SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning Project: Spotify Ranking

L. Ferraro, F. Maggioli, A. Valente

A.A. 2017/18

Contents

Chapter 1

Introduction

In this chapter we are going to introduce what is Spotify and which problem we have addressed. During the problem's introduction we are going to give a high level description of the used dataset that we are going to explain deeply in chapter 2.

1.1 What is Spotify

Spotify is an international music streaming service that provides DRM-protected content. It is an online application, but it is also provided as a multi-platform client application, available for the most part of desktop and mobile devices, including consoles.
Spotify provides access to a lot of contents and it manages several millions of users, tracking all their activities, in order to provide them a customized service.
The artists that contribute to the music database with their works are payed with royalties according to the number of streams that these tracks achieve.

Figure 1.1: Spotify's logo.

1.2 Our Problem

In our project we decided to approach two kinds of problems:

- Finding if a track will be in the list of the daily most streamed tracks in a certain country;
- Predict the number of streams that a track will achieve in a certain day.

The two problems have been faced using a machine learning approach. In particular we decided to use a set of supervised learning algorithms.
In order to train them, we have searched for a dataset (even partial) that contains a sufficient number of informations about the content of the Spotify's databases. In particular, we needed a dataset that gave us informations about tracks and number of streams for every day.
We found a dataset on Kaggle that provides many informations about the 200 most streamed songs of the day for each country for an entire year. As already said, we will talk about the dataset deeply in chapter 2.

Chapter 2

Dataset

In this chapter, we are going to introduce the dataset that we have used in our project.
First, we will give a deep description of the attributes contained in the dataset. Then, we will analyze the content, explicitly giving summary informations about it.

2.1 Features Description

Here, we give a table that describes the features contained in the dataset.

Column Name	Data Type	Column Description
Position	Integer	The rank achieved by the track at the end of the day in the country.
Track Name	String	The title of the track.
Artist	String	The name of the musician or the group that composed the track.
Streams	Integer	The number of streams achieved by the track at the end of the day.
URL	String	The URL to the track on the Spotify's website.
Date	Date	The date.
Region	String	The ISO 3166-2 country code.

Table 2.1: Description of the dataset's features.

The dataset is free to download at Spotify's Worldwide Daily Song Ranking.

2.2 Content Analysis

The content of the dataset consists in the daily ranking of the 200 most listened songs in 53 different countries between 2017 and 2018 by Spotify's users.
The dataset contains more than 3 million rows and it has been collected by Spotify's regional chart data.
Here, we present a summary of the content of it:

- *Position* spans from 1 to 200 and represents the position occupied by the track in that day and that region.
- *Track Name* has more than 18000 possible values. The dataset contains 657 rows where the track name is unknown. The rows with unknown track name are equally distributed among regions and positions. However, there are only 20 dates that contains unknown track names.
- *Artist* has more than 6500 possible values. The dataset contains 657 rows where the artist is unknown. These rows coincide with the rows where the track name is undefined.
- *Streams* contains the number of streams achieved by the track in that day and in that region. The number of streams spans from 1001 to more than 11 millions, with a mean value of about 50000.
- *URL* contains the URL to the Spotify's website page for the song. Notice that the number of URLs is more than 21000, that exceeds the number of track names. This is because of the presence of different tracks with the same title.
- *Date* spans from January 1st 2017 to January 9th 2018. The missing dates are 2017-05-30, 2017-05-31 and 2017-06-02.
- *Region* has 54 possible values that cover all the countries in which Spotify is present. Actually, the region codes are 53, because there is a special region code, namely "`global`", that represents the worldwide ranking.

Chapter 3

Project

Before we present the solution to the problems described in section 1.2, we are going to briefly introduce the tools we used and the preprocessing procedure that we applied to the data.
Finally, we will talk about the algorithms we used and which data have been used to train them.

3.1 Tools Introduction

The project has been realized in *Python* language using well known machine learning and data science packages such as *SciKit Learn*, *Pandas* and *NumPy*.
Before continuing with the next section, we will spend a few words about those tools.

Figure 3.1: Python's logo.

Python is an object-oriented scripting language. It is widely used for small projects because of its easy syntax and learning curve, but it is not very widespread for large project because of its very poor efficiency. Anyway, after the introduction of packages such as *NumPy*, *SciKit Learn* and *TensorFlow*, that are very efficient for mathematical and machine learning tasks, *Python* has gained a large segment of users in those fields.

NumPy is a widely used library for math in *Python*. It is implemented in *C* language, and because of this it is very fast and efficient. Furthermore, the *Python* interface makes it very easy to use, even to beginner programmers.

Pandas is a *Python* library for data science. It provides a very useful interface to complex data structures such as series and dataframes. Also *Pandas* is implemented in *C* language and exposes efficient operations to filter and analyze data.

SciKit Learn is a *Python* library for machine learning. It is written in *Python* and *Cython* and provides a large set of machine learning algorithms. Between those algorithms, we can find tools for classification, regression and clustering, including the best known ones.

Figure 3.2: Logos of a wide range of Python's packages

3.2 Preprocessing Data

Even the best machine learning algorithms cannot be ever fitted with raw data, but there could be needing of some preprocessing.
In particular, we could have useless or missing informations, or data that can be incomprehensible for the algorithms.

In the preprocessing procedure we faced all of those three cases and, in this section, we will expose where the issues have been found and how we have handled them.

Furthermore, we analyze two other kinds of preprocessing that we have applied to the data. These methods are called *data transformation* and *data augmentation*. We have used it on the number of streams.

