Introduction to Machine Learning
Assignment 4
Unsupervised Vector Quantization

Group 05
Bora Yilmaz (s3903125) & Felix Zailskas (s3918270)

October 13, 2021

Contents

Unsupervised Vector Quantization

1. Introduction

In this report, the Winner-Takes-All unsupervised competitive learning vector quantization is implemented, and the results discussed. In contrast to supervised vector quantization, this learning model is not supervised because classification labels are not present for the data space. Thus, at any point in learning, the prototypes are not supervised based on any classification or other label information. For a given data point, the winning prototype will be the closest prototype (under Euclidean distance). However, the update step will not refer to classification of data, and it will always move the winning prototype closer to the data point. How close it moves will again be based on the learning rate.
It will be seen how these ideas allow this model make the prototypes position themselves based on how the data is clustered. Also the effect of the number of training epochs, number of prototypes and learning rate on the results and the learning model is discussed in section 4.

2. Method

For analysing the unsupervised vector quantization we first initialize the prototypes we need for the current iteration. For a defined amount of epochs t_{max} we check, for each data point, which prototype is the closest. That prototype will then be moved closer to that data point. This update step is defined as:

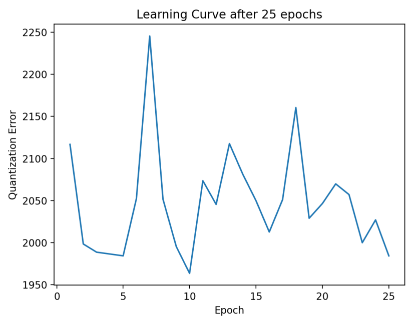
$$\mathbf{p}^i = \mathbf{p}^i + \eta(d^j - \mathbf{p}^i) \tag{1}$$

Where $\mathbf{p}^i$ is the closest prototype to the current data point d^j.
After updating the prototypes according to this procedure the quantization error is recorded. This error is the sum of squared distances between each data point and their closest prototype.
Now the amount of prototypes can be increased and the process is repeated until all amounts of prototypes of interest have been evaluated.

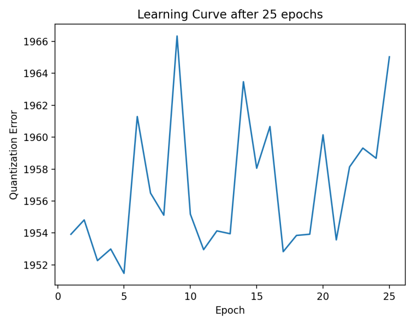
2.1. Parameter Selection

To select a good value of the learning rate η we ran the algorithm with $K = 2$ and $K = 4$ for four values of η, namely 0.1, 0.01, 0.004 and 0.001. Additionally, we wanted to determine a reasonable value for t_{max}. For this section of parameter selection we used a value of $t_{max} = 25$. In the graph of the learning curve we tried to find a smooth decrease of error for increasing epochs and a sharp elbow to determine a good value for η. The location of the elbow can give us an insight of a good value for t_{max}.

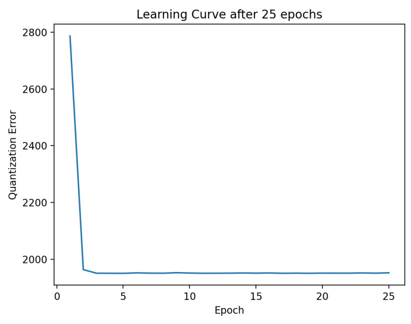
For $K = 2$ we could observe the following learning curves with $\eta = 0.1$, $\eta = 0.01$, $\eta = 0.004$, $\eta = 0.001$ respectively.



