Resource consumption based usability study for the comparison between customer segmentation with KMeans clustering using TensorFlow and scikit-learn implementation

Imdi Melvana Mauladendra (HTW Berlin), Sebastian Rauch (HTW Berlin)

Abstract

This study compares the resource consumption and efficiency of KMeans clustering implementations in TensorFlow and scikit-learn for customer segmentation. Energy usage, computational performance, and memory consumption were measured using an automated testing setup. Results indicate that scikit-learn is more efficient due to automatic convergence control, while TensorFlow requires more computational resources. Additionally, unexpected high resource consumption was observed during idle states in the automation tool. The findings highlight the importance of energy-efficient machine learning implementations and provide insights for optimizing clustering algorithms.

1 Introduction

In the field of environmental informatics, efficient energy consumption and computational performance are key factors when implementing machine learning models for data-driven decision-making. [1] This study compares energy consumption and performance metrics during the execution of customer segmentation using the KMeans clustering algorithm, implemented separately in TensorFlow and scikit-learn.

The customer dataset includes attributes such as Customer ID, Gender, Age, Annual Income, and Spending Score, providing a basis for clustering consumers based on their purchasing behaviors. By evaluating the energy efficiency and computational demands of both implementations, this analysis aims to determine which library is preferable for performing customer segmentation using the KMeans algorithm.

2 Methology

2.1 Customer Segmentation Using KMeans Clustering

Customer segmentation is a fundamental technique in data analytics, allowing businesses to group customers based on shared characteristics and behaviors. The KMeans clustering algorithm is a widely used unsupervised learning method for this purpose. It partitions data points into K distinct clusters based on similarity, where each cluster is represented by a centroid. [2]

The clustering process involves the following steps:

1. Centroid Initialization: The algorithm selects K initial centroids, which serve as the center of the clusters.

2. Cluster Assignment: Each data point is assigned to the nearest centroid based on the Euclidean distance.
3. Centroid Update: The centroids are recalculated as the mean of all points in each cluster.
4. Iteration & Convergence: Steps 2 and 3 are repeated until the centroids stabilize or a predefined number of iterations is reached. [3]

KMeans is particularly useful for customer segmentation, as it helps identify groups with similar purchasing behaviors, allowing businesses to develop targeted marketing strategies. [4] In this project, customer segmentation is performed using the implementations of KMeans in scikit-learn and TensorFlow, and their computational performance and resource consumption is compared.

2.2 Implementation in scikit-learn

Scikit-learn provides an optimized and efficient implementation of KMeans, making it a preferred choice for standard machine learning tasks. The scikit-learn implementation of KMeans is highly optimized and uses k-means++ for improved centroid initialization. This reduces the likelihood of poor starting configurations and accelerates convergence. The algorithm performs up to a number of iterations but automatically stops when cluster centers no longer change significantly. As a result, the actual number of iterations varies depending on data distribution and convergence speed. The implementation in scikit-learn is particularly efficient as it leverages optimized algorithms such as elkan for distance calculations. [5] The clustering results were visualized using 2D and 3D scatter plots, where different feature combinations such as Age, Annual Income, and Spending Score were used for segmentation. In this project 100 Iterations and a tolerance of 0.0001 were chosen.

2.2 Implementation in TensorFlow

In contrast, the TensorFlow implementation requires a manual implementation of the KMeans algorithm. Centroid initialization is done randomly, which may lead to suboptimal starting points. Cluster assignment is performed by explicitly computing Euclidean distances between data points and centroids. The centroids are then updated manually by calculating the meaning of all assigned points. A key limitation of the current TensorFlow implementation is that it runs a fixed number of iterations, regardless of whether the algorithm has already converged. This can lead to unnecessary computational overhead or premature termination before an optimal cluster distribution is achieved. [6] This fundamental difference from the scikit-learn implementation, which uses an automatic convergence criterion, makes the TensorFlow approach potentially less efficient. In this project 100 Iterations were chosen.

2.3 Comparison of the clustering results

An example data set in csv format from kaggle was used for clustering. The data set had 200 entries. [7] The structure of the data can be seen in the following figure (**Figure 1**).