3.2.1 Useless and Missing Data

In chapter 2 we showed the content of the dataset. There we have seen that data contains the *URL* column. In analyzing the data, it was immediately clear that this column was useless for our purpose. So, we decided to eliminate it from the dataset.
All the other columns are used as feature, target or to filter the dataset to get only a subset of the data which can be intersting to verify some properties.

Always in chapter 2, we have talked about rows with missing track names and artists. Because of the presence of those rows in every country and position and in a set of sparse dates, we had no way nor to infer their value, nor to infer if the missing data are related in some way. So, we decided to drop the rows containing the missing informations.
Except for track names and artists, there's no other column containing missing informations.

3.2.2 Encoding Categorical Features

In the dataset we have three categorical features, without considering the dropped ones. In particular, the categorical features are:

- The date;
- The artist;
- The track name.

Because the algorithms provided by the *SciKit Learn* package cannot work with this kind of features, we need to encode them into numerical values.
We start from the simplest encoding, that is the one used for the dates. Because dates are naturally ordered by time, we decided to encode them into an ordered integer sequence. So, we have applied a label encoding that follows the natural ordering of dates.

Now, we have to deal with artists and track names. Because there's not an implicit ordering between artists and track names, we cannot apply label encoding. The first alternative we thought of was one-hot encoding, but we immediately faced a very difficult problem. In fact, as previously introduced in section 2.2, the columns *Artist* and *Track Name* have, respectively, about 6000 and 18000 possible values. This means that the resulting encoded dataset will have more than 24000 features, which is obviously an unfeasible machine learning problem.
So we were forced to find an alternative that allows us to reduce the number of features. The alternative we used is called *label binaryzation* and now we will explain how it works.
Let f be a feature such that $|\mathrm{Dom}(f)| = n$, i.e. there are n possible values for the feature f. Let's define an arbitrary ordering $v_1, \dots, v_n$ for the values of feature f.

We create m new binary features $f_1, \dots, f_m$, where $m = \lceil \log_2(n) \rceil$ and we encode each value v_i as follows:

Algorithm 1: The binary encoding algorithm.

```
Data: $\mathcal{D}$ dataset, $f$ feature of $\mathcal{D}$, $n = |\mathrm{Dom}(f)|$
Result: $\mathcal{D}$ with the feature $f$ binary encoded
begin
    Add to $\mathcal{D}$ the features $f_1, \dots, f_m$ and initialize them to null
    Give an ordering to $\mathrm{Dom}(f) = \{v_1, \dots, v_n\}$
    /* For each row in the dataset, encode the value it has in feature $f$ */
    for $r \in \mathcal{D}$ do
        $v_i \leftarrow r[f]$
        Let $b_{i,1} \dots b_{i,m}$ the binary encoding of $i$
        for $j \leftarrow 1$ to $m$ do
            $r[f_j] = b_{i,j}$
    Drop feature $f$ from $\mathcal{D}$
```

In this way, we successfully encoded all the values of feature f, but instead of introducing n new features, as in one-hot encoding, we only introduced $\log_2(n)$ new features.

What we need to prove is that this encoding is correct and it has the same desirable properties of one-hot encoding, meaning that no encoded value is "more valuable" than others.

The first claim is easy to prove, because the values v_i are ordered and for each $i, j \in [1, n]$ we have $i \neq j \iff v_i \neq v_j$. Moreover, the binary encoding is unique for each integer value, so for each v_i, v_j we have that they are different if and only if their binary encodings are different.

The second claim is harder to prove. First, we need to clearly define what it means to be "more valuable" for binary vectors. Here, with this definition, we intend to say that an algorithm could act differently if we apply the same encoding for a different ordering of the same feature's values.
The first weakness of binary encoding is that it is possible to define a partial ordering between vectors. In fact, if we have two vectors equals in each component, except for the i^{th} one, then we could consider the vector where the i^{th} component is 1 greater than the vector where the same component is 0. However, because of the binary domain of each component, this ordering defines a lattice where the longest chain is limited to $\log_2(n)$ elements, rather than n elements, as it happens in label encoding.

$$\begin{tikzcd}
 & & & & & \langle 1,1,1,1 \rangle & & & & & \\
 & & \langle 0,1,1,1 \rangle \arrow[urrr] & & \langle 1,0,1,1 \rangle \arrow[ur] & & \langle 1,1,0,1 \rangle \arrow[ul] & & \langle 1,1,1,0 \rangle \arrow[ulll] & & \\
\langle 0,0,1,1 \rangle \arrow[urr] \arrow[urrrr] & & \langle 0,1,0,1 \rangle \arrow[u] \arrow[urrrr] & & \langle 0,1,1,0 \rangle \arrow[ull] \arrow[urrrr] & & \langle 1,0,0,1 \rangle \arrow[ull] \arrow[u] & & \langle 1,0,1,0 \rangle \arrow[ullll] \arrow[u] & & \langle 1,1,0,0 \rangle \arrow[ullll] \arrow[ull] \\
 & & \langle 0,0,0,1 \rangle \arrow[ull] \arrow[u] \arrow[urrrr] & & \langle 0,0,1,0 \rangle \arrow[ullll] \arrow[u] \arrow[urrrr] & & \langle 0,1,0,0 \rangle \arrow[ullll] \arrow[ull] \arrow[urrrr] & & \langle 1,0,0,0 \rangle \arrow[ull] \arrow[u] \arrow[urr] & & \\
 & & & & & \langle 0,0,0,0 \rangle \arrow[ulll] \arrow[ul] \arrow[ur] \arrow[urrr] & & & & &
\end{tikzcd}$$

Figure 3.3: A lattice of binary vectors of length four.

