Spotify Track Popularity Prediction

18/12/2019

Bachelor Degree on Informatics Engineering
Data Mining Course 2018-2019

Buch, Arnau
Díaz, Bernat
Heinsohn, Paul
Heredia, Gerard
Ramis, Felipe
Tro, Albert

Contents

Introduction

Spotify Technology S.A. is a Swedish international media services provider founded in 2006. The company's primary business is providing an audio streaming platform, the "Spotify" platform. This platform is a digital music, podcast, and video streaming service that gives you access to millions of songs and other content from artists all over the world. To this day, the Spotify platform provides access to over 50 million tracks.

Figure 1: Spotify's logo.

Spotify also provides calculated audio features of music tracks, accessible through their Web-API, to learn about a song's danceability, energy, valence, and more. For more advanced use cases, it is also possible to read in-depth analysis data about tracks such as the segments, tatums, bars, beats, pitches, and more.

In our project, we will be using these track features to train various classifiers in order to predict if a certain song will be popular or not. We believe this task can be useful for multiple areas, for example, building specific playlists for a user by training the classifier with the user's favorite music, studying key features in order to aid the making and marketing of music tracks to assure they are hits, etc. We will also put into account the performance of multiple classifiers (Naive Bayes, K-NN, Decision trees, Support Vector Machines and Meta-learning algorithms) to see which one is best-suited for this task.

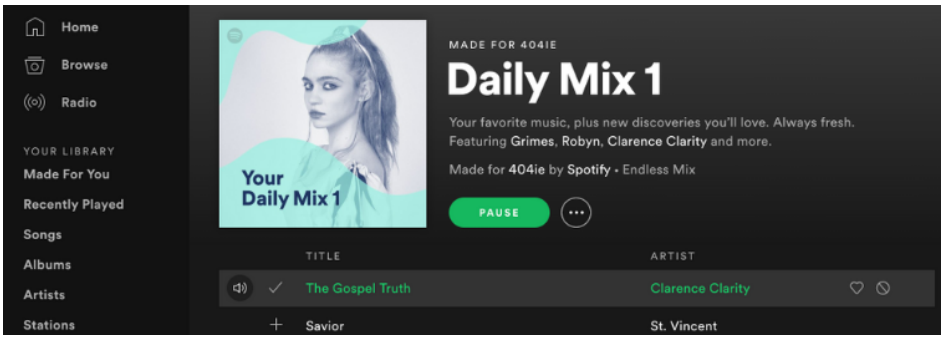


Figure 2: Example of a user personalized playlist.

Description of the original data

Our initial intention was to use the given dataset in this Kaggle page, but we realized it didn't include enough features, some of which are provided by Spotify but aren't used in the final dataset. That's why we decided to use the given script to obtain the data we wanted and modified it to include our chosen features. On the original dataset, the original author obtained data from 2018, and so we decided to update the dataset with data from 2019.

So we used the Spotify API, and the following endpoints (to get the rest of the features):

- Reference to audio-features endpoint
- Reference to tracks endpoint

Below is a table describing each of those features.

Feature	Description	Range of values	Missing data
num_artists	Number of artists	[1..n] integer values	1 (at least is made of 1 artist)
release_month	Month where the song was released represented by integer	[1, 12] (January...December)	Mean
release_weekday	Day of the week where the song was released by integer	[1, 7] (Monday...Sunday)	Mean
danceability	Describes how suitable the track is for dancing	[0.0, 1.0]	Mean
energy	Describes a measure of intensity and activity	[0.0, 1.0]	Mean
key	The key the track is in. It uses the pitch class table to represent the key as an integer. Pitch class	[0, 1, 2, ..., 11]	0 [Key in C]
loudness	The overall loudness of a track in decibels (dB)	[-60, .., 0]	Mean
mode	Boolean variable to indicate the modality of a track. The values are **major** or **minor.** Major is represented by 1 and minor by 0.	[1, 0]	0 (major)
speechiness	Presence of spoken words in a track. It's exclusive for speech recordings as poetry, audio books or something somewhat related.	[0.0, 1.0]	Mean
acousticness	Measure of whether the track is acoustic. 1.0 represents high confidence the track is acoustic	[0.0, 1.0]	Mean
instrumentalness	Predict whether a track contains no vocals. The closer to 1.0, the greater likelihood the track contains no vocal content	[0.0, 1.0]	Mean
liveness	Detects if the song is in a live recording. So mostly detects audience or crowdiness. Higher liveness, higher probability that the track was recorded live	[0.0, 1.0]	Mean
valence	Musical positiveness of the track. Higher valence means that the track emits happiness, cheerfulness or euphoria. Lower valence means that the tracks emits sadness, depression, angry..	[0.0, 1.0]	Mean

Figure 3: Feature description table.

Added features (not included in original dataset):

Feature	Description	Range of values	Missing data
time_signature	An estimate overall time signature. Is a notational convention to specify how many beats are at the same time.	[1, 5] (approx)	Mean
artist_followers	Number of followers that the **main** artist has	[0....100M] (approx)	0
artist_popularity	Popularity of the **main** artist	[0, 100]	0
duration_ms	Duration of the track	approx [100, 800]	Mean
track_len	Length of track title.	[1, 233]	Mean
tempo	Speed of the beat measured by BPM(bits per minute)	[0.0, 1.0]	100
artist_len	Number of artists for this track	[2, 49]	1

Figure 4: Added features description table.

Description of pre-processing of data

With our dataset complete, we now turn to the preprocessing applied to it. Since most features in the dataset are in different units, we decided to normalize the dataset to make it more compatible for distance related models such as K-NN or SVM. We used the scaler StandardScaler from sklearn.

Our predicted feature in the original dataset (popularity) is presented as a percentage, so we needed to decide how to convert it into a categorical variable. As seen in the description of the original data, the popularity feature follows normal distribution with a low variance and so any balanced division would happen around that area. To ensure a balanced dataset we used the pandas function qcut, which converts a given numerical value into a categorical one with the provided labels, all while maintaining a balanced dataset. We then evaluated different options to divide the predicted feature, with 2, 3 or 4 classes. In the table below we can see the obtained accuracy for a test run of a decision tree classifier using the best parameters for each number of classes.

#classes	Decision Tree accuracy	Range of classes
2	0.7384969325153374	(0, 61.0] < (61.0, 100.0]
3	0.6687116564417178	(0, 57] < (57, 65] < (65, 100]
4	0.5851226993865031	(0, 50] < (50, 61] < (61, 67] < (67, 100]

Figure 5: Decision Tree accuracy table.

With 2 classes we obtain our best accuracy, which isn't surprising. We also tested each number of classes on each of the models presented below and saw not only better accuracy with 2 classes, but improved training speeds. For some of the methods studied you will see 2 classes used and for others 3.

Evaluation criteria of data mining models

Throughout the following methods we have used these common parameters for evaluation. For the Naïve Bayes, K-NN, Decision Trees and Support Vector Machines we used the cross_val_score function from sklearn with 10 splits. For the Meta Methods we opted instead to use 50 splits. We also used the function GridSearchCV from sklearn to obtain the best parameters for some classifiers. Specifically, we used it for K-NN, Decision Trees, SVM and the K-NN method for Majority Voting, but not for Naïve Bayes and the rest of Meta Methods.

