The ‘Hit or Flop’ Predictor

Data Spaces Project - *Politecnico di Torino*

Matteo Stoisa - ***s265542***

Introduction

The aim of this study is to analyze and to optimize several models able to predict if a song is a 'hit' (a song that has been very successful) or else a 'flop', a typical example of supervised binary classification. I used a labelled dataset of songs released from the 2010 to the 2020, it is obtained using Spotify APIs and it is available on Kaggle [1]. After initial analysis of the data I applied and optimized several machine learning algorithms both on the entire dataset and on some pruned version of it. I entirely used Python 3 in the Colab environment, in particular I used the libraries: Pandas for data management, NumPy for numeric tools, Scikit-learn for algorithms, Seaborn for graphic representations. Code and results are freely available at
https://github.com/MatteoStoisa/DataSpaces_project/blob/master/projectDataSpacesMatteoStoisa_the_spotify_hit_predictor_dataset_dataset_of_10s.ipynb .

Dataset

The dataset contains 6398 entries each of which represents a song, every entry is initially composed of 18 different features that describe the song and the *target* label: the boolean value I would predict which identifies the song as hit (*1*) or flop (*0*). The following picture shows an example song of the vanilla dataset:

	track	artist	uri	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	chorus_hit	sections	target	featuring
0	Wild Things	Alessia Cara	spotify:track:2ZyuwVvV6Z3XJaXIFbspeE	0.741	0.626	1	-4.826	0	0.0886	0.02000	0.000	0.0828	0.706	108.029	188493	4	41.18681	10	1	0
1	Surfboard	Esquivel!	spotify:track:61APOtq2SSCMuK0V5w2Kgp	0.447	0.247	5	-14.661	0	0.0346	0.87100	0.814	0.0946	0.250	155.489	176880	3	33.18083	9	0	0
2	Love Someone	Lukas Graham	spotify:track:2JqnpexlO9dmvjUMCaLCLJ	0.550	0.415	9	-6.557	0	0.0520	0.16100	0.000	0.1080	0.274	172.065	205463	4	44.89147	9	1	0
3	Music To My Ears (feat. Tory Lanez)	Keys N Krates	spotify:track:0cjfLhk8WJ3etPTCseKXtk	0.502	0.648	0	-5.698	0	0.0527	0.00513	0.000	0.2040	0.291	91.837	193043	4	29.52521	7	0	0
4	Juju On That Beat (TZ Anthem)	Zay Hilfigerrr & Zayion McCall	spotify:track:1lltf5ZXJc1by9SbPeIjFd	0.807	0.887	1	-3.892	1	0.2750	0.00381	0.000	0.3910	0.780	160.517	144244	4	24.99199	8	1	1

Three features are non-numerical (*Track*, *Artist*, *URI*), I removed them from the dataset as useless for my purpose. On the other hand, I considered the fact that the *Artist* record can contain a "*Featuring*" or "*&*" specification in the case the song was made in collaboration with another or more artists, I therefore extracted and created the boolean *featuring* feature.
The dataset columns I used and which I will refer to from here on are the following:

- 7 continuous values between 0 and 1 (*danceability*, *energy*, *speechiness*, *acousticness*, *instrumentalness*, *liveness*, *valence*)
- 2 continuous positive values with different ranges (*tempo*, *chorus_hit*)
- 1 continuous negative value (*loudness*)
- 4 discrete positive values with different ranges (*key*, *duration_ms*, *time_signature*, *sections*)
- 2 boolean values (*mode*, *featuring*)
- the boolean label (*target*)

The technical and musical details of every feature are described in Appendix (1). Figure 1 shows the density distribution of each feature referred to the target label.

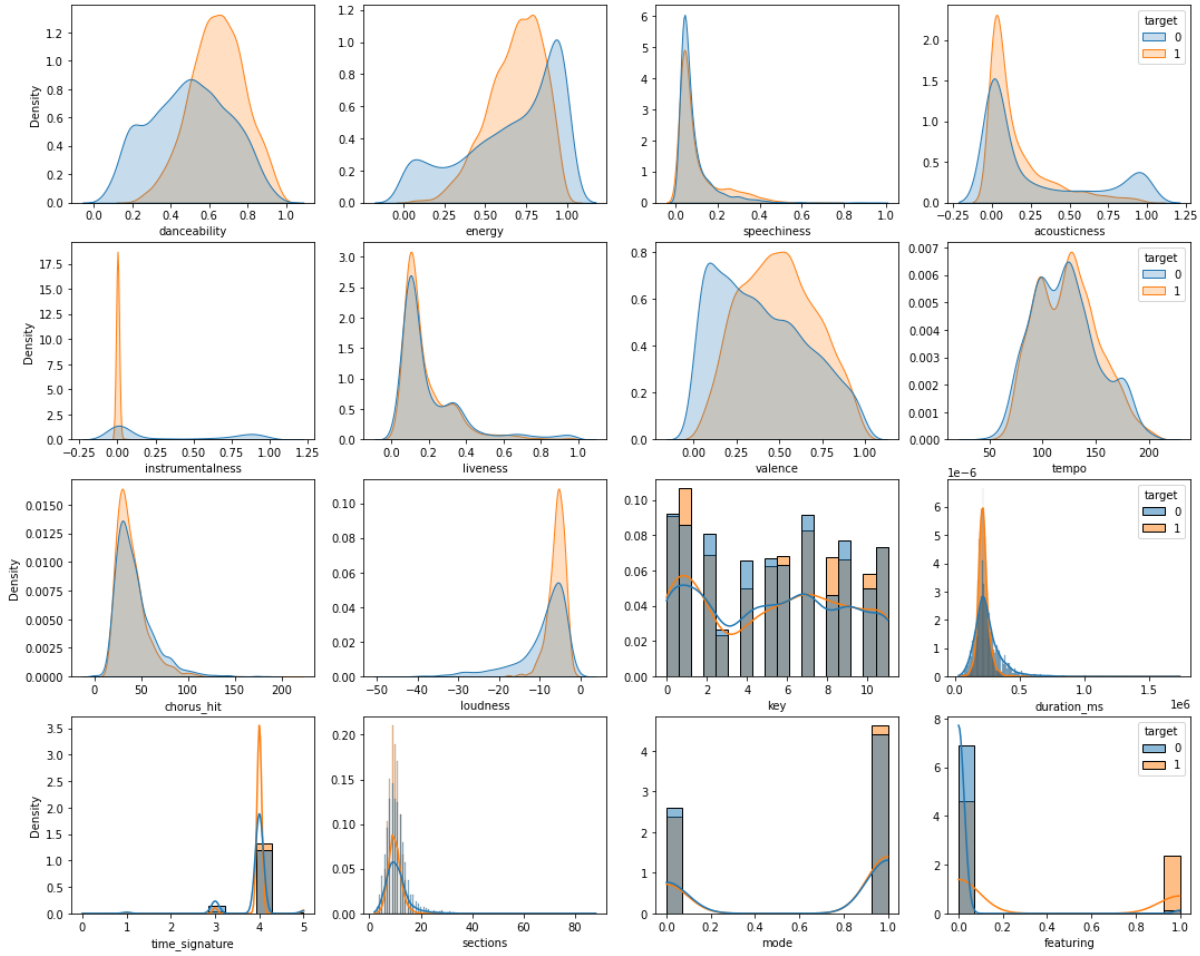

Figure 1: feature distributions, flops in blue and hits in orange

The distribution of the target label is exactly 50% hits and 50% flops, this balance is very positive for model training. There are no missing values.

