Supervised Learning on Top Hits Spotify from 2000-2019

1st Beatriz Gonçalves
*DETI**
University of Aveiro
Machine Learning Foundations
Prof. Petia Georgieva
(50% of work load)

2nd Tiago Nazário
*DETI**
University of Aveiro
Machine Learning Foundations
Prof. Petia Georgieva
(50% of work load)

*DETI: Department of Electronics, Telecommunications and Informatics

***Abstract*—Nowadays there's no better platform than Spotify, to listen to music. With this in mind, in this project, Top Hits from Spotify from 2000-2019 were analyzed. A database with 18 variables was studied, some of these variables being the song characteristics, two variables regarding the artist name and the song name, and one representing the popularity of the hit. The goal was to predict the popularity class within the Top Hits (classification problem). For this, the Decision Tree Classifier (Dt), K-nearest neighbors (Knn) and Logistic Regression L(Lr) models were applied. We divided the supervised learning into two parts where, in the first, the models were applied with their default parameters, obtaining an accuracy for the test set of $\simeq 56\%$ for the Dt model, $\simeq 52\%$ for the Knn model and $\simeq 48\%$ for the Lr model. In the second part, we tried to choose the best parameters for each model, obtaining an accuracy for the test set of $\simeq 52\%$ for the Dt model, $\simeq 48\%$ for the Knn model, and $\simeq 50\%$ for the Lr model.**

***Index Terms*—classification, Decision Tree Classifier (Dt), K-nearest neighbors (Knn), Logistic Regression (Lr), supervised learning, accuracy, best parameters**

I. Introduction

Music can raise someone's mood, get them excited, or make them calm and relaxed. Music also - and this is important - allows us to feel nearly or possibly all emotions that we experience in our lives. The possibilities are endless, it has been called the greatest human creation throughout history. Music serves a greater purpose in our lives than simply providing amusement. Taking that into account, our study aims to predict the popularity class within the already statistically considered the most popular, between 2000 and 2019. This project aimed to implement ML models to make this prediction - classification problem - to the data, and then proceeded to see the accuracy of our models compared to the real popularity class values.

II. Data description and Pre-processing

The database used was taken from the Kaggle platform, which contains audio statistics of the top 2000 tracks on Spotify from 2000-2019. The data contains about 18 columns each describing the track and its qualities. The variables used here can be described as follows.

A. Data description

- **Artist**: name of the artist;
- **Song**: name of the track;
- **`Duration_ms`**: duration of the track in milliseconds;
- **Explicit**: the lyrics or content of a song or a music video contain one or more of the criteria which could be - considered offensive or unsuitable for children;
- **Year**: release Year of the track;
- **Popularity**: the higher the value the more popular the song is;
- **Danceability**: describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is the least danceable and 1.0 is the most danceable;
- **Energy**: energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity;
- **Key**: the key the track is in. Integers map to pitches using standard Pitch Class notation. If no key was detected, the value is -1;
- **Loudness**: the overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing the relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 dB;
- **Mode**: indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0;
- **Speechiness**: detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks;
- **Acousticness**: a confidence measure from 0.0 to 1.0

of whether the track is acoustic. 1.0 represents high confidence the track is acoustic;
- **Instrumentalness**: predicts whether a track contains no vocals. ”Ooh” and ”aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly ”vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0;
- **Liveness**: detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides a strong likelihood that the track is live;
- **Valence**: a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry);
- **Tempo**: the overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration;
- **Genre**: genre of the track.

B. Preprocessing

We will use this dataset [1] for our test/train model. Before any attempt at models' implementation, we need to do some data analysis and pre-processing. We start by doing some data cleaning. It is one of the most important steps to achieve the best from the dataset. This is a process whereby data inconsistencies such as missing values, unformatted data, and noise are removed from the data.

Our first step was to select information. Since our goal was, within those listed top hits, to predict which ones are most popular and least popular, we selected the most and the least important information for this prediction. Analyzing the meaning of each column, already described above, we decided that the ”key” column has no influence on the popularity prediction, it seemed to be a too-specific feature that wasn't affecting our prediction. Second, it is a feature that makes no sense to be normalized since it acts as an ”id” for the ”key”. In that case, it would end up influencing our models, having more weight in them than what we want. The ”song” column was dropped also. Whereas the name of the song has no influence on our prediction, it also works as a kind of ”id”.

Our dataset had information about songs from years before 2000 and after 2019. This data has been removed as well.

We then proceeded to verify the existence of null values and repeated rows, dropping them.

The duration of the song that was displayed in milliseconds was converted to minutes, for better readability.

After analyzing our raw data, we realized that there were 22 entries with an undefined genre, ”set()”. To solve this problem we searched for the songs that had ”set()” as a genre and entered, by hand (since they were not many) the respective genre we found.

Our dependent variable, ”popularity”, was categorized into ”low/medium” and ”high”. This will turn the problem into a classification problem instead of a regression problem. There is no gain in predicting the exact numerical value of the population rating. This way, we added a new column, `"popularity_class"`, based on the popularity values. It was assigned the category ”low/medium” to values less than sixty-five and ”high” to values higher than sixty-five. To classify the categories we assigned the category ”low/medium” a value of 0 and the category ”high”, a value of 1, in order to streamline future exploratory analysis. After this, we dropped the ”popularity” column, so it could be replaced with the new `"popularity_class"`. It is important to note that the data, with this categorization, was divided into approximately equal portions. This is good because we won't have too little information for ”low/medium”, and too much for ”high”, for example (or the other way around). This allows our training model not to be ”influenced” by the category that has the most information.

Because they are strings, the features ”genre” and ”artist” have been encoded to be able to include them in our models. The encoding was done as follows: we added the song and artists' names as columns; when a song is of a particular genre, it returns the value 1 for the corresponding genre and the value 0 for the others. The same happens for the ”artist” feature. The ”explicit” variable is a Boolean, this way, it was encoded to True the value 1 and to False the value 0.

Our features are measured at different scales. This way, we have to normalize our data. This is because scales with higher numbers end up having more importance in our models, while smaller numbers end up having less, and this will end up misleading our models. Since it doesn't make sense to normalize the year, we will create a new column called ”age”, where we will put the ”age” of the song, that is, the current year (2022) minus the year it was released. Thus, it already makes sense for us to normalize the ”age” column later. Columns that are already classified as 0 and 1 also make no sense to be normalized as mode, all genres columns, and all artists columns. Furthermore, as the dataset legend says that for the variable ”liveness”, from 0.8 there is a higher probability that the song is live, we will also assign a classification to ”liveness”, that is, for a measure greater than 0.8 we assign the value 1 to ”liveness”, otherwise the value 0. Now we can add the variable ”liveness” to the list of features that are already classified and do not make sense to normalize.

We will then normalize the features: 'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'valence', 'time', `'duration_min'` and 'age'.

As we already know prediction of the `"popularity_class"` column was the goal. Therefore, we don't want to train this column, we want that column to be our predictable one. This way, before the models were applied, we dropped and saved the column we wanted to predict: `"popularity_class"`.

To ensure that any 'data snooping' bias was committed, it was set aside 20% for test data. We observed that the proportion of the `"popularity_class"` remained the same (approximately the same) as before. That is, the proportion is maintained in the training and test dataset.

III. Data visualization

Now, we will provide some different types of graphics and plots done before the categorization and normalization of our features, for a user-friendly comprehension of the data to study.

A. Correlation Matrix

We start by analyzing the correlation matrix of the dataset. It is a symmetric matrix and it indicates whether or not exists any correlation between all the possible pairs of values in a table according to its coefficient.

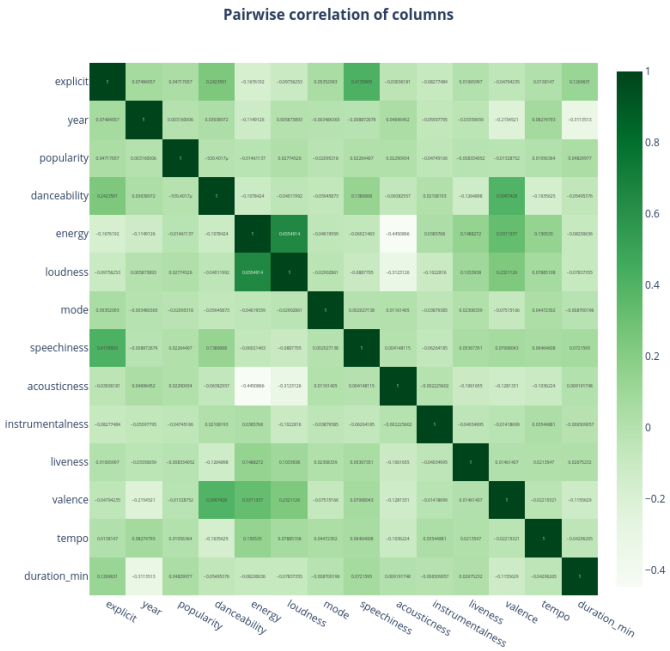

Fig. 1. Correlation matrix of our features.

Here we only see the correlation of inputs that are not strings (Boolean, float, and int). Thus, columns like "artist" and "gender" do not appear in the correlation matrix. Observing the map, the two higher positive correlations are visible between "loudness" and "energy" - the louder the song, the more energetic it is - and "speechiness" and "explicit" - the more spoken words are the song, the more explicit it is. The two lowest correlations are visible between "energy and "acousticness" - acoustic songs have less energy - and "year" and "duration" - songs tend to become shorter the more recent they are.

B. Features' Distribution

We analyzed the distribution of all our features (again, all non-strings), to see some of their behavior. The distribution are represented in Fig.31.

