Machine Learning Engineer Nanodegree

Capstone Project

João Pedro Megid Carrilho
March 27th, 2018

I. Definition

Project Overview

Machine learning is one of the most talked subjects these days, there are so many applications of this field that often entrepreneurs get lost and, applying this tools to your product and/or service brings many advantages in this ferocious market competition.

This project seeks to analyze and construct an accurate predictive model for Kaggle's "Spotify Song Attributes" dataset, a set that contains 2017 songs from a single user, each song has its own attributes and is labeled as "liked or not". From the patterns between songs attributes it's possible to use machine learning to build a model that can label other songs, from the preferences of this user and say if him/her may like it or not. This book mentions this concept of the project at section 2.8.

Problem Statement

Selling a service or a product (in this case music subscription) and keeping up with the competition is harder each day but, when you can make good use of each data and feedback a user gives you, the use of good tools like machine learning can help you win the clients' money and fidelity.

The need to classify songs in "liked or not" directs the problem to the use of discrete label classifiers, in this case, LinearSVC, SVC, KNeighboors, and Ensemble Methods (AdaBoost). These models analyze the patterns between song features and from these they can extrapolate to new songs, the goal is to find the model that best fit the purpose of suggesting music.

Before the use of the previously mentioned models, the dataset has to be analyzed and prepared, statistical general values are evaluated, histograms of each feature, the data is split into two sets: The training set, where the model will be trained to find parameters that can separate songs in songs liked or not (from this portion of data); The test set, where the model will score its predictive capacity. Feature scaling, features relevance tools are used to perform transformations in the data with the goal to help the predictive algorithms achieve better results.

After this data preparation, the chosen models will be have their performance evaluated, the higher accuracy models will have their hyperparameters tuned with the objective to raise their performance in making predictions.

At last the results will be compared and the best model will be chosen.

Metrics

One of the metrisc used in this project is the *mean accuracy of predictions* made by the Grid Search CV best estimator, for a better analysis, precision and recall will help in choosing the best model:

$$Mean\ Accuracy\ = \frac{1}{n} * \sum_{0}^{i} \frac{True\ Positives_{\ i} + True\ Negatives_{\ i}}{Total\ fold\ samples\ tested}$$

Where “n” is the number of cross-validation folds, $true\ positives_{\ i}$ is the music that was correctly predicted, in the “i” fold, as 1(one) or “being liked” and, the $true\ negatives_{\ i}$ is music correctly predicted, in the “i” fold, as 0(zero) or “not liked”.

This metric will be used because of its good and easy representation to evaluate the models performance and because this dataset has a balanced category of target labels, 1020 songs with the “like” label and 997 songs with “don’t like”.

As auxiliary measures, precision and recall will be evaluated. In pattern recognition, information retrieval and binary classification, precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances.

Precision is defined as:

$$Precision\ = \frac{TN}{TN + FP}$$

Recall is defined as:

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

Where:

- TN = True Negative
- TP = True Positive
- FN = Fake Negative
- FP = Fake Positive

These measures were selected because, as the project aims a model with good accuracy, it’s good also to have a model with high precision in suggesting songs and not so much recall, this way the provided songs have more confidence in being liked.

II. Analysis

Data Exploration

This dataset contains 2017 musics/samples (rows) from a single user, each of this samples has 16 columns of which thirteen are music attributes, one column for the songs name (a string), one for the artist name (also a string) and a 'target' column, as our target variable, that classifies if the user likes or not this song, a binary classification. The attributes of each song are the following: acousticness, danceability, duration_ms, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time_signature and valence, all these are continuous/discrete numerical values. This dataset contains 1020 songs with the "like" label and 997 songs with "don't like" (totalizing the 2017 samples). For a deeper look into what each attribute means, this link at the Spotify's website.

The trouble in preparing the columns 'song name' and 'artist name' is that it's not interesting to convert each artist into a feature, since it would create too many of these and develop a need for higher computational power, if we use label enconding (as #2 approach on this post), even on 'song name', it will create numerical similarities for the machine learning algorithms, between the feature value's and since there is no understanding between these similarities it's better not evaluate them. Thus, data analysis will be made only with the song's attributes.

The song attributes will be explored with the use of feature relevance tools, feature scaling and feature transformation to apply dimensionality reduction if needed.

Below a table with some major statistical values from the features that will be analyzed, obtained using Pandas.describe():

Feature	count	mean	std	min	max
acousticness	2017	0.18759	0.259989	0.000003	0.995
danceability	2017	0.618422	0.161029	0.122	0.984
duration_ms	2017	2.46E+05	8.20E+04	1.60E+04	1.00E+06
energy	2017	0.681577	0.210273	0.0148	0.998
instrumentalness	2017	0.133286	0.273162	0	0.976
key	2017	5.342588	3.64824	0	11
liveness	2017	0.190844	0.155453	0.0188	0.969
loudness	2017	-7.085624	3.761684	-33.097	-0.307
mode	2017	0.612295	0.487347	0	1
speechiness	2017	0.092664	0.089931	0.0231	0.816
tempo	2017	121.603272	26.685604	47.859	219.331
time_signature	2017	3.96827	0.255853	1	5
valence	2017	0.496815	0.247195	0.0348	0.992
target	2017	0.505702	0.500091	0	1

In the ‘Exploratory Visualization’ section, the histogram of each of these features will be shown and other relevant observations regarding the data are going to be made.