Finally to draw conclusions between classifiers we will compare their cross validation accuracy, their general accuracy and their confidence interval. These comparisons will be written down on a table in the conclusions section.

Naive Bayes

The first model used on our dataset will be Naive Bayes. This model is a probabilistic classifier based on applying the Bayes' theorem with a strong independence assumption between the features of the dataset.

Conditional independence assumption in the dataset

As mentioned before, the Naive Bayes classifier assumes that our dataset's features are independent from each other. To check if that is actually the case with our dataset we calculated the Pearson correlation coefficient between our numerical variables and visualized it in the following correlation matrix:

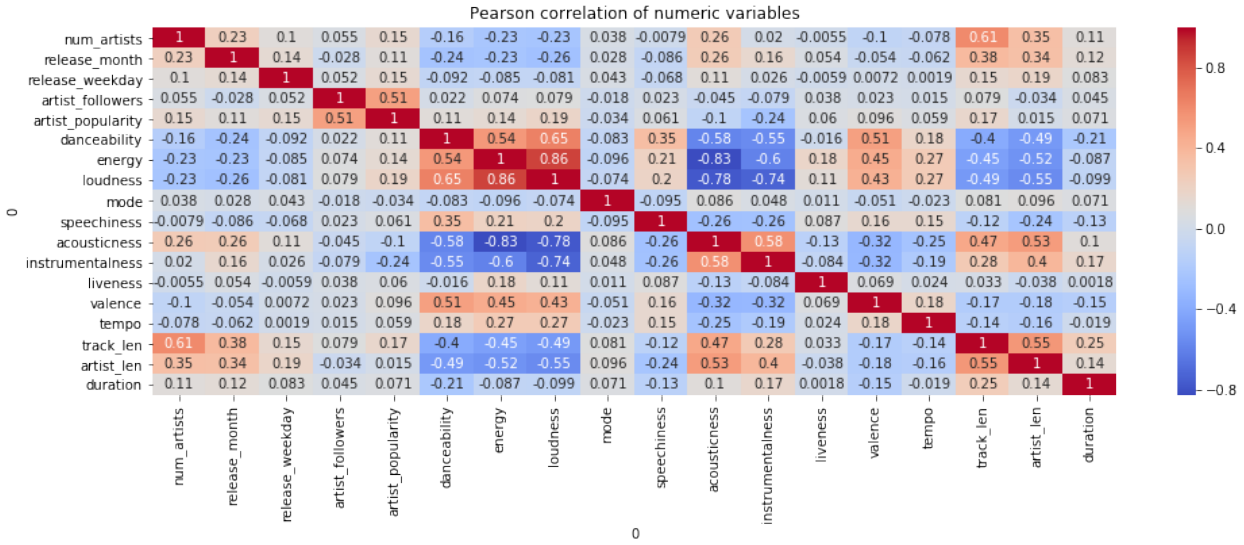


Figure 6: Pearson correlation matrix between numerical variables.

In the correlation matrix we can see that there exists noticeable correlation and inverse correlation between our variables. For example, we can spot a strong dependence between the variable energy and danceability, loudness, acousticness, instrumentalness, valence, track_len and artist_len. Therefore we can ensure that the independence assumption with which Naive Bayes works does not hold for our dataset. This does not mean the results will be catastrophic, since Naive Bayes usually performs well even if the independence assumption does not hold, but we can probably expect our classifier's performance to take a hit because of it. The resulting accuracy, 0.6 ± 0.19, is rather low.

Results

For our Naive Bayes classifier we will assume our features follow a Gaussian distribution since they are continuous variables. The model was tested with three classes and a normalized dataset was used. We also decided to drop the key and time_signature variables since they were originally categorical variables. We can see in the plot below the accuracy scores for the ten-fold cross validation performed on the model. The resulting accuracy, 0.6 ± 0.19, is rather low but still not too bad considering Naive Bayes is a really simple classifier.

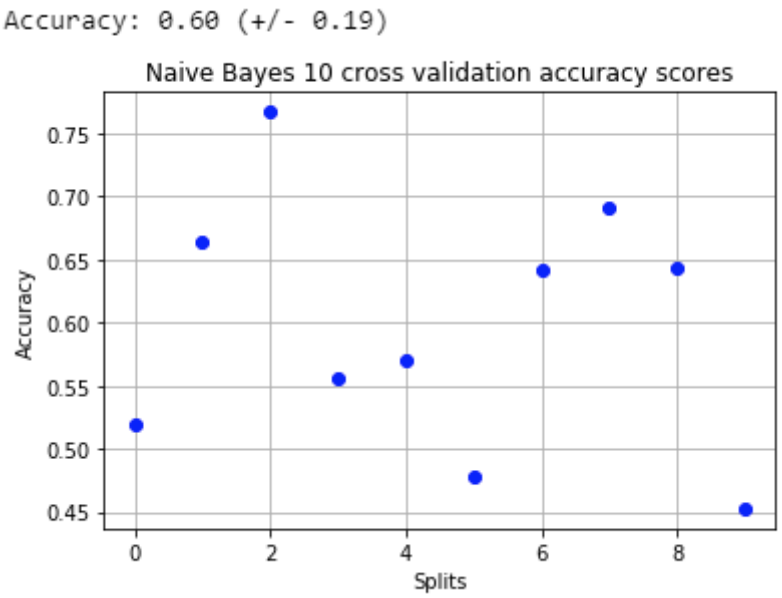
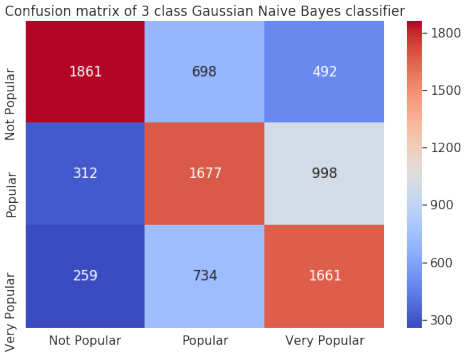


Figure 7: Accuracy scores of the 10-fold cross validation.

Below we can see the confusion matrix and classification report resulting from running our model. In the matrix we can notice that the classifier is pretty robust when it comes to classifying unpopular songs but seems to get confused between popular and very popular songs. The classification report shows us that the "Not Popular" class has a higher F-score than the other two classes which have a similar score between them which tells us that indeed our classifier is better at labelling unpopular songs than the other classes.



	precision	recall	f1-score	support
0	0.77	0.61	0.68	3051
1	0.54	0.56	0.55	2987
2	0.53	0.63	0.57	2654
accuracy			0.60	8692
macro avg	0.61	0.60	0.60	8692
weighted avg	0.61	0.60	0.60	8692

Figure 8: Resulting Gaussian Naive Bayes confusion matrix and classification report.

To further evaluate our Naive Bayes classifier we have plotted the Precision-Recall curves and the ROC

curves of our model shown in the figure below. Since we have roughly the same number of observations for each class, we should be paying attention to the ROC curve in this case. The ROC curve measures the ability of the fitted probabilities to classify correctly. We can see that the curve for the "Not Popular" class has a really good start but, as the false positive rate rises, the curve gets closer to the diagonal which means a worse fit. As for the other classes, "Popular" and "Very Popular", their respective curves start and remain mediocre which is consistent with past observations.

