CS-E4650 – Homework 1

STEFANO BOSOPPI
ID: 103370763
stefano.bosoppi@aalto.fi

BÁLINT JÁNOS PRÁGAI
ID: 103390468
balintjanos.pragai@aalto.fi

BENEDEK GABOR KISS
ID: 103574543
benedekgabor.kiss@aalto.fi

September 27, 2025

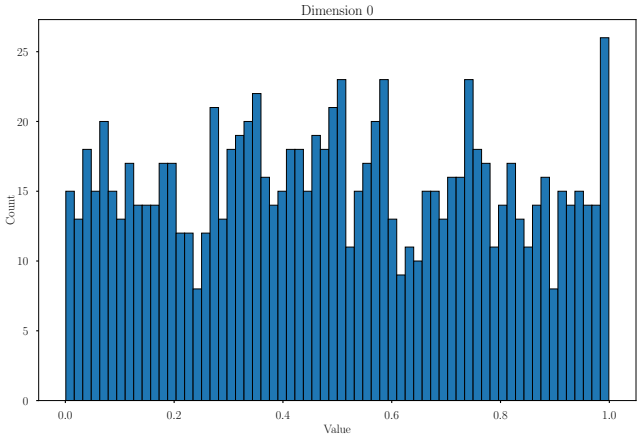
Contents

1 Methods

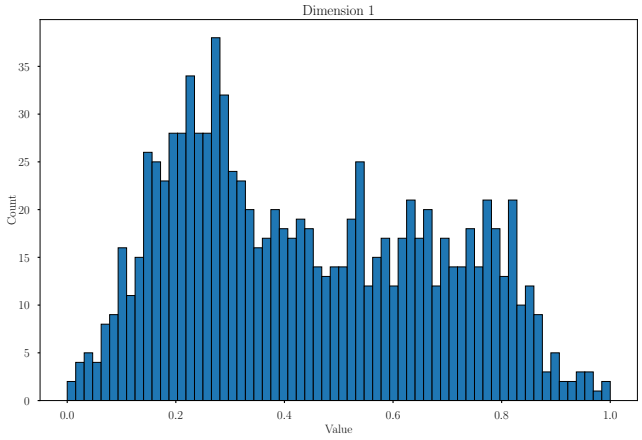
We used the Python programming language (ver. 3.11+). Libraries used: pandas, numpy, matplotlib.pyplot, Python standard libraries. Our code was developed with the use of the following tools: Jupyter notebooks, git, Visual Studio Code, and LaTeX for reporting. Additionally, we performed an extra experiment reported in Appendix A: plotting the pairwise distances between the datapoints in 1, 2, and 3 dimensions.

2 Results

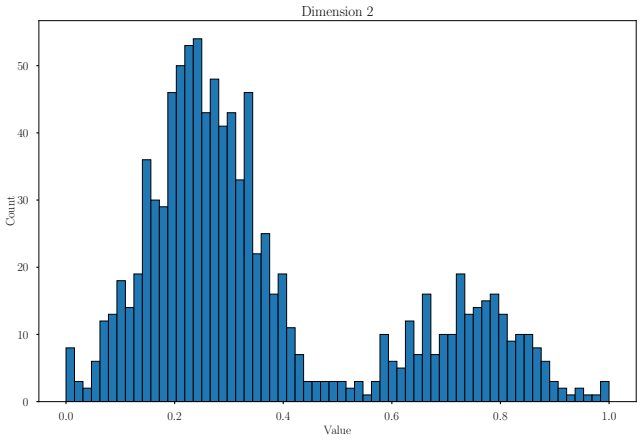
2.1 Histograms



(a) 1st component



(b) 2nd component



(c) 3rd component)

Figure 1: Histograms of the three data dimensions. The first dimension resembles a noisy uniform distribution, the second shows a single strong mode around 0.23, and the third exhibits bimodality, indicating possible clustering.

