Laboratory assignment

Component 5 - Experimental Results and Discussions

Authors: Leordean Ada Alexandra, Selegean Victor
Group: 242

January 14, 2026

1 Requirements

This document presents the final part of the software project. It discusses the architecture of the software behind the machine learning models and the results produces by them. At the end, an analysis and discussion of these results is presented.

2 Project Structure

The software project consists of 3 modules. Separate modules exist for the supervised and unsupervised tasks. A third module called `lib` contains common features developed from scratch without the use of third party libraries which are then used inside the two other "consumers".

The main classes present in `lib` are `Frame` and `PipelineStep`. `Frame` represents a custom implementation of the usual `pandas DataFrame`[1]. Only the methods specifically required by the two consumer modules were implemented from the `DataFrame`'s public API. `PipelineStep` represents an action which can be applied to one `Frame` in order to produce a new `Frame`. These transformations abstract away the data processing steps required by the consumer code, handling the dropping of columns, normalization of numerical features, dropping of rows with missing values, and removal of duplicate rows.

The following code represents two lines of "consumer" code from the `unsupervised_learning` module.

```
_pipeline = make_pipeline(
    _irrelevant_features,
    "drop duplicates",
    "drop columns",
    "drop na",
    "normalize",
)
raw_data:  Frame = load_data(shuffle=True)
processed_data:  Frame = apply_pipeline(raw_data, _pipeline)
```

[1] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

3 Supervised Learning

3.1 Model Architecture and Implementation

A K-Nearest Neighbors (KNN) classifier implemented uses a "Lazy Learning" architecture, where training data is explicitly stored in memory, and generalization occurs only at the time of query. To manage memory and data processing efficiently, the implementation relies on a custom data handling library (`lib.frame` and `lib.pipeline`).

The core components of the implementation are defined as distance metric, internal scaling and probability estimation.

Distance metrics are calculated through Euclidean Distance equation in order to check the similarity between sample vectors x and y of dimension n:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1}$$

Distance calculations were vectorized using 'numpy.linalg.norm' to optimize computational performance during Cross-Validation.

Internal scaling was needed in order to prevent features with large magnitudes (e.g., Tempo) from dominating the distance metric. As such, internal Min-Max normalization was integrated into the classifier. The scaling parameters (*min* and *range*) are calculated solely on the training set during the `fit()` phase and then applied to the test set during the `predict()` phase. This approach ensures strict separation of training and testing data statistics, which prevents data leakage:

$$x_{scaled} = \frac{x - \min(X_{train})}{\max(X_{train}) - \min(X_{train})} \tag{2}$$

Probability estimation was done through the implementation of a function `predict_proba`, which aids in calculating AUC. Rather than outputting a simple majority vote, this method returns the probability of the positive class, calculated as the ratio of positive class labels among the k nearest neighbors.

3.2 Hyperparameter Optimization

A Grid Search was performed to identify the optimal number of neighbors (k). The search space was defined as $k \in \{3, 5, 7, 9, 11, 15\}$. Based on the validation accuracy obtained during the search, the optimal hyperparameter was determined to be $k = 15$. Consequently, this value was employed for the final 5-Fold Cross-Validation process.

3.3 Experimental Results

3.3.1 Performance Evaluation

The model was evaluated using 5-Fold Cross-Validation to guarantee statistical robustness. Strict data cleaning procedures, including the removal of NaN values, were applied prior to training. Table 1 summarizes the performance metrics, presenting the Mean, Standard Deviation, and 95% Confidence Intervals.