Exploratory Visualization

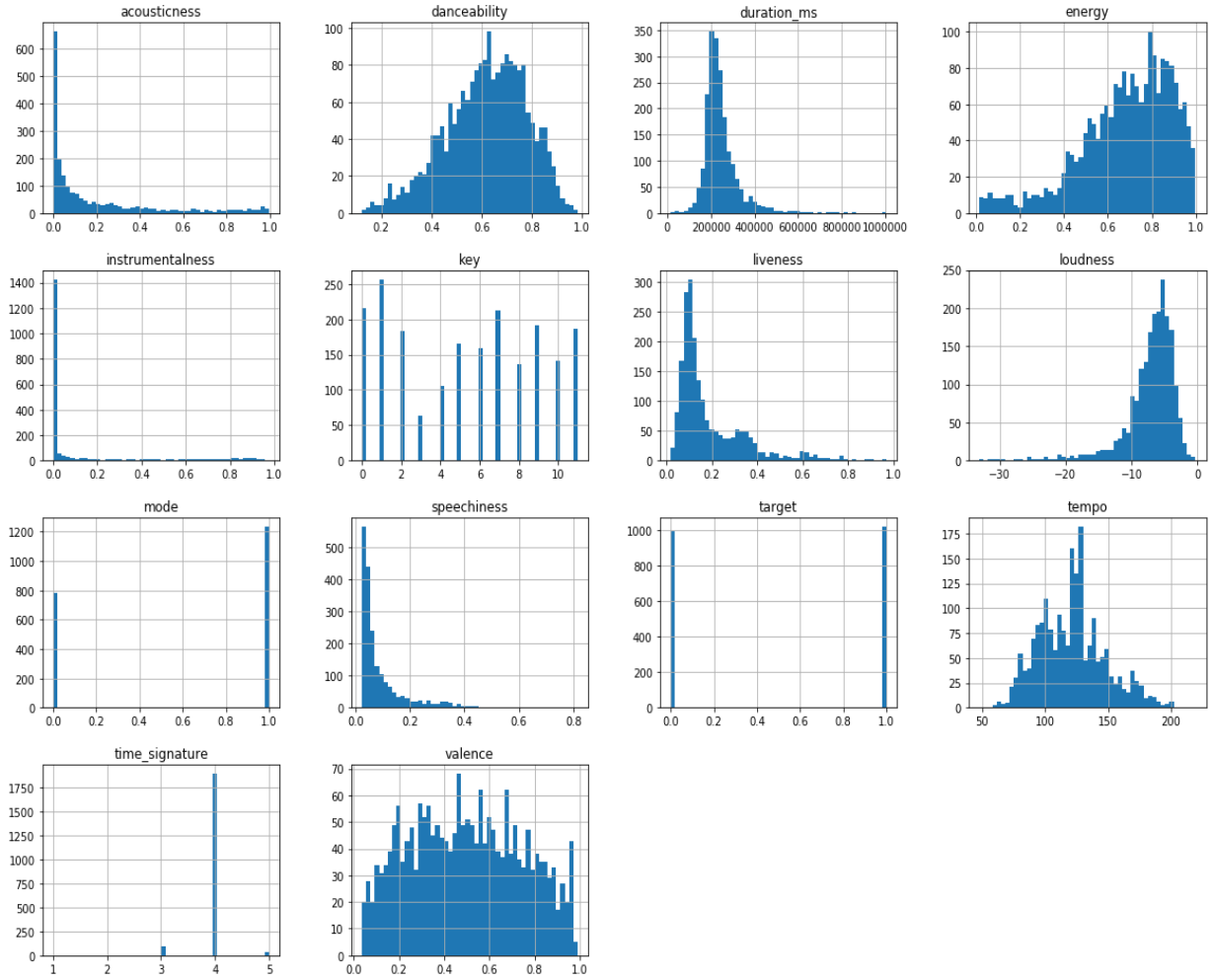

Image 2.1: Spotify data histogram. Taken from the project’s Notebook.

This histogram was made with Pandas method “.hist()” in the whole dataset. It shows important things: some features have a skewed distribution, ‘acousticness’, ‘duration_ms’, ‘loudness’ and others; Other features have discrete values, ‘mode’ and ‘time signature’; At last ‘valence’, ‘danceability’, ‘tempo’ and energy have a close to normal distribution

This visualization also shows that features have different value scales and that feature scaling will have to be used since some machine learning algorithms are sensible and perform poorly on unscaled datasets. The skewness of the data observed in some features can be

handled with outlier detection but, for the purpose of this project, it's not interesting to drop samples and lose information with target labels, in the next section this issue will be referenced.

Algorithms and Techniques

There are four algorithms chosen to solve this binary classification problem, all can be found in Sci-kit Learn API REFERENCE:
:

1. **Linear Support Vector Classification(LinearSVC):**

Support Vector Machines are powerful and versatile algorithms that can handle very well linear and non-linear classification problems in complex small-medium datasets (this case). The LinearSVC fits the data using *large margin classification*, it creates a *linear decision boundary* that separates two classes (music liked or not) in such way that a *margin* between the instances and their respective categories are well distanced and grouped, example below.