For the **"year"** feature, we do a count of how many samples, i.e. how many songs we have per year. The sum of the total samples is 1899, just like the number of rows in our dataset. We can see that the year with the most songs listed was 2012 and the year with the least number of songs listed was 2000. For the **"popularity"** feature, we have the popularity value on the x-axis and the sum of entries with the same popularity on the y-axis. We have a larger number of songs with a popularity equal to 0. But we can also see that we have a higher concentration of songs with popularity between 60 and 80. For the **"danceability"** feature, on the x-axis we have the measures for "danceability" and on the y-axis a sum of the number of songs with the same measure. We see a higher concentration of songs with danceability between 0.65 and 0.75, approximately. For the **"energy"** feature, on the x-axis, we have the energy measure, and on the y-axis the sum of the number of songs for each measure. We can see that there is a higher concentration for energy between 0.6 and 0.9, approximately. For the **"loudness"** feature, on the x-axis, we have the loudness measures, and on the y-axis the sum of songs with the same loudness measure. We see that we have a larger number of songs with a loudness of -5, approximately. For the **"mode"** feature, on the x-axis, we have the mode measures, which vary only between 0 and 1, and on the y-axis the sum of songs with the same mode measure. As we only have two measures, we can easily see that there are more songs with mode measure equal to 1 (major). For the **"speechiness"** feature, on the x-axis, we have the speechiness measures, and on the y-axis, we have the sum of the songs with the same speechiness measure. We found a higher number of songs with speechiness between 0 and 0.1, approximately. For the **"acousticness"** feature, on the x-axis, we have the acousticness measures, and on the y-axis, we have the sum of the songs with the same acousticness measure. We found a higher number of songs with acousticness equal to 0. For the **"instrumentalness"** feature, on the x-axis, we have the instrumentalness measures, and on the y-axis, we have the sum of the songs with the same measure. As we saw above, instrumentalness is measured

between 0 and 1. However, we see that most of our measures are concentrated in the 0 region because most songs have very small instrumentalness measures. For the **"liveness"** feature, on the x-axis, we have the liveness measures, and on the y-axis, we have the sum of the songs with the same measure. We can see that the measures are concentrated approximately between 0 and 0.2. For the **"valence"** feature, on the x-axis, we have the valence measures, and on the y-axis, we have the sum of the songs with the same measure. We can see that the measures are concentrated approximately between 0.4 and 0.6. For the **"time"** feature, on the x-axis, we have the time measures, and on the y-axis, we have the sum of the songs with the same measure. We can see that the measures are concentrated approximately between 120 BPM and 130 BPM, approximately. And, finally, for the **`duration_min`** feature, on the x-axis, we have the duration measurements, and on the y-axis, we have the sum of the songs with the same duration. We can see that the measures are concentrated, approximately, between 3 and 4 minutes.

C. Correlation distribution

We now distribute each of the features we saw earlier, with higher and lower correlation as a function of "popularity", that is, we distribute "energy", "loudness", "speechiness", "explicit", "acousticness", "year" and "duration" as a function of "popularity".

1) Energy Vs. Popularity: As we can see in Fig.32, we see a greater population concentration of songs with high popularity and high energy value.

2) Loudness Vs. Popularity: As we can see in Fig.33, we see a greater population concentration of songs with high loudness and high popularity.

3) Speechiness Vs. Popularity: We can see in Fig.34 that we have more songs with lower speechiness and high popularity, since we have a higher population concentration when speechiness is lower. However, we can also see that a song is unlikely to have high speechiness and have low popularity.

4) Explicit Vs. Popularity: We chose a boxplot, for Fig.35 because the "explicit" feature can only be true or false, and makes it easier to see the distribution for each of the cases. We can see that the distribution of both (explicit and non-explicit), in terms of popularity, is very similar.

5) Acousticness Vs. Popularity: In Fig.36, we have a higher population concentration of songs with lower acousticness and high popularity.

6) Yearwise Popularity: Averaging the popularity values by year we see, in Fig.37, that there were musics with the most popularity in 2018 and the least in 2013.

7) Duration Vs. Popularity: In Fig.38 we see that there is a higher population concentration of songs with a duration between approximately 3 min and 4 min and with high popularity.

IV. Implemented Machine Learning Models

After having pre-processed the data we proceeded to the implementation of ML Models. For our solution, we implemented the three algorithms below.

A. Decision Trees Classifier

As the name implies, Decision Tree [8] builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. The leaf node represents a classification or decision. The topmost decision node in a tree that corresponds to the best predictor is called the root node. Decision trees can handle both categorical and numerical data.

B. K-Nearest Neighbours

This model [9] uses 'feature similarity' to predict the values of any new data points. This means that, to the new point, it is assigned a value based on how closely it resembles the points in the training set. So, kNN classifier determines the class of a data point by majority voting principle. If k is set to 5, the classes of 5 closest points are checked. Prediction is done according to the majority class.

C. Logistic Regression

In practice, the logistic regression algorithm [10] analyzes relationships between variables. It assigns probabilities to discrete outcomes using the Sigmoid function, which converts numerical results into an expression of probability between 0 and 1.0. Probability is either 0 or 1, depending on whether the event happens or not.

V. Application and analysis of the ML models

The application of our models was divided into two parts. In the first part, we applied the models using the default values, that is, without setting the hyper-parameters. In the second part, we chose the best hyper-parameters for our models and see if there were any improvements in their performance.
For each model we apply, we also do k-fold cross-validation [10]. In addition, we present a classification report/ performance metrics [11], a confusion matrix [8], and a ROC curve.

K-fold Cross-Validation is when the dataset is split into a K number of folds and is used to evaluate the model's ability when given new data. K refers to the number of groups the data sample is split into. For example, if you see that the k-value is 5, we can call this a 5-fold cross-validation. Each fold is used as a testing set at one point in the process. The process is the following: we choose your k-value, split the dataset into the number of k folds, start off with using your k-1 fold as the test dataset and the remaining folds

as the training dataset. Then we train the model on the training dataset and validate it on the test dataset and save the validation score. We keep repeating steps 3 – 5 but changing the value of your k test dataset. So we chose k-1 as our test dataset for the first round, we then move onto k-2 as the test dataset for the next round. By the end of it we would have validated the model on every fold that you have. Finally, we just average the results that were produced in step 5 to summarize the skill of the model.

For the classification report, we obtained chose the performance metrics:

- **Precision**: tells us, out of all the positive predicted, what percentage is truly positive. The precision value lies between 0 and 1;
- **Recall**: tells us, out of the total positive, what percentage are predicted as positive. It is the same as TPR (true positive rate);
- **F1-score**: that is the harmonic mean of precision and recall. It takes both false positive and false negatives into account. Therefore, it performs well on an imbalanced dataset.

Precision, recall, and F1 score are all broken down by class, and then a macro average and weighted average are given for each.

- **Macro average**: is the usual average we're used to seeing. Just add them all up and divide by how many there were;
- **Weighted average**: considers how many of each class there were in its calculation, so fewer of one class means that it's precision/recall/F1 score has less of an impact on the weighted average for each of those things.
- **Support:**: tells how many of each class there were.

The confusion matrix represents, on the principal diagonal, the true positives and true negatives, that is, the values that the model predicted as having high or low popularity and actually are. The remaining two values are false positives or false negatives, that is, cases where the model predicted as having high popularity but has low popularity and vice versa.

Finally, we also wanted to analyze the Receiver Operating characteristic curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection.The false-positive rate is also known as probability of false alarm and can be calculated as (1 specificity). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR).

A. Using default values

In this first part of our algorithm implementation, we used the default values of each algorithm classifier. Then we compared the results with the before saved `"popularity_class"` column.

1) Decision Tree Classifier: Here we present the decision tree model where we obtained these results regarding accuracy:

```
Accuracy for training set:
 0.9980250164581962


Accuracy for testing set:
 0.5447368421052632
```

Fig. 2. Accuracy Results for Decision Tree Classifier.

Classification Report:

```
Classification report:
              precision    recall  f1-score   support

           0       0.51      0.58      0.54       177
           1       0.58      0.51      0.55       203

    accuracy                           0.54       380
   macro avg       0.55      0.55      0.54       380
weighted avg       0.55      0.54      0.54       380
```

Fig. 3. Classification Report Decision Tree Classifier.

Confusion Matrix:

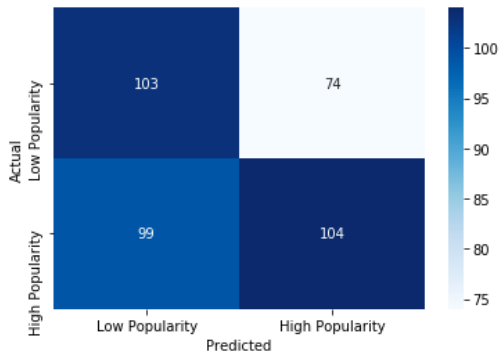

Fig. 4. 1st Confusion Matrix Decision Tree.

ROC curve:

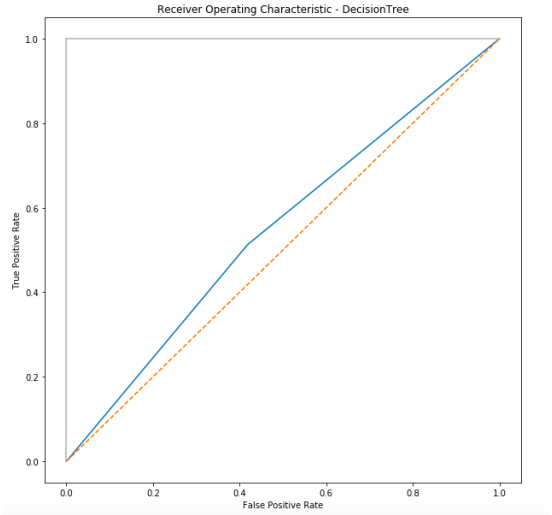

Fig. 5. ROC Curve Decision Tree.