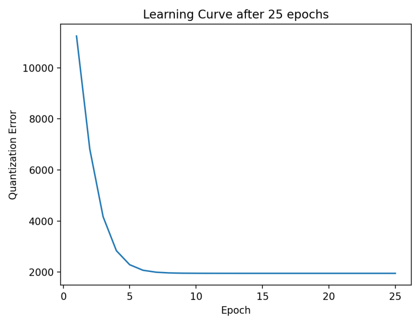
(a) *Learning curve for $K = 2$ and $\eta = 0.1$.*



(b) *Learning curve for $K = 2$ and $\eta = 0.01$.*



(c) *Learning curve for $K = 2$ and $\eta = 0.004$.*

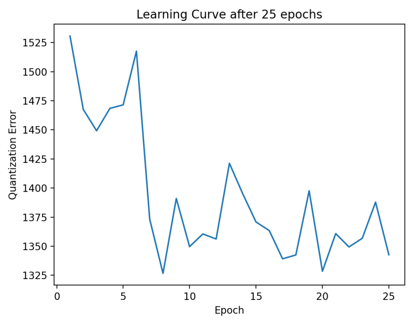


(d) *Learning curve for $K = 2$ and $\eta = 0.001$.*

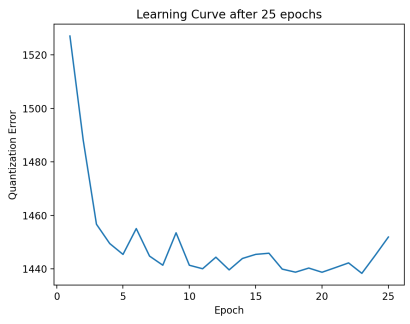
Figure 1: *Learning curve for $K = 2$ and different values of η.*

From Figure 1 we can see that for $\eta = 0.1$ and $\eta = 0.01$ the curve is not smooth at all. There are big spikes in the quantization error with increasing epochs. This indicates that the learning rate is too large. For both $\eta = 0.004$ and $\eta = 0.001$ the learning curve is smooth and constantly decreasing. The elbow of the curve in Figure 1c and Figure 1d is reached after roughly 5 epochs.

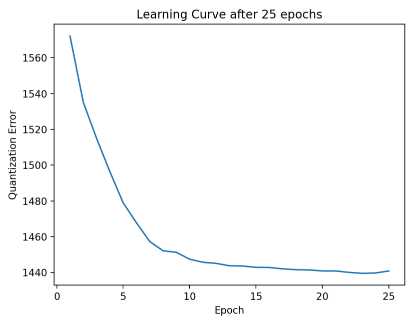
For $K = 4$ we could observe the following learning curves with $\eta = 0.1$, $\eta = 0.01$, $\eta = 0.004$, $\eta = 0.001$ respectively.



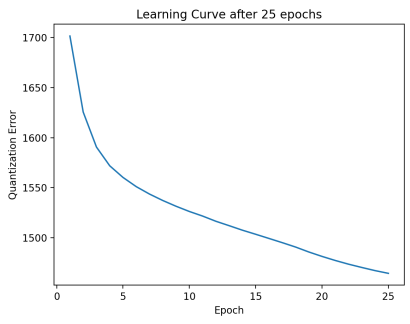
(a) *Learning curve for $K = 4$ and $\eta = 0.1$.*



(b) *Learning curve for $K = 4$ and $\eta = 0.01$.*



(c) *Learning curve for $K = 4$ and $\eta = 0.004$.*



(d) *Learning curve for $K = 4$ and $\eta = 0.001$.*

Figure 2: *Learning curve for $K = 4$ and different values of η.*

From Figure 2 we can see that for $\eta = 0.1$ and $\eta = 0.01$ the curve is not smooth at all. There are spikes in the quantization error with increasing epochs. This indicates that the learning rate is too large. For both $\eta = 0.004$ and $\eta = 0.001$ the learning curve is smooth and constantly decreasing. However, in Figure 2d we can see that the elbow of the curve is not very steep. A large decrease in quantization error can be observed after epoch 5. In Figure 2c the elbow is more steep. After roughly epoch 10 the decrease in quantization error is relatively low.