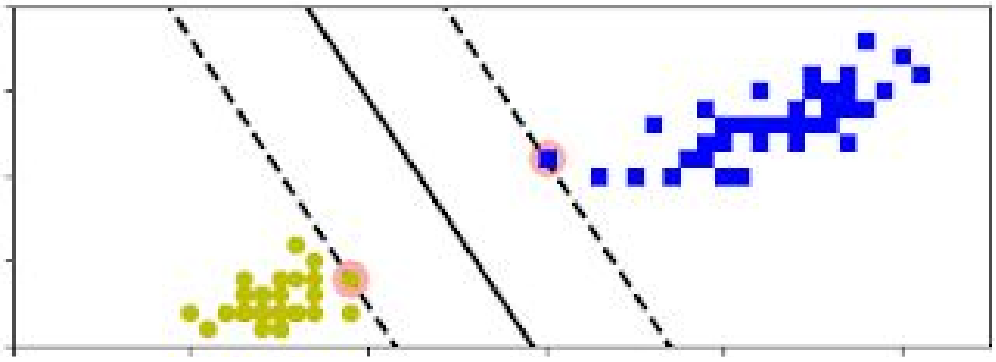

Image 2.2: Data fitted and separated with a LinearSVC (This image was taken and edited from this GitHub repositorie notebook, from GÉRON, Hands-On Machine Learning with Sci-KitLearning & TensorFlow book).

This type of data is *Linearly Separable* and it's the type of set that LinearSVC performs well. The main objective is to find a good balance between the *margin* size and *margins violations*, these can be adjusted with hyperparameters "*C*" a higher C provides a smaller margin and results in more violations. The margin created by SVM is interesting because the model, after trained, has fully determined boundaries and these can generalize well new instances.

Support Vector Machines are sensible to outliers and feature scaling, in the technique section this concerns will be mentioned. The following image shows how SVM reacts to feature scalling:

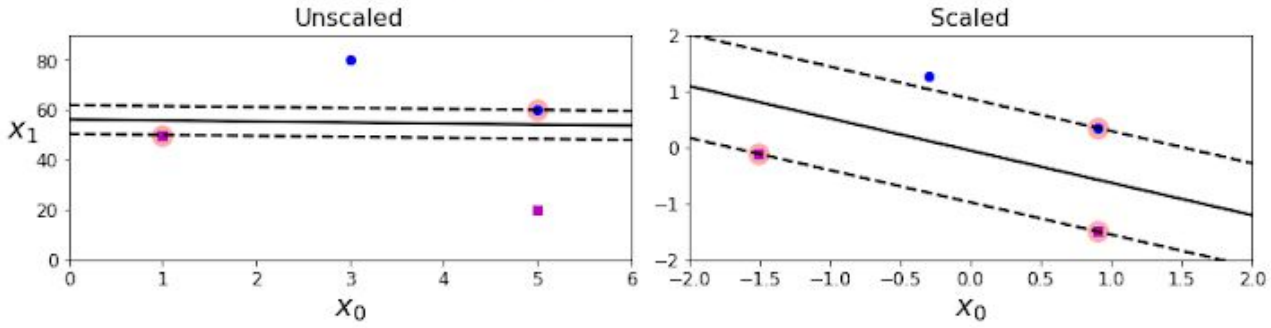

Image 2.3: How SVM reacts to feature scaling.(This image was taken and edited from this GitHub repository notebook, from GÉRON, Hands-On Machine Learning with Sci-KitLearning & TensorFlow book.)

2. **C-Support Vector Classification.(SVC):**

When the data is not Linearly separable (with a linear decision boundary), you can still use support vector machines with the help of *feature transformations*, *similarities functions* and *kernel tricks (*used in this project), this model has good characteristics and should have good performance on this problem.

The kernel trick is a mathematical technique that makes use of non-linear feature combinations without actually doing the transformation in the original set, this way the model can use complex operations more efficiently. *Image 2.4* shows an SVC using a GaussianRBF kernel:

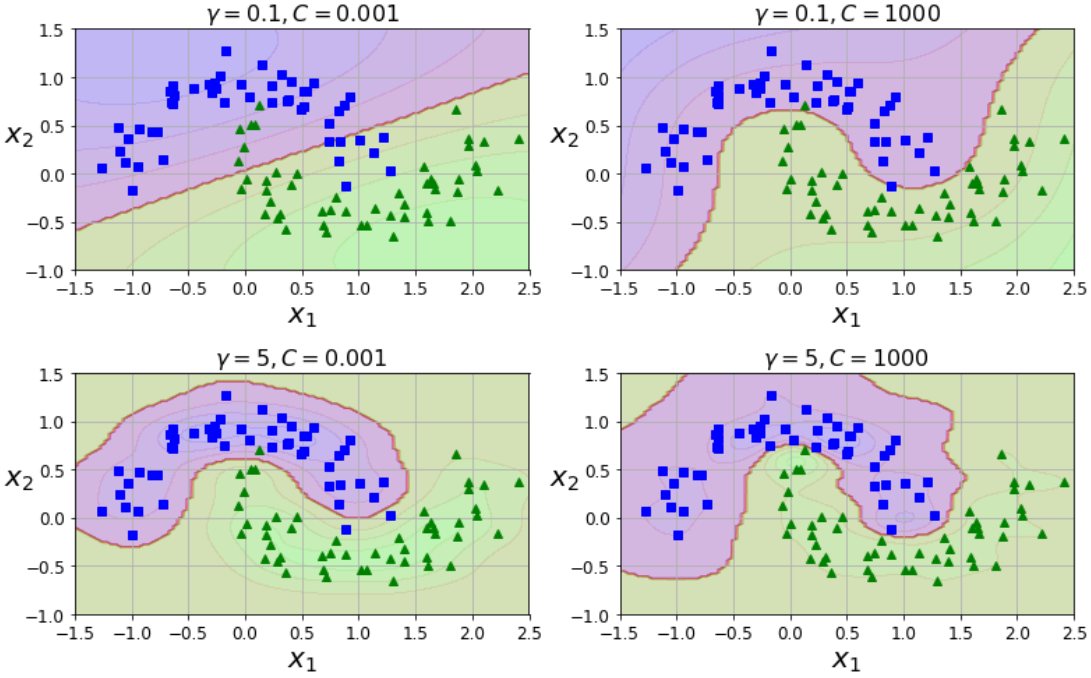

Image 2.4: SVC using a GaussianRBF Kernel with different (gamma) and C hyperparameters (Image taken and edited from this GitHub repository notebook, from GÉRON, Hands-On Machine Learning with Sci-KitLearning & TensorFlow book).