Some features have very high values (*duration_ms* values have the order of millions), I normalized the dataset to smooth out the diversity of the numerical range between 0 and 1 using the min-max scaler:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x_{min} and x_{max} are the minimum and maximum values of the dimension, naturally the density distributions remain the same as the non-normalized dataset (visible in Figure 1).

Figure 2 is the cluster map built with the correlation matrix between each value, high and low correlations represent the similarity in the two feature's trend but too high or too low correlations would denote the excessive similarity of them and the consequent uselessness of considering them both. Since overall there are not excessively high correlations and the number of dimensions is not excessive, I decided to not remove features.

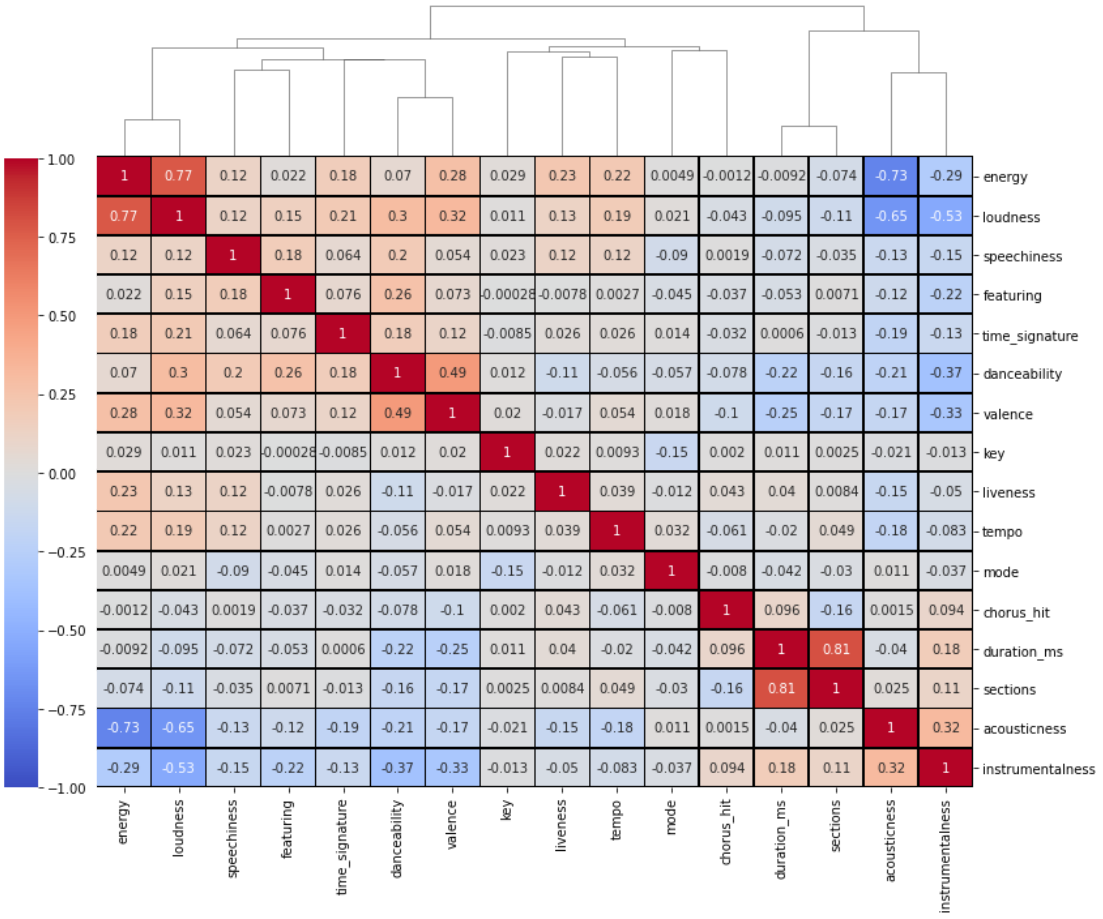

Figure 2: features cluster map

Figure 3 shows correlations between each feature and the target value in ascending order, the fact that there are no correlations that are too high or too low (not greater than 0.5 in absolute value) indicates that the discriminative problem is well posed and not trivial to be predicted.

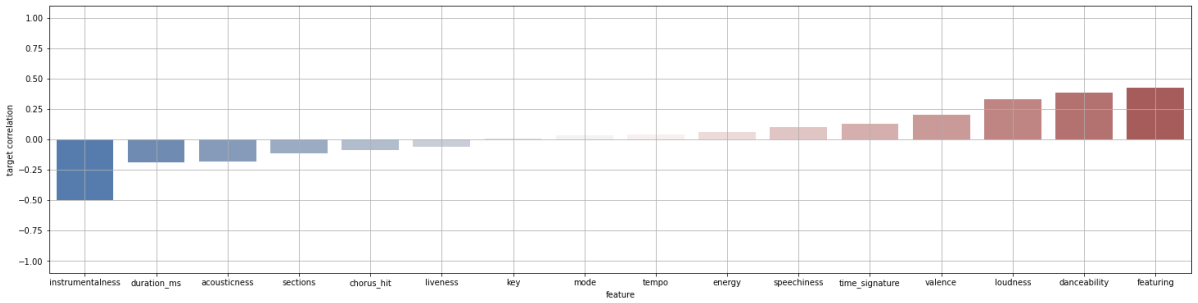

Figure 3: target correlation

In Figure 4 shows boxplots for each feature grouped by target, each boxplot is built with this method: the middle-box line is the median value of the distribution, the semi-boxes under and over the median are delimited by the first and the third quartile (so that each semi-box encloses 25% of the sample distribution), whiskers under and over the box are long 1.5 the measure of the box, samples out of the whiskers limits are considered distribution IQR outliers. It is optimal for the model training and prediction if a feature has different

distributions for hit and flop samples. For example, the *danceability* feature visibly has different medians and boxplot positions for hits and flops, indeed it has a good correlation with the target.