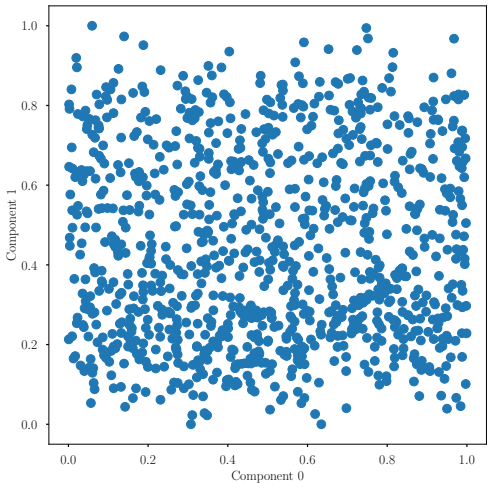
The three dimensions of the vectors provide three very different pictures of their distribution. The first component, in histogram 1.a, has a random distribution of values that resembles a uniform distribution with high noise. However, this provides no significant information about clustering tendencies since there are no clear distinctions in the frequency of values.

In histogram 1.b, the data spans values between 0 and 1, with most observations concentrated between approximately 0.1 and 0.85. The distribution shows a clear peak around 0.23, indicating that values in this range are more frequent. Beyond this peak, the distribution is irregular but relatively flat between 0.3 and 0.8, with moderate fluctuations. While the spike around 0.23 represents a local cluster of values, the overall distribution suggests a single dominant mode with variability elsewhere rather than distinct, separate clusters.

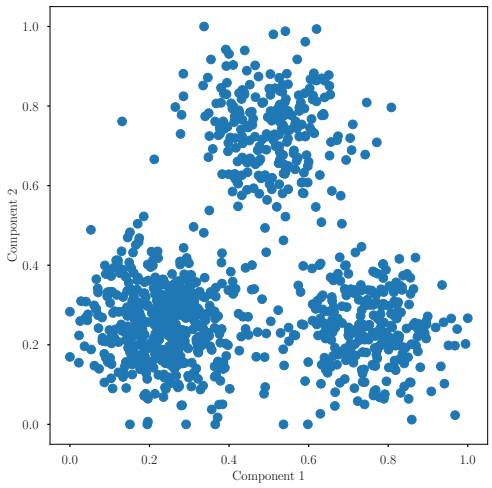
The third histogram, in Figure 1.c, clearly shows two peaks, one around 0.25 and the other around 0.75. The former is significantly denser than the latter. These two clear peaks in the distribution suggest two possible clusters of data, indicating a high clustering tendency in these two dimensions.

These observations are further reinforced by the pairwise distances between data points (see Appendix A).

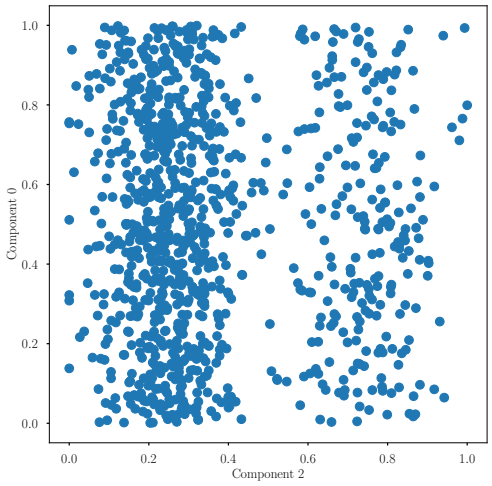
2.2 Scatterplots



(a) Dimensions 1 & 2



(b) Dimensions 2 & 3



(c) Dimensions 1 & 3

Figure 2: Two-dimensional scatterplots of pairwise component combinations. A clear clustering structure is visible in components $\{1, 2\}$, while component 0 appears uninformative.

Once again, plotting the pairwise distances between points was useful for discussing dependencies. This time, the two-dimensional analysis in Appendix A should be considered.

For the first two dimensions in Figure 2.a, both the scatterplot and the distance distribution in Figure 5.a show a pattern most akin to random data. There are no visible patterns or hotspots of data points, suggesting that these two components are independent and show no tendency to cluster.

The scatterplot in Figure 2.b shows three distinct clusters of points, suggesting a strong correlation or dependency between the two components. This is further supported by the distance distribution diagram 5.b, which hints at a high clustering tendency.

Figure 2.c shows the plots of the first and last components. Here, two groups of columnar data points are visible. The group on the left is dense, while the group on the right is less so, with only a few points between them. While this configuration could suggest a high clustering tendency, a more accurate analysis of the pairwise distances between these two groups, presented in Appendix A, shows that it is not as pronounced as the one analyzed in the previous paragraph.

