UNIVERSITÀ di VERONA

Computer Science Department

Report Machine Learning Project

Authors: Alessandro Lovo, Alessandro Pedron
GitHub - Repository

Abstract

We started from the Billboard Hot 100 dataset which takes into consideration attributes of the songs that are present in the chart. The songs and attributes detected come from Spotify.

The goal of our project is to develop a recommendation system using a Neural Network. Additionally, we will employ other models to preprocess the data before applying the Neural Network and other algorithms. This approach will help us gain a deeper understanding of the results obtained.

1. DATA - Billboard Hot 100 dataset

We started from the Billboard Hot 100 dataset which takes into consideration attributes of the songs that are present in the chart. For a better description of the features: Kaggle - Dataset: Billboard Hot weekly charts.

To build our recommendation system, we made several modifications to our dataframe, the most significant being the creation of the 'maingenre' column.

In our dataset, the 'genre' column contains a list of genres and subgenres for each song. Our goal was to identify a single representative genre for each song.

To achieve this, we created a dictionary where the keys represent the main genres, and the values indicate how many times each genre appears in the original list of genres for a given song. The genre with the highest count is assigned as the 'maingenre' of the song.

Finally, since only 17 main genres have more than 100 songs, we decided to work with a smaller version of our dataframe that includes only these 17 main genres. Additionally, we perform further operations to clean the dataset and prepare it to achieve the goals outlined in the abstract.

2. KMeans

Another modification to our initial dataset addresses the **absence of user identifiers**. This is a critical issue since we are building a recommendation system that relies on user feedback to generate accurate suggestions.

Although we have song ratings, we do not have explicit user identifiers associated with them. To overcome this limitation, **we simulated user profiles through cluster analysis**. Specifically, we applied the unsupervised machine learning algorithm **K-Means** to group similar songs into clusters, which we then interpreted as distinct user profiles.

2.1 KMeans - in brief

The algorithm works in the following steps:

- **Initialization**: Choose the number of clusters (K) and randomly select K initial centroids. We use the elbow method in order to choose the correct numbers of centroids.
- **Assignment Step**: Assign each data point to the nearest centroid based on a distance metric, usually Euclidean distance.
- **Update Step**: Recalculate the centroids by taking the mean of all data points assigned to each cluster.
- **Repeat**: Repeat the assignment and update steps until the centroids no longer change significantly, indicating convergence.

2.2 Application

We used only numerical variables, as K-Means does not perform well with categorical data. Additionally, before applying the algorithm, we standardized the features to ensure equal contribution to the clustering process.

Why did we choose this technique to generate our users?

By applying K-Means, we identified five distinct clusters of similar songs, which we interpreted as representing five different user profiles with unique musical preferences.

To assign scores to songs within each simulated user group, we leveraged the valence variable numerical value ranging from 0 to 1 that reflects the positivity associated with a song. We treated this variable as a proxy for user ratings, assuming it represents the score that a user within that cluster would assign to a song.

Note: The Silhouette Index yielded a moderate score of 0.49, indicating some degree of misclassification within our clusters.

However, we consider this result acceptable, as real-world users often have diverse musical tastes that are not strictly limited to a single genre. Having clusters that contain songs from different genres aligns with this natural variability in user preferences.

That said, this also suggests that the patterns we can extract from the data may not be very strong, as the cluster composition does not reveal clear or distinct trends in user preferences.

Please refer to the **CLUSTER.ipynb** file in our repository, specifically the **STUDY THE COMPOSITION OF THE DF FOR USER** section, to visualize the mean valence evaluation and the number of songs per genre for each cluster.

3. Neural Network

We currently have five datasets, each representing a different user. Specifically, each dataset contains the songs evaluated by the user along with their corresponding characteristics. This serves as our foundation for building the Neural Network.

The goal of our Neural Network is to predict the score that a user will assign to a new song based on their musical preferences, inferred from the songs within their cluster.

