NTNU

Kunnskap for en bedre verden

Department of Computer Science

IT3212 - Data-Driven Software

Assignment 4

Authors:
Birk Strand Bjørnaa
Carl Edward Storlien
Christian Stensøe
Åsmund Løvoll

06.11.2024

Table of Contents

List of Figures

List of Tables

1 Task 1 - Preprocessing

1.1 Dataset Selection

For this task, we used the `intel_image_classification` dataset. The dataset consists of pictures of forests, buildings, mountains and the sea. Originally, the `intel_image_classification` also contained pictures of glaciers and streets, but due to pictures of glaciers having a lot in common with pictures of mountains and pictures of streets often containing pictures of buildings we chose to remove these categories. The reason behind this is so that the model could be able to distinguish the pictures better, and not be impacted of this during trainig and testing.

1.2 Preprocessing steps

1.2.1 Image resizing

Resizing images is important to standardize input sizes for machine learning models. Most neural networks require fixed-size inputs, so resizing all images ensures consistency and prevents errors. We decided a size of 128x128 as it is often a good trade-off between detail and computational efficiency, reducing the amount of processing required compared to higher resolutions like 224x224[1]. This is also done with the CNN-task in mind as this reduces computations needed.

1.2.2 Augmentation

During preprocessing the dataset is augmented by applying random transformations like horizontal flipping and random rotations (0, 90, 180, or 270 degrees). Augmentation helps to create more diverse data from the existing set, allowing the model to generalize better. This is especially useful if you wish to reduce overfitting and increase performance[2]. By introducing augmentation and slight variations in the images, the model becomes more robust to different orientations and small changes in the image.

Although augmentation has its upside, there are also disadvantages. Data augmentation can increase the computational cost of training a model as the model needs to be trained on a larger number of images. Also, if data augmentation is not used carefully, it can introduce noise into the training data[3]. This can lead to decreased performance on the test set.

We chose to rotate the pictures using `ImageOps.rotate(angle)` and also horizontally flip them using `ImageOps.mirror(img)`.

1.2.3 Normalizing

To normalize the pixel values, we divided each value by 255, scaling them to a range of [0, 1] instead of [0, 255][4]. Normalization is a standard practice in image preprocessing. Normalizing helps machine learning algorithms and especially, the optimizer work more effectively[5]. By avoiding large gradient updates, that might occur when the data values are too large, the neurons in our network will learn faster.

[1] McDermott 2024
[2] Hallaj 2024
[3] Hallaj 2024
[4] Harsh 2024.
[5] Keldenich 2024

1.2.4 Labeling

Labels were extracted from the directory structure, where each folder corresponds to a class. Labels were then encoded using `LabelEncoder().fit_transform()` from `sklearn` to convert string labels into numeric form. In machine learning, Label encoding is a technique used in data analysis to convert categorical variables into numerical format. When working with algorithms, it is usefull to convert the data, as most machine learning models only work on numerical data[6].

1.3 Feature Extraction

Feature extraction is the process of deriving meaningful information from raw data. For this task, we used the ResNet50 model, a pre-trained deep learning model, to extract high-level features from the image dataset.

We used ResNet50 for feature extraction because it balances depth and efficiency. Unlike deeper models like ResNet101 or ResNet152, ResNet50 is computationally lighter while still capturing rich features. Its architecture uses residual connections to address vanishing gradients, ensuring robust learning even with 50 layers.

Compared to VGG16, ResNet50 is more efficient, with fewer parameters but better performance. While models like Inception focus on multi-scale feature extraction, ResNet50 specializes in hierarchical features, making it versatile for diverse datasets[7]. ResNet50 has been trained on the `ImageNet` dataset, making it highly effective at capturing complex features from images, such as edges, textures, and object shapes. Overall, ResNet50 offers a practical mix of accuracy, efficiency, and is easy to implement.

The pre-trained ResNet50 model from Keras was loaded, excluding the top layer. This allowed us to use the convolutional base of ResNet50 as a feature extractor. We then extracted features according to our trai-test-validation split and extracted train features shape was [9248, 4, 4, 2048].

1.4 Feature Selection

PCA is a widely used technique for dimensionality reduction. It identifies the principal components of the data, which capture the maximum variance. By retaining only these components, PCA reduces the number of features while preserving the most important information[8]. Other dimensionality reduction algorithms were considered like *t-SNE* and *UMAP* but these are generally more suitable for complex data[9]. They also tend to use a lot more computation time.

We used PCA to retain 80% of the variance in the dataset[10]. This approach balances dimensionality reduction and information retention, ensuring the features are representative of the original data while reducing computational cost.

The `explained_variance_ratio` shows the proportion of variance captured by each principal component, while the cumulative variance indicates how many components are needed to reach the desired threshold shown in Figure 1. The number of selected features was 34. As the numbers of needed features to retain an 80% variance threshold were 34, this indicates that the data is not linearly separable.

[6] Team 2024
[7] Huilgol 2024
[8] Frost 2024
[9] Varma 2024
[10] Bruin 2024

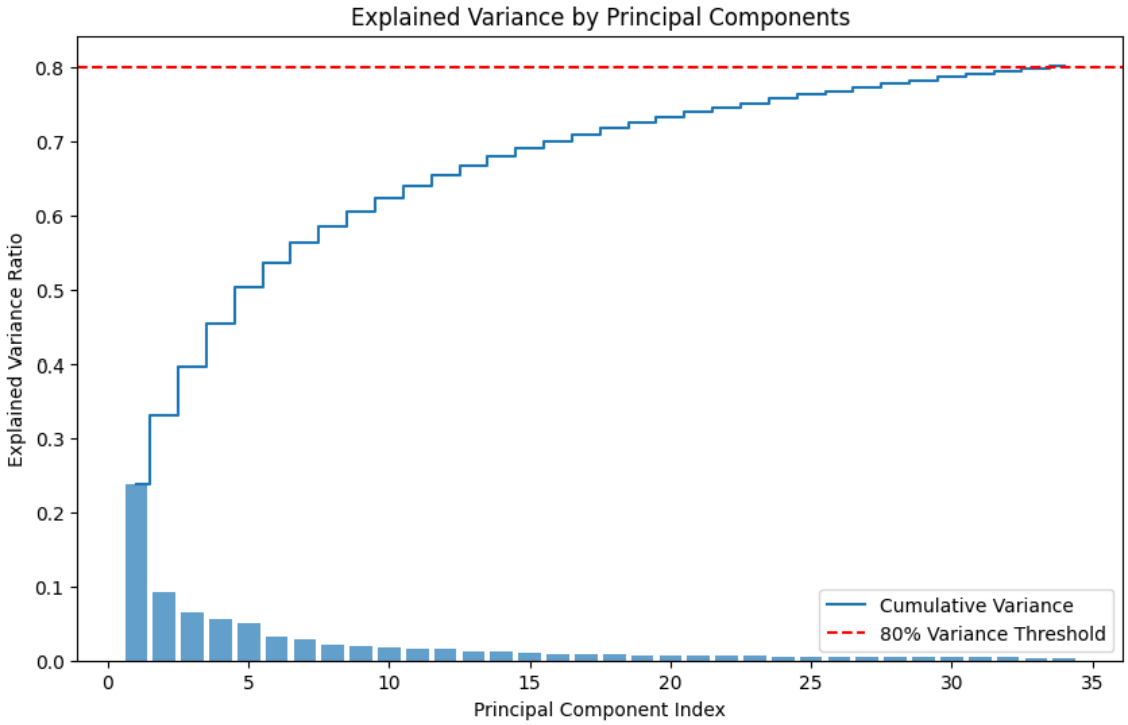


Figure 1: Cumulative variance

2 Task 1 - Implementing Basic Algorithm

2.1 Choice of Algorithm

For the task of implementing a basic machine learning algorithm for classification, we chose to apply a *Support Vector Machine (SVM)*. We considered applying both *Random Forest* and *Neural Network* as well, but ultimately decided that SVM was the most appropriate choice based on the following reasons.

1. **Effectiveness for High-Dimensional Data:**
   SVMs are well-suited for image classification, as they excel at finding optimal hyperplanes to separate data points in high-dimensional spaces. This makes them particularly effective for handling complex datasets, such as those encountered in image classification tasks, where the data may not be linearly separable. The ability of SVM to construct an optimal hyperplane for separating classes ensures reliable performance. SVMs are robust when it comes to avoiding overfitting, and this is one of its advantages compared to, for instance, Neural Networks.[11]

2. **Performance with Available Data:**
   While 9,300 images provide a reasonably sized dataset, Neural Networks typically require significantly larger datasets to generalize effectively, especially for tasks involving diverse or complex image data. In contrast, SVMs being a linear classifier can achieve strong performance with fewer samples[12], leveraging their ability to focus on the most critical data points (support vectors). This makes SVMs particularly suitable for the dataset size available in this project.

3. **Robustness Against Overfitting:**
   Overfitting is a common challenge in machine learning, particularly when the model complexity is high. SVMs use regularization through the margin-maximization principle, which makes them less prone to overfitting compared to Neural Networks that might overfit in the absence of extensive tuning or regularization techniques.

4. **Computational Efficiency:**
   While Neural Networks require substantial computational resources for training and fine-tuning multiple hyperparameters, SVMs are computationally more efficient, especially for small to medium-sized datasets. Similarly, although Random Forests can handle various types of data effectively, their ensemble nature can lead to slower predictions compared to SVM for classification tasks.

5. **Suitability for Binary and Multiclass Classification:**
   SVMs are naturally well-suited for binary classification tasks but can also be extended to multiclass problems through techniques like one-vs-one or one-vs-all.[13] This flexibility made it a suitable choice for the classification problem in this project.

[11] Geeks 2024b.
[12] Brownlee 2019.
[13] Geeks 2024c.

2.2 Implementing SVM

The SVM algorithm was implemented utilizing the SVC class from the `scikit-learn` library. The SVM model was initialized with a linear kernel, as this approach is computationally efficient and well-suited for a high-dimensional dataset such as ours. A fixed `random_state` was set to ensure reproducibility. The model was trained using the `fit` method, with the input features flattened and corresponding class labels encoded for compatibility. Once trained, the model was evaluated on the validation dataset by predicting labels with the `predict` method. The predicted labels were compared against the ground truth using the `accuracy_score` function, which provided a quantitative measure of the model's performance. To gain insight into the results, both validation accuracy and test accuracy were computed. Additionally, a classification report was generated, highlighting metrics such as precision, recall, F1-score, and support.

The test accuracy achieved by the model is **76.98%**, and the validation accuracy is **75.64%**.

Table 1: Classification Report SVM

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.74	0.78	0.76	219
1	0.91	0.88	0.89	237
2	0.69	0.82	0.75	262
3	0.77	0.61	0.68	255
Accuracy	–	–	0.77	973 samples
Macro Avg	0.78	0.77	0.77	973
Weighted Avg	0.78	0.77	0.77	973

3 Task 1 - Implementing Advanced Algorithm

3.1 Choice of Algorithm

There are multiple advanced algorithms to implement for image classification. You have, for instance, *Graph Neural Networks* or *ResNet50*, but as we already utilized ResNet50 in transfer learning in the preprocessing steps, the choice of the algorithm was ensemble learning. The reason is the advantages it has for classification tasks[14]. *CatBoost*, *AdaBoost*, and *XGBoost* are some examples just to mention a few in that are. We chose to use *CatBoost*.

1. **Suitability for Preprocessed Data:** CatBoost is particularly well-suited for structured and tabular data, as well as scenarios involving categorical features. In our project, transfer learning using ResNet50 was utilized during preprocessing to extract meaningful features from image data. Unlike some other algorithms, it does not require significant manual tuning of hyperparameters to handle complex feature interactions effectively.
2. **Handling Non-Linearity and Complexity:** CatBoost is quite good at capturing non-linear relationships in data, an essential characteristic for image classification tasks. It uses gradient boosting over decision trees and incorporates advanced techniques such as ordered boosting, which reduces overfitting caused by target leakage. Compared to simpler ensemble methods like AdaBoost, CatBoost is more robust and capable of achieving higher accuracy.
3. **Other algorithms:**
    - **Graph Neural Networks (GNNs):** While GNNs are powerful for tasks involving structured or relational data (e.g., social networks or molecular graphs)[15], they are not the best fit for traditional image classification tasks. Unlike CatBoost, GNNs would add unnecessary complexity to the pipeline for this project.
    - **XGBoost:** XGBoost is a popular gradient-boosting algorithm and shares similarities with CatBoost. However, CatBoost outperforms XGBoost when handling categorical data and requires less manual hyperparameter tuning [16]. Furthermore, CatBoost's ordered boosting approach minimizes overfitting compared to XGBoost, especially in scenarios with limited data.
    - **AdaBoost:** While AdaBoost is simpler and interpretable, it is less robust than CatBoost for complex tasks like image classification. AdaBoost tends to struggle with noisy datasets and is generally outperformed by gradient-boosting methods in terms of accuracy.

[14] Singh 2024
[15] Wikipedia 2024
[16] Swalin 2024

3.2 Implementation

The CatBoost classifier was implemented using `CatBoostClassifier()` with `MultiClass` as `loss_function` and `Accuracy` as `eval_metric`. Here we achieved **0.76%** on both the validation and test set. As we can see from the classification report, the model struggles especially with the mountain class, but is quite good at the forest class. This might be because mountains often contains forests and may confuse the model.

Table 2: Classification Report CatBoost

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Buildings	0.76	0.81	0.79	219
Forest	0.87	0.85	0.86	237
Mountain	0.69	0.73	0.71	262
Sea	0.74	0.67	0.71	255
Accuracy	–	–	0.76	973
Macro Avg	0.77	0.77	0.77	973
Weighted Avg	0.77	0.76	0.76	973

4 Task 1 - CNN

4.1 Model Architecture

The CNN algorithm for this task was implemented using TensorFlow's `Sequential()` model from `tensorflow.keras.models`. The network was constructed with a mix of convolutional, pooling, batch normalization, global average pooling, and dense layers.