The second weakness of binary encoding is that the distance between the couples of vectors is not constant. In this way, we could associate nearest vectors and preferring to associate far vectors to very different values. Anyway, if we analyze the distance distribution, we can see that not only the maximum distance is very small, but also that there is a very large amount of couples of vectors with the same distance.
Think about the vector with all zeros and the vector with all ones. They are clearly the most distant ones. If we measure their distance using the euclidean metrics, we obtain that $\text{Dist}(\mathbf{0},\mathbf{1}) = \sqrt{m} = \sqrt{\log_2(n)}$, which is, as we previously introduced, a very small value.
Now, we would like to know which is the largest distance less than the maximum one. Because the vectors are binary, we can easily see that the couples of vectors for which we obtain such a distance are the couples $(\mathbf{0},\mathbf{v})$ where $\mathbf{v}$ has all ones except for a single component and the symmetric case $(\mathbf{1},\mathbf{v})$ where $\mathbf{v}$ has all zeros except for a single component. Those couples of vectors have distance equals to $\sqrt{m-1}$ and there are $2m$ couples of this kind. If we continue the reasoning this way and we count the number of couples of vectors at distance k from the endpoints, we can obtain a closed form for the formula that gives us the number of couples of vectors at distance $\sqrt{m-i}$, that is

$$c_i = \sum_{j=0}^{i} \binom{m}{j}\binom{m}{i-j} = \sum_{j=0}^{i} \binom{\log_2(n)}{j}\binom{\log_2(n)}{i-j} \tag{3.1}$$

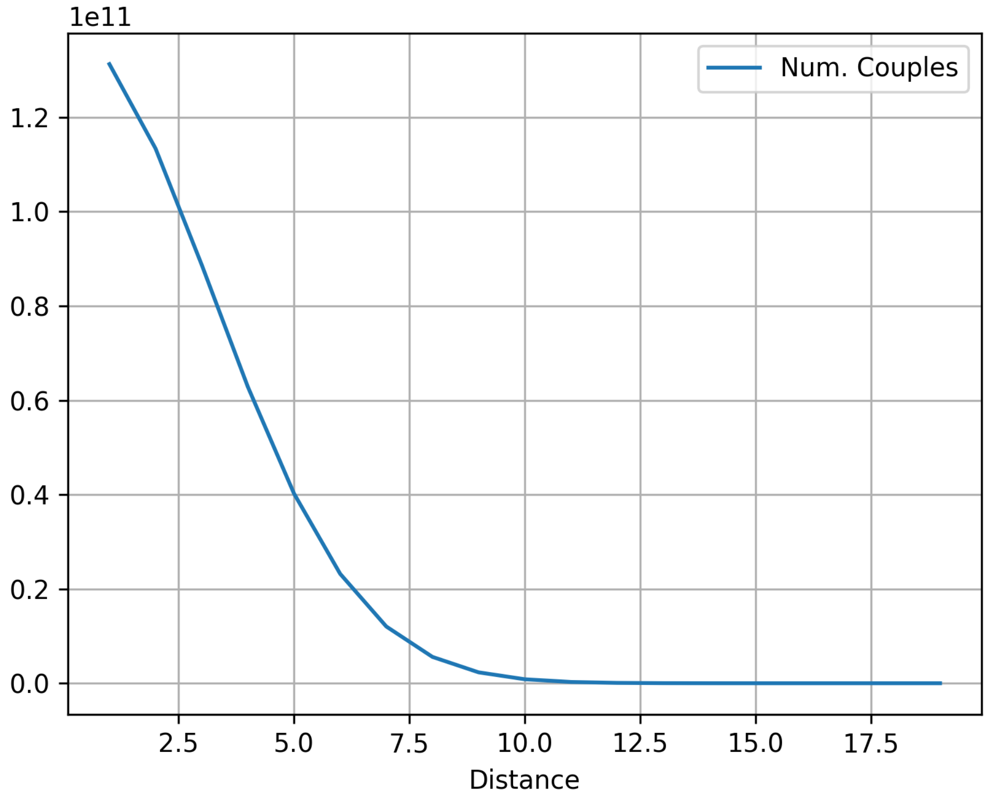


Figure 3.4: The distribution of the number of couples of vector with a certain distance. In this figure the number of binary features is $m = 20$.

Equation 3.1 defines the distribution of the distances between the couples of vectors as a function of the distance and the number of components of the vector.
As we can see in Figure 3.4, as the distance decrease, the number of couples with at that distance dramatically increases. In fact, if we compute the difference between c_i, representing the couples of vectors at distance $\sqrt{m-i}$, and c_{i+1}, representing nearest vectors at distance $\sqrt{m-i-1}$, we see that this difference is $\binom{m}{i+1}$, which can be approximated to m^{i+1}.
This leads to the conclusion that, in binary encoding, we could effectively have values represented by vectors that can be grouped by their distance, inferring a grouping of the values that does not exist, but also that this grouping is limited by two main factors:

- The maximum distance between two vectors is bounded by $\log_2(n)$;
- There is a low probability that, choosing three random vectors $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, the distances $\text{Dist}(\mathbf{v}_1, \mathbf{v}_2)$ and $\text{Dist}(\mathbf{v}_1, \mathbf{v}_3)$ are very different.

3.2.3 Data Transformation and Augmentation

The number of streams in the dataset was not very well distributed and this has led to a very imprecise prediction, when we tried to guess the number of streams.
The original distribution of the number of streams in the dataset is given by the left side of Figure 3.5. We can see that the distribution of the number of streams is concentrated on small values and that higher values can be found in very few rows.