The variable we use as the target variable is **valence**, which serves as the output of our model.

Conversely, the features used to train our model are some of the numerical variables, which act as the input.

3.1 Content-based approach

Content-Based Filtering is a recommendation system approach that suggests items based on their content features (e.g., genre, description, attributes) and the user's past preferences.

We use this approach for the characteristics of our data set:

- **Source of recommendations**: based on item features.
- **Personalization**: focused on the individual user's preferences.

3.2 Fully Connected Neural Network - in brief

- **Input Layer**: Receives the raw data (features).
- **Hidden Layers**: One or more layers where each neuron takes a weighted sum of inputs from the previous layer, adds a bias, and applies an activation function (like ReLU or sigmoid) to introduce non-linearity.
- **Output Layer**: Produces the final prediction. The number of neurons here depends on the task (e.g., one neuron for regression, multiple for classification).
- **Forward Propagation**: Data flows from the input layer through the hidden layers to the output layer, generating a prediction.
- **Loss Calculation**: The prediction is compared to the actual value using a loss function (e.g., MSE for regression).
- **Backpropagation**: The network adjusts the weights and biases based on the error, using an optimization algorithm like gradient descent to minimize the loss.
- **Iteration**: This process repeats over many epochs until the model learns to make accurate predictions.

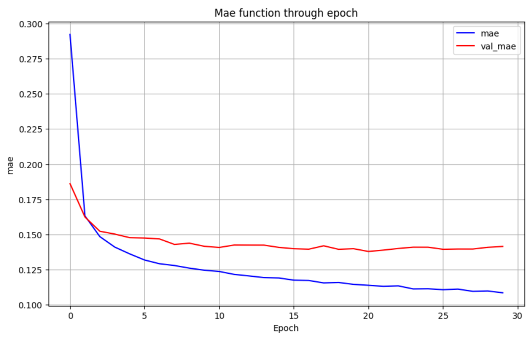
3.3 NN 0

The first NN that we build had this structure:

- **FIRST LAYER:** 64 neurons
- **SECOND LAYER:** 32 neurons
- **LAST LAYER:** 1 neuron, since is the output one.

From the application of this type of neural network, we can observe that learning plateaus after the fifth epoch. This is evident from the loss graph, where the validation MAE remains relatively constant beyond epoch five.

Here an example of the graphs that we obtain:



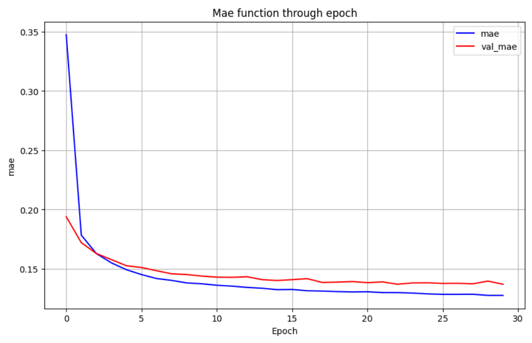
From this beta version of our neural network, we can observe that the model seems to learn too quickly. This is a plausible scenario, given that we are using 14 features while the first layer of our network contains 64 neurons. This imbalance could explain why the loss remains constant over the five epochs. Based on this observation, we will attempt to address the issue to verify if our analysis is correct.

3.4 NN 1

In this second model, we address the issues highlighted in the previous section by reducing the number of neurons. We have designed a neural network with the following characteristics:

- **FIRST LAYER:** 32 neurons
- **SECOND LAYER:** 16 neurons
- **LAST LAYER:** 1 neuron, since is the output one.

We want to immediately show how the MAE function changes as we decrease the number of neurons. It is evident that learning continues throughout the epochs, as the function keeps decreasing.



To improve our neural network, we also attempted to add additional layers to make it deeper. However, this did not lead to significant improvements. What seems to be more effective is the use of dropout neurons, which help mitigate overfitting by deactivating certain neurons that learn more than others. This, in turn, enhances the generalization of the results.