The convolutional layers (`Conv2D`) are the core of the network, designed to extract spatial features from the input images. These layers detect basic features such as lines and edges. For our dataset, this might include edges where the sky meets the mountain, or the edge of a building. The second convolutional layer combines these patterns to create more complex shapes, such as the peak of a mountain or the structure of the range of trees in a forest[17]. For these layers, we tuned the number of filters, allowing values between 16 and 64 in the first block and 32 to 128 in the second block, in steps of 16 and 32 respectively. These layers use 3×3 kernels with ReLU activation and `padding='same'` to preserve spatial dimensions[18].

Following each convolutional layer, we applied max pooling (`MaxPooling2D`) to downsample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network, thereby focusing on dominant features[19]. The batch normalization layers (`BatchNormalization`) were used to stabilize training by normalizing the outputs, which helps the model converge faster and reduces sensitivity to initialization[20]. This works well in our relatively simple algorithm.

The network concludes the convolutional blocks with a `GlobalAveragePooling2D` layer, which condenses the spatial information into a single vector. In contrast to max pooling, which takes the maximum value, it takes the average of the features presented in a patch, as the name suggests.

The dense layer acts as a fully connected layer, meaning every neuron is connected to every output, in our case, from the `GlobalAveragePooling2D`. The number of units is tunable between 64 and 256 in steps of 64. Again, with ReLU activation, the non-linearity helps the model understand even more complex features, such as the smooth surface of water or the square-like form typical for a building. These layers are responsible for making predictions based on the high-level features learned by the previous layers[21]. A dropout layer, with a tunable rate between 0.2 and 0.5, is added to further mitigate overfitting by randomly deactivating neurons during training.

Finally, the output layer consists of 4 neurons with a softmax activation function, producing class probabilities for our classification.

The stride, which determines the number of pixels by which the filter moves across the input image, was left at default (1,1) for our filters, as this worked well[22].

[17] Geeks 2024a.
[18] Geeks 2024a.
[19] Geeks 2024a.
[20] Riva 2024.
[21] Geeks 2024a.
[22] Sengupta 2024.

4.2 Hyperparameter tuning

For hyperparameter tuning, we used `KerasTuner`'s Hyperband algorithm to optimize the model's performance. Key hyperparameters included the number of filters in the convolutional layers, the units in the dense layer, the dropout rate, and the learning rate of the Adam optimizer. The learning rate was sampled logarithmically between 1×10^{-4} and 1×10^{-3} to explore a wide range of step sizes.

Hyperband adaptively allocates resources, training configurations for up to 20 epochs with a reduction factor of 3, ensuring an efficient exploration of the hyperparameters. Callbacks such as `EarlyStopping` and `ReduceLROnPlateau` were incorporated to further improve the training process, stopping early when improvements plateaued and reducing the learning rate dynamically[23].

4.3 Performance

The best-performing hyperparameters gave us the following: The first convolutional block used 16 filters, while the second used 96 filters. This distribution shows that the model benefited from starting with fewer filters to detect simple edges and patterns, scaling up to more complex features in the second block. The dense layer had 192 units, sufficient for combining high-level features from the `GlobalAveragePooling2D` layer. A dropout rate of 0.2 provided enough regularization to avoid overfitting, which is supported by the high generalization accuracy. The learning rate of approximately 0.000378 allowed for steady convergence without premature stagnation.

The hyperparameter tuning process led to consistent improvement in validation accuracy across trials, with the best validation accuracy of 91.98% achieved in Trial 29, as shown in Figure 2. The earlier trials displayed greater variability, with some achieving lower accuracies due to suboptimal hyperparameter configurations. However, as the search continued, the tuning process identified increasingly effective hyperparameter sets, leading to improved performance in later trials.

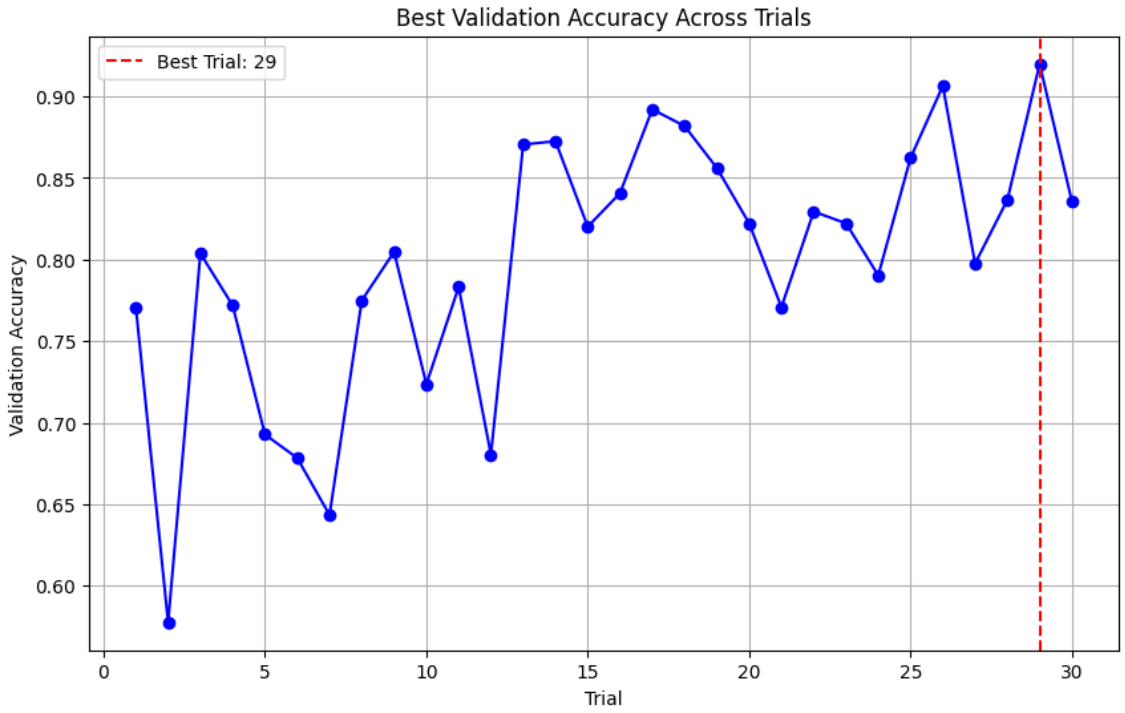


Figure 2: Validation Accuracy (Training)

[23]Sengupta 2024.

This steady increase highlights the model's ability to adapt and extract features when provided with well-optimized parameters. These results reinforce the effectiveness of the tuning strategy, giving a test loss of 0.2277 and a test accuracy of 92.19%, further validating the robustness of the selected hyperparameters.

It is also worth mentioning that we conducted a lot of manual experimentation with different numbers of filters and filter sizes. We quickly discovered that while larger values for these parameters might allowed the algorithm to capture more detailed information, they also significantly increased computational time[24]. Given the relative small size of our dataset this was also probably optimal for our model.

While the model achieved strong overall accuracy, there still is some misclassification, likely due to overlapping or ambiguous features in certain images. For instance, images of mountains with partially visible forrest or water, buildings obscured by trees, or seas with minimal visible water surface could pose challenges for the model. Figure 3 illustrates examples where multiple categories are present within the same image, potentially contributing to these inaccuracies. All of these images are categorized as "Mountain" in the dataset.

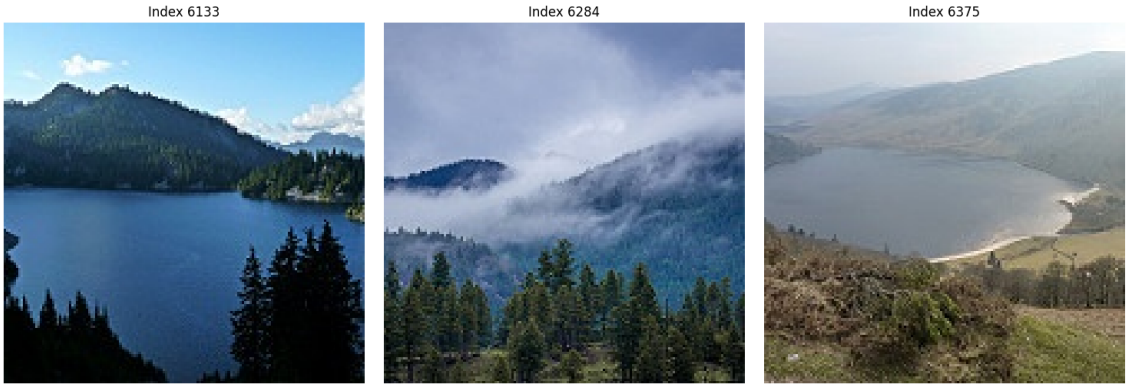


Figure 3: Selected images from the training dataset

Table 3: Classification Report CNN

Class	Precision	Recall	F1-Score	Support
Building	0.95	0.95	0.95	219
Forest	0.98	0.96	0.97	237
Mountain	0.86	0.94	0.90	262
Sea	0.90	0.85	0.87	255
Accuracy			0.92	973
Macro Avg	0.93	0.92	0.92	973
Weighted Avg	0.92	0.92	0.92	973

To conclude, as shown in Table 3, the model performed exceptionally well, achieving validation and test accuracies above 92%. For example, it achieved a precision of 98% and an F1-score of 97% for the "Forest" class, demonstrating its ability to classify this category with high accuracy. These results reflect the model's effectiveness in learning meaningful patterns from the dataset. The use of well-tuned hyperparameters and training callbacks also played an important role in ensuring stable convergence and reducing overfitting.

[24]Sengupta 2024.

4.4 Comparison and Explanation

4.4.1 Computation Time

The total computational time for the hyperparameter tuning and final training process on the CNN model was approximately 126 minutes. This includes 113 minutes spent during the hyperparameter tuning phase, where the model was trained and evaluated across multiple trials with varying hyperparameter configurations. Each trial could run for up to 20 epochs, but the `EarlyStopping` callback often reduced this to fewer epochs when validation performance stopped improving.

For instance, one of the tuned hyperparameters, the number of filters in the second convolutional layer (`conv2_filters`), was sampled from the range 32 to 128 with a step size of 32, producing four possible values (32, 64, 96, and 128) to explore. Another example is the learning rate, which was sampled logarithmically between 10^{-4} and 10^{-3}, testing multiple learning rates within this range. These combinations contributed to the overall number of trials conducted. `KerasTuner`'s Hyperband algorithm explored the hyperparameter space by focusing resources on promising configurations, balancing computational cost and performance optimization[25].

The final training of the model, using the best hyperparameters identified, required an additional 13 minutes for 30 epochs.

CatBoost was the most computationally efficient among the three algorithms. The training time for CatBoost was approximately 1.5 minutes, compared to the significantly longer 126 minutes for CNN, which included hyperparameter tuning. CatBoost's efficiency can be attributed to its gradient-boosting approach and preprocessed feature handling, which minimizes computational overhead. SVM, on the other hand, also demonstrated fast training times but struggled with scalability as the dataset size and feature complexity increased. CNN required the longest computation time due to the complexity of convolutional layers and the large number of parameters involved, particularly during hyperparameter optimization.

4.4.2 Performance

Looking at the classification reports of the three algorithms (Tables 1, 2, and 3), CNN demonstrated the best overall performance with an accuracy of 92.19%, significantly outperforming CatBoost (76%) and SVM (76.98%). CNN did great in all classes, achieving a precision of 0.95 and an F1-score of 0.95 for the "Building" class and an F1-score of 0.90 for "Mountain," which was the most challenging class for both CatBoost and SVM. This highlights CNN's ability to capture complex spatial features in raw image data.

CatBoost, despite its lower overall accuracy, performed competitively for certain classes, such as "Forest", which might have more distinct features, achieving an F1-score of 0.86. However, its performance dropped for categories like "Sea", where it achieved an F1-score of only 0.71. CatBoost, relying on features extracted using ResNet50, achieved 76% accuracy. While this was lower than CNN, it was still competitive given its efficiency and ease of use.

SVM showed similar accuracy to CatBoost but struggled even more with classes that had overlapping or nonlinear feature boundaries. For instance, SVM achieved an F1-score of 0.68 for "Sea", the lowest among the three models, and 0.75 for "Mountain", reflecting its difficulty in handling complex patterns and textures that are common in these kinds of landscapes. Its reliance on linear separability likely contributed to these challenges, particularly for images where categories shared similar visual features as mentioned earlier in the report.

[25] Wadekar 2024.

Overall, CNN's ability to deliver consistently high metrics across all classes, particularly in seemingly challenging categories like "Mountain" and "Sea", demonstrates its superiority for this task. While CatBoost and SVM offered computational efficiency, their reliance on pre-extracted features with ResNet50 for CatBoost and linear separability for SVM limited their effectiveness compared to CNN's learning capabilities. One can argue that CNN's higher computational cost is justified by its ability to learn and generalize from raw image data, making it the most effective model for this classification problem.

4.4.3 Strengths and Weaknesses

- **CatBoost:**
  - **Strengths:** Highly efficient computationally due to built-in categorical handling and gradient boosting. Robust against overfitting due to ordered boosting and effective handling of preprocessed features.
  - **Weaknesses:** Limited by the quality of feature extraction (ResNet50), which may not capture fine-grained details present in raw images.
- **CNN:**
  - **Strengths:** Excels at learning complex patterns directly from raw image data, offering the best performance among the three models. Highly customizable with hyperparameter tuning to adapt to various datasets.
  - **Weaknesses:** Computationally intensive, with high training times and resource requirements. More prone to overfitting without careful regularization and data augmentation.
- **SVM:**
  - **Strengths:** Effective for smaller datasets and less computationally expensive than CNN. Strong generalization for binary or simple multiclass problems.
  - **Weaknesses:** Struggles with scalability and non-linear relationships in high-dimensional data, making it less effective for complex image classification tasks.

4.4.4 Conclusion

The choice of algorithm depends on the trade-off between computational efficiency and accuracy. CNN offers the best performance for image classification tasks but at a higher computational cost. CatBoost provides a middle ground, balancing efficiency and performance when using pre-extracted features. SVM, while efficient and easy to implement, is limited in handling the complexities of image data compared to the other two.