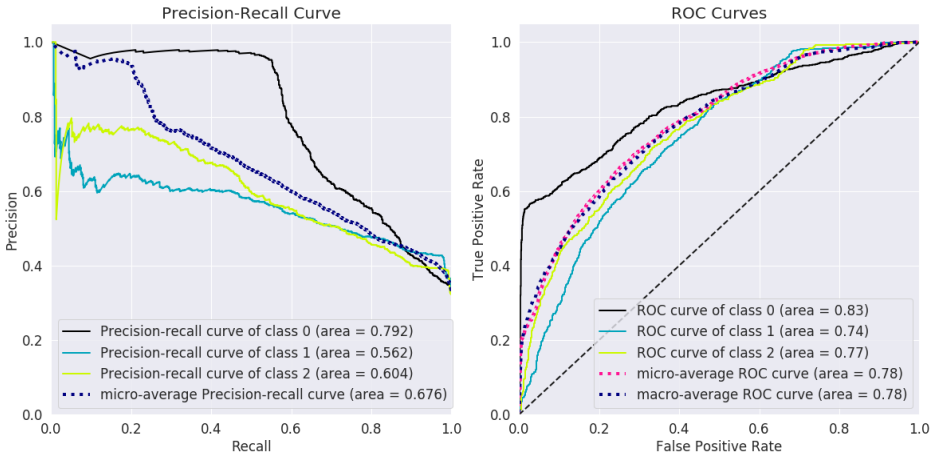


Figure 9: Precision-Recall curves and ROC curves.

From our cross validation results and the fact that some variables are strongly correlated, we can deduce that the Gaussian Naive Bayes classifier is not the classifier best-suited for our dataset. Overall, the classifier achieved mediocre accuracy for our classes, the most accurately predicted one being "Not Popular", with only 0.68 accuracy, and "Popular" and "Very Popular" having 0.55 and 0.57 respectively. Our dataset is not imbalanced and the classifier has more than enough variables to work with so we can only assume that Naive Bayes is simply not the right classifier for us.

These results are consistent with the results given by the predictions of the test data which are the following:

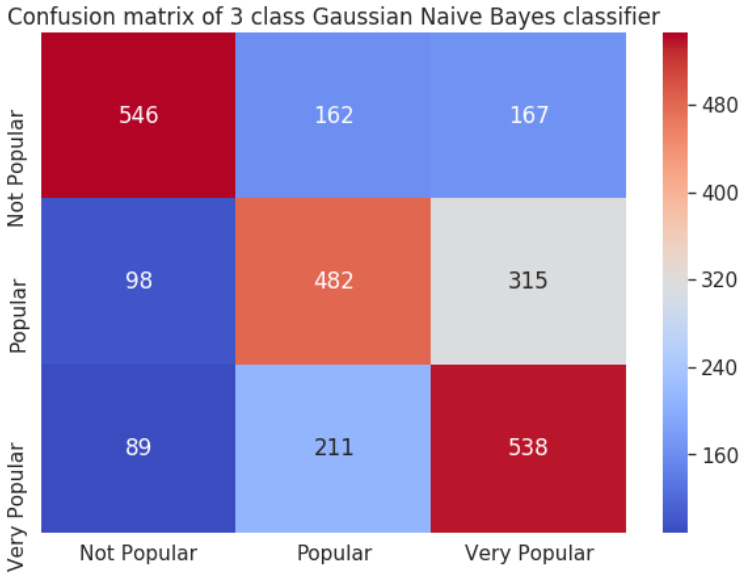


Figure 10: Resulting Gaussian Naive Bayes confusion matrix using test data.

	precision	recall	f1-score	support
0	0.74	0.62	0.68	875
1	0.56	0.54	0.55	895
2	0.53	0.64	0.58	838
accuracy			0.60	2608
macro avg	0.61	0.60	0.60	2608
weighted avg	0.61	0.60	0.60	2608

Figure 11: Classification report using test data.

As we can see, the “Popular” and “Very Popular” class still follow the same behaviour, as does the “Not Popular” class which is to be expected that our test results follow the same behaviour as our ten-fold cross validation results.

K-NN

For the second model to be evaluated we have a K-Neighbors Classifier. With it we used the dataset with 3 classes in the predicted label. With a 10-fold cross-validation we get an error mean of 0.618 and a standard deviation of 0.0705.

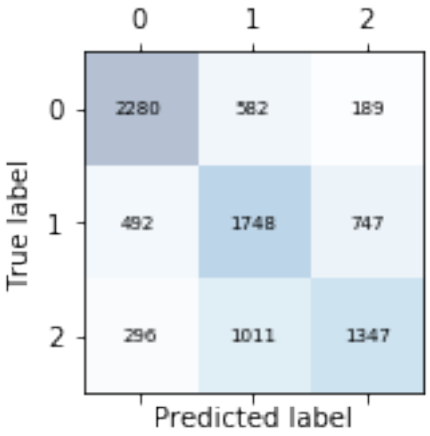


Figure 12: Confusion matrix of K-NN classifier.

The best parameters for the classifier are k = 27 neighbours and a uniform weight. To find them we used a grid search cross-validation (GridSearchCV).

With this parameters approximating the data by a Normal Distribution we get a 95% interval of confidence of (0.6386, 0.6750) accuracy and with a binomial distribution (0.6379, 0.6745).

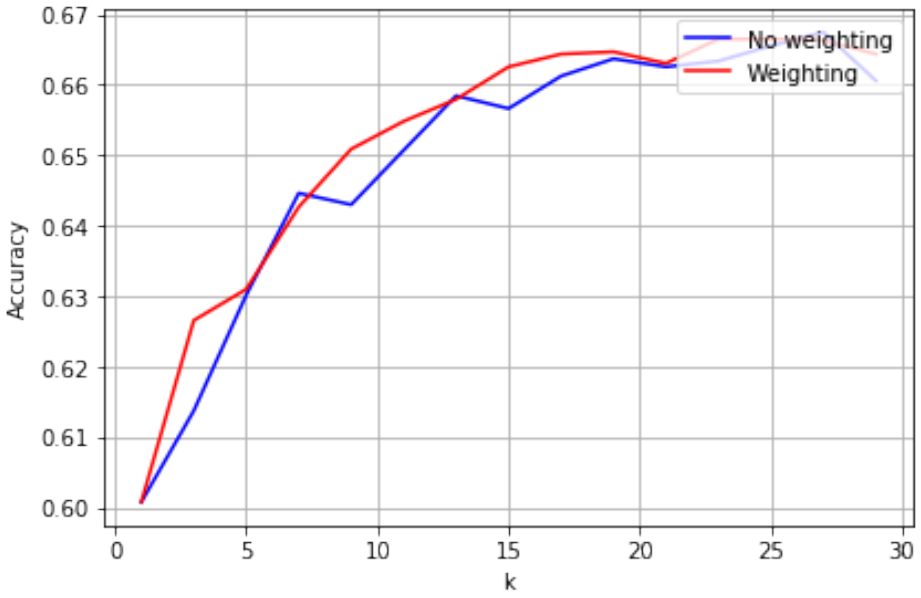


Figure 13: Accuracy scores by k nearest neighbors.

