Relationships between the timbre of songs and their year

PABLO BEN LESTÓN*, CRISTIAN PÉREZ GÓMEZ, and BRAIS PÉREZ VÁZQUEZ, Universidad de Vigo, España

In this paper we will be analysing the relationship between different timbre variables of the songs and their year of publication. To analyse this relationship we will use different techniques: ANN, kNN, SVM, Decision Trees and Ensemble Models. For this purpose we will perform 4 different approaches to the problem, comparing them to finally determine which is the best model for this type of problem and if it is a problem that is feasible to solve with these machine learning techniques.

CCS Concepts: • **Computing methodologies** → *Machine learning approaches.*

Additional Key Words and Phrases: Machine Learning, ANN, Decision Trees, kNN, SVM, Ensemble Models, Cross-Validation

ACM Reference Format:

Pablo Ben Lestón, Cristian Pérez Gómez, and Brais Pérez Vázquez. 2023. Relationships between the timbre of songs and their year. 1, 1 (December 2023), 37 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The main problem to solve in this paper is to develop, with different machine learning techniques, some methods to classify songs in their respective release years. For this purpose we are selecting the ***Million Song Dataset*** [3], a freely-available collection of audio features and metadata for a million contemporary popular music tracks. In order to have numerical data instead of song tracks we download the UCI dataset linked with this project [2] where each song is converted in 90 different values, 12 related with the timbre average and 78 related with timbre covariance.

1.1 Description of the Dataset

As we say in the previous section, our dataset is a collection of different characteristics of songs. More particularly we have **90 characteristics**, *12 related with the timbre average* and *78 related with timbre covariance* of *515345 different songs*. In fact, we have 91 characteristics because we have the target characteristic, ***the release year of the song***. The songs are not equally *distributed between 1922 and 2011*, with much more songs in the range from 1996 to 2010. It should be noted that this dataset could be expanded and updated, having the necessary software available [3], to obtain more songs up to the present day.

*Authors contributed equally to this research.

Authors' address: Pablo Ben Lestón, pablobenleston@gmail.com; Cristian Pérez Gómez, cpgomez18@esei.uvigo.es; Brais Pérez Vázquez, 14brais@gmail.com, Universidad de Vigo, Ourense, Galicia, España, 32004.

Manuscript submitted to ACM

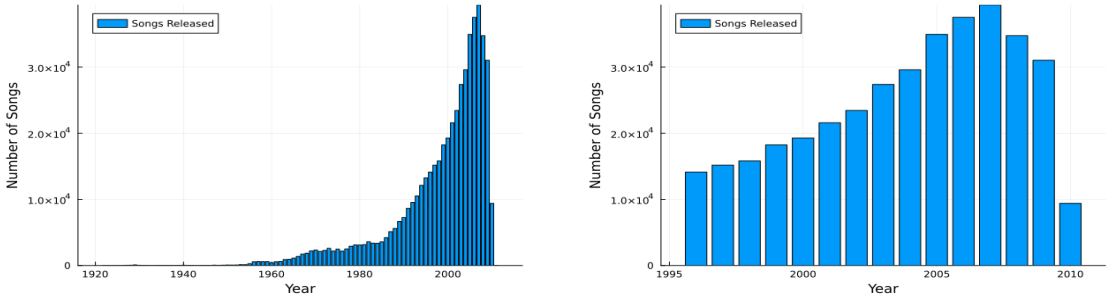

Fig. 1. Number of songs between 1922 and 2011 and between 1996 and 2010

Looking at the above graph we can clearly see the very uneven distribution of the songs, because of this, we will choose a subset of the original dataset. We will choose 15 years, between the year 1996 and the year 2010, both included (all years have at least 10000 songs except 2010 that have a little less). In addition, we will tackle the problem in 4 different approaches, which we will explain in more detail in each section, where, within each approach, we will carry out a more detailed treatment of the data. However, in all of them, we will normalise each characteristic, leaving them with mean 0 and standard deviation 1, with the aim of leaving a dataset with uniformly distributed characteristics.

1.2 Methodology

Within each approach, we will use the characteristics to get the year of release of the song. For this purpose we will use different methods; *ANN, SVM, kNN, Decision Trees and an ensemble model* using the above models. Later we will explain in detail the hyperparameters used in each model, however, in all of them we will use the same metrics, ***accuracy*** and ***weighted F1*** score. Where we define accuracy as the proportion of correct classifications among the total classifications made. In addition, the weighted F1-score is defined as the weighted average of the precision and recall [1], it is worth noting that being in a multiple classification problem the F1-score is defined as the weighted average of each of the classifications, i.e. we calculate the F1-score for each of the classes and we perform the weighted average of the F1-scores of each of the classes.

We use these metrics because the accuracy is a good metric to observe the general behaviour of the models, seeing how many hits it gets (which makes sense since our goal is to get as many hits as possible). However, as this is a multiple classification problem, it can happen that we get a very good classifier for certain class types and a bad classifier for other classes. To avoid this influencing the metric, we use the weighted F1-score, which takes into account the individual classifications within each class.

With respect to the code we have used a main program, *main.jl*, from which we load (and download if necessary) the necessary packages to carry out the functions we develop, for this we include two documents, *LoadPackages.jl* and *DownloadPackages.jl*. Inside this program we open the original dataset and finally we select the approach we want to work with. Within each approach we filter the data as needed and finally we select if we want to load the data from the models trained in this project or if we want to train them directly. It should be noted that we considered saving the filtered datasets in a .txt file to open it directly in case we wanted to use it, however, due to the fact that the data transformations are done in a very short time and because we can change the parameters in a much faster way by using

[1]We define ***precision*** as the ratio of positive patterns that have been correctly classified and ***recall*** as the probability that a positive classification result is obtained for a positive case. Notice that when we are talking about positive classification we are assuming that we are using one class against the others (positive if the attribute is from that class and negative if not).

the developed functions, we considered not saving it directly and doing the transformation each time it is used. Let's look at the different .jl developed:

- ***LoadPackages.jl*** and ***DownloadPackages.jl***: As already mentioned, they are used to open and download the necessary packages.
- ***LoadData.jl***: includes the functions with which we open and filter the data.
- ***Train_models.jl*** and ***Train_PCA.jl***: in these programs we developed the functions that allow us to select the models and train the methods (the first document for the first 3 approaches and the second one for the 4th one, in which we apply PCA). Note that we developed an option that not only allows you to train the models used in this project but also to define your own hyperparameters. In the second document we also slightly modified some training functions developed in the units to implement them with the PCA technique.
- ***LoadModels.jl***: in this document we developed a function that allows us to open the models we trained for this project and observe their metrics.
- ***FunctionsUnits.jl***: In this document we include the functions developed in the units.

Finally, it should be added that within the training functions the ouptuts of the models trained from the main.jl and the targets are saved in a ModelsTrained folder in the form of a .jld2 document. In the same way, there are 4 folders, one per approach where the data used for this project are stored, which can be opened, as we have already said, with the *LoadModels.jl* function.

1.3 Bibliographic analysis

This is a dataset widely used in different papers. Proof of this is that even the author of the dataset *T.Bertin-Mahieux* has a web page where he explains the dataset and works with it training different models [1]. It is also used in several published papers, some of which are explained below:

(1) ***Anytime Stochastic Gradient Descent: A Time to Hear from all the Workers*** [4]: The paper discusses a method for making machine learning problems run faster. Specifically, it focuses on a common approach called *synchronous stochastic gradient descent (SGD)* but addresses a problem where some parts of the computer are slower than others, causing delays. Unlike typical solutions that ignore the slow parts, the proposed method makes sure all parts contribute equally, resulting in much faster and more efficient computations, as demonstrated through improved performance in numerical tests. For this method comparation the paper uses the *Million Song Dataset*.

(2) ***Closed Form Variational Objectives For Bayesian Neural Networks with a Single Hidden Layer*** [5]: The paper explores a method for efficiently calculating probabilities in Bayesian neural networks with specific configurations, like single-layer networks using piecewise polynomial activation functions (e.g., ReLU). It demonstrates that for certain cases, a mathematical formula can be used to estimate uncertainty and predictions, aiding both training efficiency and quick predictions during testing. The approach is illustrated through experiments, showcasing its potential benefits in practical applications.

(3) ***Controversy Rules — Discovering Regions Where Classifiers (Dis-)Agree Exceptionally*** [6]: The paper introduces an algorithm, based on the Exceptional Model Mining framework, for identifying areas of disagreement among different classifiers. By applying this method to various public datasets (one of them, the *Million Song Dataset*), the study demonstrates its utility in classification tasks, uncovering valuable insights that challenge assumptions and reveal previously unknown phenomena.

2 FIRST APPROACH

In this first approach we will use the *dataset with 15 classes*, i.e. with 15 different years, from 1996 to 2010 inclusive. *For each class we will include 5000 songs*. We do not include more songs because otherwise the computational cost of training the models would be very high (with 5000 songs per year some models take 4 hours to train). Thus, we have a dataset ***with 75000 songs divided into 15 classes***, where each song has ***90 attributes*** (12 related with the timbre average and 78 related with timbre covariance).

We will use the 90 attributes of each song, however, we will apply a ***normalisation*** to convert them into attributes with ***mean 0 and standard deviation 1***. It is necessary to apply this normalisation because they are attributes with very different values, so when introducing them into the model it can lead to results that are not desirable. This can happen because if one value is greater than another it may have a greater weight in the model than one that has a lower value. In addition, we use the normalisation with mean 0 and standard deviation 1 for the same reason, because if we use the minima and maxima in the normalisation it could happen that one attribute is much smaller than another.

When using the different training models, we will apply ***Crossvalidation to all models with 10 folds***, calculating the mean and standard deviation of the metrics for all 10 folds. That is, we will divide the dataset into 10 equal subsets, choosing 1 of them to be used as test and the other 9 for training (repeating this process until all 10 subsets are used as test). In case we need a validation set (for ANN models), we will use a *HoldOut function where we will select 20% of the training set* (which in turn is 90% of the original dataset). In addition, to make the results repeatable, we will use *Random.seed!* in all models. In order to avoid having to train the models every time we work with the models, we will save the model outputs and the targets that should be obtained in a specific folder for each model.

As already stated and justified in the introduction we will use the ***accuracy*** and the ***weighted F1-score*** as metrics for the models. In addition we will compute (manually) the ***training time of each model*** (in minutes). To compute the training time we will perform the experiments under the same circumstances trying to keep external factors as small as possible.

2.1 ANN

The first machine learning method we will use is *artificial neural networks (ANN)*, with one or two hidden layers. As this is a multiclassification problem, we will have a last layer of the model with ***15 different outputs*** (one per year) and a ***softmax transfer function***. In addition, as is evident, the ANN will have ***90 inputs***, one per attribute of each song. Thus, we will train different ANNs, varying the number of hidden layers (one or two), varying the number of neurons in the hidden layers and varying the transfer function. As fixed hyperparameters we have that the *learningRate is 0.01*, the *Adam optimazer* and the *maximum number of epochs is 1000*. Furthermore, the training function does not allow overfitting to occur. Once we have all this in mind, let's see which ANNs are used for the problem, distinguishing between two cases, ANNs with one hidden layer and ANNs with two hidden layers.

One Hidden Layer. For the ANNs with one hidden layer we have the next models: 1 Hidden Layer with ***15*** neurons and a ***softmax*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***tanh*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***sigmoid*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***30*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***120*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***240*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***480*** neurons and a ***relu*** transfer function.

We first of all, try the more use transfer functions with a low number of neurons in the hidden layer and select the best model (all are very similar but the one with the relu transfer function is slightly better). Once we know that we increase the number of neurons of the hidden layer until we reach a model with a high execution cost (almost 2 hours to train the entire model) then we use all this models. We can see all the details of this models in the following table:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
15, softmax	0.165972	0.0043587	0.152187	0.004835	8 min
15, tanh	0.165972	0.0043587	0.152187	0.004835	8 min
15, sigmoid	0.165972	0.0043587	0.152187	0.004835	8 min
15, relu	0.165972	0.0043587	0.152187	0.004835	7.5 min
30, relu	0.195761	0.007689	0.184105	0.008244	8 min
120, relu	0.348891	0.012662	0.343507	0.013146	12 min
240, relu	0.550315	0.011722	0.548446	0.012338	30 min
480, relu	0.769748	0.012483	0.769506	0.012530	100 min

Table 1. Table with the metrics of the ANNs with a single hidden layer

What we can see here is that if we only have *15 neurons* in the hidden layer is *not enough* for such a complex model (15 classes). We can also see that by having relatively few neurons, compared to the attributes we have, the uncertainty is not large (*the standard deviation in all models and for both metrics is at most about 1 percent*). In addition, we can clearly see that the more neurons in the hidden layer, the better the results, however, as we have already said, it reaches the point that due to the complexity of the model it is not advisable to continue increasing the number of neurons. Another significant fact is that, although the difference between the accuracy and the F1-score is not very large, it is always smaller (as we predicted in the previous section) and the more reliable the model, the more similar they are.

Below we have a ***representation of the metrics for the above models***. In this graphic we represent the mean as a *dot*, in case of the *accuracy*, and as a *square*, for the *F1-score*, and above them a line representing the standard deviation. In this case the lines are barely visible because, as we said, the standard deviation is very low. In addition, we join the models with the transfer function relu, since this way we can observe the dependence between the number of neurons and the best behaviour of the model.