3.5 Cross Validation and One - Hot Encoding

In order to improve our results, We tested different techniques, such as:

- **Cross-Validation:** Method for assessing the performance of a machine learning model with greater reliability. It works by dividing the dataset into several folds (or partitions), training the model on one subset of the data, and evaluating it on a different subset. This process is repeated multiple times, with a different portion of the data used for testing in each iteration. In particular, we used the **k-fold cross validation**, where we divide the dataset in k folds with the same size.

  Main advantages: More robust evaluation, better generalization and maximizing the data usage.

- **One Hot Encoding** is a technique used to convert categorical variables into a numerical format that machine learning models can process. We apply it to the **maingenre** variable to leverage the information from songs that belong to the same genre and share similar characteristics.

  Instead of having a categorical variable, we obtain a set of binary variables, where a value of 1 indicates the genre a song belongs to, and 0 otherwise.

In the end, **we combined these two techniques** to achieve more robust results while also leveraging the information provided by the maingenre variable.

3.6 RESULTS NN (Using One Hot and Cross Validation)

In this final section on the neural network, we aim to highlight some key results.

First, we decided to use 100 epochs, as this setting allowed us to achieve a mean absolute error (MAE) that does not exceed 0.138. This choice was made after conducting several tests to optimize the results.

```
{'User0': {'Mean MAE': 0.1376309871673584},
 'User1': {'Mean MAE': 0.1329177975654602},
 'User2': {'Mean MAE': 0.09580752849578858},
 'User3': {'Mean MAE': 0.1331585168838501},
 'User4': {'Mean MAE': 0.1377421259880066}}
```

Are these good results for a recommender system?

A "good" MAE value depends on the context, but in general:

- MAE $< 0.05 \rightarrow$ Excellent model (average error of 5%)
- MAE between 0.05 and 0.1 $\rightarrow$ Good model.
- MAE between 0.1 and 0.2 $\rightarrow$ Acceptable but improvable.
- MAE $> 0.2 \rightarrow$ Low-precision model.

Our MAE is around 0.1, it means that, on average, the model is off by 10 percentage points on the 0 - 1 rating scale, which might be acceptable in our scenarios.

Why is this margin of error acceptable in our scenario?

We must take into account the nature of the data on which our model is trained. Since we identified users using an unsupervised algorithm (K-means) with a silhouette score of 0.49, as previously explained, this indicates low homogeneity within the clusters.

As a result, our neural network, trained on this data, is unlikely to capture strong representative relationships, given that the users, as they have been defined, do not exhibit clear musical preferences.

To achieve better results, we would need users who listen exclusively to specific genres. This would reduce the error by allowing the model to identify more precise patterns in the data.

4. Recommendation System

Now, we can finally evaluate our Recommendation system.

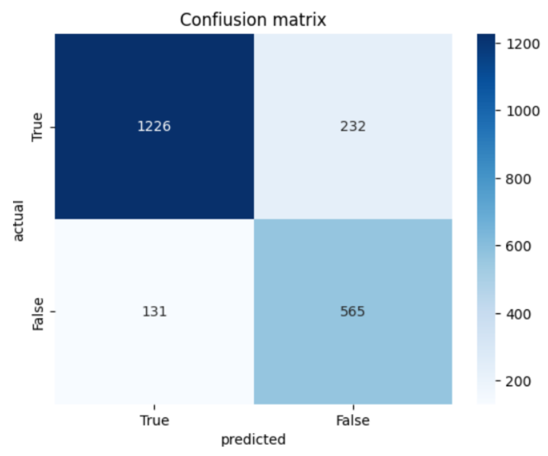
To do this, we constructed a dataframe called **df_prediction**, which contains **y_test** (the actual values of the valence variable used for testing) and **y_hat** (our predicted values corresponding to y_test).