We then removed noise by selecting different numbers of features. As we can see in the plot below, the best is 4 features.

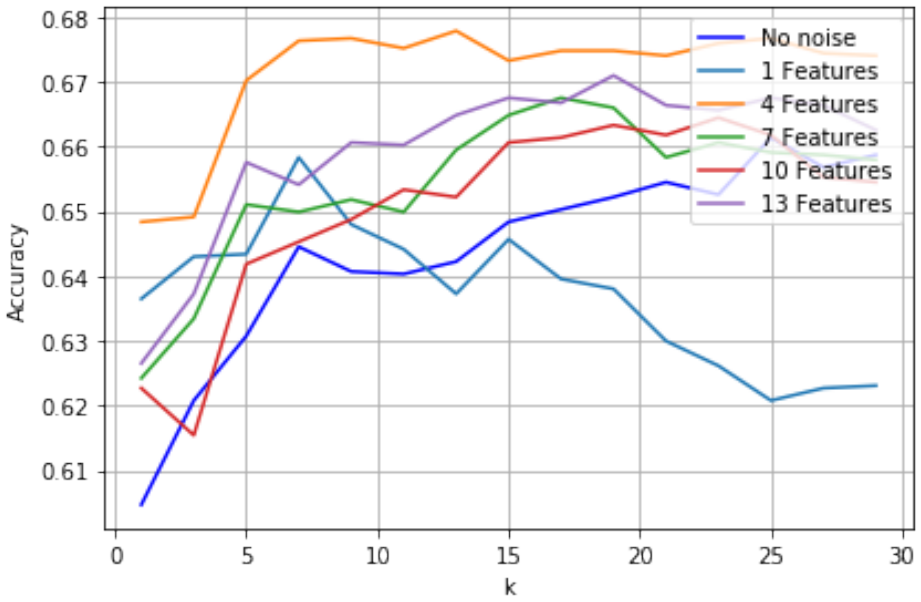


Figure 14: Accuracy scores by k nearest neighbors and number of features.

Decision Trees

For this classifier we will use two classes ("Not Popular" and "Popular"). To get a better analysis for the decision trees, we can do a few tests over our dataset, building a Decision Tree. Using a cross validation score, we want to check the randomness of our model giving us a mean of 71,5%.

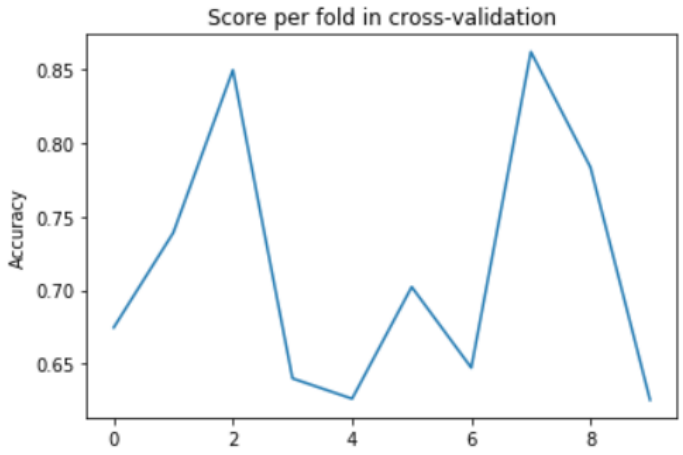


Figure 15: Accuracy scores of ten-fold cross validation.

The accuracy follows a distribution, but we don't know why yet. Therefore we build a model using some general parameters to see what it's built. The resulting decision tree can be found in the next page:

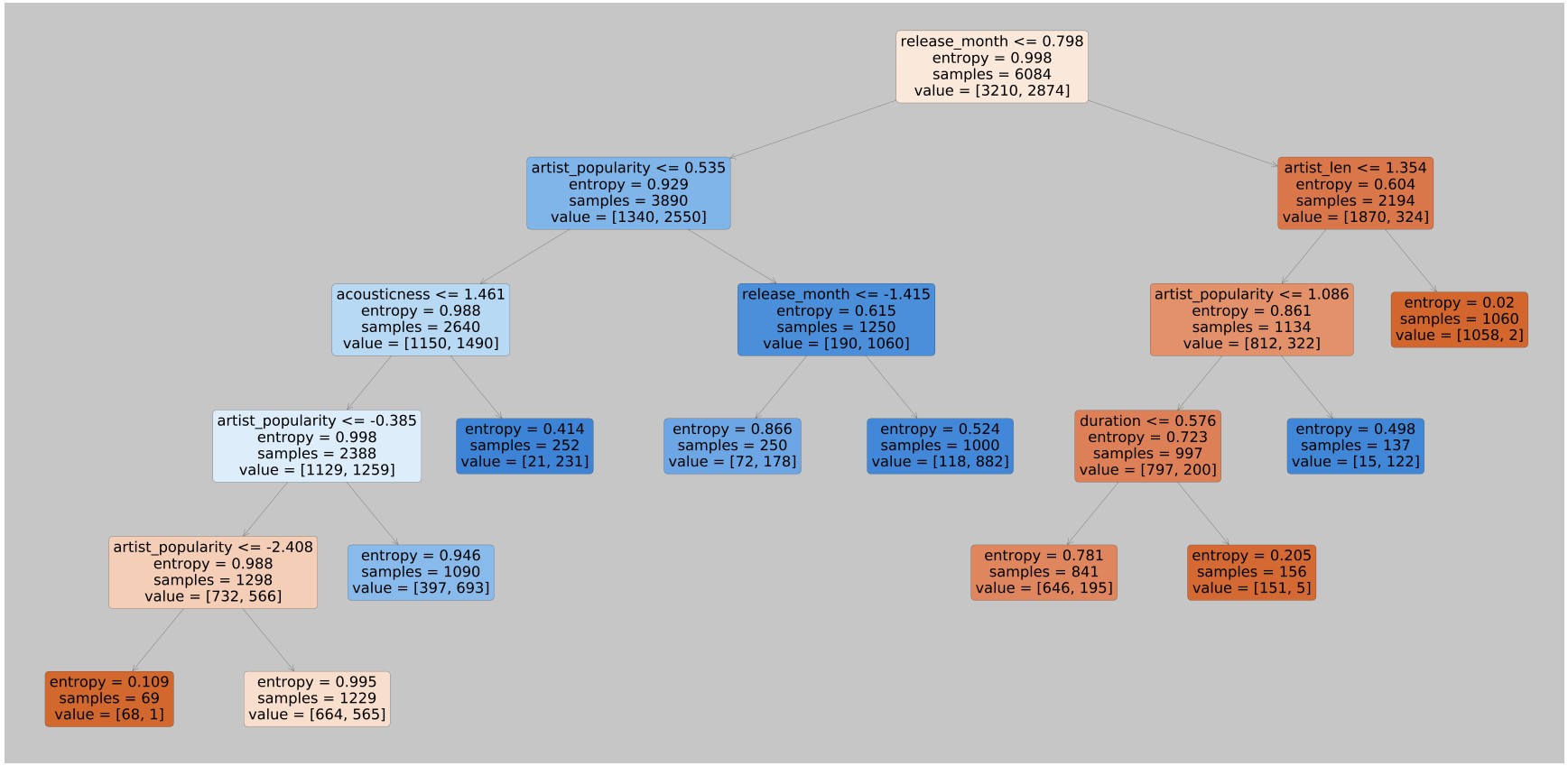


Figure 16: Decision Tree built with general parameters.