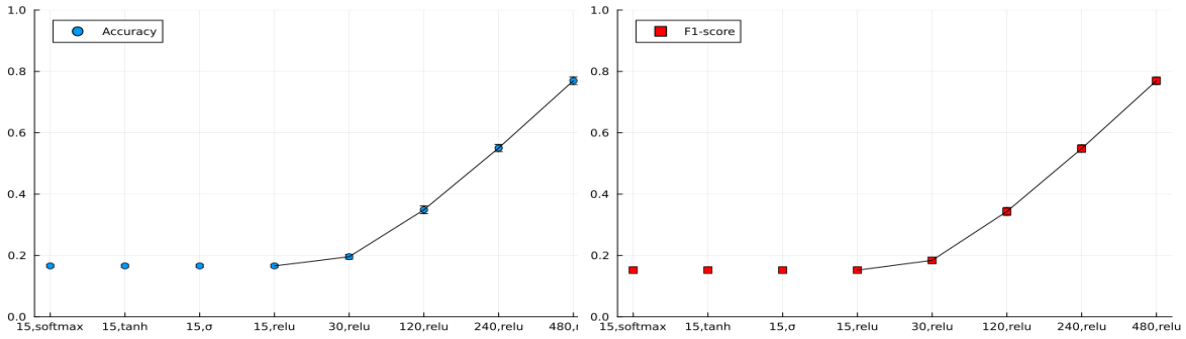

Fig. 2. Accuracy and F1-score for the ANN models with one hidden layer

Two Hidden Layers. For the ANNs with two hidden layer we have the next models: 1 Hidden Layer with ***(15,15)*** neurons and a ***softmax*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***tanh*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***sigmoid*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(120,120)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240,240)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(480,480)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one and 2 Hidden Layer with ***(480, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one.

The procedure we followed to choose these models was similar to the previous one, we tested a very simple model to compare the results, when we obtained similar results we used the relu transfer function (when we obtained good results in the previous case). We eliminated the model with two layers of 30 neurons because the difference was not significant. Finally we tried to mix two transfer functions to see if we got any different results. As in the previous case, below we have a table where we include all the results.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
(15,15), softmax	0.165584	0.006835	0.152011	0.004835	10 min
(15,15), tanh	0.165584	0.006835	0.152011	0.006881	10 min
(15,15), sigmoid	0.165584	0.006835	0.152011	0.006881	10 min
(15,15), relu	0.165584	0.006835	0.152011	0.006881	9.5 min
(120,120), relu	0.594427	0.008147	0.593142	0.008389	52 min
(240,240), relu	0.773548	0.014017	0.773360	0.014096	100 min
(480,480), relu	0.581056	0.100312	0.576640	0.104416	100 min
(240,240), sigmoid+relu	0.773548	0.014017	0.769506	0.014096	80 min
(480,240), sigmoid+relu	0.624050	0.068212	0.769506	0.071520	60 min

Table 2. Table with the metrics of the ANNs with two hidden layer

We can observe that the models with two hidden layers with 15 neurons do not give a good result. We can see that the model with two hidden layers of 120 neurons obtains similar results to the model with only one of 240 neurons (although the first one achieves it in more time). Something similar for the model with two hidden layers of 240 neurons and the model with one of 480 neurons, the execution time of these models are similar. However with the model with two layers of 480 neurons the results are much worse with a very large standard deviation (probably because of having too many neurons, i.e. too many parameters to train). Finally, the models with two transfer function together get good results, but as with the previous model, the one with 480 and 240 neurons gets worse results with a large standard deviation. The same can be applied as we said in the single hidden layer models with respect to the accuracy being slightly higher than the F1-score. As a conclusion we can say that both for results and execution time ***the best model is the model with 240 neurons in both layers***.

Similar to the single-layer models, we represent models with both metrics. Note how we can see the large standard deviation that some models have and how both metrics decrease in these models ((480,480), relu and (480,240), sigmoid+relu). Also note that in the axis we denote * as the transfer functions sigmoid + relu.

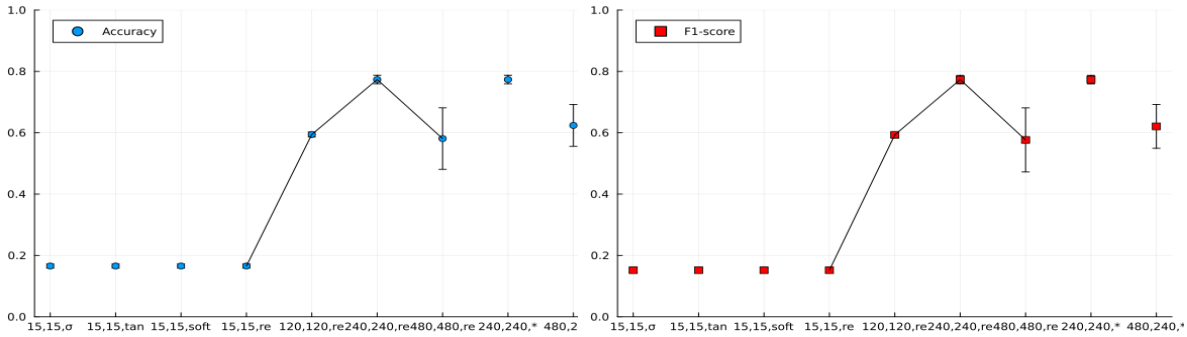

Fig. 3. Accuracy and F1-score for the ANN models with two hidden layers

As a final summary, we represent all models in terms of their F1-score and run time (in minutes). Note that the worst models are not shown because they are too close together. In addition, the model with a single hidden layer and 480 neurons is also not represented because it is approximately the same point as the model with two hidden layers with 240 neurons and the relu transfer function.

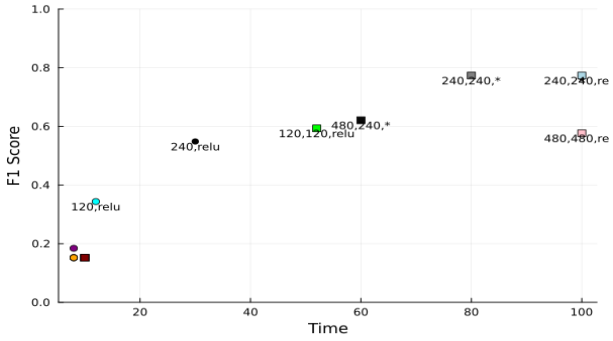

Fig. 4. Execution time in the ANNs

2.2 SVM

In this section we will use ***Support Vector Machines*** to try to solve our problem. We will use different models where we will vary the kernel and different hyperparameters. First we will use the simplest kernel, **linear**, then we will use the polynomial kernel varying the degree of the **polynomial** (of degree 2, 3 and 5), we will also use an **rbf** [2] kernel and a **sigmoid** [3] kernel (in both we will vary the value of C in order to obtain better results, and for the rbf kernel we will also vary the value of gamma). The value of C when increasing it causes that the tolerance to the error is greater (which can lead to overfitting), due to obtain very poor results in the model with sigmoid kernel we increase it drastically to try to obtain better results. In the case of the rbf model we made also a very huge changfe in order to see if we can have different results from the model.

We will therefore test the following eight models: *SVM with linear kernel*, *SVM with polinomial kernel with a degree of 2, 3 and 5*, *SVM with rbf kernel* (with C equal to 1 and 1000 and gamma equal to 3 and 20) and *SVM with sigmoid kernel* (with C equal to 1 and 1000). In the following table we can see the metrics concerning these models:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
linear	0.146187	0.002141	0.112901	0.001748	240 min
poly 2	0.876506	0.002546	0.876757	0.002560	290 min
poly 3	0.880880	0.002383	0.881041	0.002395	50 min
poly 5	0.881226	0.002128	0.881396	0.002082	70 min
rbf 1 3	0.871013	0.002278	0.890210	0.001756	120 min
rbf 1000 20	0.870346	0.002469	0.889696	0.001901	120 min
sigmoid 1	0.069574	0.007380	0.055607	0.003977	50 min
sigmoid 1000	0.070334	0.007540	0.056309	0.004095	50 min

Table 3. Table with the metrics of the SVM

Looking at the table we can draw some conclusions. The first one is that, taking into account all metrics, the **best model** is clearly the **polynomial degree 3 model**, obtaining the best accuracy and f1-score (*approx. 88%*) with a small runtime (compared to the other models). In terms of results, the model with the *polynomial kernel of degree 5* achieves *similar results* but the execution time is slightly longer, as well as the model with the *polynomial kernel of degree 2*, where the metrics are similar but the *execution time is much longer*. In the case of the models with the *rbf kernel* the results of the metrics are also *very good (approx. 87%)* but the execution time is more than double that of the polynomial model of degree 3. Clearly *the models with the linear and sigmoid kernel are the worst*, reaching only 14% accuracy in one case and in the other two not even reaching 10% (similar results with the F1-score). The first is that t*he standard deviation is very low* in all the models, i.e., no matter which set we take in the crossvalidation, the result will be very similar, and the second interesting fact is that in *the models with a high accuracy and F1-score the first is slightly lower than the second*, contrary to what happened in the ANN models.

We will now plot the above models with the selected metrics in a similar way to ANNs, where we represent the accuracy as a circle, the F1-score as a square and above it a line representing the standard deviation. In addition, we will also see the representation of the F1-score with the execution times. With these representations we can clearly see that the model with the polynomial kernel of degree 3 is the best.

[2]RBF means ***radial basis function***, where the gamma parameter defines how far the influence of a single training example reaches, with low values meaning "far" and high values mean "close".
[3]The ***sigmoid*** kernel is the same function as in the transfer functions in the ANN models.

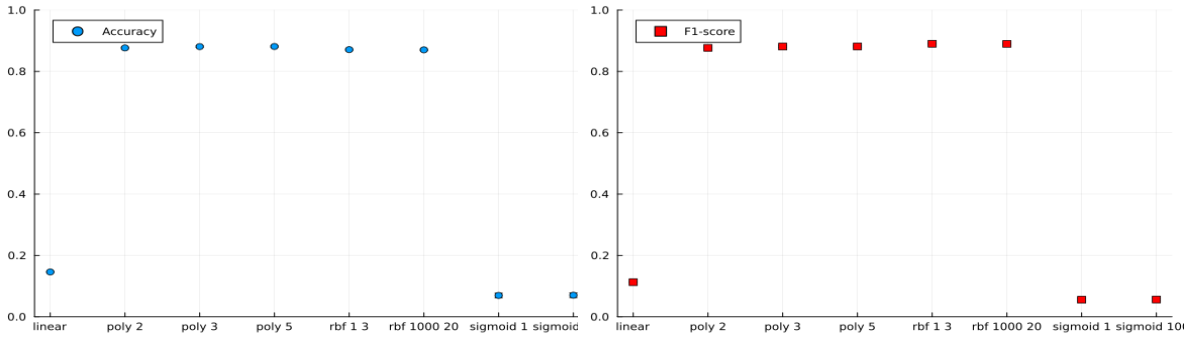

Fig. 5. Accuracy and F1-score for the SVM models

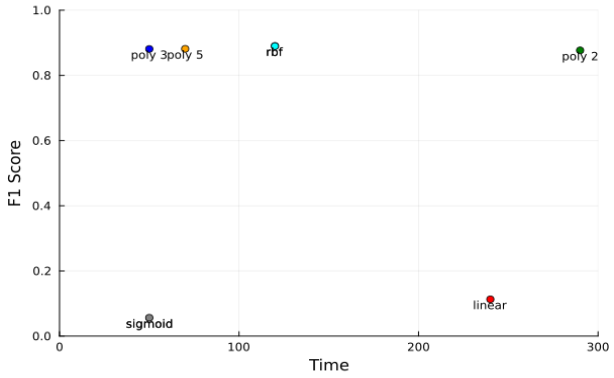

Fig. 6. Execution time in the SVM

2.3 kNN

The next models we will be using are the kNN (***k-nearest neighbours***), for which we will use six different models varying the value of k. First we will test with k=5 and then we will vary the k in order to find the best model. Therefore we are left with the following models: *kNN with k=1, kNN with k=2, kNN with k=3, kNN with k=5, kNN with k=10 and kNN with k=15*. In the next table we can see the metrics of the different models, we don't use the time cause in this models the execution time is despicable (*less than 5 minutes*).

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
neighbors 1	0.889413	0.004500	0.889408	0.004485
neighbors 2	0.806227	0.005153	0.806346	0.005016
neighbors 3	0.745081	0.004800	0.745279	0.004900
neighbors 5	0.618867	0.007415	0.619026	0.007485
neighbors 10	0.447773	0.007384	0.447765	0.007506
neighbors 15	0.392612	0.005891	0.392638	0.005859

Table 4. Table with the metrics of the kNNs

The higher the value of the hyperparameter k, the lower the metrics. With this we can clearly see that the best model is with ***k=1*** reaching an ***accuracy and F1-score of 89%***. Furthermore, we can also see that the standard deviation of the metrics is very small, it does not reach 1% in any model. Finally we are going to see the representation of the metrics of the models in two graphs similar to those used in the previous sections. To represent this dependence relationship between the value of k and the metrics we join all of them with a line. Clearly in the graphs we can see how the best model, by far, is the model with k=1.