K-fold Cross Validation:

```
Fold:  1, Training/Test Split Distribution: [699 668], Accuracy: 0.599
Fold:  2, Training/Test Split Distribution: [699 668], Accuracy: 0.645
Fold:  3, Training/Test Split Distribution: [699 668], Accuracy: 0.586
Fold:  4, Training/Test Split Distribution: [699 668], Accuracy: 0.559
Fold:  5, Training/Test Split Distribution: [699 668], Accuracy: 0.533
Fold:  6, Training/Test Split Distribution: [699 668], Accuracy: 0.559
Fold:  7, Training/Test Split Distribution: [699 668], Accuracy: 0.533
Fold:  8, Training/Test Split Distribution: [700 667], Accuracy: 0.592
Fold:  9, Training/Test Split Distribution: [700 667], Accuracy: 0.638
Fold: 10, Training/Test Split Distribution: [700 668], Accuracy: 0.563

Cross-Validation mean accuracy: 0.581 +/- 0.037

Cross-Validation top accuracy: 0.645
```

Fig. 6. K-fold Cross Validation Decision Tree.

Using K-fold cross-validation we get an accuracy of $\simeq 58\%$, as we can see in Fig.6.

To better our accuracy, we went on to analyze which features had the most relevance and the most weight in our model. For better accuracy, it made sense to drop the features with less importance. So we decided to test the model only with features whose importance is higher than 0.01. Aplicando, de novo, o modelo ao nosso dataset após dar drop destas colunas, obtemos as accuracies dadas pela Fig.7

```
Accuracy for training set:
 0.9980250164581962


Accuracy for testing set:
 0.5631578947368421
```

Fig. 7. Accuracy of Decision Tree Classifier for transformed dataset.

As we can see, dropping those columns did improve our model, but very little, so we're going to keep the dataset before the "drop".

Clearly, analyzing the accuracies we have obtained in Fig.2 and Fig.7, we are facing a case of overfitting for this model. This happens when the model completely fits the training data but fails to generalize the testing unseen data. Overfit condition arises when the model memorizes the noise of the training data and fails to capture important patterns. There are various techniques to prevent the decision tree model from overfitting, we will use "pruning" [13]. By default, the decision tree model is allowed to grow to its full depth. Pruning refers to a technique to remove the parts of the decision tree to prevent growing to its full depth. By tuning the hyperparameters of the decision tree model one can prune the trees and prevent them from overfitting. There are two types of pruning Pre-pruning and Post-pruning. Pre-pruning is what we will do in the second part that we talked about above, we will clarify this one in more detail later. So, here, we will apply post-pruning. It allows the decision tree model to grow to its full depth, then removes the tree branches to prevent the model from overfitting. Cost complexity pruning (ccp) is one type of post-pruning technique. In case of cost complexity pruning, the ccp.alpha can be tuned to get the best fit model. Scikit-learn package comes with the implementation to compute the ccp.alpha values of the decision tree using function cost.complexity.pruning.path(). With the increase in ccp.apha values, more nodes of the tree are pruned.

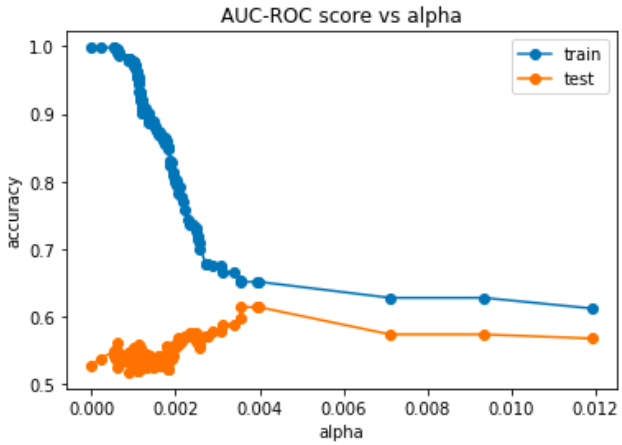

Fig. 8. AUC-ROC score vs alpha.

From the above plot, Fig.8 ccp.alpha can be considered as the best parameter when it is the same for train and test, as AUC-ROC (Area Under the ROC Curve) scores for train and test are the correspondent ordinates.

2) K-Nearest Neighbours: As in the Decision Tree model, we are going to show the results of the application of this model to our data.

Accuracy results:

```
Accuracy for training set:
 0.7379855167873601


Accuracy for test set:
 0.5184210526315789
```

Fig. 9. Accuracy Results K-Nearest Neighbours.

Classification report:

```
Classification report:
              precision    recall  f1-score   support

           0       0.49      0.58      0.53       177
           1       0.56      0.46      0.51       203

    accuracy                           0.52       380
   macro avg       0.52      0.52      0.52       380
weighted avg       0.53      0.52      0.52       380
```

Fig. 10. Classification Report K-Nearest Neighbours.

Confusion Matrix:

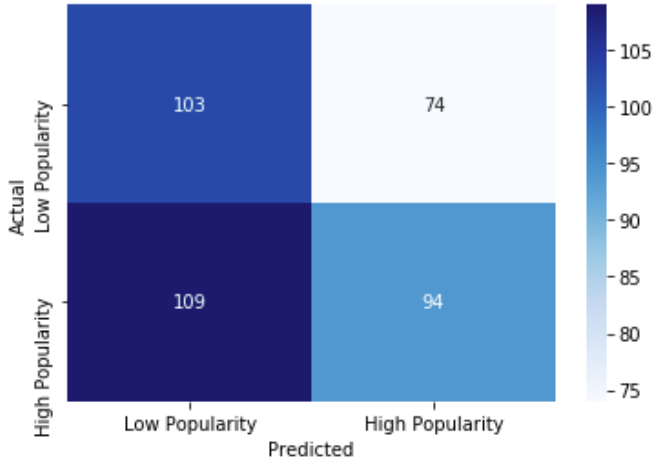

Fig. 11. Confusion Matrix K-Nearest Neighbours.

ROC curve:

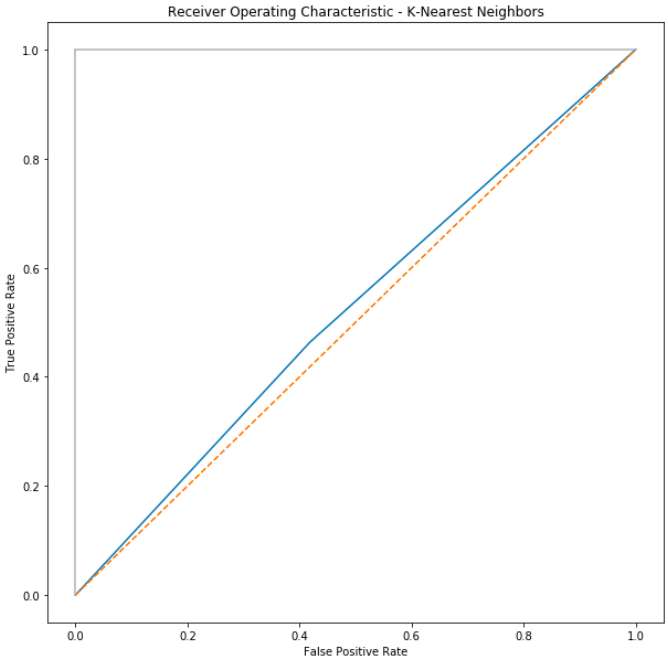

Fig. 12. ROC Curve K-Nearest Neighbours.

K-fold cross-validation:

```
Fold:  1, Training/Test Split Distribution: [699 668], Accuracy: 0.559
Fold:  2, Training/Test Split Distribution: [699 668], Accuracy: 0.546
Fold:  3, Training/Test Split Distribution: [699 668], Accuracy: 0.645
Fold:  4, Training/Test Split Distribution: [699 668], Accuracy: 0.586
Fold:  5, Training/Test Split Distribution: [699 668], Accuracy: 0.566
Fold:  6, Training/Test Split Distribution: [699 668], Accuracy: 0.553
Fold:  7, Training/Test Split Distribution: [699 668], Accuracy: 0.579
Fold:  8, Training/Test Split Distribution: [700 667], Accuracy: 0.618
Fold:  9, Training/Test Split Distribution: [700 667], Accuracy: 0.618
Fold: 10, Training/Test Split Distribution: [700 668], Accuracy: 0.523

Cross-Validation accuracy: 0.579 +/- 0.036

Cross-Validation top accuracy: 0.645
```

Fig. 13. K-fold Cross Validation K-Nearest Neighbours.

After K-fold cross-validation we got an accuracy of $\simeq 58\%$.

One technique to improve the accuracy of this model is by scaling our features, rather than normalizing them. Normalization adjusts the values of your numeric data to a common scale without changing the range whereas scaling shrinks or stretches the data to fit within a specific range. Unfortunately by doing this, we did improve, but very little. So, we kept the previous dataset (with normalization instead of scalling).

3) Logistic Regression: For the Logistic Regression model, we got the following results.

Accuracy results:

```
Accuracy for training set:
  0.5220539828834759


Accuracy for test set:
  0.4842105263157895
```

Fig. 14. Accuracy Results Logistic Regression.

Classification Report:

```
Classification report:
              precision    recall  f1-score   support

           0       0.48      0.69      0.56       183
           1       0.50      0.29      0.37       197

    accuracy                           0.48       380
   macro avg       0.49      0.49      0.47       380
weighted avg       0.49      0.48      0.46       380
```

Fig. 15. Classification Report Logistic Regression.