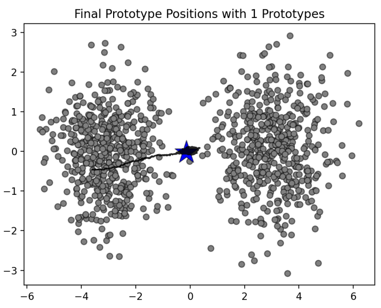
As we have found a smooth, decreasing learning curve with a sharp elbow for both $K = 2$ and $K = 4$ at $\eta = 0.004$, we have decided to use this value as the learning rate throughout this analysis. Furthermore, we chose a value of $t_{max} = 15$, since this value is slightly larger than the epoch, where the elbow of the curves lie. This parameter selection should result is a good trade off between accuracy of results and computation complexity.

3. Results

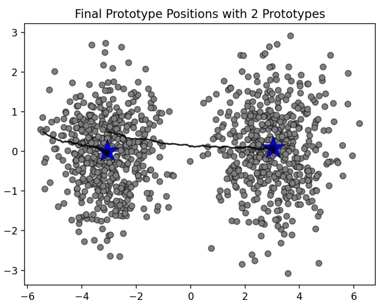
3.1. Amount of Prototypes K

For our analysis we have investigated the values $K = 1$, $K = 2$, $K = 3$, $K = 4$, $K = 5$. We will present both the quantization error as a function of K as well as the final positions and traces of the prototypes for each K. These results will be presented and discussed for two different methods of initializing the prototypes.
First the prototypes will be initialized at a random position of a data point in the data set.
Second the prototypes will be initialized at a position that we know is far from all data points, this will be referred to as subpar initialization.

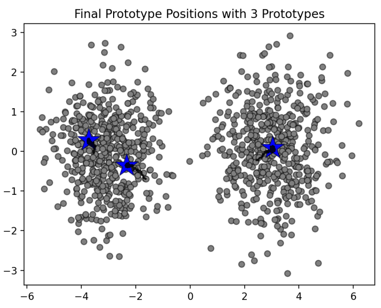
3.1.1. Random Initialization



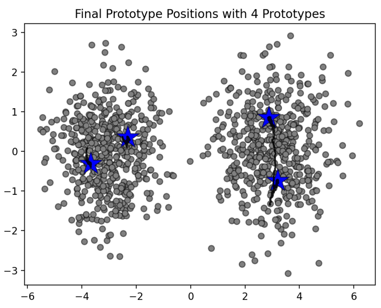
(a) *Final prototype position with 1 prototype.*



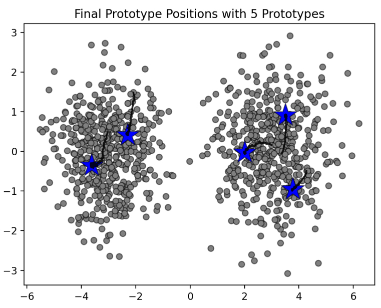
(b) *Final prototype position with 2 prototypes.*



(c) *Final prototype position with 3 prototypes.*



(d) *Final prototype position with 4 prototypes.*



(e) *Final prototype position with 5 prototypes.*

Figure 3: *Final prototype positions for different K, with random initialization.*

In Figure 3 we can see that for 1 prototype the prototype finalizes its position in between the two clusters of data points. For all higher values of K the prototypes finalize their positions within the two clusters.

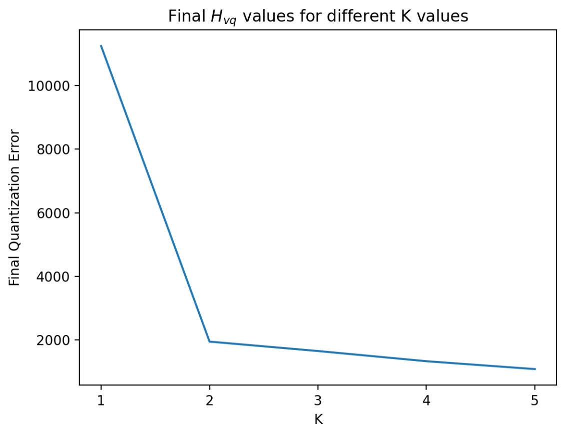
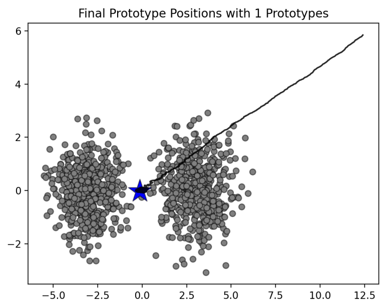