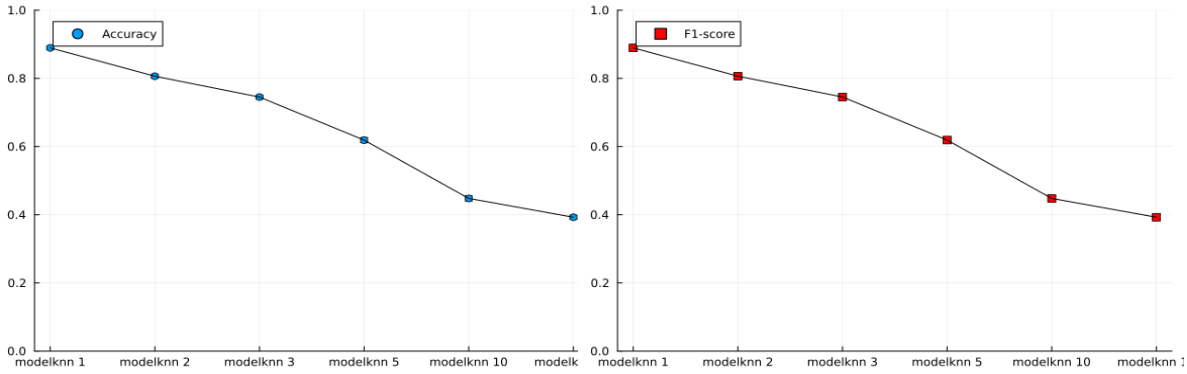

Fig. 7. Accuracy and F1-score for the kNN models

2.4 Decision Tree

The next method we will use will be ***Decision Tree,*** within which we will test 6 different models with varying depths. To adapt the depth of the model we started at a low value and gradually increased it to see how the model behaved, with this, the depths we have used are: *5, 20, 30, 40, 50 and 100.* In the following table we can see the results of the models we have tested, as in the kNN models as the execution times were very low we have not quantified them in the table.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
depth 5	0.121934	0.003025	0.084818	0.003500
depth 20	0.566120	0.012136	0.567053	0.012524
depth 30	0.826016	0.010468	0.826123	0.010506
depth 40	0.867146	0.003705	0.867118	0.003728
depth 50	0.869999	0.003280	0.869983	0.003286
depth 100	0.869946	0.003223	0.869927	0.003233

Table 5. Table with the metrics of the Decision Trees

Observing the results we can see the clear tr***end of increasing metrics as the depth value increases***, however, at a certain depth value the model does not achieve a higher accuracy or F1-score (*it stagnates at approximately 87%*). With this we can say that the optimum depth is 30, although in this case the execution time has no influence, it is true that the shallower the depth, the lower the computational cost. Finally, it should also be noted that *the standard deviation of these models is very low* (around 1%). Next, as with the rest of the models, we will see a representation of the models in similar graphs, again, we draw a line joining the different models representing the existing dependence between the depth and the depth and the metrics.

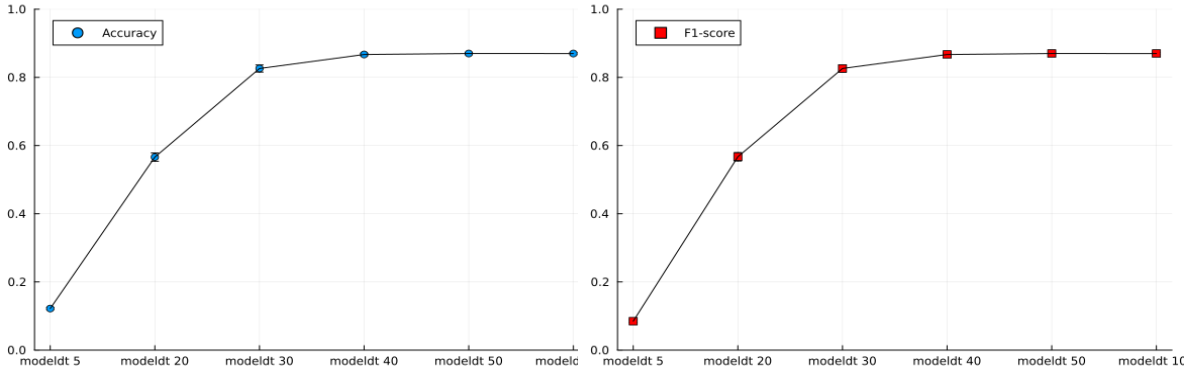

Fig. 8. Accuracy and F1-score for the Decision Trees models

2.5 Ensemble model

The last method we will use to solve this problem through this approach is an **ensemble model** of some of the best models we have come up with. For this we will be using the **Stacking method** of a ***kNN model with k=1 and two Decision Tree models with depth equal to 30.*** We have decided to use these models because they are clearly the best in terms of execution time and with respect to the metrics they are similar to the rest of the models. To compare the models we will put the best models of each method in a table: for the *ANN model* we chose the model with *two hidden layers of 240 neurons and a sigmoid and a relu transfer function*, for the *SVM model* we chose the *polynomial kernel of degree 3*, for the *kNN* we chose *k=1* and for the *Decision Trees* we chose a depth of *50*. As we have already mentioned, the kNN and Decision Trees models have an execution time of less than 5 minutes, but here we represent them with that execution time.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
ANN (240,240) sigmoid+relu	0.773548	0.014017	0.769506	0.014096	80 min
SVM Poly 3	0.880880	0.002383	0.881041	0.002395	50 min
kNN Neighbor 1	0.889413	0.004500	0.889408	0.004485	5 min
Decision Tree depth 50	0.869999	0.003280	0.869983	0.003286	5 min
Ensemble	0.888933	0.003093	0.888953	0.003113	45 min

Table 6. Table with the best models

Looking at the results we have obtained we can see that clearly **the worst model is the ANN** where it has the *lowest metrics* (although acceptable at around 77%) and the *highest execution time* (over 80 minutes). We can also highlight that the ***two best models*** in general are the ***kNN*** and the ***Decision Tree*** as they achieve very good metrics (over 89% in the first case and over 87% in the second). If we do not take into account the execution time, the most reliable model is the Ensemble model as it achieves 89% accuracy and F1-score with the second lowest standard deviation.

In addition, in order to be able to interpret the results more comfortably, we include the same graphs used in the previous sections, where we plot the mean and standard deviation of the two metrics used and where we compare the running time and F1-score of the models.

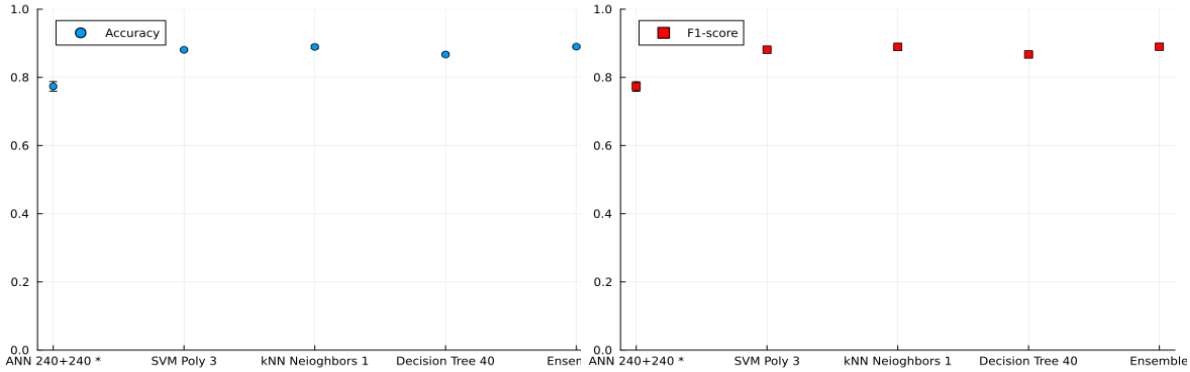

Fig. 9. Accuracy and F1-score for the Ensemble model and the best models

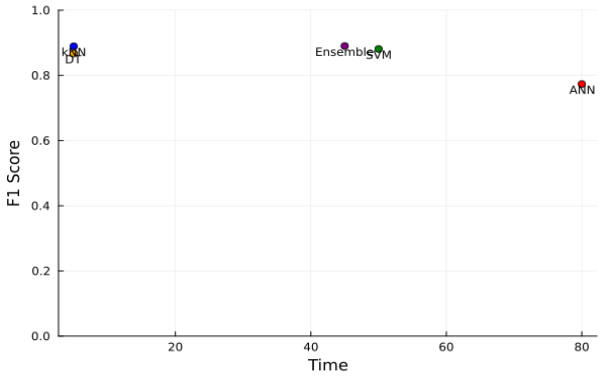

Fig. 10. Execution time of the best models

As a conclusion to this first approach we can say that ***we achieved quite good results in terms of metrics*** (around 90% accuracy and F1-score in some models). Furthermore, we can state that the worst models to perform this problem with this approach are ANN and SVM. Finally, kNN models are the best in general, but if you are looking for a very accurate model regardless of the execution time, the ensemble model could be taken into account.

3 SECOND APPROACH

Next we present the ***second approach*** where, in principle, we will use the *same models as in the previous one* (to be able to compare them directly) but varying slightly the inputs of the model, i.e. the dataset we will use. In this case we will also use songs from *15 different years*, as in the first approach, however when obtaining the different characteristics of the songs we will simply use those with respect to the timbre mean, that is, *the first 12 variables*. As a consequence of this decrease of the characteristics we will increase the number of songs per year by 2500, i.e. 7500 songs per year. In summary, we will have a dataset of **112500 songs** evenly distributed over **15 years**, with **12 characteristics** each.

As in the previous approach we will use a *normalisation* to convert the attributes into attributes of *zero mean and standard deviation 1*. As we justified in the previous section (*see in [2]*), this normalisation is necessary so that each of the characteristics have the same weight within the methods. In addition, as in the first approach, we will use *cross validation to train the model with 10 folds*. In addition, for ANN models we will use a *validation set* (20% of the training set calculated through the HoldOut function). We will also apply a *Random.seed!* to all models to make the results repeatable. In addition, in order to avoid having to train the models every time we work with the models, we will save the model outputs and the targets that should be obtained in a specific folder for each model. Finally, we will again use the same metrics: **accuracy**, **F1-score** and **execution time** (in minutes).

3.1 ANN

The first method we will use is ANNs, with one or two hidden layers. By not changing the number of classes in the dataset we will still have a model with 15 different outputs (15 neurons in the last layer) with a softmax transfer function. In this case the input of the model will be a layer with 12 neurons, due to the 12 characteristics of this model. The rest of the hyperparameters of the model will remain the same as the previous approach, and the models we will train will also be the same in order to be able to compare them directly.

One Hidden Layer. For the ANNs with one hidden layer we have the next models: 1 Hidden Layer with ***15*** neurons and a ***softmax*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***tanh*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***sigmoid*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***30*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***120*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***240*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***480*** neurons and a ***relu*** transfer function. The reason for the choice of these models is already justified in the first approach (*see in* [2.1]). In the following table we can see the results of the metrics for ANNs with only one hidden layer:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
15, softmax	0.138249	0.004772	0.115260	0.003422	14 min
15, tanh	0.138249	0.004772	0.115260	0.003422	14 min
15, sigmoid	0.138249	0.004772	0.115260	0.003422	14 min
15, relu	0.138249	0.004772	0.115260	0.003422	13 min
30, relu	0.153163	0.002738	0.137351	0.002340	17 min
120, relu	0.208604	0.004063	0.201690	0.003947	39 min
240, relu	0.260808	0.004836	0.256994	0.004416	67 min
480, relu	0.327661	0.004681	0.324835	0.004560	120 min

Table 7. Table with the metrics of the ANNs with a single hidden layer

Clearly the most relevant of these models for this approach are the **very poor results** achieved, where the best model achieves a meagre 32% accuracy and F1-score with a much longer run time than the models in approach one (details in *table* [1]). Seeing the poor results that these models achieve, no further analysis is necessary as these models are not viable. In spite of this, and following the structure used so far, we will represent the models in the same graphs as the previous models in order to be able to interpret the results in a more graphical way. As in approach one, by using the same transfer function we can see a clear dependence between the metrics and the increase of neurons in the hidden layer, however, as we commented in this case it is less evident as it does not reach relatively good metrics.