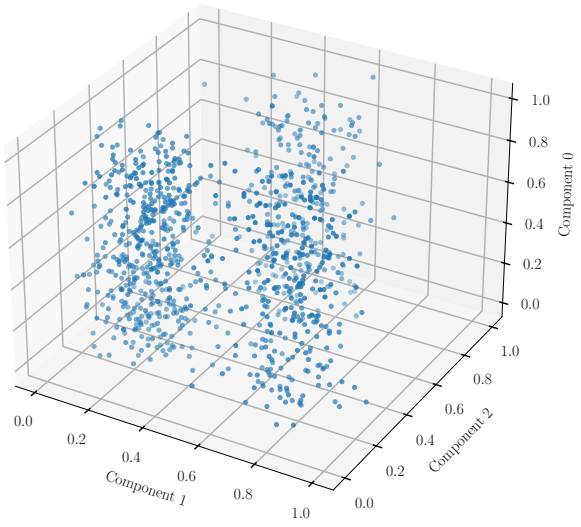


Figure 3: Three-dimensional scatterplot of the dataset. The clustering structure visible in lower dimensions is partly diluted by the inclusion of component 0.

2.3 Entropies

Feature set	Entropy
{0, 1, 2}	4.808
{0, 1}	4.978
{0, 2}	4.758
{1, 2}	4.393
{0}	5.010
{1}	4.968
{2}	4.703

Table 1: Entropy values (rounded to three decimals) for different feature subsets used in clustering. Lower entropy indicates stronger clustering structure, while higher entropy suggests weaker separation.

Table 1 shows the entropy values for the different feature sets, ranging from approximately 4.393 (for $\{1, 2\}$) to about 5.010 (for $\{0\}$). This gives a total width range of roughly 0.617. Despite its modesty, this difference suggests that the choice of feature set influences how well the data clusters.

The lowest entropies, as seen in the feature set $\{1, 2\}$, indicate the strongest clustering tendencies. Lower entropy indicates a more skewed or decisive distribution of cluster memberships, implying clearer separation when these feature combinations are used.

The highest entropies, such as in feature set $\{0\}$, suggest the weakest clustering tendencies. The distributions are more similar to uniform, meaning the data points are spread more evenly across clusters, and separation is less distinct.

Finally, intermediate values, such as those observed for $\{0, 1, 2\}$ and $\{2\}$, demonstrate that utilizing all features or just a single one does not necessarily result in the lowest entropy. This implies that adding or removing dimensions could introduce noise, overlook relationships between features, and create redundancy, which would dilute cluster separation.

2.4 Greedy backward selection

The execution of Code 8 resulted in the following prints:

1. `Starting with features [0, 1, 2] and entropy 4.8075123183976665`. This showed that we began by considering all the features and the associated entropy. Our objective was to drop the feature that caused the largest increase in entropy.

2. `Dropped feature 0, new features [1, 2], new entropy 4.392500622437077`. This is the result of comparing the entropies of all possible pairs of features. We chose to eliminate the feature that was not in the pair with the lowest entropy, *0* in this case.

3. `No further improvement, stopping.` This means that dropping any feature in {1, 2} would not decrease the entropy and, therefore, there is no point in continuing.

4. `Final selected features: [1, 2] with entropy 4.392500622437077`. This means that the optimal feature set for clustering, as selected by the algorithm, is {1, 2}.

2.5 Greedy forward selection

Running Code 9 resulted in the following prints:

1. `Starting with features [] and entropy inf`. We started from an empty feature set, looking for the feature that caused the largest decrease in entropy.

2. `Added feature 2, new features [2], new entropy 4.702932411963032`. This implied that out of all the entropies of the individual features, *2*'s was the lowest one.

3. `Added feature 1, new features [2, 1], new entropy 4.392500622437077`. Similarly to what happened above, out of all the pairs in $\{\{2,x\} : x \in \{0,1\}\}$, {2, 1} had the lowest entropy.

4. `No further improvement, stopping.` Since considering all the features would not decrease the entropy, we stopped.

5. `Final selected features: [2, 1] with entropy 4.392500622437077`. Which means that {1, 2} was identified as the minimal feature set for optimal clustering.