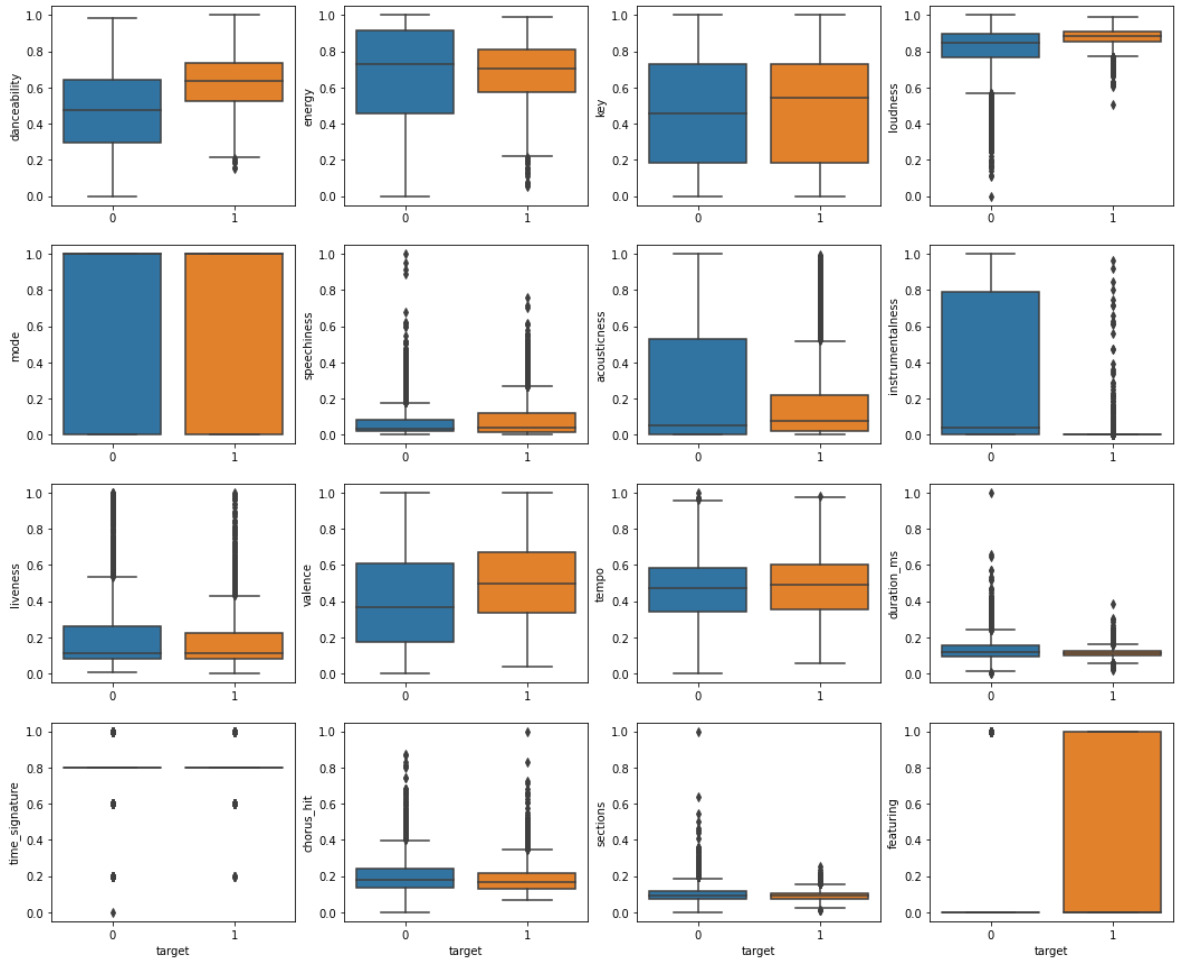

Figure 4: feature boxplots

Principal Component Analysis

The Principal Component Analysis (PCA) is an unsupervised learning technique for dimensionality reduction. Starting from a *d*-dimensional dataset (and ignoring labels) it finds a lower-dimensional representation that preserves as much data information as possible.
If $[X_1, \dots, X_d]$ are the samples of features $(1, \dots, d)$, the first principal component Z_1 is the normalized linear combination

$$Z_1 = \varphi_{1,1}X_1 + \dots + \varphi_{d,1}X_d$$

that has the largest variance; the vector $\varphi_1 = [\varphi_{1,1}, \dots, \varphi_{d,1}]^T$ defines a direction in the feature space along which the data vary the most. After φ_1 has been defined, φ_2 is calculated similarly maximising the variance but with orthogonal direction to φ_1. Iterating this method *d* principal components can be calculated, the first principal components enclose most of data information in terms of variance, the latter can be truncated as they carry little information.

Figure 5 shows the first two principal components projections calculated on the song dataset, samples are grouped by target, the green vectors represent the initial features directions. In an optimal discrimination problem samples of different label form separated clusters, in this case hits and flops have quite different distribution, as density curves suggest.

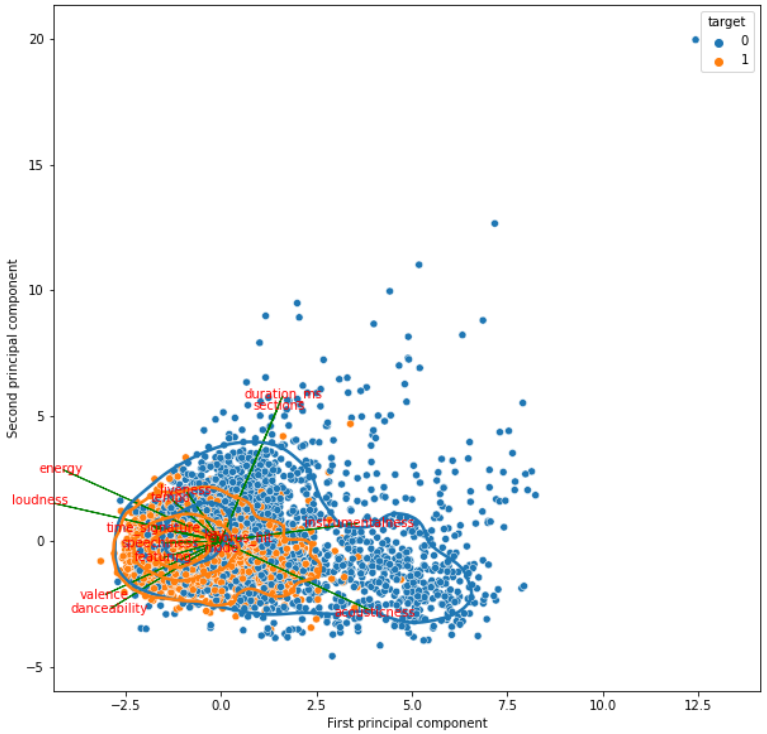

Figure 5: PCA first two principal components

The explained variance expresses how data variance is enclosed in each principal component, as Figure 6 shows, the first principal component carries 20% of the entire variance and the first 5 more than 50%.