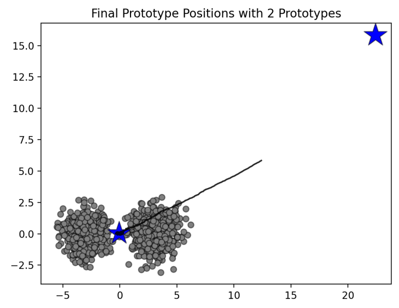
Figure 4: *Final Quantization Error for different K, with random initialization.*

In Figure 4 we can see that there is a steep decrease in quantization error from one prototype to two prototypes. The elbow of the curve is also observed at $K = 2$. This is because the decrease in quantization error is significantly lower for increasing K to a value higher than 2.

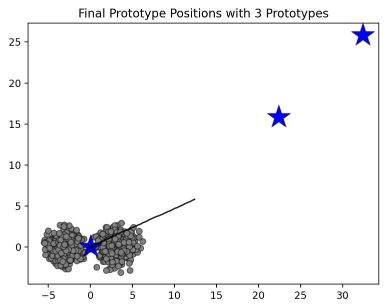
3.1.2. Subpar Initialization



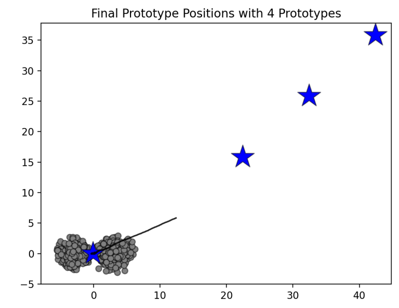
(a) *Final prototype position with 1 prototype.*



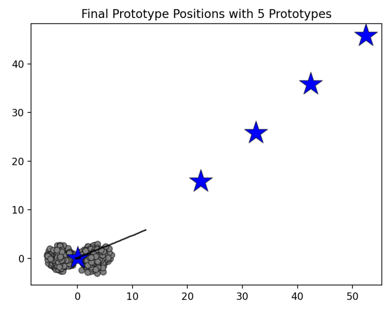
(b) *Final prototype position with 2 prototypes.*



(c) *Final prototype position with 3 prototypes.*



(d) *Final prototype position with 4 prototypes.*



(e) *Final prototype position with 5 prototypes.*

Figure 5: *Final prototype positions for different K, with subpar initialization.*

In Figure 5 we can see that for all values of K exactly one prototype finalizes its position in between the two clusters of data points, as in Figure 3a, while the other prototypes do not move at all.

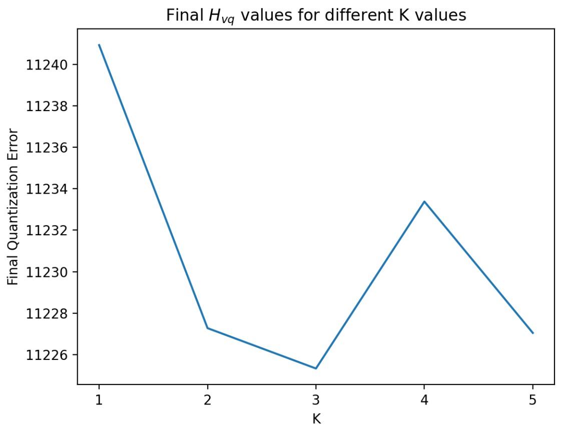


Figure 6: *Final Quantization Error for different K, with subpar initialization.*

In Figure 6 we can see that the quantization error does fluctuate for different values of K. However, the maximum difference between two errors ($K = 1$ and $K = 3$) is only 14. This is not a significant difference in quantization error compared with the values that can be seen in Figure 4.