This tree gives us an accuracy of 75,27%, and an interval of confidence of (73.56% - 76.9%). It seems like the release month of the song is the most determinant attribute, but we can check the feature importance plotted below.

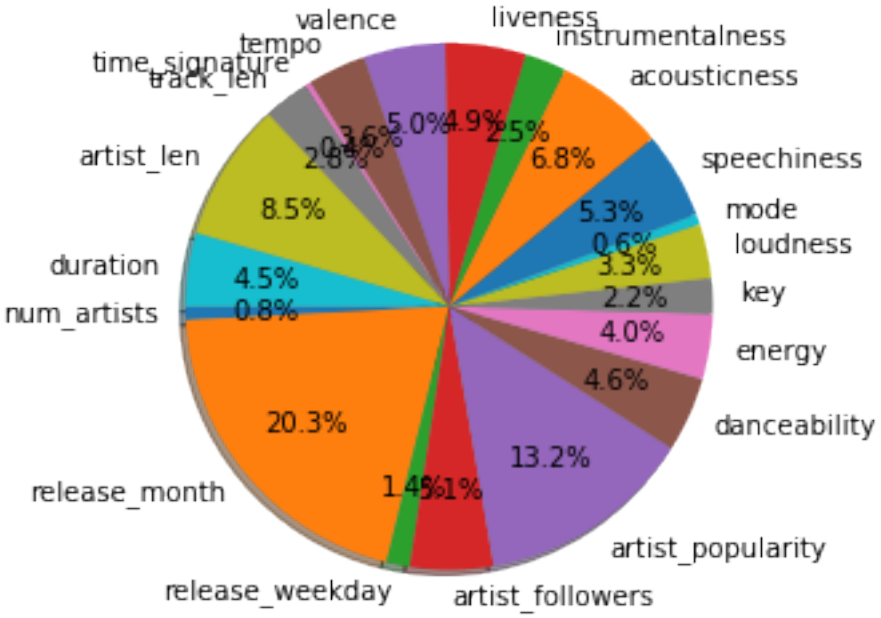


Figure 17: Feature importance pie chart of general params Decision Tree

Although the release month stands out over all the other features, we can see a lot of variance that may give us the initial randomness of the scores. So, to make the classifier more robust, we use a GridSearch to look for the best parameters of the model, being these attributes:

- Minimum impurity decrease
- Minimum samples split
- Criterion (Entropy or gini, we used entropy for the first one)
- Splitter: If we split nodes randomly or by the best attribute
- Maximum depth of the tree
- Maximum features to compare a single node
- Maximum leaf nodes of the tree

(We use the depth, sample split and leaf nodes to prune the tree)

Once we have the best parameters, we build the tree again giving us the structure in the page below:

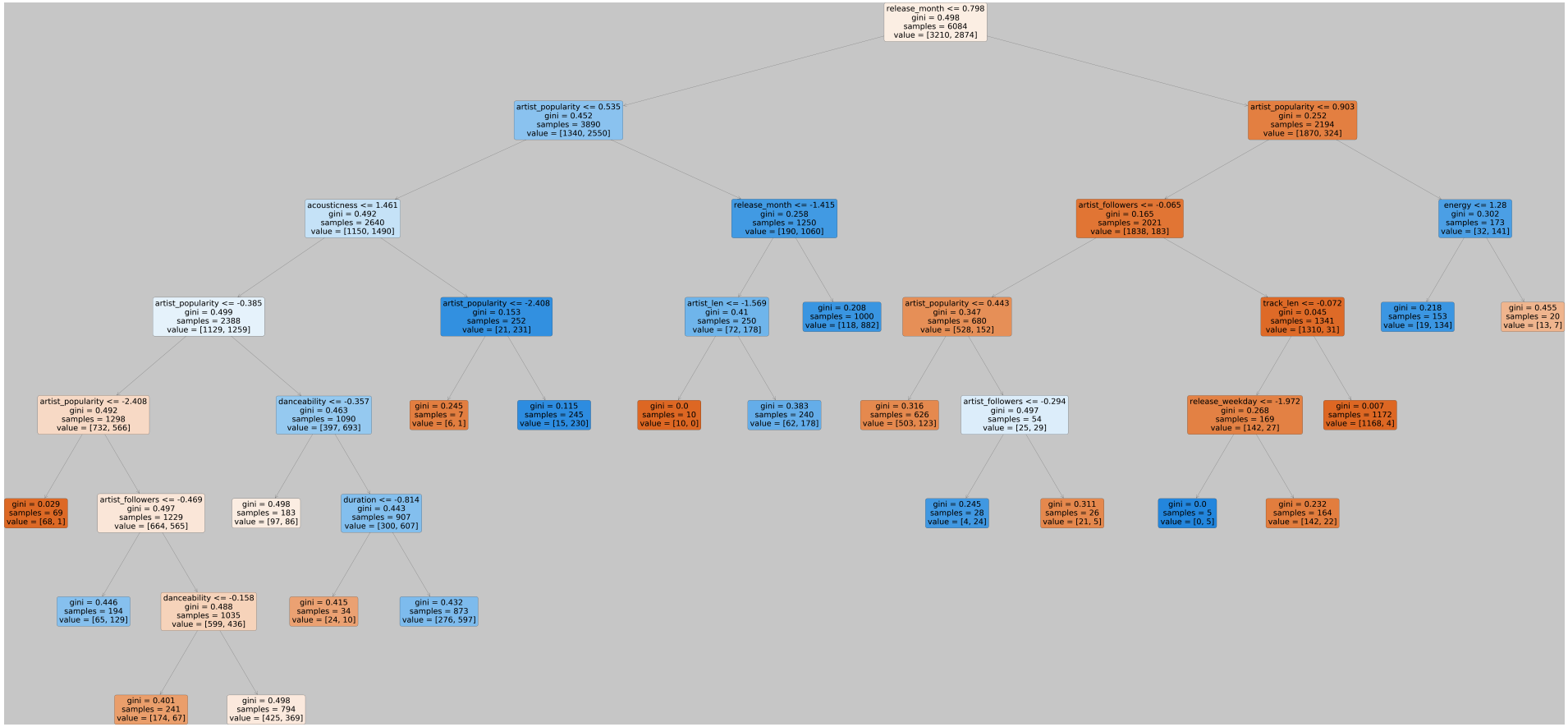


Figure 18: Decision Tree built with best parameters.

If we zoom in we can see almost the same distribution as before where the 3 main attributes are the release month, the artist popularity and how many followers the artist has.

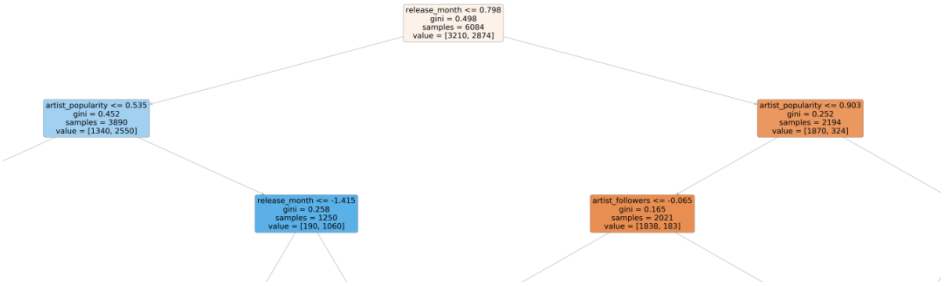


Figure 19: Section of decision tree with best parameters.