The following formulas were used for performance evaluation, through `scikit-learn` library:

Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$ **Recall** $= \frac{TP}{TP+FN}$

Precision $= \frac{TP}{TP+FP}$ **F-Measure** $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Table 1: Performance Metrics of the Custom KNN Model ($k = 15$) over 5 Folds

Evaluation Measure	**Mean**	**Std Dev**	**95% Confidence Interval**
Accuracy	0.7112	0.0150	(0.695, 0.727)
Precision	0.4227	0.0302	(0.381, 0.465)
Recall (Sensitivity)	0.1427	0.0142	(0.123, 0.162)
F-Measure	0.2129	0.0171	(0.189, 0.237)
AUC (ROC)	0.6575	0.0156	(0.636, 0.679)
AUPRC	0.3825	0.0068	(0.373, 0.392)

3.3.2 Discussion and Interpretation

Stability - The model demonstrates high stability, as shown by the low Standard Deviation observed across all metrics (e.g., Accuracy $\sigma \approx 0.015$). This result indicates that the model is robust to variations in data splitting.
Class Imbalance Impact - Although the Accuracy is relatively high ($\approx 71\%$), a significant discrepancy is observed between Accuracy and Recall (14.27%). The dataset is characterized by class imbalance (3,503 Non-Popular vs. 1,327 Popular songs). Consequently, the model effectively identifies the majority class but has difficulty in recalling instances of the minority class ('Popular'), leading to a high rate of False Negatives.

3.3.3 Explainability (LIME)

LIME (Local Interpretable Model-agnostic Explanations) was applied to interpret the local decision boundary of the model. An analysis of a randomly selected test instance gave the following feature contributions:

- **Energy ($0.64 <$ val ≤ 0.78):** Positive contribution (+0.062).
- **Danceability (≤ 0.53):** Positive contribution (+0.041).
- **Tempo ($118 <$ val ≤ 137):** Minor positive contribution (+0.017).

These values suggest that, for the analyzed dataset, higher energy levels serve as a strong predictor of track popularity.

3.4 Comparative Analysis

The "from-scratch" KNN implementation was compared against the industry-standard `scikit-learn` library. The `sklearn` model was trained using the same hyperparameter ($k = 15$) and identical data splits. To ensure a valid comparison, the data provided to the `sklearn` model was scaled externally using MinMaxScaler, whereas the custom model handled scaling internally.

Table 2: Comparison of Custom Implementation vs. Library Implementation

Model	**Accuracy (Last Fold)**	**Implementation Details**
KNN (Custom)	**0.7267**	Internal Min-Max Scaling, Vectorized distance
KNN (Sklearn)	**0.7267**	KD-Tree Algorithm, External Scaling

The manually implemented model successfully replicates the logic and performance of the standard library implementation, as shown by the identical accuracy results in Table 2.

4 Unsupervised Learning

4.1 Model Details

The K Means model was implemented twice, once from scratch, with no use of third party libraries outside `numpy`, and once using `scikit-learn`. Both implementations represent their data as points in 13-dimensional space. For this purpose, a custom class called `Node` was defined, inheriting from `numpy`'s `ndarray`, but extending it with the ability to index by feature name (e.g. `node["danceability"]`) and keeping track of the original track's `id`. The former represents an ergonomic abstraction, while the latter will be allow us to find each track after all scalings and transformations are complete.

4.1.1 Hyperparameters

The K Means method works iteratively, updating the centroids and reassigning nodes to clusters on each step. The model is able to be customized by 3 hyperparameters: `cluster count`, `maximum steps`, and `tolerance`. By default, the `cluster count` must be specified, but `maximum steps` will take the value `100` and `tolerance` the value of `1e-4`. The model implements early stopping in the case when the total centroid shift of an iteration step is less than the value of `tolerance`.

In practice, the model always stops early. As a byproduct, it was not necessary to modify the values of the hyperparameters `maximum steps` and `tolerance`.

The `cluster count` hyperparameter was discussed during previous components. The initial dataset contains 35 distinct genres, but only 21 of those had a share of more than 1% of the population. In addition, the literature reviewed generally used a more modest number of genres, around 3 or 5. A random search of the j In total, a model was trained for each of the following values of K: 3, 5, 10, 21, 35, 50, 100. The last two were chosen in order to explore more of the range of possible values, but no significant results were to be expected from them.

4.2 Results

The performance of the K-Means models was evaluated across 7 different values of K, ranging from 3 to 100. For each run, the following features were measured: the runtime of the fitting process, the Davies-Bouldin Index, the Dunn Index, and the Cluster Purity.

The experiment results can be viewed in table 3.