We set a threshold of 0.7: if both y_test and y_hat exceed this threshold, the label is set to True; otherwise, it is set to False for values that do not meet the criterion.

This approach is useful for building our confusion matrix, allowing us to assess the number of incorrectly recommended songs for each user.

Below, **we provide an example using one-hot encoding without cross-validation, as it makes it easier to visualize the confusion matrix**:

This is the confusion matrix for **User2** (User with best result).



Here, we can see that there are 1226 true positives (TP), meaning the model correctly recommended songs that the user liked, and 565 true negatives (TN), meaning it correctly avoided recommending songs the user would not have liked. Overall, these are fairly good results.

Now, let's analyze the **misrecommendation**:

- False positives (FP) = 232 → The model recommended songs that the user did not like.
- False negatives (FN) = 131 → The model failed to recommend songs that the user would have liked.

Now we have also to evaluate the metrics related to this confusion matrix, in order to understand the results.

	Value
Accuracy	0.83
Precision	0.71
Recall	0.81
F1-score	0.76

- **Accuracy = 0.83** This means the model correctly classified 83% of the recommendations. This is a fairly good value, but accuracy alone is not always reliable, especially if the classes are imbalanced.
- **Precision = 0.71** Precision indicates the percentage of recommended songs that the user actually liked. A value of 0.71 means that 39% of the recommended songs were not liked (false positives - FP = 232). This suggests that the system might be making too many incorrect recommendations and could be improved by reducing false positives.
- **Recall = 0.81** A value of 0.81 indicates that 81% of the songs the user would have enjoyed were correctly recommended. This is a good result, but there are still false negatives (FN = 131), meaning the model failed to recommend some songs the user would have liked.
- **F1-score = 0.76** The F1-score is the harmonic mean between precision and recall, balancing both metrics. 0.76 indicates a good compromise between precision and recall, but it suggests that the system could still improve in selecting the songs to recommend (by increasing precision).

4.1 RESULTS - Recommendation System (Using One-Hot Encoding and Cross-Validation)

	User0	User1	User2	User3	User4
Accuracy	0.739273	0.804883	0.823721	0.822511	0.773801
Precision	0.728627	0.785661	0.749835	0.808589	0.774979
Recall	0.537818	0.676216	0.715534	0.339071	0.549971
F1-score	0.613859	0.726151	0.729466	0.475305	0.642376

The results show that the recommendation system has **good overall accuracy for all five users**, with values ranging between 0.73 and 0.82. However, the analysis of **precision, recall, and F1-score metrics highlights some significant variations**. In particular, User 3 has a very low recall (0.339), indicating that the system may not be effectively retrieving all relevant recommendations for this user. On the other hand, User 2 demonstrates the best overall performance, with an F1-score of 0.729, suggesting a good balance between precision and recall.

	Mean_Metrics
Accuracy	0.793
Precision	0.770
Recall	0.564
F1-score	0.637

The average metric values indicate that the recommendation system has good overall accuracy (0.793) and high precision (0.770), **suggesting that most recommended items are indeed relevant**. However, the **average recall value (0.564) is relatively low**, indicating that the **system may not be retrieving all relevant recommendations for users**. The average **F1-score (0.637)** confirms that there is room for improvement, particularly in balancing precision and recall to optimize the system's overall performance.

Once again, we can conclude that the main improvement, which also enhances our results, would be to identify more recognizable patterns in user preferences. This would lead to clearer and more accurate recommendations.

5. Support Vector Machine

The goal of applying SVM is to utilize a supervised machine learning algorithm to classify our data.

In particular, we are interested in classifying songs by their main genre.

5.1 SVM - in brief

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that best separates data points of different classes in a high-dimensional space. The goal is to maximize the margin between the classes, with the support vectors being the data points closest to the hyperplane. SVM is effective in handling both linear and non-linear data through the use of kernel functions.