Confusion Matrix:

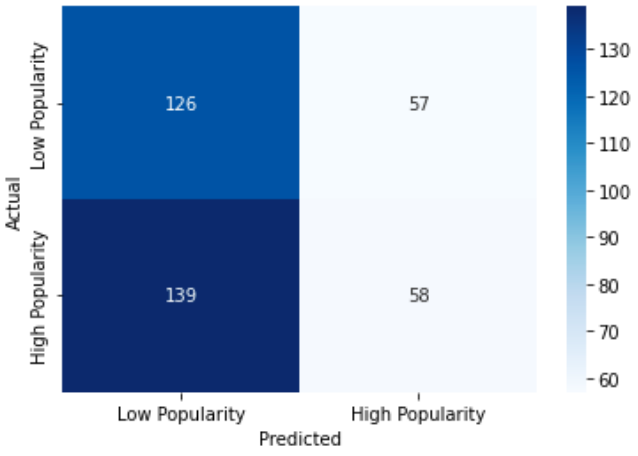

Fig. 16. Confusion Matrix Logistic Regression

ROC curve:

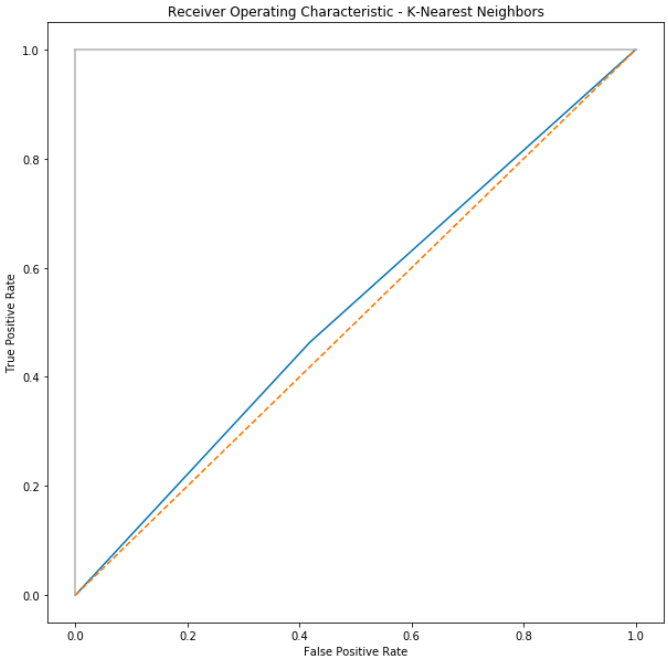

Fig. 17. ROC Curve Logistic Regression.

K-fold cross-validation:

```
Fold:  1, Training/Test Split Distribution: [694 673], Accuracy: 0.428
Fold:  2, Training/Test Split Distribution: [694 673], Accuracy: 0.487
Fold:  3, Training/Test Split Distribution: [694 673], Accuracy: 0.474
Fold:  4, Training/Test Split Distribution: [694 673], Accuracy: 0.447
Fold:  5, Training/Test Split Distribution: [694 673], Accuracy: 0.513
Fold:  6, Training/Test Split Distribution: [694 673], Accuracy: 0.553
Fold:  7, Training/Test Split Distribution: [694 673], Accuracy: 0.500
Fold:  8, Training/Test Split Distribution: [694 673], Accuracy: 0.447
Fold:  9, Training/Test Split Distribution: [693 674], Accuracy: 0.447
Fold: 10, Training/Test Split Distribution: [694 674], Accuracy: 0.457


Cross-Validation accuracy: 0.475 +/- 0.036

Cross-Validation top accuracy: 0.553
```

Fig. 18. K-fold Cross Validation Logistic Regression.

As we can see in Fig.18, after applying the K-fold cross-validation we got an accuracy of $\simeq 48\%$.

4) Algorithm Comparison and Voting Classifier: We can see, in Fig.19, a comparison of the three models' accuracy.

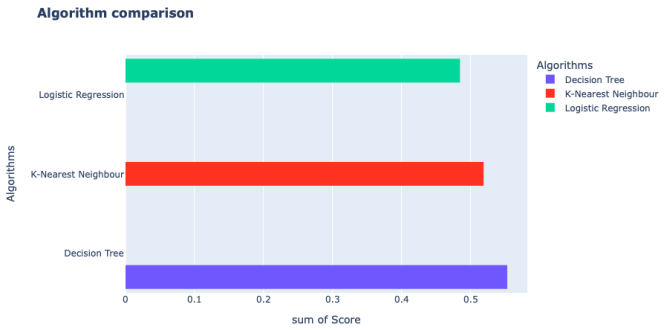

Fig. 19. Models Comparison using default hyper-parameters.

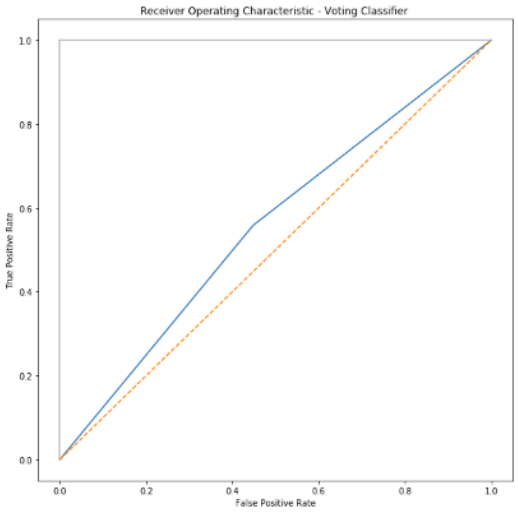

Fig. 21. ROC Curve for Voting Classifier.

Then, we used the Voting Classifier to train the 3 models. A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output based on their highest probability of chosen class as the output. It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class. Using this method, we got an accuracy of $\simeq 56\%$.

Applying the K-fold and ROC curve:

```
Fold:  1, Training/Test Split Distribution: [694 673], Accuracy: 0.461
Fold:  2, Training/Test Split Distribution: [694 673], Accuracy: 0.467
Fold:  3, Training/Test Split Distribution: [694 673], Accuracy: 0.480
Fold:  4, Training/Test Split Distribution: [694 673], Accuracy: 0.467
Fold:  5, Training/Test Split Distribution: [694 673], Accuracy: 0.428
Fold:  6, Training/Test Split Distribution: [694 673], Accuracy: 0.546
Fold:  7, Training/Test Split Distribution: [694 673], Accuracy: 0.480
Fold:  8, Training/Test Split Distribution: [694 673], Accuracy: 0.493
Fold:  9, Training/Test Split Distribution: [693 674], Accuracy: 0.395
Fold: 10, Training/Test Split Distribution: [694 674], Accuracy: 0.477

Cross-Validation accuracy: 0.469 +/- 0.038

Cross-Validation top accuracy: 0.546
```

Fig. 20. K-fold cross-validation for Voting Classifier.

For the ROC curve:

B. Parameter Tuning using GridSearchCV

In this second part of our algorithm implementation, we implemented GridSearch for the algorithms. This means that, for each algorithm used, we define a group of parameters and run all the algorithms with those parameters. Then we can see which is the best parameter combination by choosing the one with the best score.

1) Decision Trees: The parameters [8] tested were:

- **criterion**: function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “log.loss” and “entropy” both for the Shannon information gain;
- **splitter**: strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split;
- **max.depth**: maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min.samples.split samples;
- **max.features**: number of features to consider when looking for the best split.

For accuracy, best parameters and best estimator we got the results of Fig.39.

For the confusion matrix:

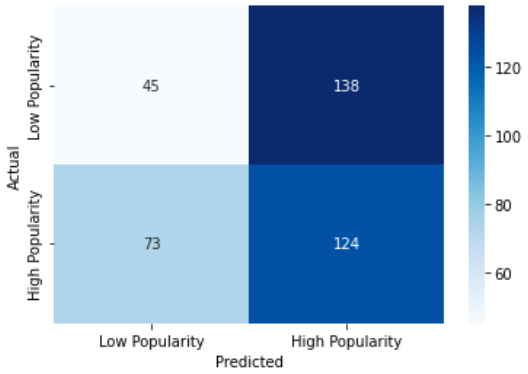

Fig. 22. Confusion Matrix Decision Tree using GridSearchCV.

ROC curve:

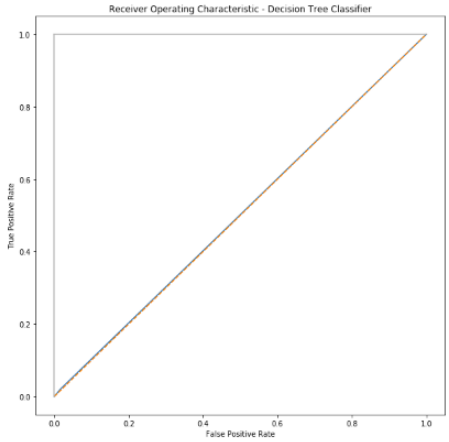

Fig. 23. ROC Curve Decision Tree Classifier using GridSearchCV.

To choose the hyperparameters for our model, we plotted the performance as a function of the hyperparameters we tested, and also took into account the best time we could get. We represented our results in Fig.40.

From this, we can conclude that the best accuracy we had was using the default parameters.

2) K-NN: The parameters tested were:

- **n.neighbors**: number of neighbors to use by default for k-neighbors queries.
- **n.neighbors**: weight function used in prediction. Possible values:
  'uniform': uniform weights. All points in each neighborhood are weighted equally.
  'distance': weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  [callable]: a user-defined function that accepts an array of distances, and returns an array of the same shape containing the weights.
- **algorithm**:algorithm used to compute the nearest neighbors: 'ball.tree' will use BallTree; 'kd.tree' will use KDTree; 'brute' will use a brute-force search.

We can see the results for the best score, best parameters, best estimator, and classification report in Fig.41.