The hyperparameter C is the analogous to the one explained in Linear Support Vector Machines, gamma acts as a regularizer, if the model is overfitting, you should reduce it, else, raise it.

3. **AdaBoost Classifier(AdaBoostClassifier):**

This boosting algorithm trains a group of predictors (*an ensemble)* called *weak learners* in random subsets of the training data and, with an iterative weighted strategy, it focuses on model's misses, *boosting* the weight on these so the next cycle can do a better job in classifying them. Generally, the *boosting* technique results in better results than models solely.

The predictors used in this project are the *DecisionTreeClassifier* and *Random Forest Classifier.* The use of AdaBoost with Random Forest can be viewed in this article, a Random Forest Classifier is an ensemble of Decision Trees.

The hyperparameters explored in this project's model are: The '*base_estimator"*, this sets the predictor used by the *ensemble*, '*n_estimators*', sets how many *weak learners* are going to be trained and *'learning_rate' this* parameter has a tradeoff reaction with '*n_estimators*', it sets the contribution of each *weak learner* in the decision, generally, the small it gets, more weak learners are needed.

4. **K-nearest Neighbors(KNeighborsClassifier):**

KNN is one of the simplest machine learning algorithms, in the feature vector space, the learner makes use of a voting process regarding the points next to the instance that is being evaluated, see the image below:

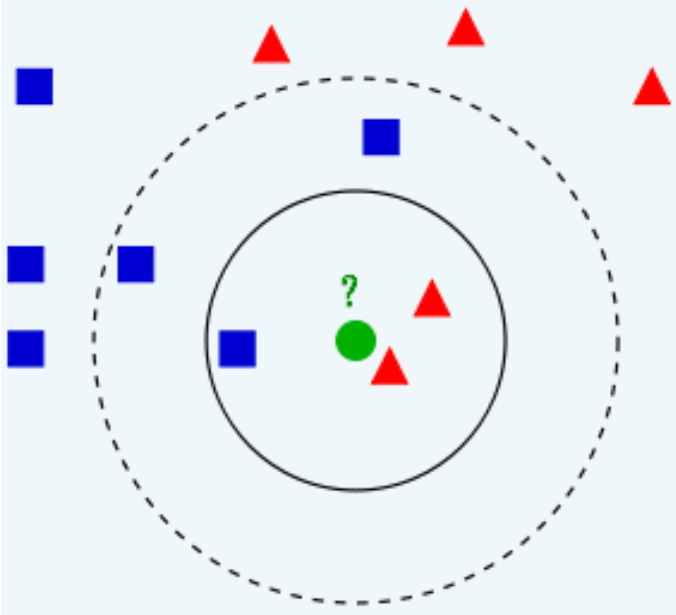

Image 2.5: Example of *k*-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If *k = 3* (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If *k = 5* (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle). Image and text are taken from Wikipedia.

This project may evaluate hyperparameters including, *'metric''* (the type of distance measure used between samples), *'n_neighbors'* (number of near points used to vote), *'weights'* (defines if the votes have uniform weights or if nearer points have more relevance).

Also, to help solve this problem, the following techniques were implemented:

1. **Feature Scaling**

Feature Scaling is one of the most important *transformers* that the data need to pass through, in this project *Standardization* is used. Sci-Kit's StandardScaler() method is applied to the training set, each feature's mean is turned null by subtracting the original mean in each sample and then it divides by the variance so that the resulting distribution has unit variance. This technique is preferred over other *normalization* processes because it's not much affected by outliers.

2. **Feature Relevance**

After scaling the data feature relevance was used in the *scaled_data* set with the objective to analyze if there are major correlations between features and the target label or if there are features that have no relevance in this problem thus, can be ignored. With Panda.corr() method, the correlation matrix of a data frame is computed, it returns the *Pearson's r* values between features on a scale of 1 (for directly related) to -1 (inversely related), 0 (zero) means no relation. In this project, only the 'target' column was evaluated.

3. **Grid Search Cross-Validation**

Sci-Kit's GridSearchCV() is a very useful *class,* after training and selecting the models with the best performance, this tool can be used to make a search (in a dictionary provided by the developer) of the best combinations of hyperparameters for the model, by evaluating them in a cross-validation, this way it's possible to be more confident in the predictions obtained by the algorithm.

4. **Confusion Matrix**

As the *Metrics* section presented, *Precision* and *Recall* are important measures to be evaluated and the *Confusion Matrix* can help us analyze these. Sklearn's *confusion_matrix* makes it easy to analyze the True/False Positive/Negative numbers of predictions for each model, helping to select the one that fits the best. Each row in the confusion matrix represents an *actual class* (songs liked or not) while each column represents a *predicted class* (songs correctly or not predicted in their class).