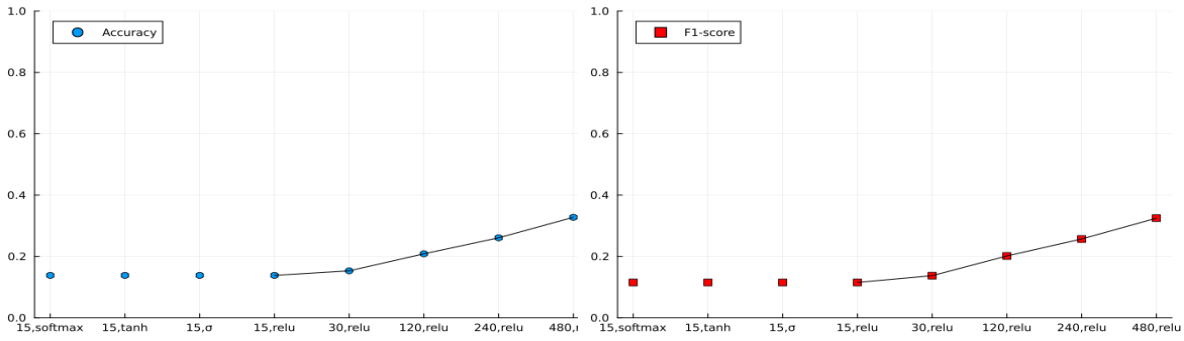

Fig. 11. Accuracy and F1-score for the ANN models with one hidden layer

Two Hidden Layer. As for the models with a hidden layer we will use the same models as in the first approach: 1 Hidden Layer with ***(15,15)*** neurons and a ***softmax*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***tanh*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***sigmoid*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(120,120)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240,240)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(480,480)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one and 2 Hidden Layer with ***(480, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
(15,15), softmax	0.143919	0.003064	0.123355	0.003401	20 min
(15,15), tanh	0.143919	0.003064	0.123355	0.003401	20 min
(15,15), sigmoid	0.143919	0.003064	0.123355	0.003401	20 min
(15,15), relu	0.143919	0.003064	0.123355	0.003401	20 min
(120,120), relu	0.365963	0.005523	0.362821	0.006393	90 min
(240,240), relu	0.486407	0.011994	0.485197	0.012271	125 min
(480,480), relu	0.337068	0.121217	0.325691	0.133647	135 min
(240,240), sigmoid+relu	0.486407	0.011994	0.485197	0.012271	240 min
(480,240), sigmoid+relu	0.388093	0.033253	0.385010	0.034524	185 min

Table 8. Table with the metrics of the ANNs with two hidden layer

Although the results are somewhat better than those of a single hidden layer (see in *table* [7]), ***they are not similar to those obtained in the first approach*** (see in *table* [2]). We obtain poor results, with a lower accuracy and F1-score than the first approach and a relatively slower execution time. The only models with a slightly good result are the model with two hidden layers of 240 neurons and a relu transfer function and the same model with relu+sigmoid transfer function, achieving metrics of approximately 48%, however the run time is 2 hours for the first one and 4 hours for the second one. It should also be noted that when a certain number of neurons is reached, the standard deviation of the model shoots up (as there are too many parameters and too few data to be able to adjust them and in each training session the data are different), in this case this occurs when there are two hidden layers of 480 neurons. Finally, let's look at the metric representations of the models and a representation of the F1-score with the execution times of each model, where it is very clear that none of these models are viable models to use [4].

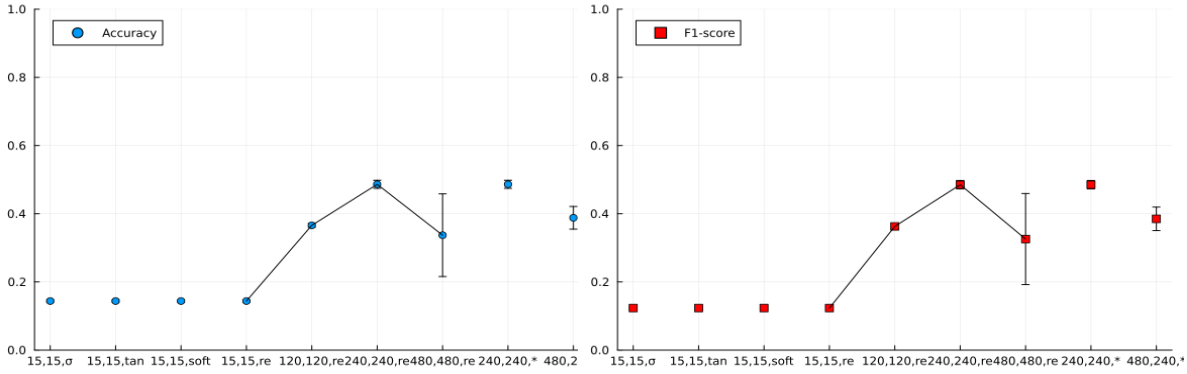

Fig. 12. Accuracy and F1-score for the ANN models with one hidden layer

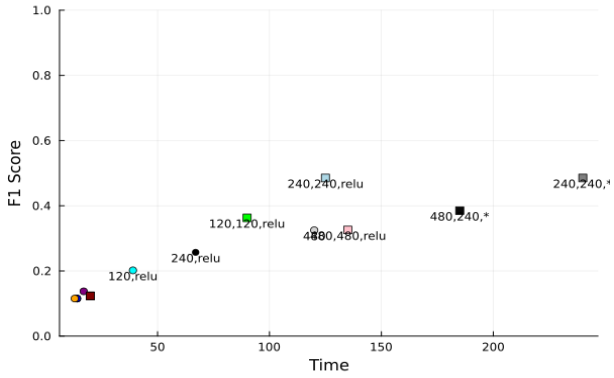

Fig. 13. Execution time in the ANNs

[4]For the worst models we don't put any label in order to have a more clean graph

3.2 SVM

The next method we will use is ***support vector machines*** SVM. For these models we tried to follow the same structure as for the first approach, however we could not achieve it because the models with the polynomial kernel had a very high execution time (more than 5 hours per model), because of this we use the models with a rbf, linear and sigmoid kernel. Therefore we used one model with the linear kernel, four models with the rbf kernel (the two already used in the previous model and two that are an intermediate point) and three models with the sigmoid kernel (two already used and one with an intermediate C value). In summary we will have the following models: SVM with a **linear kernel** with **C** equal to **1**, SVM with a **rbf kernel** with **C** equal to **1, 500 or 1000** and **gamma** equal to **1, 10 or 20** and SVM with a **sigmoid kernel** with C equal to **1, 500 and 1000**.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
linear	0.116427	0.001808	0.062318	0.001468	100 min
sigmoid 1	0.070506	0.005099	0.057475	0.002186	60 min
sigmoid 500	0.070372	0.004421	0.057746	0.002239	55 min
sigmoid 1000	0.066257	0.002092	0.055101	0.004675	50 min
rbf 1 3	0.935431	0.001784	0.939422	0.001559	100 min
rbf 1 20	0.932035	0.002998	0.939034	0.002476	100 min
rbf 500 10	0.932035	0.002727	0.938988	0.002268	110 min
rbf 1000 20	0.931688	0.002844	0.938765	0.002339	115 min

Table 9. Table with the metrics of the SVM

Considering the runtime problems with SVM models with polynomial kernel, the best models are the models with the ***rbf kernel***. These models obtain very good results, with *approximately 93% accuracy and F1-score*. It is true that these metrics are obtained with a very high execution time, almost two hours. Among the models with an rbf kernel, the best model could be the one with a gamma of 3 and c of 1 (i.e. a low gamma and low tolerance to error). Despite this, we can state that the parameter changes (the change of c for the models with rbf and sigmoid kernel and the change of gamma for the models with rbf kernel) hardly affect or are not significant, the biggest change is in the execution time.

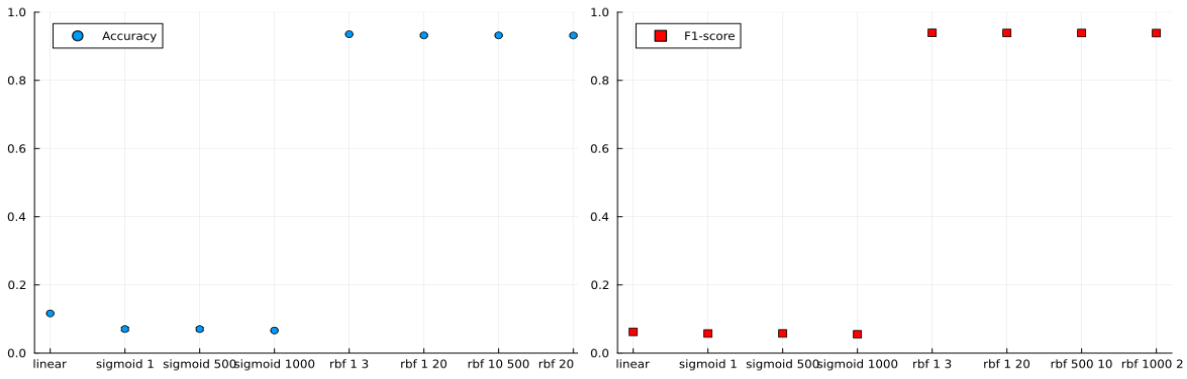

Fig. 14. Accuracy and F1-score for the SVM models

Looking at the metric graphs of the different models, as we did in the previous sections [5], we can see very clearly how ***the best models for the SVM method are the models with an rbf kernel***, although the execution times are worse, there is no comparison with respect to the metrics (9 times higher in the case of the rbf kernel). Finally, let's take a look at the graph where we represent the F1-score with the execution times, showing even more clearly the differences between one models and another.

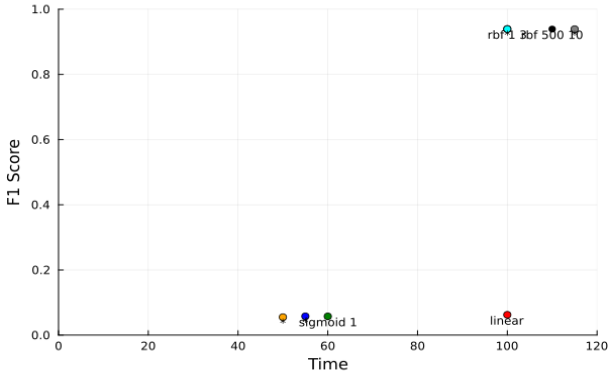

Fig. 15. Execution time in the SVM

3.3 kNN

The next method we will use is the ***kNN models***, varying the value of the k. As in the previous section the running time of these models is negligible so we will not take it into account and, again, in order to be able to compare between the different perspectives we are using we will use the same values of k used in the first approach (***k=1, 2, 3, 5, 10 and 15***). The following table shows the results obtained for each of the models:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
neighbors 1	0.937155	0.001555	0.937153	0.001562
neighbors 2	0.887529	0.002036	0.887608	0.002070
neighbors 3	0.847502	0.003031	0.847548	0.003044
neighbors 5	0.747369	0.004836	0.747370	0.004832
neighbors 10	0.522125	0.004644	0.521936	0.004796
neighbors 15	0.420045	0.005516	0.419779	0.005511

Table 10. Table with the metrics of the kNNs

As in the models of the first approach, ***the optimal k is k=1*** (results of the first approach in *table* [4]). However, in this case the results obtained are somewhat *better*, obtaining metrics **greater than 90%** (close to 94%), which can be understood as a model that is correct in the vast majority of cases. We can also observe that *the standard deviation is really small* (clearly less than 1% in all cases). Therefore, in this case, we can state that the models for this approach are clearly better than the previous ones, obtaining very favorable metrics. Let us also see the graphical representation of the metrics, as in the previous models we can see the *dependence relationship between the value of k and the metrics* (as the value of k increases, the metrics decrease).

[5] We put only one label for all sigmoid in order to have a clear graphic

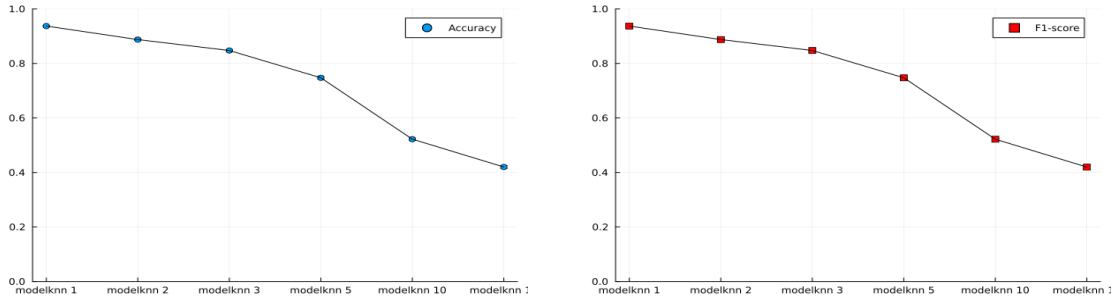

Fig. 16. Accuracy and F1-score for the kNN models

3.4 Decision Trees

The method we will use next is ***Decision trees***, varying the depth of the model. As in the previous ones we will use the same depth to be able to compare the results, that is, the *depth will be equal to 5, 20, 30, 40, 50 and 100*. Again, we will only use accuracy and F1-score as metrics since the runtime in these models is negligible. Let's see the results obtained in the models in the following table:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
depth 5	0.125484	0.002675	0.092859	0.003727
depth 20	0.631252	0.014982	0.632608	0.014689
depth 30	0.892063	0.012260	0.892146	0.012190
depth 40	0.929911	0.002168	0.929899	0.002164
depth 50	0.930826	0.001544	0.930815	0.001544
depth 100	0.930809	0.001563	0.930798	0.001561

Table 11. Table with the metrics of the Decision Trees

The most remarkable feature of the models is ***that from a depth of 30 the accuracy and F1-score increases to approximately 90%, even reaching 93% at a depth of 50***. Taking into account that, although in this case computational cost is not a problem, *the best model all things considered is the one with a depth of 30*. Similarly to the kNN, the results of *these models are better than the results obtained in the first approach* (both metrics were close to 87%, see in *table* [5]).