For the confusion matrix:

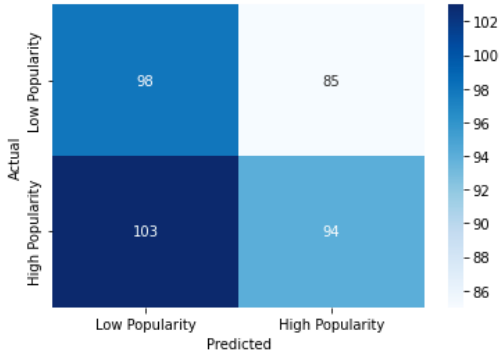

Fig. 24. 2st Confusion Matrix K-NN

ROC curve:

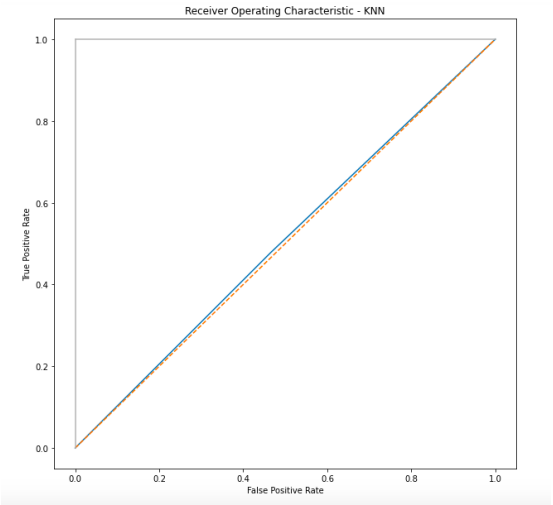

Fig. 25. ROC Curve K-NN using GridSearchCV.

The plots for choosing the best hyper-parameters are given in Fig.42.

3) Logistic Regression: The parameters tested were: - C:Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
penalty: Specify the norm of the penalty: - 'none': no penalty is added;
- 'l2': add a L2 penalty term and it is the default choice;
- 'l1': add a L1 penalty term;
- 'elasticnet': both L1 and L2 penalty terms are added.
- solver: Algorithm to use in the optimization problem.

We got for the accuracy, best parameters and best estimator we got the result of Fig.43.

For the confusion matrix:

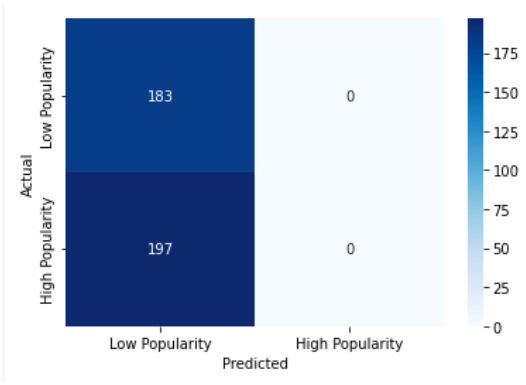

Fig. 26. 2st Confusion Matrix Logistic Regression

And we got our performance per hyper-parameter in Fig.44.

We conclude that the best results were obtained using the default parameters.

4) Comparison of the accuracy obtained by our models: For the comparison of the accuracy of the 3 models we got:

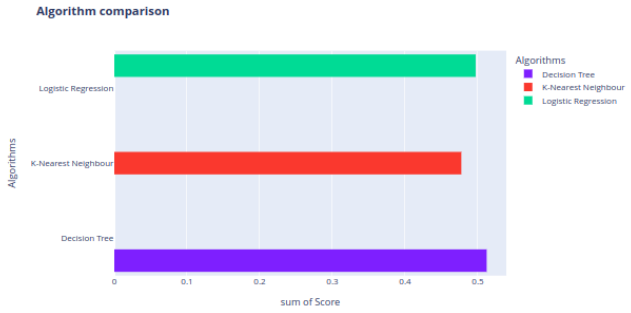

Fig. 27. Comparison between the models.

5) Voting Classifier: For the voting classifier model we got the following results. Accuracy:

```
Final Accuracy Score
0.5473684210526316
```

Fig. 28. Accuracy voting classifier using GridSearchCV.

K-fold cross-validation:

```
Fold:  1, Training/Test Split Distribution: [694 673], Accuracy: 0.447
Fold:  2, Training/Test Split Distribution: [694 673], Accuracy: 0.474
Fold:  3, Training/Test Split Distribution: [694 673], Accuracy: 0.520
Fold:  4, Training/Test Split Distribution: [694 673], Accuracy: 0.474
Fold:  5, Training/Test Split Distribution: [694 673], Accuracy: 0.401
Fold:  6, Training/Test Split Distribution: [694 673], Accuracy: 0.500
Fold:  7, Training/Test Split Distribution: [694 673], Accuracy: 0.487
Fold:  8, Training/Test Split Distribution: [694 673], Accuracy: 0.533
Fold:  9, Training/Test Split Distribution: [693 674], Accuracy: 0.388
Fold: 10, Training/Test Split Distribution: [694 674], Accuracy: 0.464

Cross-Validation accuracy: 0.469 +/- 0.044

Cross-Validation top accuracy: 0.533
```

Fig. 29. K-fold voting classifier using GridSearchCV.

ROC curve:

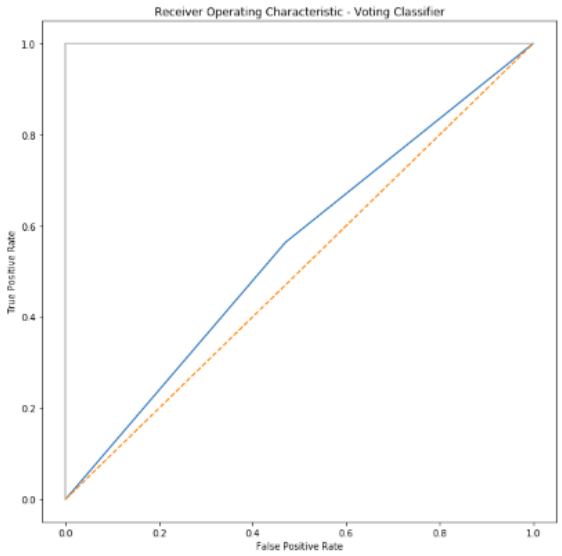

Fig. 30. ROC curve voting classifier using GridSearchCV.

Novelty and Contributions

After performing all cleaning and pre-processing, data analysis and evaluation and lastly running some models to predict the popularity we made a research on other assignments. As shown below, we can see all the improvements we would make to another project as well as comparisons with our implemented features. These notebooks were extracted from the same website as our dataset.

Example 1 [14]

Data Cleaning and Pre-Processing: This user dropped the columns 'artist' and 'song'. Event though we have dropped the column 'song' too, deleting 'artist' is definitely something that we would not do. Firstly, our basic knowledge in the area indicates that are some major names in the industry of music that can easily reach top hits on spotify and secondly this statement is proved by some of our statistics presented before. Another thing that we would improve is to implement normalization (as we have in our notebook). The given dataset has a lot of numerical columns with very different powers, for example year values are in the thousands while instrumentalness, itself, varies from thousandth to millionth. This improperly normalized condition can result in extensive data redundancy and subsequent poor system performance to inaccurate data.

Data Analysis and Evaluation: In this example there isn't any type of data analysis, only a simple plot related to missing data. This is totally a bad practice because by doing this we have zero understanding on how data behaves which is very important to take action on how to manipulate data to improve accuracy.

Model Training: The main problem in this model training is that it only uses classifiers default values. There isn't any hyper-parameter accuracy variation testing and those parameters can have different accuracy scores which may mean an improve on our model. The percentage of data used for testing is default and isn't tested any other percentage, only that default.

Example 2 [15]

In this example the variable to predict is different from our. Instead of predicting 'popularity' they predicted 'danceability' but as the dataset is the same we decided to compare it too.

Data Cleaning and Pre-Processing: In this notebook there is no sign of treatment of duplicated data which can lead to misleading data when calculating accuracy. Although it is visible and even used in some code samples, the value 'set()' as a genre it is incorrect. However, this mistake is not corrected.

Data Analysis and Evaluation: Very few data analysis. As the same as the previous example, the lack of information on our dataset leads to less data manipulation and subsequent lower probability to have a better score.

Model Training: In this model training there is only one classifier used. Different algorithms have different behaviours which lead to different results. Focus only on one classifier might not be the better solution at all, but since the title of this notebook is 'Spotify SVM Analysis w/ Gridsearch', let's assume that the goal of it was only to implement this classifier. But again different approaches lead to different results, they can be better or worse, but that's what the testing is all about, to improve our model. However very hyper-parameters and types of SMV's are tested which seems to be good.

Example 3 [16]

In this example the variable to predict is different from our. Instead of predicting 'popularity' they predicted 'danceability' but as the dataset is the same we decided to compare it too. By far, over the 3 notebooks found, this was the most complete of all 3 and so the one that required our most attention for comparison and analysi

Data Cleaning and Pre-Processing: It seems that duplicated values are not handled. All columns are consider for evaluation. Can lead to over fitting and misleading if results due to use of meaningless features

Data Analysis and Evaluation: A lot of information of each table is printed which is good, however the easiest way to analyze data is using plots which are not much in this assignment.

Model Training: Cross validation is used, different types of algorithms are used and grid search is also present. The only thing that we would do different would be the number of parameters used in the grid search which seems to be low.

Conclusions

To sum up, we have covered almost every required criteria content. Some of them were a little bit more difficult than others, but after doing some web research on specific examples, we were then able to apply it in our assignment. Only the cost function turned out to be the most difficult at all, specially applying it to the GridSearch. Apart from that, everything is working as expected. All data cleaning and pre-processing done revealed important to improve model accuracy, our implemented plots and graphics were a major assistance to decide how to manipulate data and the variety of models and how we applied also allowed us, not only to expand our knowledge on the topics, but also to increase results score. Then, we believe that our accuracy can be increased even more. A small dataset is likely the cause of the low performance in some of the models as the original dataset is biased towards popular songs in general. Some things that we would do differently were trying to increase the size of the dataset with more and more data, for example appending similar datasets or performing some Web Scrapping or accessing Spotify API. In the absence of doing this something to try would be splitting the dataset into smaller ones divided by 'genres' to reduce multilabel without increasing the amount of data.