1	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
2	1	Male	19	15	39
3	2	Male	21	15	81
4	3	Female	20	16	6

Figure 1: Structure of used data records

The two clustering methods via TensorFlow and scikit-learn were each implemented with Python. As initial Source for the code a repository in Kaggle was used and adjusted for the needs of this project. [8] The code adapted and the database used are provided on the zenodo platform. [9]

After performing the clustering, the clusters produced almost the same results. The deviations are small and can be neglected. This can be seen in the following cluster graphics (**Figure 2-7**).

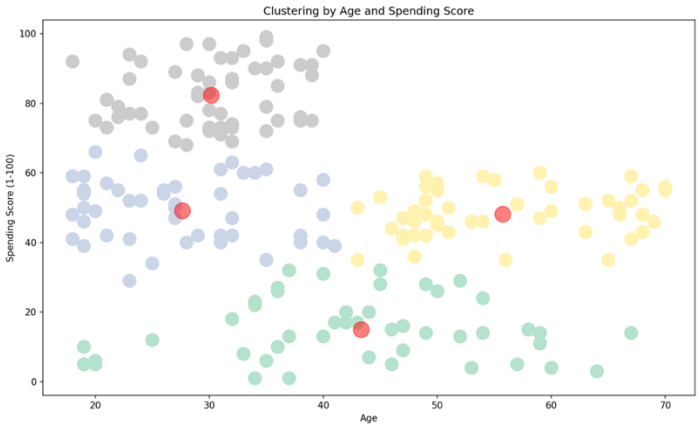

Figure 2: TensorFlow clustering age and spending score

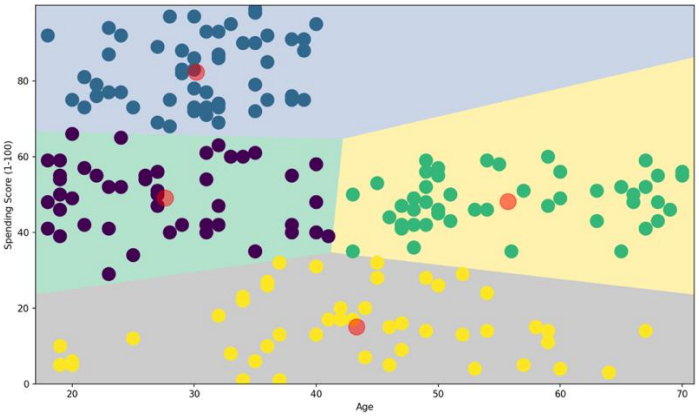

Figure 3: Scikit-learn clustering age and spending score

Figure 4: TensorFlow clustering annual income and spending score

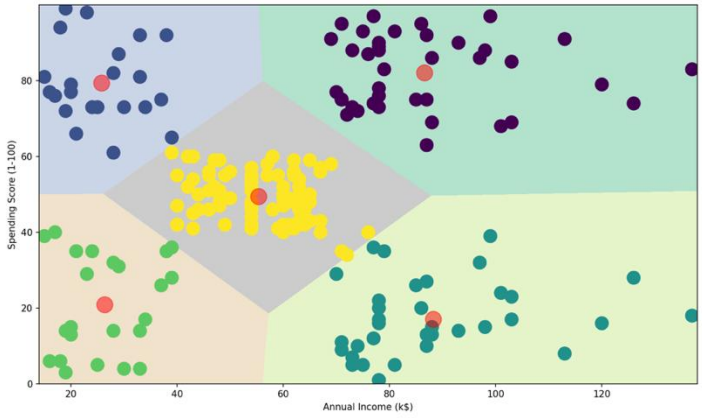

Figure 5: Scikit-learn clustering annual income and spending score

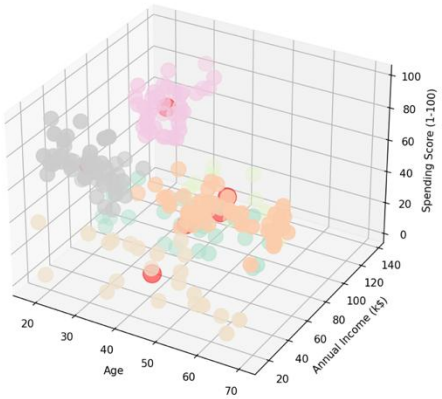

Figure 6: TensorFlow clustering age, annual income and spending score

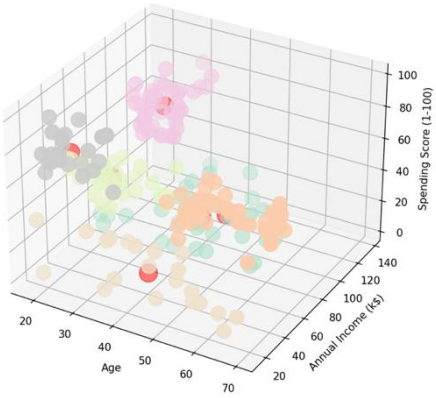

Figure 7: Scikit-learn clustering age, annual income and spending score

3. Test setup

3.1 Measurement setup

To accurately evaluate the energy consumption and hardware utilization of the software, a structured measurement setup was established, building upon prior methodologies from research at HTW Berlin. As outlined by Junger et al., several key preparations are necessary to ensure reliable energy consumption testing.