4. Discussion

4.1. Learning Rate η

In Equation 1, the learning rate η is a constant multiplier, and it moderates the intensity of the update step. With a low learning rate, the prototypes will move less towards the data points, and with a large learning rate, the prototypes will take move further towards the data points.
As discussed in section 2, different learning rates were tried to observe how the learning curve changes as epochs go on. A good value for η had to be found before moving on to running the algorithm for different K values. Looking at Figure 1 and Figure 2, we can discuss again how the learning rate affects the learning curve of prototypes. With a large learning rate of $\eta = 0.1$, the error fluctuates a lot and a constant smooth decrease is not observed between the epochs. By the first lowering to $\eta = 0.01$, a much better graph is observed in Figure 2b, with a constant decrease visible, but still there are fluctuations with a lower K in Figure 1b. When $\eta = 0.004$, a smoothly decreasing learning curve is observed in Figure 1c and Figure 2c. and the elbow of the curve is very clear at $t_{max} = 5$ and $t_{max} = 10$ respectively. The team decided to use this value for the implementation because of this learning curve. With even smaller learning rate, $\eta = 0.001$, the curves elbow is not steep, and the error does not decrease fast enough compared to other values.

4.2. Random Initialization and K

As seen in section 3, the data space along with the traces and final positions of the prototypes are displayed for each K. With random initialization, a random already existing data point is selected for each prototype and then the prototype initialized at that data point's position. A good and expected result is that the prototypes are positioned well with respect to the data clusters. The observer knows beforehand that there are two clusters of data and there is a split between the two clusters in the middle at $x = 0$. So to evaluate how the prototypes performed is can be done simply by looking at the scatter plots. If all prototypes are positioned so that they represent the data clusters equally, then it means that they are positioned well.

4.2.1. $K = 1$

As seen in Figure 3a, the single prototype positioned itself right in the middle of the two clusters. If it was to be a part of one of the clusters or lean towards either one, then it would mean that this prototype is not able to represent and make up for the further cluster. But that is not the case, and the prototypes achieves the best position it could possibly achieve. The quantization error is the highest in this one but that is expected since there is only one prototype.

4.2.2. $K = 2$

Figure 3b shows how the 2 prototypes positioned themselves right in the middle of the two clusters. Since there are two clusters, a randomly picked point is going to be either from the left or right cluster, which will then pull the closer prototype towards it. And since there are 2 prototypes total, the clusters share them perfectly since $2 mod 2 = 0$. This will not be the case when $K mod 2 = 1$, so for $K = 3, 5$ one cluster will hold one more prototype in itself.

4.2.3. $K = 3$, $K = 4$ and $K = 5$

Figure 3c, Figure 3d and Figure 3e shows how the prototypes keep joining either cluster. There are only 2 clusters, so increasing the amount of prototypes is not necessary. Even though it is not detrimental, this leads to excess amount of prototypes representing the same cluster. All prototypes keep moving towards the center of the clusters. Overall, $K = 2$ proves the best, according to the elbow method heuristic. Looking at Figure 4, the elbow appears clearly at $K = 2$, then without knowing the scatter plot showing the clusters, this allows a good guess at the number of clusters, which is 2. For all $K \geq 2$, the prototypes positioned themselves inside one of the clusters, so no prototype goes into vain and updates correctly. After the elbow method and observing the scatter plot, it is clear that there are 2 obvious clusters, so $K = 2$ is enough to represent the rough shape of the data.

4.3. Subpar Initialization and K

With Subpar initialization, as expected, the prototype that is initialized closest to the data will get pulled towards the middle of the clusters. All other distant prototypes will not update at all since they can never win and get updated closer to any data (in this only one-winner implementation). So then the closest prototype is always the closest to all data. For all K, the result is the same with the closest prototype positioning in the middle of the clusters. The error does not change much because the one prototype moving is always the closest to all data. Essentially, the comparison is between multiple models using a single prototype. This way of initialization should be avoided because it has this unavoidable issue of ignoring further prototypes, and also wastes training epochs.