References

[1] Dataset https://www.kaggle.com/datasets/paradisejoy/top-hits-spotify-from-20002019 Last accessed 14 November 2022.

[2] Kaggle Datasets https://www.kaggle.com/datasets. Last accessed 10 November.

[3] Pandas User Guide https://pandas.pydata.org/docs/user_guide/index.html. Last accessed 12 November 2022.

[4] Numpy https://numpy.org/. Last accessed 12 November 2022.

[5] Plotly https://plotly.com/. Last accessed 12 November 2022.

[6] Matplotlib https://matplotlib.org/. Last accessed 12 November 2022.

[7] Scikit-Learn https://scikit-learn.org/stable/. Last accessed 12 November 2022.

[8] Decision Tree Algorithm https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html Last accessed 12 November 2022.

[9] Nearest Neighbors Algorithm https://towardsdatascience.com/k-nearest-neighbors-knn-explained-cbc31849a7e3. Last accessed 14 November 2022.

[10] Logistic Regression Algorithm https://www.kdnuggets.com/2022/07/logistic-regression-work.html. Last accessed 14 November 2022.

[10] K Fold Cross Validation https://www.kdnuggets.com/2022/07/kfold-cross-validation.html. Last accessed 14 November 2022.

[11] Performance Metrics https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_algorithms_performance_metrics.htm. Last accessed 12 November 2022.

[12] Confusion Matrix https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. Last accessed 12 November 2022.

[13] Pruning https://towardsdatascience.com/3-techniques-to-avoid-overfitting-of-decision-trees-1e7d3d985a09 Last accessed 12 November 2022.
[14] Solution Example 1 https://www.kaggle.com/code/merencelebi/spotify-music-popularity-prediction. Last accessed 15 November 2022.
[15] Solution Example 2 https://www.kaggle.com/code/chancev/spotify-svm-analysis-w-gridsearch. Last accessed 15 November 2022.
[16] Solution Example 3 https://www.kaggle.com/code/vrenganathan/top-hits-spotify-popularity-prediction-ml-models. Last accessed 15 November 2022.

APPENDIX

A. *Features' Distribution*

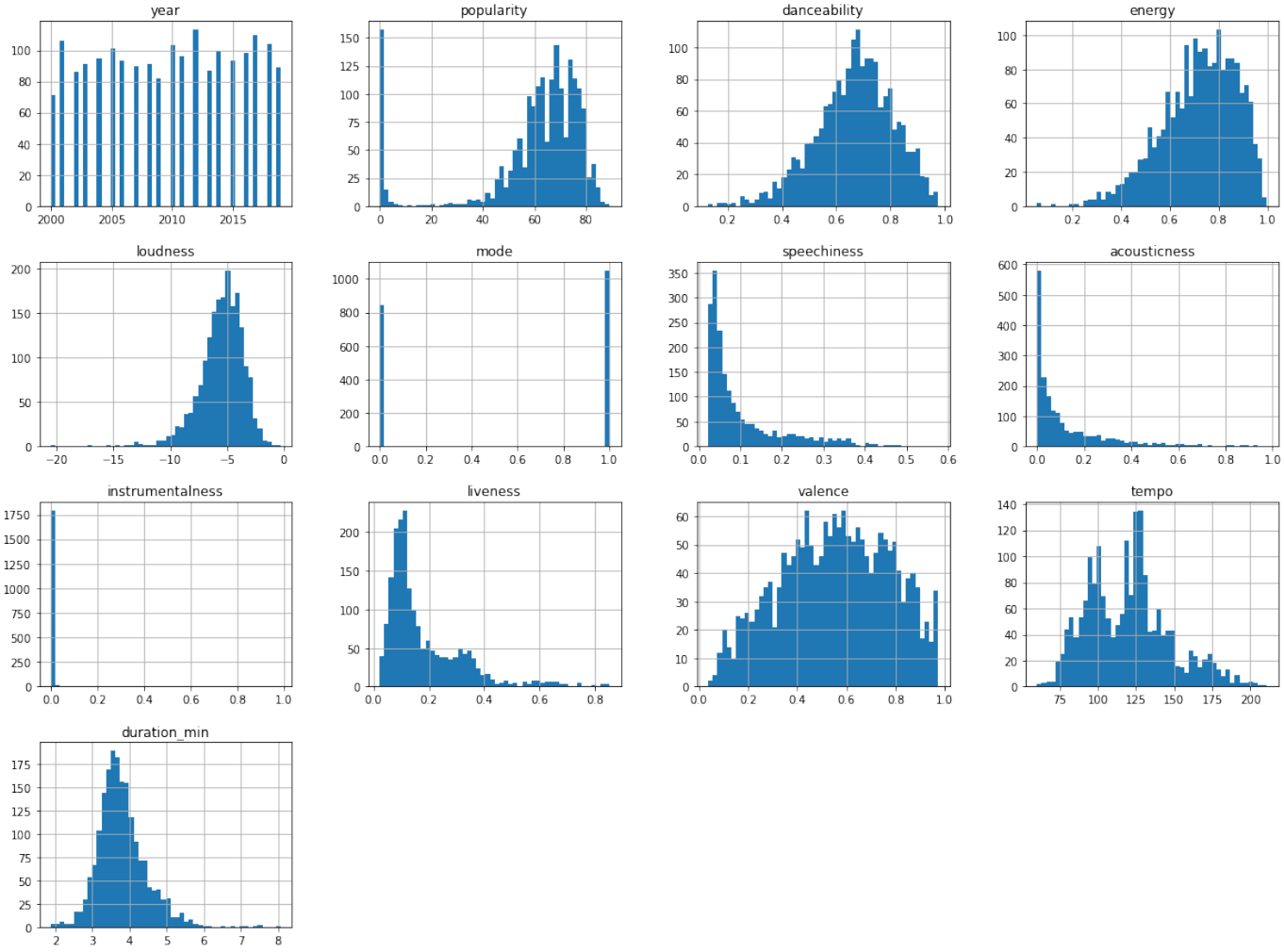

Fig. 31. Features' Distribution.

B. Energy Vs. Popularity

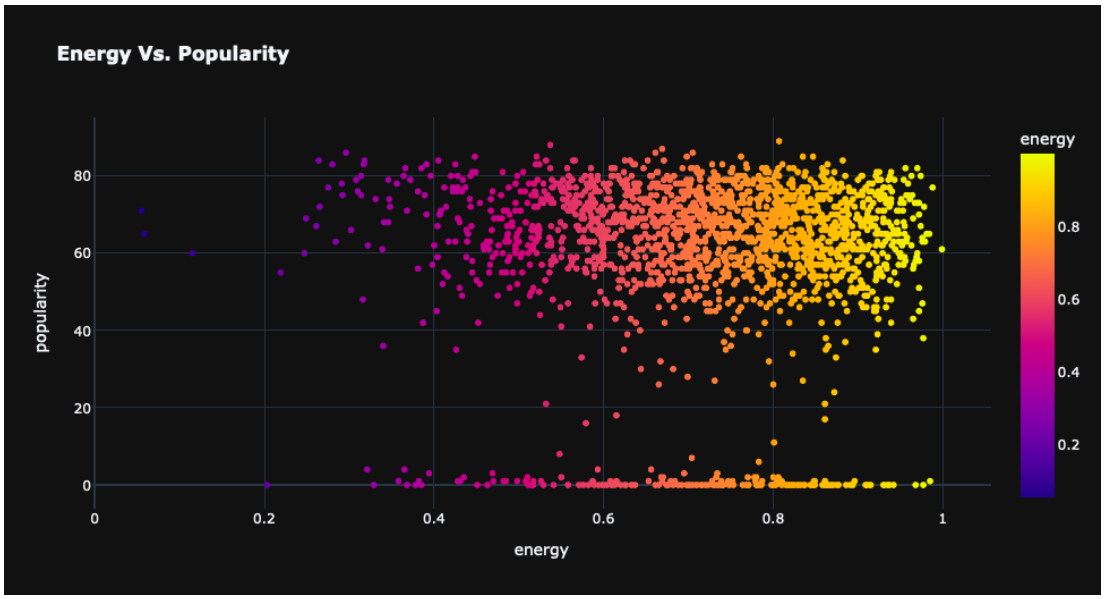

Fig. 32. Energy VS Popularity.

