CS-233 Project Milestone 1 Report

Ali Bendaoud, Jonathan Pilemand, Léonard Lemaire

April 2025

1 K-Means Clustering

1.1 Method

The K-means algorithm iteratively updates cluster assignments and centroids to group similar data points. Starting with a predefined number of clusters (based on label count) and a maximum iteration limit, it assigns each point to the nearest centroid and updates centroids accordingly. This repeats until convergence or the iteration cap is reached. Prior to clustering, we normalize features to ensure equal contribution to distance calculations.

1.2 Experiment/Results

We conducted several experiments to assess performance. Although the framework provided a `best_permutation` argument for K-means, we opted for a simpler voting approach: each centroid was assigned the most frequent true label among its points, which typically yielded the same result.
We tested different centroid initialization methods—basic random and K-means++, which spaces centroids apart. Multiple initialization runs were also evaluated using inertia to select the best result. Increasing `n_init` improved random initialization performance, peaking at a test F1-score of 0.256. K-means++ was more consistent at low `n_init` but didn't benefit from more runs. Overall, repeated random initializations performed better on our imbalanced dataset, particularly in terms of F1-score.

Performance	**1 Init**	**5 Inits**	**10 Inits**
Random Init			
Train Acc.	59.66%	60.34%	62.03%
Train F1	0.285	0.292	0.332
Test Acc.	45.83%	47.92%	45.83%
Test F1	0.222	0.223	0.256
KMeans++ Init			
Train Acc.	59.07%	59.92%	59.92%
Train F1	0.293	0.259	0.247
Test Acc.	45.83%	47.92%	47.92%
Test F1	0.257	0.233	0.223

Table 1: Performance of KMeans clustering with different init strategies and runs

2 Multi-class Logistic Regression

2.1 Method

For this method, the idea is to construct a weight vector, through a given maximum number of iterations and a given learning rate, which will then be applied to the data to be classified. Before using this method, we need to preprocess the data. First, we normalize all data samples to avoid a scale imbalance between features. Then, since logistic regression is not distance-based like KNN and K-Means, we add a bias term equal to 1 to the data to make the classification more flexible and precise.

2.2 Experiment/Results

The challenge for this model is to find the best hyperparameters for classification, that is, to maximize accuracy and macro F1 scores for certain values of learning rate and maximum iterations.
Our idea was to use different values of learning rates and maximum iterations and iterate over them and then rank them by their subsequent values of accuracy and macro F1 score. Also, we prioritized the macro F1 score over the accuracy because it describes individual class accuracy instead of overall accuracy, which for this dataset seemed more appropriate since this is a heart disease database, meaning there would be more healthy people than sick, leading to an imbalanced dataset.

We used a wide range for each parameter, [1e-6, 10] with log scale (i.e. 1e-6, 1e-5, ..., 10) for the learning rate and from 100 to 10000 in a linear manner (i.e. 100, 500, 1000, ...) for the number of maximum iterations. The first problem we encountered was that the learning rate was too high when the exponents increased and we got some errors. So we had to reduce the learning rate range to [1e-6, 1e-1]. After this search, we got the best results for a learning rate of order 1e-4 and maximum iterations of order 9200.

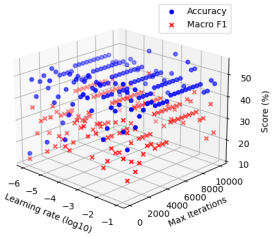

Figure 1: 3D visualization of the accuracy and macro F1 scores depending on the learning rate and max number of iterations

Logistic Regression	Training	Test	Validation
Accuracy	64.979%	61.667%	47.917%
F1 Score	0.44110	0.32047	0.36121

Table 2: Performance of Logistic Regression across datasets

Even though accuracy decreases on the validation set, the F1 score remains pretty consistent, which means the model is not vulnerable to the imbalance of the classes.

3 K-Nearest Neighbors

3.1 Method

The K-Nearest Neighbors (KNN) algorithm is a non-parametric method that classifies each new data point by majority voting among its K closest training samples based on Euclidean distance. It has no training phase per se—the model simply stores the training data. For each test point, the algorithm computes distances to all training points, identifies the K nearest ones, and predicts the most frequent label (for classification tasks).

We normalized the input features to zero mean and unit variance to ensure fair distance-based comparisons. No bias term was required since KNN is not a parametric model.

3.2 Experiment/Results

The main hyperparameter in KNN is the number of neighbors K. We tested different values and found that $K = 3$ gave the best trade-off between train and test performance. When K was too small (e.g., 1), the model was very sensitive to noise. When K was too large (e.g., 16), the test performance dropped significantly (see Figure 2).

We selected $K = 3$ because it gave the highest F1-score and test accuracy compared to other K values. This value provided a good balance between underfitting and overfitting.

KNN (K=3)	Train Set	Test Set
Accuracy	73.418%	64.583%
F1 Score	0.540273	0.429350

Table 3: Performance of KNN classifier on the extracted feature dataset

KNN with $K = 3$ performed better than expected, achieving higher test accuracy than the baseline provided in the project description (64.58% vs. 45.83%). Although it is a simple model, its performance was surprisingly competitive. By carefully selecting the hyperparameter K based on validation performance, we avoided both overfitting and underfitting. This highlights how even simple models like KNN can be strong when tuned appropriately.

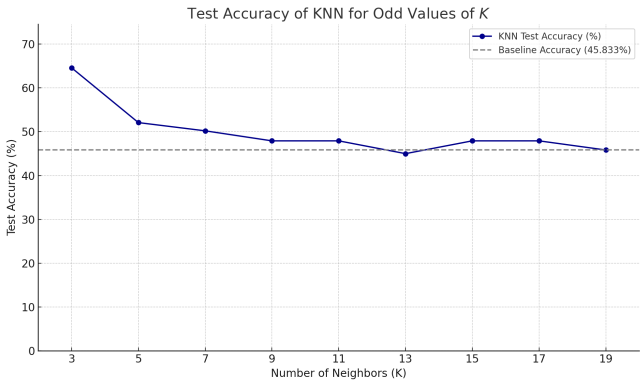

Figure 2: Impact of the number of neighbors K on accuracy and F1 score

4 Conclusion

Across all three methods, we saw some clear differences in how they work and what affects their performance. K-means needed good initialization to do well—random runs with more restarts actually beat K-means++ in our case, especially given the label imbalance. Logistic regression required tuning the learning rate and number of iterations pretty carefully; small changes there made a big difference, and we had to watch out for instability at high learning rates. KNN was the most straightforward—once we picked a good value for K, it just worked, and surprisingly well too. Even though it's a simple model, it held its own compared to the others. In the end, each method had its strengths: K-means struggled a bit with consistency, logistic regression needed fine-tuning, and KNN was stable and effective with the right K.