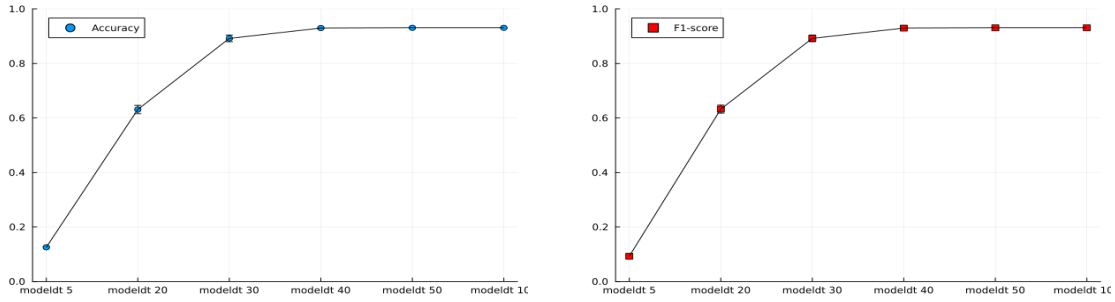

Fig. 17. Accuracy and F1-score for the Decision Trees models

We see the graphical representation (with the same graphs as before) of the models where we can see more clearly their high accuracy and F1-score. In addition, the dependence of the metrics on the depth of the model can be seen.

3.5 Ensemble model

For the last method, we will use an **ensemble model** with the **Stacking technique**. For this ensemble model we will use the same models as in the previous approach: ***a kNN model with k=1 and two Decision Trees with a depth of 30***, the justification for the selection of these models is the same as that given in the *section* 6 on approach one.

To compare the models we choose the best models of each of the methods: within the *ANN* we choose the model with two hidden layers and the relu transfer function, for the *SVM* we choose the model with the rbf kernel, C=1 and gamma equal to 3, for the *kNN* we choose the model with k=1 and for the *Decision Tree* we choose the model with depth 50 (we choose the model with the best metrics). We complete the table by putting a runtime for the kNN and Decision Trees of 5 min although as we have already said it is shorter. So let's look at the table comparing the best models of each method and the ensemble model:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
ANN (240,240), relu	0.486407	0.011994	0.485197	0.012271	125 min
SVM rbf 1 3	0.935431	0.001784	0.939422	0.001559	100 min
kNN Neighbor 1	0.937155	0.001555	0.937153	0.001562	5 min
Decision Tree depth 50	0.930826	0.001544	0.930815	0.001544	5 min
Ensemble	0.936293	0.001841	0.934613	0.002608	50 min

Table 12. Table with the best models

We can highlight two very relevant facts in the following table. The first of them is that all the ***ANN models*** (even taking the best of them), are ***much worse than the models of the rest of the methods***, if we observe the metrics, the metrics of the other methods obtain almost twice the results with much less standard deviation of those obtained by the ANN with two hidden layer of 240 neurons and the relu transfer function, if we observe the execution times we can see that the ANN is close to the SVM but much both with much difference with respect to the rest of the models.

Another of the most relevant data that we obtain by observing the table is that the ***best model is the kNN model with k=1***, obtaining *metrics close to 94%* and a really low execution time. It is true that the *Decision Tree could be a good model since the metrics are slightly lower and the execution time is very low.* As in the first approach, the *Ensemble model is a model that can be taken into account* if only effectiveness is sought and computational cost is not taken into account, an analogous reasoning could be extrapolated to the *SVM model.*

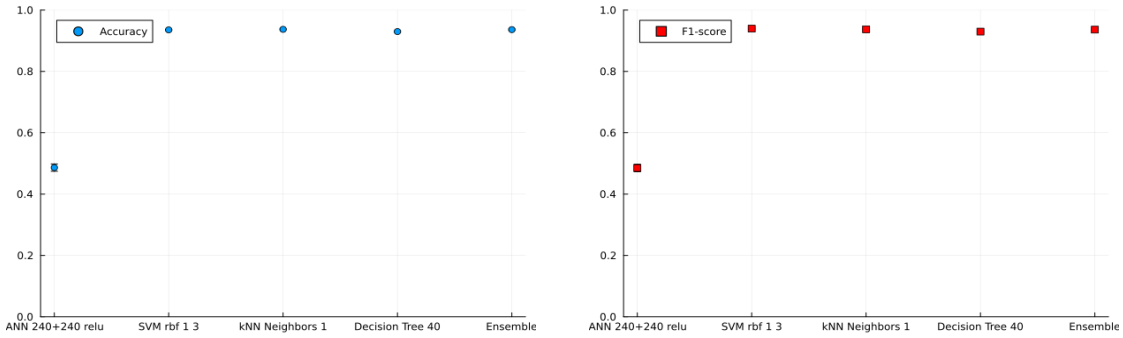

Fig. 18. Accuracy and F1-score for the Ensemble model and the best models

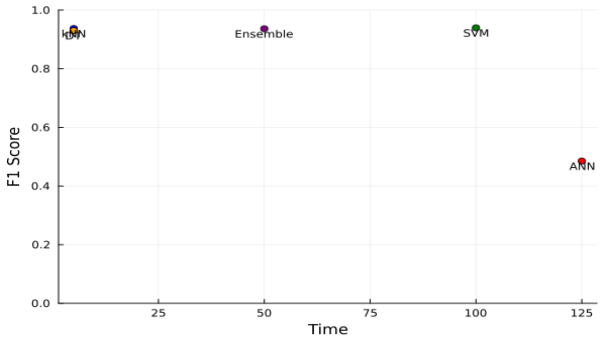

Fig. 19. Execution time of the best models

As a conclusion to this approach we can say that by treating the data in this way, manually reducing the dimensionality, we obtain better results in all the methods except in the ANN where we obtain worse metrics with a worse computational cost, this could be due to the fact that having less data the ANN parameters cannot be adjusted correctly to the desired results. Therefore, in general we can see that it is a better approach as we get very good results and we are dealing with a smaller number of data (we have the same number of instances but a smaller number of characteristics) [6].

[6]The Decision Tree model haven't label cause the kNN model have almost the same point

4 THIRD APPROACH

In this third approach we will try to use the same methodology as in the previous two approaches, trying to use always the same models in order to compare results and metrics. Again, in order to try to obtain better results, or at least to try to draw relevant conclusions about how our dataset works, we will slightly vary the inputs of the model. In this case we will continue to use songs from **15 different years** (the same years as previously used), and contrary to the previous approach we will use the same number of songs as in the first case, i.e. **5000 songs per year**. In this case we will use the remaining **78 features** of the previous model, the features that refer to the covariance of the timbre of the songs.

We will use again the *normalisation* of the characteristics to get them to have *mean 0 and standard deviation 1*, the justification is analogous to the one made in the first two approaches *section* [2] and *section* [3]. To train and test the models we will continue to use *Crossvalidation* with 10 folds, and for those models where a subset of validation is needed we will use the *HoldOut* function which will choose 20% of the training set to use. We will apply *Random.seed!* to all models in order to ensure that the results obtained are repeatable. Finally, to analyse all the methods we will use the metrics we have been using so far: **accuracy, F1-score and execution time**.

4.1 ANN

The first method we will use to solve the problem with this third approach is the ***ANN***, the characteristics and hyperparameters we will use will be the same as in the previous approaches (see in *section* [2]) and we discuss them below. It should be noted that, as in the previous models, these models will also have 15 outputs, but the inputs in this case will be 78 (the amount of characteristics that we have in account in the dataset).

One Hidden Layer. For the ANNs with one hidden layer we have the next models: 1 Hidden Layer with ***15*** neurons and a ***softmax*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***tanh*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***sigmoid*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***30*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***120*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***240*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***480*** neurons and a ***relu*** transfer function. The reason for the choice of these models is already justified in the first approach (*see in section* 2.1). In the following table we can see the results of the metrics for ANNs with only one hidden layer:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
15, softmax	0.149879	0.004876	0.134809	0.005329	10 min
15, tanh	0.149879	0.004876	0.134809	0.005329	10 min
15, sigmoid	0.149879	0.004876	0.134809	0.005329	10 min
15, relu	0.149879	0.004876	0.134809	0.005329	9 min
30, relu	0.174814	0.006337	0.162631	0.004838	12 min
120, relu	0.306373	0.009025	0.298348	0.010184	15 min
240, relu	0.481069	0.014155	0.477235	0.015036	40 min
480, relu	0.728399	0.006672	0.728032	0.006801	110 min

Table 13. Table with the metrics of the ANNs with a single hidden layer

Looking at the data obtained for ANN models with a single hidden layer, we can draw certain conclusions. The first and most obvious conclusion is that a*ll those models with 120 neurons or less obtain very poor results.* In the case of the model with 240 neurons, the results obtained are beginning to be decent (almost 50%), although with a relatively high standard deviation. **The best model** among these models, therefore, is the **model with 480 neurons in the hidden layer**. With this model we obtain more than 70% in both metrics with a very low standard deviation, the main drawback is that it is a model with a relatively high execution time, close to two hours. Let's see all these metrics in the graphics below, highlighting the dependency relationship between the increase of the metrics and the increase of the number of neurons (using the same transfer function).

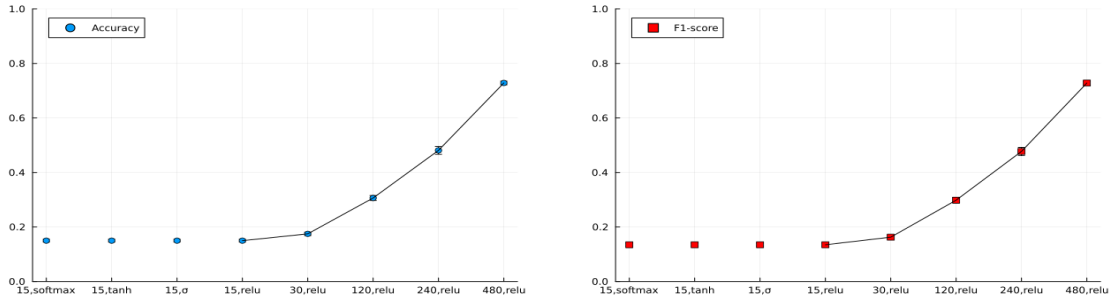

Fig. 20. Accuracy and F1-score for the ANN models with one hidden layer

Two Hidden Layer. As for the models with a hidden layer we will use the same models as in the first approach: 1 Hidden Layer with ***(15,15)*** neurons and a ***softmax*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***tanh*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***sigmoid*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(120,120)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240,240)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(480,480)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one and 2 Hidden Layer with ***(480, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
(15,15), softmax	0.153333	0.007600	0.137237	0.009427	12 min
(15,15), tanh	0.153333	0.007600	0.137237	0.009427	12 min
(15,15), sigmoid	0.153333	0.007600	0.137237	0.009427	12 min
(15,15), relu	0.153333	0.007600	0.137237	0.009427	10 min
(120,120), relu	0.549198	0.008611	0.547147	0.009370	60 min
(240,240), relu	0.735451	0.022754	0.735140	0.022780	110 min
(480,480), relu	0.584230	0.086891	0.580557	0.089436	120 min
(240,240), sigmoid+relu	0.738574	0.02010	0.738089	0.020276	115 min
(480,240), sigmoid+relu	0.554887	0.050035	0.549585	0.052205	65 min

Table 14. Table with the metrics of the ANNs with two hidden layer

With respect to the metrics of the ANN models with two hidden layers, we can highlight that, as expected, the models with only 15 neurons per layer obtain very poor results. The model with 120 neurons per layer obtains relatively good results. By far the two best models are the two models with ***two hidden layers with 240 neurons***, both the model with two relu transfer function and the model with one relu transfer function and one sigmoid transfer function obtain *metrics close to 74%* with a very low standard deviation, the biggest problem with these two models is their **long execution time**, almost two hours. On the other hand, the models that have some layers with 480 neurons obtain much worse results, moreover, the model that has both layers with 480 neurons has a very high standard deviation, almost 10%. Let's see below the representation of all these metrics of the models, where we can see very clearly the high standard deviation of some models and that the two models that we said were the best.

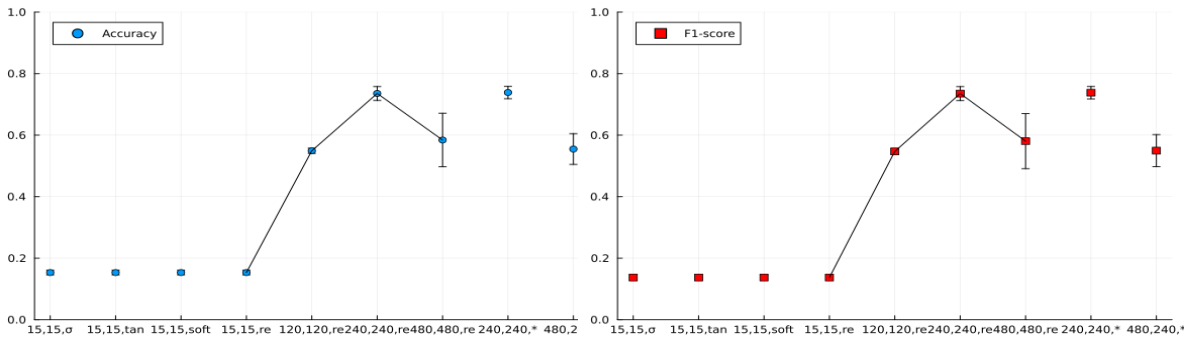

Fig. 21. Accuracy and F1-score for the ANN models with one hidden layer

Finally, in this image we can see all the previous models, represented according to their F1-score and execution time. This image makes clear what was said in the previous paragraph, the models with two hidden layers obtain the best results, although the execution time is quite high, which, seeing the trend of the previous approaches, could mean that the results are not good enough in comparison with other models.