C. Loudness Vs. Popularity

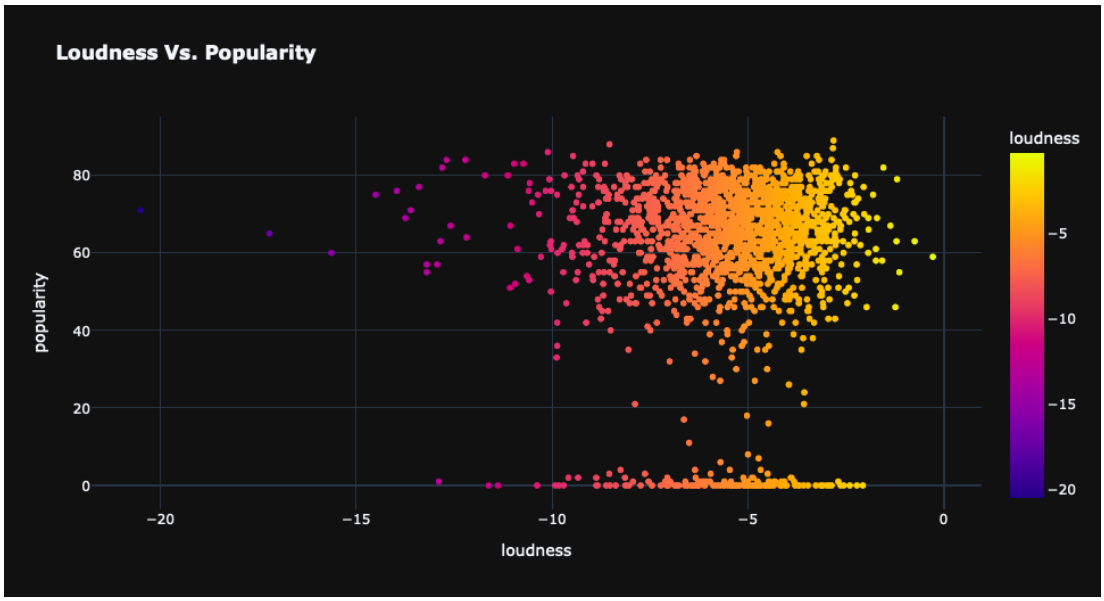

Fig. 33. Loudness VS Popularity.

D. Speechiness Vs. Popularity

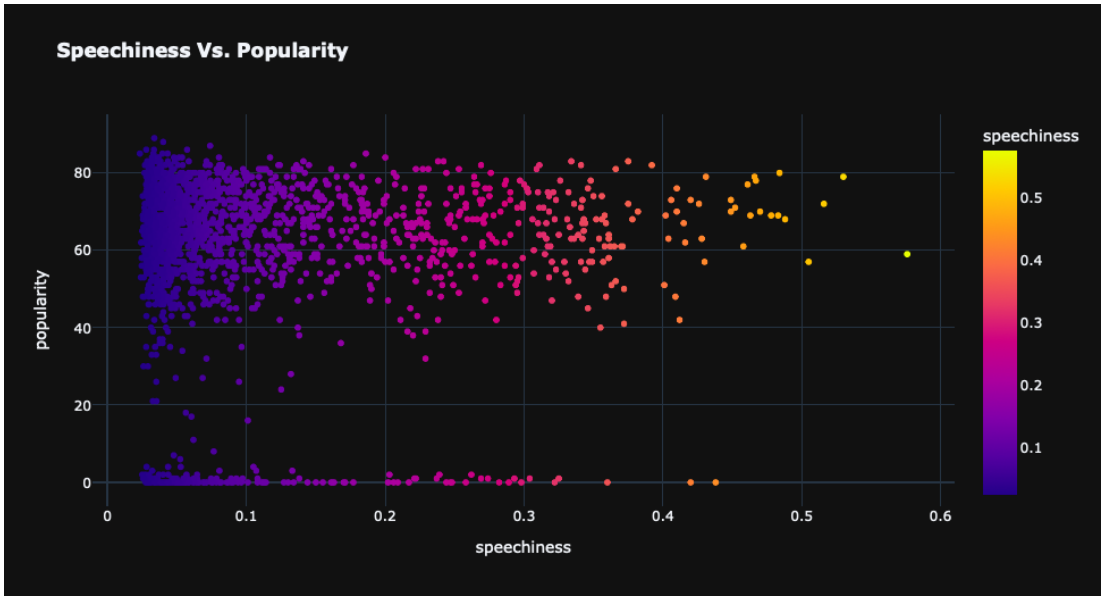

Fig. 34. Speechiness VS Popularity.

E. Explicit Vs. Popularity

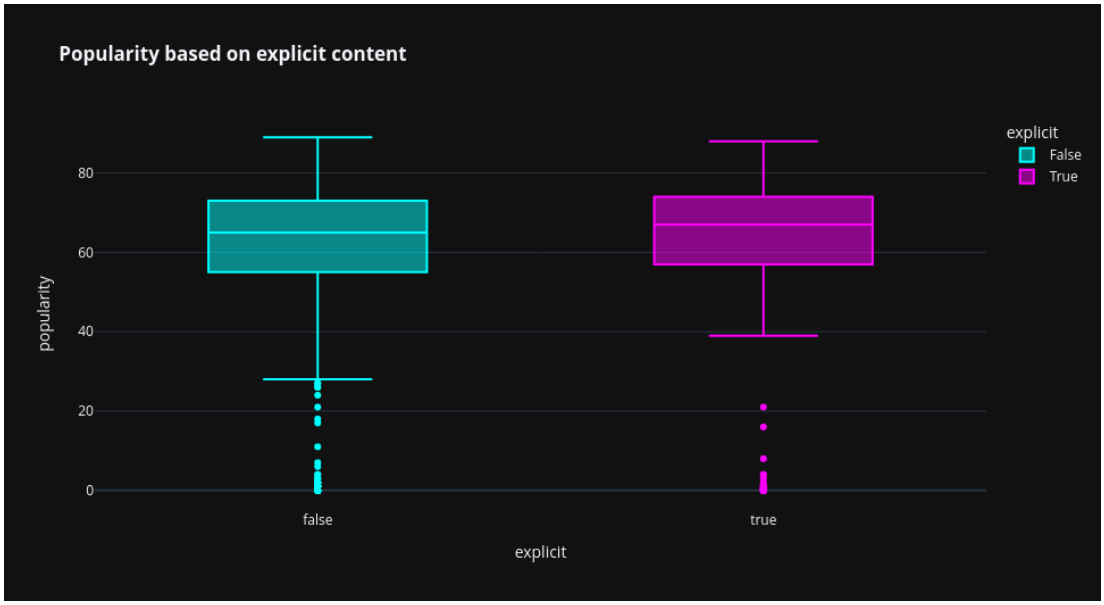

Fig. 35. Explicit VS Popularity.

F. Acousticness Vs. Popularity

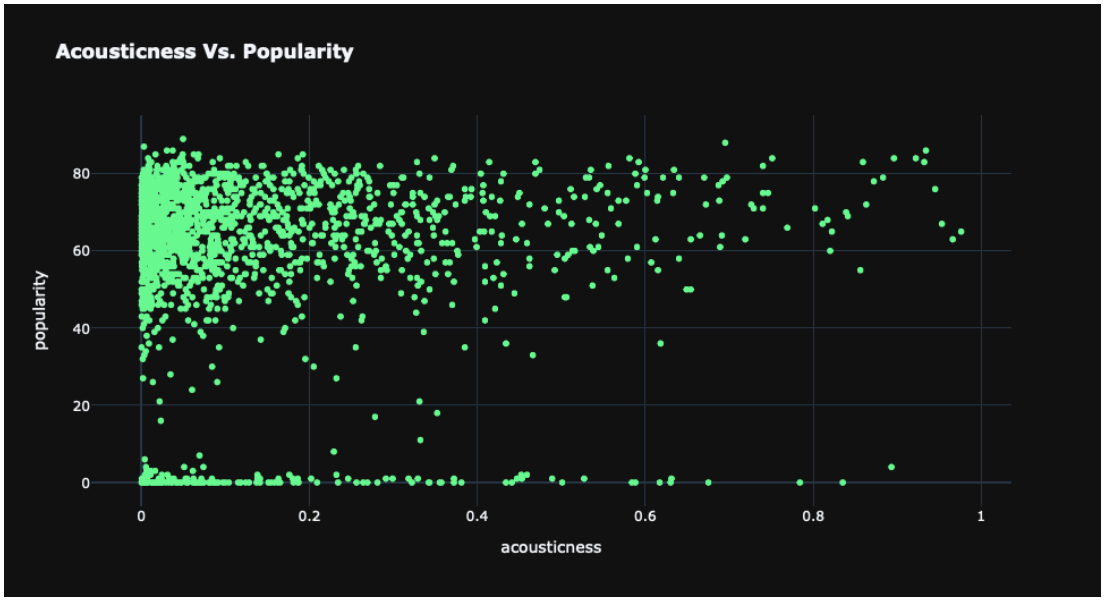

Fig. 36. Acousticness VS Popularity.

G. Year Vs. Popularity

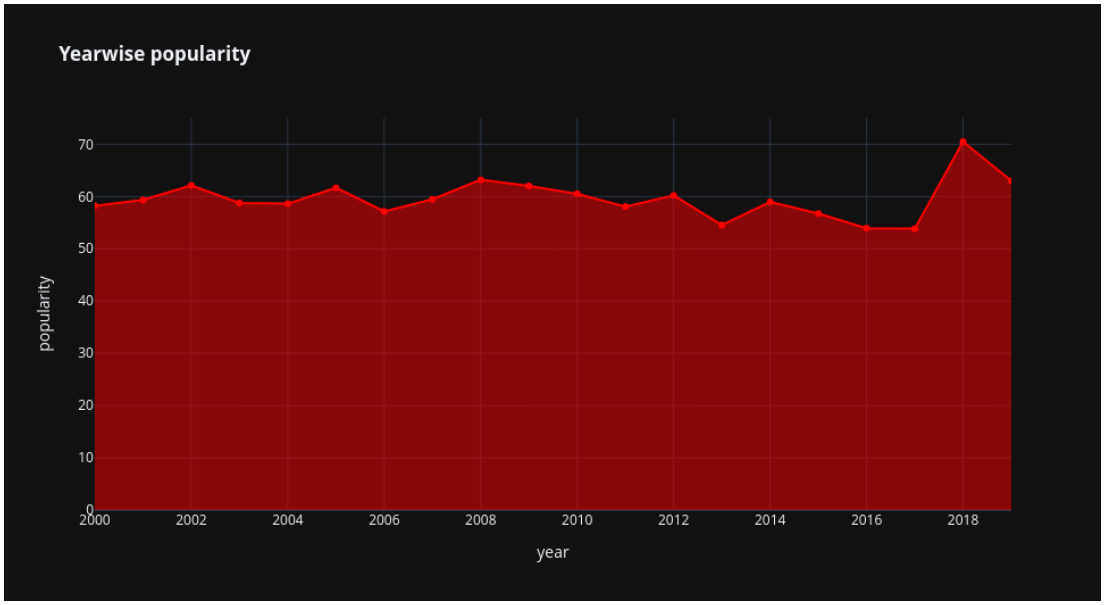

Fig. 37. Yearwise Popularity.