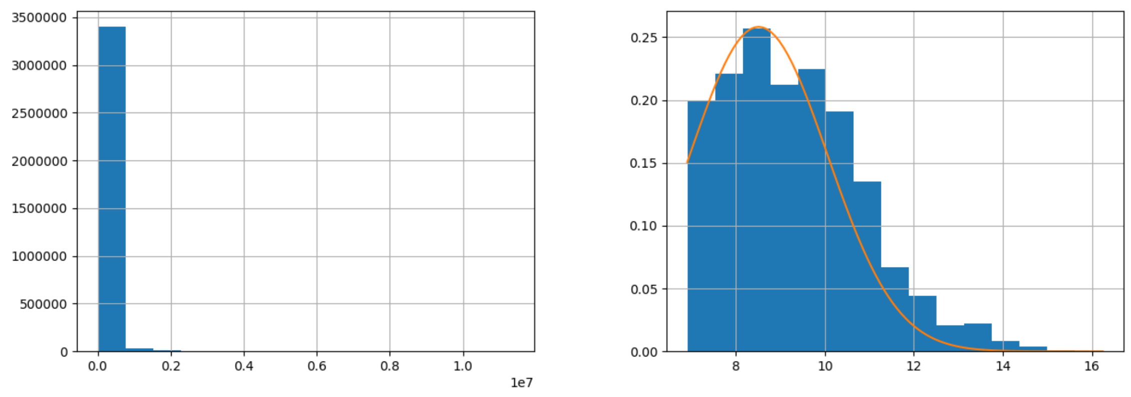


Figure 3.5: Difference between the distribution of the number of streams in the dataset and the distribution of the same values after the application of the function $\ln(x+1)$.

To solve this problem we decided to apply a data transformation method. In particular, we tried to make the distribution more Gaussian-like applying the function $\ln(x+1)$.
The application of the function to the *Streams* column of the dataset resulted in a distribution that fits better a Gaussian one. This can be seen in the right side of Figure 3.5.

Another data manipulation we took advantage of is the *data augmentation*. We used this manipulation to add the column *Previous Streams* to the dataset. In each row, containing a track t, a date d and a region r, the column *Previous Streams* contains the number of streams registered by the track t in the greatest date $d' < d$ when it reached the top 200 in region r.

3.3 Algorithms

We informally introduced our problems in section 1.2. Now, we are going to define them more precisely and in terms of the dataset content. Furthermore, in this section we are going to describe the kind of approach and the algorithms that we have used for the predictions.

Let's start defining the first problem. We would like to know if a track is likely to be in the daily top listened tracks for a certain country.
To do so, we decided to filter the dataset in function of the *Region* column, concentrating us only on the country we are interested in.
For this first problem we decided to ignore the *Streams* column, because knowing the number of streams in a given day makes deterministic (and very easy) the problem of computing the list of the top songs for that day. So, our problem can be defined as follows:

Definition 3.1 ("Top Songs" classification problem). Let k be a positive integer. Using as features the columns *Date*, *Track Name* and *Artist*, we would like to predict if the value in the column *Position* is less than or equals to k.

Notice that the features we are using are all categorical and, except for the column *Date*, which is encoded into a numerical feature, the columns *Artist* and *Track Name* are encoded with the binary encoding method described in subsection 3.2.2, that sufficiently maintains the categorical nature of the features.

Due to this strong presence of categorical features, we decided to use a class of classification algorithms that works well with this kind of values: the decision trees. In particular, we used both the *Decision Tree Classifier* and the *Random Forest Classifier* implemented in the *SciKit Learn* library.
In the first phases we also used other kinds of algorithms in order to be sure that our thoughtful idea was better than a random choice. We tested the trees against *Ada Boost Classifier*, *Support Vector Classifier*, *Logistic Regressor* and the *Multi-Layer Perceptron Classifier*. In all those cases, the best achieved performances were worse than those achieved by the trees.

Let's now define the second problem. We would like to predict the number of streams that a track will achieve in a certain day. Once again, we decided to filter the dataset in function of the column *Region*. Furthermore, we decided to drop the column *Position*, because, likely, if we want to predict the number of streams of a track we cannot know the position that it will obtain. So, our problem can be defined as follows:

Definition 3.2 ("Today's Streams" regression problem). Using as features the columns *Date*, *Track Name*, *Artist* and *Previous Streams* (we defined this latter column in subsection 3.2.3), we would like to predict the value of the column *Streams*.

As said before for the "Top Songs" problem, most features are categorical. So, once again, we decided to use trees. Nevertheless, the presence of a numerical feature (that is *Previous Streams*) led us to use also other regressors that works well in presence of some categorical features: the *Linear Regressor* and the *Multi-Layer Perceptron Regressor*.
Also in this second problem, we decided to use other random algorithms, in particular we used again *Ada Boost Regressor* and *Support Vector Regressor*.
Even if our choice resulted to be better than the random ones, we found that also the *Linear Regressor* did not work well with this problem.

Chapter 4

Testing

In this chapter we are going to present the experimental results obtained using the algorithms presented in section 3.3.
In each of the two sections we will first present the experiment's settings and then the results that turned out by the execution of the algorithms on those settings.

4.1 "Top Songs" Classification Problem

To test the "Top Songs" feature we settled up three experiments of different nature:

- Trying to compute if a track will be one of the top 10 tracks of the day in Italy, varying the fraction of the dataset to use as training set;
- Trying to compute if a track will be one of the top tracks of the day in U.S.A. and Canada, varying the length of the top songs' list and using as training set the 75% of the dataset;
- Trying to compute if a track will be one of the top 5 tracks of the day in a country, using as training set the data about a month and as test set the first week of the following month.

We are going to examine the settings and the results of these three experiments separately and more in depth.