3 Analysis

The diverse methodologies employed, encompassing visual inspection, entropy calculations, and greedy feature selection, culminate in a cogent depiction of the dataset's clustering tendencies.

The first dimension appears largely uninformative from the visual analysis. Its histogram resembles a noisy uniform distribution, and its pairwise distances do not indicate the presence of any strong concentration of points. In contrast, the second and third dimensions display patterns that split the data into distinct groups. The third dimension, in particular, shows two clear modes, hinting at a clustering tendency even when considered in isolation.

The entropy calculations quantitatively confirm these observations. Entropy values were lowest when dimensions one and two, components two and three, were considered together. However, the inclusion of dimension zero consistently

increased entropy. This suggests that dimension 0 primarily contributes noise rather than useful clustering information, while dimensions 1 and 2 contain the strongest signal for cluster separation.

Finally, both the greedy backward and forward selection strategies converged on the same solution: the optimal feature subset for clustering is {1, 2}. The consistency across all strategies solidifies {1, 2} as the feature set with the highest clustering tendency.

3.1 Why 64 bins?

As 64 is both a perfect cube and a perfect square. This is the smallest 2-digit number for which this is true. The next 2 possibilities would be 729 and 4096. In case of higher-dimensional data, finding these kinds of perfect numbers becomes harder and harder, and they become larger and larger. In the case of 4-dimensional data, the two smallest numbers would be 256 and 65536.

4 Conclusions

Collectively, the results point to the conclusion that the dataset exhibits a moderate but detectable clustering tendency, primarily concentrated in the second and third dimensions. Dimension 0 contributes little to the separation structure and appears to be noise. Both entropy analysis and greedy feature selection identified {1, 2} as the most informative feature subset. This aligns with visual observations of multi-modality in the third dimension and a correlated structure between the second and third dimensions.

Overall, the findings suggest that clustering algorithms would benefit from focusing on the combination of dimensions 1 and 2, while down-weighting or excluding dimension 0. While the evidence does not establish perfectly distinct clusters, it strongly suggests that the data has an underlying structure that could be exploited in further analyses.

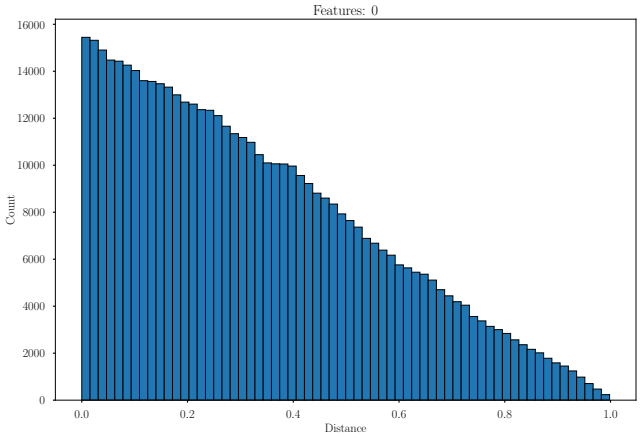
A Extra experiments

To complement the histogram and scatterplot analyses, we examined the distance distributions of each dimension individually, as well as all pairs and the full three-dimensional dataset.

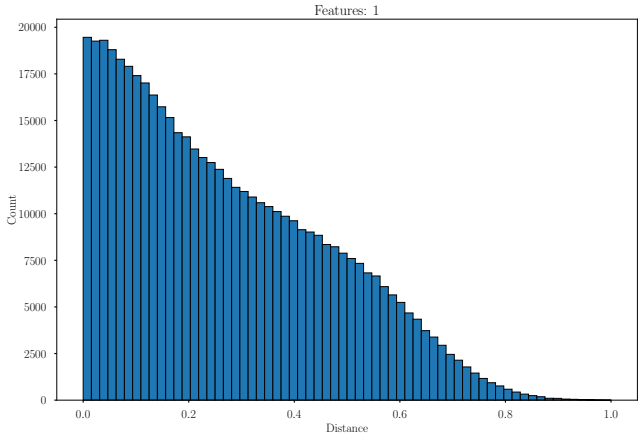
For the individual dimensions, depicted in Figure 4, the first and second components produced distance distributions similar to those of uniformly distributed data, which confirms their weak clustering tendency. However, the third dimension exhibited a higher frequency of short pairwise distances, which is consistent with the presence of a dense region of points and suggests a clustering tendency.