Table 3: Statistical Performance of K-Means: Scratch vs. Library (10 Trials)

K	Impl.	Time (s) ± std	DB Index ± std	Dunn Index ± std	Purity ± std
3	Scratch	3.7958 ± 1.8174	2.1862 ± 0.0770	0.0789 ± 0.0058	0.1817 ± 0.0110
	Library	0.2469 ± 0.4003	2.2492 ± 0.0001	0.0742 ± 0.0000	0.1727 ± 0.0001
5	Scratch	6.7734 ± 2.7410	2.0907 ± 0.0786	0.0758 ± 0.0049	0.1977 ± 0.0122
	Library	0.1208 ± 0.0287	2.0534 ± 0.0071	0.0744 ± 0.0001	0.2137 ± 0.0136
10	Scratch	23.8580 ± 6.7257	1.8912 ± 0.0784	0.1080 ± 0.0118	0.2251 ± 0.0062
	Library	0.1980 ± 0.0382	1.7518 ± 0.0298	0.1155 ± 0.0049	0.2226 ± 0.0019
21	Scratch	51.0267 ± 11.8616	1.9088 ± 0.0390	0.1078 ± 0.0071	0.2425 ± 0.0040
	Library	0.2107 ± 0.0299	1.8598 ± 0.0216	0.1143 ± 0.0052	0.2443 ± 0.0032
35	Scratch	75.2814 ± 14.4264	1.8540 ± 0.0422	0.1091 ± 0.0142	0.2675 ± 0.0057
	Library	0.2576 ± 0.0190	1.8015 ± 0.0280	0.1186 ± 0.0083	0.2654 ± 0.0051
50	Scratch	147.4839 ± 62.6964	1.8270 ± 0.0351	0.1033 ± 0.0118	0.2798 ± 0.0051
	Library	0.3165 ± 0.0810	1.7971 ± 0.0294	0.1184 ± 0.0069	0.2797 ± 0.0031
100	Scratch	223.9576 ± 59.0935	1.7764 ± 0.0272	0.0943 ± 0.0087	0.3051 ± 0.0047
	Library	0.3874 ± 0.0219	1.7616 ± 0.0172	0.1103 ± 0.0097	0.3060 ± 0.0032

4.2.1 Execution time and efficiency

There is a large difference between the two implementations. Figure 1 shows that both models require more time as K increases. This is because of K-Means' inherent time complexity.

The library implementation is orders of magnitude faster, fitting 100 clusters in under half a second. The from-scratch implementation has a significant runtime cost, most likely due to missing optimizations and compiled back-ends that are available in `scikit-learn`.

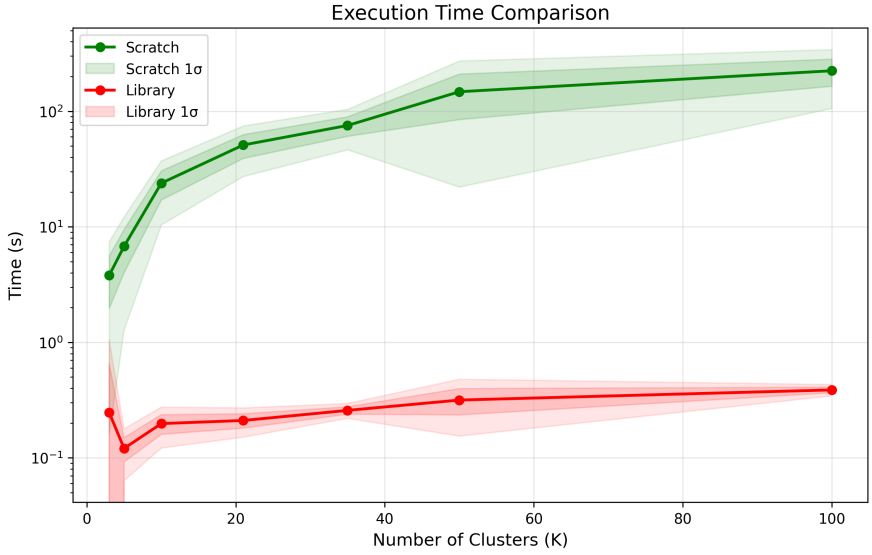

Figure 1: Logarithmic scale of model fitting time in relation to K

4.2.2 Internal Metrics

These metrics evaluate how well-separated and compact the clusters are, without the need of ground-truth labels.