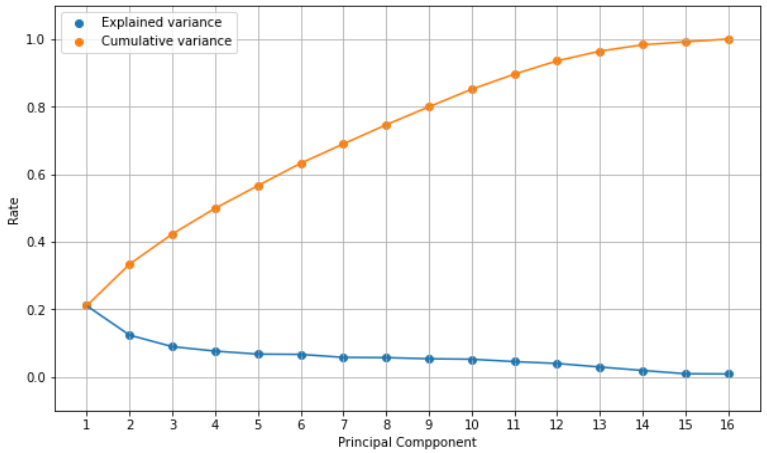

Figure 6: principal components explained variance rate

Since this curve is not very sudden and the initial number of features is not high, the dimensionality reduction does not seem necessary for this dataset.

IQR Outliers

Outliers are the samples that most differ from the pattern distribution, wanting to build a discriminative model it could be useful to drop outliers in order to accentuate the distribution density of the two target clusters, for this reason I analyzed the outliers separately for different targets. The boxplot representation, seen in Figure 4, offers a method to detect outliers that are those out of the upper and lower whiskers, they are out of the limit $1.5\ x\ IQR$ where IQR is the distance of the first and the third interquartile. Figure 7 shows the percentage of IQR outliers per feature grouped by target; high percentages of outliers, up to 10%, are caused by dense distribution, the mean IQR outliers percentage is 1.7% for target 0 and 2.0% for target 1.

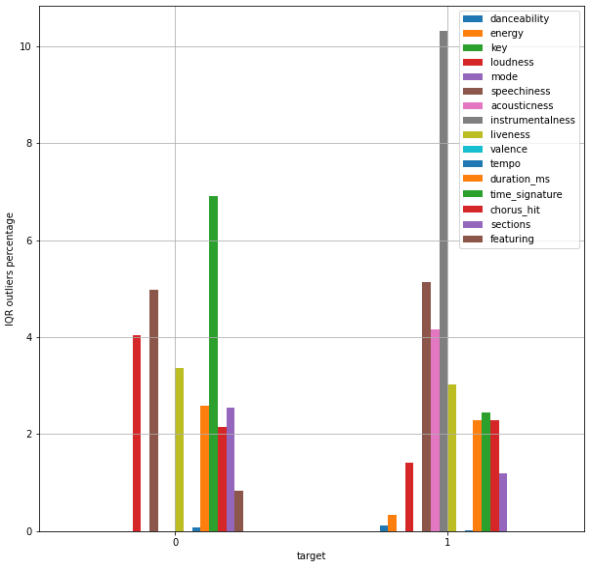

Figure 7: IQR outliers percentage per feature grouped by target

I decided to test the following three approaches of sample cleaning in order to understand if they could be useful, the outliers identification strategies are based on the following scores calculated for each sample:

- Strategy 1: the rate of features marked as IQR outlier over all dataset samples:

$$S_1 = \frac{\#IQR_{out}}{d}$$

  where $\#IQR_{out}$ is the number of features where the sample proves to be an IQR outlier and d is the number of the features, 16 in this case.

- Strategy 2: the sum of sample distances from the median for each feature:

$$S_2 = \sum_i |x_i - m_i|$$

  where x_i is the sample value for the feature i and m_i is the dataset median for feature i.

- Strategy 3: the sum of sample distances from the median of samples with the same label for each feature minus the sum of sample distances from the median of samples with the opposite label for each feature multiplied by a 1.5 factor:

$$S_3 = \sum_i (|x_i - m'_i| - 1.5 * \sum_i |x_i - m''_i|)$$

where x_i is the sample value for the feature i, m'_i is the median for the feature i calculated on samples with the same target as x and m''_i is the median for the feature i calculated on samples with the opponent target.

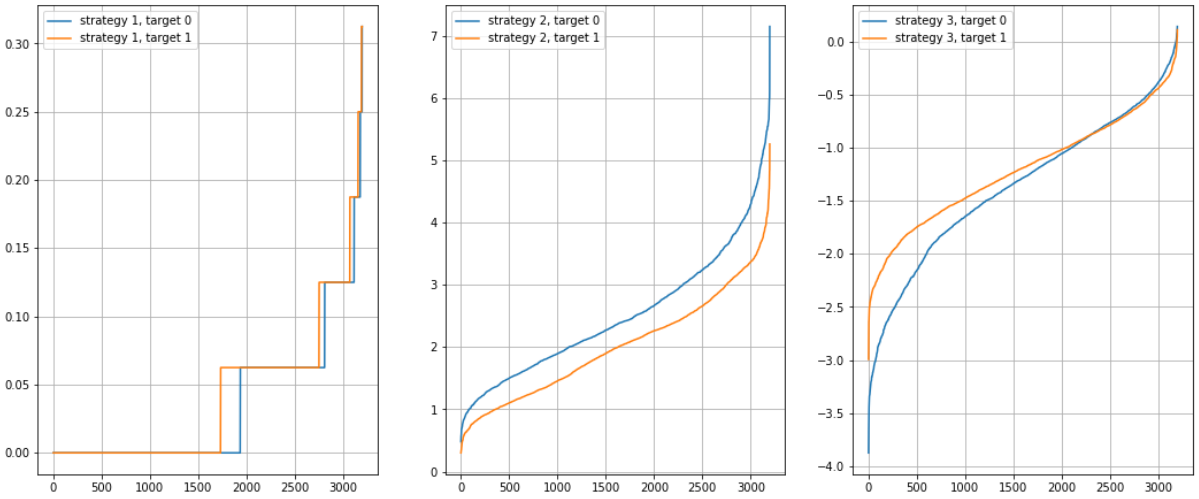

Figure 8: outliers scores

Higher scores identify outlier samples, Figure 8 shows the scores of samples grouped by target. Figure 9 and 10 show the representation of the first two principal components obtained applying respectively strategy 1 and 2 with a 0.75 drop rate, obviously the 75% drop is huge and it is done only to observe the limit behavior of strategies. Those two strategies do not consider labels, so that clusters become less dense but they do not distance themselves.