5. **ROC Curve Analysis**

The *receiver operating characteristic* curve is a very useful tool in binary classification. Sklearn's *roc_curve()* method easily returns the values needed for the plot, the only adjustment needed to make in the models is that, instead of using the labels predicted by the algorithms, the *decision_function()* values are used, this way the method can adjust the *threshold of decisions* and evaluate how de precision/recall trade-off occurs.

Benchmark

For use of supervised learning model, on the same dataset, this Kaggle submit used Decision Tree and Random Forest algorithms to make predictions, resulting in 0.6998 and 0.7295 of accuracy (the number of predictions made right divided by the total number of predictions) respectively. Also, this blog page, made by the user who uploaded the dataset, GeorgeMcIntire, talks about his own predictive model to this problem and his step-by-step to solve it.

III. Methodology

Data Preprocessing

After the histogram analysis, the first thing done with the data is the split between training and testing set. This is one indispensable procedure to prevent biased assumptions and

for a better generalization and confidence in the model, it's important to generate a representative testing set and put this part of the analysis until further evaluation.

In a first process during the projects coding there were doubts in determining which of the sets should each technique be applied, and the order of these. Should the feature scaling be done before or after feature relevance, are there interferences between these? Also, should these be applied in which sets, only the training or the full set? These problems were solved with a better study using the Nanodegree material and discussion forums, this book was also very helpful. Beside these, there were no major problems in implementing the projects code, the previous done projects were provided a good course of actions and code templates.

Train and Testing Data Split

This procedure first separates the features and target columns in the *X_all, y_all* sets, and prints some details to confirm the chosen data. Sklearn *train_test_split* is imported, this method can easily separate *X_all and y_all* in *X_test, X_train* and *y_test,y_train* using the user's defined number of instances (parameter) for the training (80%) and test set (20%). Other useful parameters defined are: *random_state,* this parameter makes the method picks randomly, for the state defined, the same samples for each set, this makes the results equal on the same code running more than once; *stratify* is another parameters, in this project '*stratify = y_all'* was used so that the sets result in balanced portions of the two classes of target labels (1 or 0) reducing the bias in the model. Some characteristics of the training and testing set were printed to confirm good split values.

Feature Scaling

Now that we split the dataset the following procedure is applied only to the training data and later to the test set. The *StandardScaler()* transformer is created and assigned to the *scaler* variable, it's fitted to the training data with the *.fit()* method, the data is transformed by the *.transform()* method and assigned to the *scaled_data* variable. A new histogram of the data is presented below, the notebook contains a new *.describe() of the data*:

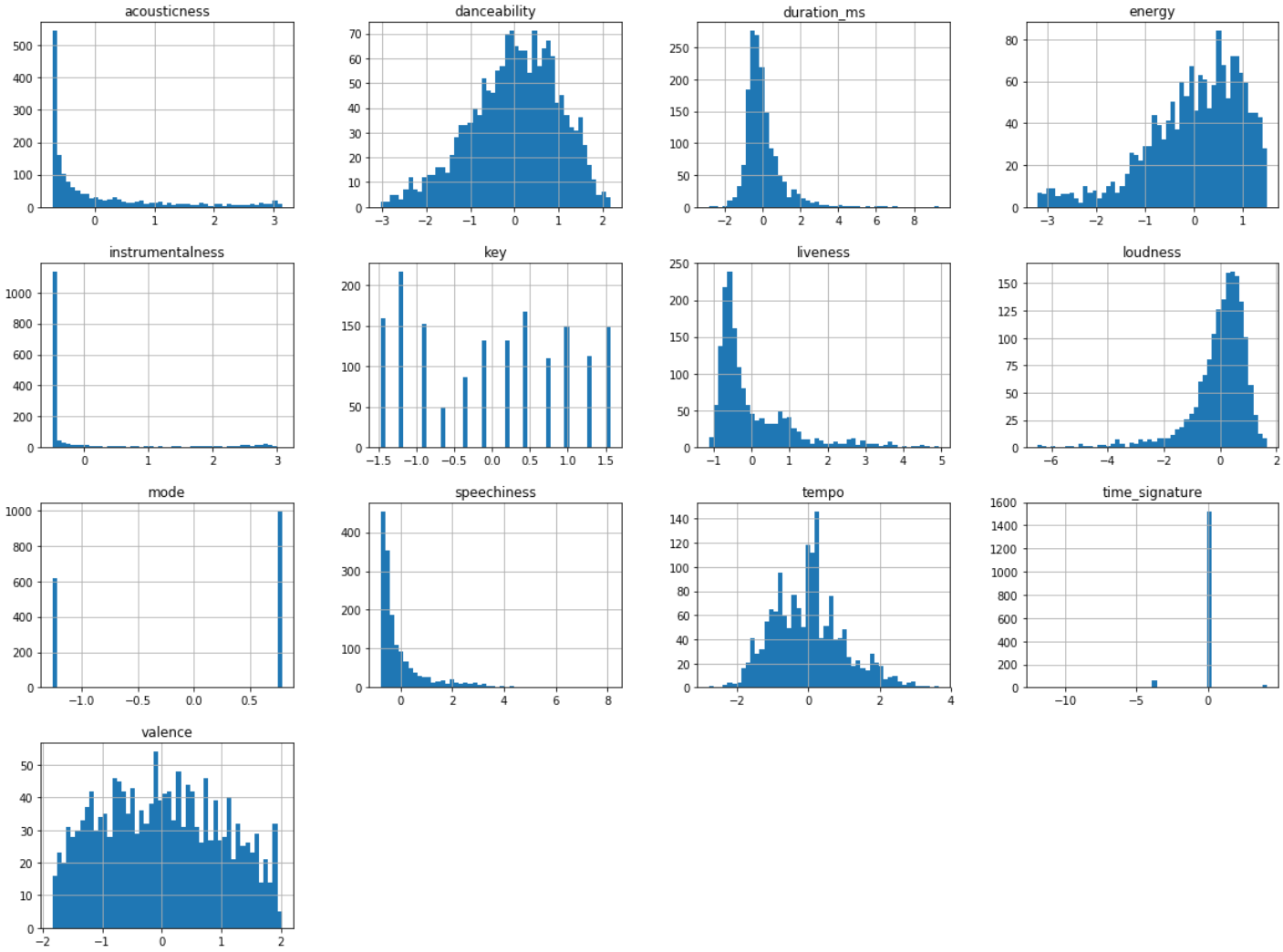

Image 3.1: Histogram of the scaled data, originated from the project's notebook.

The resulted distribution is far from Gaussian for some features, but the scaling brought the max/min values and outliers much closer from the distribution, this helps algorithms like SVM to produce better results.

Feature Relevance

The resulted values originated from the *.corr()* method for the 'target' label are documented below:

Feature	*Pearson's r*
target	1
speechiness	0.03686
acousticness	0.030874
duration_ms	0.017278
loudness	0.01479
valence	0.012553
instrumentalness	0.010685

Feature	*Pearson's r*
tempo	0.008538
danceability	0.005346
energy	0.003407
time_signature	-0.000392
liveness	-0.018815
key	-0.02679
mode	-0.05125

Table 3.1: Correlation between target label and features.

There are no major correlation between features and liking or not songs and this is one problem that led to the decision of having no feature selection made.

Training and evaluating models

In this section, a portion of *Student Intervention* project was used, the changes made were the scoring function from *f1_score* to *accuracy_score* and the training/prediction time print statements were removed. The following functions were used: '*train_classifier',* a function that receives a classifier (*clf*), training set (in this case the *scaled_data),* the target labels of these samples (*y_train)* and use the .fit() method of the classifier; *'predict_labels',* a function that receives a classifier(*clf)*, features(*scaled_data)* and target (*y_train),* uses the *.predict()* method to create another array *y_pred* with the predictions of the classifier and returns the accuracy of the model by comparing *y_pred* and *y_train* using the *accuracy_score*; '*train_predict'',* this function receives X_train(scaled_data), y_train, X_test, y_test and calls the two previous functions in order to fit, use the classifiers to predict the labels and return their scores. It prints the size of the set used to make predictions, training and testing score.

In the next cell, models were imported *SVC, LinearSVC, KNeighborsClassifier, AdaboostClassifier,* and *RandomForestClassifier*, assigned to *'clf_A'*, *'clf_B'*, *'clf_C'* ,*'clf_D'* respectively and added to '*classif*' list. The test set was scaled by the *scaler* transformer, previous fitted to the training set.

Two *for* loops were created, one in the *classif* list and other to vary the size of the training sets evaluated (500,1000, the total size of the training set) and each iteration trained a model with different training sets. The full scores are presented in the jupyter notebook, the best accuracy scores for each model are documented below:

Model	**Training Score (best)**	**Testing Score (best)**
Linear SVC	0.6677	0.6832

SVC	0.8022	0.7723
AdaBoost Classifier	0.9994	0.7748
KNeighbors Classifier	0.8115	0.6980

Table 3.2: Best accuracy scores for each model obtained in the training and evaluation section.

LinearSVC had worst scores in the training and testing set, this shows that the data isn't easily separable, by a linear decision boundary. KNeighborsClassifier had good scores in the training set, but not so much at testing, meaning it couldn't easily generalize the data.

The more prominent models, the ones with the highest accuracy using the entire training set, are C-Support Vector Machines, it's hoped that with model tuning these scores get raised and AdaBoostClassifier, it's clearly overfitting the training data. These models are going to be refined and commented on the next section.

A good observation is that all models scores scaled with the size of the training set thus if the dataset was larger, the performance of all models could be significantly better.

Refinement

Grid Search CV:

The refinement made in the models was an implementation of a grid search with cross-validation, to find better hyperparameters. At first, some general values were attributed to the grid search and later the search was narrowed.

Below the results of the SVC GridSearchCV():

C	Gamma	Kernel	Training Score	Testing Score

1	0.1	‘rbf’	0.7942	0.7351
1.5	0.1	‘rbf’	0.8419	0.7748
1.7	0.1	‘rbf’	0.8469	0.7797
1.9	0.1	‘rbf’	0.8500	0.7748

Table 3.3: Best combination of hyperparameters for each dictionary used in the Grid SearchCV.

These table rows were obtained with the following dictionaries of hyperparameters:

ROW 1: {'C':[0.1,1,10,100],'kernel':['rbf','poly’, ‘sigmoid’],'gamma': [0.001, 0.01, 0.1,1,10]}
These are general values used to evaluate a basic grid search.

ROW 2: { 'C':[0.5, 1, 1.5], 'kernel':['rbf', ’poly’, ‘sigmoid’], 'gamma': [0.05 ,0.1, 0.5]}
In the model evaluation section, the ‘gamma’ parameter was the default ‘auto’, this value is equal to 1/number features, this brought the idea to use near values in the dictionaries list. C was narrowed to near 0.1.

ROW 3: {'C':[1.5, 1.6, 1.7], 'kernel': ['rbf'], 'gamma': [0.1] }
Narrowing more the search resulted that the gamma optimal value for testing was 0.1, the C parameter will be raised one more iteration.