H. Duration Vs. Popularity

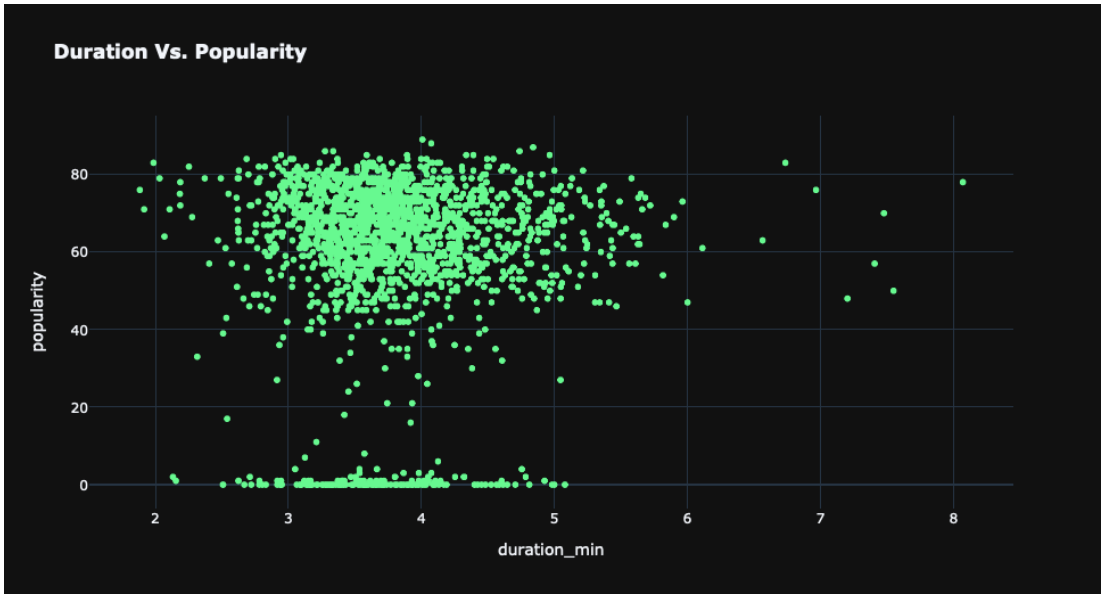

Fig. 38. Duration VS Popularity

I. Accuracy, best parameters and best estimator

```
Best score: 0.514813523875915
Best parameters: {'criterion': 'entropy', 'max_depth': 6, 'max_features': 13, 'splitter': 'best'}
Best estimator: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=6, max_features=13, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Fig. 39. Accuracy, best parameters e the best estimator.

J. Performance vs Hyper-parameters for Decision Tree Classifier.

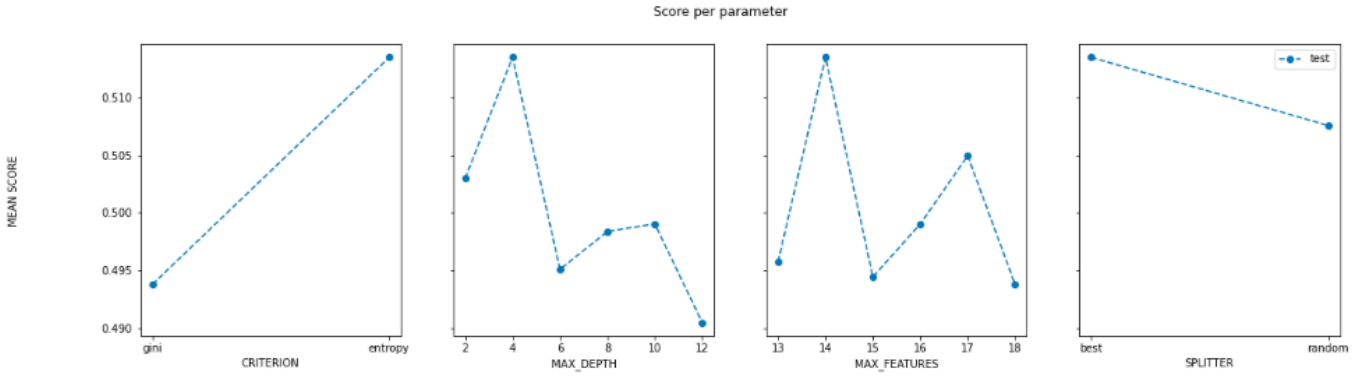

Fig. 40. Performance vs Hyper-parameters for Decision Tree Classifier.

K. Accuracy, best parameters and best estimator for KNN

```
Fitting 10 folds for each of 24 candidates, totalling 240 fits
Best score: 0.47732223771348903
Best parameters: {'algorithm': 'kd_tree', 'n_neighbors': 13, 'weights': 'uniform'}
Best estimator: KNeighborsClassifier(algorithm='kd_tree', n_neighbors=13)
---  ---


Classification report:
              precision    recall  f1-score   support

           0       0.49      0.54      0.51       183
           1       0.53      0.48      0.50       197

    accuracy                           0.51       380
   macro avg       0.51      0.51      0.51       380
weighted avg       0.51      0.51      0.51       380
```

Fig. 41. Classification Results K-NN, best parameters and best estimator.

L. Performance vs Hyper-parameters for Knn.

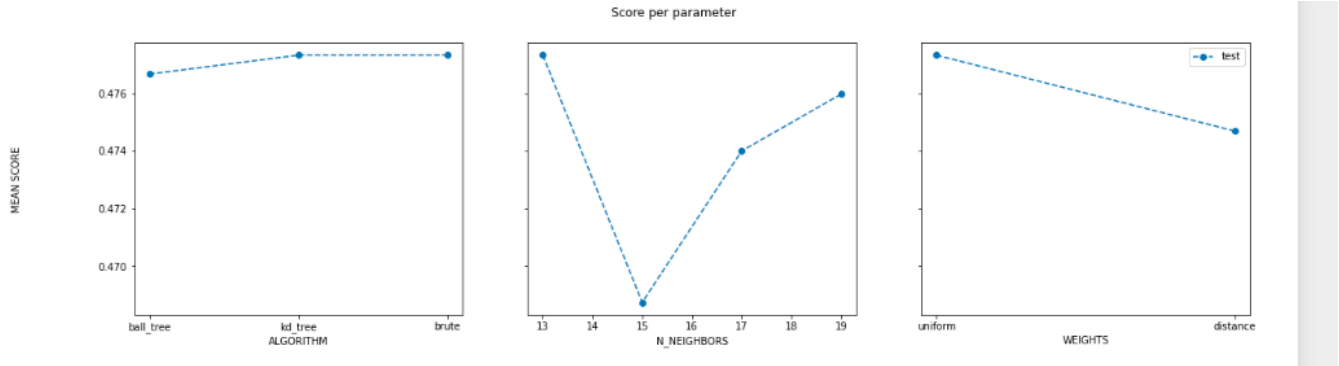

Fig. 42. Score per Parameter K-NN.

M. Accuracy, best parameters and best estimator for Lr

```
Best score: 0.497041652143604
Best parameters: {'C': 10, 'penalty': 'none', 'solver': 'saga'}
Best estimator: LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='saga', tol=0.0001, verbose=0,
                   warm_start=False)
---  ---


Classification report:
              precision    recall  f1-score   support

           0       0.49      0.54      0.51       183
           1       0.53      0.48      0.50       197

    accuracy                           0.51       380
   macro avg       0.51      0.51      0.51       380
weighted avg       0.51      0.51      0.51       380
```

Fig. 43. Classification Results LR, best parameters and best estimator.

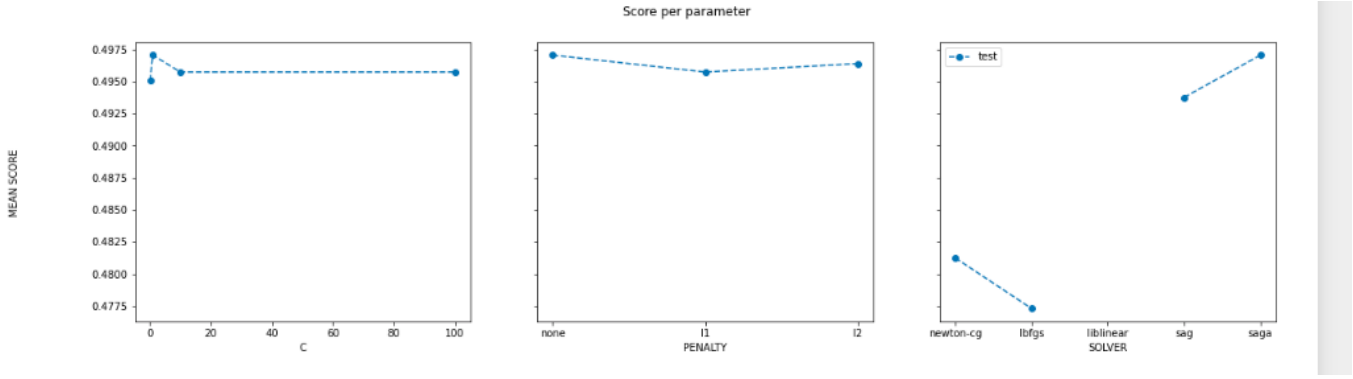

Fig. 44. Score per Parameter Logistic Regression