The setup consists of four main components. First, the System Under Test (SUT) is a Windows-based machine configured to run scikit-learn and TensorFlow scripts. Second, the Load Driver (LT), using Microsoft Power Automate, automates workloads to guarantee consistent testing conditions. Third, the Gude Power Control 1202 power strip logs energy consumption at one-second intervals, following the "Blauer Engel" guidelines. [10] Finally, performance metrics such as CPU load and memory usage are collected and analyzed using OSCAR, generating detailed reports on energy consumption.

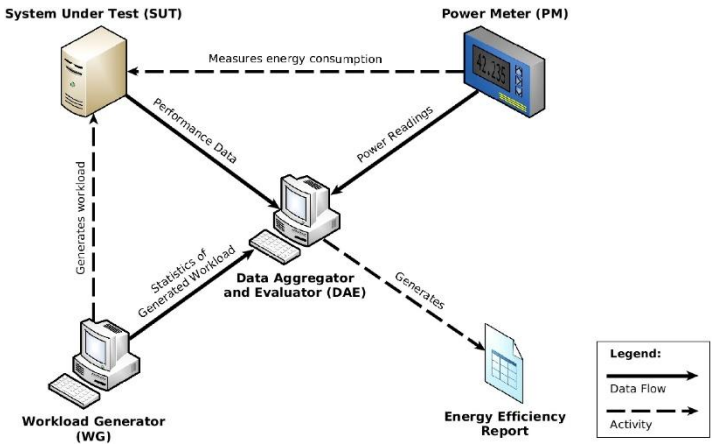

Figure 8: Setup for measuring the energy consumption of software [11]

The testing process was conducted over two days. On the first day, scikit-learn and TensorFlow were tested under load. On the second day, baseline measurements were taken

without running any software, providing a reference point for comparing the energy efficiency of both frameworks.

3.2 Power Automate Simulation

This section explains the automation process in Power Automate. The simulation follows a structured workflow to automate repetitive tasks, improving efficiency and reliability with minimal human input. Power Automate is part of the recommended standard usage scenario for scenario tests and long-term tests as outlined by the Blauer Engel certification. [12] The diagram below illustrates the sequence of steps involved in the automation process. These steps include executing a CMD session, running a Python script, capturing the output, implementing waiting periods, and closing the windows.

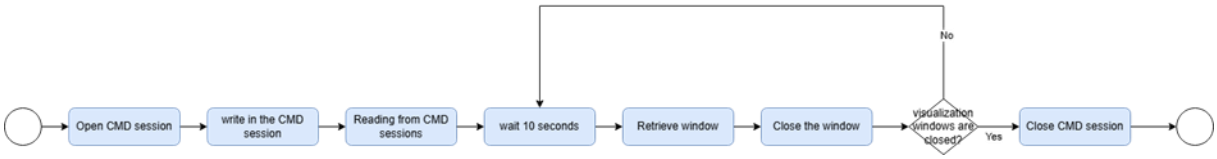

Figure 9: Usage Scenario

The automation is organized into five subflows:

1. Aktions_Template_1: Executes the predefined scenario.
2. Main: Manages the test and logs each run.
3. Start_Log_Item: Logs the start time of each test cycle.
4. Stop_Log_Item: Records the end time and waits for 30 seconds before the next iteration.
5. Create_Path_File_Time: Stores data like timestamps, test names, and necessary commands.