This model has slightly better accuracy with 76.34% and an interval of confidence of (74.62% - 77.91%).

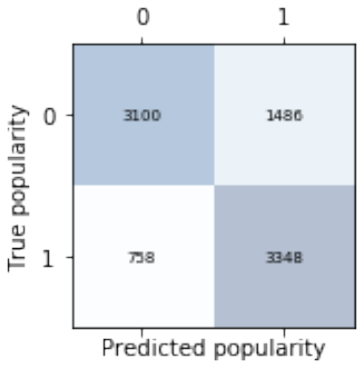


Figure 20: Confusion matrix of Decision Tree with best parameters.

The model struggles more to say if a song is going to be popular rather than saying one is not popular.

- To predict if a song is popular we have 77% precision, 71% recall and 74% f-measure
- To predict if a song is not popular we have 76% precision, 81% recall and 78% f-measure.

If we plot our features importance, we can see that our model behaves less random in terms of the features relevance.

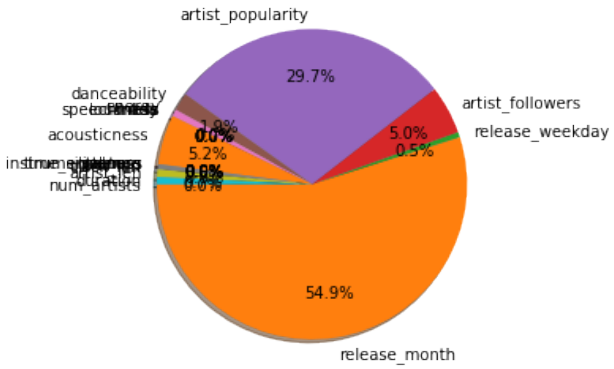


Figure 21: Feature importance pie chart of best params Decision Tree.

The most relevant attributes that determines if a song is popular is not, given by the Decision Trees are:

- Release month: This is the most interesting attribute. There may be some months more suitable to release the song.
- Artist popularity: This is the most obvious attribute. If an artist is popular, her/his songs are going to be popular easier.
- Acousticness: Normally songs with some voice, have a lot of acousticness added to it
- Artist followers: Same as the artist popularity
- Danceability: We can see that the most commercial songs follows this

As we said in the release month attribute, we want to see what distribution follows for the songs that are popular. We use a barplot to visualize this grouped by the popularity.

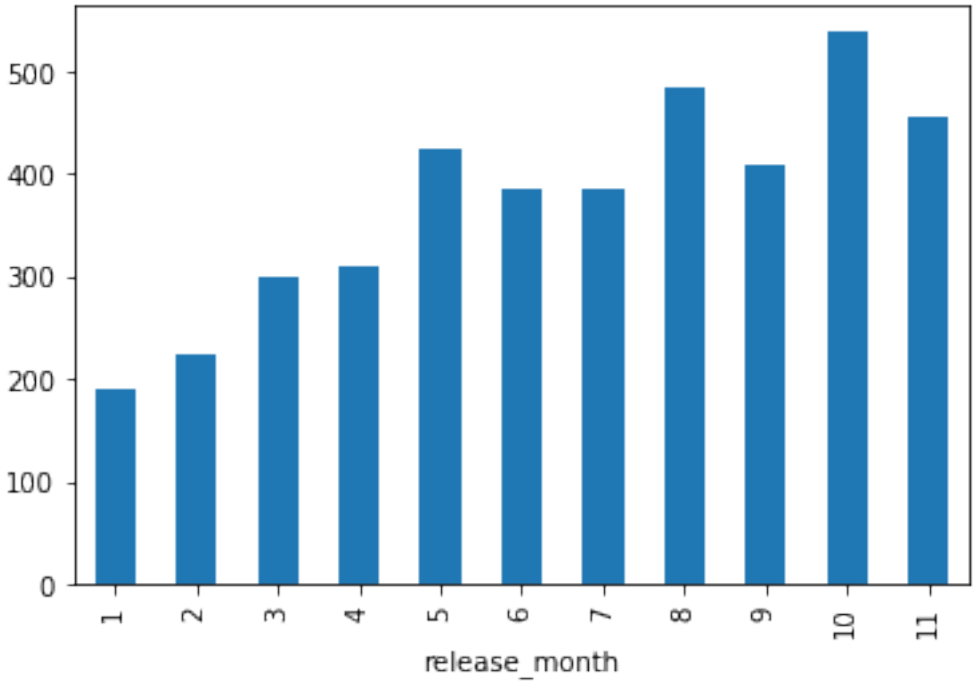


Figure 22: Popularity by release month Bar plot.

Conclusion may be, if you want your song to be popular, maybe you will need to release on the last months of the year, preferably October.

Support Vector Machines

In order to decide the kernel for our support vector machine, we made assumptions based on the simplest hypothesis, starting from the least to the most complex kernel; we trained four different support vector machines with a 10-fold cross-validation instead of a 5-fold to have a more robust classificator with less variance. Also, we used the best parameters to get the best results given our datasets and the dataset with three different predicted labels ("Not Popular", "Popular" and "Very Popular").

The three parameters we will be playing with are:

- **C**: Regularization parameter. The higher C is, the less regularized the data.
- **Gamma**: Kernel coefficient
- **Decision function shape**: We need a decision function to decide whether an individual song is in one class or another. There are two strategy types: one vs rest, where we compare one class to the other ones, or one vs one, where we compare all the classes but only two at the same time.

We will build SVM for every possible kernel: linear, radial basis function (rbf), polynomial and sigmoid.

The first SVM trained was the linear one, which gave us the following results:

- Execution Time: 12 hours
- Decision Function Shape: One versus Rest
- Accuracy Training Data: 0.6394
- Accuracy Test Data: 0.6400
- Recall/Sensitivity: 0.5623
- F - Measure: 0.5986
- Best C: 100000.0
- N° Supports: 4051 (6421 have slacks)
- Prop of Supports: 1.55

And the following confusion matrix:

Classes	0	1	2
0	1135	92	149
1	287	151	170
2	174	67	383

Figure 23: Confusion matrix of SVM with Linear kernel.

Despite a lot of time was spent finding the best parameters and training the machine, we can see that the output is not as good as we expected; low scores and a massive amount of supports, which leads us to think our classifier is actually very bad, so we proceeded to train a support vector machine with a more complex kernel like Sigmoid which outputted the following results:

- Execution Time: 18 minutes
- Decision Function Shape: One versus Rest
- Accuracy Training Data: 0.6427
- Accuracy Test Data: 0.6461
- Recall/Sensitivity: 0.5640

- F - Measure: 0.5986
- Best C: 100000.0
- Best Gamma: 0.0001
- N° Supports: 4153 (6735 have slacks)
- Prop. of Supports: 1.59

And the following confusion matrix:

Classes	0	1	2
0	1558	65	153
1	297	141	170
2	189	49	386

Figure 24: Confusion matrix of SVM with Sigmoid kernel.

