Scientifically approved prediction methods

J.Corazza*, Cs.Dobi** and D.Skoko***
* V. gimnazija, Zagreb, Croatia
** Dobó István Gimnázium, Eger, Hungary
***NOVA International Schools Gymnasium, Skopje, Macedonia
yannbane@gmail.com
d.cseni13@gmail.com
dominik.skoko@gmail.com

Abstract - In order to make a prediction tool for music taste, three machine learning methods were utilized: decision tree, random forest, and an implementation of a genetic algorithm. The program was made in Python, and as the dataset, it received 30 music samples from the training set. It could then be tested with a separate test set which consisted of 19 samples, and its performance was evaluated. Guitar chords of different songs were processed, and, at first, only the numbers of individual chords was extracted. That proved ineffective, so a more complex analyzing method had to be used. That method involved detecting progressions in the song. The three different algorithms proved very effective in detecting whether someone will like a particular song, and random forest performed the best.

I. INTRODUCTION

The growing necessity to process large amounts of data has led to a wide array of research in the field of machine learning and artificial intelligence. AI is a branch of computer science that deals with implementing the traits of human intelligence to a machine. The algorithms can be applied to many different cases, and their main characteristics involve dynamic alteration of some criteria, making them adaptive, and essentially granting them the ability to learn. The learning is achieved via iterative presentation of new data samples, and can be divided into three categories: supervised, unsupervised, and reinforcement learning.

Machine learning is a branch of artificial intelligence that deals with algorithms that allow the machine to create behaviors based on datasets from the database (observations from previous examples).

Music has always been a part of human life. Each individual has his own particular taste in music. But, is there some kind of a pattern? Is machine learning suitable for this kind of task?

The goal of our project was to make a classification tool that could be used as a music sample - classifier. More precisely, it would be able to tell whether someone likes a particular song, before that person actually hears it, based off an already existing dataset of songs that person likes and dislikes, that exists in the program's database. During the classification process, and after it, the program will modify itself to include the new music sample in its database, thus making it more efficient and precise.

A. *Classification*

Machine learning deals with two types of problems – regression and classification. Regression is a type of a problem which tries to fit a function to some points in a dataset. Classification is the problem of categorizing objects depending on their characteristics.

An algorithm that implements classification is known as a *classifier*. The *classifier* is a mathematical function that maps characteristics of objects to a specific category. In the terminology of machine learning, classification is an example of supervised learning in which the algorithm is aware of the object's class.

II. METHODS

A. *Decision tree*

The machine learning algorithm used in this paper is a *decision tree*. It is a robust algorithm that is capable of processing vast amounts of data and solving classification problems of sorting dataset entries (in this case, songs) based on their attributes into two or more categories, called *classes*. These classes can be imagined as leaves at the end of the tree, and they represent the final result of the classification process. Only two classes were used, one for a positive result (subject should like the song), and one for the negative result (subject should not like the song). It is a supervised learning method, because the program is informed about these classes. The programming language that was used was Python, an interpretative and highly expressive tool often used in other machine learning projects.

The structure of the tree is very similar to its biological equivalent. It is constructed of many nodes, branches, and leaves (which are described above), and the decision

process goes from one *root node*, down the branches and other child nodes, and onto the leaves. The nodes represent attributes, while branches represent the values these attributes can obtain.

Two very important pieces of information that were worked with were entropy and information gain. Entropy is defined as

$$E = -p_+ \log_2 p_+ - p_- \log_2 p_-,$$

and information gain is defined as

$$I = Entropy(S) - \sum_{values \in S} \frac{|S_v|}{|S|} Entropy(S_v).$$

These two statistical terms were very useful because they were at the core of the decision tree construction process. Entropy was defined as a measure of data impurity, and information gain as the amount of useful knowledge one can obtain with a certain choice. Entropy of the end results (leaves, classes) was used, which showed how random they are, to derive the information gain of various attributes, which can be viewed as a "reduction in entropy". The acquired information gain was used to determine the root node, and the downwards structure of the tree.