Once the subflows are set up, the measurement process can be started in Power Automate.

3.3 Measurement Procedure and Implementation

The automation software (**Chapter 3.2**) plays a key role in measuring and comparing resource and energy consumption. It synchronizes the system time, logs relevant activities on the SUT, and automates user inputs for Python scripts that cannot be measured manually. The measurement follows a structured approach to ensure reliability and repeatability.

The measurement setup defined in the Main sub flow of Power Automate includes a logging resolution of 0.1 seconds and a test duration of 120 seconds per run, covering scenarios such as baseline (pre-installation of TensorFlow and scikit-learn), idle state, matrix computations, and statistical analyses. Each test run is repeated 30 times to calculate statistical averages, ensuring robust and reliable results.

Following the measurement setup outlined in **Chapter 3.1**, the measurement procedure is carefully executed to ensure the accuracy and consistency of the collected data. The measurement procedure consists of seven steps:

1. Start energy measurement on the energy data aggregator.

2. Begin performance data collection using the Windows monitoring tool on the SUT.
3. Open Power Automate, execute the corresponding automation (TensorFlow or Scikit-learn), and then close Power Automate to avoid interference.
4. Upon completion of 30 repetitions, a pop-up window signals the end of the measurement.
5. Stop energy measurement on the energy data aggregator.
6. Save PC performance logs and action logs, and export energy measurement data to an external storage device.

If technical issues occur during the process, the measurement must be restarted. After a successful measurement session, the collected data is analyzed using OSCAR, as described in Chapter 3.4.

3.4 Report Generation and Analysis with Oscar

OSCAR, an open-source tool developed at the Umwelt-Campus Birkenfeld, is designed for analyzing and reporting the environmental impact of software usage. [13] The analysis process involves uploading three key data sets from the measurement:

1. Actions: Log files from the SUT, capturing timestamps of test executions and actions performed.
2. Energy Consumption: Data from the power meter, showing average, minimum, and maximum energy usage synchronized with logged actions.
3. Hardware Utilization: Timestamped data on CPU usage, RAM load, disk activity, and network traffic.

The measurement data, which includes readings from idle (baseline), TensorFlow, and scikit-learn scenarios, is uploaded separately as CSV files into OSCAR. Each data set is processed individually, allowing for a direct comparison later. Minor adjustments, such as encoding corrections or timestamp edits, are made before generating the final report. As outlined in the instructions in OSCAR, these adjustments ensure that the data is processed accurately and consistently. [14]

The final report includes a detailed analysis of energy consumption, resource usage, and graphical representations of processor load, memory usage, and data transfer.

4 Results

The results of the testing (**Figure 10**) show a comparison of energy consumption and resource usage between Idle, TensorFlow, and Scikit-Learn conditions. In the Idle state, electricity consumption was recorded at 1.02 Wh, with CPU usage at 6.89%, RAM usage at 34%, network data transfer at 2.26 Mbyte, and permanent memory usage reaching 194.6 Mbyte. When using TensorFlow, electricity consumption dropped to 0.89 Wh, with CPU usage at 5.59%, RAM usage at 45%, network data transfer at 1.08 Mbyte, and permanent memory usage of 142.85 Mbyte. Scikit-Learn demonstrated the highest efficiency, with electricity consumption of 0.88 Wh, CPU usage of 5.78%, RAM usage of 42%, network data transfer at 0.96 Mbyte, and the lowest permanent memory usage at 127.75 Mbyte. Overall, Scikit-Learn proved to be more energy- and memory-efficient compared to TensorFlow, while the Idle condition surprisingly showed higher resource consumption than both frameworks during active use. The reasons why the idle state consumes more resources will be discussed further in **Chapter 5**.