5 Task 2 - Preprocessing

5.1 Problem Statement

For the second task, we chose to experiment with clustering algorithms on the `smoking_drinking_dataset`. The intention behind the dataset is a classification task of predicting smokers and drinkers based on their body signals, but we find it useful for experimenting with clustering as a type of unsupervised learning.

The dataset is quite large with almost `1 million` data points. It contains features on sex, age, height, and weight as well as other body signals and the status of smoking and drinking behavior[26]. When experimenting with different clustering algorithms, it will be interesting to see how well the clusters correlate with the smoking and drinking status.

5.2 Preprocessing, Feature Extraction and Feature Selection

Preprocessing of the dataset is important to ensure the input to the machine learning models is of high quality and relevance. This will lead to better performance in the clustering of data points. The first five rows of the dataset are shown in table 4.

Table 4: First five rows

Variable	**0**	**1**	**2**	**3**	**4**
sex	Male	Male	Male	Male	Male
age	35	30	40	50	50
height	170	180	165	175	165
weight	75	80	75	80	60
waistline	90.0	89.0	91.0	91.0	80.0
sight_left	1.0	0.9	1.2	1.5	1.0
sight_right	1.0	1.2	1.5	1.2	1.2
hear_left	1.0	1.0	1.0	1.0	1.0
hear_right	1.0	1.0	1.0	1.0	1.0
SBP	120.0	130.0	120.0	145.0	138.0
DBP	80.0	82.0	70.0	87.0	82.0
BLDS	99.0	106.0	98.0	95.0	101.0
tot_chole	193.0	228.0	136.0	201.0	199.0
HDL_chole	48.0	55.0	41.0	76.0	61.0
LDL_chole	126.0	148.0	74.0	104.0	117.0
triglyceride	92.0	121.0	104.0	106.0	104.0
hemoglobin	17.1	15.8	15.8	17.6	13.8
urine_protein	1.0	1.0	1.0	1.0	1.0
serum_creatinine	1.0	0.9	0.9	1.1	0.8
SGOT_AST	21.0	20.0	47.0	29.0	19.0
SGOT_ALT	35.0	36.0	32.0	34.0	12.0
gamma_GTP	40.0	27.0	68.0	18.0	25.0
SMK_stat_type_cd	1.0	3.0	1.0	1.0	1.0
DRK_YN	Y	N	N	N	N

5.2.1 Missing Values and Duplicates

There are `991346` rows, and the dataset has no missing values. The Pandas method `isna()` was used to inspect this. The Pandas method `duplicated()` found `26` duplicate rows. It is not known whether those duplicate rows represent the same individual or have the same body signals by coincidence. `26` rows out of `991346` is nonetheless a negligible ratio, so we chose to remove them since duplicates can lead to overfitting of the models.

[26] Soo.Y 2024.

5.2.2 Categorical Features

The categorical features are `sex, hear_left, hear_right, urine_protein, SMK_stat_type` and `DRK_YN`. The categorical string values must be encoded to numerical values before they can be processed by a machine learning model. To do this, we used the class `LabelEncoder` from `sklearn.preprocessing` package. The unique values and their mappings are shown in table 5. We encoded the already numerical values too, so that all values are either 0/1 or start from 0 for the sake of simplicity.

Categorical Feature	Original Values	Encoded Values
Sex	Female, Male	0, 1
hear_left	1.0 (normal), 2.0 (abnormal)	0, 1
hear_right	1.0 (normal), 2.0 (abnormal)	0, 1
urine_protein	1 (-), 2 (+/-), 3 (+1), 4 (+2), 5 (+3), 6 (+4)	0, 1, 2, 3, 4, 5
SMK_stat_type_cd	1.0 (never), 2.0 (used to smoke but quit), 3.0 (still smoke)	0, 1, 2
DRK_YN	N, Y	0, 1

Table 5: Label-encoded categorical features and their mappings

5.2.3 Outliers

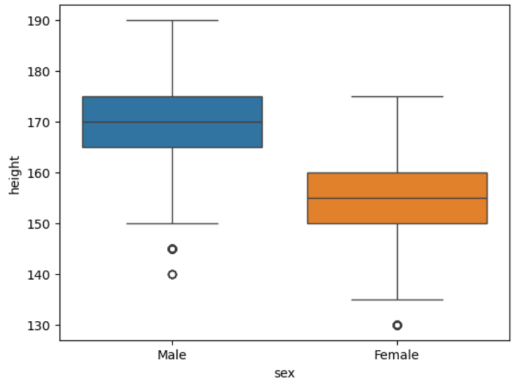
The univariate outliers in the dataset can be identified by using a box plot for the features. The `seaborn` package was used to generate the box plots. Univariate outliers have been identified visually in almost all of the features in this dataset:
`height, weight, waistline, sight_left, sight_right, SBP, DBP, BLDS, tot_chole, HDL_chole, LDL_chole, triglyceride, hemoglobin, serum_creatinine, SGOT_AST, SGOT_ALT` and `gamma_GTP`. Figure 4 shows some of the box plots.

The IQR (Inter-Quartile Range) threshold method of marking univariate outliers marked **36%** of the dataset as outliers with the standard threshold of `1.5 * IQR`. This is a very high ratio of outliers, which might suggest that the dataset has very high variability and that we should try to increase the threshold to only capture the extreme outliers, like the ones in figure 4b and 4c. We increased the threshold to `2 * IQR` and got a ratio of **21%**. We stuck with the standard `1.5 * IQR` and capped the outlier values at the upper and lower bounds. Table 6 shows the summary statistics of the numerical features after handling outliers.

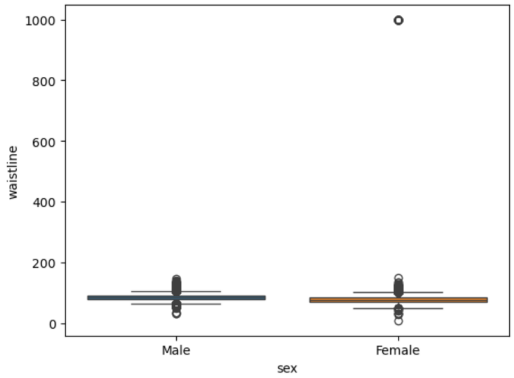
5.2.4 Scaling

Models benefit from scaled data because the magnitude of feature values does not affect the feature importance, meaning how much the model's behavior is affected by a feature value. For scaling, we used the `StandardScaler` from `sklearn.preprocessing`. This scaling method normalizes the feature values by adjusting the mean to 0 and variance to 1. Another common method is `MinMaxScaler` which scales the feature values to a fixed range, commonly 0-1, where the range is determined by the edge values. Algorithms like `PCA` and `k-means clustering` are sensitive to the feature variance, and `StandardScaler` is therefore often a better choice. Because we intend to use these two algorithms later, we chose to scale our data with `StandardScaler`.

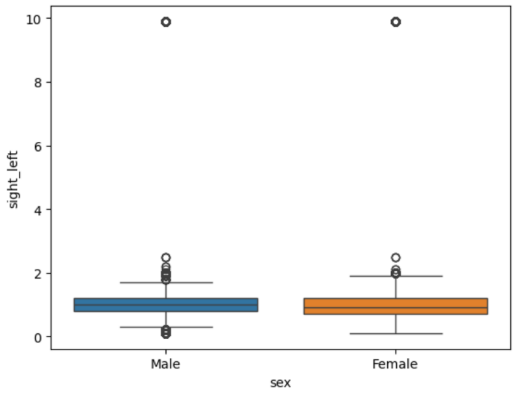
After scaling, we shuffled the dataset because we might need to cut down on the number of rows due to limited computational power for the clustering algorithms. Shuffling ensures that the x number of rows we, in that case, end up with is a representative sample of the dataset.



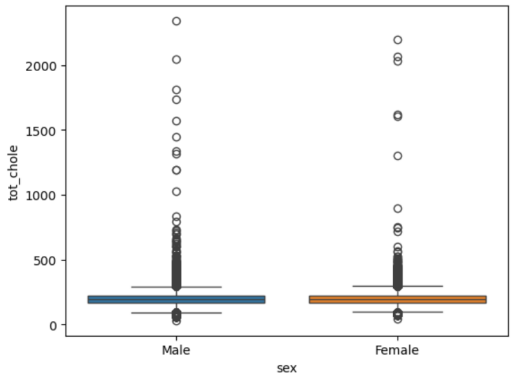
(a) Box plot of `height`



(b) Box plot of `waistline`



(c) Box plot of `sight_left`



(d) Box plot of `tot_chole`

Figure 4: Examples of univariate outliers identified with box plots.

Table 6: Summary Statistics

Variable	**Count**	**Mean**	**Std**	**Min**	**25%**	**50%**	**75%**	**Max**
age	991346.0	47.6	14.2	20.0	35.0	45.0	60.0	85.0
height	991346.0	162.2	9.3	135.0	155.0	160.0	170.0	190.0
weight	991346.0	63.1	11.9	35.0	55.0	60.0	70.0	90.0
waistline	991346.0	81.1	9.5	50.5	74.1	81.0	87.8	105.0
sight_left	991346.0	1.0	0.3	0.1	0.7	1.0	1.2	2.0
sight_right	991346.0	1.0	0.3	0.2	0.7	1.0	1.2	1.8
hear_left	991346.0	0.0	0.2	0.0	0.0	0.0	0.0	1.0
hear_right	991346.0	0.0	0.2	0.0	0.0	0.0	0.0	1.0
SBP	991346.0	122.3	14.1	80.0	112.0	120.0	131.0	160.0
DBP	991346.0	76.0	9.7	49.0	70.0	76.0	82.0	105.0
BLDS	991346.0	98.0	14.2	63.0	88.0	96.0	105.0	135.0
tot_chole	991346.0	195.3	37.2	94.0	169.0	193.0	219.0	295.0
HDL_chole	991346.0	56.7	14.4	17.5	46.0	55.0	66.0	101.0
LDL_chole	991346.0	112.8	33.9	20.0	89.0	111.0	135.0	204.0
triglyceride	991346.0	125.3	70.1	1.0	73.0	106.0	159.0	332.5
hemoglobin	991346.0	14.3	1.5	10.5	13.2	14.3	15.4	18.1
urine_protein	991346.0	0.1	0.4	0.0	0.0	0.0	0.0	5.0
serum_creatinine	991346.0	0.9	0.2	0.3	0.7	0.8	1.0	1.6
SGOT_AST	991346.0	24.6	8.0	3.5	19.0	23.0	28.0	47.5
SGOT_ALT	991346.0	23.8	12.7	1.0	15.0	20.0	29.0	60.5
gamma_GTP	991346.0	31.5	23.6	1.0	16.0	23.0	39.0	103.5

5.3 Feature Extraction

We decided not to extract any features from the dataset, as we are interested in the clustering of the only information present in the dataset.

5.4 Feature Selection

Since we are experimenting with clustering algorithms in this task, we chose not to select features based on relevance for smoking and drinking status. We want to inspect the clusters based on all of the body signals, not only the ones our domain knowledge tells us are relevant. After all, we are not trying to cluster the dataset based on the smoking and drinking status. Instead, we want to see how well the smoking and drinking status aligns with the clusters found for the sake of curiosity. We dropped the `SMK_stat_type_cd` and `DRK_YN` features from the dataset, and will instead use them as labels.

5.4.1 Correlation Matrix

Figure 5 shows a pairwise correlation matrix between the features. For instance, we can see that `waistline` is highly correlated with `weight` (**0.8**), which is natural. Since male and female bodies are fundamentally different, splitting the dataset into male and female gives a better picture of which features are correlated. Figure 6 and 7 show the correlation matrix for males and females, respectively.

If we were to perform feature selection, these correlation matrices give an indication of which features could be redundant and irrelevant. However, we keep all of the features as mentioned earlier.