Let's start with the first experiment, about the Italian top 10 on the whole database. We will examine the results obtained by the *Decision Tree Classifier* and those obtained with the *Random Forest Classifier*. For each of those classifiers, we will see the different performances for accuracy, precision and recall.
On Figure 4.1 we can see that increasing the training set dimension does not affect heavily the accuracy score of the prediction. Anyway, we can notice that the *Random Forest Classifier* obtains a better results than the *Decision Tree Classifier*.
Nevertheless, if we examine the question we posed to the algorithm, it turns out that the accuracy is not a very good metric. In fact, for each day we have the Italian top 200 and we are asking if a song is in the Italian top 10. The songs in the top 10 are the 5% of the total, so a classifier that predict all negatives will achieve a 95% accuracy score. This means that, even if we achieve 99% accuracy, the classifier could fail in predicting many positives. In particular, it could fail on the 20% of the positives' classification. For this reason, we will examine also the precision and the recall score.
In Figure 4.2 we can see that our classifiers achieve a very good results even when considering the precision and the recall scores. In particular, it is interesting to notice how better are the performances of the *Random Forest Classifier* on the precision score and how the training set dimension affects the recall score.

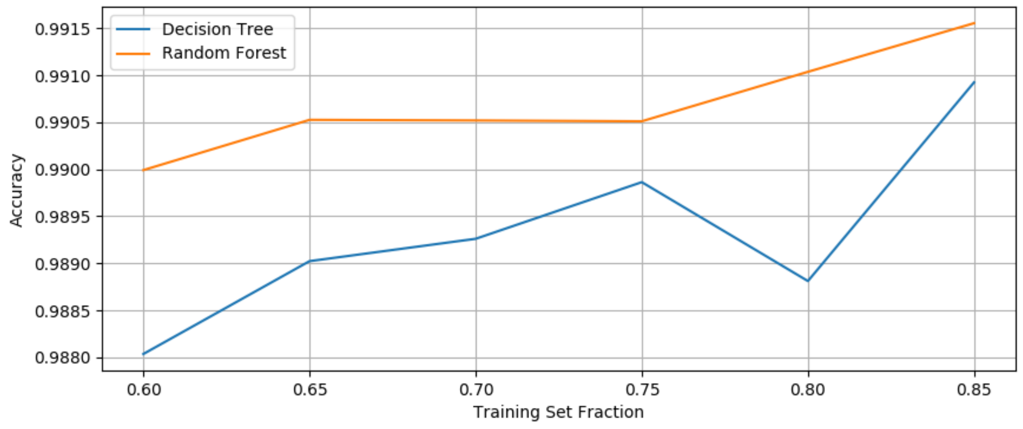


Figure 4.1: The accuracy score for the "Top Songs" prediction on the Italian top 10 at varying of the training set dimension.

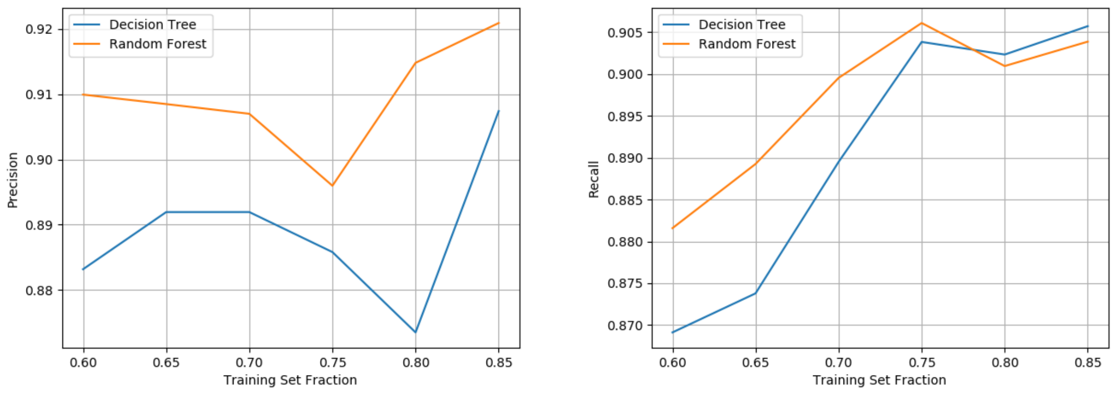


Figure 4.2: The precision and the recall scores of the "Top Songs" prediction on the Italian top 10 at varying of the training set dimension.

The analysis of the other two experiments will show the limitations of such a prediction approach.

The second experiment involves a variable top list for U.S.A. and Canada. So, we are trying to predict a top for two countries, instead of one. The training set dimension is fixed to the 75% of the original dataset and the length of the top list varies from 3 to 10.

It is interesting to notice that as the length of the top list increases (and so, intuitively, the prediction becomes easier), the accuracy of the prediction decreases.

This non-intuitive variation in the accuracy can be explained looking at the precision and recall scores. In fact, they are more or less constants in function of the length of the top list and they move around the 65% and the 68%. This means that, when our algorithms predicts three positive instances, one of them is a false positive and, for each three positive instances, our algorithm classifies one of them as negative.
This constant percentage says that the fraction of positive instances in the dataset increases as the length of the top list increases. So, having a fixed percentage of failure on the positive instances in-