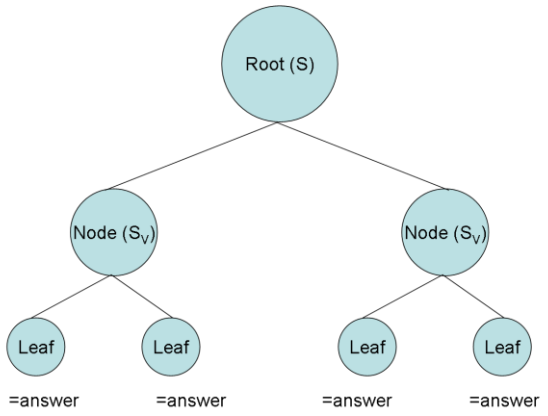

Figure 1 Example of decision tree

B. *Random forest*

A decision tree is not a perfect algorithm since it has quite a few drawbacks:

a) Generalization error is a type of overfitting. This error is common in decision trees, if only one type of data is trained, meaning there is a large amount of data in one class and a small amount of data in the other class. Since only one type of data is trained, only that type of data will be recognized by the decision tree.

b) Large amount of attributes causes the tree to be complex. Complex trees have less and less information gain as they branch out. This means that the decision becomes more and more inaccurate, and is based more on randomness than on decision making.

Ensemble learning is a set of machine learning methods that combines many different classifiers as it tries to solve the same problem. Then, using *democracy*, classifiers vote for the answer to the solution and the largest amount of votes wins.

Another algorithm that was looked into was the random forest algorithm. Just as the name suggests, it is a combination of many decision trees. What is interesting about it is that the dataset is not supplied as a whole, but rather, it is split into many parts. Each tree gets a random slice of the *attributes*, and is constructed based on that. Then the "votes" of each tree are counted, and the most voted class is chosen as the solution to the classification problem. This technique generally works much better on bigger datasets with a higher amount of attributes, so it was implemented in the program. Most of the code from the original algorithm could be used, only the functionality from dataset splicing was required. After splicing the data into chunks, already existing methods of creating trees and testing performance were used. Random forest algorithm is shown in figure 3.

1. Let the number of training cases be *N*, and the number of variables in the classifier be *M*.
2. We are told the number *m* of input variables to be used to determine the decision at a node of the tree; *m* should be much less than *M*.
3. Choose a training set for this tree by choosing *n* times with replacement from all *N* available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
4. For each node of the tree, randomly choose *m* variables on which to base the decision at that node. Calculate the best split based on these *m* variables in the training set.
5. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

Figure 3 Random forest algorithm

C. *Data set*

The starting dataset is composed of 30 songs, which are defined by their attributes (for example: percentages of chords in the song), and 20 of these songs belong to the positive class, while 10 of them belong to the negative one. We are given a basic description of the song and the person's opinion of it by the information in the dataset. After that, the initial decision tree is constructed, and the predictions can then begin. Any new songs are classified based on their attributes, and the decision tree is changed after the process to reflect its performance.

III. DATA PROCESSING

Obviously, there must be a separate decision tree for each person, as they have different musical preferences and so the starting dataset and the starting decision tree are different for each individual (there is always a chance that they supply the same 30 songs, though, in that case, the basic decision tree would be the same). The input to the program, as mentioned before,

I-IV-V	I-II-V	I-V-VI	I-II-VI	I-II-III	I-II	I-IV	I-V	IV-V	average	dominant	Like?
high	low	high	low	low	low	low	low	low	high	major	yes
low	low	low	low	low	medium	low	low	medium	medium	none	no

Figure 2. Dataset based on statistics of chords

was 20 positive and 10 negative songs. They were supplied in the form of textual files containing *guitar chords* and lyrics of the song. Before our program could use those files, they need to be correctly processed and turned into a data type that Python can manipulate. The processing was done by the parsers, and involved removing lyrics and isolating the guitar chords. These guitar chords were processed manually and then they were fed to the main algorithm, which constructed a decision tree.