5.2 SVM

Our initial approach focused on applying a basic SVM to classify songs based on their primary genre. However, we encountered some issues that need to be addressed. Before diving into these challenges, we would like to outline some of the results we obtained.

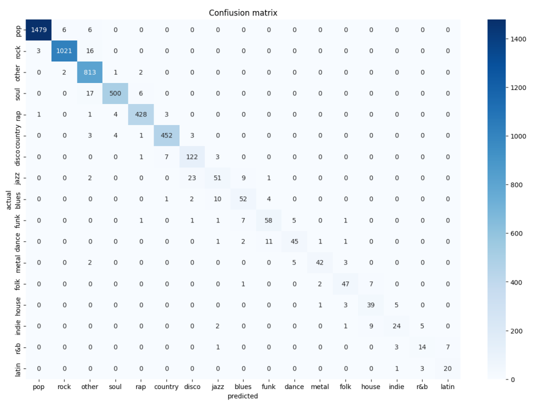
```
              precision    recall  f1-score   support

         pop       1.00      0.99      0.99      1491
        rock       0.99      0.98      0.99      1040
       other       0.95      0.99      0.97       818
        soul       0.98      0.96      0.97       523
         rap       0.97      0.98      0.98       437
     country       0.98      0.98      0.98       463
       disco       0.81      0.92      0.86       133
        jazz       0.74      0.59      0.66        86
       blues       0.73      0.75      0.74        69
        funk       0.78      0.78      0.78        74
       dance       0.82      0.74      0.78        61
       metal       0.91      0.81      0.86        52
        folk       0.84      0.82      0.83        57
       house       0.71      0.81      0.76        48
       indie       0.73      0.59      0.65        41
         r&b       0.64      0.56      0.60        25
       latin       0.74      0.83      0.78        24

    accuracy                           0.96      5442
   macro avg       0.84      0.83      0.83      5442
weighted avg       0.96      0.96      0.96      5442
```

We can immediately observe that genres with the highest number of songs are classified more accurately. This is due to the fact that a larger number of observations leads to better classification performance. Conversely, genres with fewer songs tend to have lower classification scores.

To gain a deeper understanding of the misclassified songs, we can perform an additional analysis using a **confusion matrix**.



actual \ predicted	pop	rock	other	soul	rap	country	disco	jazz	blues	funk	dance	metal	folk	house	indie	r&b	latin
pop	1479	6	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rock	3	1021	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
other	0	2	813	1	2	0	0	0	0	0	0	0	0	0	0	0	0
soul	0	0	17	500	6	0	0	0	0	0	0	0	0	0	0	0	0
rap	1	0	1	4	428	3	0	0	0	0	0	0	0	0	0	0	0
country	0	0	3	4	1	452	3	0	0	0	0	0	0	0	0	0	0
disco	0	0	0	0	1	7	122	3	0	0	0	0	0	0	0	0	0
jazz	0	0	2	0	0	0	23	51	9	1	0	0	0	0	0	0	0
blues	0	0	0	0	0	1	2	10	52	4	0	0	0	0	0	0	0
funk	0	0	0	0	1	0	1	1	7	58	5	0	1	0	0	0	0
dance	0	0	0	0	0	0	0	1	2	11	45	1	1	0	0	0	0
metal	0	0	2	0	0	0	0	0	0	0	5	42	3	0	0	0	0
folk	0	0	0	0	0	0	0	0	1	0	0	2	47	7	0	0	0
house	0	0	0	0	0	0	0	0	0	0	0	1	3	39	5	0	0
indie	0	0	0	0	0	0	0	2	0	0	0	0	1	9	24	5	0
r&b	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3	14	7
latin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	20

The confusion matrix confirms the observations highlighted above. Proportionally, the highest number of misclassified songs is associated with the genres that have the fewest samples.

To address this issue, we explored an alternative approach aimed at mitigating the problems related to the imbalance in sample size.