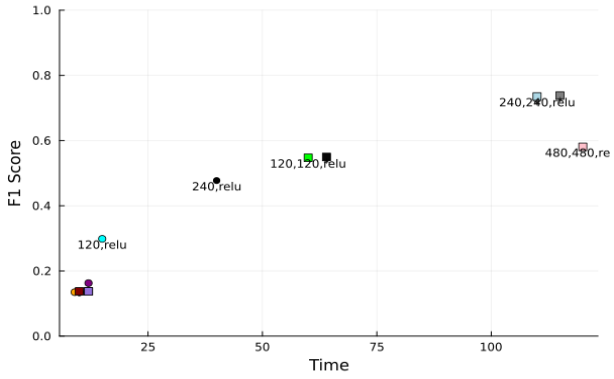

Fig. 22. Execution time in the ANNs

4.2 SVM

In this section we will work with the ***SVMs***, similar to the previous approach we tried to use the same models used in the first approach in order to compare them, however, we could only run one model with the polynomial kernel (we tried two others and they had too long run times). With this in mind, we ran eight different models, varying the hyperparameters (already explained in the first approach, see in *section* [2.2]), these models use these kernels: one with a ***linear kernel***, one with a ***polinomial kernel***, three with a ***rbf kernel*** and one with a ***sigmoid kernel***. In the models with the rbf kernel and the sigmoid kernel we are using different hyperparameters. For the rbf we use *c=1, 500 and 1000* and *gamma=1, 10 and 20*, for the sigmoid kernel we use *c=1, 500 and 1000*. Once we know all this, there is the table with all the metrics of the models, note that execution time is not in the table cause it is so low that it is negligible.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
linear	0.081629	0.001717	0.081629	0.001717	200 min
poly 3	0.877944	0.004320	0.877944	0.004320	60 min
rbf 1 3	0.892414	0.003514	0.892414	0.003514	65 min
rbf 500 10	0.890104	0.002258	0.890104	0.002258	70 min
rbf 1000 20	0.889990	0.002238	0.889990	0.002238	85 min
sigmoid 1	0.058084	0.004282	0.058084	0.004282	47 min
sigmoid 500	0.057838	0.002455	0.057838	0.002455	55 min
sigmoid 1000	0.057799	0.002285	0.057799	0.002285	65 min

Table 15. Table with the metrics of the SVM

Observing the results we can see that the models with the **linear kernel** and the **sigmoid kernel** have **very poor metrics**, not even reaching 10%, while the models with the **polynomial kernel** and the **rbf kernel** have **very good metrics**, close to 90%. Going into more detail with these two models we can see that the rbf models are slightly better, although the execution time is also slightly longer. Two other facts that we can highlight are that the change of parameters within the models hardly influences the result and that the results obtained are worse than those of the previous approach, in terms of metrics, but better in terms of execution time (see in *table* [3]). While with respect to the first approach they are similar, however, they have a shorter execution time (see in *table* [9]).

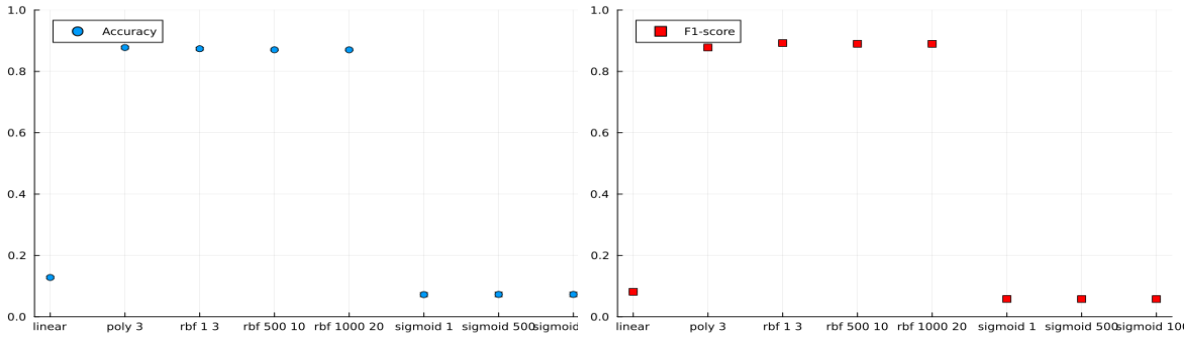

Fig. 23. Accuracy and F1-score for the SVM models

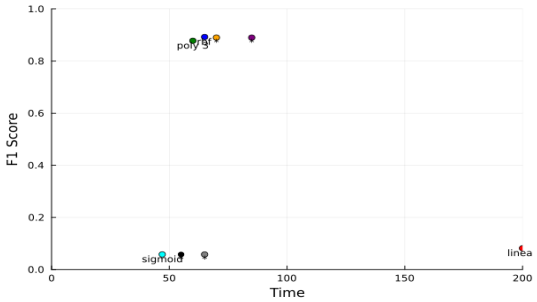

Fig. 24. Execution time in the SVM

4.3 kNN

The next method we will use is the ***kNN models***, varying the value of the k. As in the previous sections the running time of these models is negligible so we will not take it into account and, again, in order to be able to compare between the different perspectives we are using we will use the same values of k used in the first approach (***k=1, 2, 3, 5, 10, 15***).

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
neighbors 1	0.881971	0.004047	0.881963	0.004070
neighbors 2	0.792971	0.004813	0.793044	0.004749
neighbors 3	0.727546	0.004366	0.727843	0.004317
neighbors 5	0.593892	0.004283	0.594075	0.004256
neighbors 10	0.409852	0.003267	0.409864	0.003260
neighbors 15	0.350345	0.004299	0.349969	0.004124

Table 16. Table with the metrics of the kNNs

As usual in this type of model (within this problem), **the highest metrics are obtained with the value of k=1**. These metrics decrease inversely proportional to k (as we will see in the following graphs). The main interest of these models, therefore, is obtained in the results obtained from the model with k=1 and whether they are good or bad in comparison with the previous approaches. In this case both metrics have a value of approximately 88%, when for the previous approach we obtained metrics close to 94% (see in *table* [10]) and in the first approach the metrics were 89% (see in *table* [4]), so we can say that this is ***one of the worst kNN models we have trained so far***.

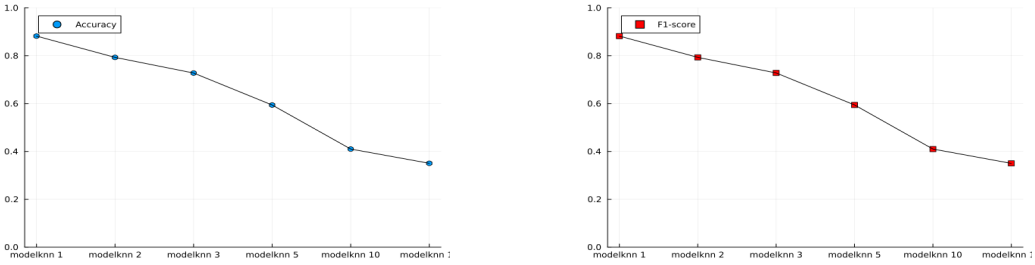

Fig. 25. Accuracy and F1-score for the kNN models

4.4 Decision Trees

The method we will use next is ***Decision trees***, varying the depth of the model. As in the previous ones we will use the same depth to be able to compare the results, that is, the *depth will be equal to* ***5, 20, 30, 40, 50, 100***. We will only use accuracy and F1-score as metrics since the runtime in these models is negligible.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
depth 5	0.118399	0.003170	0.072382	0.005603
depth 20	0.483787	0.014699	0.483558	0.015312
depth 30	0.792663	0.014383	0.793026	0.014103
depth 40	0.860956	0.008552	0.860945	0.008549
depth 50	0.868117	0.005244	0.868100	0.005236
depth 100	0.868451	0.005291	0.868437	0.005289

Table 17. Table with the metrics of the Decision Trees

As in the previous two sections, *as the depth of the model increases, we obtain better results*, starting with very poor results at a depth of 5 and reaching very good results **at a depth of 100, with metrics reaching almost 87%.** Although the general behaviour is similar to the two previous approaches, there are a couple of different nuances in this case. The first is that in this case, *the difference between the model with a depth of 30 and the model with a depth of 40 is much greater than in the previous cases* (almost 7% difference when in the previous cases it was less than 4%). Another relevant fact is that the results obtained are worse than those of the previous approach and very similar to the first one (see in *table* [5] and *table* [11]). Let's look at the graphical representation of both metrics and see the relationship of dependence of these metrics with the depth.

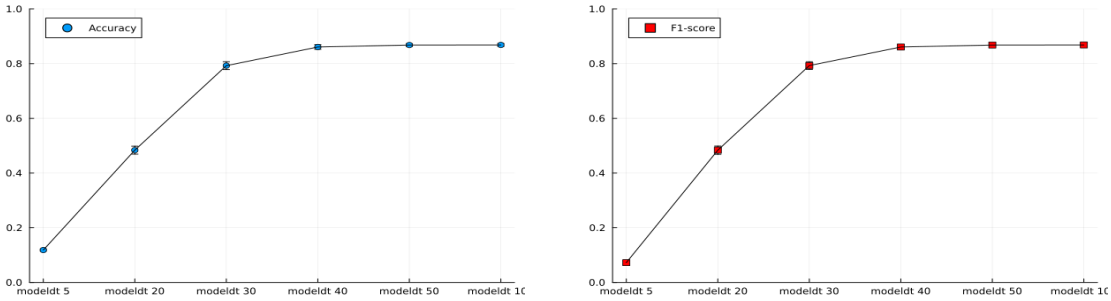

Fig. 26. Accuracy and F1-score for the Decision Trees models

4.5 Ensemble model

Finally, we will use an **ensemble model** with the **Stacking technique**. For this ensemble model we will use the same models as in the previous approach: ***a kNN model with k=1 and two Decision Trees with a depth of 30***, the justification for the selection of these models is the same as that given in the section 6 on approach one (in this case the decision tree with a depth of 30 is not the best model but we select it because the difference between this model and the best one is not too much and with this we can compare the approaches more directly).

To compare the models we choose the best models of each of the methods: within the **ANN** we choose the model with two hidden layers and the transfer function, for the **SVM** we choose the model with the rbf kernel, C=1 and gamma equal to 3, for the **kNN** we choose the model with k=1 and for the **Decision Tree** we choose the model with depth 50 (we choose the model with the best metrics). We complete the table by putting a runtime for the kNN and Decision Trees of 5 min although as we have already said it is shorter. So let's look at the table comparing the best models of each method and the ensemble model:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
ANN (240,240), relu	0.735451	0.022754	0.735140	0.022780	110 min
rbf 1 3	0.892414	0.003514	0.892414	0.003514	65 min
kNN Neighbor 1	0.881971	0.004047	0.881963	0.004070	5 min
Decision Tree depth 50	0.868117	0.005244	0.868100	0.005236	5 min
Ensemble	0.880359	0.003964	0.880511	0.003907	34 min

Table 18. Table with the best models

Clearly we see that the **ANN model metrics are the worst**, the lowest metrics, the highest run time (almost twice the run time of the second model) and the largest standard deviation. Taking out the rest of the models, we can see that the rest of the models obtain very good results (even if they are worse than the previous approach, see in *table* [12]). Without taking into account the computational cost the best model would be the **SVM** with rbf kernel, c=1 and gamma=3, taking into account the computational cost the best model would be the **kNN** with k=1 since the metrics are slightly lower (around 1%) and the execution time is much lower. Let us therefore look at the graph of these models:

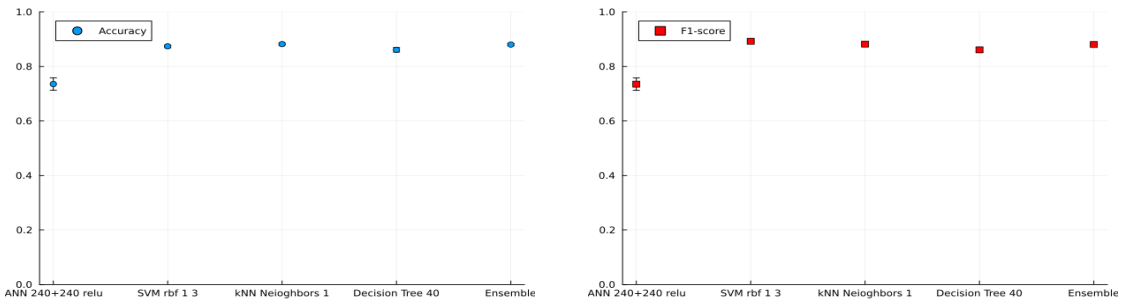

Fig. 27. Accuracy and F1-score for the Ensemble model and the best models

As a conclusion of this approach we can say that, although it may seem counter-intuitive, **by performing the dimensionality reduction we obtain similar results to approach one**, both in execution times and metrics. However, we obtain worse results with respect to approach two, where we also performed the dimensionality reduction manually (leaving even less features) but we had a higher number of songs.