Two different approaches were used to process the data. The first approach calculated the percentages of each chord in a song. However, this method proved unsuccessful. Just measuring the percentages of chords isn't precise enough to define a song, and there were simply too many different chords for a small dataset. Although this method failed, few interesting features were noticed. Some chord sequences occurred more often than others.

The following progressions have been frequently used: I-IV-V, I-II-V, I-V-VI, I-II-VI, I-II-III, I-II, I-IV, I-V, IV-V. Chords in a progression are marked relatively to a scale.

A more complex way of describing songs had to be chosen: progressions. Progressions are frequent occurrences of some chords in the same order, and many songs are based off the same progressions. If there were a way for these progressions to be noticed in the song, the very root of musical preferences could be analyzed. The problem is, analyzing just one aspect of the dataset might not be a suitable way of handling the statistics, so certain elements were added, such as the *average*, which can describe the span of chords, and the *dominant*, which shows whether there are more major or minor chords.

All attributes, except *dominant*, can adopt three different values. They can adopt *low* as a value if that progression occurs in less than 10 percent of the song, value of *medium* if it represents between 10 and 20 percent of all the progressions in the song and value of *high* if it is found in more than 20 percent of the song. An exception is the *average* attribute, since it represents, in percentage, the abundance of each chord found in that song. The *dominant* attribute corresponds to a type of chord which is used more often, the *major* chord or the *minor* chord.

IV. RESULTS

Two datasets have been used, one to train the algorithms, and another one to test them. The accuracy was calculated by dividing the number of correctly classified examples with the total number of examples in the dataset. The single decision tree displayed an accuracy of 78.9%. It was a satisfying result, as anything more than 50% is not random, and is relatively precise. The random forest algorithm proved to be even more successful. It performed with an accuracy of 89.5%.

A. *Rule extraction*

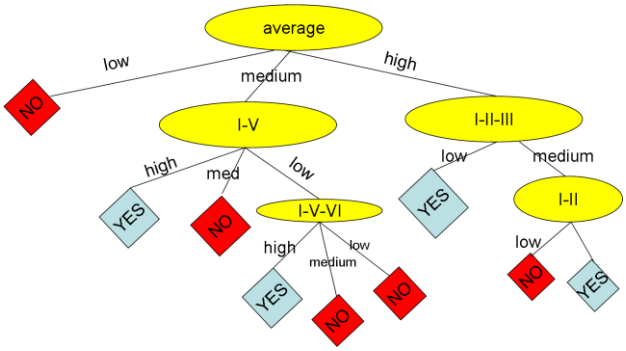

Figure 3 Example of built tree

Once a tree is built, it is easy to extract the rules which represent the data set best. Finding a rule from built decision tree is done by following the way in a decision tree from root to one of the leaves. Few rules found by this research are listed in the table below:

Table 1 Examples of extracted rules

IF	THEN
average = *low*	Like = *no*
average = *medium* **AND** I-V = *high*	Like = *yes*
average = *high* **AND** I-II-III = *medium* **AND** I-II = *low*	Like = *no*
average = *high* **AND**	Like = *yes*

I-II-III = *low*	

Table 2 Test results

Sample	Actual	Predictions		
		Decision tree	Random forest	Genetic algorithm
S1	yes	yes	yes	yes
S2	yes	yes	yes	yes
S3	yes	yes	yes	yes
S4	yes	yes	no	yes
S5	yes	yes	yes	yes
S6	yes	no	yes	yes
S7	yes	yes	yes	yes
S8	yes	no	yes	no
S9	yes	yes	yes	yes
S10	yes	no	yes	no
S11	yes	yes	no	yes
S12	yes	yes	yes	yes
S13	no	no	no	no
S14	no	no	no	no
S15	no	no	no	no
S16	no	no	no	no
S17	no	yes	no	no
S18	no	no	no	no
S19	no	no	no	no
Mistakes:		4	2	2

B. *Evaluation techniques*

For precise performance measure *precision* and *recall* parameters have been used. Predicted results are separated in four classes illustrated on figure 4.

	actual class (expectation)	
predicted class (observation)	**tp** (true positive) Correct result	**fp** (false positive) Unexpected result
	fn (false negative) Missing result	**tn** (true negative) Correct absence of result

Figure 4 Result classification

Having that parameters calculated, precision and recall are defined as:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

In this way, precision represents a fraction of retrieved results that are relevant to the search, and recall represents a fraction of retrieved results that are successfully retrieved. Results are represented in table X.