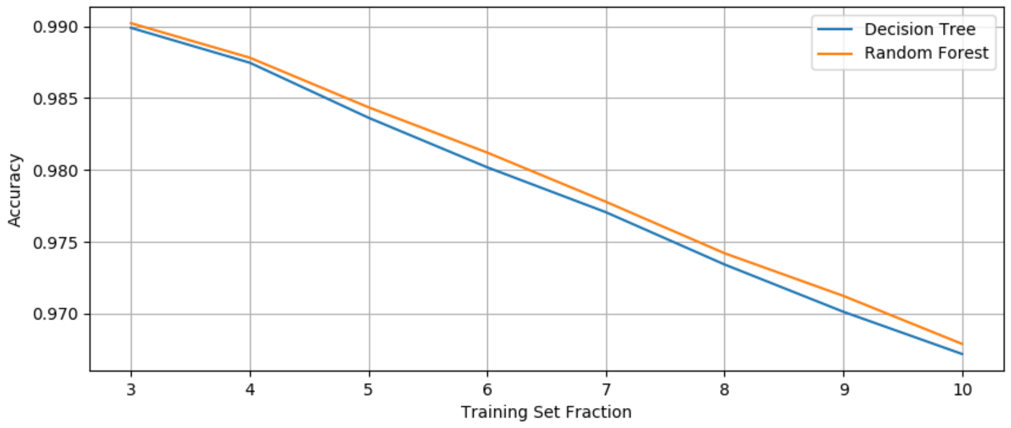
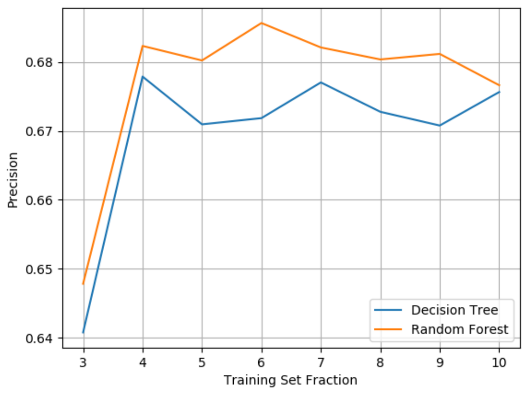


Figure 4.3: The accuracy score for the "Top Songs" prediction on U.S.A. and Canadian top at varying of the length of the top list.



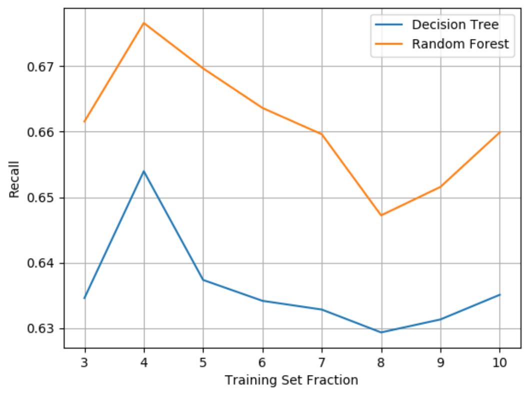


Figure 4.4: The precision and the recall scores of the "Top Songs" prediction on U.S.A. and Canadian top at varying of the length of the top list.

creases the total number of failures in the dataset, as the number of positive instances increases. This low scores on precision and recall let us suppose that a multi-country prediction is more difficult than a single-country one.

Finally, let's talk about the third experiment. We tried to predict if a song was in the top 5 tracks of the day in a country. The prediction tries to guess the top 5 in the first week of a month, learning from the previous month.

On Figure 4.5 we can see how the accuracy, the precision and the recall scores vary in function of the examined month. The figure shows the average value of metrics, between all the 14 countries where it has measured, in relation to the examined month.

It is very interesting to notice how the precision score and the recall score are very similar. Using their definitions, respectively $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$, since the number of true positives is the same it must be that the number of false positives is very close to the number of false negatives.

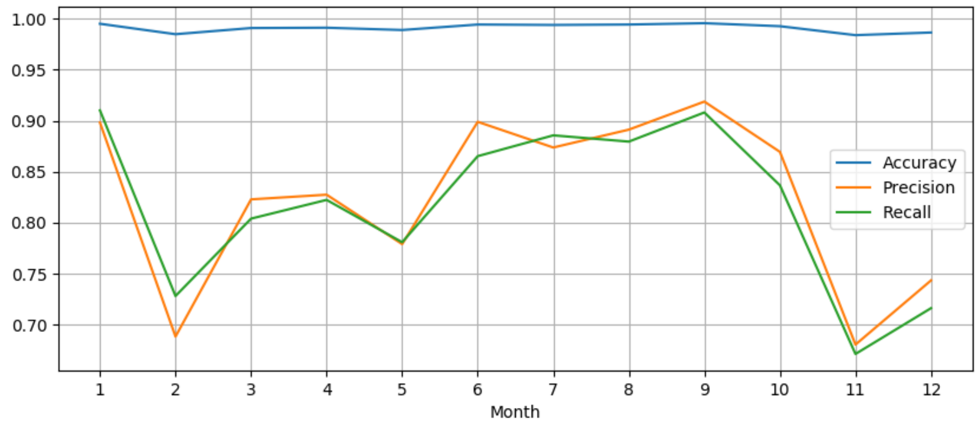


Figure 4.5: The average of the three metrics (accuracy, precision and recall) used to test the "Top Songs" algorithm. On the x-axis, the ordinal of the month used as training set.

After this test, we were interested in examining how long the training set has to be in order to maximize performances. This test has been considered because we thought that classifications should depend more from instances that are near in time, rather than instances from a remote past. To perform this test, we selected the two instances whose performed, respectively, better and worse in predicting the first week of June, and we executed the same algorithm extending the training set by two weeks in the past at each iteration.
Contrary to our thinking, there was no increasing or decreasing in the performances. We suppose that the label encoding of the dates somehow forces the algorithm to "cluster" an instance with others that are near in time, ignoring the others.

In conclusion, we can state that our algorithm's performances are low when we try to predict a random fraction of the dataset that involves two or more countries.
On the other hand, if we consider a single country, performances increases dramatically. Furthermore, if we still consider a single country, but rather than randomly sampling the training set and the test set from the dataset, we define them in order to classify future instances training the algorithm on past ones, we also obtain very good performances. Those performances seems to not depend from how long in the past we look.