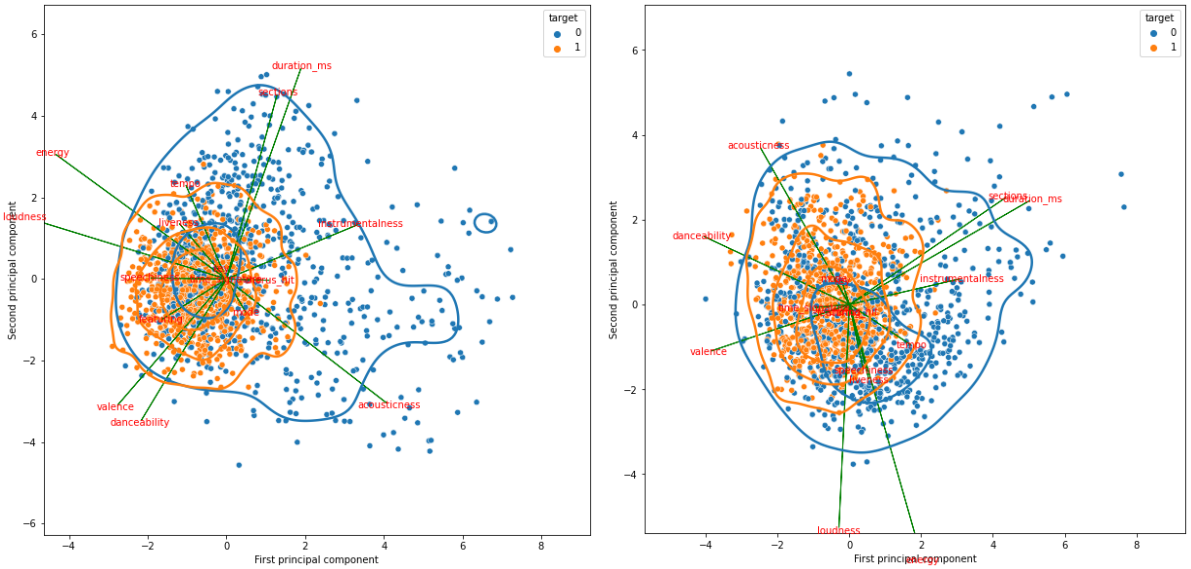

Figure 9: first two principal components obtained with strategy 1 (75% of the dataset dropped)

Figure 10: first two principal components obtained with strategy 2 (75% of the dataset dropped)

On other hand, the third strategy distances the two clusters dropping samples labelled as 0 that are similar to samples with label 1 and vice versa, this behaviour could improve the accuracy of discrimination models. This distancing is evident in four significant feature's violin-plots (Figure 11), the third strategy sample drop separates the distributions grouped by labels. In Figure 12, that shows the first two principal components, label clusters are much more separated than in the vanilla dataset. The outlier drop is performed removing a rate of samples with higher score per label so that the label proportion is kept equal.

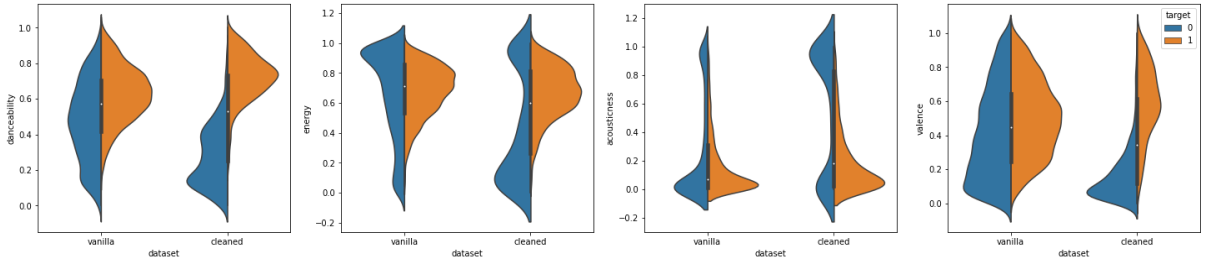

Figure 11: violinplots of four features, for vanilla and strategy 3 dataset (75% dropped)

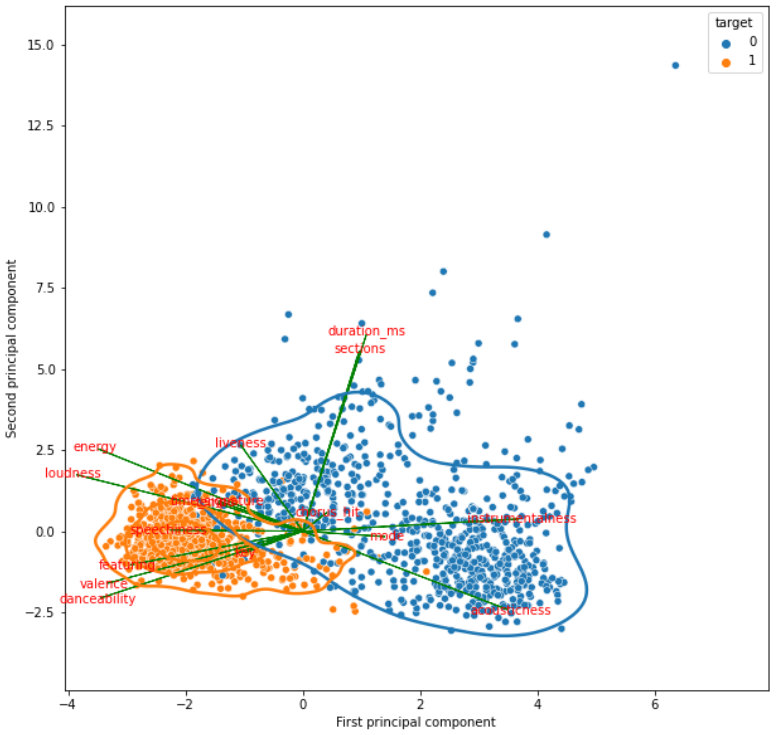

Figure 12: first two principal components for strategy 3 dataset (75% dropped)

Train-Validation-Test Splits

The following picture shows as example the first five entry of the dataset after the min-max normalization, called vanilla dataset:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	chorus_hit	sections	featuring
0	0.738790	0.626533	0.090909	0.899432	0	0.070809	0.020080	0.000000	0.068476	0.723361	0.400098	0.093080	0.8	0.193225	0.093023	0
1	0.418807	0.247058	0.454545	0.687954	0	0.012962	0.874498	0.818090	0.080700	0.256148	0.676658	0.086266	0.6	0.155665	0.081395	0
2	0.530910	0.415269	0.818182	0.862211	0	0.031601	0.161647	0.000000	0.094582	0.280738	0.773251	0.103036	0.8	0.210605	0.081395	0
3	0.478668	0.648560	0.000000	0.880682	0	0.032351	0.005151	0.000000	0.194033	0.298156	0.305743	0.095749	0.8	0.138515	0.058140	0
4	0.810623	0.887860	0.090909	0.919516	1	0.270487	0.003825	0.000000	0.387755	0.799180	0.705958	0.067117	0.8	0.117248	0.069767	1

From the vanilla dataset, composed of 6398 entries, I extracted 15% of samples for the test-set; it remains the same for each procedure in order to better compare results. The training of each model is performed using a 5-fold cross validation, so that the model that performs better on the validation-set is applied to the test-set. For the pruned datasets, I decided to use drop rates 0.15, 0.25 and 0.4 for each drop strategy, as said the sample pruning is performed only on the train-set and in each split the rate of labels are kept equal. The percentages of the dataset used for the train-validation-test iterations are shown in Figure 13. Each algorithm can be fitted using different parameters, each algorithm is tested using all combinations of parameters reported below in order to find the best performing combination.