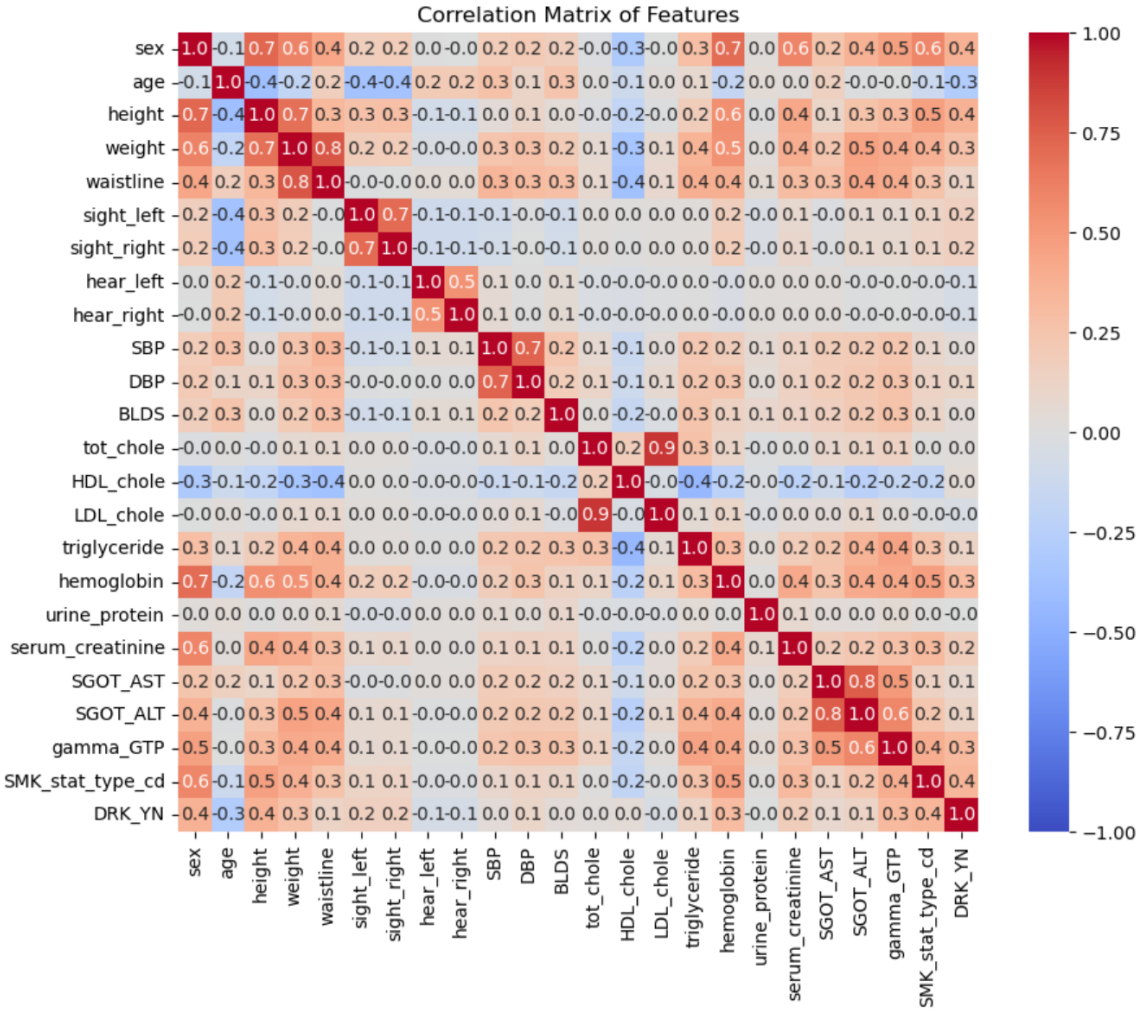


Figure 5: Correlation Matrix

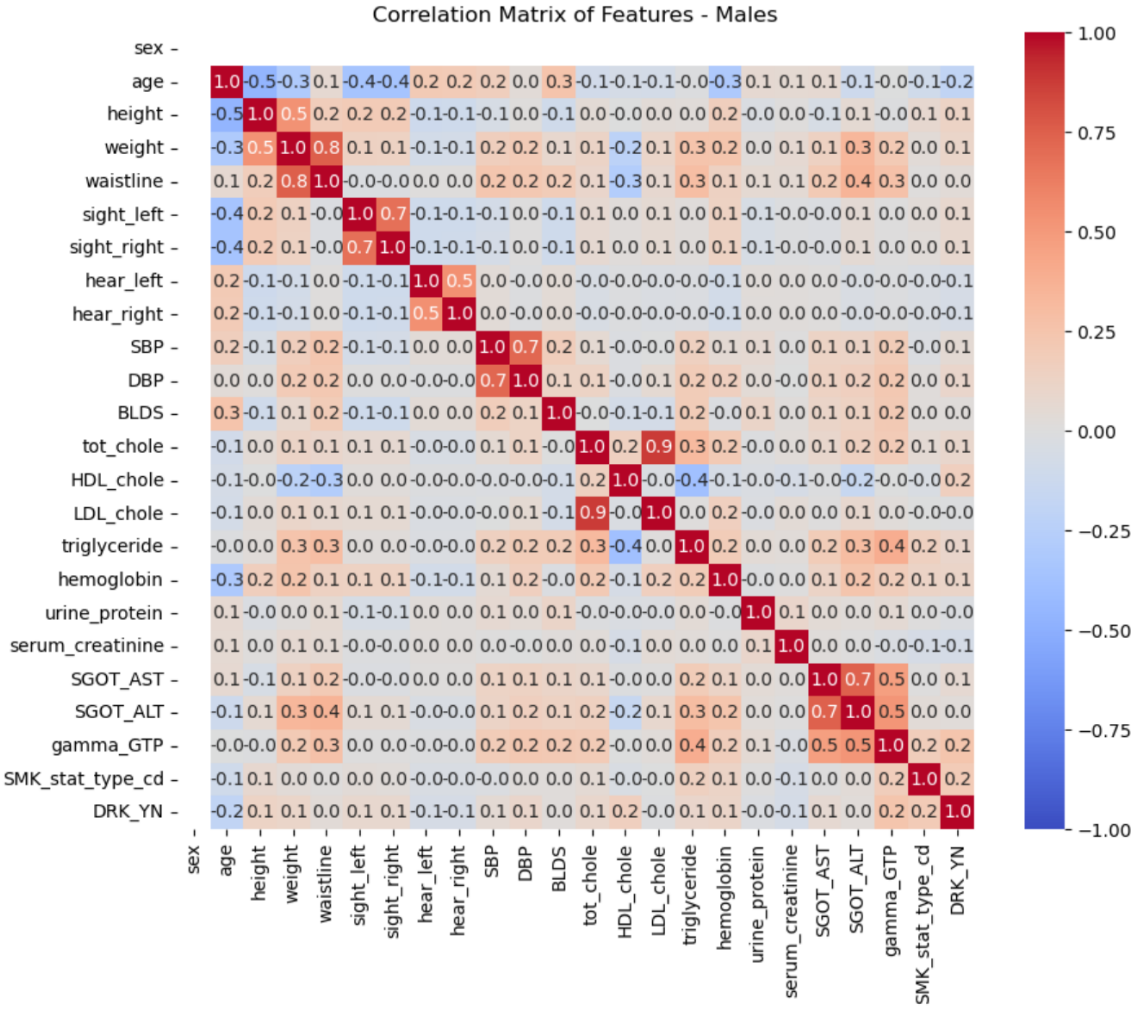


Figure 6: Correlation Matrix for Males

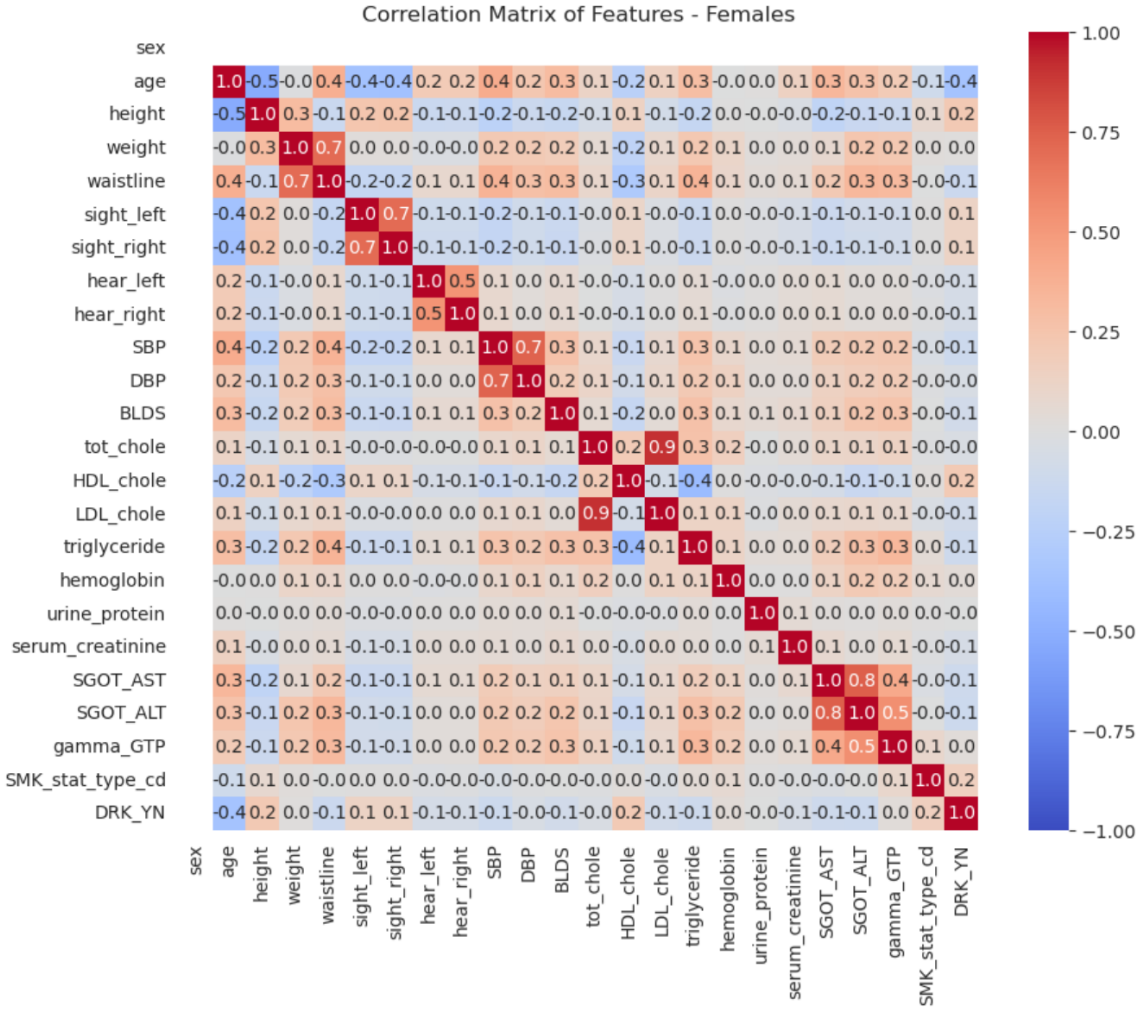


Figure 7: Correlation Matrix for Males

5.5 Dimensionality Reduction

Running the clustering algorithms with all the features is too computationally expensive for some of the models. An option for dimensionality reduction is using `Principal Component Analysis (PCA)`. This method tries to retain the variance in the data while capturing linear relationships between the features. It is therefore suitable for clustering algorithms that work in linear space. The amount of retained variance in the dataset is a function of the number of components, i.e. dimensions, kept.

For the models that need to work with fewer features than in the original preprocessed dataset, we used `PCA` to reduce the dimensionality down to `2` components, which allows us to plot the data on a 2D plot. See Figure 8 and 9 for a visualization of the data points in 2 dimensions after using PCA.

The visualizations reveal several aspects of the data. First off, the males constitute the largest part of smokers ... Secondly, both the drinkers and smokers are evenly spread out in the PCA visualization of males. This makes it hard later when we want to visually determine how well the clustering aligns with the drinkers and smokers.

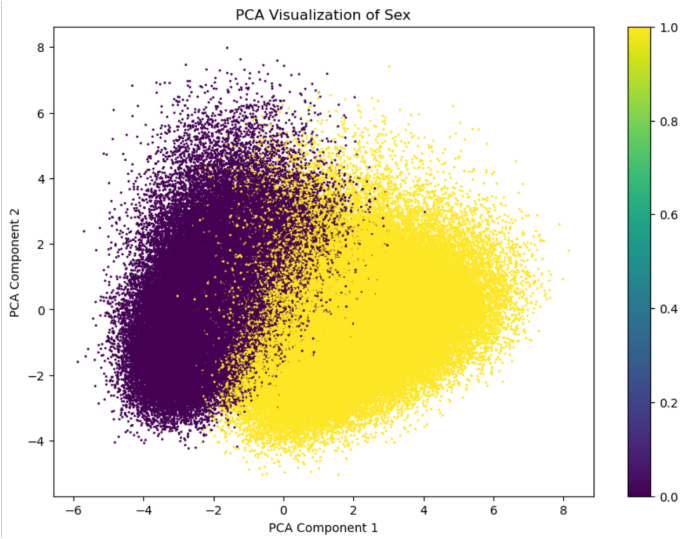
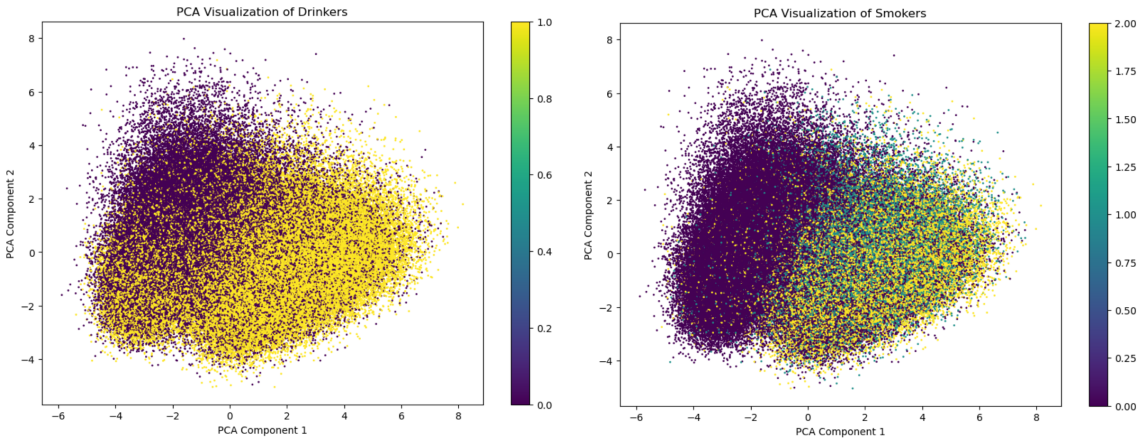


Figure 8: PCA Visualization of Sexes (Males are Yellow)



(a) PCA Visualization of Drinkers

(b) PCA Visualization of Smokers

Figure 9: PCA Visualization of Drinkers and Smokers

6 Task 2 - Clustering Algorithms

There are essentially 6 different clusters we ideally would want our clustering algorithm to identify: 2 drinking states * 3 smoking states. However, we cannot train an unsupervised learning model to identify the clusters that we ideally want. The model finds its own patterns in the data. It is, for instance, intuitively easier to cluster based on sex.

With the features `SMK_stat_type_cd` and DRK_YN removed, our dataset has **22** features, i.e. dimensions. To visualize the clustering, we utilize the `t-SNE` algorithm from the `openTSNE` package and the `PCA` decomposition from the `sklearn` package. These two algorithms can bring down the dimensions of the dataset to 2 for a 2D visualization of the data points. We could also bring it down to 3 for a 3D visualization. It would be more complex to view but could reveal clusters better.

The main difference between the two dimensionality reduction techniques lies in the assumption of linear data. `PCA` assumes a linear combination of properties in the data, whereas `t-SNE` does not. This means that `PCA` can fail to preserve the structure of the data if the data is non-linear. This may not be the case for `t-SNE`. The reason is that `PCA` projects the data down to a lower dimension based on which directions preserve the highest amount of variance in the data, but non-linear structures might not be captured in the calculation of variance in different directions. The `t-SNE` algorithm does not try to maximize the amount of variance in the data preserved, instead, it tries to preserve the similarity between data points based on their relative distances.