4.2 "Today's Streams" Regression Problem

The test of the "Today's Streams" feature involves a single experiment. This experiment consists in trying to predict the number of streams that track will achieve in a certain day in Italy, knowing the number of streams it achieved the last time it obtained a position in the Italian top 200.

Even if we already justified the logarithmic scaling of the number of streams in subsection 3.2.3, we wanted to try the prediction also with the normal scaling, in order to compare the performances.

The metrics we used analyzing the results was the *Root of the Mean Squared Error* (*RMSE*) and the *R Squared* (R^2). Because we don't want that a small relative error on large values affect to much the RMSE, we are going to use a variant that measures the relative error. The RMSE we are using is measured as

$$\text{RMSE}(\mathbf{x}, \mathbf{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - y_i}{x_i}\right)^2}$$

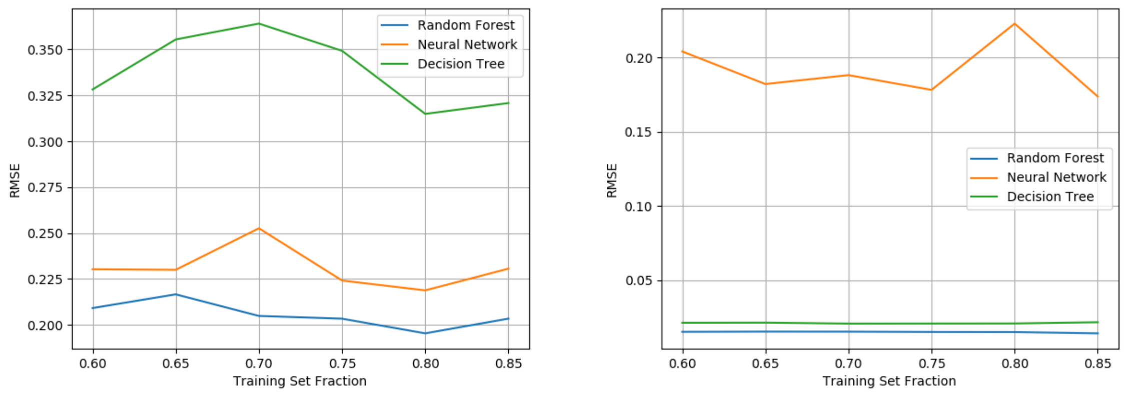


Figure 4.6: The *RMSE* obtained at varying of the training set dimension. To the left, the results on the original dataset, to the right, the results obtained with the logarithmically scaled number of streams.

First, we are going to examine the *RMSE* of the two predictions obtained with the *Decision Tree Classifier*, the *Random Forest Classifier* and the *Multi-Layer Perceptron Classifier*.
In Figure 4.6 we can see the results obtained by the classifiers in the original dataset and in the dataset modified by logarithmically scaling the number of streams.
The first thing to notice is that the RMSE computed by any of the three classifiers is not affected by the dimension of the training set, both in the normal and the logarithmic scaling of the data. In fact, in both the graphs, lines float around the same value for any fraction of the training set.
The second thing is that the scale of the data seems to be not really significant for the *Multi-Layer Perceptron Classifier*, while the performances of the other two classifiers grows incredibly up to a best RMSE score of 0.015 for the *Random Forest Classifier* and 0.020 for the *Decision Tree Classifier*. Notice that the variant of the RMSE we are using computes the relative error, so it does not matter the value of the data and it makes sense to compare the two graphs.

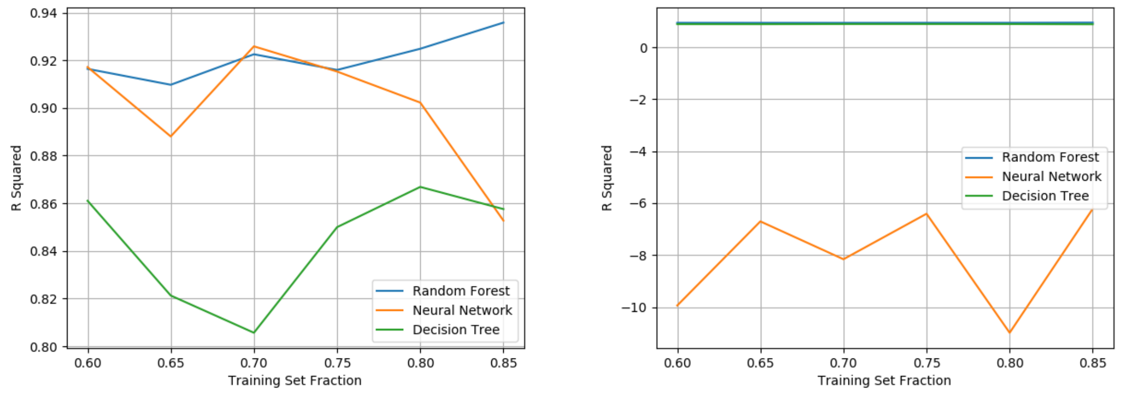


Figure 4.7: The R^2 obtained at varying of the training set dimension. To the left, the results on the original dataset, to the right, the results obtained with the logarithmically scaled number of streams.

At this point, we can proceed analyzing the R^2.
In Figure 4.7 we can see how this value varies when the training set dimension increases. As in the RMSE, the best score is achieved by the *Random Forest Classifier*, followed by the *Multi-Layer*

Perceptron Classifier. The score achieved by the *Decision Tree Classifier* is significantly worse.