ROW 4: {'C':[1.7, 1.8, 1.9], 'kernel':['rbf'], 'gamma': [0.9, 0.1] }
Raising C didn’t raise the testing score, the final hyper-parameter combination is the one generated by row 3.

For the AdaBoostClassifier the following values were obtained:

n_estimators	learning_rate	base_estimator	Training Score	Testing Score
30	1	RandomForestClassifier	0.9994	0.7475
30	0.5	RandomForestClassifier	0.8134	0.7475
30	1	RandomForestClassifier	0.9169	0.7723
50	0.7	RandomForestClassifier	0.9033	0.7624

Table 3.4: Best combination of hyperparameters for each dictionary used in the Grid SearchCV.

ROW 1: { 'base_estimator': [RandomForestClassifier( ), DecisionTreeClassifier()], 'n_estimators': [30,50,70], 'learning_rate': [0.5,1,1.5] }

These are general values used to evaluate a basic grid search. The models continue to overfit, a '*max_depth'* parameter will be defined in the next grid search.

ROW 2: { 'base_estimator' : [RandomForestClassifier(max_depth = 2), DecisionTreeClassifier()], 'n_estimators': [30,50,70], 'learning_rate': [0.5,1,1.5] }

The max_depth helped lowering the overfitting, still, there is the need to find a better combination.

ROW 3: { 'base_estimator' : [RandomForestClassifier(max_depth = 3), DecisionTreeClassifier()], 'n_estimators': [30,50,70], 'learning_rate': [0.5,1,1.5] }

The overfitting is slightly worse, but a better result at testing was obtained

ROW 4: { 'base_estimator' : [RandomForestClassifier(max_depth = 3), DecisionTreeClassifier()], 'n_estimators': [25,30], 'learning_rate': [0.7,1,1.3] }

There is a big oscillation between overfitting and good testing score, the model that will be chosen to further evaluation is the row 3 model.

Confusion Matrix:

To decide between these two models, we will analyze their precision/recall tradeoff, since both had near accuracy score, the Sklearn '*confusion_matrix'*, *precision* and *recall* measure of each model in the testing set can help this analysis:

SVM	Negatives	Positives
Songs Not Liked	161	39
Songs Liked	47	157

$$Precision = \frac{161}{161 + 39} = 0.805$$

$$Recall = \frac{157}{157 + 47} = 0.76961$$

AdaBoostClassifier	Negatives	Positives
Songs Not Liked	155	45
Songs Liked	47	157

$$Precision = \frac{155}{155 + 45} = 0.775$$

$$Recall = \frac{157}{157 + 47} = 0.76961$$

The SVM model has better precision will be chosen as the definitive model. It also had less gap between the learning curves, it's generalizing better the data.

IV. Results

Model Evaluation and Validation

The final model proposed in this project is a C-Support Vector Machine that performed with approximately 80% of accuracy and precision in the testing set. The parameters of this model were grid searched in 4 iterations to provide a nice fit between the training and testing set, avoiding at maximum over/underfitting the data and including an '*rbf*' kernel with a '*C*' parameter of 1.7 and '*gamma*' parameter of 0.1. This was first chosen between four types of algorithms, compared by accuracy and not tuned. At last, compared with another that performed closely in the first selection, a final selection was made by doing a grid search cross-validation with both algorithms, this model had higher precision and testing accuracy, which made him the best to fit this problem.

A final analysis is made with the ROC Curve of the two models (Image 4.1) that were lastly compared and a K-Fold Cross Validation using the SVM Classifier:

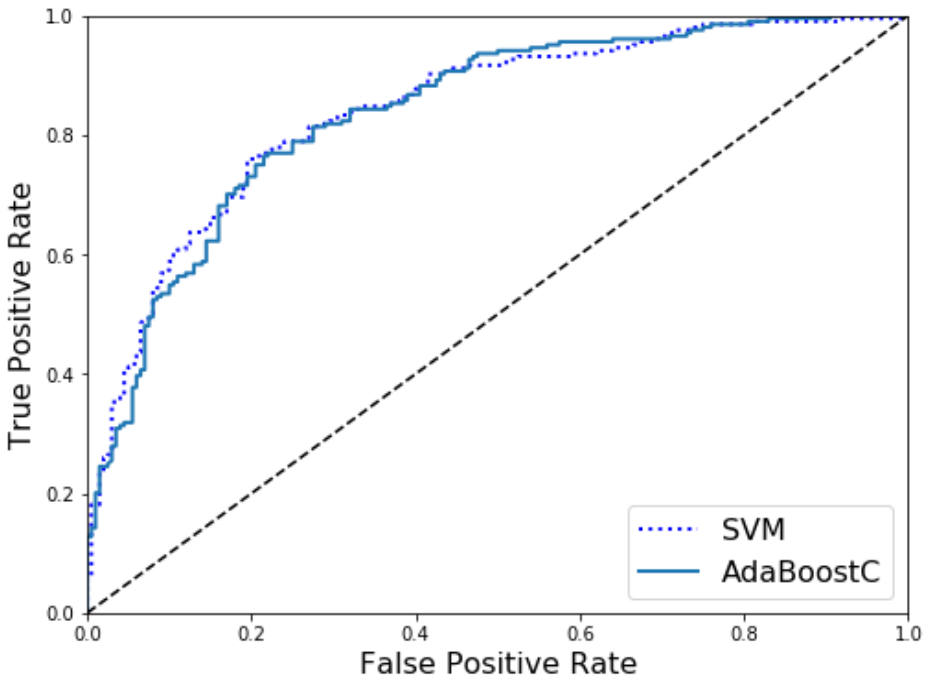

Image 4.1: ROC Curves of SVM and AdaBoostClassifier. Taken from the jupyter notebook of the project.

