university of groningen / faculty of science and engineering / mathematics and applied mathematics

Bayesian Networks of Spotify's audio features

Bachelor's Project Mathematics

18 May 2020

Student: S.M.R. Kockelkorn

First supervisor: dr. M.A. Grzegorczyk

Second assessor: dr. R.I. van der Veen

Contents

1 Introduction

More than ever before in history, scientists and developers are analyzing music and looking into related applications of the analysis. [Luo, 2018] The modern ability to stream music using services such as Spotify, Pandora, and Apple music has revolutionized how music is consumed [Amsterdam, 2019] and has thereby also opened up many import areas of research. "There are researches that focus on underlying technologies, working mechanisms, user experience and other specific topics in music analysis field." [Luo, 2018]

Not only is there more demand for research from the streaming music industry, they also provide new research opportunities. Spotify for instance, created something called *Spotify API for developers*, an open database from which for every song on Spotify various audio features can be downloaded. Examples of such features are *danceability*, *loudness*, and *instrumentalness*.

Various research has been done with their openly accessible audio features. Topics include history of music, genre prediction [Luo, 2018], prediction of hit songs [Georgieva et al.], music recommendations and many more.

Bayesian Networks, a probabilistic graphical model, can be used to graphically represent conditional (in)dependencies between variables learnt from data. [Pearl, 2011] It combines elements from *graph theory* (directed acyclic graphs) and *probability theory* (Bayesian statistics [Bolstad and Curran, 2016]). Probabilistic models based on directed acyclic graphs (DAGs) such as Bayesian networks have a long and rich tradition, which began with the geneticist Sewall Wright (1921) [Pearl, 2011]. "Graphical modeling is an important method to efficiently represent and analyze uncertain information in knowledge-based systems. Its most prominent representatives are Bayesian networks and Markov networks for probabilistic reasoning." [Borgelt et al., 2002]

The use of Bayesian networks is, however, not the only method of graphical modeling, there are also *Gaussian Graphical Models* (GGMs) and *Relevance Networks* (RNs). [Werhli et al., 2006]. The problem of the Relevance Networks is that is can not distinguish between direct and indirect interactions. Therefore there are in general many false positives extracted. Graphical Gaussian Models have the disadvantage of rarely being mentioned in literature.

Why would one choose the use of Bayesian statistics over of methods? "MCMC simulations are indispensable when adopting a proper Bayesian approach to inference, which tends to be computationally less expensive than the frequentist approach of bootstrapping (Larget and Simon, 1999)." The Bayesian approach also guards against *overfitting*, as it includes an intrinsic penalty for unnecessary complexity [Cawley and Talbot, 2007].

In Bayesian statistics, usually the ultimate goal is to know the posterior distribution. It is denoted by $P(\text{graph}|\text{data})$, the probability of the graph given the data. Bayesian statistics is different in approach than classical or frequentist statistics. In classical statistics, the parameters are seen as fixed and the data as varying, whereas in Bayesian statistics it is exactly the other way around. The data is fixed and the parameters vary. This approach has some advantages and disadvantages, some would argue [Bolstad and Curran, 2016].

In classical statistics, a probability is seen as the relative frequency of a certain event in a long scenario of identical events. In Bayesian statistics, a probability is seen as the total number of correct outcomes divided by the total number of initial conditions. How certain are we of a specific outcome? This can be seen as more subjective, which is due to the choice of *prior*. The exact definition of a prior will be introduced in a couple of sentences. It is not necessarily more subjective, because by choosing the right informative prior and by having a relatively large enough data set, the choice of prior becomes less and less important. However, Bayesian statistics of course also has a lot of objectivity.

There are many important applications of Bayesian networks. This can be read as well in Iqbal. "The Bayesian Network (BN) is a widely applied technique for characterization and analysis of uncertainty in real-world domains. Thus, the real application of BN can be observed in a broad range of domains such as image processing, decision making, system reliability estimation and PPDM (Privacy Preserving in Data Mining) in association rule mining and medical domain analysis." [Iqbal et al., 2015] Although not mentioned above, many important applications come from systems biology, where gene-regulatory networks [Friedman et al., 2000] and protein pathways [Sachs et al., 2005] can be learnt from post-genomic data.

There are two variants of Bayesian networks, the dynamic and static variant. In this paper, only the static Bayesian network approach will be used on our data. Bayesian networks are also used in two different ways, first having the data and then providing most likely graph and then the other way around.

The conditional (in)dependencies can be learnt by using *Metropolis-Hastings Markov Chain*

Monte Carlo (MCMC) algorithms. In this paper, the *structure MCMC sampler* of Friedman and Koller [Friedman and Koller, 2003] will be used. This algorithm allows us to take a sample of the set of all graphs from the posterior distribution. $P(\text{graph}|\text{data})$ and represents the probability of the graph given the data. The greedy-search algorithm is an alternative method to the structure Metropolis-Hastings MCMC but will not be used, because it is used to search for the "single-best graph" and searching for that doesn't make a lot of sense if an inference will be made for large data sets. Which is usually the case. The linear Gaussian BGe scoring metrics as developed by Geiger and Heckerman [Geiger and Heckerman, 1994] will be used to describe a score for each graph such that multiple graphs can be compared.

Bayesian networks have been used before in relation to music, context-aware music recommendation system using fuzzy Bayesian networks with utility theory [Park et al., 2006]. This paper attempts to learn conditional (in)-dependencies between Spotify's audio features, studying different sample size data.

In this paper, first, there will be looked into Bayesian statistics, Bayesian networks, BGe score and structure Metropolis-hasting MCMC method and then those will all be applied to data sets containing 10 audio features for different sizes of songs column vectors. In this way, the dependencies between the variables will be learned. All the programming and computing will be done in R.

In order to run the structure MCMC algorithm, the number of iterations needs to be determined. This will be different for each separate size of the data set. The number of iterations used will be chosen by looking at some convergence diagnostics. The number of iterations required for convergence for different data sample sizes will be presented in a table. After that it will be repeated for 10 different data sets down sampled from the original data set for the sizes of $10, 100$ and 500 songs. By means of some scatter plots the similarities between those 10 different samples will be assessed and there will be looked into providing a explanation for that. After that also the averages of the results for different data samples of the same size will be taken. These will be compared in three different scatter plot. Whether or not they are (dis)similar, will be shown in the results section and an explanation for the outcome will be given.

After representing the conditional (in)-dependencies between the audio features in a graphical model, a verdict will be made whether or not the method used (Bayesian Networks) was appropriate for this particular predicted inference. This method for learning which graphs represent the data best is based on Bayesian statistics to compute the posterior probability for each graph. For computing the posterior probability, the BGe scoring metric will be used. There are however some alternatives that may lead to different results. The BDe likelihood is one of them. Then, it is important to compare all the posterior probabilities for each graph and choose the best graph. (This can be done by using greedy search) Most often, however, looking for the single best graph doesn't make any sense. In this case, we would rather use model averaging. We would like to check the relative frequency multiplied

by the posterior distribution for each edge. The posterior distribution is proportional to the marginal likelihood, in other words the score. Unfortunately computing the scores for all the graphs is too much to compute. This is where the Metropolis Hasting Structure MCMC comes into play. It enables us to take a sample from the posterior distribution.

This thesis is organized as follows: In 2 an introduction into Bayesian Networks is provided. After that in 3, information is given regarding the data set. In 4 the method used in this research is explained. After that the results and discussion and conclusion will be given.

In the last section a conclusion and discussion is provided.

2 Introduction Bayesian Networks

The alternatives for Bayesian Networks are, assuming everything is conditionally independent or assuming everything is conditionally dependent, bayesian networks are just in between that. That's why it is the best method for looking at causal interactions.

When can it be applied? The Static Bayesian Network method can be applied to data stemming from various domains if they satisfy the criteria. The conditions for the data set for which this method can be applied needs to have is an n by m matrix, where each column m is one independent observation of the n variables which the conditional (in) dependencies want to be learnt from. So the n variables make up the row and the columns are the independent observations. The entries, so the measurements of each observation per variable need to be continuous."Bayesian networks are ideal for taking an event that occurred and predicting the likelihood that anyone of several possible known causes was the contributing factor."

Why do we want to know conditional dependencies? The ultimate goal for Bayesian statistics would be to learn causal networks, but unfortunately, Bayesian Networks not necessarily represent that. Bayesian networks represent conditional (in)dependency relations -not necessarily causal interactions.

We would only like our not yet discussed method to differentiate between direct and indirect edges. We would like for example, let's say the amount measured of one hormone or anything is correlated to two others, but we would like to know if the number of spotted sharks had a direct correlation between the number of sales from ice cream or that maybe there is no direct edge and only two direct edges between the temperature on the beach and the spotted sharks and ice cream sales. Bayesian networks represent conditional (in)dependency relations not necessarily causal interactions.

Goal bayesian networks. Distinguish between direct and indirect edges include examples, t_1 and t_2 are dependent but conditional on the coin they're not dependent. In this case, if there would be only looked at dependencies relations then between t1 and t2 would be an

edge. But that can causally not be.

Introduction to graphs?probabilistic graphical models This part will be largely based on Koski and Noble [2011].

2.1 Independence and factorization of the joint probability function

This section is largely based on Koski and Noble [2011] and Bishop [2006].

Let us say there exists a random vector $\bar{X} = (X_1, \ldots, X_n)$ consisting of n random variables, defined on a continuous state space. Although the joint probability function $P(X_1, \ldots, X_n)$ contains full information about the n random variables $X_1, \ldots, X_n$ Koski and Noble [2011], it is usually not the most useful description in practice. The important features of the distribution may not be immediately clear in a complex joint distribution. Also, in many situations, the elementary building blocks of the joint probability function are low order conditional probabilities (will shortly be explained in more detail), each defined over small groups of variables and those conditional probability distributions are often easier to access. The following few paragraphs elaborate on those smaller building blocks of the joint probability function. First the notion of *independence* is defined.

Definition 2.1 (Independence)**.** *Two random variables X and Y are **independent** if and only if*

$$P(X = x, Y = y) = P(X = x)P(Y = y) \tag{1}$$

The notion of *joint independence* is defined as follows.

Definition 2.2 (Joint independence)**.** *A collection of n random variables $(X_1, \ldots, X_n)$ is said to be **jointly independent** if*

$$P(X_1, \ldots, X_n) = \Pi_{i=1}^{n} P(X_i) \tag{2}$$

As can be seen, if a collection of n random variables would be jointly independent, the equation above would already show the smaller building blocks of the joint probability function. In many cases, the n random variables that are under examination are not jointly independent but contain some conditional independencies. Soon it will be shown that the elementary building blocks of the joint probability function will be low order conditional probabilities. First, the notion of conditional probability will be defined and from there it will be built up to multiple definitions of conditional independence.

Definition 2.3 (Conditional Probability)**.** *For any two of the n random variables $X_1, \ldots, X_n$ the following holds, where the two are represented as X and Y. The probability of Y conditional on X $P(Y = y|X = x)$, describes the probability of Y being equal to its realisation*

y, given that you already know $X = x$. The ***conditional probability*** *is defined in the following way:*

$$P(Y = y|X = x) = \frac{P(X = x, Y = y)}{P(X = x)}. \tag{3}$$

The probability of Y conditional on X is given by the joint probability of X and Y divided by the marginal probability of X. Here, $P(X = x, Y = y)$ is the joint probability and $P(X = x)$, the marginal probability defined as

$$P(X = x) = \sum_{Y} P(X = x|Y = y). \tag{4}$$

Intuitively, this makes sense. Let us say there are only two possible outcomes of y namely either 0 or 1. If the probability of X being equal to 0 and the probability of X being equal to 1 are summed up, indeed one would receive $P(X = x)$. Sometimes the above equation is referred to as the *sum rule*. Besides the *sum rule*, a so-called *product rule* exists as well. A joint probability function of two variables can always be rewritten by using the product rule of probability in the following way:

$$\begin{aligned} P(X, Y) &= P(Y|X)P(x) \\ &= P(X|Y)P(Y) \end{aligned} \tag{5}$$

If the two variables, X and Y, would be independent then the following would hold.

$$P(X = x, Y = y) = P(X = x)P(Y = y), \tag{6}$$

which is the definition of X and Y being independent. If X and Y are independent, the following also holds.

$$\begin{aligned} P(Y = y|X = x) &= \frac{P(X = x, Y = y)}{P(X = x)} \\ &= \frac{P(X = x)P(Y = y)}{P(X = x)} \\ &= P(Y = y) \end{aligned} \tag{7}$$

where [1] was used in the second line. This make sense intuitively. If X and Y are independent, knowing the outcome of X would not change the probability of Y being equal to y. Now the concept of conditional dependence will be introduced. *Conditional dependence* is the concept of two or more events being dependent on the conditional of a third.

Definition 2.4 (Conditional independence)**.** *Given three random variables X, Y and Z, with $P(Z) > 0$, X and Y are* ***conditionally independent given Z*** *if*

$$P(X, Y|Z) = P(X|Z) \cdot P(Y|Z) \tag{8}$$

The following theorem about conditional independence holds.

Theorem 2.1. *X and Y are **conditionally independent** given Z if and only if*

$$P(X|Y,Z) = P(X|Z) \tag{9}$$

Proof.

$$\begin{aligned} P(X|Y,Z) &= \frac{P(X,Y,Z)}{P(Y,Z)} \\ &= \frac{P(X,Y|Z)\cdot P(Z)}{P(Y|Z)\cdot P(Z)} \\ &= \frac{P(X,Y|Z)}{P(Y|Z)} \\ &= \frac{P(X|Z)\cdot P(Y|Z)}{P(Y|Z)} \quad \text{by using8} \\ &= P(X|Z) \end{aligned} \tag{10}$$

□

Because of symmetry, one could also show the following.

Theorem 2.2. *If X and Y are conditionally independent given Z, then*

$$P(Y|X,Z) = P(Y|Z) \tag{11}$$

Another way of expressing the conditional independence is to write it as a factor of $P(X|Z)$ and $P(Y|Z)$. More precisely,

$$\begin{aligned} P(X,Y,Z) &= P(X,Y|Z)\cdot P(Z) \\ &= P(X|Z)\cdot P(Y|Z)\cdot P(Z) \text{by using (8)} \end{aligned} \tag{12}$$

As can been seen in the above definition, a specific factorization of the joint probability function is closely related to (conditional) independencies. By applying the product rule as given in (5) multiple times, every joint distribution can be factorized into various combinations of low order conditional probabilities, 'the elementary building blocks of the joint probability function'. As an example, this will be done with a joint distribution over three random variables a, b, c. The joint distribution $P(a,b,c)$ can be factorized by applying the product rule multiple times in six unique ways.

$$\begin{aligned} P(a,b,c) &= P(a|b,c)P(b,c) \\ &= P(a|b,c)P(b|C)P(c) \end{aligned} \tag{13}$$

As seen here, the first choice was picking a then b then c. So there are $3 \cdot 2 = 6$ possibilities. All are listed below.

$$\begin{aligned} P(a,b,c) &= P(a|b,c)P(b|c)P(c) \\ &= P(a|b,c)P(c|b)P(b) \\ &= P(b|a,c)P(a|c)P(c) \\ &= P(b|a,c)P(c|a)P(a) \\ &= P(c|a,b)P(a|b)P(b) \\ &= P(c|a,b)P(b|a)P(a) \end{aligned} \tag{14}$$

The above possible ways of factorization, will be used in the next few paragraphs to motivate the use of directed graphs to describe probability distributions. Let us represent the right-hand side of the last variation of the factorization in terms of a *simple graphical model* as follows.

A *simple graphical model* consists of nodes and edges. First for each of the random variables a, b, c a node will be introduced and each node is associated with the corresponding conditional distribution on the right-hand side of the last variation of the factorization. So, for example, node c is associated with the factor $P(c|a,b)$. Then for each conditional distribution, directed edges are added in the following way. A directed edge will be placed, with its head pointing to the node associated with the conditional distribution factor and its tail pointing to variables which the distribution is conditioned. Meaning for the factor $P(c|a,b)$, there will be a directed edge pointing from a to c and one from b to c. The result is the following graph.

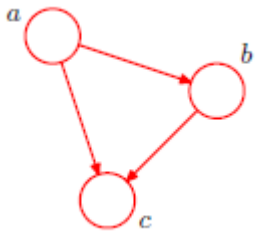

Figure 1: A directed graphical model representing the joint probability distribution over three variables a,b,c corresponding to right-hand side factorization of last of the variations in (14)

Whereas for the factor $P(a)$ there will be no edges direct towards node a. The result is the graph shown in 1. If there is a directed edge coming from node a to a node b, then a is called a *parent* of node b, and b is called the *child* of node a.

Each of those six representations gives a different graphical model. In the same way, any joint distribution can be factorized by using the product rule multiple times. The following is one variation of this. Let $p(X_1, \ldots, X_n)$ be the joint distribution of n random variables then

$$P(X_1, \ldots, X_n) = P(X_n|X_1, \ldots, X_{n-1}) \ldots P(X_2|X_1) \tag{15}$$

For any given choice of n, this factorization described above can be represented in a graphical model with n nodes, one for each conditional distribution on the right-hand side, with each node having incoming edges from all lower-numbered nodes. This graph belongs to the group of *fully connected graphs*, because there is an edge between every pair of nodes.

Up until now, the graphs shown so far represented ways of factorization corresponding to fully connected graphs. It will be shown in the next paragraphs that it is the *absence* of edges in the graph that conveys interesting information about the properties of the class of distributions that the graph represents.

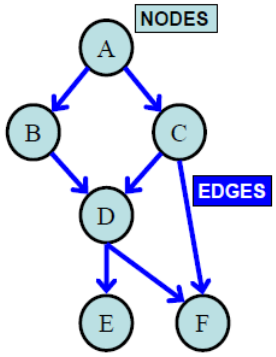

Figure 2: Graphical model corresponding to factorization (16)

The following factorization is represented in Figure 2.

$$P(A, B, C, D, E, F) = P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(D|B, C) \cdot P(E|D) \cdot P(F|C, D) \tag{16}$$

An assumption called the *Markov assumption* will be made in order to differentiate between direct dependency relations and indirect dependency relations. Such a nice factorization is necessary to get as close as possible to representing causal relationships.

The *local Markov assumption* in Bayesian Networks says: "Conditional on its parents' nodes, each node X_i is stochastically independent of its non-descendants." Recall that *stochastically independent* can be interpreted as adding the independent terms as given. Because they are independent knowing them or being conditional won't make a difference. As is explained here [7]. To define the Markov assumption more formally, consider the following.

Definition 2.5 (Markov assumption)**.** *Let $X_1, ..., X_n$ be the topological order of the n nodes. The **Markov assumption** states that for each $i \in \{1, \ldots .n\}$, and $pa(X_i) \subset \{X_1, ..., X_{i-1}\}$*

$$P(X_i|pa(X_i)) = P(X_i|X_1, ..., X_{i-1}), \tag{17}$$

*where $pa(X_i)$ is the set of parent nodes for X_i. The **local Markov assumption** is defined as*

$$P(X_1, ..., X_n) = \Pi_{i=1}^{n} P(X_i|pa(X_i)). \tag{18}$$

The local Markov assumption is now proved.

Proof. Without loss of generality it is assumed that $X_1, \ldots, X_n$ is the topological order of the n nodes.

$$\begin{aligned}
P(X_1, ..., X_n) &= P(X_n|X_{n-1}, \ldots, X_1) \cdot P(X_{n-1}, ..., X_1) \\
&= P(X_n|X_{n-1}, ..., X_1) \cdot P(X_{n-1}|X_{n-2}, ..., X_1) \cdot P(X_{n-2}, ..., X_1) \\
&= P(X_n|X_{n-1}, ..., X_1) \cdot P(X_{n-1}|X_{n-2}, ..., X_1) \cdot P(X_{n-2}|X_{n-3}, ..., X_1) \cdot P(X_{n-3}, ..., X_1) \\
&= \Pi_{i=3}^{n} P(X_i|X_1, ..., X_i - 1) \cdot P(X_1, X_2) \\
&= \Pi_{i=3}^{n} P(X_i|X_1, ..., X_i - 1) \cdot P(X_2|X_1) \cdot P(X_1) \\
&= \Pi_{i=3}^{n} P(X_i|X_1, ..., X_i - 1) \cdot P(X_2|pa(X_2)) \cdot P(X_1|pa(X_1)) \\
&= \Pi_{i=1}^{n} P(X_i|pa(X_i))
\end{aligned} \tag{19}$$

□

Here, the last equality is obtained by using equation (17). The directed graphs that are being considered are subject to an important restriction, there must be no *directed cycles*. In other words, there must be no closed paths within the graph such that there is a path from node to node along edges following the direction of the arrows and end up back at the starting node. Such graphs are called *directed acyclic graphs* or *DAGs*.