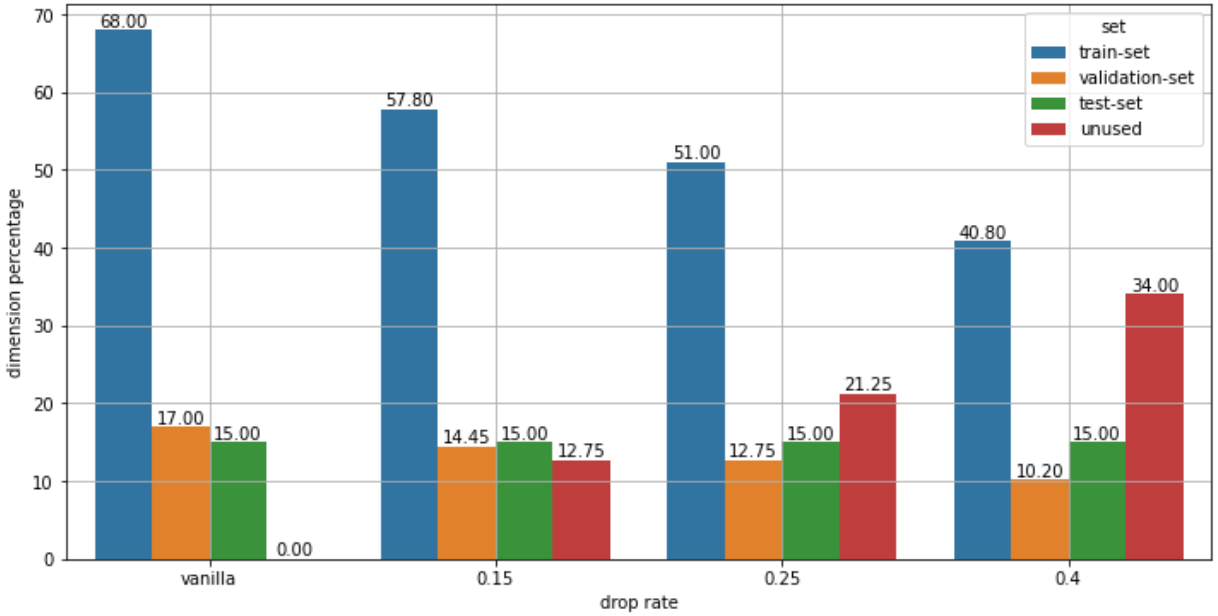

Figure 13: dimension of splits for each procedure

Algorithms

I applied the following discrimination algorithms:

- **Logistic Regression**: the probabilities describing the possible outcomes of a single trial are modeled using the logistic function:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

where X_1 through X_p are distinct independent variables and β_0 through β_p are the regression coefficients. After being fitted, the function gives the label prediction as probability between 0 and 1. The logistic regression classifier is fitted using the following parameters: the solver is the algorithm used for the optimization problem ('*newton-cg*', '*lbfgs*', '*liblinear*', '*sag*' or '*saga*'); the penalty specifies the norm used in the penalization ('*l1*', '*l2*', '*elasticnet*' or '*none*'); the parameter C inverse of regularization strength (*0.001*, *0.005*, *0.01*, *0.05*, *0.1*, *0.5*, *1*, *5*, *10* or *50*) where smaller values specify stronger regularization.

- **K-Nearest-Neighbor**: the KNN classifier labels a sample as the K nearest samples from him are labelled the most. It takes as parameters the value K (odd integers from *1* to *19*), the distance metric ('*euclidean*', '*manhattan*', '*minkowski*') and the weights ('*uniform*', '*distance*') in case further samples have less relevance.
- **Gaussian Naive Bayes**: Naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, the likelihood of the features is assumed to be gaussian

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

where the parameters σ_y and μ_y are estimated using maximum likelihood, then the label is chosen using

$$\hat{y} = \underset{y}{argmax}\, P(y) \prod_{i=1}^{n} P(x_i|y)$$

The parameter var_smoothing (*1e-2*, *1e-3*, *1e-4*, *1e-5*, *1e-6*, *1e-7*) is the portion of the largest variance of all features that is added to variances for calculation stability.

- **Random Forest**: the prediction is the class selected by most decision trees, each decision tree is created from different subsets of data (thanks to bagging: using multiple times the same sample) but also using different features to make decisions. The parameter n_estimators indicates the number of single decision trees (*250, 500, 750, 1000, 1250, 1500*) and the max_features parameter ('*sqrt*', '*log2*') indicates the number of features randomly used to build a tree out of the total number of features.
- **Support Vector Classifier**: the Support Vector Machine is based on the idea of finding a hyperplane that best divides the dataset into the two classes. Hyperplane coefficients are calculated to maximize the sum of the distances from points of different labels ("hard margin") but assuming that some points exceed the limit ("soft margin"), different functions called kernels transform non-linearly separable data into separable data. Parameters are the kernel type ('*poly*', '*rbf*', '*sigmoid*') and the C parameter (*from 1 to 50 with 1 step*) that is inversely proportional to the strength of the regularization

Results

Figure 14 shows the accuracy scores obtained on the test-set with models trained on the vanilla dataset, scores are generally quite high: the Random Forest algorithm is the best with 85.6% of successful prediction.