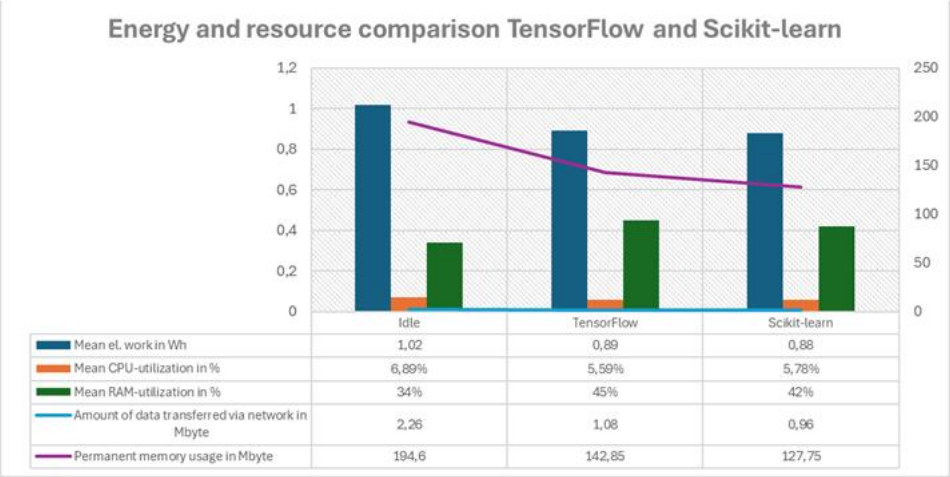

Figure 10: Comparative results of energy and resource usage of Scikit-learn and TensorFlow

The reports that were created with the Oscar software and were used as a source are provided on the zenodo platform. [9]

5 Discussion

As shown in the previous chapter, the resources required to perform clustering with TensorFlow are higher than with scikit-learn. This can be explained by the fact that with scikit-learn the algorithm does not necessarily perform 100 iterations but stops automatically when the cluster centers no longer change significantly (tol=0.0001). With TensorFlow, on the other hand, 100 fixed iterations are carried out each time.

However, there are unusual deviations when comparing the resource consumption of TensorFlow and scikit-learn and the baseline. The resource consumption in the baseline measurement is higher than when performing the clustering with the help of TensorFlow and scikit-learn.

Due to the deviations measured, it was initially assumed that errors had occurred during the measurements. For this reason, the measurements were repeated a total of three times. However, the measurement results were similar. For this reason, the last measurements were observed live. It was recognized that the resource consumption in the Power Automate waiting state in the baseline measurement levels off at approx. 30 watts [W] of electrical power per second [s]. This is also reflected in the report (**Figure 11**).

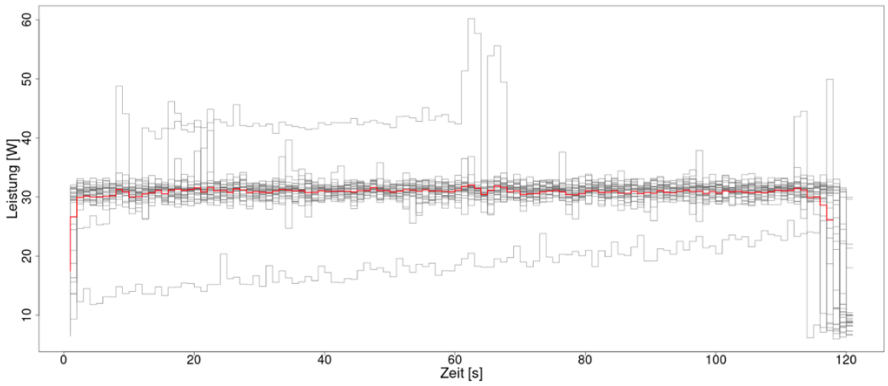

Figure 11: Electrical power in watts [W] per second [s]

This suggests that Power automate uses more resources in the baseline in the following process step (**Figure 12)** than during the actual execution of the clustering. One possible reason for this is that a relatively small number of data sets were used for clustering. Another reason could be a less efficient implementation of the wait state in Power Automate.