Neverthless, when we switch to logarithmic scaling, two considerations come to mind. The first one is that the *Decision Tree Classifier*'s performances goes closer to the performances of the *Random Forest Classifier*. Due to the scale of the image, it is difficult to see, but the first classifier achieves a score of, more or less, 0.89, while the second one achieves a score of 0.94.
The second consideration is that the R^2 obtained with the prediction of the *Multi-Layer Perceptron Classifier* could justify its RMSE score. In fact, such a large negative score means that the model fitted by the classifier completely miss the significance of the real model.

Finally, we are going to examine how the robustness of our precision varies during the year and how the length of the training set affects it.

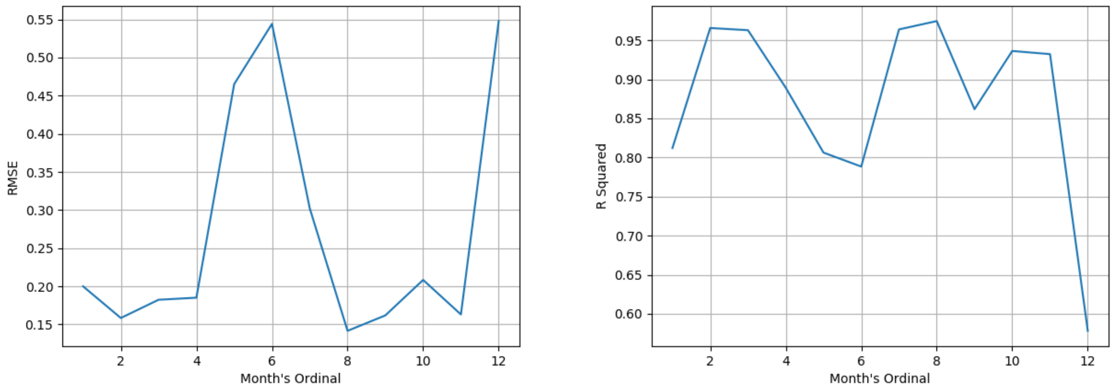


Figure 4.8: The graphs of the RMSE and the R^2 in the various months of the year.

Let's start from the variation of the RMSE and the R^2 during the months of the year. As expected, the RMSE grows when the R^2 decreases and vice-versa
However, Figure 4.8 shows us that in certain months the prediction obtains impressive results, whereas in other periods of the year the error grows dramatically. The only justification we can find to this behavior relies in the number of different tracks in the predicted month, because even the distribution seems to be very similar.

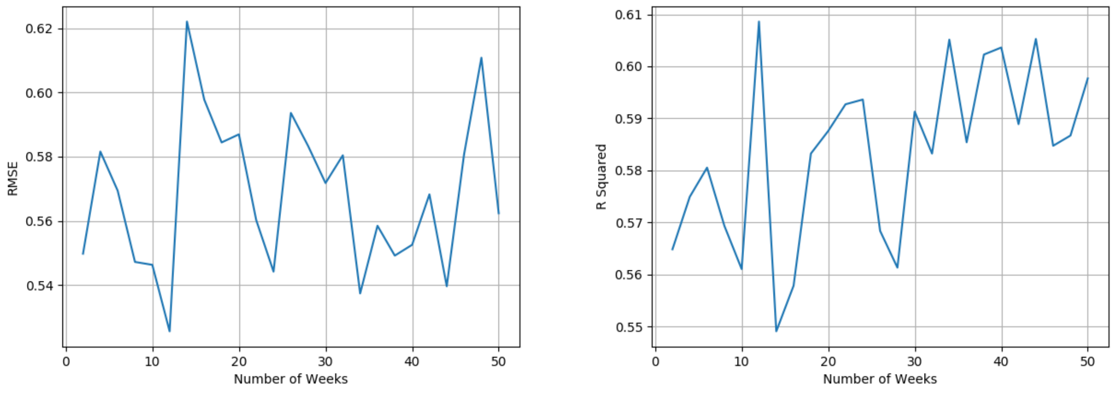


Figure 4.9: The graphs of the RMSE and the R^2 at the variation of the number of weeks used in the training set.

At this point we can examine how the length of the training set affects the prediction. We decided to use as reference the month with the worse results, that is the prediction of the first week of January 2018 using as training set the month of December 2017.
We progressively increased the training set by two weeks, as we did in section 4.2. Once again, the length of the training set seems to not affect much the performances of the prediction, because the RMSE floats around 0.57.
Nevertheless, the small variations in the R^2 seems to have an increasing shape. So, it is possible that the variation of the number of streams has a model that depends also on informations very far in time. However in our opinion Spotify has an always increasing number of user and therefore looking at far days the prediction model can understand better that the number of streams of the tracks in the top 200 are going to generally increase. This could justify why the prediction model get closer to the real model, while the RMSE does not decrease.

Chapter 5

Conclusions

In conclusion, we took a dataset that gave us informations about the 200 most streamed songs on Spotify for every day in every country and we tried to analyze it in order to predict if a track could reach a position in the list of the top tracks of the day and the number of streams that it could achieve in that day.
To do so, we first preprocessed the data, removing, adding and transforming features. Then we developed two algorithms that compute those predictions.
In analyzing these algorithms, we found their strengths and their weaknesses:

- The "Top Songs" prediction gives very good results, but only if it is applied to a single country (or a single region code).
- The "Top Songs" prediction's performances are good even if we use the algorithm to predict the future, training it on the past. Furthermore, it seems to be unaffected by how long in the past we look.
- The "Today's Streams" prediction gives results that floats from good to very poor. It seems to depend on the number of tracks in the dataset.
- The "Today's Streams" prediction model seems to depend on the length of the training set, even if the precision of the results seems not.