It's possible to see that, although these curves look very similar, the SVM is much smoother, with fewer oscillations, meaning a better generalization of the data and it's closer to the top left corner, having a better precision-recall tradeoff, an important matter in this project.

A K-Fold Cross validation was made by creating a new scaled set (by using the same *scaler* transformer) with all the data and splitting it in six subsets of training and testing data, twelve subsets at a total. At each iteration the classifier was fitted to a new training set and used to make predictions at another new testing set, the accuracy score of the models in each cycle are documented below:

Number of the K-Fold:	**Accuracy Score**
1	0.7388724035608308
2	0.7589285714285714
3	0.7410714285714286

4	0.7380952380952381
5	0.7916666666666666
6	0.7619047619047619

The accuracy scores are a bit lower (number 5 is greater) than the final model's, this oscillation is expected due to the different preparation of each sets. Still these are acceptable and reasonable values that are in agreement with the expected performance.

Justification

This problem raised nice results and conclusions about the problem domain and data set, had higher scores than the benchmark projects proposed, this submit achieved 0.7295 of maximum accuracy, while the final model achieved 0.7797, even in comparison with the user that uploaded the dataset, in his report at the previous mentioned blog page, he achieved around 70% of accuracy and precision while this project, with higher accuracy, resulted in 80% of precision.

Besides the small quantity of data, the model generalized a fair amount of information from the user and makes good predictions.

V. Conclusion

Free-Form Visualization

Below, in Image 5, a plot of the precision-recall tradeoff with different thresholds:

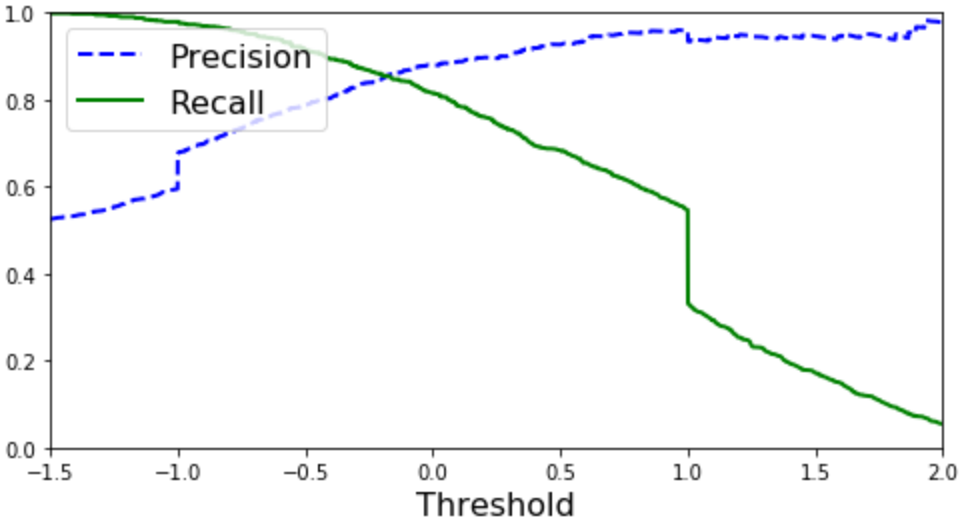

Image 5.1: Precision, Recall X Threshold from the definitive model of the project.

The last characteristic that is made concerning this model is that, if we manipulate the support vectors that make the decision boundaries (thresholds), it's possible to find a better combination that brings a higher precision at some lower recall, this is interesting because, if this model was used with a higher number of instances, it's preferred that more correct songs were labeled as right (higher precision) by losing some that the algorithm stays in doubt (lower recall).

Reflection

The project consisted in finding a model that best predicts music for a determined user (the one that shared the dataset), with a basic exploratory analysis of the dataset, looking at some major statistical details and feature values distributions, it was possible to determine how the pre-processing could be done, splitting the data, categorical features dropped, numerical features scaled and relevance analyzed. Some examples of basic algorithms were fitted to the training data and evaluated by accuracy. At last the two best models were tuned with a grid search cross-validation and evaluated by precision and recall scores.

It's important to look at this type of problem with a different perspective, a bigger picture concerning the dataset. The data obtained from only one user is very small for a good generalization of many users, while this model can predict with 78% of accuracy music for one in particular, it doesn't have the generalization power that a huge dataset of many users would require, the structure would have to be changed in order to achieve a higher complexity model. An interesting aspect of the research for making this was the discovery that the song attributes being generated by *convolutional neural networks*.

Improvement

The project was superficial, as commented in the previous section, has space for a lot of improvements, one is the better understanding of how the features (song attributes) are generated and propose a new processing of it that could benefit the procedure of the machine learning algorithms, such as how the data is outputted by the neural network. A deeper evaluation on the model tuning can be done, by studying more of each model and changing more parameters.

Another improvement can be included in the categorical features evaluation, the features pre-processing methods had no efficient ways of handling it, but it's possible to use *embeddings* if neural networks were more studied. This could provide better features for a model, probably providing better results.

At last, and already discussed in this project, the dataset does not favor a general user model, that could deliver good results of predictions for more than the specific one addressed here, so there is plenty of space for a deeper research and creation of an improved model.