In this case, we can see that the execution time decreased drastically and both accuracies, training and testing, increased. Although the amount of supports increased, the proportion is practically the same, so we can say that changing Linear kernel to Sigmoid was a good choice. However, accuracy on test data only improved approximately a 0.5%, so we proceeded to try with another kernel; polynomial, which gave us the following results:

- Execution Time: 22 minutes
- Decision Function Shape: One versus Rest
- Accuracy Training Data: 0.6571
- Accuracy Test Data: 0.6541
- Recall/Sensitivity: 0.5615
- F - Measure: 0.6043
- Best C: 1000.0
- N° Supports: 3758 (5887 have slacks)
- Prop. of Supports: 1.44

And the following confusion matrix:

Classes	0	1	2
0	1205	53	118
1	327	137	144
2	210	50	364

Figure 25: Confusion matrix of SVM with polynomial kernel.

Even though the execution time increased 3 minutes, accuracies improved over 1%, while the proportion of support deceased. With this kernel, the model has a good accuracy to determine if a song is not popular or if it's popular (Class 0 and 2), but struggles on predicting the ones that are in the middle that only has the range of 57 to 65 of popularity. We had more songs predicted as 0 or 2 where they were actually 1.

Moreover, we still are getting more false negatives than the correct answer in class number one, so we trained a final machine with "rbf" kernel and got the following results:

- Execution Time: 105 minutes
- Decision Function Shape: One versus Rest
- Accuracy Training Data: 0.6589
- Accuracy Test Data: 0.6607
- Recall/Sensitivity: 0.5697
- F - Measure: 0.6118
- Best C: 100.0
- Best Gamma: 0.1
- N° Supports: 3828 (5668 have slacks)
- Prop. of Supports: 1.47

And the following confusion matrix and heatmap showing the validation accuracy based on the gamma and C parameters:

Classes	0	1	2
0	1208	48	120
1	321	139	148
2	201	47	376

Figure 26: Confusion matrix of SVM with rbf kernel.

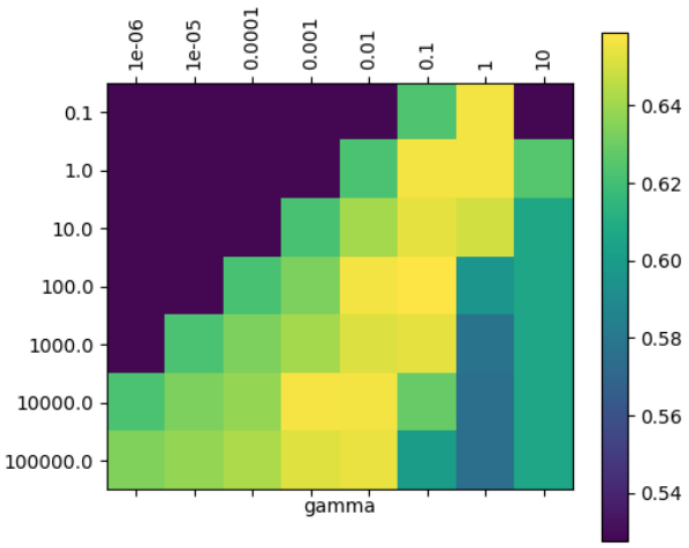


Figure 27: Heatmap of validation accuracy as a function of gamma and C of SVM with rbf kernel.

However, as we can see, this final SVM could not correct the misclassification problem, moreover, it did not matter if the method used was "one versus rest" or "one versus one", the first one was better, but not enough due to the high amount of supports in the machine. This vectors added ambiguity and difficulty to discern between classes, and explains why all confusion matrix outputs a lot of misclassification, especially with class number one.

Also, despite the fact that the selection of the kernel was not much influential in terms of accuracy (improvement of 2%), and that there was an increase in the usage of computational resources, we considered that a support vector machine with "rbf" kernel is the best. It ran in a reasonable amount of time (less than 2 hours), it gives us the bests accuracies, the best sensitivity, and the best harmonic mean between these values. Finally, even though polynomial kernel has the best number of supports, the number is still high and has no much difference with this one.

In order to speed up the build of the model, we tried different combinations of C, gamma and decision function shapes ranges for finding the best values; having low ranges leads to less iterations, which traduces in a lower execution time and therefore, more executions and trials to improve our classifier. Nonetheless, there was nothing else we could do to speed execution times.

As for the parameters, C tells the SVM optimization how much we want to avoid misclassifying the training examples, a large value will choose a smaller margin hyperplane for separating classes, whilst a low value the opposite. In this case we have that C is 100, so we do penalize misclassification, but there is a contradiction because our SVM has low accuracy in both, training and testing. This leads us to the absence of overfitting and the possible presence of underfitting.

On the other hand, gamma parameter defines how far the influence of a single training example reaches. Intermediate values means that the region of influence of the support vector is not small enough to be constrained and not high enough to include the support vector itself. As gamma is 0.1, we have an intermediate value, meaning that our model has a good structural regularizer, yet this was not reflected in the outcome; we have high bias and high variance, thus, underfitting theory is more believable.

In conclusion, despite of our efforts invested on searching for the best parameters and training multiple support vector machines, we failed building a good classifier. As we mentioned before, the underfitting problem could be because of our dataset; maybe the amount of data is not enough or is not well distributed to build a strong SVM.

Meta-learning algorithms

Meta-learning algorithms aim to facilitate the task of choosing the right models and their parameters in order to acquire the best classifier performance for our dataset. As mentioned before, we will work with two classes ("Popular" and "Not Popular") in this section.

Majority voting

The chosen models for the majority voting Meta Method are a Gaussian classifier, a Knn classifier with the best parameters (using GridSearchCV) and a Decision Tree classifier. In the table below we can see what each classifier's accuracy is, with the Majority Voting method's accuracy below as well as the Majority Voting using weights. We employ a cross validation with 50 splits throughout the whole section.

Classifier	Accuracy
Naive Bayes	0.634
K-NN (25 neighbors, weighted)	0.746
Decision Tree	0.734
Hard voting	0.751
Soft voting [1, 2, 2]	0.754

Figure 28: Table of classifier accuracies.

Both the K-NN and Decision Tree classifiers give us a better accuracy than the Naive Bayes, but we still obtain a better accuracy from the Majority Voting.

We then apply the same Majority Voting classifier but with 'soft' voting. As seen in the table, the Naive Bayes classifier has a lower accuracy than the other two and thus we give it half as much weight in the voting. The other two get the same amount of weight since they provide a very similar accuracy. The final weights are then 1, 2 and 2 for Naive Bayes, K-NN and Decision Tree, respectively. With this new classifier we obtain a slightly improved accuracy of 0.754 which represents a mere 0.3% increase.