When moving to two-dimensional projections, Figure 5 shows clearer patterns emerging. The pair {1, 2} produced a distance distribution with two sharp peaks, a strong indicator of clustering structure. In contrast, pair {0, 1} showed no evidence of clustering, while pair {0, 2} exhibited some irregularity, though not as pronounced as pair {1, 2}.

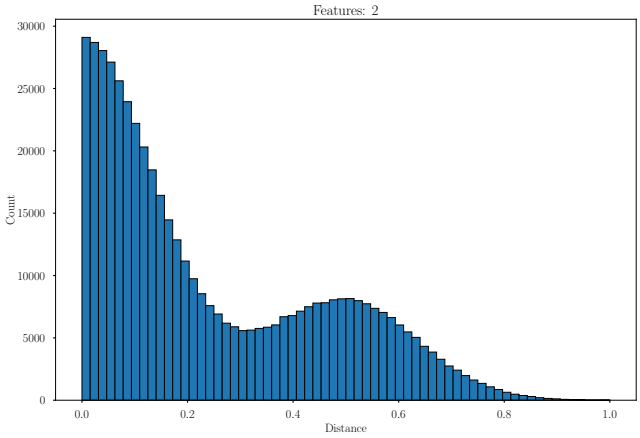
In the three-dimensional case, shown in Figure 6, the distance distribution was influenced by the noisy first component, diluting but not erasing the evidence of clustering. This further supports the conclusion that dimensions 1 and 2 contain the most meaningful structure.



(a) 1st component

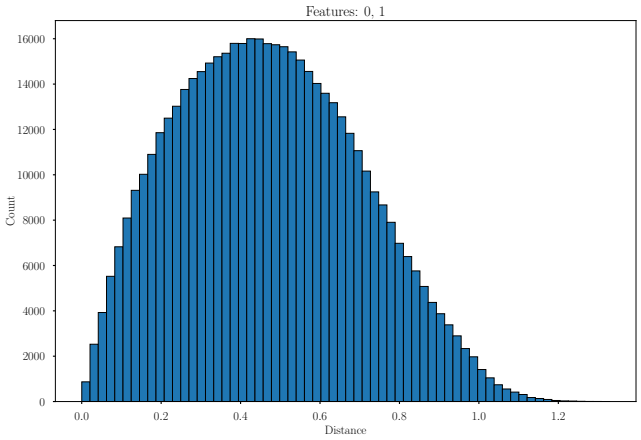


(b) 2nd component

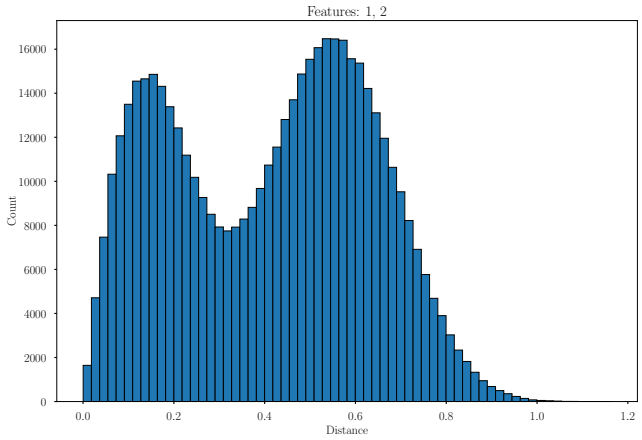


(c) 3rd component)

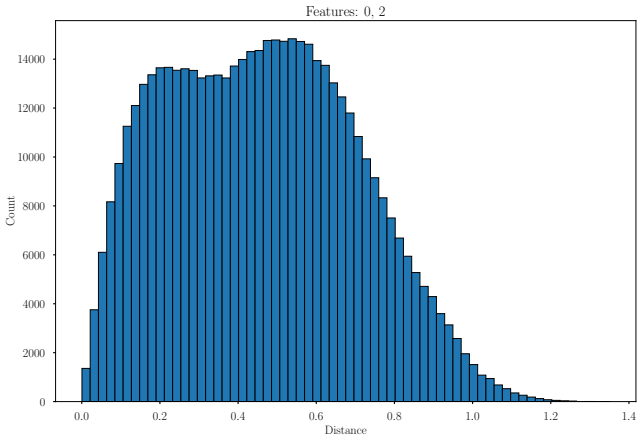
Figure 4: Pairwise distance distributions for individual dimensions. The third component shows evidence of clustering through an excess of short distances, while dimensions 0 and 1 resemble uniform noise.