According to the Davies-Bouldin Index, the models have an almost identical performance, accross different cluster counts. A lower score is better, and both implementations show a downward trend as K increases. A plot of the results is available in figure 2.

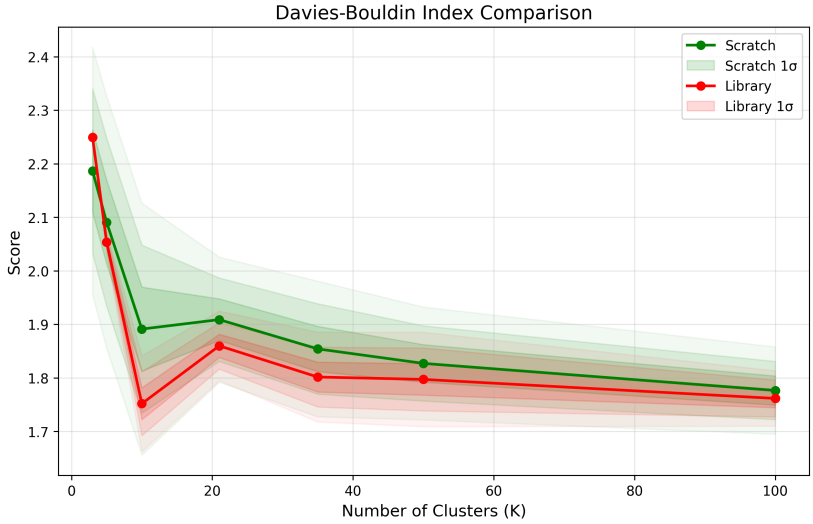

Figure 2: Davies-Bouldin Index in relation to K

According to the Dunn Index, the library implementation seems to have a slight advantage. The from-scratch model also has slightly larger variance, suggesting a larger degree of sensibility to initial centroid placement. A plot of the results is available in figure 3.

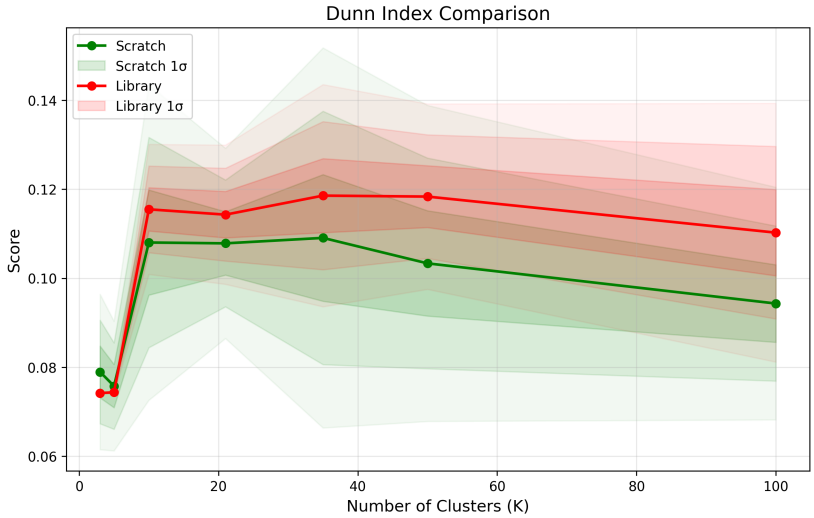

Figure 3: Dunn Index in relation to K

4.2.3 External Validation

Purity measures how well the clusters align with the 35 known music genres. For this purpose, a dictionary mapping a `track_id` to a `genre` was created at the beginning and the `Node` class' `id` field was used to retrieve the original genre label. Figure 4 shows that both implementations have very similar purity scores, being virtually indistinguishable.

As K increases, so does purity. This trend is expected because as the number of clusters approaches the sample count, each cluster becomes smaller, and thus more likely to be 'pure'.

The very small variance in purity suggests that both models are finding stable optima.