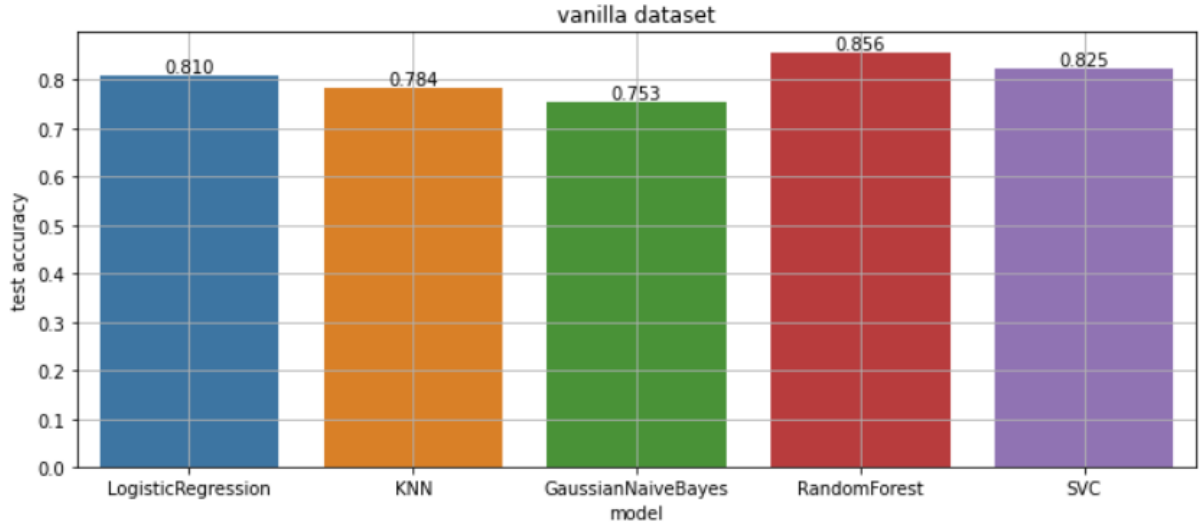

Figure 14: accuracy scores on test with vanilla dataset train

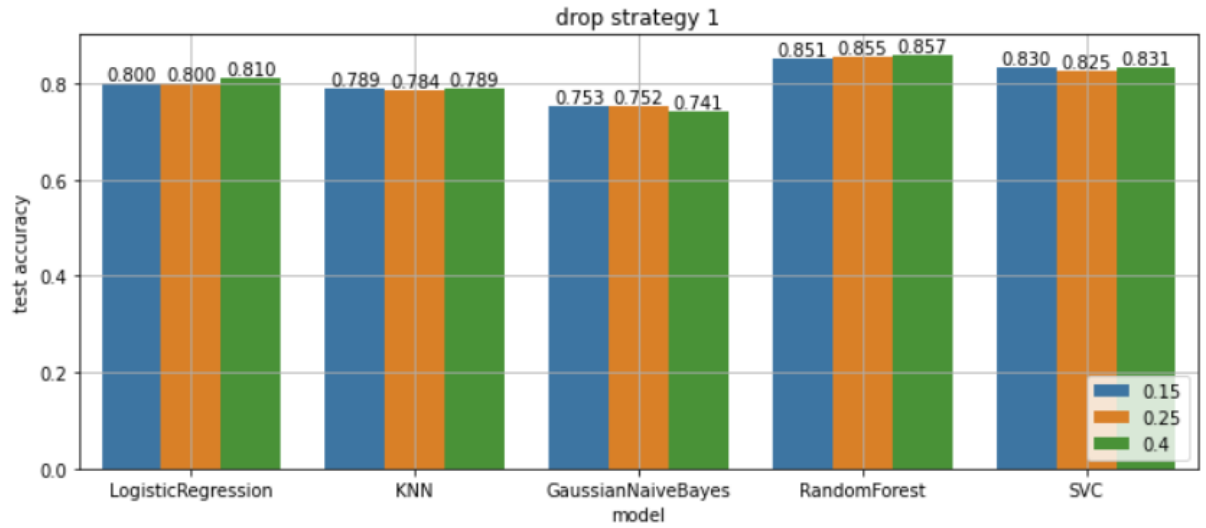

Figure 15: accuracy scores on test with strategy 1 pruned dataset train

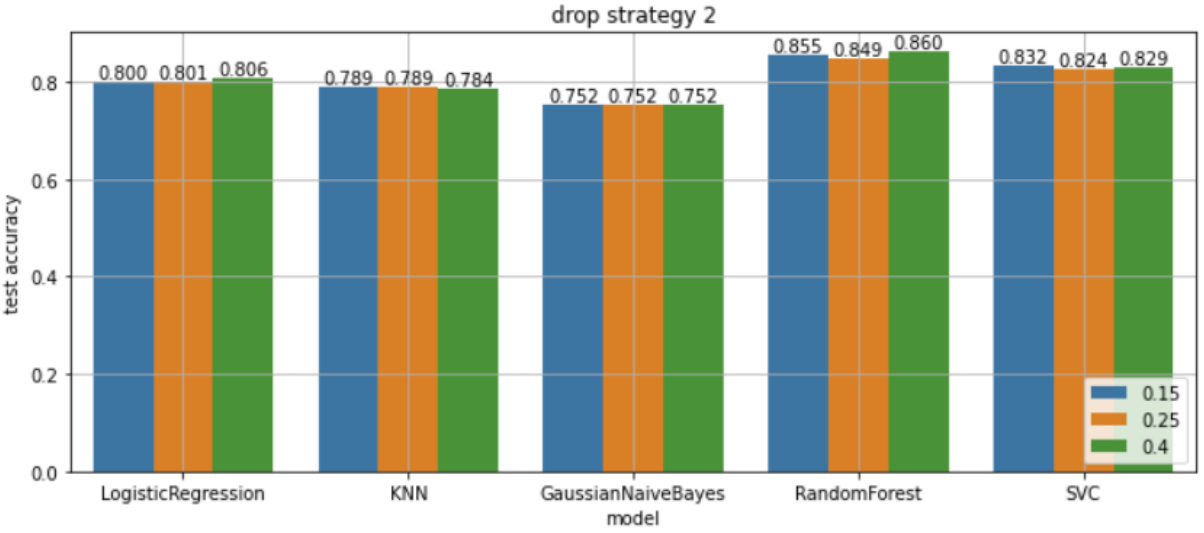

Figure 16: accuracy scores on test with strategy 2 pruned dataset train

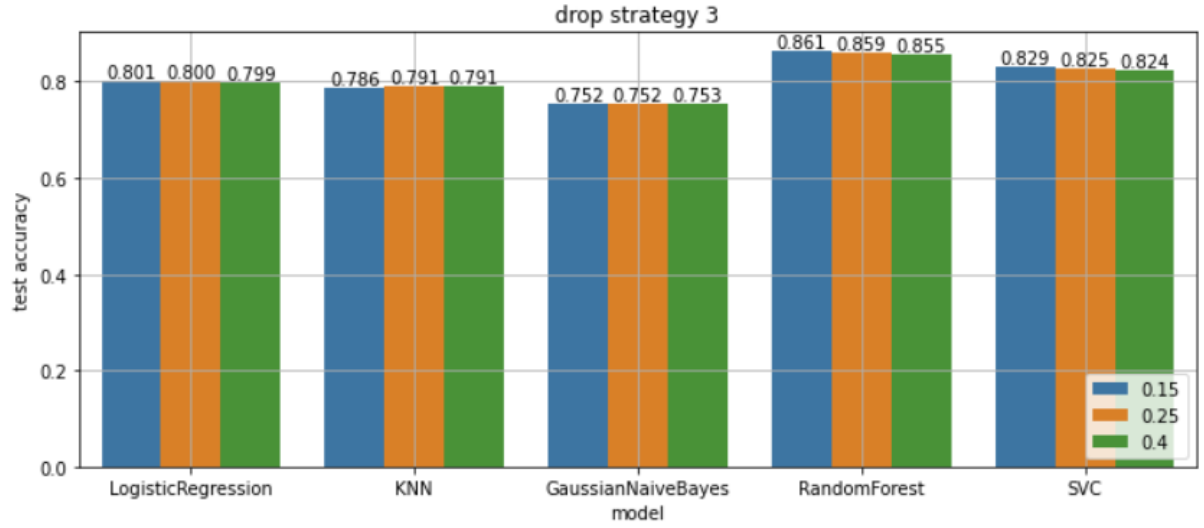

Figure 17: accuracy scores on test with strategy 3 pruned dataset train

Figures 15, 16 and 17 show accuracy scores obtained with the pruned datasets, each graph grouped by amount of pruning. The best result is reached again with the Random Forest algorithm, using the third drop strategy pruning 15% of train samples: it achieves 86.1% of accuracy. For each algorithm and considering every type of pruning and pruning percentage the result differs in magnitude by no more than 1.2% (only ~10 single samples). Keeping the vanilla dataset as reference in many cases the use of pruning degrades performance, especially for Logistic Regression and Gaussian Naive Bayes. Furthermore, contrary to what one might think, it is interesting to see that using the third drop strategy, that drops taking into account the label, the accuracy decreases in many cases with higher drop percentage, this probably because overfitting is introduced.

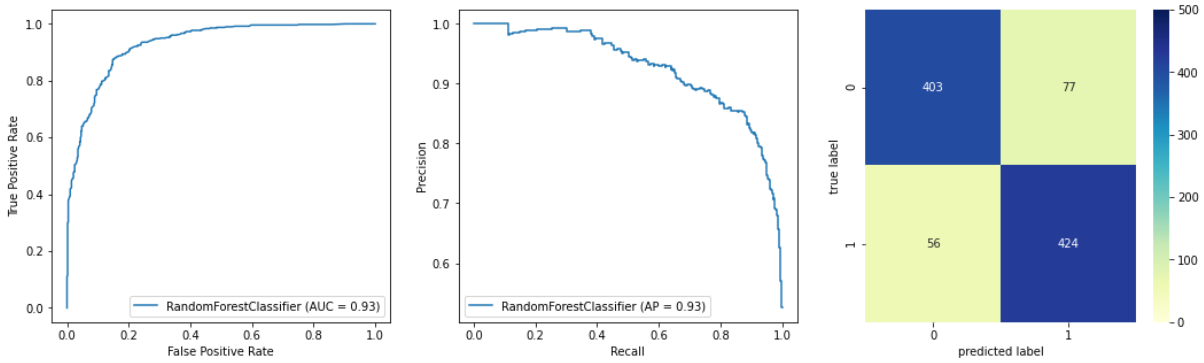

Figure 18: statistics of Random Forest model on vanilla dataset

Figure 18 shows statistics for the best Random Forest model: the confusion matrix (on the right) is a chart that allows us to compare the number of predictions with the true labels. The ROC curve (on the left) is a plot that represents the variance of true positive rate and false positive rate; the area under the graph is the AUC metrics: an excellent model has the AUC close to 1 which means it has a good measure of separability and vice versa. The graph on the middle shows precision and recall metrics calculated as:

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

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

Average precision (AP) summarizes such a plot as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight:

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n$$

where P_n and R_n are the precision and recall at the nth threshold.
In models that have performed worse it is evident that it is more difficult to identify the samples labeled as 0 rather than samples labelled as 1, as visible for example in figure 19.

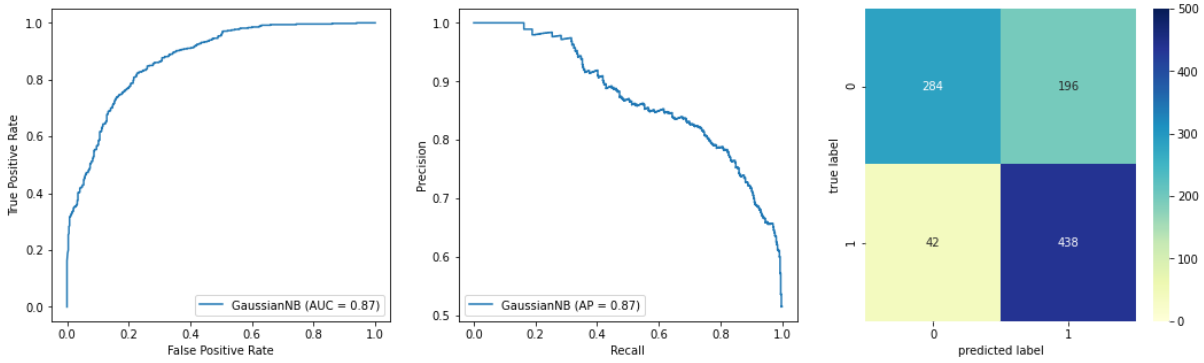

Figure 19: statistics of Gaussian Naive Bayes model on 25% strategy-3-dropped dataset