5 FOURTH APPROACH

In this last approach we will apply the same methods as those used so far, we will simply use a slightly different treatment of the data than in the previous approaches. In the first approach we chose the **15 years** we are working with and used 5000 songs per year with their 90 features, in the second approach we increased the dataset by 2500 songs per year and decreased the dimensions to 12 and in the previous approach we chose 78 dimensions and 5000 songs per year. In this last case we will again make a **dimensionality reduction by choosing 5000 songs per year**, however, this reduction **will be applied by performing the PCA** (*Principal Component Analysis*) method. This method transforms the dataset by projecting the components into a set of orthogonal axes. To do this, it searches for the best linear combination of the original characteristics, trying to maximise the variance. Once this linear combination of the characteristics has been made, those with the highest variance are chosen. Thus, in our case we will apply the method to our characteristics with the intention of maintaining 95 percent of the original variance.

Therefore, for this approach we will have a dataset of **75000 songs** spread over **15 years** where each song will have a certain number of characteristics, calculated by PCA maintaining 95% of the variance of the original dataset. It should be noted that this principal component analysis is used only with the training dataset, since in case of doing it with the test dataset it could contaminate the data, causing that when calculating the principal components it is influenced by the test data, taking those characteristics that have more influence on these specific data and not in general on the whole sample. Taking this into account and because we are going to use, as in the previous sections, Crossvalidation, the number of characteristics will depend directly on the training and test set that we are using, which in turn will depend on the fold of the crossvalidation in which we find ourselves. Looking at the models we will train next and the folds they use we can see that the number of features will vary between 66 and 67.

In addition, the other features of the models will be similar to those used previously. The data will be *normalised* to obtain characteristics of *mean 0 and standard deviation 1* (note that the normalisation is performed before applying the PCA). In the *crossvalidation* we will use *10 folds* as in the rest of the cases and in case of needing validation set we will use the *HoldOut* function to obtain 20% of the training sample. We will also apply Random.seed! to ensure that the results are repeatable. Finally we will use the same metrics used in the previous approaches: **accuracy**, **F1-score** and **execution time** (in minutes).

5.1 ANN

The first method we will use in this last approach is **ANNs**. Within this method we will use the same models that we have used so far. The major change between the models used in the previous approaches and the models we will study next are the number of model inputs, since the rest of the hyperparameters will be identical to the previous approaches. Regarding the number of inputs, it will depend on the fold we are working on, as already mentioned in the introduction of the model will vary between 66 and 67.

One Hidden Layer. For the ANNs with one hidden layer we have the next models: 1 Hidden Layer with ***15*** neurons and a ***softmax*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***tanh*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***sigmoid*** transfer function, 1 Hidden Layer with ***15*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***30*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***120*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***240*** neurons and a ***relu*** transfer function, 1 Hidden Layer with ***480*** neurons and a ***relu*** transfer function. The reason for the choice of these models is already justified in the first approach (*see in section* 2.1). In the following table we can see the results of the metrics for ANNs with only one hidden layer:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
15, softmax	0.154880	0.004565	0.138821	0.004565	5 min
15, tanh	0.154880	0.004565	0.138821	0.004565	5 min
15, sigmoid	0.154880	0.004565	0.138821	0.004565	5 min
15, relu	0.154880	0.004565	0.138821	0.004565	5 min
30, relu	0.180000	0.006722	0.167694	0.006722	6 min
120, relu	0.303307	0.006750	0.295452	0.006750	12 min
240, relu	0.466710	0.009735	0.462888	0.009735	22 min
480, relu	0.729118	0.009732	0.728650	0.009732	60 min

Table 19. Table with the metrics of the ANNs with a single hidden layer

We can see that similar to the previous approaches the models **with only 15, 30 or 120 neurons are not the most desirable models**, however the interesting thing about these models is not to identify that the best model is the model with 480 neurons in the hidden layer, the most interesting thing is to compare the results obtained with the previous approaches we have made (see in the *table* [1], *table* [7], *table* [13]). With this reasoning we can clearly see that ***the execution time of these models is much shorter than the previous approaches***, probably due to the fact that we managed to maintain a large part of the variability by reducing the number of features. Another relevant fact is that despite having fewer features, t**he metrics are similar to those obtained in the best models** (obtained in the first approach) and better than the data from the last two approaches. Following the structure of the previous methods, let's look at the graphs of the models as a function of the metrics (representing the mean and the standard deviation).

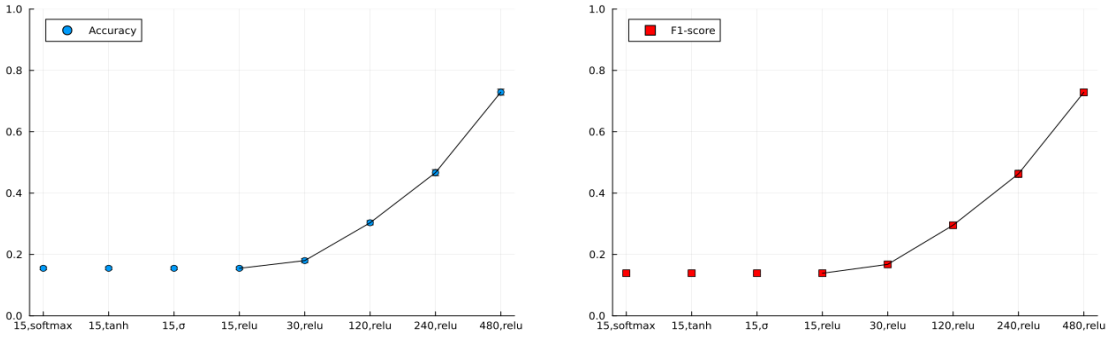

Fig. 28. Accuracy and F1-score for the ANN models with one hidden layer

Two Hidden Layer. As for the models with a hidden layer we will use the same models as in the first approach: 1 Hidden Layer with ***(15,15)*** neurons and a ***softmax*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***tanh*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***sigmoid*** transfer function in both layers, 2 Hidden Layer with ***(15,15)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(120,120)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240,240)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(480,480)*** neurons and a ***relu*** transfer function in both layers, 2 Hidden Layer with ***(240, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one and 2 Hidden Layer with ***(480, 240)*** neurons and a ***sigmoid*** transfer function for the first layer and a ***relu*** transfer function for the second one.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
(15,15), softmax	0.158989	0.008494	0.142836	0.007975	9 min
(15,15), tanh	0.158989	0.008494	0.142836	0.007975	9 min
(15,15), sigmoid	0.158989	0.008494	0.142836	0.007975	9 min
(15,15), relu	0.158989	0.008494	0.142836	0.007975	9 min
(120,120), relu	0.555703	0.013987	0.553596	0.014180	45 min
(240,240), relu	0.768189	0.009826	0.767909	0.009947	60 min
(480,480), relu	0.658614	0.029600	0.656288	0.030283	95 min
(240,240), sigmoid+relu	0.768189	0.009826	0.767909	0.009947	65 min
(480,240), sigmoid+relu	0.660225	0.014761	0.657467	0.015854	60 min

Table 20. Table with the metrics of the ANNs with two hidden layer

With respect to the models with two hidden layers we can have a similar reasoning to the models with only one, the most interesting thing is not to see the best of the models, that seeing the dynamics of the previous approaches to solve this problem it is logical that it is one of the models with 240 neurons per hidden layer, the most interesting thing is to analyse the comparison with the previous models (see the *table* [2], *table* [8], *table* [14]). Again we can see that **the execution times are the shortest of the 4 approaches** while **the results are very similar** to the best ones we have obtained (in all of them slightly below 80%). Let us now see the graphical representation of the metrics of each model and finally the representation of the execution times as a function of the F1-score.

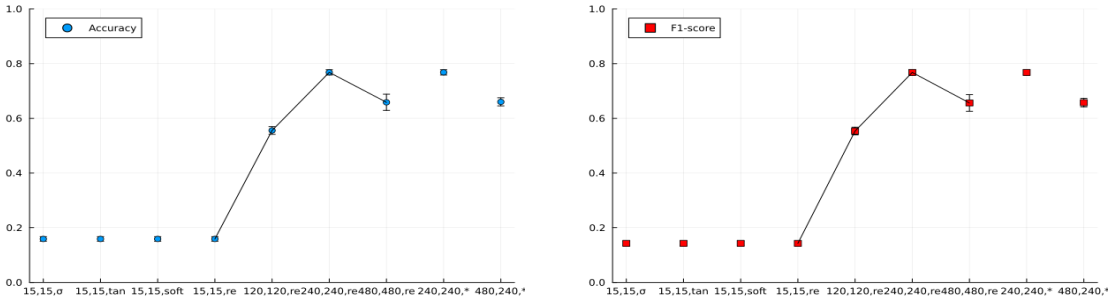

Fig. 29. Accuracy and F1-score for the ANN models with one hidden layer

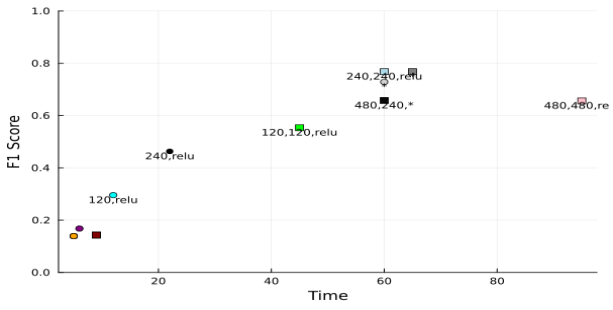

Fig. 30. Execution time in the ANNs

Taking into account all the tables and graphs, we can affirm that the model with two hidden layers, 240 neurons and relu transfer function is the best model, although the same model with the relu transfer function and the model with sigmoid transfer function and the model with only one hidden layer and the same topology have similar results (note that the time graph has not been represented for convenience). In addition, **in comparison with the rest of the approaches** (with respect to the ANNs) **we achieved very good results**, having almost the best metrics and also having a slightly lower execution time than the rest.

5.2 SVM

In this section we will use the ***SVM*** method. As for the rest of the models we will try to use the same hyperparameters to be able to compare the results, however, when running the programs, due to the high computational cost we could not use the models with polynomial kernel except for the polynomial kernel of degree 3. Therefore, with this we will use the models with **linear kernel**, with **rbf kernel**, with **sigmoid kernel** and with the **polynomial kernel**. For the sigmoid and rbf kernel models we will vary their hyperparameters, for the sigmoid we will vary the value of c while for the rbf model we will vary the parameter c and the parameter gamma.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
linear	0.132941	0.002868	0.090027	0.002022	210 min
poly 3	0.878709	0.003405	0.878909	0.003402	65 min
rbf 1 3	0.872252	0.003094	0.891371	0.002387	60 min
rbf 500 10	0.871852	0.003479	0.890998	0.002707	70 min
rbf 1000 20	0.871372	0.003322	0.890699	0.002554	75 min
sigmoid 1	0.067534	0.002993	0.057847	0.002639	45 min
sigmoid 500	0.066079	0.008090	0.057477	0.004365	50 min
sigmoid 1000	0.068934	0.002748	0.059258	0.002919	60 min

Table 21. Table with the metrics of the SVM

As in the previous approaches we can see that **the models that best fit this problem are the polynomial kernel or rbf models**. All these models achieve metrics of approximately 87%. The most outstanding model is the **rbf model with c=1 and gamma=3, which obtains the second best metrics and the best execution time**. With respect to the other approaches, we can see that *they are not the best models we have obtained*, in fact, although there is not too much difference, they are the worst models we have obtained since in the rest we have achieved, at least, metrics of 88%.

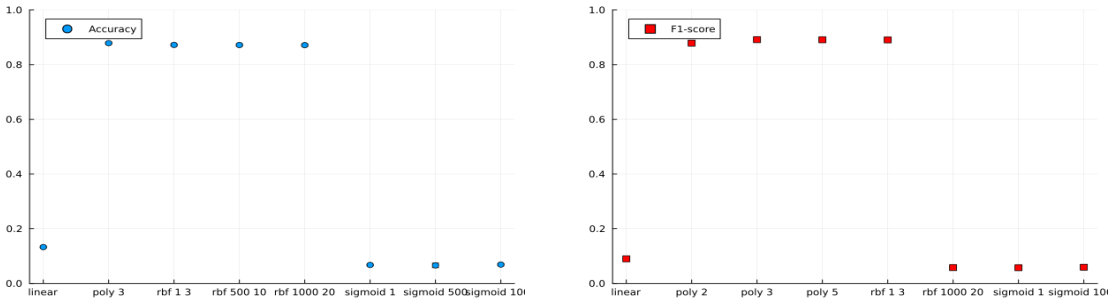

Fig. 31. Accuracy and F1-score for the SVM models

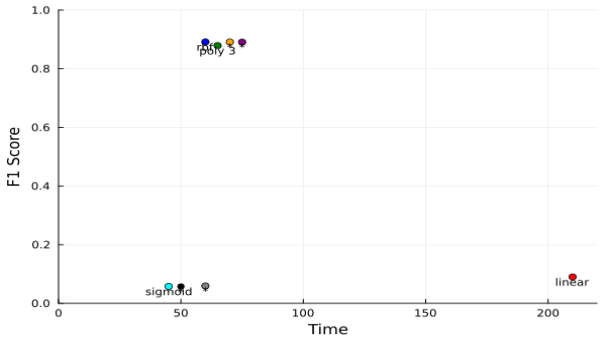

Fig. 32. Execution time in the SVM

5.3 kNN

The next method we will use is the ***kNN models***, varying the value of the k. As in the previous sections the running time of these models is negligible so we will not take it into account and, again, in order to be able to compare between the different perspectives we are using we will use the same values of k used in the first approach (***k=1, 2, 3, 5, 10, 15***).