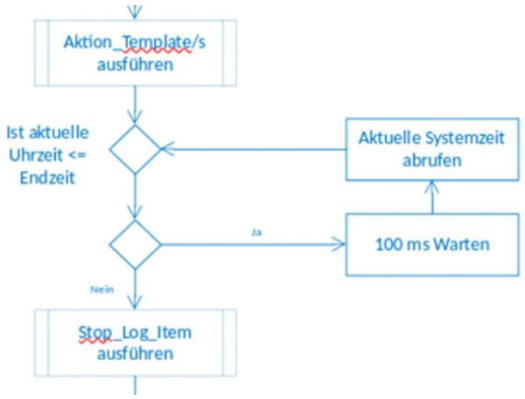

Figure 12: Process step Waiting state [15]

6 Conclusion

Overall, the results indicate that the scikit-learn implementation is more efficient due to its built-in optimizations and automatic convergence control, making it the preferred choice for classical customer segmentation tasks. The TensorFlow implementation, while offering greater flexibility, is more computationally intensive due to manual calculations and requires additional optimizations to achieve similar efficiency.

Regarding the baseline measurement, it can be said that automation using Power Automate is not suitable for the comparison made between the baseline and the implementation with TensorFlow and scikit-learn with small data sets. For future similar comparisons, either much larger data sets should be used or a different automation tool should be used, which may require fewer resources in the wait state.

References

1. **Strubell, E.; Ganesh, A.; McCallum, A. (2019):** Energy and Policy Considerations for Deep Learning in NLP; https://arxiv.org/pdf/1906.02243
2. **Chinonso Ipiankama, S. (2022):** Customer Segmentation Using K-Means Clustering; https://sampsonipiankama.medium.com/customer-segmentation-using-k-means-clustering-ae73e3d82934
3. **Kaggle (2019):** Customer Segmentation - K-Means Analysis; https://www.kaggle.com/code/obrunet/customer-segmentation-k-means-analysis
4. **Chinonso Ipiankama, S. (2022):** Customer Segmentation Using K-Means Clustering; https://sampsonipiankama.medium.com/customer-segmentation-using-k-means-clustering-ae73e3d82934
5. **Scikit-learn (2025):** https://scikitlearn.org/stable/modules/generated/sklearn.cluster.KMeans.html
6. **Vitality Learning (2022):** https://vitalitylearning.medium.com/k-means-algorithm-in-tensorflow-ad89ef14b4bd
7. **Choudhary, V. (2018):** https://www.kaggle.com/code/vjchoudhary7/kmeans-clustering-in-customer-segmentation
8. **Choudhary, V. (2018):** https://www.kaggle.com/code/vjchoudhary7/kmeans-clustering-in-customer-segmentation
9. **Melvana Mauladendra, I.; Rauch, S. (2025):** Resource consumption based usability study for the comparison between customer segmentation with KMeans clustering using TensorFlow and scikit-learn implementation; https://zenodo.org/records/14991151?token=eyJhbGciOiJIUzUxMiJ9.eyJpZCI6ImU4MGVkYTU1LTgzNTItNGY0MS05MWJmLWIwMTgyMzk4Y2JiNiIsImRhdGEiOnt9LCJyYW5kb20iOiI3NDAxNjAzMDAwMWViM2JkOTFhNmYzY2VhYzhmYzY5ZiJ9.S05uM5sUNBizLTcVkEYP-p9foLyzEDuchiI8s8vpiM1IZRo0fdI3xTUHYSAV5_FN-2AHG1hhC99qOvgnMzcAOA
10. **Junger et al.:** Conception and test of a measuring station for the analysis of the resource and energy consumption of material flow-oriented environmental management information systems (EMIS); https://moodle.htw-berlin.de/pluginfile.php/2194050/mod_resource/content/1/PaperMessAufbauJungerEtAl_EN.pdf
11. **Kern et al. (2018):** Sustainable software products—Towards assessment criteria for resource and energy efficiency; https://www.sciencedirect.com/science/article/pii/S0167739X17314188
12. **Blauer Engel:** Ressourcen- und energieeffiziente Softwareprodukte; https://produktinfo.blauer-engel.de/uploads/criteriafile/de/DE-UZ 215-202001-de Kriterien-V4.pdf
13. **Oscar:** v0.190404; https://oscar.umwelt-campus.de/
14. **Oscar:** Anleitung zur Durchführung einer Softwaremessung zur Erfassung von Energieverbrauchsdaten und Hardware-Auslastungen Messungen („Rohdaten Erfassung“), https://oscar.umwelt-campus.de/session/798ce3f0b2e98c691599b337d20be536/download/downloadDurchfuehrungEinerMessung?w=
15. **Wohlgemuth, V(2024):** Nutzungszenario erstellen