Conclusions

The used algorithms show very good performances even though the discriminatory problem is not trivial, it must be said that the exact balancing of the labels facilitates a lot. The tested

drop strategies seems, after all, superfluous probably because the dataset contains balanced samples and has no outlier problems; they also risk leading to overfitting.

References

[1] https://www.kaggle.com/theoverman/the-spotify-hit-predictor-dataset#dataset-of-10s.csv
[2] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[4] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Appendix

1. Features meaning:
   - *track*: The Name of the track.
   - *artist*: The Name of the Artist.
   - *uri*: The resource identifier for the track.
   - *danceability*: 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 least danceable and 1.0 is most danceable.
   - *energy*: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
   - *key*: The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C?/D?, 2 = D, and so on. 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 relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
   - *mode*: 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*: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, 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. The distribution of values for this feature look like this:
   - *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. The distribution of values for this feature look like this:
   - *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 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.
   - *duration_ms*: The duration of the track in milliseconds.
   - *time_signature*: An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
   - *chorus_hit*: This is the author's best estimate of when the chorus would start for the track. It's the timestamp of the start of the third section of the track. This feature was extracted from the data received by the API call for Audio Analysis of that particular track.
   - *sections*: The number of sections the particular track has. This feature was extracted from the data received by the API call for Audio Analysis of that particular track.
   - *featuring*: It indicates if the song was made by a single artist (0) or in collaboration with one another or more artists (1).
   - *target*: The target variable for the track. It can be either '0' or '1'. '1' implies that this song has featured in the weekly list (Issued by Billboards) of Hot-100 tracks in that decade at least once and is therefore a 'hit'. '0' Implies that the track is a 'flop'.