5.3 SMOTE - Synthetic Minority Oversampling Technique

It is a **regularization method** used to address class imbalance by generating synthetic samples for minority classes. It creates new data points between existing samples and their nearest neighbors, helping models like SVM improve classification performance by reducing bias toward majority classes.

By applying this method, we obtained the following results:

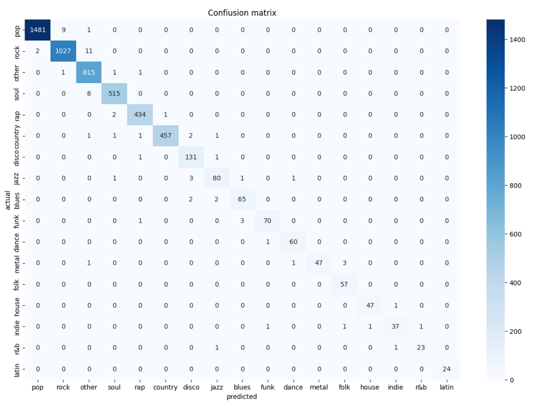
```
              precision    recall  f1-score   support

         pop       1.00      0.99      1.00      1491
        rock       0.99      0.99      0.99      1040
       other       0.97      1.00      0.98       818
        soul       0.99      0.98      0.99       523
         rap       0.99      0.99      0.99       437
     country       1.00      0.99      0.99       463
       disco       0.95      0.98      0.97       133
        jazz       0.94      0.93      0.94        86
       blues       0.94      0.94      0.94        69
        funk       0.97      0.95      0.96        74
       dance       0.97      0.98      0.98        61
       metal       1.00      0.90      0.95        52
        folk       0.93      1.00      0.97        57
       house       0.98      0.98      0.98        48
       indie       0.95      0.90      0.93        41
         r&b       0.96      0.92      0.94        25
       latin       1.00      1.00      1.00        24

    accuracy                           0.99      5442
   macro avg       0.97      0.97      0.97      5442
weighted avg       0.99      0.99      0.99      5442
```

We can immediately observe an improvement in our results after applying this technique, particularly for the main genres with a smaller number of songs.

Next, we will revisit the confusion matrix to further analyze the performance:



actual \ predicted	pop	rock	other	soul	rap	country	disco	jazz	blues	funk	dance	metal	folk	house	indie	r&b	latin
pop	1481	9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rock	2	1027	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
other	0	1	815	1	1	0	0	0	0	0	0	0	0	0	0	0	0
soul	0	0	8	515	0	0	0	0	0	0	0	0	0	0	0	0	0
rap	0	0	0	2	434	1	0	0	0	0	0	0	0	0	0	0	0
country	0	0	1	1	1	457	2	1	0	0	0	0	0	0	0	0	0
disco	0	0	0	0	1	0	131	1	0	0	0	0	0	0	0	0	0
jazz	0	0	0	1	0	0	3	80	1	0	1	0	0	0	0	0	0
blues	0	0	0	0	0	0	2	2	65	0	0	0	0	0	0	0	0
funk	0	0	0	0	1	0	0	0	3	70	0	0	0	0	0	0	0
dance	0	0	0	0	0	0	0	0	0	1	60	0	0	0	0	0	0
metal	0	0	1	0	0	0	0	0	0	0	1	47	3	0	0	0	0
folk	0	0	0	0	0	0	0	0	0	0	0	0	57	0	0	0	0
house	0	0	0	0	0	0	0	0	0	0	0	0	0	47	1	0	0
indie	0	0	0	0	0	0	0	0	0	1	0	0	1	1	37	1	0
r&b	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	23	0
latin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24

We can observe an almost perfect classification, with a very low number of misclassifications.

REFERENCES

- sklearn - KMeans
- Keras - The Sequential model.
- SVM
- Material from the course Machine Learning by Professor Francesco Setti
- Pattern Recognition and Machine Learning by Christopher M. Bishop
- Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville