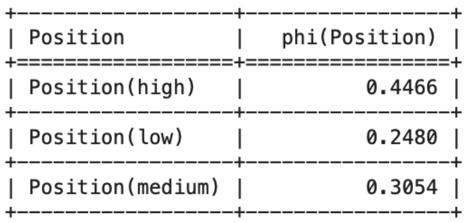
Figure 4: CPTs bpm, bandwidth and energy

2.3 Queries and results

We can now proceed to build some queries in order to inference the indipendencies. The first thing that we have to do is to create an instance that deploys the variable elimination on the model for each query. The variable elimination consists in delete the irrilevant values carrying out probability summation right-to-left, storing intermediate results to avoid recomputation. This relevancy attribute is given by the d-separacy rule. Here we have such instance:

```
exact_inference = VariableElimination(model)
```

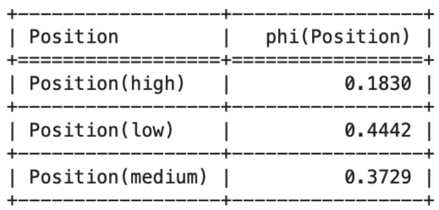
The first example of query concerns the Position attribute, so first we define the simple probability of Position as divided for high, medium and low, so approximately `33,33..%` to each one, because of the tresholding operation in the dataset preprocessing. An interesting result is that if the Genre is rap, then the probability that the Position in ranking will be low is small :



```
+------------------+-----------------+
| Position         |   phi(Position) |
+==================+=================+
| Position(high)   |          0.4466 |
+------------------+-----------------+
| Position(low)    |          0.2480 |
+------------------+-----------------+
| Position(medium) |          0.3054 |
+------------------+-----------------+
```

Figure 5: The CPT of Position given genre rap

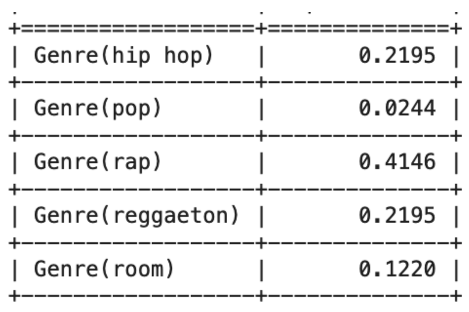
While if the Genre is room, the situation seems to be the opposite. A conclusion future we want



```
+------------------+-----------------+
| Position         |   phi(Position) |
+==================+=================+
| Position(high)   |          0.1830 |
+------------------+-----------------+
| Position(low)    |          0.4442 |
+------------------+-----------------+
| Position(medium) |          0.3729 |
+------------------+-----------------+
```

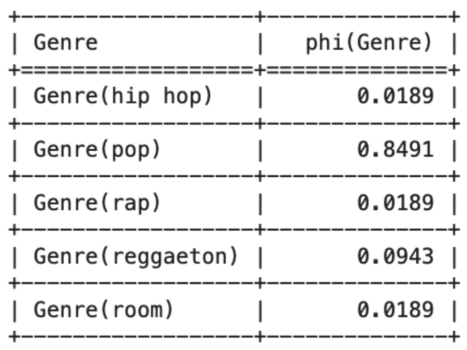
Figure 6: The CPT of Position given genre room

also to discuss, is that the most contribution in the change of Position is the Genre. Now if we want to identify a Genre, it is necessary to use the spectral bandwith and bpm since they are the features that we have used to classify them. For example:



```
+==================+==============+
| Genre(hip hop)   |       0.2195 |
+------------------+--------------+
| Genre(pop)       |       0.0244 |
+------------------+--------------+
| Genre(rap)       |       0.4146 |
+------------------+--------------+
| Genre(reggaeton) |       0.2195 |
+------------------+--------------+
| Genre(room)      |       0.1220 |
+------------------+--------------+
```

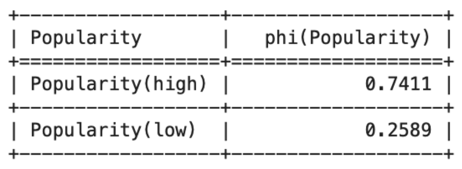
Figure 7: The CPT considering low bandwidth and high Bpm



```
+------------------+--------------+
| Genre            |   phi(Genre) |
+==================+==============+
| Genre(hip hop)   |       0.0189 |
+------------------+--------------+
| Genre(pop)       |       0.8491 |
+------------------+--------------+
| Genre(rap)       |       0.0189 |
+------------------+--------------+
| Genre(reggaeton) |       0.0943 |
+------------------+--------------+
| Genre(room)      |       0.0189 |
+------------------+--------------+
```

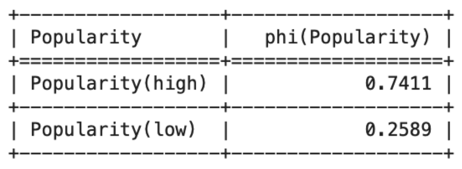
Figure 8: The CPT considering high bandwidth and high Bpm

Now we can check the indipendencies by considering some of these results. By the model representation we can see for example that Popularity is only a child of Position, so given the Position, all the other nodes have not to be influential in the Popularity's CPT definition. We can check this by printing the CPT associated to the Popularity given Position and Genre, and the other one given only the Position. The indipendence is verified if the two CPTs are equal, as it is the case.



```
+------------------+-------------------+
| Popularity       |   phi(Popularity) |
+==================+===================+
| Popularity(high) |            0.7411 |
+------------------+-------------------+
| Popularity(low)  |            0.2589 |
+------------------+-------------------+
```

Figure 9: The CPT of Popularity given high Position and pop Genre



```
+------------------+-------------------+
| Popularity       |   phi(Popularity) |
+==================+===================+
| Popularity(high) |            0.7411 |
+------------------+-------------------+
| Popularity(low)  |            0.2589 |
+------------------+-------------------+
```

Figure 10: The CPT of Popularity given only high Position

3 Some conclusions

By the implementation of this model it is interesting to see how a Bayesian Network could be used to predict the position of a song given its genre or some other feature, or to identify a genre given some generic musical or sound properties.

References

[Paolo Torroni] Uncertainty and probabilistic reasoning's slides

[pandas] Documentation on https://pandas.org

[Numpy] Documentation on https://numpy.org

[pgmpy] Documentation on https://pgmpy.org

[Kaggle top50Spotify] https://www.kaggle.com/sagittarius3/top50spotify

[Kaggle musical genres] https://www.kaggle.com/harish24/music-genre-classification