(a) Dimensions 1 & 2



(b) Dimensions 2 & 3



(c) Dimensions 1 & 3

Figure 5: Pairwise distance distributions for two-dimensional subsets. Dimensions {1, 2} exhibit multiple sharp peaks, indicating strong clustering, while {0, 1} and {0, 2} show no or little signs of clustering tendency.

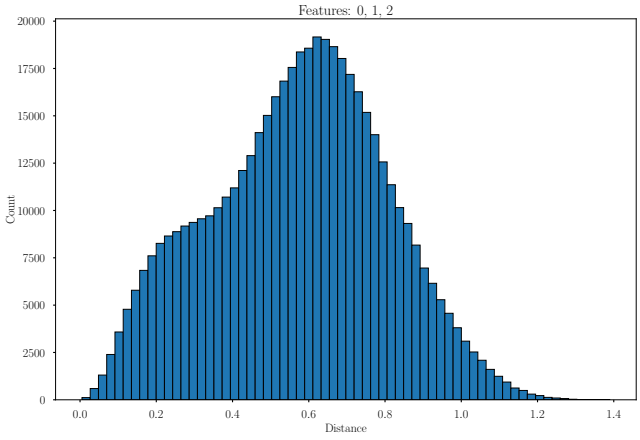


Figure 6: Pairwise distance distribution for all three dimensions combined. The inclusion of component 0 increases noise and reduces the clarity of the clustering signal.

B Code

Source Code 1: Imported libraries, plot setup, and dataset loading.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from itertools import chain, combinations
from scipy.spatial.distance import pdist
from math import floor

matplotlib.use("pgf")
plt.style.use('seaborn-v0_8-poster')
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})
img_folder = "images/"

df = pd.read_csv(".\clustering-tendency.csv", header =None)
```

Source Code 2: Code for task 1.4.2.

```
for i in range(0,3):
    plt.hist(
        df[i],
        bins = 64,
        edgecolor='black',
        histtype='bar'
        )
    plt.title(f'Dimension {i}')
    plt.xlabel(f"Value")
    plt.ylabel(f"Count")
    plt.tight_layout()
    plt.show()
    plt.savefig(f'{img_folder}hist_1dim_{i}.pdf', bbox_inches='tight')
    plt.clf()
```

Source Code 3: Code for task 1.4.3.

```
# 1- and 2-dimensional data
for i in range(0,3):
    plt.scatter(
        df[i],
        df[(i+1)%3],
    )
    plt.xlabel(f"Component {i}")
    plt.ylabel(f"Component {(i+1)%3}")
    plt.gca().set_aspect('equal')
```

```
    plt.tight_layout()
    plt.show()
    plt.savefig(f'{img_folder}scatter_2dim_{i}{(i+1)%3}.pdf', bbox_inches='tight')
    plt.clf()

# 3-dimensional data
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

ax.scatter(df[1], df[2], df[0])
ax.set_xlabel('Component 1', labelpad=20)
ax.set_ylabel('Component 2', labelpad=20)
ax.set_zlabel('Component 0', labelpad=20)
plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=1.0)
plt.show()
plt.savefig(f'{img_folder}scatter_3dim.pdf', bbox_inches='tight', pad_inches=0.5)
plt.clf()
```

Source Code 4: Code used to analyze the distance distribution in the dataset for task 1.4.4.

```
def powerset(iterable):
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))

for feature_set in powerset([0, 1, 2]):
    if len(feature_set) == 0:
        continue

    df_subset = df[list(feature_set)]
    dists = pdist(df_subset, metric='euclidean')

    plt.hist(dists, bins=64, edgecolor="black", histtype="bar")
    plt.title(f"Features: {set(feature_set)}")
    plt.xlabel("Distance")
    plt.ylabel("Count")
    plt.tight_layout()
    plt.show()
    plt.savefig(f'{img_folder}hist_{len(feature_set)}dim_dist{"".join(map(str, feature_set))}.pdf', bbox_inches='tight')
    plt.clf()
```

Source Code 5: Code for task 1.4.5.