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
neighbors 1	0.888813	0.004208	0.888781	0.004181
neighbors 2	0.806173	0.005782	0.806151	0.005811
neighbors 3	0.744600	0.006478	0.744780	0.006499
neighbors 5	0.611720	0.004019	0.611876	0.003946
neighbors 10	0.432746	0.004577	0.432700	0.004674
neighbors 15	0.381146	0.003371	0.380948	0.003530

Table 22. Table with the metrics of the kNNs

We can see that **the best model** continues to be, as in the previous approaches, **the model with k=1**. Furthermore, the inversely proportional dependence of the metrics on the value of k is very evident (which we will see more clearly in the graphs below). With respect to the metrics, we can affirm that in the case of k=1 we obtain a method with very good metrics, both higher than 88%. However, co**mparing them with the rest of the approaches** (especially with the second one, see in *table* [10]) we can see **that the results are a bit lower**.

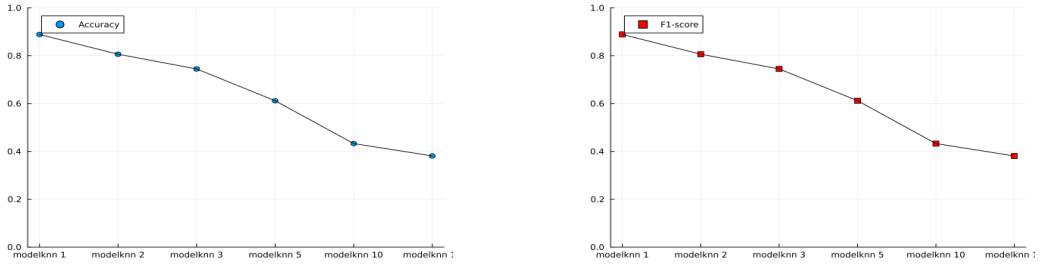

Fig. 33. Accuracy and F1-score for the kNN models

5.4 Decision Trees

The method we will use next is **Decision trees**, varying the depth of the model. As in the previous ones we will use the same depth to be able to compare the results, that is, the *depth will be equal to **5, 20, 30, 40, 50, 100***. Again, we will only use accuracy and F1-score as metrics since the runtime in these models is negligible.

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score
depth 5	0.079822	0.006269	0.079822	0.006269
depth 20	0.558774	0.028826	0.558774	0.028826
depth 30	0.824754	0.008535	0.824754	0.008535
depth 40	0.866842	0.005175	0.866842	0.005175
depth 50	0.869218	0.005696	0.869218	0.005696
depth 100	0.869221	0.005700	0.869221	0.005700

Table 23. Table with the metrics of the Decision Trees

Analogous to the previous models, *the metrics increase directly proportional to the depth of the model*, reaching a point where they no longer increase. It should be noted that in this approach the model with a depth of 20 has a standard deviation much higher than those obtained in the rest of the models. We can also observe how **the metrics obtained in this model are a little worse than those obtained in the models of the second approach** and are very similar to the other approaches (see *table* [5], *table* [11], *table* [17]).

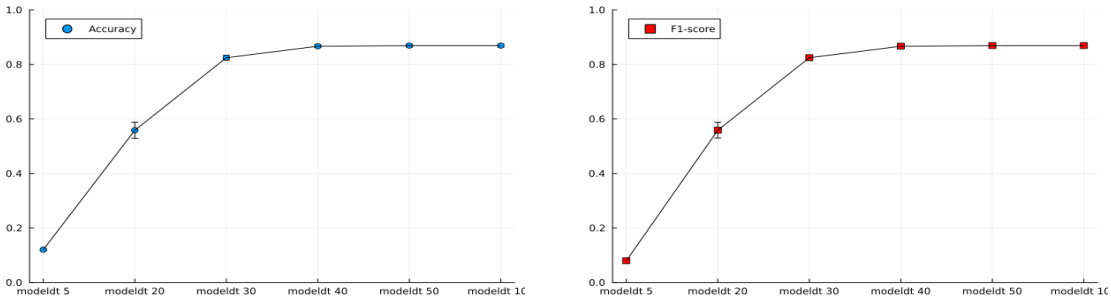

Fig. 34. Accuracy and F1-score for the Decision Trees models

5.5 Ensemble model

Finally, we will use an **ensemble model** with the **Stacking technique**. For this ensemble model we will use the same models as in the previous approach: ***a kNN model with k=1 and two Decision Trees with a depth of 30***, the justification for the selection of these models is the same as that given in the *section* 6 on approach one.

To compare the models we choose the best models of each of the methods: within the **ANN** we choose the model with two hidden layers and the transfer function, for the **SVM** we choose the model with the rbf kernel, C=1 and gamma equal to 3, for the **kNN** we choose the model with k=1 and for the **Decision Tree** we choose the model with depth 50 (we choose the model with the best metrics). We complete the table by putting a runtime for the kNN and Decision Tree of 5 min although as we have already said it is shorter. So let's look at the table comparing the best models of each method and the ensemble model:

Models	Mean Accuracy	St. Dev. Accuracy	Mean F1-Score	St. Dev. F1-Score	Time
ANN (240,240), relu	0.768189	0.009826	0.767909	0.009947	60 min
SVM poly 3	0.872252	0.003094	0.891371	0.002387	60 min
kNN Neighbor 1	0.888813	0.004208	0.888781	0.004181	5 min
Decision Tree depth 50	0.869218	0.005696	0.869218	0.005696	5 min
Ensemble	0.886491	0.004130	0.886539	0.004080	35 min

Table 24. Table with the best models

We can draw some analogous conclusions to the previous sections: **the worst method is the ANN**, with the worst metrics and a very high execution time (although in this case it is lower than in other occasions), the **best model** is the **kNN** with k=1 and *if we do not take into account the execution time any model of the rest could be a good model.* Finally, let's look at the graphs representing the metrics and the execution time.

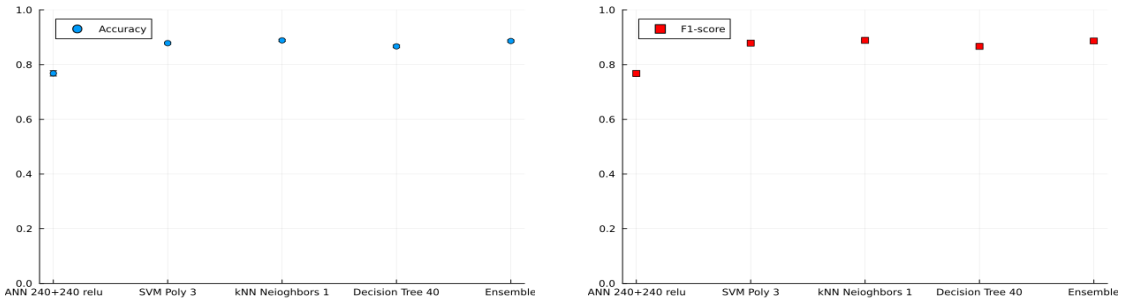

Fig. 35. Accuracy and F1-score for the Ensemble model and the best models

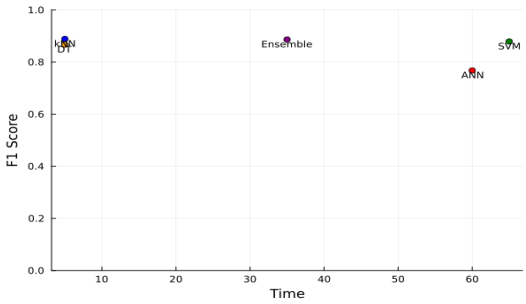

Fig. 36. Execution time of the best models

The most interesting analysis we can perform is to compare these models with the models of the other approaches, where, with respect to **run time** they are the **best approaches** (similar also to the third approach), while for **metrics they are clearly below the second approach**.

6 FINAL DISCUSSION

To conclude the project, we will discuss in general terms the results we have obtained in each of the methods, comparing both the approaches and the different methods and models within each method. Since all the results are already included throughout the work, we will not make any additional tables or graphs, but whenever we refer to a result we will refer to it. With this in mind we can start talking about the different methods and the performance of each one of them:

(1) **ANN** [7]: With respect to the ANN models, we can say that they are models that obtain relatively good results in most cases, obtaining metrics that in the best of cases are around 75%, except in the case of approach 2, which obtains much worse data. However, to achieve these results we need to have a large number of neurons, around 480 neurons, which means that training times are relatively long. It should also be noted that at a certain number of neurons, due to the uncertainty of having to adjust so many parameters, the standard deviation of the models increases a lot. In addition, we can assume that these are models that, in our case, need a large number of features, or at least with a large amount of variability because the model in which we obtained very poor results was the model in which we had the fewest features, despite having a larger number of songs. As a general summary, we can say that these models can be used as long as you do not need very reliable results and as long as you do not need short execution times.

(2) **SVM** [8]: SVM models are models which, in order to solve this problem, generated many inconveniences, especially when training models with the linear or polynomial kernel, obtaining execution times that were too high. The models with the rbf kernel obtained very good results in all approaches, with around 90% accuracy. In contrast to the ANN models, these models seem to indicate that they perform better the more data they have, even though the characteristics and variability are smaller (approach two clearly obtaining the best results). Generally speaking, we can also state that the execution times of these models are very high, being almost always longer than one hour.

(3) **kNN** [9]: The most remarkable thing about these models is the very short execution time they have in all cases, obtaining very good results. As is evident, these are models which cannot have a high value of k, however, it is worth noting that in all approaches, the best model is always the model with k=1. Analogously to the SVM models it can be assumed that they are models that obtain better results the more songs and not according to the characteristics, it obtains the same results for approaches one, three and four, having a very different number of characteristics while for approach 2 it obtains better metrics.

(4) **Decision Trees** [10]: The conclusions for these models are similar to those for the kNN models, they are models with very short run times despite always obtaining very good results, anyway, the results are better for the second approach, with more songs, and are almost the same for the other approaches.

(5) **Ensemble model** [11]: Both in terms of metrics and variations of these, the results of this model are very similar to the two superior ones, obtaining identical metrics. However, this model has longer execution times, at most 30 minutes. The ensemble models are models that are not worthwhile because, despite using models with very low run times, their run time was much longer and, moreover, they did not obtain very different results.

[7] *We can compare the results in tables [1], [2], [7], [8], [13], [14], [19], [20]*
[8] *We can compare the results in tables [3], [9], [15], [21]*
[9] *We can compare the results in the tables [4], [10], [16], [22]*
[10] *We can compare the results in the tables [5], [11], [17], [23]*
[11] *We can compare the results in the tables [6], [12], [18], [24]*

With regard to the different approaches we can draw the following conclusions. It should be noted that in this case we will not list them one by one because they have already been discussed in each respective section, we will simply make some general comments.

(1) The most obvious conclusion that can be drawn from all the methods and approaches is that the problem posed is a feasible problem and can be solved with any of the tools discussed.
(2) While this is a problem that can be solved and tackled in many different ways, there are clearly some better methods. In our case we can say that the best methods are the kNN and the Decision Tree models. In terms of approaches we can interpret that for these two models the best results can be obtained by increasing the number of songs used [12].

As a final note, in this work we had a concrete way of working with the data and choosing the models. It may be that there are models that are more optimised, that achieve better results or that can also be taken into account.

[12] As a note, we tested a kNN model with 10000 songs per year and got even better metrics, however we could not realise it because the other models had an excessively high runtime.

REFERENCES

[1] T. Bertin-Mahieux. 2011. *Year Prediction - Million Songs Database.* Retrieved November 27, 2023 from https://samyzaf.com/ML/song_year/song_year.html

[2] T. Bertin-Mahieux. 2011. YearPredictionMSD. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50K61.

[3] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011).*

[4] Nuwan S. Ferdinand and Stark C. Draper. 2018. Anytime Stochastic Gradient Descent: A Time to Hear from all the Workers. *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2018), 552–559. https://api.semanticscholar.org/CorpusID:52942895

[5] Martin Jankowiak. 2018. Closed Form Variational Objectives For Bayesian Neural Networks with a Single Hidden Layer. *ArXiv* abs/1811.00686 (2018). https://api.semanticscholar.org/CorpusID:53299227

[6] Oren Zeev-Ben-Mordehai, Wouter Duivesteijn, and Mykola Pechenizkiy. 2018. Controversy Rules - Discovering Regions Where Classifiers (Dis-)Agree Exceptionally. *ArXiv* abs/1808.07243 (2018). https://api.semanticscholar.org/CorpusID:52070818