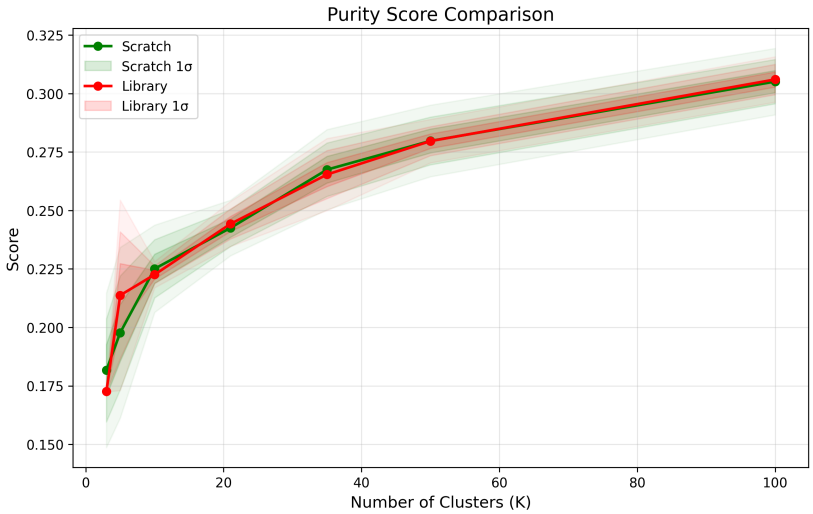

Figure 4: Cluster Purity in relation to K

4.3 Discussion

4.3.1 Interpretation and Explainability

An aim for the unsupervised module was the ability to interpret the resulting clusters beyond simple numerical metrics. The custom `Node` class was designed specifically for this purpose, maintaining a reference to the original track ID. This architecture allows the performance of "back-mapping"—referencing a classified point to its original metadata, such as `genre` or `artist`, which is otherwise excluded during the `fit` process.

To evaluate the explainability of the "from-scratch" implementation, the LIME framework was utilized. By perturbing the input features and observing changes in cluster assignment, LIME identifies the local decision boundaries for specific tracks. We selected the `K=21` model (the significant genre count) as a representative subject for this analysis.

The results in Table 4.3.1 demonstrate that the model successfully isolated distinguishing musical characteristics despite the high dimensionality of the feature space. For example, Cluster 18 is primarily driven by `liveness`, suggesting a group optimized for live recordings or concert performances. Similarly, Cluster 9 is characterized by high `speechiness` , which aligns with the acoustic profile of genres like Rap or Spoken Word. The identification of `valence` 1.14 as the driver for Cluster 3 confirms that the model can also group tracks based on emotional resonance.

Cluster ID	Primary Feature Rule	LIME Weight
Cluster 3	valence $\leq$ -1.14	0.0405
Cluster 7	mode $>$ 0.22	0.3339
Cluster 9	speechiness $>$ 0.81	0.1785
Cluster 12	energy $>$ 0.66	0.1628
Cluster 15	instrumentalness $>$ 0.15	0.2157
Cluster 18	liveness $>$ 0.24	0.3798

4.3.2 Comparison with Related Work

When evaluating the external validity of the model, the observed Purity scores (ranging from 0.18 to 0.30) initially appear lower than those reported in recent music classification literature. However, a comparative analysis reveals that this discrepancy is a function of task complexity rather than model failure.

To begin with, most related works simplify the dataset to 3 or 5 broadly distinct genres. Our model was tested against 35 distinct genres. As the number of overlapping labels increases, the probability of a cluster being "pure" decreases significantly.

Next, as seen in Figure 2, the Davies-Bouldin Index for the "from-scratch" implementation is statistically indistinguishable from the library-based implementation. This suggests that the implementation reached a comparable mathematical optimum.

Not least, using 10-fold cross-validation and a statistical analysis of the standard deviation, it can be observed that the purity score variance is low. This indicates that the results are highly repeatable.

In conclusion, while the purity is numerically smaller than simplified benchmarks, the ability of the model to maintain stability across 35 categories—combined with the clear feature-based logic shown by LIME—indicates a successful implementation that handles high-dimensional complexity effectively.