```
def categorize(data_point):
    return (floor(data_point[0] / 0.25)) * 16 + (floor(data_point[1] / 0.25)) * 4 + (floor(data_point[2] / 0.25))

categories_3d =  df.apply(categorize, axis="columns")
```

```
rel_prob_3d:pd.Series = categories_3d.value_counts(normalize=True)

entropies = {tuple(sorted(feature_set)): -1 for feature_set in
 ↪  powerset([0,1,2])}

def calc_entropy(probs: pd.Series):
    sum = 0
    for i in range(0, 63):
        value = probs.get(i, default=0)
        if 0 < value < 1:
            sum += (value * log(value) +
                    (1 - value) * log(1 - value))
    return sum * -1

entropies[(0,1,2)] = calc_entropy(rel_prob_3d)
```

Source Code 6: Code for task 1.4.6.

```
def calculate_2d(data_point: pd.Series):
    return (floor(data_point.values[0] / 0.125)) * 8 +
     ↪  (floor(data_point.values[1] / 0.125))

for pair in itertools.combinations(df.columns, r=2 ):
    data = df[list(pair)]
    bined_data = data.apply(calculate_2d, axis=1).value_counts(normalize=True)
    entropies[tuple(sorted(pair))] = calc_entropy(bined_data)
```

Source Code 7: Code for task 1.4.7.

```
def calculate_1d(data_point: float, bucket_number: int = 64):
    return floor(data_point / (1/bucket_number))

for column in df.columns:
    binned_data = df[column].apply(calculate_1d)
    count_data = binned_data.value_counts(normalize=True)
    entropies[tuple(column)] = calc_entropy(count_data)
```

Source Code 8: Implementation of the greedy backward selection algorithm.

```
def backward_selection(entropies, features):
    current_features = features.copy()
    current_entropy = entropies[tuple(sorted(current_features))]
    print(f"Starting with features {current_features} and entropy
     ↪  {current_entropy}")

    while len(current_features) > 1:
        entropies_if_dropped = {}
        for feature in current_features:
            reduced_features = tuple(sorted(f for f in current_features if f !=
             ↪  feature))
            entropies_if_dropped[feature] = entropies[reduced_features]
```

```
        feature_to_drop = min(entropies_if_dropped,
        ↪  key=entropies_if_dropped.get)
        new_entropy = entropies_if_dropped[feature_to_drop]

        if new_entropy < current_entropy:
            current_features.remove(feature_to_drop)
            current_entropy = new_entropy
            print(f"Dropped feature {feature_to_drop}, new features
            ↪  {current_features}, new entropy {current_entropy}")
        else:
            print("No further improvement, stopping.")
            break

    return current_features, current_entropy

final_features, final_entropy = backward_selection(entropies, [0, 1, 2])
print(f"Final selected features: {final_features} with entropy {final_entropy}")
```

Source Code 9: Implementation of the greedy forward selection algorithm.

```
def forward_selection(entropies, features):
    starting_features = []
    current_entropy = float("inf")
    print(f"Starting with features {current_features} and entropy
    ↪  {current_entropy}")

    while len(starting_features) < len(features):
        entropies_if_added = {}
        for feature in features:
            if feature not in starting_features:
                new_features = tuple(sorted(starting_features + [feature]))
                entropies_if_added[feature] = entropies[new_features]

        feature_to_add = min(entropies_if_added, key=entropies_if_added.get)
        new_entropy = entropies_if_added[feature_to_add]

        if new_entropy < current_entropy:
            starting_features.append(feature_to_add)
            current_entropy = new_entropy
            print(f"Added feature {feature_to_add}, new features
            ↪  {starting_features}, new entropy {current_entropy}")
        else:
            print("No further improvement, stopping.")
            break

    return starting_features, current_entropy

```

```
final_features_forward, final_entropy_forward = forward_selection(entropies, [0,
    1, 2])
print(f"Final selected features: {final_features_forward} with entropy
    {final_entropy_forward}")
```