As shown before, the factorization of the joint distribution together with the conditional independence relations are often not unique. This gives rise to the notion of *equivalence classes* for the graphs.

Every specific factorization gives rise to a unique graphical model. Two different graphs are in the same *equivalence class* if their underlying mathematical structure (the factorization) is equivalent, meaning they all have the same joint distribution.

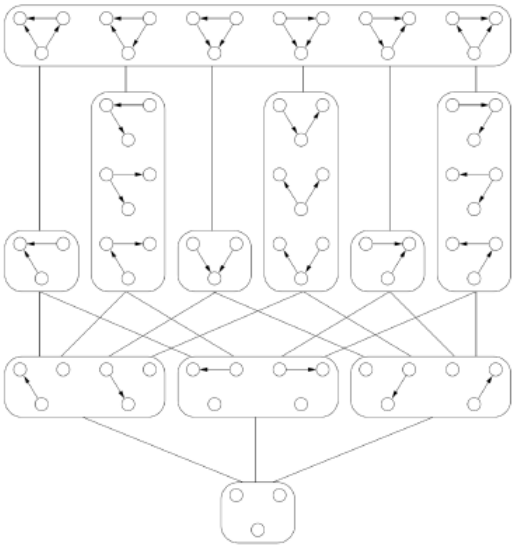

Figure 3: Equivalence classes of network of three nodes

An entire equivalence class of graphs can be represented by their so-called *CPDAG* representation. A CPDAG contains directed and undirected, biconditional edges.

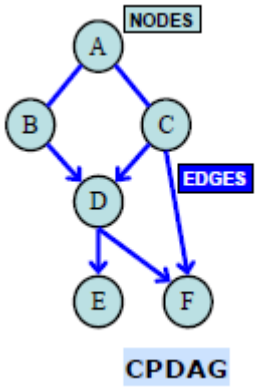

Figure 4: CPDAG of Figure 2

An algorithm to convert any DAG to their corresponding CPDAG will be later discussed.

To infer the conditional independence statements directly from a given DAG, there exists a theory called *d-separation*. D-separation will not be discussed in this paper, but if the

reader wants to know more about it they are invited to look at the equivalently named sections in Koski and Noble [2011] or in Bishop [2006]. The most important conditions related to d-separation are listed below:

1. Each node is only conditional on its parent set, not on other ancestors (Markov assumption);
2. If nodes don't have common ancestors then nodes are marginally stochastically independent;
3. If nodes have common descendant, when they are conditional on one or more of their mutual descendants, they are stochastically dependent.

3 Data

The data set that will be used in this research consists of measurements of Spotify of the variables listed before for 2017 different songs. This data set was found on the internet and can be downloaded through this link: https://www.kaggle.com/geomack/spotifyclassification. The data set included some more then the ten listed variables, but the others weren't continuous entries. Something that is necessary for this method.

Before the algorithms were performed on this data , the non continuous variables were excluded. The variables that were left are presented together with the explanations given by Spotify in the figure below.

Audio Features Object

KEY	VALUE TYPE	VALUE DESCRIPTION
acousticness	float	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	float	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
duration_ms	int	The duration of the track in milliseconds.
energy	float	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
instrumentalness	float	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
liveness	float	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	float	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
speechiness	float	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
tempo	float	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
valence	float	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Figure 5: All ten features and their explanations given by Spotify

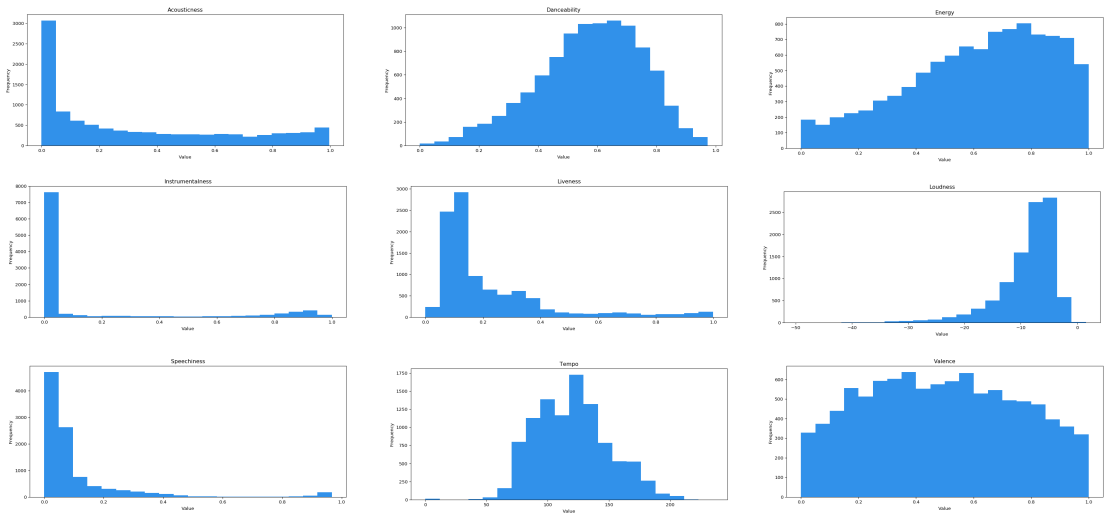

Figure 6: The distributions for nine of the variables, downloaded from Spotify. From row one to row three and from left to right: accousticness, dancebiity, energy, intruentalness, liveness, loudness, speechiness,tempo and valence

After excluding the non-continuous variables from the data, the data matrix also needed to be transposed. After transposing a the required matrix of n variables x m observations was obtained. In this case, there are $n = 10$ variables and $m = 2017$ observations, songs in this case together given a 10x2017 matrix.

In order for the design method to work best, it is necessary that the songs are randomly sampled. Unfortunately this is not the case, the songs are not completely random sampled. The data set is composed of approximately a 1000 songs liked and a 1000 songs disliked by someone, who gathered this data for a personal project. His project can be read here: https://opendatascience.com/a-machine-learning-deep-dive-into-my-spotify-data/. He selected the songs however as di verse as possible. He chose different genres and at most 5 songs from the same artist.

Later will be assumed that our variables are Gaussian Normal distributed. From the Spotify for developers website, figures of the distributions for all of the variables, except the one of duration could be downloaded. This is important information. Those figures can also be studied in more detail in the appendix.

4 Method

The underlying theory of the method is based upon [Grzegorczyk].

4.1 Marginal edge posterior probabilities: A strategy to build-up a graph that fits the data best

In this paper, the main objective is to find a Bayesian network that represents the data best. Something called the posterior distribution, denoted as $P(\text{graph}|\text{data})$, will be used for this. This distribution $P(\text{graph}|\text{data})$ describes the probability of a specific graph representing the actual network given the data. The last section has shown that the underlying mathematical structure of a graph is a specific factorization of the joint distribution of all variables included in the network. So $P(\text{graph}|\text{data})$ can be seen as the probability of some particular factorization of the joint probability function given that the data is known.

There exist multiple ways to achieve the main objective, i.e. to find the best graph that fits the data. Graph and Directed acyclic graph will be used interchangeably.

In this paper, the best graph that fits the data will be found by building up one graph through looking at which edges occur frequently in the most likely graphs that fit the data. This approach is chosen because if a large network (many nodes) has to be learnt from only a small number of observations, it does not usually make sense to search for a single "best" graph. After all, there is no single graph that has a 'significantly greater' posterior probability than the other graphs. In that case, there are usually multiple "good" graphs whose posterior probabilities are approximately equally high. The reason for that is when there are not enough observations there is not enough information to see the exact nuances, differentiate between different graphs, in the dependency relations, because the number of possible graphs grows super-exponentially in the number of nodes. In that case, it makes a lot more sense to look at features all those "good" graphs share.

This approach may raise a question: "Why not look directly at the probability of an edge occurring instead?" This will not be done, because the main reason for using Bayesian Networks is to look at an entire network and in doing so also differentiate between direct and indirect dependencies. Using the approach of solely looking at an edge instead of an entire network will undermine exactly what we are trying to accomplish.

Looking at the features all those "good" graphs share can be computationally expensive, so only the frequencies of some edges occurring will be looked at. It could, of course, be the case that some particular information will get lost at this point. Say, for instance, two edges are occurring only simultaneously or not at all in all "good" graphs, but it is probably better than looking only at the single best graph. However, our built up graph can theoretically have a lower posterior probability than one of the "good" graphs, but still this approach is preferred in this situation. The reason for this is that the result will, in the end, contain

more information. That is because our goal is to find only direct edges, so we are interested in the direct edges obtained from the multiple graph structures. Because we will be only looking at edge features, their topological similarities may be overseen.

There is a need to differentiate between two uses of the word graph: Our final graph (build-up from the weighted graphs of others) and all the possible graphs (so all possible networks from which the final graph will be build up). The most likely graphs,(latter definition) that fit the data is described by $P(\text{graph}|\text{data})$.

If the posterior probability would be calculated for each graph it would be computationally too expensive or even impossible in some cases, that's why there will be taken a sample of graphs of the posterior distribution. When the posterior probabilities are known, the sample and the graphs can be weighted by their posterior probabilities such that the so-called "good" graphs have a greater influence. First, two more actions need to be done to the found group before we can look at the edge frequencies.

First, as shown in the previous section, for each sampled graph there exist other graphs in their equivalence class which have the same posterior probability. So these other graphs are considered equally "good". That is why for each graph of the sample their CPDAG will be calculated. This will be done by using an algorithm , which takes as input an incidence matrix and gives as output the CPDAG incidence matrix, where each undirected edge is presented as a bi-directional . The matrix consist of zeroes and ones. The second action that needs to be done is the filtering of invalid DAGs. We would like to know what the similarities are between them and if they have something in common. More precisely, it is checked whether there are edges that every "good" graph has or edges that no graph has. This can be done by calculating the posterior probability that there is an edge averaged across all "good" graphs that are still directed acyclic graphs. This is called marginal edge posterior probabilities.

Calculation of the marginal edge posterior probabilities will be done in the following way. First, an indicator function will be used to indicate whether or not the CPDAG of a graph G possesses the edge in question. Now, the equation to calculate the marginal edge posterior probability is given as follows.

$$P(A \to B|D) := \sum_{G} P(G|D) \cdot I(G), \tag{20}$$

where $I(G)$ is the binary indicator function given by

$$I(G) = \begin{cases} 1, & \text{if CPDAG of G possesses the edge} \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

Here, $P(A \to B|D)$ is the probability, given the data, for a directed edge from node A to node B to be present in our best possible graph. Recall the posterior distribution is denoted

by $P(\text{graph}|\text{data})$, and $P(G|D)$ for short, which was briefly discussed before. The name stems from Bayesian Statistics, the field that will be used in this approach and that is closely looked into in Section 4.3. $P(\text{graph}|\text{data})$ can be seen as the likelihood of each particular network structure given the data. As we have seen before, each network structure together with its equivalence class is characterized by their unique (for the whole class) factorization of the joint probability function.

The following two figures serve to give the reader some more intuition into Equation 20 of the marginal edge posterior distribution. Let us say there are some graphs who follow the following posterior distribution.

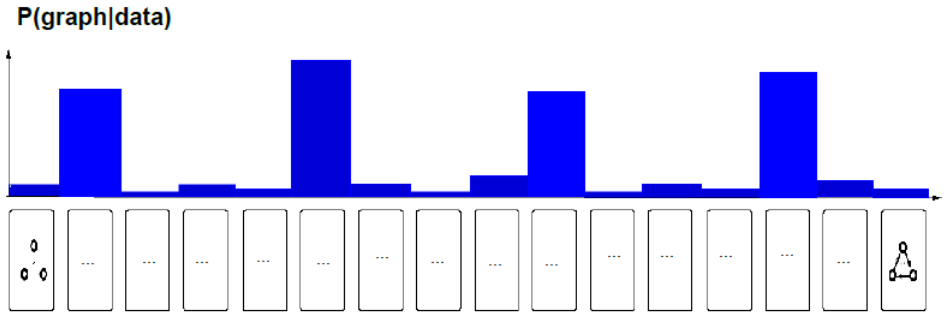

Figure 7: Visual representation of some graphs and their corresponding posterior probabilities $P(\text{graph}|\text{data})$

If $P(\text{graph}|\text{data})$ is then multiplied by the indicator function for a specific edge, the following figure is obtained.

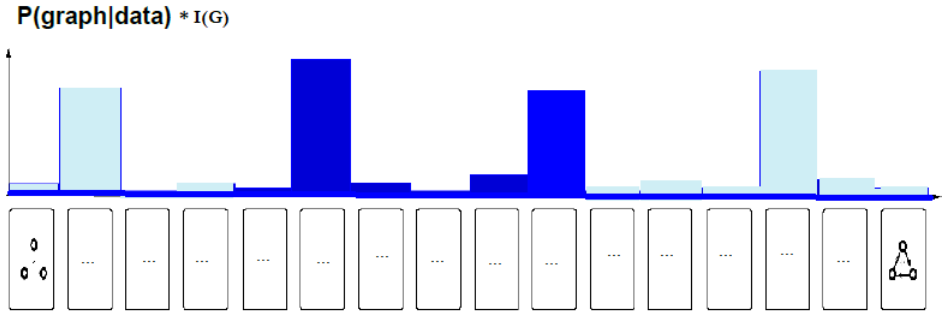

Figure 8: Visual intuition of the marginal edge posterior distribution

Remember that the following must always hold.

$$\sum_{G} P(G|D) = 1 \tag{22}$$

So the equation $P(A \to B|D) := \sum_G P(G|D) \cdot I(G)$ can in this setting be interpreted as the percentage of the total area in the figure has the specif edge under examination.

Another method is to simply look at which sole graph describes the data best by $P(\text{graph}|\text{data})$. So for each possible option. This method could be done by the so-called Greedy search algorithm. This paper will not go into that.

For both methods, just computing $P(\text{graph}|\text{data})$ for all possible graphs is computationally very costly and not sufficient. To calculate the above for each graph and for each edge that would be computationally for large networks really expensive and maybe even nearly impossible.

That is why a random sample of graphs will be taken from the posterior distribution $P(\text{graph}|\text{data})$. Then, the frequency of the edges occurring in the graph sample would say something about how likely they are. For comparison, if we would sample from the normal distribution we would get most realisations of the random variables near the mean.

In our case, if we sample from the posterior distribution, the realisations – in this case realisations of the graphs have a score – are proportional to the posterior near the mean. Therefore, if we can sample from the posterior distribution, we could give a pretty good estimation of the marginal edge posterior probabilities.

In our method this will be done by taking a sample form $P(\text{graph}|\text{data})$, and estimating the mean. This will be done using the *Metropolis-Hastings structure Markov Chain Monte Carlo algorithm*, which will be more thoroughly discussed in Section 4.2.

As mentioned before, the marginal edge posterior probability needs to be calculated. This will be done by estimation from a sample of $P(\text{graph}|\text{data})$, which is a discrete distribution. The next paragraph will elaborate on what sampling from $P(\text{graph}|\text{data})$ will look like. The posterior distribution will look something like Figure 7. So the realisations making up a sample from $P(\text{graph}|\text{data})$ will all be graphs.

For clarity it will now be compared to a Poisson distribution.

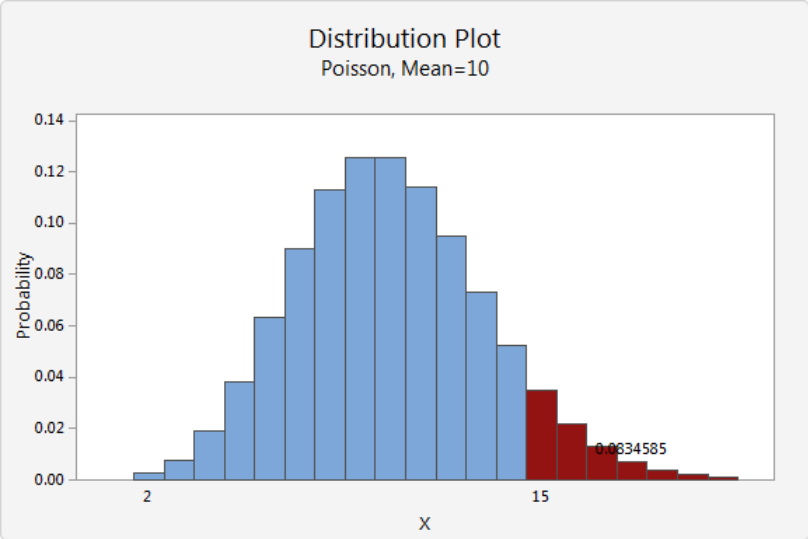

Figure 9: Poisson distribution as demonstrative example of sampling of a distribution

The mean of this Poisson distribution is 10. It is a discrete distribution, so it has discrete states. In this specific example, the realisations of X can take on numbers 1 to 20. A random sample of this Poisson distribution might be $\{10, 11, 9, 8, 3, 4, 2\}$ The case of taking a sample of $P(\text{graph}|\text{data})$ is rather similar. The realisations of the posterior distributions are specific graphs. It could help to think about them first enumerating all the possible graphs and then drawing a sample, just as in the Poisson case. The marginal edge posterior probability can be rewritten as:

$$\begin{aligned} P(A \to B|D) &:= \sum_{G} P(G|D) \cdot I(G) \\ &= P(I(G) = 1|D) \end{aligned} \tag{23}$$

By using the *Law of large numbers* it will be shown that we can estimate the unknown marginal edge posterior probability by the corresponding relative frequency in the graph sample. Rewritten in mathematical terms, it reads.

$$\hat{P}(A \to B|D) \to P(A \to B|D) \qquad \text{for } \; T \to \infty \tag{24}$$

Here $\hat{P}(A \to B|D) = \frac{1}{T}\sum_{i=1}^{T} I(G_i)$ and is called the *estimator*. It can be seen as the relative frequency of an edge in the sample. T is the sample size. The *Law of large numbers* is given as follows.

Theorem 4.1 (Law of large numbers)**.** *Let $X_1, X_2, \ldots$ be i.i.d random variables. Let $E[X]$ denote the expectation of a random variable X. Then the* ***Law of large numbers*** *states*

that:

$$\frac{\sum_{i=1}^{T} x_i}{n} \to E[X] \qquad for \;\; T \to \infty, \tag{25}$$

where $E[X] = \sum_x x \cdot P(X = x)$.

Let in our case X be the binary indicator function $I(G)$. Then, by law of large numbers it holds that

$$\frac{\sum_{i=1}^{T} I(G_i)}{T} \to E_{\text{data}}[I(G)], \tag{26}$$

where $I(G_i)$ is a realisation of $I(G)$ indicating with either 0 or 1 that G_i possesses the specific edge. Now the only step left to be shown is $E_{\text{data}}[I(G)] = P(I(G) = 1|D)$

$$\begin{aligned} E_{\text{data}}[I(G)] &= 0 \cdot P_{\text{data}}(I(G) = 0) + 1 \cdot P_{\text{data}}(I(G) = 1) \\ &= P(I(G) = 1|D) \end{aligned} \tag{27}$$

If we can generate a sample from our posterior distribution then, as mentioned earlier, two actions still need to be executed in order to First, check if the graphs satisfy the acyclicity constraint and secondly , compute their corresponding CPDAG.

After those algorithms are performed, this procedure results in a bunch of incidence matrices. From those it can be checked for each edge if it is there in the following way. For the edge from node X_i to X_j look at incidence matrix I for entry $I_{i,j}$ if it is 1, it is clear that the graph possesses that edge. Then $\frac{\sum_{i=1}^{T} I(G_i)}{T}$ can be calculated as estimation for $P(X_i \to X_j|D)$.

All of this will be stored in a matrix, now indicated by A,here $A_{i,j} = P(X_i \to X_j|D)$. And after that is all learnt, a cut-off is searched for and a graph will be presented. The next section will elaborate on how exactly a sample will be taken from the posterior distribution.

4.2 Metropolis-Hastings Structure MCMC sampler: Sampling to efficiently compute all possible graphs of P(graph|data)

The Metropolis-Hastings Structure Markov Chains Monte Carlo simulation uses, as the name suggests, insights from *Markov Chains*. Markov chains give us the chance to determine the trajectory of discrete states. What is meant here is that in this paper this means a path or walk through discrete states of our graphs. The State space is composed of all possible graphs. In Markov Chains, it is not required to know the whole distribution. It is only required to know the conditional probabilities, in other words only the probability of the system going to one particular state given that the system is already in one of the states.

Let us look at an example. Say there is a system with discrete state-space $S = \{A, B, C\}$ meaning some system can be in three different states, namely state A, B or C. Assume the system started in a randomly chosen state A. It will soon be explained that this choice doesn't influence the results if the system moves from state to state long enough. The probabilities to go from one state to an other are known and presented in the so-called transition matrix P.

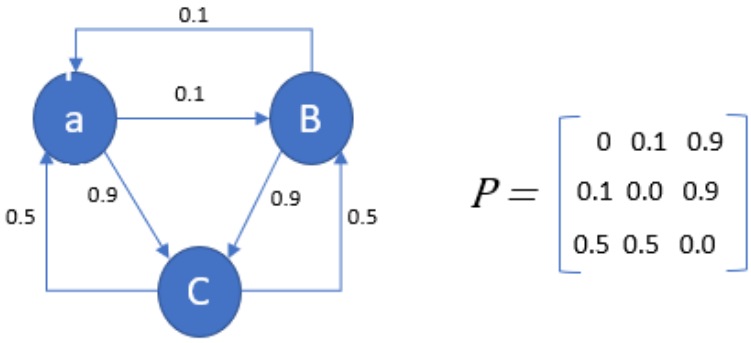

Figure 10: Example of Markov chain

On the left of Figure 10 the state diagram is shown and on the right the transition matrix P. The transition matrix P can be interpreted in the following way: $P_{i,j} = P(X_j|X_i)$ for $i, j \in A, B, C$ and X_i, X_j being the nodes corresponding to the states A, B, C respectively. So $P_{A,B} = P(B|A) = 0.1$ describes the probability of moving to state B given that the system is in system A.
So if the beginning state is known together with the transition matrix P, an trajectory of the systems states can be sampled.

Let us assume the system is now in state A. Then a possible trajectory sampled from the transition matrix could be: $A \to C \to B \to C$. In this paper, the state space will be the space containing all possible graphs. So a sample of a trajectory can then, for example, be the following: $G_6 \to G_3 \to G_2 \to G_5 \to G_7$, where G_i is a graph and i can be ranging from 0 to the number of all possible graphs.

If one would like to calculate the probability that a system X is in state i at step t, $P(X_t = i)$, this can be done in the following way.

Let $t = 1, 2, 3, \ldots$ be the number of steps or iterations. Let X be a system with state space $S = \{1, \ldots, k\}$, with k a Let the k-dimensional row vector composed of $P(X_t = i)$ for each i in the state space be defined in the following way:

$$p_t := (p_{t,1}, \ldots p_{t,k}) = (P(X_t = 1), \ldots, P(X_t = k)) \tag{28}$$

First, an initial distribution is required, which gives the probabilities of the systems being initially in either one of all the states of the state space. This is presented as a row vector, denoted her as p_1. In our example from before this could be

$$p_1 = (P(X_1 = A), \ldots, P(X_1 = C) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}),$$

meaning for each state it is equally likely that the system is initially in that state. Then the probability that a system X is in state i at step t, can be read off by calculating p_t for each $i \in S$, using the equation below.

$$p_t = p_1 \cdot T^{t-1} \tag{29}$$

The nice thing about Markov chains for this situation, is that there exists a so-called *stationary distribution* of a discrete Markov chain with transition matrix T. $J = (p(1), ..., p(k))$ is the stationary distribution of T if $\pi \cdot T = \pi$. The stationary distribution of a Markov chain does not depend on initial distribution. Because $\pi = p_1 \cdot T^t$ for $t \to \infty$ there is a correlation between π and T. Unique for both. If $\pi = (p(i), ..., p(k))$ then the equation of detailed balance is given by

$$\frac{T_{i,j}}{T_{j,i}} = \frac{p(j)}{p(i)}. \tag{30}$$

This concept of stationary distribution will be used to sample graphs from the P(graph|data) distribution. A specific transition matrix is sought that leads to the stationary distribution of the posterior distribution P(graph|data). Then the initial graph that is chosen has no influence if we iterate long enough. So we look for a transition matrix corresponding to P(graph|data) as a stationary distribution. This is be done by looking at the equation of detailed balance. We want $\pi = (P(G_1|D), ..., P(G_k|D))$.

$$\frac{T_{i,j}}{T_{j,i}} = \frac{T(G_i, G_j)}{T(G_j, G_i)} = \frac{P(G_j|P)}{P(G_i|D)}, \tag{31}$$

for all $G_i, G_j \in S$ where $(i \neq j)$ with $P(G_i|D) > 0$ and $T_{j,i} > 0$. It is common for the Metropolis-Hastings method to take $T(i, J)$ as the product of $Q(i, j)$ and $A(i, j)$ for $j \neq i$ and for $j = i$

Q is usually chosen and is the proposed probability. A is the acceptance probability.

One of them can be chosen and the other is determined by the equation of detailed balance because it needs to be satisfied to get π to be P(graph|data). Usually, Q is chosen. So A is fixed as we set Q. The equation of detailed balance is satisfied if

$$A(G_i, G_j) = \min \left\{ 1, \frac{P(G_i|D)}{P(G_j|D)} \cdot \frac{Q(G_i, G_j)}{Q(G_i, G_j)} \right\} \tag{32}$$

We choose Q such that we randomly select one of the neighboring graphs. That is because the ratio $\frac{P(G_i|D)}{P(G_j|D)}$ is easier to determine for neighboring graphs. So

$$Q(G_i, G_j) = \begin{cases} \frac{1}{|N(G)|}, & G^* \in N(G) \\ 0, & G^* \notin N(G) \end{cases} \tag{33}$$

Here, $N(G)$ is set of neighbor graphs of G. So if $G^* \in A(G_i, G_j) = \min\left\{1, \frac{P(G_i|D)}{P(G_j|D)} \cdot \frac{Q(G_i,G_j)}{Q(G_i,G_j)}\right\}$, it holds that

$$\begin{aligned} A(G_i, G_j) &= \min\left\{1, \frac{P(G_i|D)}{P(G_j|D)} \cdot \frac{Q(G_i, G_j)}{Q(G_i, G_j)}\right\} \\ &= \min\left\{1, \frac{P(D|G_j) \cdot P(G_j)}{P(D|G_i) \cdot P(G_i)} \cdot \frac{|N(G_i)|}{|N(G_j)|}\right\} \end{aligned} \tag{34}$$

where $\frac{P(D|G_j)}{P(D|G_i)}$ is called the likelihood ratio, $\frac{P(G_j)}{P(G_i)}$ is called the prior ratio and $\frac{|N(G_i)|}{|N(G_j)|}$ is called the Hastings-ratio. Needs to be calculated. The names stem from Bayesian Statistics and we will dive into them more closely in Section 4.3. Per construction the transition matrix $A \cdot Q$ is guaranteed to converge to $P(\text{graph}|\text{data})$. So if we take trajectory $G_1 \to G_5 \to G_7 \to G_1 00$, after it is iterated long enough, it will be sampled from that distribution. This procedure is called the *structure MCMC sampling scheme*.

The graph sample can be used to estimate marginal edge posterior probabilities.

1. Observation 1
   Burn-in and sampling phase
2. Observation 2
   Thinning because of auto correlation.
3. Observation 3.
   Method
4. Observation 4.
   Number of observations

efficient computation Hastings-ratio number of neighbouring → still DAG and single edge operations incidence matrix. ALgorithms single edge CITE SINGLE EDGE ALGORITHMS We will now look more closely at the acceptance probabilities A(G....) and dive into some Bayesian Statistics. Bayesian and stat other two names come from.

4.3 Bayesian Statistics

Another field of Statistics used in this research is *Bayesian Statistics.* This area is largely based on *Bayes' rule* or sometimes called *Bayes' theorem.*

Theorem 4.2 (Bayes' theorem)**.** *Let A and B are events. The* ***conditional probability*** *of A given B is given by*

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{35}$$

with $P(B) \neq 0$. This expresses the likelihood of event A given that event B is true.

deduction likelihood and prior, prior becomes uniform if edge deletion or addition and parent node set s.414

$$\begin{aligned} A(G_i, G_j) &= \min\left\{1, \frac{P(G_i|D)}{P(G_j|D)} \cdot \frac{Q(G_i, G_j)}{Q(G_i, G_j)}\right\} \\ &= \min\left\{1, \frac{P(D|G_j) \cdot P(G_j)}{P(D|G_i) \cdot P(G_i)} \cdot \frac{|N(G_i)|}{|N(G_j)|}\right\} \end{aligned} \tag{36}$$

Posterior distribution ultimate goal Bayesian statistics Bayesian statistics is largely based on Bayes rule which is made by using two times the conditional probability rule.

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{37}$$

$$P(X|Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(Y|X) \cdot P(X)}{P(Y)} = \frac{P(Y|X) \cdot P(X)}{\sum_X P(Y,X)} = \frac{P(Y|X) \cdot P(X)}{\sum_X P(Y|X) \cdot P(X)} \tag{38}$$

$$P(\text{graph}|\text{data}) = \frac{P(data, graph)}{P(data)} = \frac{P(data|graph) \cdot P(graph)}{P(data)} \tag{39}$$

This equation will be used a lot since it is central to our inference, so a name will be introduced for every factor. These names are commonly used in the paradigm Bayesian statistics.
P(graph|data) is the posterior probability. It is the posterior (*aposteriori*) probability of the graph (in sense of model) given by the observed data P(data|graph) is the marginal likelihood. (score) "Assuming that the true (in-)dependencies among the variables are inferrable from the data, D, high marginal likelihoods can only be reached by those graphs that imply or approximate these true relationships" (cite Marco)

P(graph) is the prior probability. (apriori)prior; Which values is theta or graph in our case likely to take on. In our case will be uniform. Need to be uninformative –> Jeffrey's prior

P(data) is the probability given the data and is called the *normalization constant*. It is called the normalization constant because in conditional probability there is a shift from the whole probability space to the smaller space of Y. Here, the thing it is conditional on is now the whole probability space, so it is equal to one. Therefore, there is a need to normalize by a factor. That factor is the normalization constant.

So our normalization constant P(data), or $P(D)$, is actually defined as follows:

$$P(D) = \sum_{G^\star} P(D|G^\star)P(G^\star), \tag{40}$$

where we sum over all valid directed graphs G. This definition, as was mentioned before, ensures that $\sum_{\text{graphs}}$P(graph|data)= 1 holds.

The normalization constant P(data) does not depend on the choice of a graph, so we have that the posterior probability P(graph|data) is proportional to the score times the prior . The MH MCMC is based on the denominator remaining fixed.

$$P(\text{graph}|\text{data}) = \frac{P(\text{data}|\text{graph}) \cdot P(\text{graph})}{P(\text{data})} \propto P(\text{data}|\text{graph}) \cdot P(\text{graph}) \tag{41}$$

The marginal likelihood in sense of the model is given by

$$P(\text{data}|\text{graph}) = \int P(\text{data,q}|\text{graph})dq \tag{42}$$

Because q is supposed to be continuous maybe? So integral over joint probability. What is q? q is a vector of still unknown parameters that are required. What can we think of my model? model 1. that it is random sample iid 2. give the distribution of data. The marginal likelihood can be rewritten in the following way.

$$\begin{aligned} P(D|G) &= \int P(D, q|G)dq \\ &= \int \frac{P(D, q, G)}{P(G)}dq \\ &= \int \frac{P(D, q, G)}{P(G)} \cdot \frac{P(q|G)}{P(q|G)}dq \\ &= \int \frac{P(D, q, G)}{P(q, G)} \cdot P(q|G) \\ &= \int P(D|q, G) \cdot P(q|G)dq \end{aligned} \tag{43}$$

We will now be focussing on how we will find the probability distribution of a graph. to calculate the score. the joint distribution will be separated and each random variable maybe gaussian normal, will be explained in BGe section

We will now look into something called CPDAG representation. The end must perform to calculate the accurate relative edge frequency. Because all scores are the same for graphs in the same equivalence class, they all represent the found score, which places all of them in the highest score graph pool.

To obtain an algorithm which takes as input a DAG incidence matrix and converts it into a CPDAG representation incidence matrix, we need to take a look at two sub algorithms. The first one aims to assess the so-called topological order of the nodes of a DAG. The second one aims to asses the order of the edges. It will use the topological ordering of the node to do this. Finally, the algorithm that converts a DAG into a CPDAG can be constructed.

A sketch of the algorithm is given in the following informal way.

- Step 1: Give skeleton
- Step 2: Give skeleton with directed v structures, called G_v
- Step 3: Give graph G_{v+} (give edges direction back which reversal would create new v structures)
- Step 4: Add direction to edges whose reversal gives invalid cycles

We will now look into the marginal likelihood of the graph in the sense of the model. The goal here is to learn whether there is a possibility to make the graph simpler and closed form. That is why we assumed and looked into the local Markov assumption.

In the last section, the following was concluded.

$$P(D|G) = \int P(D|q, G) \cdot P(q|G) dq \tag{44}$$

Now, we would like write $P(D|q, G)$ with our knowledge of the Markov assumption. If the realizations of the dataset are **independent**, the following holds

$$P(D|q, G) = \Pi_{i=1}^{n} \Pi_{j=1}^{m} P(X_i = D_{i,j} | pa(X_i) = D_{pa(X_i),j}, q_i \tag{45}$$

If we know the graph, then we also know part of the structure of the joint probability funtion of the variables. This fact follows from the factorization or Markov assumption (18). Since the observations are independent, we can write:

$$\begin{aligned} P(D|q, G) &= \Pi_{j=1}^{m} P((X_1, ..., X_n) = (D_{1,j}, ..., D_{n,j}) | G, q) \\ &= \Pi_{j=1}^{m} P((X_1 = D_{1,j}, ..., X_n = D_{n,j}) | G, q) \end{aligned} \tag{46}$$

Now we will use the fact that if we know the graph, we also know from (18) that we can simplify the joint density distribution in the following way. If we simply take one observation from the realizations of the data set, so some column j of our data matrix, we can use the Markov assumption again to see that

$$P(X_1, ..., X_n) = \Pi_{i=1}^{n} P(X_i | pa(X_i)). \tag{47}$$

$$\begin{aligned} P((X_1 = D_{1,j}, ..., X_n = D_{n,j})|G, q) &= \Pi_{i=1}^{n} P(X_i = D_{i,j} | pa(X_i) = D_{pa(X_i),j}, q) \\ &= \Pi_{i=1}^{n} P(X_i = D_{i,j} | pa(X_i) = D_{pa(X_i),j}, q_i) \end{aligned} \tag{48}$$

$$P(D|q, G) = \Pi_{i=1}^{n} \Pi_{j=1}^{m} P(X_i = D_{i,j} | pa(X_i) = D_{pa(X_i),j}, q_i \tag{49}$$

4.4 BGe scoring metric: How to calculate the last terms

In the former section, we showed a more explicit method to calculate the marginal likelihood. What we are missing, however, is our choices for the parameters and hence also for distribution of our variables. These parameters are denoted as the vector q.

This vector will be looked into more closely now, and we will assume for all parameters in it both independence and modularity. The definition of these assumptions is given below.

Assumption 1 – Parameter independence.

Definition 4.1 (Parameter independence)**.** ***Parameters*** *are called* ***independent*** *if the distributions of the separate parameters (for the local probability distributions) are stochastically independent.*

Assumption 2 – Parameter modularity

Definition 4.2 (Parameter modularity)**.** ***Parameter modularity*** *means the density of the parameters (of each local distrubition) depends on the local structure (i.e. the parents of X_i only. (18) The following holds.*

$$P(q_i|G) = P(q_i|X_i, pa(X_i)) \quad \textit{for } i = 1, \ldots, n \tag{50}$$

Assumptions 1 & 2 entail the following.

$$P(q|G) = \Pi_{i=1}^{n} P(q_i|pa(X_i)) \tag{51}$$

$$
\begin{aligned}
& \downarrow \boxed{\text{(Bayes Theorem) } \textbf{Proof ??}} \\
P(graph|data) &= \frac{P(data, graph)}{P(data)} \\
& \downarrow \boxed{P(D,G)=P(D|G)\cdot P(G) \text{ (conditional) } \textbf{Proof ??}} \\
&= \frac{P(data|graph) \cdot P(graph)}{P(data)} \\
& \downarrow \boxed{P(D)\text{is constant (Normalisation constant) } \textbf{Proof ??}} \\
&\propto P(data|graph) \cdot P(graph) \\
&= \int P(D|q, G) \cdot P(q, G)dq \cdot P(G) \\
&= \int \Pi\Pi\Pi P(X_i = D_{i,j}|pa(X_i) = D_{pa(X_i),j,q_i}) \cdot P(q|G)dq \cdot P(G) \\
&= \int \Pi\Pi\Pi P(X_i = D_{i,j}|pa(X_i) = D_{pa(X_i),j,q_i}) \cdot \Pi_{i=1}^{n} P(q_i|pa(X_i))dq \cdot P(G)
\end{aligned}
\tag{52}
$$

$$
\begin{aligned}
&P(\text{graph}|\text{data}) = \\
&= \int \Pi\ \Pi P(X_i = D_{i,j}|pa(X_i) = D_{pa(X_i),j,q_i}) \cdot \Pi_{i=1}^{n} P(q_i|pa(X_i))dq \cdot P(G) \\
&= \int^{n \text{ times}} \cdots \int (P(q_i|pa(X_i)) \cdot \Pi_{j=1}^{m} p(x_i = D_{i,j}|pa(X_1) = D_{pa(X_i),j,qi}))d_{q_1}, \ldots d_{q_n} \cdot P(G) \\
&= \Pi_{i=1}^{n} \int (P(q_i|pa(X_i)) \cdot \Pi_{j=1}^{m} P(X_i = D_{i,j}|pa(X_i) = D_{pa(X_i),i,q_i})dq_i \cdot p(G) \\
&= \Pi_{i=1}^{n} \Psi_i(pa(X_i), D^{X_i}, D^{pa(X_i)}) \cdot P(G).
\end{aligned}
\tag{53}
$$

The idea is now to go into Hastings ratio and simplify. First let us consider the Likelihood ratio.

$$
\frac{P(D|G_j)}{P(D|G_i)} = \frac{\Pi_{k=1}^{n} \Psi_k(pa(X_k|G_j), DX_k, D^{pa(X_k|G_j)})}{\Pi_{k=1}^{n} \Psi_k(pa(X_k|G_i), DX_k, D^{pa(X_k|G_i)})} \tag{54}
$$

Here, G_i and G_j are two neighbouring graphs. This implies that to go from G_i to G_j, a single edge deletion, a single edge addition or an edge reversal has been performed. Call the node where the altered edge was pointing to X_u, then the two neighboring graphs G_i and G_j can only differ with respect to the parent node set of X_u. From this the expression below follows for an edge deletion or addition but not both.

$$
\frac{P(D|G_j)}{P(D|G_i)} = \frac{\Psi_u(pa(X_u|G_j), DX_u, D^{pa(X_u|G_j)})}{\Psi_u(pa(X_u|G_i), DX_u, D^{pa(X_u|G_i)})} \tag{55}
$$

If an edge reversal was the change between G_i to G_j, more precisely the reverse of an edge from node X_w to node X_u, then the two neighboring graphs G_i and G_j can only differ with respect to the parent sets of the two nodes X_u and X_w. From this the expression below follows for an edge reversal.

$$
\frac{P(D|G_j)}{P(D|G_i)} = \frac{\Psi_u(pa(X_u|G_j), DX_u, D^{pa(X_u|G_j)})}{\Psi_u(pa(X_w|G_i), DX_w, D^{pa(X_w|G_i)})} \cdot \frac{\Psi_w(pa(X_w|G_j), DX_w, D^{pa(X_w|G_j)})}{\Psi_w(pa(X_w|G_i), DX_w, D^{pa(X_w|G_i)})} \tag{56}
$$

Model assumption 1: The n domain variables $X_1,\ldots,X_n$ are **multivariate Gaussian** $N(\mu,\Sigma)$ **distributed** with unknown expectation vector μ and unknown covariance matrix Σ (precision matrix $W = \Sigma^{-1}$ respectively)

Model assumption 2:

The unknown parameters are normal-Wishart distributed.

That is, the expectation vector μ is Gaussian $N(\mu_0, \nu^{-1}\cdot\Sigma) = N(\mu_0, (\nu W)^{-1})$

distributed with expectation vector μ_0

and covariance matrix $\nu^{-1}\cdot\Sigma = (\nu W)^{-1}$

The matrix W is $W(\alpha, T_0)$ Wishart-distributed with $\alpha > n+1$ degrees of freedom and precision matrix T_0.

The hyperparameters α, ν, μ_0 and T_0 must be set fixed.

Figure 11: [1]

Let's have a closer look at those factors. As seen before this below holds.

$$\begin{aligned} &\Psi_i(pa(X_i), D^{X_i}, D^{pa(X_i)}) \cdot P(G) = \\ &\int (P(q_i|pa(X_i)) \cdot \Pi_{j=1}^{m} P(X_i = D_{i,j}|pa(X_i) = D_{pa(X_i),i,q_i}) dq_i \cdot p(G) \end{aligned} \tag{57}$$

$$\begin{aligned} P(\mu, W|X_i) &= \frac{P(X_1,\ldots,X_n|\mu,W)\cdot P(\mu,W)}{P(X_i)} \\ &= \frac{P(X_1,\ldots,X_n|\mu,W)\cdot P(\mu|W)\cdot P(W)}{P(X_i)} \end{aligned} \tag{58}$$

where $P(X_1,\ldots,X_n|\mu,W)$ is multivariate Gaussian distributed. $P(\mu,W)$ is normal Wishart distributed. And $P(\mu|W)$ is multivariate normal distributed $N(\mu_0,\nu W)$ such that $\nu > 0$, $P(W)$ is Wishart distributed. $\alpha > n-1$ degrees of freedom. Precision matrix T, denoted by $W(\alpha, T_0)$.
μ expectation vector is Gaussian $N(\mu_0, \nu^{-1}\cdot\Sigma = N(\mu_0,(\nu W)^{-1}) = P(\mu|W)$ and $P(W) = W(\alpha, T_0)$ with $\alpha > 0$ and precision matrix T_0. The hyperparameters (called hyper, because parameters are μ and W in this case) must be set fixed α, ν, μ_0 and T_0

[1]

Likelikhood	Model parameters	Conjugate prior distribution	Prior hyperparameters
Multivariate normal	μ (mean vector) W (precision matrix)	Normal Wishart	μ_0, k_0, v_0, V (μ_0, v, α, T_0)
Multivariate normal with known μ	W	Wishart	v, V (a, T_0)

Table 1:

4.5 Convergence Diagnostics

In order to check if the algorithm was iterated long enough such that the Markov Chain is iterated long enough such that it converged to the stationary distribution, such that we indeed have a sampling of the posterior distribution there is a need for convergence diagnostics. Both the Trace plot and Scatter plots, which will be explained shortly, provide a necessary but not sufficient condition for convergence.

4.5.1 Trace plot

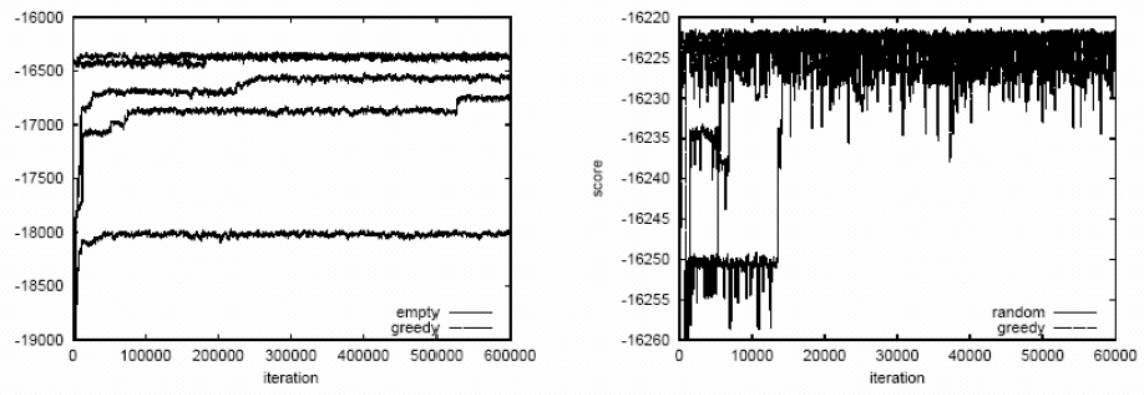

Figure 12: Examples of Trace plots [2]

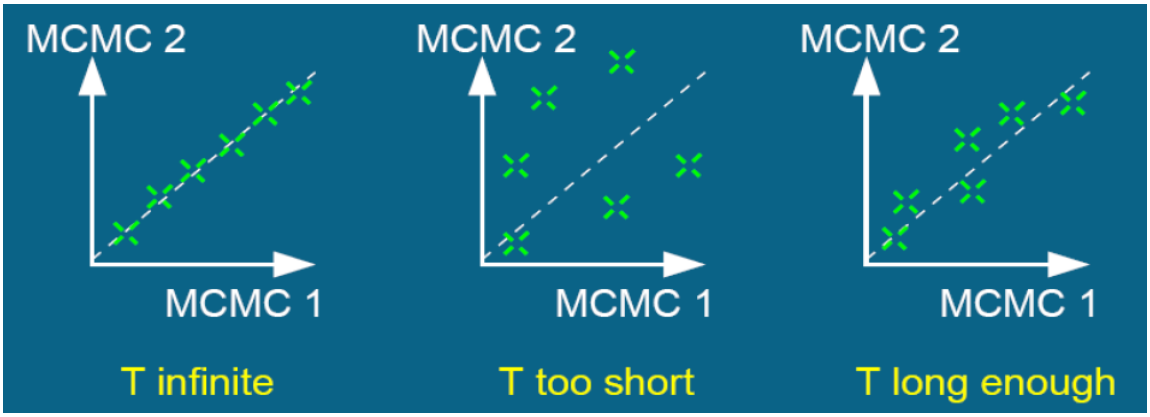

Figure 13: Examples of Scatter plots [3]

4.5.2 Scatter plot

As mentioned before, for each data set, there will be done three independent runs of the structure MCMC algorithm in order to check for convergence. Three scatter plots will be made to examine the differences in edge scores of the three runs.

A scatter plot, possesses the ability to compare two different edge score result of the two runs at the same time. This means there will be made three scatter plots in total, one for run 1 versus run 2, one for run 1 versus run 3 and one for run 2 versus run 3.

After the structure MCMC algorithm is performed, the edge scores will be presented in a matrix. Before the scatter plots can be madem the entries of the matrices will be stored in a vector except the entries on the diagonal of the matrix. The entries on the diagonal correspond to probabilities of an edge from a node to itself and by the acyclicity contraint those are excluded. Now those two vectores wil be combined to create n 2-tuples with the x- coordinate being the edge score for one edge from one of the runs and the y-coordinate for the other run. The 2-tuples can be interpreted as x- and y-coordinates and can be plotted in a graph. The line $y = x$ will also be plotted as reference.For all of them will be checked whether or not the points lie nicely near the diagonal, indicating that their edge scores do not differ so much with respect to one another.

[2] Grzegorczyk, M., *Statistical Genomics*. University of Groningen, 2019.
[3] Grzegorczyk, M., *Statistical Genomics*. University of Groningen, 2019.

4.6 Work in R

As mentioned before, in this research there will different numbers of songs, so columns sampled from the original data consisting of 2017 unique songs. This will be done for different sizes of samples from the original data for sizes of 10, 100 and 500 songs.

Each of these three sample sizes will be examined 10 times. Hence, we end up with 10 unique matrices of size $10x10$, 10 unique matrices of size $10x100$ and 10 unique matrices of size $10x500$. Each of them are made by first drawing a random vector of size $10, 100$ and 500 from numbers 1 to 2017 and then merging the corresponding columns together in one matrix.

For each unique matrix of n-variables by m-observations, three independent runs of the structure MCMC algorithm were done. As described in the convergence diagnostics section, this is done to check for convergence. Have we iterated long enough such that our sample indeed can be seen as drawn from the posterior distribution?

Necessary but not sufficient conditions for convergence are that the three independent runs of the structure MCMC algorithm don't differ that much from each other. They should, if iterated long enough, end up with approximately the same results. If this is indeed the case can be checked by looking at if the same plateaus are reached in the Trace plot and the points being plotted near the diagonal line. The Trace plots will then be checked. After that three scatter plots will be made, one for run 1 versus run 2, one for run 1 versus run 3 and one for run 2 versus run 3. For all of them will be checked whether or not the points lie nicely near the diagonal, indicating that their edge scores do not differ so much with respect to one another.

For every run of the structure MCMC, two variables need to be set: the thinning constant and the number of iterations. The thinning constant indicates how many out of a sequence of graphs will be stored. A thinning constant of 100 indicates that out of a sequence of 100 graphs one will be stored, so one each 100. This need to be done because as mentioned before graphs succeeding one another will be auto-correlated because to move from one graph to another only graphs are allowed that differ only with respect to one single edge. That's why there is a need to filter out some graphs and only save some of them. The number of iterations needs to be large enough so that convergence can be seen by means of the convergence diagnostics. This will be done by trial and error. For every unique data set of the same size, the same number of iterations will be done.

The results, the edge scores, for all ten different data samples for each size, will be compared. This will be done by using again scatterplots. One scatter plot can only compare two results at the same time, meaning two compare ten result, 10 choose $2 = 45$ scatterplots need to be made for each category of sample size $10, 100$ and 500. For this the averages of the three independent runs of the structure MCMC algorithms will be calculated. In order to come up with solely one result for each unique data sample. Taking the average is chosen

because it is not possible anymore that those three runs differ much, because the number of iterations was already increased such that the scatterplots revealt that.
After that the average will also be taken of the edge scores of those 10 different samples of the same size. Such that we only end up with three different vectors of edge scores for $10, 100$ and 500 respectively.
The scripts that were written for these purposes, can be found in the appendices A.3 and A.2. The other scripts used in this paper can also be found there.

5 Results

This section presents the results of the R scripts as given in the appendix A.3. The form of presentation is by presenting scatter plots for different data samples of 10, 100 and 500 songs.

5.1 Number of iterations needed

The number of iterations needed to obtain a 'reasonably good' plot differs for the different sample sizes. Judging from the plots, the following values are decided on, where m denotes the size of the dat set.

m	# iterations
10	80.000
100	50.000
500	100.000

Table 2: Number of iterations needed to get a 'reasonably good' scatter plot

5.2 Example of sufficient Trace plot & scatter plot

Visually, the 'sufficiently well' plot looks like the following.

5.3 The 45 scatter plots between two out of the ten edge scores of the data samples of 10 songs

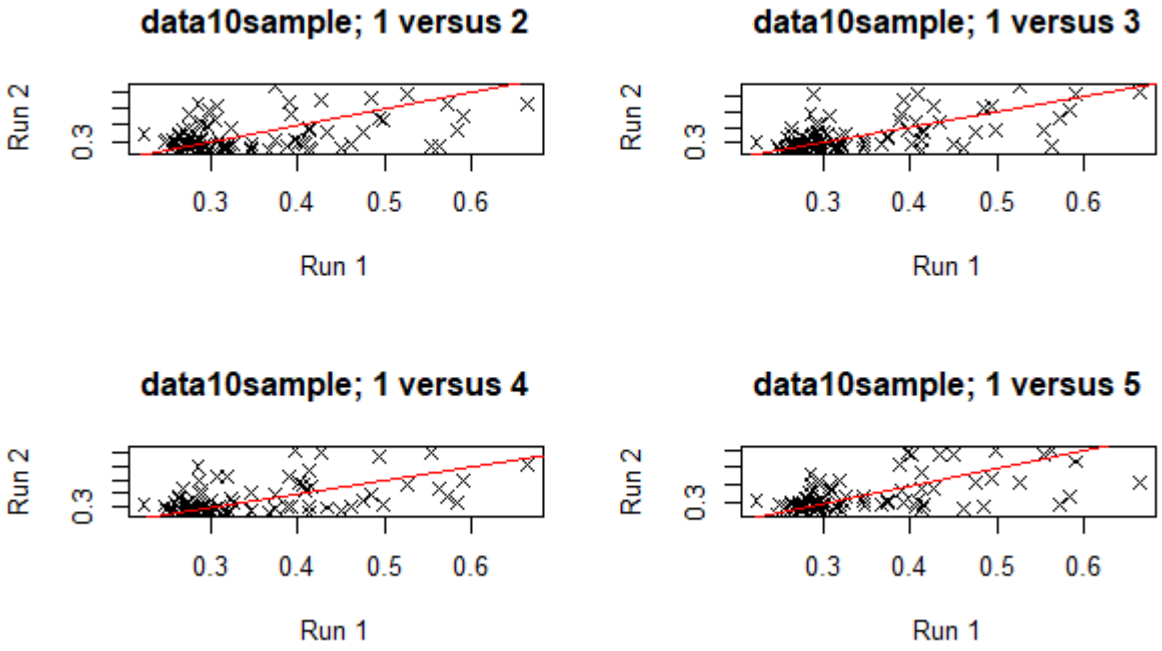

Figure 14: Caption

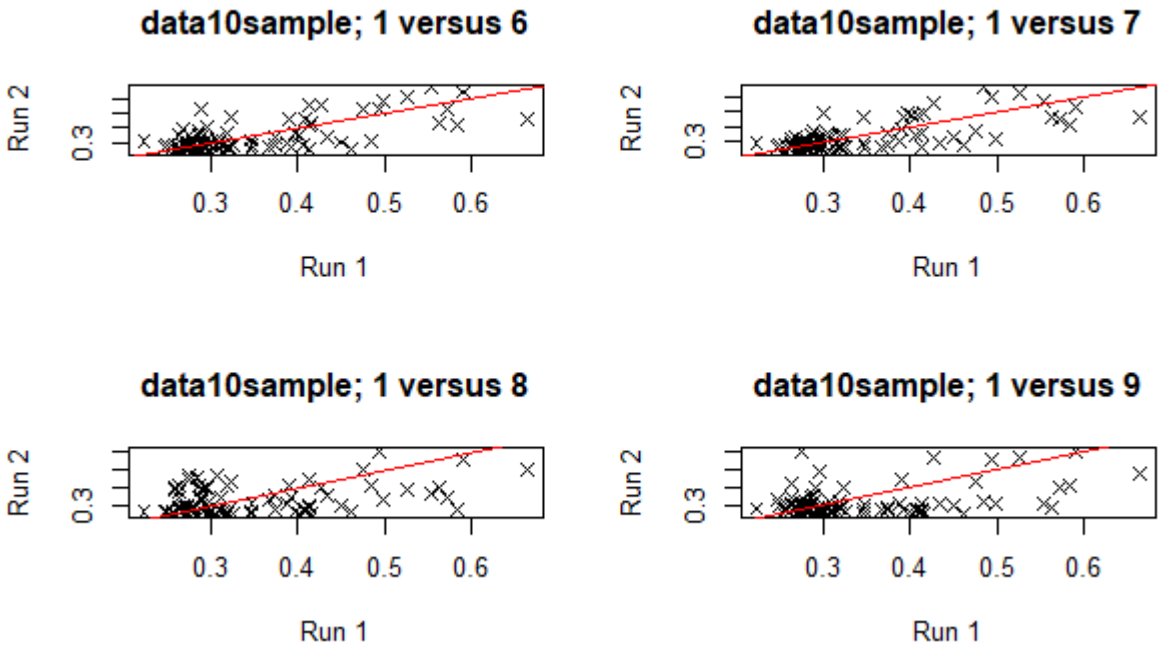

Figure 15: Caption

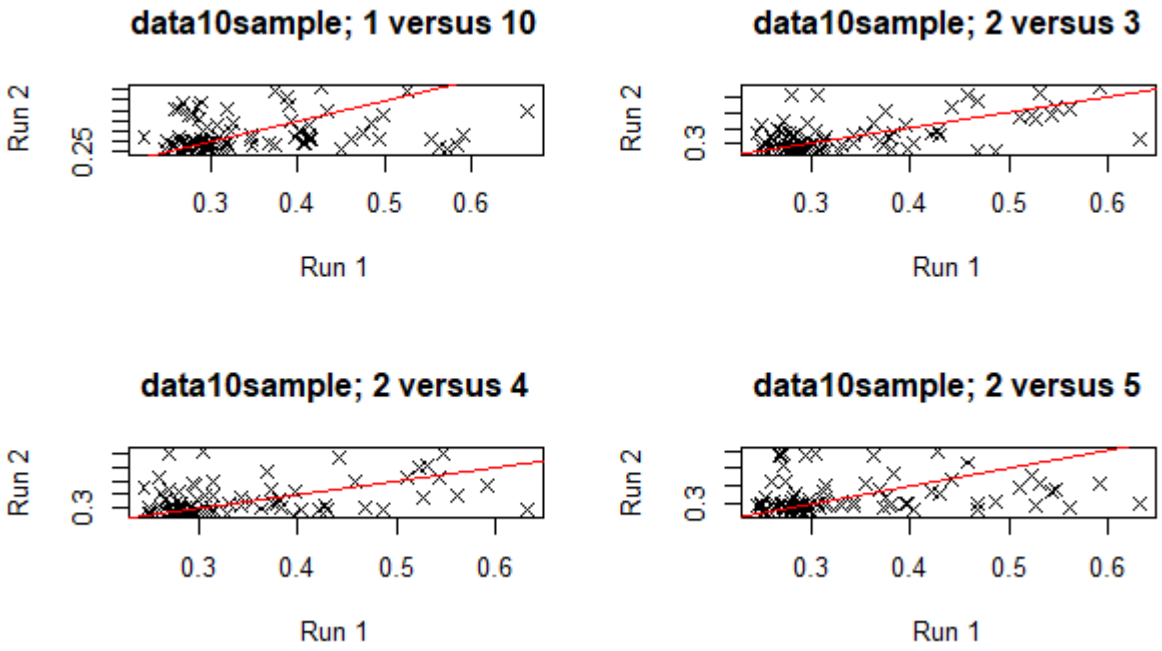

Figure 16: Caption

5.4 The 45 scatter plots between two out of the ten edge scores of the data samples of 100 songs

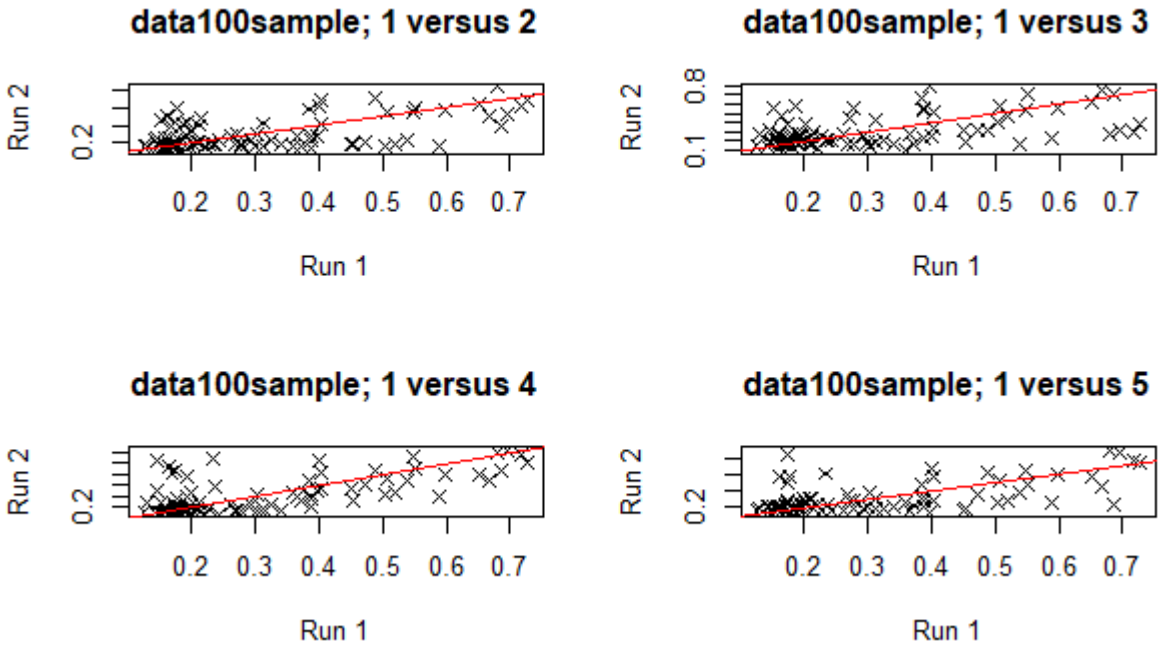

Figure 17: Caption

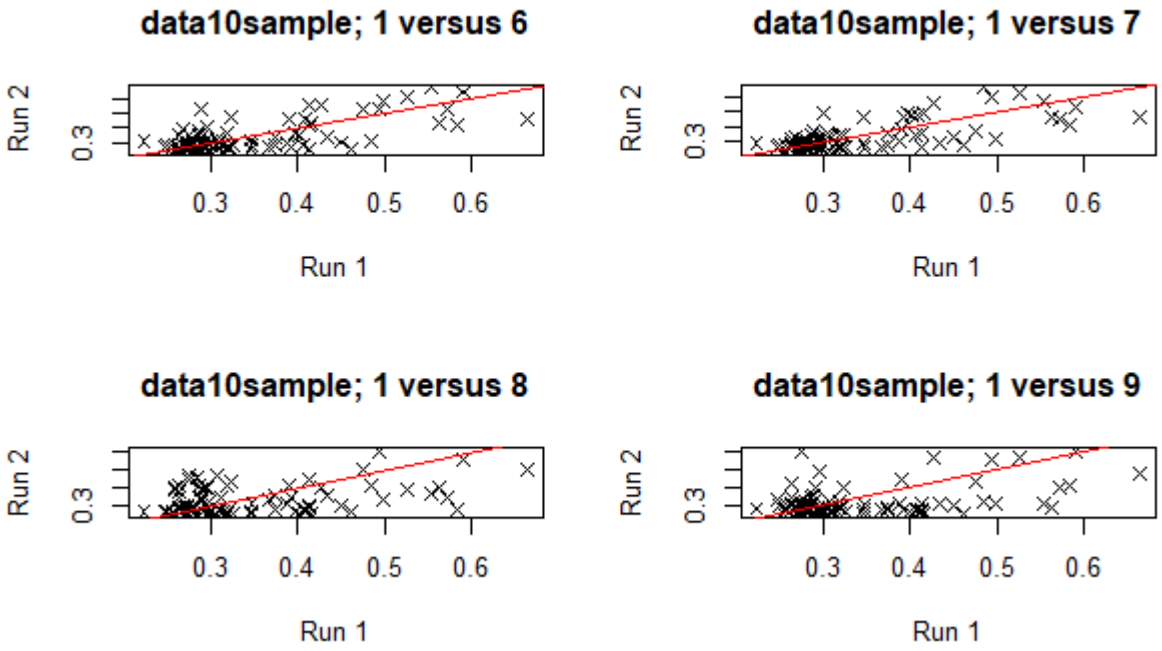

Figure 18: Caption

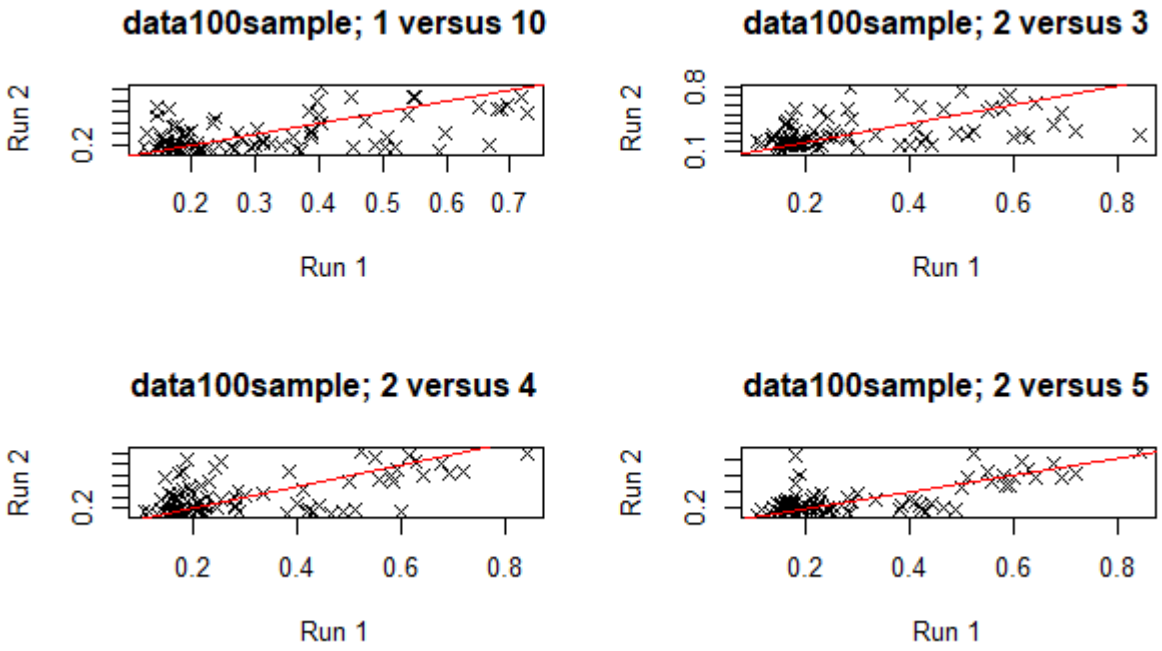

Figure 19: Caption

5.5 The 45 scatter plots between two out of the ten edge scores of the data samples of 500 songs

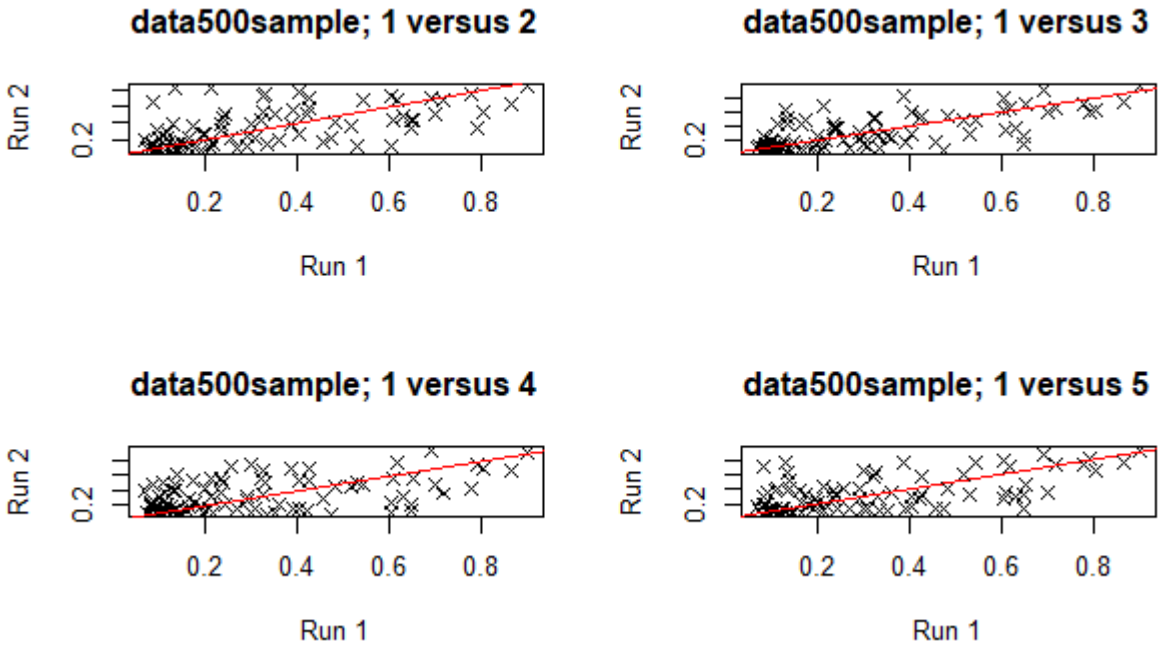

Figure 20: Caption

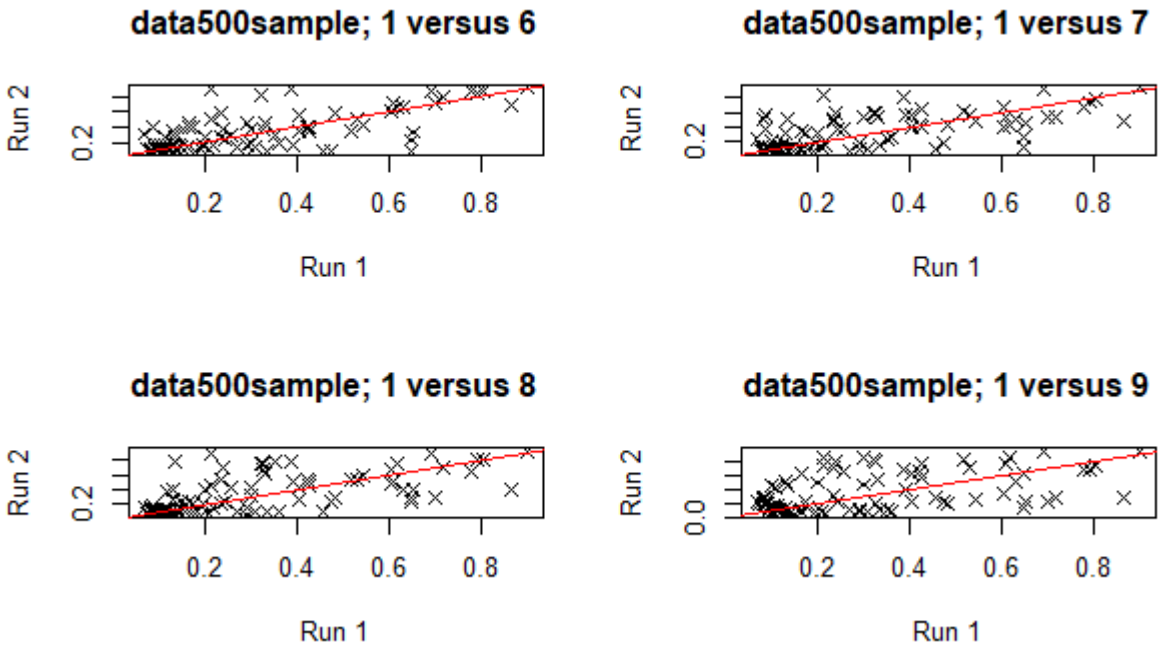

Figure 21: Caption

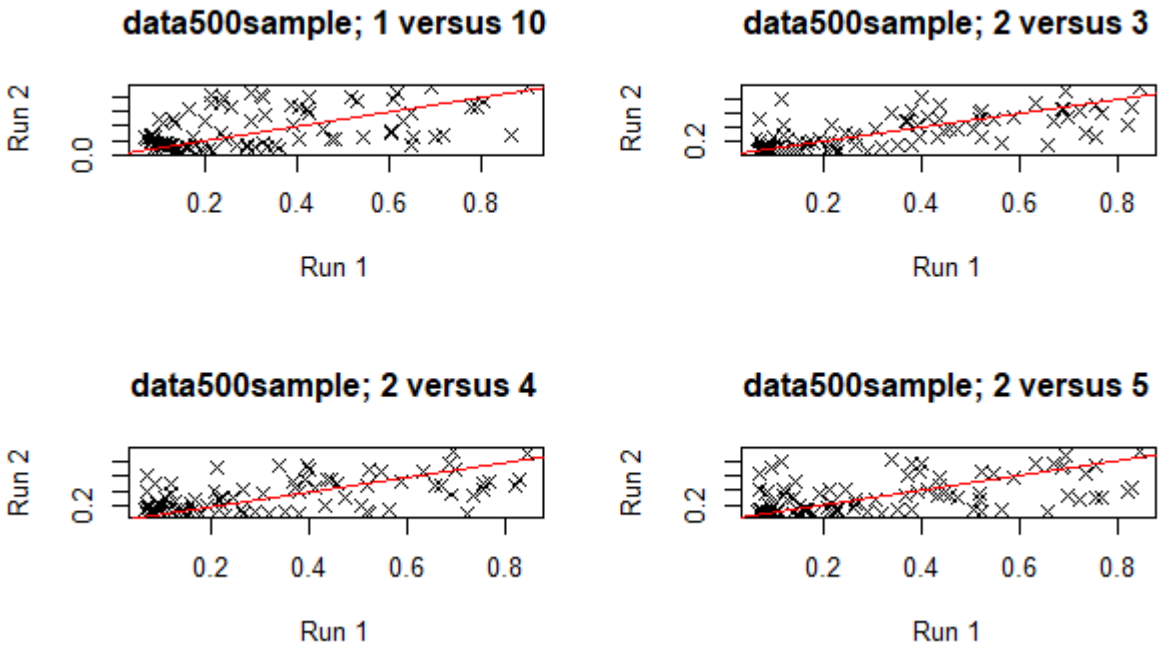

Figure 22: Caption

One result was quite peculiar, as all points seemed to be extremely close to the diagonal line. The plot is presented below.

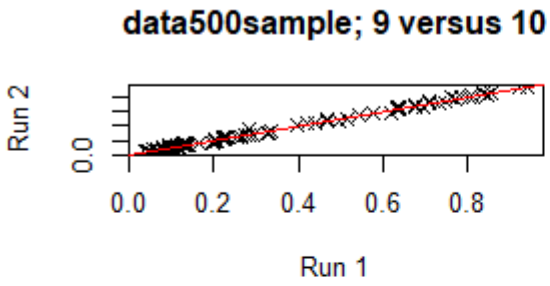

Figure 23: Caption

5.6 The 3 scatter plots of averages of the edge scores of the ten different samples of sizes 10, 100 and 500

- Made by average of 10 times down-sampling and average of 3 runs - Avav of 500 generally higher edge scores

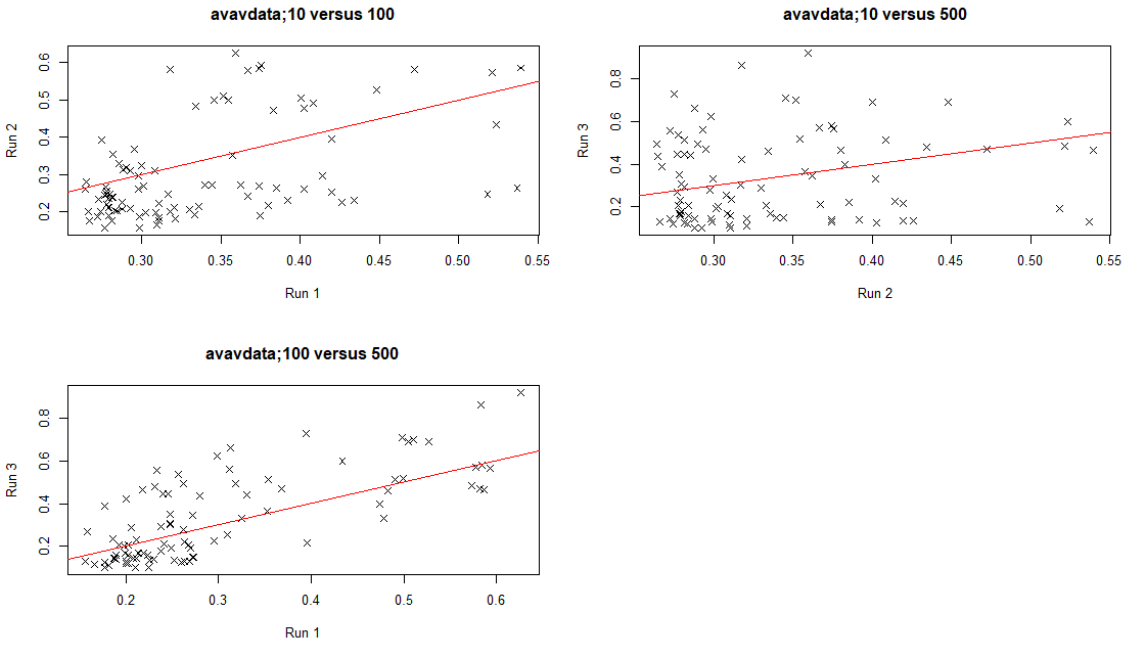

Figure 24: Scatter plots to compare different sample sizes

5.7 Resulting DAG

6 Discussion

The results weren't quite were was hoped for. This can have multiple good reasons. Unfortunately it is not clear yet, what was the main cause. However underneath there will be multiple options listed how to improve the research.
The songs weren't a randomly sampled. To improve the research, maybe in future research it could be sampled somehow directly from Spotify.
A second improvement would be, instead of assuming the variables to be Gausian normal distributed, they could be assumed to be distributed as the pictures downloaded from Spotify suggested.
A third improvement would be, to take for all of the different sample sizes the same number of iterations. Or to maybe include a metric to measure exactly how close the points are to the diagonal.

7 Conclusion

In this thesis, Bayesian networks were used to analyze a data set from Spotify API. The MCMC algorithm was applied to gain information about possible best graphs of zoiets. From this, scatter plots were made to compare different sample sizes of the data.

As can be seen from the scatter plots, the edge scores differ a lot for different data samples. That's why it wouldn't be good to infer one Bayesian Network out of them and their corresponding conditional (in)-dependencies.

References

Noah Amsterdam. Analyzing popular music using spotify's machine learning audio features. 2019.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

William M Bolstad and James M Curran. *Introduction to Bayesian statistics*. John Wiley & Sons, 2016.

Christian Borgelt, Jörg Gebhardt, and Rudolf Kruse. Graphical models. In *In Proceedings of International School for the Synthesis of Expert Knowledge (ISSEK'98*, pages 51–68. Wiley, 2002.

Gavin C Cawley and Nicola LC Talbot. Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, 8(Apr):841–861, 2007.

Nir Friedman and Daphne Koller. Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. *Machine learning*, 50(1-2):95–125, 2003.

Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe'er. Using bayesian networks to analyze expression data. *Journal of computational biology*, 7(3-4):601–620, 2000.

Dan Geiger and David Heckerman. Learning gaussian networks. In *Uncertainty Proceedings 1994*, pages 235–243. Elsevier, 1994.

Elena Georgieva, Marcella Suta, and Nicholas Burton. Hitpredict: Predicting hit songs using spotify data.

Marco Grzegorczyk. Statistical genomics.

Khalid Iqbal, Xu-Cheng Yin, Hong-Wei Hao, Qazi Mudassar Ilyas, and Hazrat Ali. An overview of bayesian network applications in uncertain domains. *International Journal of Computer Theory and Engineering*, 7(6):416, 2015.

Timo Koski and John Noble. *Bayesian networks: an introduction*, volume 924. John Wiley & Sons, 2011.

Kehan Luo. Machine learning approach for genre prediction on spotify top ranking songs. 2018.

Han-Saem Park, Ji-Oh Yoo, and Sung-Bae Cho. A context-aware music recommendation system using fuzzy bayesian networks with utility theory. In *International conference on Fuzzy systems and knowledge discovery*, pages 970–979. Springer, 2006.

Judea Pearl. Bayesian networks, 2011.

Karen Sachs, Omar Perez, Dana Pe'er, Douglas A Lauffenburger, and Garry P Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529, 2005.

Adriano V Werhli, Marco Grzegorczyk, and Dirk Husmeier. Comparative evaluation of reverse engineering gene regulatory networks with relevance networks, graphical gaussian models and bayesian networks. *Bioinformatics*, 22(20):2523–2531, 2006.

A R Code

A.1 Codes for structure MCMC Grzegorczyk

```
strMCMC <- function(Data,incidence,iterations,step_save, fan.in=nrow(Data)
    -1, v=1, mu=numeric(nrow(Data)), a=nrow(Data)+2, T_0=diag(0.5,nrow(Data)
    ,nrow(Data))){
n <- nrow(Data)                                    # number of nodes
m <- ncol(Data)                                    # number of observations

T_m <- T_0 + (m-1)* cov(t(Data)) + ((v*m)/(v+m))* (mu - rowMeans(Data))%*%t
    (mu - rowMeans(Data))

L1 <- list()     # incidence matrix
L2 <- list()     # log BGe score
#
    ##############################################################################

##### functions we need in the algorithm

### calculation of the first ancestor matrix:
ancestor <- function(incidence){
incidence1 <- incidence
incidence2 <- incidence
k <- 1
while (k < nrow(incidence)){
incidence1 <- incidence1%*%incidence
incidence2 <- incidence2 + incidence1
k <-k+1
}
incidence2[which(incidence2[,]>0)] <- 1
return(t(incidence2))}

### function for the computation of c(n, alpha)
c_function <- function(N,A){
fact <- numeric(N)
for (i in 1:N){
fact[i] <- -lgamma((A+1-i)/2)
}
product <- sum(fact) -(A*N/2)*log(2)- (N*(N-1)/4)*log(pi)
return(product)}


top_order <- function(incidence){
Order <- numeric(n)
fan_in <- numeric(n)
no_fan_in <- numeric(0)
m <- 1
```

```
for (p in 1:n){                                          # number of parent
    nodes at the beginning
fan_in[p] <- sum(incidence[,p])
}
no_fan_in <- which(fan_in==0)
while (length(which(Order==0))>0){                       # as long as there is
     a node without an order
fan_in[which(incidence[no_fan_in[1],]==1)] <- fan_in[which(incidence[no_fan
    _in[1],]==1)] - 1
no_fan_in <- c(no_fan_in, c(which(incidence[no_fan_in[1],]==1),which(fan_in
    ==0))[duplicated(c(which(incidence[no_fan_in[1],]==1),which(fan_in==0)))
    ])
Order[m] <- no_fan_in[1]
no_fan_in <- no_fan_in[-1]
m <- m+1
}
return(Order)
}


### assign the topological order of the descendants of the child
des_top_order <- function(incidence, ancest1,child){
top <- top_order(incidence)
position_child <- which(top==child)
top_all_after <- top[position_child:n]                   # top. order without
    the "first" nodes
desc <- which(ancest1[,child]==1)                        # descendants of the
    child
inter_step <- c(child,desc,top_all_after)
des_top <- inter_step[which(duplicated(inter_step))]
return(des_top)
}

#
    ##############################################################################

### computation of the (logarithmizid) BGe Score of the FIRST graph
P_local_num <- numeric(n)    ###  numerator of the factors
P_local_den <- numeric(n)    ### denumerator of the factors

for (j in 1:n)  {
n_nodes <- which(incidence[,j]==1)          # parents of j
P_local_num[j] <- (-(length(n_nodes)+1)*m/2)*log(2*pi) + ((length(n_nodes)
    +1)/2)*log(v/(v+m)) + c_function((length(n_nodes)+1),a)-c_function((
    length(n_nodes)+1),a+m)+ (a/2)*log(det(as.matrix(T_0[sort(c(n_nodes,j)),
    sort(c(n_nodes,j))])))+ (-(a+m)/2)*log(det(as.matrix(T_m[sort(c(n_nodes,
    j)),sort(c(n_nodes,j))])))
if(sum(incidence[,j])>0){           # if j has at least one parent
P_local_den[j] <- (-(length(n_nodes))*m/2)*log(2*pi) + (length(n_nodes)/2)*
    log(v/(v+m)) + c_function(length(n_nodes),a)- c_function(length(n_nodes)
```

```
    ,a+m)+ (a/2)*log(det(as.matrix(T_0[n_nodes,n_nodes])))+ (-(a+m)/2)*log(
    det(as.matrix(T_m[n_nodes,n_nodes])))
}
else{                                   # if j has no parents
P_local_den[j] <- 0
}
}
bge_old <- (sum(P_local_num))-(sum(P_local_den))

# first ancestor matrix
ancest1 <- ancestor(incidence)

####### ... the number of neighbour graphs/proposal probability for the
    FIRST graph
### 1.) number of neighbour graphs obtained by edge deletions
num_deletion <- sum(incidence)

### 2.) number of neighbour graphs obtained by edge additions    1- E(i,j)
    - I(i,j) - A(i,j)
inter_add <- which(matrix(rep(1,n*n),nrow=n) - diag(1,n,n) - incidence -
    ancest1 >0)
add <- matrix(numeric(n*n),nrow=n)
add[inter_add] <- 1
add[,which(colSums(incidence)>fan.in-1)] <- 0
num_addition <- sum(add)

### 3.) number of neighbour graphs obtained by edge reversals    I - (I^t *
     A)^t
inter_rev <- which(incidence - t(t(incidence)%*% ancest1)==1)
re <- matrix(numeric(n*n),nrow=n)
re[inter_rev] <- 1
re[which(colSums(incidence)>fan.in-1),] <- 0 # CORRECTED!!!???!!!
num_reversal <- sum(re)

##### total number of neighbour graphs:
total <- sum(num_deletion,num_addition,num_reversal)

### proposal probability:
proposal <- 1/total

############## sampling a new graph (or rather sampling an edge to shift)
### sample one of the three single edge operations
random <- sample(1:total,1)

operation <- 0                          # memorise, if the single edge
    operation is (will be) an edge reversal
if (random > total - num_reversal){
operation <- 1}

#### shifting of the incidence matrix
```

```
incidence_new <- incidence

if (random <= num_deletion){              # if edge deletion was sampled
if(length(which(incidence>0))>1){
new_edge <- sample(which(incidence>0),1)} # sample one of the existing
    edges
else
{new_edge <- which(incidence>0)}
incidence_new[new_edge] <- 0}             # and delete it

if (random > (total - num_reversal)){       # if edge reversal was sampled
if(num_reversal>1){
new_edge <- sample(which(re==1),1)}  # sample one of the existing edges
    where a reversal leads to a valid graph
else{
new_edge <- which(re==1)}
incidence_new[new_edge] <- 0              # delete it
junk <- matrix(numeric(n*n),nrow=n)       # creating a matrix with all
    entries zero
junk[new_edge] <- 1                       # an only a "1" at the entry of
    the new (reversed) edge
incidence_new <- incidence_new + t(junk)}# sum the deleted matrix and the "
    junk-matrix"

if (random <= (total - num_reversal) & random > num_deletion){     # if
    edge addition was sampled
if(num_addition>1){
new_edge <- sample(which(add==1),1)} # sample one of the existing edges
    where a addition leads to a valid graph
else{
new_edge <- which(add==1)}
incidence_new[new_edge] <- 1              # and add it
}


#################### Updating the ancestor matrix

# creating a matrix with dimensions of the incidence matrix and all entries
     zero except for the entry of the chosen edge
help_matrix <- matrix(numeric(n*n),nrow=n)
help_matrix[new_edge] <- 1

# numbers of the nodes that belong to the shifted egde
parent <- which(rowSums(help_matrix)==1)
child <- which(colSums(help_matrix)==1)

### updating the ancestor matrix (after edge reversal)
## edge deletion
ancestor_new <- ancest1
if (operation==1){
```

```
ancestor_new[c(child,which(ancest1[,child]==1)),] <- 0            # delete
    all ancestors of the child and its descendants
                                                    #
top_name <- des_top_order(incidence_new, ancest1, child)
for (d in top_name){
for(g in which(incidence_new[,d]==1)) {
ancestor_new[d,c(g,(which(ancestor_new[g,]==1)))] <- 1
}
}
## edge addition
anc_parent <- which(ancestor_new[child,]==1)                      #
    ancestors of the new parent
des_child <- which(ancestor_new[,parent]==1)                      #
    descendants of the child
ancestor_new[c(parent,des_child),c(child,anc_parent)] <- 1
}

### updating the ancestor matrix (after edge deletion)
if (random <= num_deletion){
ancestor_new[c(child,which(ancest1[,child]==1)),] <- 0            # delete
    all ancestors of the child and its descendants
                                                    #
top_name <- des_top_order(incidence_new, ancest1, child)
for (d in top_name){
for(g in which(incidence_new[,d]==1)) {
ancestor_new[d,c(g,(which(ancestor_new[g,]==1)))] <- 1
}
}
}

# updating the ancestor matrix (after edge addition)
if (random <= total - num_reversal & random > num_deletion){
anc_parent <- which(ancest1[parent,]==1)         # ancestors of the new
    parent
des_child <- which(ancest1[,child]==1)           # descendants of the child
ancestor_new[c(child,des_child),c(parent,anc_parent)] <- 1
}

####### ... the number of neighbour graphs/proposal probability for the
    proposed graph
### 1.) number of neighbour graphs obtained by edge deletions
num_deletion_new <- sum(incidence_new)

### number of neighbour graphs obtained by edge additions    1- E(i,j) - I(
    i,j) - A(i,j)
inter_add.new <- which(matrix(rep(1,n*n),nrow=n) - diag(1,n,n) - incidence_
    new - ancestor_new >0)
add.new <- matrix(numeric(n*n),nrow=n)
add.new[inter_add.new] <- 1
add.new[,which(colSums(incidence_new)>fan.in-1)] <- 0
```

```
num_addition_new <- sum(add.new)

### number of neighbour graphs obtained by edge reversals    I - (I^t * A)^
    t
inter_rev.new <- which(incidence_new - t(t(incidence_new)%*% ancestor_new)
    ==1)
re.new <- matrix(numeric(n*n),nrow=n)
re.new[inter_rev.new] <- 1
re.new[which(colSums(incidence_new)>fan.in-1),] <- 0   # CORRECTED!!!???!!!
num_reversal_new <- sum(re.new)

##### total number of neighbour graphs:
total_new <- sum(num_deletion_new,num_addition_new,num_reversal_new)

### proposal probability:
proposal_new <- 1/total_new

### BGe Score for the new graph
P_local_num_new <- P_local_num
P_local_den_new <- P_local_den
n_nodes_new <- which(incidence_new[,child]==1)

P_local_num_new[child] <- (-(length(n_nodes_new)+1)*m/2)*log(2*pi) + ((
    length(n_nodes_new)+1)/2)*log(v/(v+m)) + c_function((length(n_nodes_new)
    +1),a)-c_function((length(n_nodes_new)+1),a+m)+ (a/2)*log(det(as.matrix(
    T_0[sort(c(n_nodes_new,child)),sort(c(n_nodes_new,child))])))+ (-(a+m)/
    2)*log(det(as.matrix(T_m[sort(c(n_nodes_new,child)),sort(c(n_nodes_new,
    child))])))

if(sum(incidence_new[,child])>0){           # if child at least one parent
P_local_den_new[child] <- (-(length(n_nodes_new))*m/2)*log(2*pi) + (length(
    n_nodes_new)/2)*log(v/(v+m)) + c_function(length(n_nodes_new),a)- c_
    function(length(n_nodes_new),a+m)+ (a/2)*log(det(as.matrix(T_0[n_nodes_
    new,n_nodes_new])))+ (-(a+m)/2)*log(det(as.matrix(T_m[n_nodes_new,n_
    nodes_new])))
}
else{                                       # if child has no parents
P_local_den_new[child] <- 0
}

if (operation==1){                          # if single edge operation was
    an edge reversal
n_nodesP <- which(incidence_new[,parent]==1)
P_local_num_new[parent] <- (-(length(n_nodesP)+1)*m/2)*log(2*pi) + ((length
    (n_nodesP)+1)/2)*log(v/(v+m)) + c_function((length(n_nodesP)+1),a)-c_
    function((length(n_nodesP)+1),a+m)+ (a/2)*log(det(as.matrix(T_0[sort(c(n
    _nodesP,parent)),sort(c(n_nodesP,parent))])))+ (-(a+m)/2)*log(det(as.
    matrix(T_m[sort(c(n_nodesP,parent)),sort(c(n_nodesP,parent))])))
if(sum(incidence_new[,parent])>0){          # if parent at least one parent
P_local_den_new[parent] <- (-(length(n_nodesP))*m/2)*log(2*pi) + (length(n_
```

```
    nodesP)/2)*log(v/(v+m)) + c_function(length(n_nodesP),a)- c_function(
    length(n_nodesP),a+m)+ (a/2)*log(det(as.matrix(T_0[n_nodesP,n_nodesP])))
    + (-(a+m)/2)*log(det(as.matrix(T_m[n_nodesP,n_nodesP])))
}
else{                                           # if parent has no parents
P_local_den_new[parent] <- 0
}
}
bge_new <- (sum(P_local_num_new))-(sum(P_local_den_new))


L1[[1]] <- incidence                            # initial graph
L2[[1]] <- bge_old                              # and it's BGe score

acceptance <- min(1, exp((bge_new + log(proposal_new)) - (bge_old  + log(
    proposal))))
rand <- runif(1)

if(acceptance > rand){
incidence <- incidence_new
bge_old <- bge_new
P_local_num <- P_local_num_new
P_local_den <- P_local_den_new
proposal <- proposal_new
ancest1 <- ancestor_new
total <- total_new
num_deletion <- num_deletion_new
num_addition <- num_addition_new
num_reversal <- num_reversal_new
add <- add.new
re <- re.new
}

#
    ##############################################################################################

#
    ###############################################################################


for (z in 2:((iterations/step_save)+1)){
for (count in 1:step_save){

############## sampling a new graph (or rather sampling an edge to shift)
### sample one of the three single edge operations
random <- sample(1:total,1)

operation <- 0                              # memorise, if the single edge
    operation is (will be) an edge reversal
if (random > total - num_reversal){
```

```
operation <- 1}

#### shifting of the incidence matrix
incidence_new <- incidence

if (random <= num_deletion){               # if edge deletion was sampled
if(length(which(incidence>0))>1){
new_edge <- sample(which(incidence>0),1)} # sample one of the existing
    edges
else
{new_edge <- which(incidence>0)}
incidence_new[new_edge] <- 0}             # and delete it

if (random > (total - num_reversal)){     # if edge reversal was sampled
if(num_reversal>1){
new_edge <- sample(which(re==1),1)}       # sample one of the existing edges
     where a reversal leads to a valid graph
else{
new_edge <- which(re==1)}
incidence_new[new_edge] <- 0              # delete it
junk <- matrix(numeric(n*n),nrow=n)       # creating a matrix with all
    entries zero
junk[new_edge] <- 1                       # an only a "1" at the entry of
    the new (reversed) edge
incidence_new <- incidence_new + t(junk)}# sum the deleted matrix and the "
    junk-matrix"

if (random <= (total - num_reversal) & random > num_deletion){     # if
    edge addition was sampled
if(num_addition>1){
new_edge <- sample(which(add==1),1)} # sample one of the existing edges
    where a addition leads to a valid graph
else{
new_edge <- which(add==1)}
incidence_new[new_edge] <- 1              # and add it
}


### Updating the ancestor matrix

# creating a matrix with dimensions of the incidence matrix and all entries
     zero except for the entry of the chosen edge
help_matrix <- matrix(numeric(n*n),nrow=n)
help_matrix[new_edge] <- 1

# numbers of the nodes that belong to the shifted egde
parent <- which(rowSums(help_matrix)==1)
child <- which(colSums(help_matrix)==1)

### updating the ancestor matrix (after edge reversal)
```

```
## edge deletion
ancestor_new <- ancest1
if (operation==1){
ancestor_new[c(child,which(ancest1[,child]==1)),] <- 0    # delete all
    ancestors of the child and its descendants
                                                          #

top_name <- des_top_order(incidence_new, ancest1, child)
for (d in top_name){
for(g in which(incidence_new[,d]==1)) {
ancestor_new[d,c(g,(which(ancestor_new[g,]==1)))] <- 1
}
}

anc_parent <- which(ancestor_new[child,]==1)           # ancestors of the
    new parent
des_child <- which(ancestor_new[,parent]==1)           # descendants of the
    child
ancestor_new[c(parent,des_child),c(child,anc_parent)] <- 1
}

### updating the ancestor matrix (after edge deletion)
if (random <= num_deletion){
ancestor_new[c(child,which(ancest1[,child]==1)),] <- 0    # delete all
    ancestors of the child and its descendants
                                                          #
top_name <- des_top_order(incidence_new, ancest1, child)
for (d in top_name){
for(g in which(incidence_new[,d]==1)) {
ancestor_new[d,c(g,(which(ancestor_new[g,]==1)))] <- 1
}
}
}

# updating the ancestor matrix (after edge addition)
if (random <= total - num_reversal & random > num_deletion){
anc_parent <- which(ancest1[parent,]==1)              # ancestors of the new
     parent
des_child <- which(ancest1[,child]==1)                # descendants of the
    child
ancestor_new[c(child,des_child),c(parent,anc_parent)] <- 1
}

####### ... the number of neighbour graphs/proposal probability for the
    proposed graph
### 1.) number of neighbour graphs obtained by edge deletions
num_deletion_new <- sum(incidence_new)

### number of neighbour graphs obtained by edge additions    1- E(i,j) - I(
    i,j) - A(i,j)
```

```
inter_add.new <- which(matrix(rep(1,n*n),nrow=n) - diag(1,n,n) - incidence_
    new - ancestor_new >0)
add.new <- matrix(numeric(n*n),nrow=n)
add.new[inter_add.new] <- 1
add.new[,which(colSums(incidence_new)>fan.in-1)] <- 0
num_addition_new <- sum(add.new)

### number of neighbour graphs obtained by edge reversals    I - (I^t * A)^
    t
inter_rev.new<- which(incidence_new - t(t(incidence_new)%*% ancestor_new)
    ==1)
re.new <- matrix(numeric(n*n),nrow=n)
re.new[inter_rev.new] <- 1
re.new[,which(colSums(incidence_new)>fan.in-1)] <- 0
num_reversal_new <- sum(re.new)

##### total number of neighbour graphs:
total_new <- sum(num_deletion_new, num_addition_new, num_reversal_new)

### proposal probability:
proposal_new <- 1/total_new

### BGe Score for the new graph
P_local_num_new <- P_local_num
P_local_den_new <- P_local_den
n_nodes_new <- which(incidence_new[,child]==1)

P_local_num_new[child] <- (-(length(n_nodes_new)+1)*m/2)*log(2*pi) + ((
    length(n_nodes_new)+1)/2)*log(v/(v+m)) + c_function((length(n_nodes_new)
    +1),a)-c_function((length(n_nodes_new)+1),a+m)+ (a/2)*log(det(as.matrix(
    T_0[sort(c(n_nodes_new,child)),sort(c(n_nodes_new,child))])))+ (-(a+m)/
    2)*log(det(as.matrix(T_m[sort(c(n_nodes_new,child)),sort(c(n_nodes_new,
    child))])))

if(sum(incidence_new[,child])>0){        # if child at least one parent
P_local_den_new[child] <- (-(length(n_nodes_new))*m/2)*log(2*pi) + (length(
    n_nodes_new)/2)*log(v/(v+m)) + c_function(length(n_nodes_new),a)- c_
    function(length(n_nodes_new),a+m)+ (a/2)*log(det(as.matrix(T_0[n_nodes_
    new,n_nodes_new])))+ (-(a+m)/2)*log(det(as.matrix(T_m[n_nodes_new,n_
    nodes_new])))
}
else{                                    # if child has no parents
P_local_den_new[child] <- 0
}

if (operation==1){                       # if single edge operation was an
    edge reversal
n_nodesP <- which(incidence_new[,parent]==1)
P_local_num_new[parent] <- (-(length(n_nodesP)+1)*m/2)*log(2*pi) + ((length
    (n_nodesP)+1)/2)*log(v/(v+m)) + c_function((length(n_nodesP)+1),a)-c_
```

```
    function((length(n_nodesP)+1),a+m)+ (a/2)*log(det(as.matrix(T_0[sort(c(n
    _nodesP,parent)),sort(c(n_nodesP,parent))])))+ (-(a+m)/2)*log(det(as.
    matrix(T_m[sort(c(n_nodesP,parent)),sort(c(n_nodesP,parent))])))
if(sum(incidence_new[,parent])>0){       # if parent at least one parent
P_local_den_new[parent] <- (-(length(n_nodesP))*m/2)*log(2*pi) + (length(n_
    nodesP)/2)*log(v/(v+m)) + c_function(length(n_nodesP),a)- c_function(
    length(n_nodesP),a+m)+ (a/2)*log(det(as.matrix(T_0[n_nodesP,n_nodesP])))
    + (-(a+m)/2)*log(det(as.matrix(T_m[n_nodesP,n_nodesP])))
}
else{                                    # if parent has no parents
P_local_den_new[parent] <- 0
}
}
bge_new <- (sum(P_local_num_new))-(sum(P_local_den_new))

acceptance <- min(1, exp((bge_new +log(proposal_new)) - (bge_old +log(
    proposal))))
rand <- runif(1)

if(acceptance > rand){
incidence <- incidence_new
bge_old <- bge_new
P_local_num <- P_local_num_new
P_local_den <- P_local_den_new
proposal <- proposal_new
ancest1 <- ancestor_new
total <- total_new
num_deletion <- num_deletion_new
num_addition <- num_addition_new
num_reversal <- num_reversal_new
add <- add.new
re <- re.new
}
}

L1[[z]] <- incidence
L2[[z]] <- bge_old
}
return(list(L1,L2))
}

#
    ##############################################################################


child <- function(edges,n){          # input: the numbers of the edges in
    the incidence matrix and the number of nodes
p <- ceiling(edges/n)
return(p)
}
```

```
parent <- function(edges,n){
ch <- edges + n - child(edges,n)*n
return(ch)
}


top_order <- function(incidence){
n <- nrow(incidence)
Order <- numeric(n)
fan_in <- numeric(n)
no_fan_in <- numeric(0)
m <- 1
for (p in 1:n){                                          # number of parent
    nodes at the beginning
fan_in[p] <- sum(incidence[,p])
}
no_fan_in <- which(fan_in==0)
while (length(which(Order==0))>0){                       # as long as there is
     a node without an order
fan_in[which(incidence[no_fan_in[1],]==1)] <- fan_in[which(incidence[no_fan
    _in[1],]==1)] - 1
no_fan_in <- c(no_fan_in, c(which(incidence[no_fan_in[1],]==1),which(fan_in
    ==0))[duplicated(c(which(incidence[no_fan_in[1],]==1),which(fan_in==0)))
    ])
Order[m] <- no_fan_in[1]
no_fan_in <- no_fan_in[-1]
m <- m+1
}
return(Order)
}

#
    ##############################################################################

order.edges <- function(incidence){
top.order <- top_order(incidence)
n <- length(top.order)
edges <- which(incidence!=0)
children <- child(edges,n)
parents <- parent(edges,n)
m <- length(edges)
ordered_edges  <- numeric(m)
incidence_n <- incidence
tog <- matrix(c(edges,parents,children,ordered_edges),ncol=4, byrow=FALSE)
  k <- 1
 while(any(tog[,4]==0)){
  node1 <- top.order[which(colSums(incidence_n[,top.order])>0)][1]     #
      first node in top. order that has at least one parent
   par1<- tog[which(tog[,3]==node1),2]                   # find the parents of
```

```
        first child in the top. order that has an unordered edge incident
       into it
   g <- par1[which(par1>0)]
   f1 <- numeric(length(g))
     for (i in 1:length(g)){
      f1[i] <- which(top.order==g[i])
      }
   par2 <- g[which.max(f1)]                              # find the highest
       ordered node that has an edge leading into node1
   tog[which(tog[,2]==par2 & tog[,3]==node1),4] <- k
   k <- k + 1
   incidence_n[tog[which(tog[,2]==par2 & tog[,3]==node1),1]] <- 0      #
       delete the edge in the "incidence" matrix
   tog[which(tog[,2]==par2 & tog[,3]==node1),2] <- 0
    }
    to <- matrix(c(edges,parents,children,tog[,4]),ncol=4,byrow=FALSE)
   return(to)                                             # return the whole
       matrix, the order is the fourth column
}

#
   ##############################################################################

### DAG-to-CPDAG algorithm
# +1 if the edge is "compelled"
#   -1 if the edge is "reversible"
#########################################################################
cpdag <- function(incidence){
z <- order.edges(incidence)
 new_mat <- cbind(z,numeric(nrow(z)))     # edges, parents, children, order,
     zeros
 n_mat <- new_mat[order(new_mat[,4]),]    # sort the edges by its order
 vec <- numeric(nrow(z))
  while(any(vec==0)){                                    # while there are
      unlabeled edges            l.3
  if (length(vec)>1){                                    # if there are at
      least 2 edges
    first <- which(n_mat[,5]==0)[1]                      # first EDGE that
       ist labeled "unknown" (0)  l.4
    parent1 <- n_mat[first,2]                            # x   parent NODE
    child1 <- n_mat[first,3]                             # y   child NODE
     comp1 <- n_mat[which(n_mat[,3]==parent1 & n_mat[,5]==1),2]       # w
         NODES that have an edge incident into the parent labeled compelled)
     }
      if (length(vec)==1){
     first <- which(n_mat[5]==0)                         # first edge that ist
          labeled "unknown" (0)
      parent1 <- n_mat[2]                                # x   parent
      child1 <- n_mat[3]                                 # y   child
      comp1 <- numeric(0)
```

```
    }
for (j in comp1){                                          #
                                                  l.5
   if (incidence[j,child1]==0){                            # if w is not a
       parent of the child          l.6
    n_mat[first,5] <- 1                                    # label x -> y
        compelled                        l.7
    n_mat[which(n_mat[,3]==child1),5] <- 1                 # label every edge
        incident into y compelled l.7
    vec[first] <- 1
    vec[which(n_mat[,3]==child1)] <- 1
    break
   }
   if (incidence[j,child1]!=0)    {
    n_mat[which(n_mat[,2]==j & n_mat[,3]==child1),5] <- 1  # label w -> y
        compelled                 l.10
    vec[which(n_mat[,2]==j & n_mat[,3]==child1)] <- 1
    }
}
if (length(vec)>1){
if(n_mat[first,5]==0){

   moep <- n_mat[which(n_mat[,3]==child1 & n_mat[,2]!=parent1),2]       #
       other parents of the child
   if(length(moep)>0){                                     #
                          l.11
   for(o in moep){
   if(incidence[o,parent1]==0){
     vec[first] <- 1
     vec[which(n_mat[,3]==child1 & n_mat[,5]==0)] <- 1
     n_mat[first,5] <- 1                                                # label x ->
         y compelled
     n_mat[which(n_mat[,3]==child1 & n_mat[,5]==0),5] <- 1   # label all
        "unknown" edges incident into y compelled
      break
      }
    if(all(incidence[moep,parent1]!=0)){
     vec[first] <- -1
     vec[which(n_mat[,3]==child1 & n_mat[,5]==0)] <- -1
     n_mat[first,5] <- -1                                               # label x ->
         y reversible
     n_mat[which(n_mat[,3]==child1 & n_mat[,5]==0),5] <- -1  # label all
        "unknown" edges incident into y reversible
   }
}
}
if(length(moep)==0){
     vec[first] <- -1
     vec[which(n_mat[,3]==child1 & n_mat[,5]==0)] <- -1
     n_mat[first,5] <- -1                                               # label x ->
```

```
            y reversible
        n_mat[which(n_mat[,3]==child1 & n_mat[,5]==0),5] <- -1   # label all
            "unknown" edges incident into y reversible
      }
                    }
      }
   if (length(vec)==1){
      n_mat[5] <- -1                                    # label x -> y
          reversible
      vec <- -1
     }
}
return(n_mat)
}


#
    ##############################################################################


cpdag_list <- function(list.inc,E){     # E: end of burnIn phase
L <- list()
G <- list()
nodes <- dim(list.inc[[1]])[1]
mat.sum <- matrix(numeric(nodes*nodes),nrow=nodes)
  for (i in E:length(list.inc)){
    k <- cpdag(list.inc[[i]])
    dummy <- matrix(numeric(nodes*nodes),nrow=nodes)
    if(length(nrow(k))!=0){
    dummy[k[,1]] <- k[,5]
   L[[i]] <- dummy
   }
   if(length(nrow(k))==0 && length(k)>0){
     dummy[k[1]] <- k[5]
     L[[i]] <- dummy
    }
   mat.com <-matrix(numeric(nodes*nodes),nrow=nodes)
   mat.re <- matrix(numeric(nodes*nodes),nrow=nodes)
   com <- which(L[[i]]>0)
   re <- which(L[[i]]<0)
   mat.com[com] <- 1
   mat.re[re] <- 1
   mat <- mat.com + mat.re + t(mat.re)
   G[[i]] <- mat
     mat.sum <- mat.sum + mat
  }
 return(list(L,G, (mat.sum/(length(list.inc)- E+1))))
}
```

```
# m_obs: number of samples
# var_noise: variance of Gaussian distributed noise terms

make_test_Data <- function(m_obs, var_noise){
edges <- 20
a1 <- runif(edges,0.5,2)
a2 <- sample(c(-1,1),edges, replace=TRUE)
a <- a1*a2     # vector with regression coefficients for the 20 edges

# 1. pip3
x_pip3 <- rnorm(m_obs, sd=1)
pip3 <- (x_pip3 - mean(x_pip3))/sd(x_pip3)

# 2. plcg
x_plcg <- a[1]* pip3 + rnorm(m_obs, sd=sqrt(var_noise))
plcg <- (x_plcg - mean(x_plcg))/sd(x_plcg)

# 3. pip2
x_pip2 <- a[2]* pip3 + a[3]*plcg + rnorm(m_obs, sd=sqrt(var_noise))
pip2 <- (x_pip2 - mean(x_pip2))/sd(x_pip2)

# 4. pkc
x_pkc <- a[4]* pip2 + a[5]*plcg + rnorm(m_obs, sd=sqrt(var_noise))
pkc  <- (x_pkc - mean(x_pkc))/sd(x_pkc)

# 5. pka
x_pka <- a[6]* pkc + rnorm(m_obs, sd=sqrt(var_noise))
pka  <- (x_pka - mean(x_pka))/sd(x_pka)

# 6. jnk
x_jnk <- a[7]* pkc + a[8]* pka + rnorm(m_obs, sd=sqrt(var_noise))
jnk  <- (x_jnk - mean(x_jnk))/sd(x_jnk)

# 7. p38
x_p38 <- a[9]* pkc + a[10]* pka + rnorm(m_obs, sd=sqrt(var_noise))
p38  <- (x_p38 - mean(x_p38))/sd(x_p38)

# 8. raf
x_raf <- a[11]* pkc + a[12]* pka + rnorm(m_obs, sd=sqrt(var_noise))
raf  <- (x_raf - mean(x_raf))/sd(x_raf)

# 9. mek
x_mek <- a[13]* pkc + a[14]* pka + a[15]* raf + rnorm(m_obs, sd=sqrt(var_
    noise))
mek  <- (x_mek - mean(x_mek))/sd(x_mek)

# 10. erk
x_erk <- a[16]* pka + a[17]* mek + rnorm(m_obs, sd=sqrt(var_noise))
erk  <- (x_erk - mean(x_erk))/sd(x_erk)
```

```
# 11. akt
x_akt <- a[18]* pip3 + a[19]* pka + a[20]* erk + rnorm(m_obs, sd=sqrt(var_
    noise))
akt  <- (x_akt - mean(x_akt))/sd(x_akt)

daten <- cbind(pip3, plcg, pip2, pkc, pka, jnk, p38, raf, mek, erk, akt)

return(t(daten))
}


#
    ##############################################################################

###################### function for plotting the ROC curve

library(limma)
library(ROC)

draw_ROC <- function(postP, trueEdges, steps=0.01){


n1 <- numeric(0)
for(i in 1:dim(postP)[1]){
        for(j in 1:dim(postP)[2]){
                                    if (abs(i-j)>0){
                                    n1 <- c(n1, postP[i,j])
                                    }
                                    }
                           }

n2 <- numeric(0)
for(i in 1:dim(trueEdges)[1]){
       for(j in 1:dim(trueEdges)[2]){
                                if (abs(i-j)>0){
                                n2 <- c(n2, trueEdges[i,j])
                                        }
                                }
                           }

sp <- sort(n1)          # posterior probabilities
st <- n2[order(n1)]     # true ordered by posterior probabilities

rocc.obj <- rocdemo.sca(n2, n1, rule=NULL, cutpts=NA)#seq(0,1,steps))

return(list(plot((1-rocc.obj"spec"),
    rocc.obj"sens", type="l",las=1, xlab="1 - specificity", ylab="sensitivity
    ", main='ROC curve'), abline(0,1, lty=2)))
```

```
}



#
    ##############################################################################


make_true_Net <- function(){

NETWORK <- matrix(numeric(11*11),11,11)

# 1. pip3

# 2. plcg
NETWORK[1,2] <- 1

# 3. pip2
NETWORK[1,3] <- 1
NETWORK[2,3] <- 1

# 4. pkc
NETWORK[2,4] <- 1
NETWORK[3,4] <- 1

# 5. pka
NETWORK[4,5] <- 1

# 6. jnk
NETWORK[4,6]  <- 1
NETWORK[5,6]  <- 1

# 7. p3B
NETWORK[4,7]  <- 1
NETWORK[5,7]  <- 1

# 8. raf
NETWORK[4,8]  <- 1
NETWORK[5,8]  <- 1

# 9. mek
NETWORK[4,9]  <- 1
NETWORK[5,9]  <- 1
NETWORK[8,9]  <- 1

# 10. erk
NETWORK[5,10]  <- 1
NETWORK[9,10]  <- 1
```

```
# 11. akt
NETWORK[1,11]   <- 1
NETWORK[5,11]   <- 1
NETWORK[10,11]   <- 1

return(NETWORK)
}




#
    ##############################################################################

###################### function for computing the AUROC value and plotting
    the ROC

library(limma)
library(ROC)

compute_AUROC <- function(postP, trueEdges, steps=0.01){

n1 <- numeric(0)
for(i in 1:dim(postP)[1]){
        for(j in 1:dim(postP)[2]){
                                    if (abs(i-j)>0){
                                    n1 <- c(n1, postP[i,j])
                                    }
                                    }
                           }

n2 <- numeric(0)
for(i in 1:dim(trueEdges)[1]){
       for(j in 1:dim(trueEdges)[2]){
                                if (abs(i-j)>0){
                                n2 <- c(n2, trueEdges[i,j])
                                        }
                                }
                           }


sp <- sort(n1)          # posterior probabilities
st <- n2[order(n1)]     # true ordered by posterior probabilities

rocc.obj <- rocdemo.sca(n2, n1, rule=NULL, cutpts=NA)#seq(0,1,steps))

return(auROC(st,sp))

}
```

```
#
    ##############################################################################

# Extrahiere das CPDAG des wahren Netwzerks

extract_cpdag_of_dag <- function(true_incidence){

L <- list()

nodes <- dim(true_incidence)[1]

k <- cpdag(true_incidence)


dummy <- matrix(numeric(nodes*nodes),nrow=nodes)

if(length(nrow(k))!=0){
    dummy[k[,1]] <- k[,5]
    L <- dummy
}

if(length(nrow(k))==0 && length(k)>0){
     dummy[k[1]] <- k[5]
     L <- dummy
}

   mat.com <- matrix(numeric(nodes*nodes),nrow=nodes)
   mat.re  <- matrix(numeric(nodes*nodes),nrow=nodes)

   com <- which(L>0)
   re  <- which(L<0)

   mat.com[com] <- 1
   mat.re[re] <- 1
   mat <- mat.com + mat.re + t(mat.re)

 return(mat)
}
```

Listing 1: HOI?

A.2 Code to perform the structure MCMC algorithm for sample size 10

```
#function that takes as input Data_2017 and randomises 10 obs and then runs
     structure MCMC and then takes edgesaveragethreshold as final
###eveytime change in the end number of edgesaveragethreshold,   3times

#generate random vector with 10 integers, ranging from 0 tot 2017
```

```
randomvec=floor(runif(10, min=1, max=2018))

#select 10 columns of the randomvec
Data_10obs=Data_2017obs[,randomvec]

#call it Data, because is used in algorithm
Data=Data_10obs

# no. of network nodes
n_nodes      <- nrow(Data)

# Total no. of MCMC iterations per run
#20000 default
iterations  <- 80000
# Thin-out by a factor of 100
step_save   <- 100

start_incidence <- matrix(0,n_nodes,n_nodes)

Sim_1 <- strMCMC(Data, start_incidence, iterations, step_save)
Sim_2 <- strMCMC(Data, start_incidence, iterations, step_save)
Sim_3 <- strMCMC(Data, start_incidence, iterations, step_save)

# The graphs:
Sim_1[[1]]

# The scores:
Sim_1[[2]]

# Exercise 3
n_1 <- length(Sim_1[[2]])
n_2 <- length(Sim_2[[2]])
n_3 <- length(Sim_3[[2]])

plot(1:n_1,Sim_1[[2]], type="l", ylab="log P(D|G)", xlab="graphs", main="
    Trace Plots")
lines(1:n_2,Sim_2[[2]], col="blue")
lines(1:n_3,Sim_3[[2]], col="red")
cpdags_1 <- cpdag_list(Sim_1[[1]],100)
cpdags_2 <- cpdag_list(Sim_2[[1]],100)
cpdags_3 <- cpdag_list(Sim_3[[1]],100)
edges_mat_1 <- cpdags_1[[3]]
edges_mat_2 <- cpdags_2[[3]]
edges_mat_3 <- cpdags_3[[3]]
edges_mat_1
edges_mat_2
edges_mat_3

# Arrange the marginal edge posterior probability matrices as vectors:
```

```
edges_vec_1 <- numeric(0)
edges_vec_2 <- numeric(0)
edges_vec_3 <- numeric(0)

for (i in 1:n_nodes){
        for (j in 1:n_nodes){
                        if (abs(i-j)>0){
                                edges_vec_1 <- c(edges_vec_1,edges_mat_1[i,
                                    j])
                                edges_vec_2 <- c(edges_vec_2,edges_mat_2[i,
                                    j])
                                edges_vec_3 <- c(edges_vec_3,edges_mat_3[i,
                                    j])
                                }
                         }
                       }

# PLOT THE RESULTS

par(mfrow=c(2,2))

plot(edges_vec_1,edges_vec_2,pch=4, xlab="Run 1", ylab="Run 2", main="1
    versus 2")
lines(0:1,0:1, type='l', col="red")

plot(edges_vec_2,edges_vec_3,pch=4, xlab="Run 2", ylab="Run 3", main="2
    versus 3")
lines(0:1,0:1, type='l', col="red")

plot(edges_vec_1,edges_vec_3,pch=4, xlab="Run 1", ylab="Run 3", main="1
    versus 3" )
lines(0:1,0:1, type='l', col="red")

#########################################################################

#function that takes average of three matrixes and then takes threshold

#take average marginal edge posterior of the three runs
#edgesaverage=(edges_mat_1+edges_mat_2+edges_mat_3)*(1/3)

#take threshold
#edgesaveragethreshold10=threshold(edgesaverage,0.55)
data10sample10_1=edges_mat_1
data10sample10_2=edges_mat_2
data10sample10_3=edges_mat_3

#save in .csv file
write.table(data10sample10_1, file="data10sample10_1.csv", row.names=F, sep
    =",")
write.table(data10sample10_2, file="data10sample10_2.csv", row.names=F, sep
```

```
    =",")
write.table(data10sample10_3, file="data10sample10_3.csv", row.names=F, sep
    =",")
```

Listing 2: HOI?

A.3 Script for scatterplots different data10samples

```
#script that makes scatter plots for different data10samples.
#choose if takes average of 3 best or chooses just one
#I have chosen that it takes average of them all
#function that takes average of three matrixes and then takes threshold

#take average marginal edge posterior of the three runs of
#a specific data10sample

averagedata10sample1=(data10sample1_1+data10sample1_2+data10sample1_3)*(1/
    3)
averagedata10sample2=(data10sample2_1+data10sample2_2+data10sample2_3)*(1/
    3)
averagedata10sample3=(data10sample3_1+data10sample3_2+data10sample3_3)*(1/
    3)
averagedata10sample4=(data10sample4_1+data10sample4_2+data10sample4_3)*(1/
    3)
averagedata10sample5=(data10sample5_1+data10sample5_2+data10sample5_3)*(1/
    3)
averagedata10sample6=(data10sample6_1+data10sample6_2+data10sample6_3)*(1/
    3)
averagedata10sample7=(data10sample7_1+data10sample7_2+data10sample7_3)*(1/
    3)
averagedata10sample8=(data10sample8_1+data10sample8_2+data10sample8_3)*(1/
    3)
averagedata10sample9=(data10sample9_1+data10sample9_2+data10sample9_3)*(1/
    3)
averagedata10sample10=(data10sample10_1+data10sample10_2+data10sample10_3)*
    (1/3)

# Arrange the averagedata100sample matrices as vectors:

avdata10_vec_1 <- numeric(0)
avdata10_vec_2 <- numeric(0)
avdata10_vec_3 <- numeric(0)
avdata10_vec_4 <- numeric(0)
avdata10_vec_5 <- numeric(0)
avdata10_vec_6 <- numeric(0)
avdata10_vec_7 <- numeric(0)
avdata10_vec_8 <- numeric(0)
avdata10_vec_9 <- numeric(0)
avdata10_vec_10 <- numeric(0)
```

```
for (i in 1:n_nodes){
  for (j in 1:n_nodes){
    if (abs(i-j)>0){
          avdata10_vec_1 <- c(avdata10_vec_1,averagedata10sample1[i,j])
          avdata10_vec_2 <- c(avdata10_vec_2,averagedata10sample2[i,j])
          avdata10_vec_3 <- c(avdata10_vec_3,averagedata10sample3[i,j])
          avdata10_vec_4 <- c(avdata10_vec_4,averagedata10sample4[i,j])
          avdata10_vec_5 <- c(avdata10_vec_5,averagedata10sample5[i,j])
          avdata10_vec_6 <- c(avdata10_vec_6,averagedata10sample6[i,j])
          avdata10_vec_7 <- c(avdata10_vec_7,averagedata10sample7[i,j])
          avdata10_vec_8 <- c(avdata10_vec_8,averagedata10sample8[i,j])
          avdata10_vec_9 <- c(avdata10_vec_9,averagedata10sample9[i,j])
          avdata10_vec_10 <- c(avdata10_vec_10,averagedata10sample10[i,j])
                                }
                            }
                          }

##################

my_list <- list(avdata10_vec_1, avdata10_vec_2,avdata10_vec_3,avdata10_vec_
    4,avdata10_vec_5,avdata10_vec_6,avdata10_vec_7,avdata10_vec_8,avdata10_
    vec_9,avdata10_vec_10)

#names(my_list) <- c("avdata10_vec_1", "avdata10_vec_2","avdata10_vec_3" ,"
    avdata10_vec_4","avdata10_vec_5","avdata10_vec_6","avdata10_vec_7","
    avdata10_vec_8","avdata10_vec_9","avdata10_vec_10")

######################
# PLOT THE RESULTS
par(mfrow=c(2,2))
for (i in 1:9){
  a=as.numeric(i)+1
  for (j in as.numeric(a):n_nodes){
    plot(my_list[[i]],my_list[[j]],pch=4, xlab="Run 1", ylab="Run 2", main=
        paste("data10sample;",i,"versus", j))
lines(0:1,0:1, type='l', col="red")
                                }
                            }
```

Listing 3: HOI?

A.4 Script for scatterplots different sample sizes

```
#avavdata10_vec is average of 10 runs of sample10
avavdata10_vec=(avdata10_vec_1+avdata10_vec_2+avdata10_vec_3+avdata10_vec_
    4+avdata10_vec_5+avdata10_vec_6+avdata10_vec_7+avdata10_vec_8+avdata10_
    vec_9+avdata10_vec_10)*(1/10)
```

```
 #save in .csv file
 write.table(avavdata10_vec, file="avavdata10_vec.csv", row.names=F, sep=",
     ")

#avdata100_vec is average of 10 runs of sample100
avavdata100_vec=(avdata100_vec_1+avdata100_vec_2+avdata100_vec_3+avdata100_
    vec_4+avdata100_vec_5+avdata100_vec_6+avdata100_vec_7+avdata100_vec_8+
    avdata100_vec_9+avdata100_vec_10)*(1/10)

 #save in .csv file
 write.table(avavdata100_vec, file="avavdata100_vec.csv", row.names=F, sep=
     ",")


#avdata500_vec is average of 10 runs of sample500
avavdata500_vec=(avdata500_vec_1+avdata500_vec_2+avdata500_vec_3+avdata500_
    vec_4+avdata500_vec_5+avdata500_vec_6+avdata500_vec_7+avdata500_vec_8+
    avdata500_vec_9+avdata500_vec_10)*(1/10)

 #save in .csv file
 write.table(avavdata500_vec, file="avavdata500_vec.csv", row.names=F, sep=
     ",")




#
    ##############################################################################

avavdata10_vec=c(avavdata10_vec)
avavdata100_vec=c(avavdata100_vec)
avavdata500_vec=c(avavdata500_vec)



#########################################################################
#plots
par(mfrow=c(2,2))

plot(avavdata10_vec[[1]],avavdata100_vec[[1]],pch=4, xlab="Run 1", ylab="
    Run 2", main="avavdata;10 versus 100")
lines(0:1,0:1, type='l', col="red")

plot(avavdata10_vec[[1]],avavdata500_vec[[1]],pch=4, xlab="Run 2", ylab="
    Run 3", main="avavdata;10 versus 500")
lines(0:1,0:1, type='l', col="red")

plot(avavdata100_vec[[1]],avavdata500_vec[[1]],pch=4, xlab="Run 1", ylab="
    Run 3", main="avavdata;100 versus 500")
```

```
lines(0:1,0:1, type='l', col="red")
```

Listing 4: HOI?

B Spotify distributions

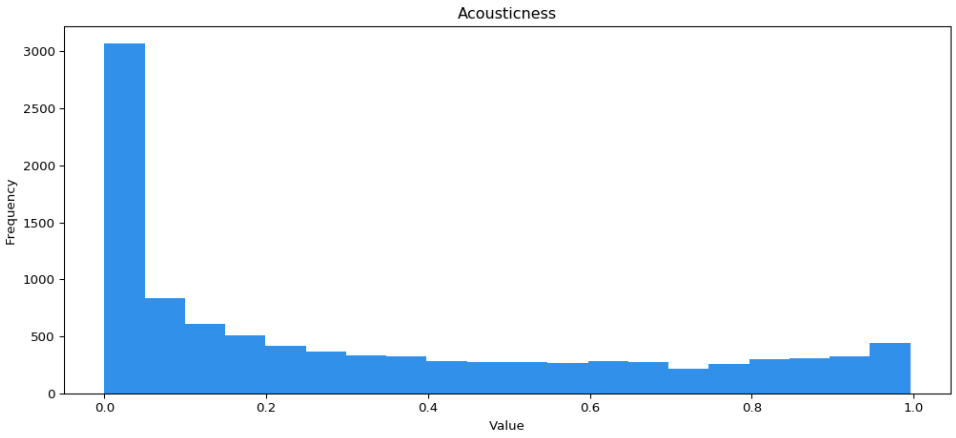

Figure 25: Distribution acousticness downloaded from Spotify

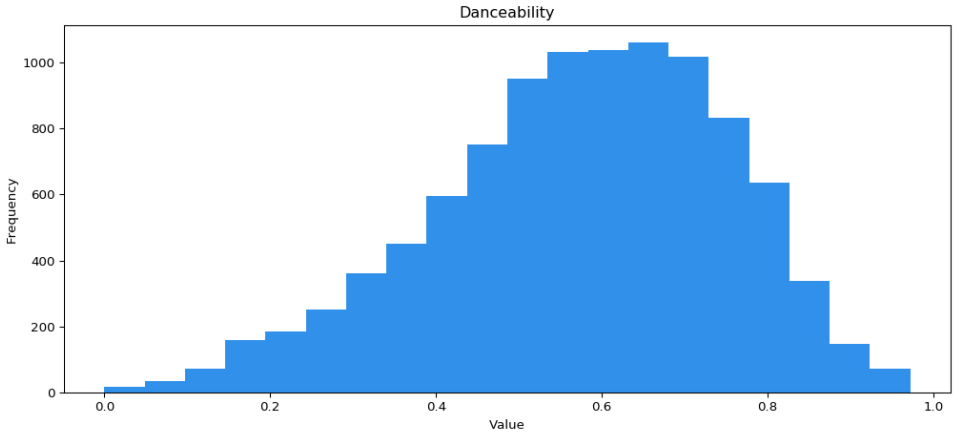

Figure 26: Distribution danceability downloaded from Spotify

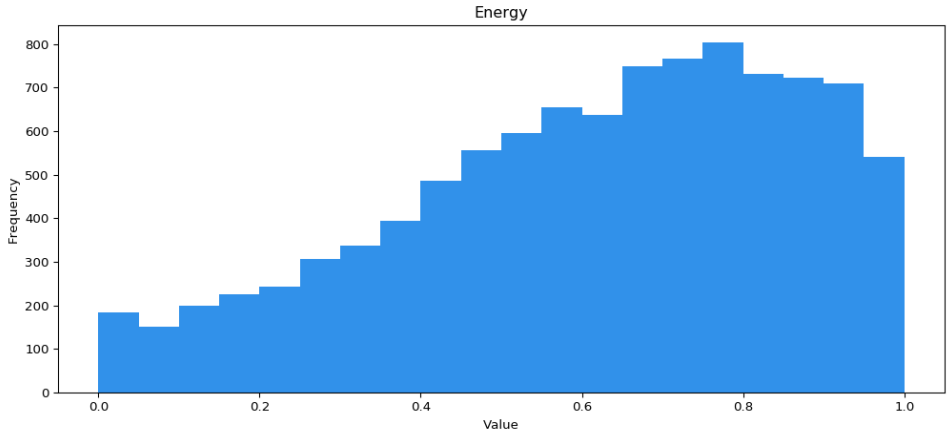

Figure 27: Distribution energy downloaded from Spotify

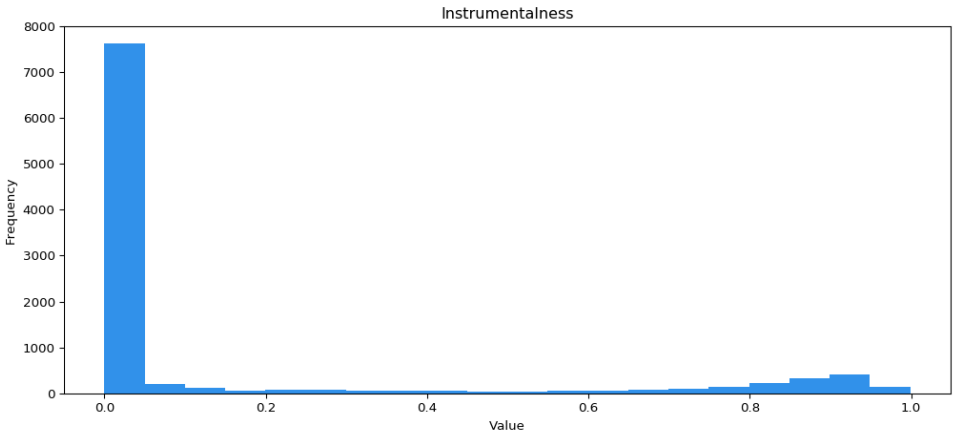

Figure 28: Distribution instrumentalness downloaded from Spotify

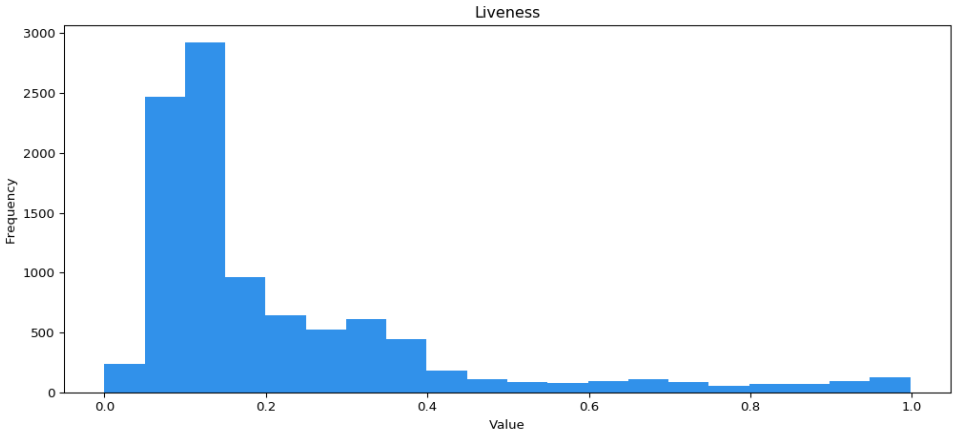

Figure 29: Distribution liveness downloaded from Spotify

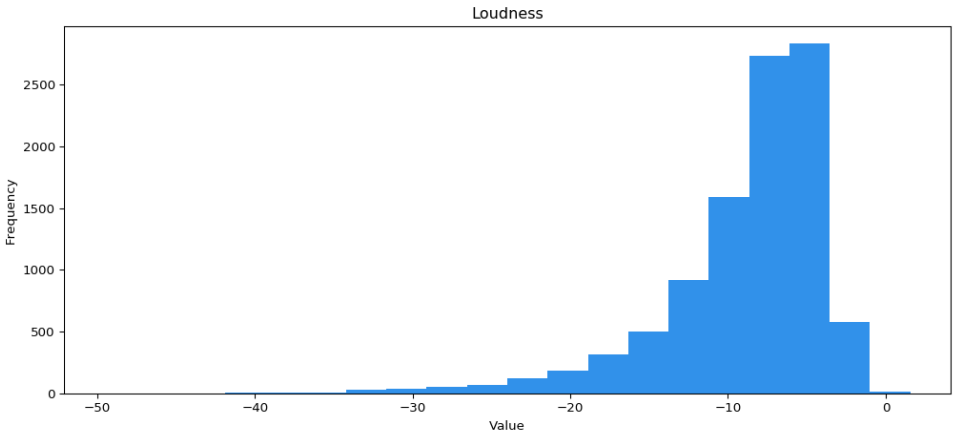

Figure 30: Distribution loudness downloaded from Spotify

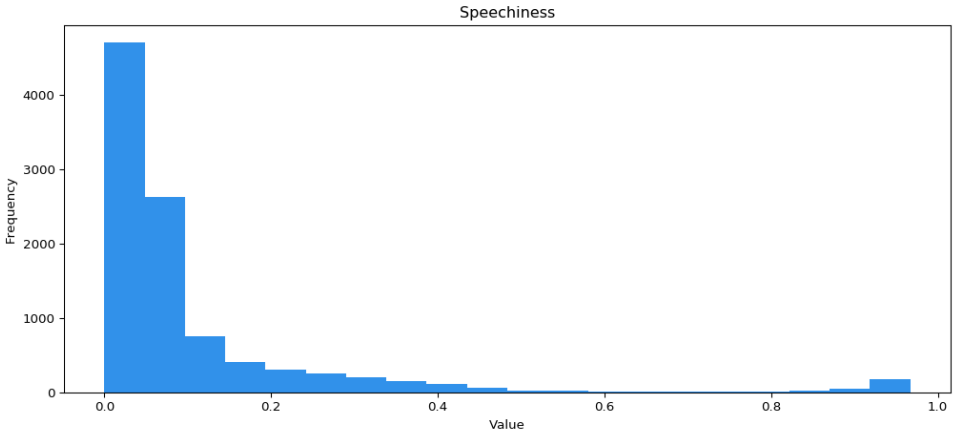

Figure 31: Distribution speechiness downloaded from Spotify

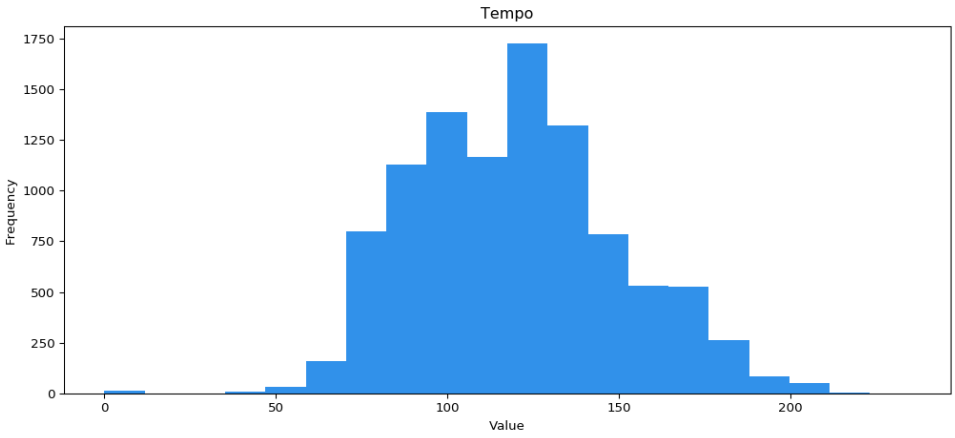

Figure 32: Distribution tempo downloaded from Spotify

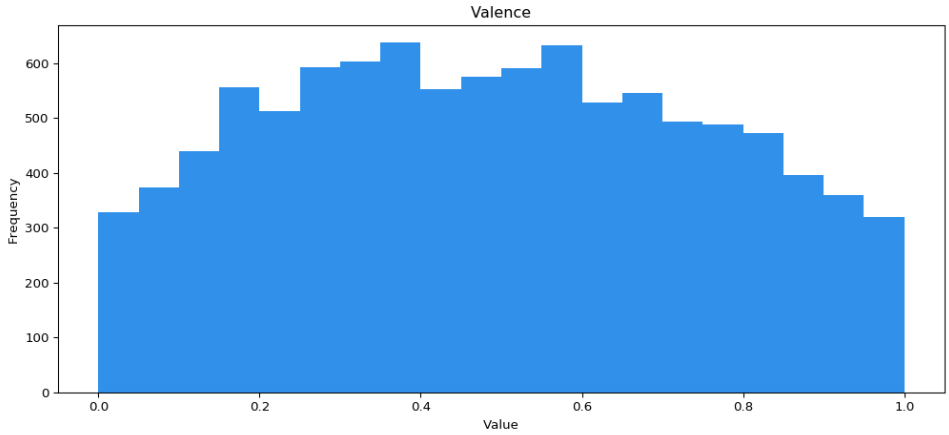

Figure 33: Distribution valence downloaded from Spotify