Bagging and Boosting

We now move on to Bagging and Boosting type qualifiers. Here, we don't specify which classifiers we couple together but simply state how many and the models do the rest for us. We can divide them into two groups, Bagging and Boosting. Here are they all listed:

- Bagging:
  - Bagging DT
  - Bagging DT with forced variance (max_features=0.35)
  - Random Forest
  - Extra Trees
- Boosting:
  - AdaBoost using Decision Stumps

– AdaBoost using Decision Trees (max_depth=5)
– Gradient Boosting

These sub-classifiers that each of the previous methods uses are called estimators, and for each of the seven methods tested we trained them with 1, 2, 5, 10, 20, 50, 100 and 200 estimators. The accuracy scoring results are presented below, with the accuracy on the vertical axis and the number of estimators on the horizontal axis.

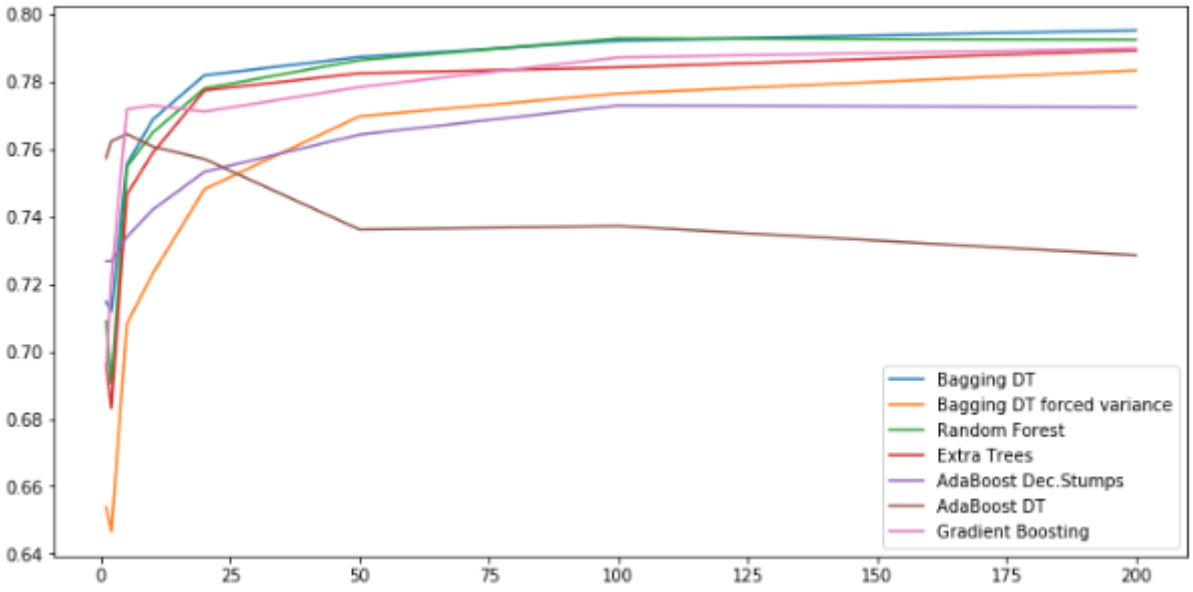


Figure 29: Accuracy scorings with different estimators and qualifiers.

There are many observations to be had from it. With a low number of estimators, Boosting methods such as Gradient Boosting or AdaBoost with Decision Trees seem to perform better, but Bagging methods catch up and surpass them with more estimators. Only with the case of Gradient Boosting we get an on par performance with some of the other Bagging methods, but it's still not the best performer. Still, the accuracy difference with 50+ estimators is generally low between methods and all of them achieve a similar performance. Furthermore, all methods plateau after 25 or 50 estimators.

The case of the AdaBoost method using Decision Trees with a depth of 5 is an interesting one. Boosting methods work by correcting mistakes made in previous training runs. So, it getting a lower accuracy with more estimators might be a signal of overfitting in the Decision Trees it uses. A depth of 5 is clearly too much for the estimators, especially seeing how better the AdaBoost using Decision Stumps performs.

Another interesting comparison to be made is that between the Bagging Decision Trees with and without forced variance. The latter, with a default value of max_features=1, clearly performs better than the former, with max_features=0.35, and is, in fact, the best overall method

In conclusion, we know Boosting methods are more sensitive to noise than Bagging, so it's not surprising to see Bagging methods perform slightly better. Another important conclusion to take away from our Meta Methods evaluation is their stabilization in accuracy after 50 or so estimators. Knowing this, in a real world example, we wouldn't advise using a large number of estimators due to their high training time, especially over a larger dataset than ours.

Our method of choice would then be a Bagging Method with Decision Trees with 50 or 100 estimators (depending on the size of the dataset).

Comparison and conclusions

After a thorough review and analysis of all the presented methods we have reached the following conclusions.

Naive Bayes is a fast and simple classifier but its results with our dataset are left to be desired. The resulting mean accuracy of the model is mediocre and it gets confused between "Popular" and "Very Popular". Since we saw from the start that the features in our dataset are not independent between each other and the results not being outstanding, we can assume that Naive Bayes is simply not the best classifier for our task.

K-NN gives us an average accuracy with the best parameters and after removing some noise, but the model's fast training speed gives it a slight edge over other classifiers.

Majority Voting doesn't present much advantage over other methods in using it with low gains in accuracy, but it's possibly it's more robust against noise and doesn't overfit as easily. As for the best meta methods such as bagging decision trees or Random Forests give good results but take very long to train. We conclude they are specially useful with a more powerful training machine or a smaller dataset.

Decision trees gives us a lot of clarity in terms of the results. It works well when it comes to reaching conclusions such as finding out that songs released in October are generally more popular by inspecting the release month attribute. It gave us a good accuracy when used with the "2 classes" dataset, so the tree could be built better as the decisions where binomial (only 2 children nodes per parent). Nevertheless, we found that every time it was executed, the model gave us a different result, which means poor robustness and overall not trustworthy.

In summary, no model performed outstandingly with our dataset, which tells us that our features weren't relevant enough or that the estimation of popularity given by Spotify may not follow a strong pattern or maybe we followed the wrong one. However, we saw that some attributes affect more to the popularity of a song. A few of them are really obvious like the artist popularity and followers, and other ones more interesting to talk about like the release month, acousticness and danceability.

Finally, we can say that the best method is the Majority Voting method using Naïve Bayes, K-NN and a Decision Tree. Thanks to its combination of different methods it achieves a good accuracy and better robustness against noise and false positives.

	Accuracy	Accuracy on cross validation	Confidence interval
KNN (3 classes)	65.68 %	61.84%	(63.86% - 67.50%)
Naïve Bayes (3 classes)	60.04%	59.81%	(58.15% - 61.93%)
Decision Trees (2 classes)	76.34%	71.5%	(74.62% - 77.91%)
SVM (3 classes)	66.07%	65.90%	
Majority voting (2 classes)	75.0%	75.40%	(66.0% - 84.0%)

Figure 30: Comparison table for accuracies and confidence intervals between classifiers

Overall there is not much difference between the accuracy of the model and the one given by the cross validation, being the latter worse.

Division of tasks

Description of the original data	Felipe
Description of pre-processing of data	Arnau
Evaluation criteria of data mining models	All
Execution of different machine learning methods	
• Naïve Bayes	Gerard
• K-NN	Bernat
• Decision Trees	Felipe y Albert
• Support Vector Machines	Felipe y Paul
• Meta-learning algorithms	Arnau
• Comparison and conclusions	All

Figure 31: Table of tasks