To start, we run the t-SNE algorithm on the dataset with 2 components (dimensions) and plot the result. Figure 10 shows the `200 000` first rows in our original dataset plotted on a 2D scatter plot with males marked as yellow. We can easily see that the `t-SNE` has already clustered the sexes very well, because `t-SNE` is in some sense a clustering algorithm. However, we are looking to explore other clustering algorithms. We used the first `200 000` rows because of `t-SNE` being a computationally expensive algorithm to run.

If we now take a look at the drinking and smoking labels, we get the plots in Figure 11. It is possible to see that the females have a low number of drinkers and almost no smokers if we compare the clusters to Figure 10. This aligns with what we visualized with PCA as well.

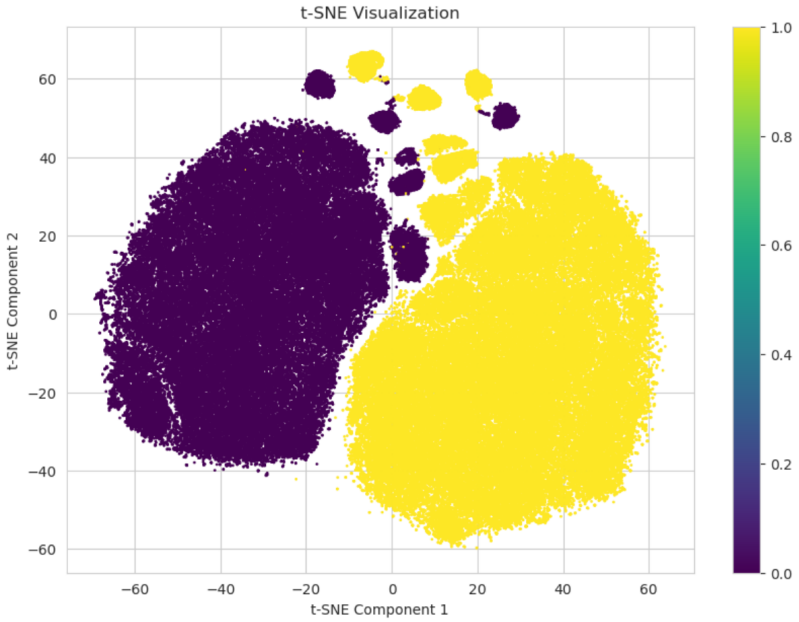
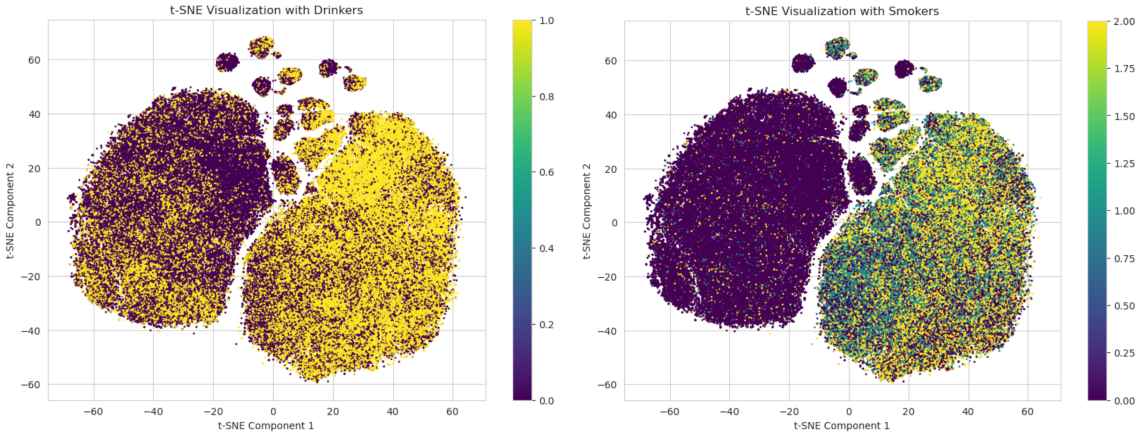


Figure 10: TSNE Visualization of Sexes (Males are Yellow)



(a) TSNE Visualization of Drinkers

(b) TSNE Visualization of Smokers

Figure 11: TSNE Visualization of Drinkers and Smokers

6.1 K-Means

K-Means is a centroid-based clustering algorithm that aims to partition the dataset into `k` distinct clusters. It works iteratively by initializing `k` cluster centroids, assigning data points to the nearest centroid based on Euclidean distance, and updating the centroids to the mean position of their assigned points. This process repeats until convergence, where the centroids stabilize. K-Means is efficient for large datasets, making it suitable for our dataset of nearly `1 million` rows. It assumes spherical clusters of relatively equal sizes, which aligns well with many features in our dataset, such as height, weight, and waistline, which tend to form Gaussian-like distributions. The hard part with `K-Means` is to find the number of clusters, i.e. centroids/`k`.

The choice of `K-Means` for this dataset is justified due to its simplicity and scalability. By using `K-Means`, we can analyze whether the primary patterns in the dataset align with intuitive clusters, such as males and females or smokers and drinkers. Its reliance on centroids makes it computationally efficient for our large dataset. Additionally, since the dataset has been standardized, `K-Means` is particularly effective as it minimizes distortions caused by varying feature scales. Since `K-Means` is computationally efficient, it is possible for us to run the clustering algorithm on the dataset without dimensionality reduction, thus preserving both linear and non-linear structures. For visualization, the `t-SNE` components were used to view this in 2D.

6.1.1 Elbow Plot

A method of finding an appropriate `k` to begin with is plotting the `inertia` when running the `K-Means` algorithm with different `k-values`.

As shown in Figure 12, the elbow plot suggests diminishing returns in inertia reduction beyond `k = 2`, making it a suitable choice for initial cluster analysis.

We will also try `k = 5` to experiment with another value for `k` and compare the results.

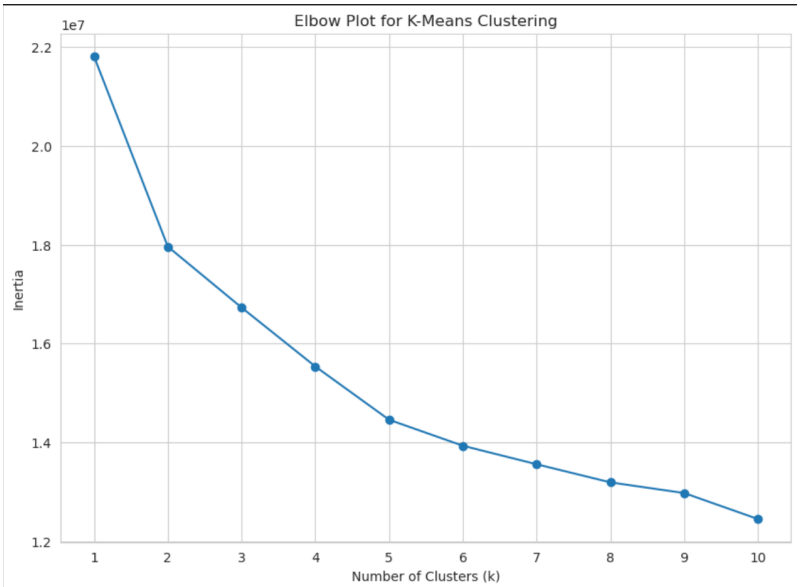


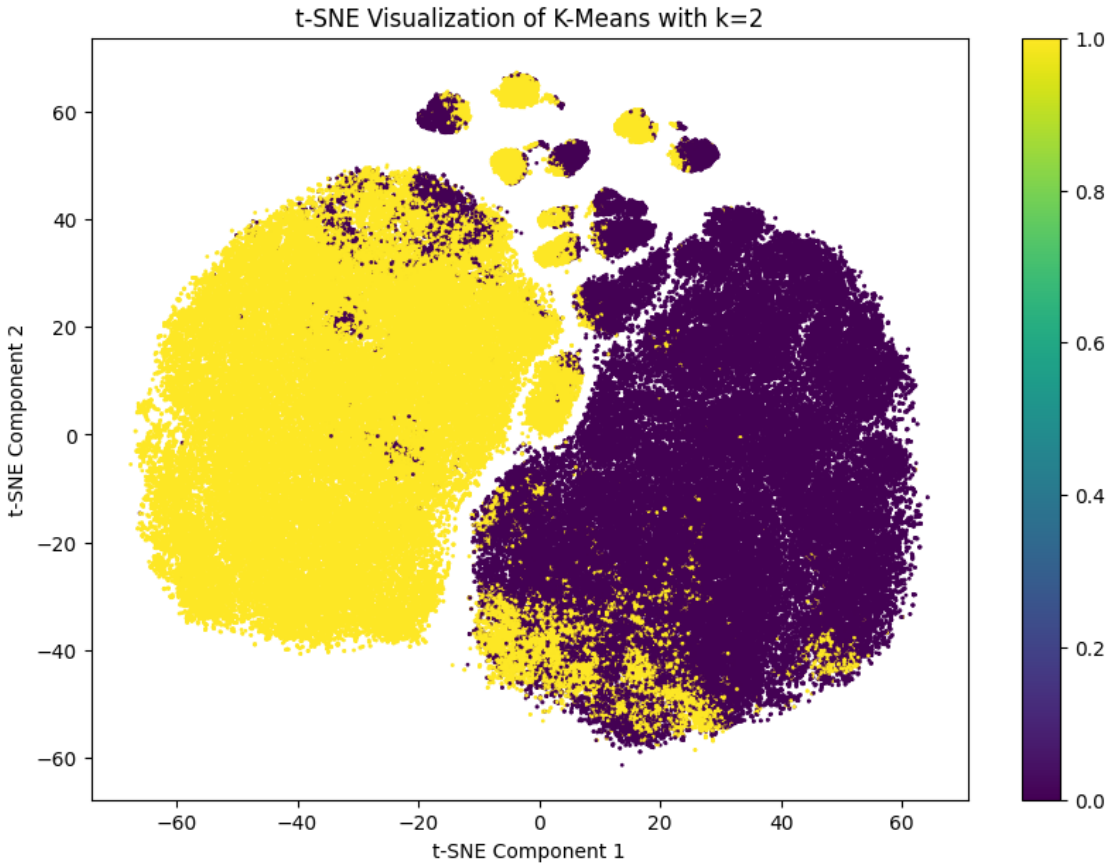
Figure 12: K-Means Elbow Plot

6.1.2 K-Means with `k = 2`

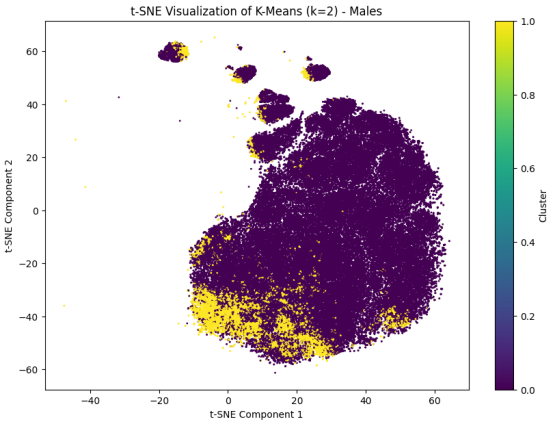
The K-Means clustering algorithm performed effectively with `k = 2`, successfully separating the data into two distinct clusters, as seen in figure 13. This separation aligns well with clear demographic distinctions in the dataset, such as sex, which often correlates strongly with features like height, weight, and waistline. In the t-SNE visualization, the two clusters are spatially distinct, with minimal overlap, indicating that the clustering captures a primary division in the data. The purple cluster corresponds to males (opposite of earlier t-SNE visualization), while the yellow cluster represents the females. This demonstrates that K-Means with `k = 2` identifies a dominant natural division in the dataset. Only a small ratio of males and females are misclassified, as indicated by Figure 13.

6.1.3 K-Means with `k = 5`

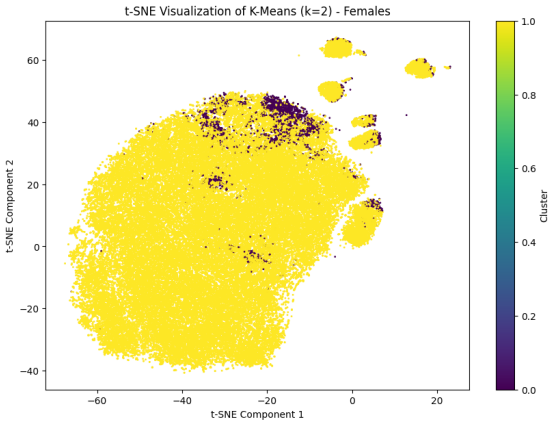
With `k = 5`, the clustering algorithm reveals more nuanced groupings within the dataset. The t-SNE visualization shows subclusters within the broader clusters identified with `k = 2`. The two clusters split into smaller subgroups, reflecting finer divisions within the group of males and females. This suggests that K-Means with `k = 5` captures additional complexity in the data on body signals. The "male cluster" has been split into mainly two subclusters (purple and cyan). The "female cluster" has also been split into mainly two subclusters (green and blue). The small yellow cluster is distinct with a clear separation from the other clusters, split between males and females. While the clusters align reasonably well with the structure revealed by t-SNE, some overlap between clusters remains, indicating that `k = 5` introduces granularity at the cost of sharper boundaries — at least when projected in **t-SNE space**.



(a) K-Means Clustering with $k = 2$

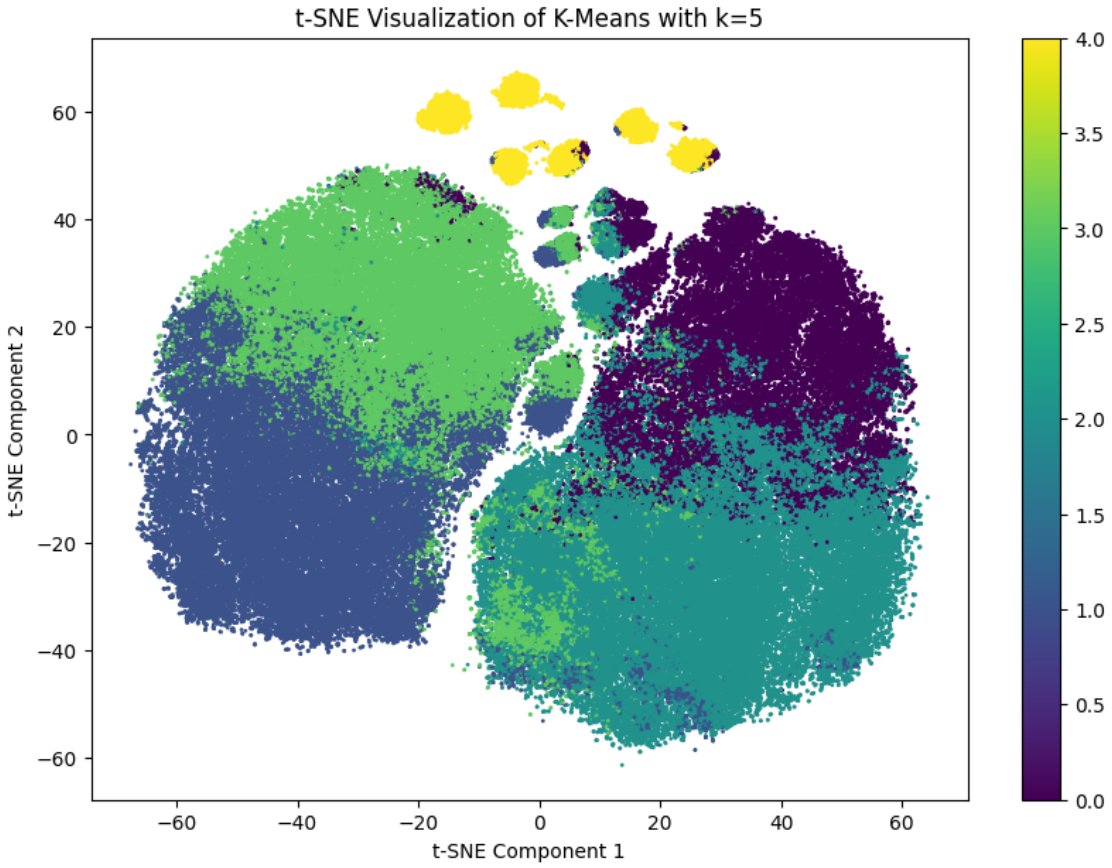


(b) K-Means Clustering with $n = 2$ of Males

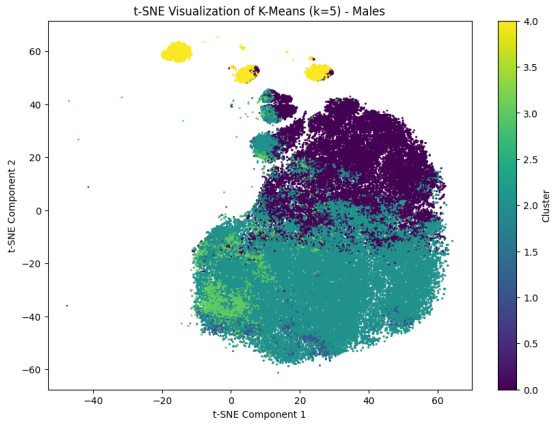


(c) K-Means Clustering with $n = 2$ of Females

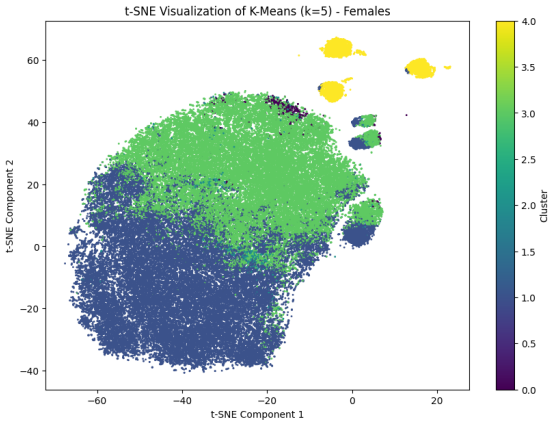
Figure 13: K-Means Clustering Results with $n = 2$



(a) K-Means Clustering with $k = 5$



(b) K-Means Clustering with $n = 5$ of Males



(c) K-Means Clustering with $n = 2$ of Females

Figure 14: K-Means Clustering Results with $n = 5$

6.2 Hierarchical Clustering

`Hierarchical Clustering` is clustering technique, where initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed [27]. This can be visualized through dendrograms, like the ones in Figure 16 and Figure 18. This is also referred to as the Agglomerative Clustering method within the Hierarchical Clustering methods [28]. Another reason for implementing this Hierarchical Clustering method is that it is less computationally expensive than the traditional implementation. Therefore we predefined the number of clusters.

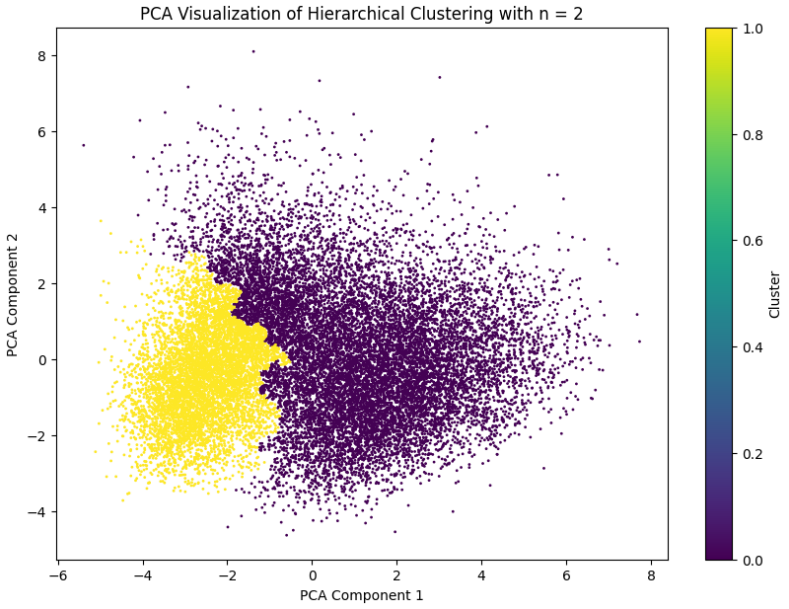
This method has both advantages and drawbacks. The method is quite easy to implement and also provides insight into the inherent connections between observations within the data through the mentioned dendrogram. A quick look into the diagram gives perspective into how similarities brought certain portions of the information into clusters. Also, hierarchical clustering can capture clusters of different shapes and sizes with much better efficiency than k-means [29].

On the other hand, the method is quite sensitive to outliers and noise, and also its computational complexity. It is less scalable for bigger datasets. Despite the disadvantages, we wished to implement this algorithm as it is quite different from **Fuzzy-C means** and **K-means** in the method it uses. As a result, we had to create a subset of the data due to the computational requirement of the algorithm. By testing the method at a subset of 50000 data points the algorithm crashed and we were forced to run the algorithm on a 20000 data point subset. This is not optimal due to such large chunks of data being left out, however the results were not too bad.

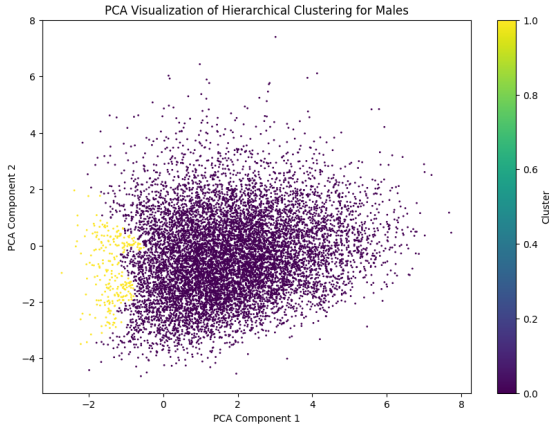
6.2.1 Hierarchical Clustering with `n = 2`

We predefined the number of clusters as `n = 2`. In Figure 15, the scatter plots of the dataset with PCA components are shown. In Figure 15b and 15c, only the males and females are plotted with the cluster labels. Here we can visualize how the clusters behave when filtering out females and males. This method tends to struggle to distinguish males and females for the dataset. It produces quite a good split for males, but struggles with distinguishing females.

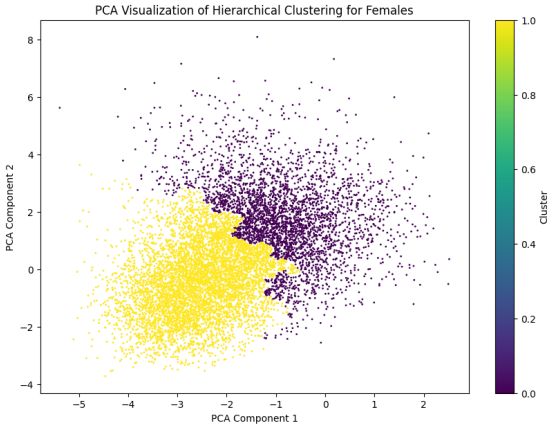
[27] Patlolla 2024
[28] Aditya 2024
[29] Kuchciak 2024



(a) Hierarchical Clustering with $n = 2$



(b) Hierarchical Clustering with $n = 2$ of Males



(c) Hierarchical Clustering with $n = 2$ of Females

Figure 15: Hierarchical Clustering Results with $n = 2$

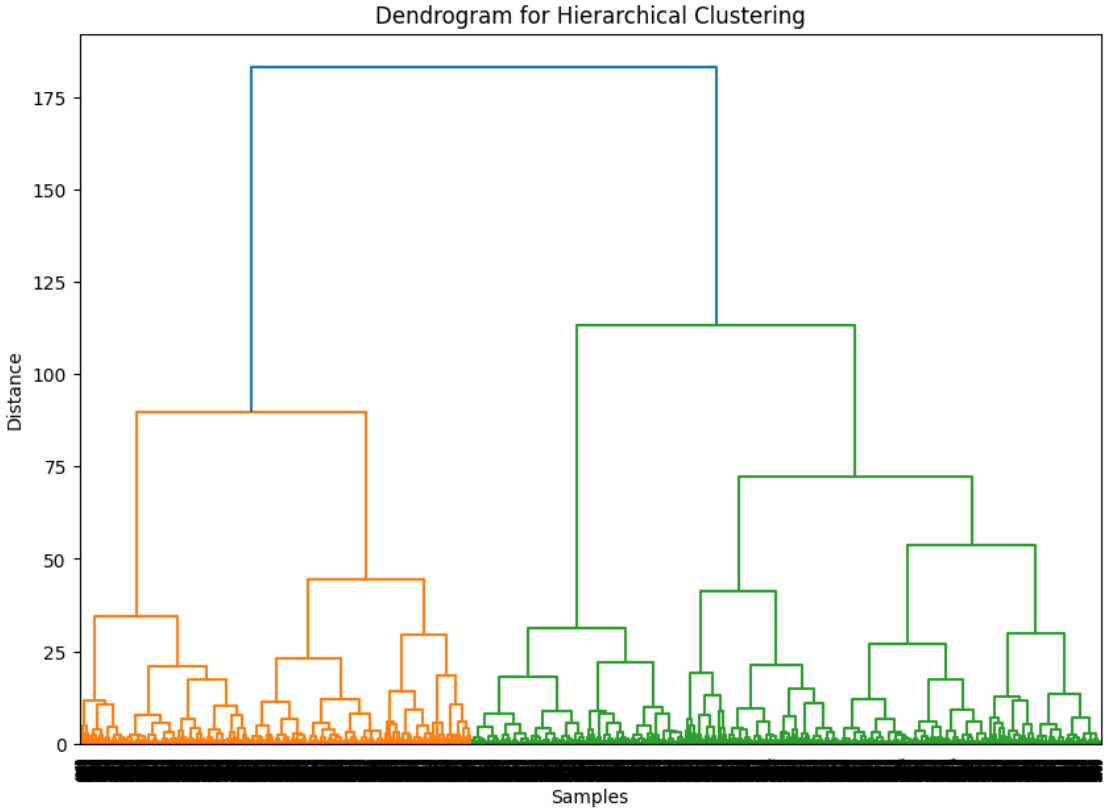
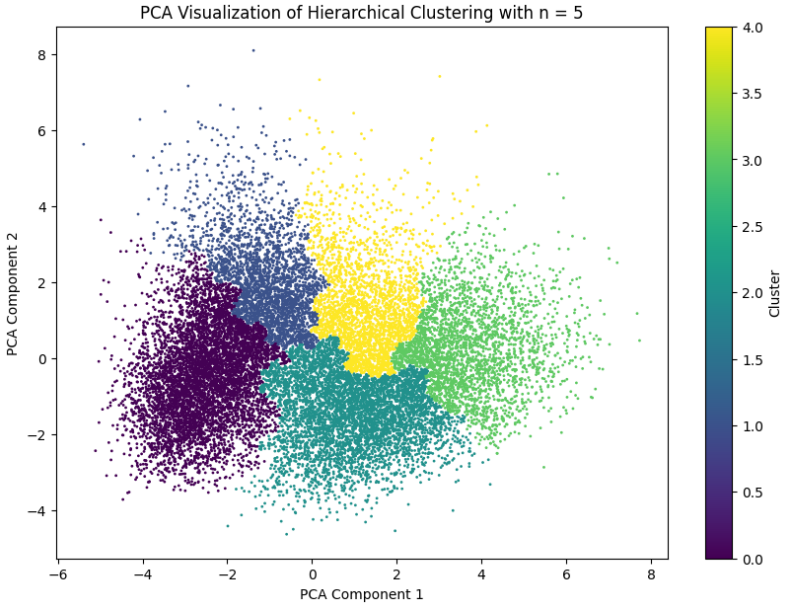


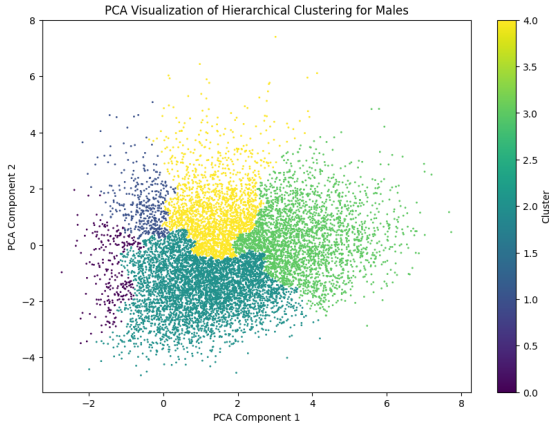
Figure 16: Hierarchical Clustering dendrogram with $n = 2$

6.2.2 Hierarchical Clustering with `n = 5`

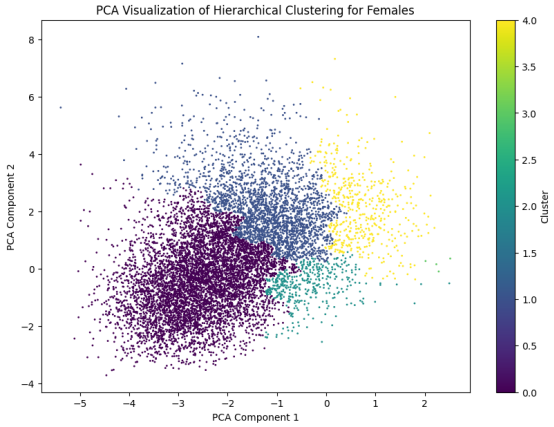
Here we predefined the number of clusters as `n = 5`. In Figure 17, the scatter plots of the dataset with PCA components are shown. In Figure 17b and 17c, the males and females are again plotted with the cluster labels. Here we can visualize how the clusters behave when filtering out females and males. When increasing the number of predefined clusters, the issues with hierarchical clustering really shows. Since the predefined clusters force the model to split data points, it leads to the algorithm clustering data into wrong categories. As Figure 17b and 17c shows, there is no distinguished difference between male and female.



(a) Hierarchical Clustering with $n = 5$



(b) Hierarchical Clustering with $n = 5$ of Males



(c) Hierarchical Clustering with $n = 5$ of Females

Figure 17: Hierarchical Clustering Results with $n = 5$

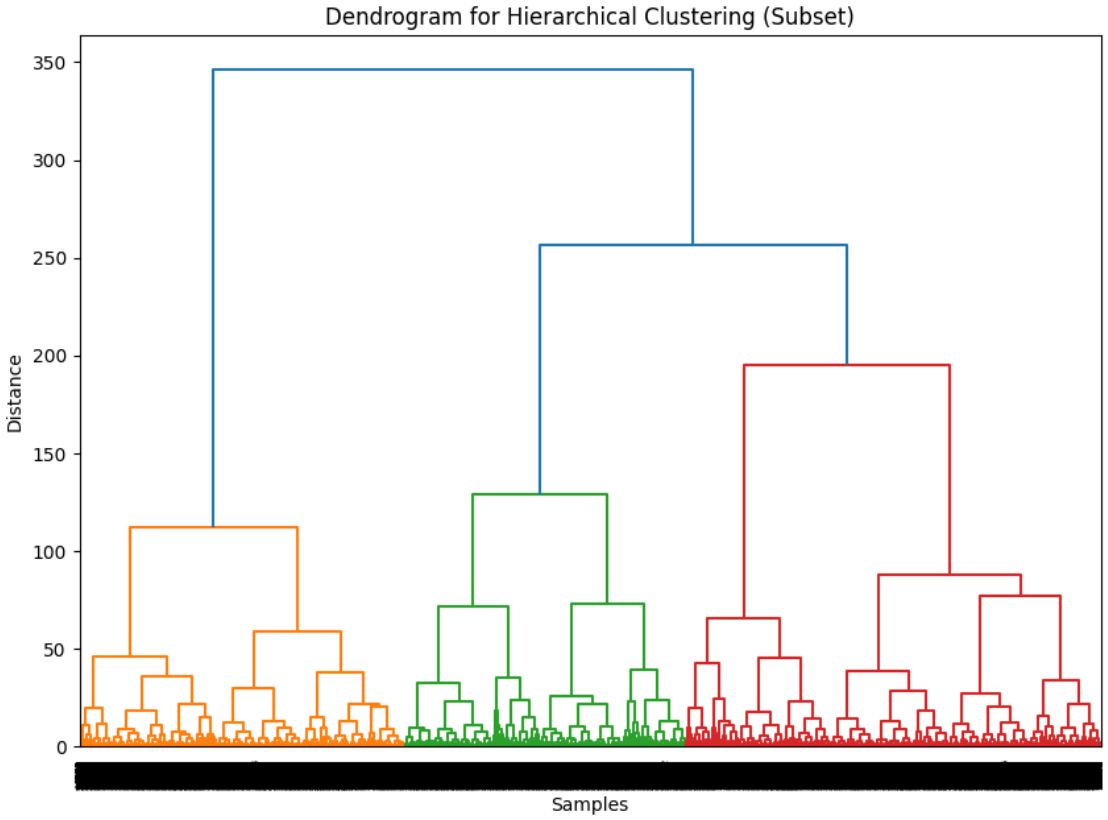


Figure 18: Hierarchical Clustering dendrogram with $n = 5$

6.3 Gaussian Mixture Model

The `Gaussian Mixture Model (GMM)` is a probabilistic clustering algorithm that models the data as a mixture of Gaussian distributions. Unlike K-Means, which uses hard assignments, GMM employs soft clustering, assigning probabilities to each data point for belonging to each cluster. This flexibility allows GMM to identify clusters with varying shapes, densities, and sizes, making it particularly advantageous for datasets where the underlying clusters may not be spherical or uniformly distributed.

GMM is a preferable choice for this dataset because it accounts for variations in the shapes and densities of clusters. Given the diversity of features in our dataset, such as continuous variables like height and weight and potentially overlapping features such as LDL and HDL cholesterol, GMM's ability to model elliptical clusters provides a more nuanced understanding of the data. Additionally, its probabilistic assignments offer insight into how well data points fit into specific clusters, which can be useful for analyzing ambiguous states. This makes GMM a robust alternative to the deterministic clustering of K-Means.

6.3.1 GMM with `n = 2`

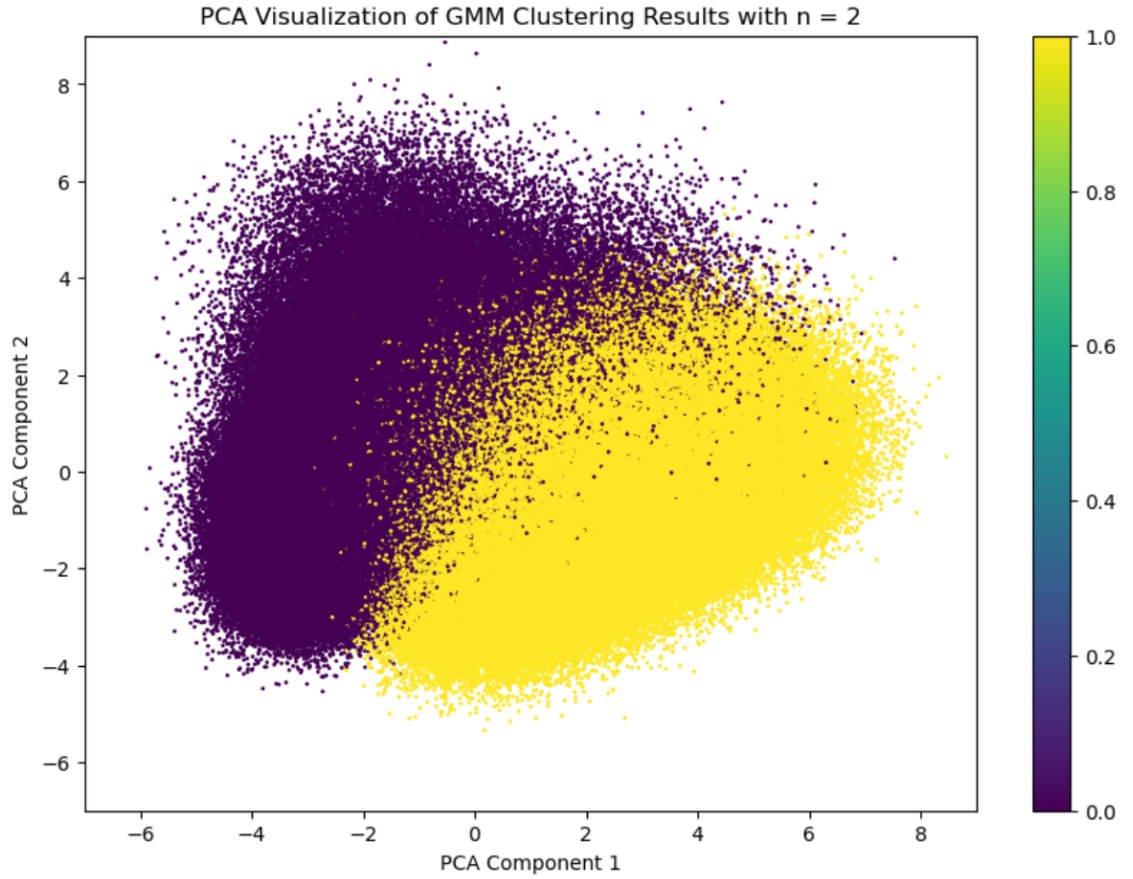
We started off with two clusters, i.e. `n = 2`. In Figure 19, the scatter plots of the dataset with PCA components are shown. In Figure 19b and 19c, only the males and females are plotted with the cluster labels. This allows us to see the ratio of males and females in the two clusters. The GMM clustering algorithm did a fairly good job of clustering with two clusters. We can see that all of the females are a part of the same cluster, whereas only a small part of the men are incorrectly grouped.

6.3.2 GMM with `n = 5`

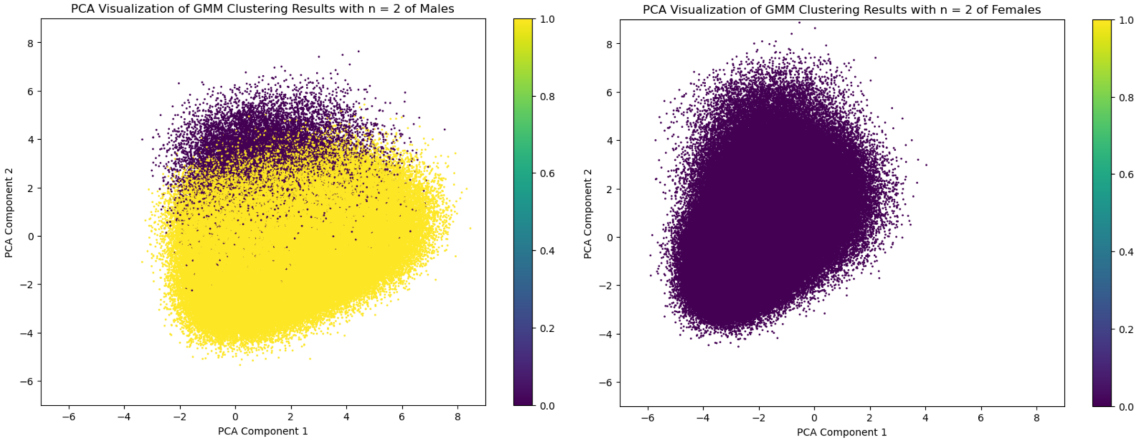
We set the number of clusters to `5` to compare with K-means with `k = 5`. Figure 20 shows the scatter plots with the `5` clusters in different colors. Figure 20b and 20c show the clusters of males and females, respectively. We can see that there are `3` main clusters: purple, green and yellow. Because mostly all of the data points are assigned to one of these 3 clusters, we could set `n = 3`.

We can see that all of the data points in the green cluster are females, and all of the data points in the yellow cluster are females. The data points in the purple cluster are divided approximately 50-50 between males and females.

Using "explainable" methods would help understand which common feature values lead to a third cluster with both males and females, but this is not something we intend to do in this assignment.



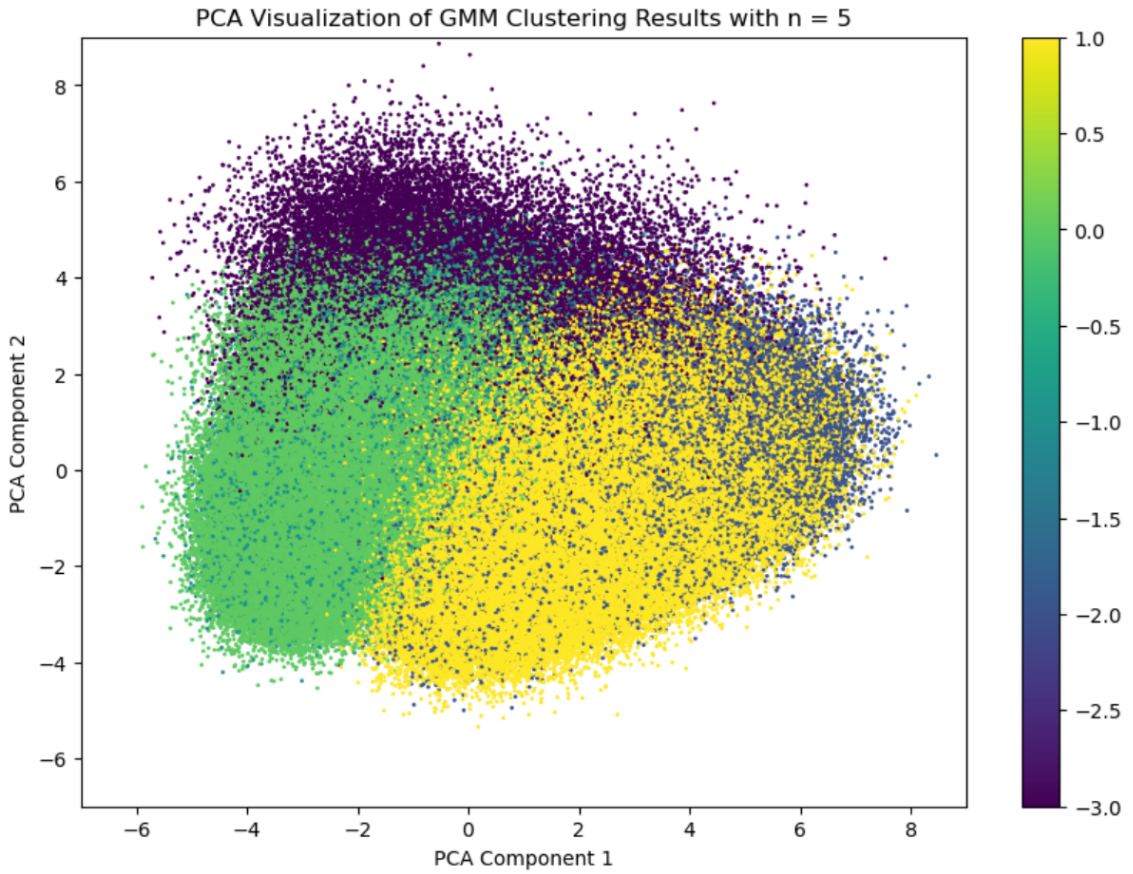
(a) GMM Clustering with $n = 2$



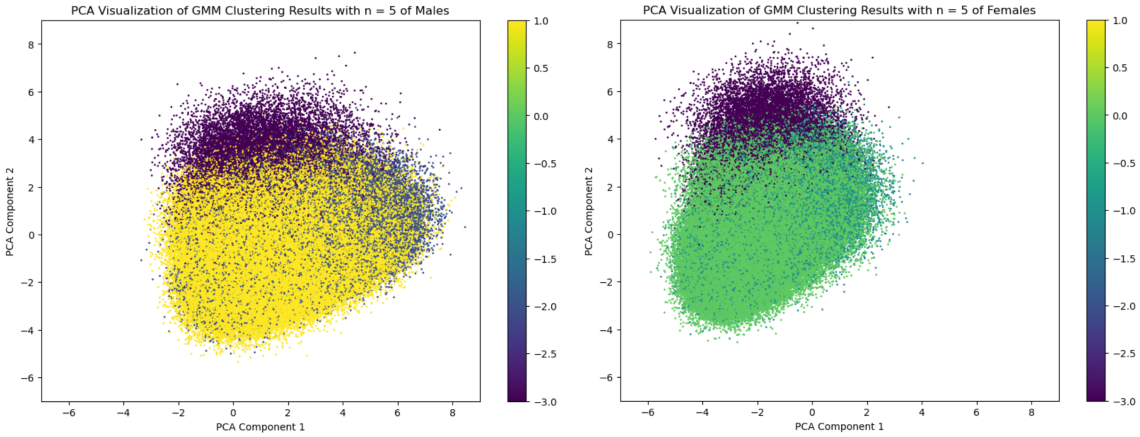
(b) GMM Clustering with $n = 2$ of Males

(c) GMM Clustering with $n = 2$ of Females

Figure 19: GMM Clustering Results with $n = 2$



(a) GMM Clustering with $n = 5$



(b) GMM Clustering with $n = 5$ of Males

(c) GMM Clustering with $n = 5$ of Females

Figure 20: GMM Clustering Results with $n = 5$

6.4 Comparison

We include three metrics commonly used for clustering in unsupervised learning to compare the three clustering algorithms that were experimented with. Figure 7 shows the results.

6.4.1 Metrics Explanation

Inertia: Measures the sum of squared distances between data points and their closest cluster centroids. A lower Inertia indicates tighter clusters, making it a useful metric for centroid-based algorithms like K-Means. While Inertia is specific to centroid-based algorithms like K-Means, the Silhouette and Calinski-Harabasz scores provide a standardized framework for evaluating all clustering methods. Thus, we only use this metric to compare the different k values of K-Means, not to compare with the other algorithms.

Silhouette Score: Evaluates cluster quality by comparing the average distance between a point and others in its cluster (cohesion) to the average distance between a point and points in the nearest cluster (separation). Scores range from -1 (poor clustering) to 1 (ideal clustering), where higher values indicate well-defined clusters. It is useful for comparing all algorithms but is sensitive to overlapping clusters.

Calinski-Harabasz Score: Also known as the Variance Ratio Criterion. It measures the ratio of between-cluster variance to within-cluster variance. A higher score suggests better-defined clusters with greater separation. This metric is algorithm-independent and works across all clustering methods.

Table 7: Clustering Performance Metrics for Different Algorithms

Algorithm	**Clusters (n)**	**Inertia**	**Silhouette Score**	**Calinski-Harabasz Score**
Hierarchical	2	–	0.3393	11800.98
Hierarchical	5	–	0.3230	14457.29
K-Means	2	180610.62	0.1694	2160.81
K-Means	5	145641.01	0.1229	1252.57
GMM	2	–	0.1599	1877.85
GMM	5	–	0.1003	832.71

6.4.2 Comparison of Metrics

Choosing parameters like the number of clusters (for example, `k` in K-Means or `n` in Hierarchical Clustering and GMM) or the number of iterations can greatly affect the results and shouldn't be seen as "correct" in all situations. For instance, using too many clusters may create meaningless splits, while too few can hide important differences in the data. Also, measures like the **Silhouette Score** and **Calinski-Harabasz Score** rely on assumptions about cluster shape and spacing that may not fully capture the real complexity of the dataset. Some metrics expect clusters to be roughly spherical or struggle when clusters overlap, which can lead to misleading conclusions about quality. Adjusting parameters is therefore a careful balance: increasing complexity can reveal subtle patterns but make results harder to interpret, while simpler settings risk oversimplifying the data. Factors like scaling, outliers, and noise can further reduce trust in these measures. Therefore, metrics are used as a guide, and no absolute truth.

Hierarchical Clustering:

For n = 2, the Silhouette Score (0.3393) and Calinski-Harabasz Score (11800.98) are among the highest of the methods, indicating well-defined clusters.

However, with n = 5, the Silhouette Score drops slightly and the Calinski-Harabasz Score increases slightly (Silhouette Score = 0.3230, Calinski-Harabasz Score = 14457.29). This indicates that increasing to n = 5 slightly improves cluster separation (higher Calinski-Harabasz) but reduces cohesion within clusters (lower Silhouette Score). This could happen because increasing the number of clusters to n = 5 forces some data points into separate groups that may not naturally align with the underlying structure of the data, leading to less compact clusters even though the overall separation improves.

K-Means:

For n = 2, the algorithm achieves an Inertia of 180610.62 and a Calinski-Harabasz Score of 2160.81. While the Silhouette Score (0.1694) is lower than that of Hierarchical Clustering, it still identifies reasonably distinct clusters. It also indicates that the clusters are reasonably distinct but not as cohesive as those formed by Hierarchical Clustering.

With n = 5, the Inertia reduces (145641.01), but the Silhouette Score (0.1229) and Calinski-Harabasz Score (1252.57) indicate weaker cluster definition. This highlights K-Means' difficulty in handling complex or overlapping data distributions. The reduced Inertia suggests tighter clusters.

Gaussian Mixture Model (GMM):

For n = 2, GMM's Silhouette Score is 0.1599 and Calinski-Harabasz Score is 1877.85. The Silhouette Score being low suggests that GMM's clusters have weaker cohesion and more overlap compared to more distinct clustering algorithms. The Calinski-Harabasz Score, while lower than K-Means, reflects GMM's ability to account for more complex and overlapping cluster structures. It supports the idea that GMM can handle overlapping clusters better than K-Means, but at the cost of sharp cluster boundaries.

For n = 5, the scores drop further (Silhouette Score = 0.1003, Calinski-Harabasz Score = 832.71). It suggests that the algorithm struggles to maintain clear separation and cohesion as the number of clusters increases. This is expected due to GMM's soft clustering approach, which allows overlap but reduces sharp boundaries.

6.4.3 Summary and Limitations

K-Means is efficient and scales well with large datasets, making it a practical choice for this task. However, its reliance on centroid-based clustering struggles with non-spherical or overlapping data.

GMM offers flexibility in capturing complex cluster shapes and overlapping data, but its scores reflect the trade-off between flexibility and cluster definition, particularly as the number of clusters increases. However, when visualizing the clusters for n = 2, it is clear that GMM actually captures the difference in females / males the best out of the three algorithms, and is able to distinguish them quite well.

Hierarchical Clustering performed best for n = 2, capturing well-separated clusters, and somewhat being able to distinguish males and females. Although it is able to do so, the large issue is that the model is trained on a subset of data, and was quite challenging to work with due to the computationally complexity.

For clear separations, GMM did best in separating the clusters, while Hierarchical Clustering and K-Means were able to somewhat capture the results but were more challenging to work with and get them right.

References

Aditya (2024). *Hierarchical Clustering: Applications, Advantages, and Disadvantages*. URL: https://codinginfinite.com/hierarchical-clustering-applications-advantages-and-disadvantages/ (visited on 5th Dec. 2024).

Brownlee, Jason (23rd May 2019). *How Much Training Data is Required for Machine Learning?* URL: https://machinelearningmastery.com/much-training-data-required-machine-learning/.

Bruin, J. (2024). *Principal Components (PCA) and Exploratory Factor Analysis (EFA) with SPSS*. URL: https://stats.oarc.ucla.edu/spss/seminars/efa-spss/ (visited on 26th Nov. 2024).

Frost, Jim (2024). *Principal Component Analysis Guide Example*. URL: https://statisticsbyjim.com/basics/principal-component-analysis/?utm_source=chatgpt.com (visited on 26th Nov. 2024).

Geeks, Geeks for (2024a). *Convolutional Neural Network (CNN) in Machine Learning*. URL: https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/ (visited on 20th Nov. 2024).

— (2024b). *Image classification using Support Vector Machine (SVM) in Python*. URL: https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/ (visited on 17th Nov. 2024).

— (2024c). *Support Vector Machine (SVM) Algorithm*. URL: hhttps://www.geeksforgeeks.org/support-vector-machine-algorithm/ (visited on 27th Nov. 2024).

Hallaj, Pouya (2024). *Data Augmentation: Benefits and Disadvantages*. URL: https://medium.com/@pouyahallaj/data-augmentation-benefits-and-disadvantages-38d8201aead (visited on 24th Nov. 2024).

Harsh, Patel (2024). *Normalization in Image Preprocessing: Scaling Pixel Values by 1/255*. URL: https://medium.com/@patelharsh7458/normalization-in-image-preprocessing-scaling-pixel-values-by-1-255-111b2fa496d4 (visited on 23rd Nov. 2024).

Huilgol, Purva (2024). *Top 4 Pre-Trained Models for Image Classification with Python Code*. URL: https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/#h-resnet50 (visited on 26th Nov. 2024).

Keldenich, Tom (2024). *Why and How to normalize data – Object detection on image in PyTorch Part 1*. URL: https://inside-machinelearning.com/en/why-and-how-to-normalize-data-object-detection-on-image-in-pytorch-part-1/?utm_source=chatgpt.com#Normalizing_data (visited on 25th Nov. 2024).

Kuchciak, Maciej (2024). *Hierarchical clustering – pros, cons, interpretation, application*. URL: https://rpubs.com/TusVasMit/HierarchicalClusteringOverwiev (visited on 5th Dec. 2024).

McDermott, James (2024). *Convolutional Neural Networks — Image Classification w. Keras*. URL: https://www.learndatasci.com/tutorials/convolutional-neural-networks-image-classification/?utm_source=chatgpt.com (visited on 24th Nov. 2024).

Patlolla, Chaitanya Reddy (2024). *Understanding the concept of Hierarchical clustering Technique*. URL: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec (visited on 5th Dec. 2024).

Riva, Martin (2024). *Batch Normalization in Convolutional Neural Networks*. URL: https://www.baeldung.com/cs/batch-normalization-cnn (visited on 20th Nov. 2024).

Sengupta, Joy (2024). *How to decide the hyperparameters in CNN*. URL: https://medium.com/@sengupta.joy4u/how-to-decide-the-hyperparameters-in-cnn-bfa37b608046 (visited on 20th Nov. 2024).

Singh, Aishwarya (2024). *A Comprehensive Guide to Ensemble Learning (with Python codes)*. URL: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/#h-catboost (visited on 26th Nov. 2024).

Soo.Y (2024). *Smoking and Drinking Dataset with body signal*. URL: https://www.kaggle.com/datasets/sooyoungher/smoking-drinking-dataset (visited on 6th Nov. 2024).

Swalin, Alvira (2024). *CatBoost vs. Light GBM vs. XGBoost*. URL: https://www.kdnuggets.com/2018/03/catboost-vs-light-gbm-vs-xgboost.html? (visited on 30th Nov. 2024).

Team, Great Learning Editorial (2024). *Label Encoding in Python – 2024*. URL: https://www.mygreatlearning.com/blog/label-encoding-in-python/?utm_source=chatgpt.com (visited on 26th Nov. 2024).

Varma, Aastha (2024). *Dimensionality Reduction: PCA, t-SNE, and UMAP*. URL: https://medium.com/@aastha.code/dimensionality-reduction-pca-t-sne-and-umap-41d499da2df2 (visited on 26th Nov. 2024).

Wadekar, Shakti (2024). *Hyperparameter Tuning in Keras: TensorFlow 2: With Keras Tuner: RandomSearch, Hyperband, BayesianOptimization*. URL: https://medium.com/swlh/hyperparameter-tuning-in-keras-tensorflow-2-with-keras-tuner-randomsearch-hyperband-3e212647778f (visited on 20th Nov. 2024).

Wikipedia (2024). *Graph neural network*. URL: https://en.wikipedia.org/wiki/Graph_neural_network (visited on 26th Nov. 2024).