Table 3 Precision and recall measures

	Decision tree	Random forest	Genetic algorithm
True positive	9	10	10
True negative	1	0	0
False positive	6	7	7
False negatives	3	2	2
Precision	0.9	1	1
Recall	0.75	0.83	0.83

V. DISCUSSION

These results were quite satisfying, although there is much room for improvements, since not all factors were taken into consideration. For example, a more complex way of analyzing songs could have been used, instead of just progressions of guitar chords. Maybe, detecting different melodies and other patterns might have improved the performance, but that was out of scope of the project. The single decision tree had the worst accuracy out of all the algorithms that were used, but that was normal, as it is much simpler and much less adaptable. The random forest split the dataset into manageable chunks and constructed many partial decision trees that processed the chunks, so it was logically much better. According to Breiman, founder of this method, the explanation of improvement is that random forest performs better if separate trees, and attributes in training set, are slightly less correlated.

A. *Different Approach*

After using a random forest instead of a decision tree for improvement. It was decided that a genetic algorithm should be used in order to increase precision. A genetic algorithm is one of the many nature-inspired machine learning techniques, and it is different in that it doesn't involve only one instance of an algorithm (being evolved), but rather a population of many algorithms. The name “genetic algorithm” obviously implies that there is some evolution involved; after all, evolution is one of the most useful ways of determining some parameters. The algorithm usually creates many instances of parameters

for another algorithm, and then applies natural selection on the population of algorithms and chooses the best ones. This process continues and the population, on average, starts performing better. After a certain number of iterations, the best set of parameters in the population is chosen. In this particular case, random forests were the algorithms being evolved (or rather, their parameters were).

While it was an upgrade of the random forest algorithm, the genetic algorithm seemed as a good way to improve the performance of the music sample-classifier program. However, the results showed differently. The program with the genetic algorithm performed at the same level of accuracy as random forest one, with an accuracy of 89.5%. One explanation to why the program was not more accurate could be, that the genetic algorithms usually require richer chromosomes (sets of parameters) to be evolved, but our program contained only two genes (parameters). In addition, the dataset was too small, so a less complex algorithm could construct a better tree or forest which proved to be more efficient. Nonetheless, it was a useful addition to our tool chain, and proved to be good enough.

VI. CONLUSION

The program, created in this project was sufficient for a prediction program, which predicts if we would like future's music. We have concluded that it is not easy task, to solve, it is hard approximate differences, but we can still mine some basic correlations in the way of a genre. We have found that some progressions used more often in the specific genre. To conclude, our program offers 95% of accuracy prediction and not just a good recommender tool, but a fine entertainer.

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression, “One of us (R. B. G.) thanks . . .” Instead, try “R. B. G. thanks”.

REFERENCES

[1] T.Mitchell, “Machine learning”, McGraw-Hill International Editions, pp. 52-80, 1997.

[2] E.Alpaydin, “Introduction to Machine learning”, 2nd ed., The MIT Press, 2010, pp.185-208.

[3] A.Liaw, M.Wiener “Classification and Regression by randomForest ,” in R News, vol. II, no. 3, 2002, pp. 18-22.

[4] X.Chen, M.Wang, H.Zhang, “The use of classification trees for bioinformatics,” in Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol I, Issue 1, 2011, pp.55-63

[5] L.Breiman, “Random Forests,” in Machine Learning, Kluwer Academic Publishers, vol 45, 2001, pp.5-32

[6] P.N. Tan, A. Gupta, G.Karypis and V.Kumar, “Introduction to Data Minig,” Addison-Wesley, 2005, pp145-205