4.4. Final H_{vq} Values

4.4.1. Random Initialization

In Figure 4, the elbow occurs at $K = 2$ because of the immense decrease in error from $K = 1$ to $K = 2$, and then the continuous decrease for $K \geq 2$. The error then keeps decreasing as K increases, because there are more prototypes, which allows for less overall distance between prototypes and data points, but it does not necessarily mean a better representation of the data, since $K = 2$ is already enough as already mentioned.

4.4.2. Subpar Initialization

In Figure 6 it is seen from the $Y - axis$ values that the overall error change is very low. This is because only one of the prototypes moves for any K, while $K - 1$ prototypes do not move ever. Thus, the elbow does not appear clearly at $K = 2$, because for $K \geq 2$, the error fluctuates a bit, while it should have kept decreasing constantly. As mentioned, this subpar initialization is like running multiple models with only one active prototype, so it's results are not really concluding, leading to the conclusion that this should be avoided.

5. Contribution

Both team members attended the lab session for this assignment and both agree that the contribution to this assignment was equal and fair.

5.1. Code

The development of the code for the assignment was done in a lab session. Therefore, the contribution to the code base was entirely equal for both group members.

5.2. Report

Sections were split in half and thus the report was worked on by both members. After looking over the report together one last time, it was finalized and ready.

6. Code Appendix

6.1. vq.py

```
import random
import numpy as np
import pandas as pd
from plotting import *
from calculation import *
from matplotlib import pyplot as plt


# Reading in the Data File
df = pd.read_csv("data/simplevqdata.csv", sep=",")

# Determining the size of the input vectors and the amount of vectors
amt_points, input_dim = df.shape

# Setting learning variables
learning_rate = 0.004
t_max = 15
amt_protos = 1  # adjust for different prototype amount per class
max_protos = 5
seed_val = 31
random.seed(seed_val)
np.random.seed(seed_val)
```

```
# Index i of this holds the Hvq for epoch i.
quantization_errors = []
final_hvq = []
final_protos = []
final_proto_trace = []


print("\tStarting Vector Quantization!")
print("Parameters")
print("K = "+str(amt_protos))
print("tmax = " + str(t_max))
print("Learning Rate = " + str(learning_rate))
print("Seed = " + str(seed_val))


while amt_protos <= max_protos:
    # Creating the prototypes change the function to initialize in a
   different way
    # prototypes is also a pandas dataframe
    prototypes, prototype_trace = create_prototypes_random(amt_protos,
    df)
    # reset values for this iteration
    min_error = float('inf')
    min_error_prototypes = None
    min_error_trace = None
    quantization_errors.append([])
    for i in range(0, t_max):
        # Make a copy of df before each epoch to allow for random
   selection of points in each epoch, and to keep df same
        df_copy = df.copy()

        # Show progress on command line
        print('\r', "Epoch = " + str(i), end='')
        for j in range(amt_points):

            # Get single example
            drop_index = np.random.choice(df_copy.index, 1, replace=
   False)
            row = df_copy.loc[drop_index[0]]
            df_copy = df_copy.drop(drop_index[0])  # Get rid of the
   example so we don't get it again this epoch

            # Get winning prototype
            closest_prototype, closest_idx = get_closest_prototype(
   prototypes, row)
            function = (lambda x, y: x + y)
            closest_prototype['X'] = function(closest_prototype['X'],
   learning_rate * (row['X'] - closest_prototype['X']))
            closest_prototype['Y'] = function(closest_prototype['Y'],
   learning_rate * (row['Y'] - closest_prototype['Y']))
```

```
            prototype_trace.iloc[closest_idx]['trace_X'].append(
    closest_prototype['X'])
            prototype_trace.iloc[closest_idx]['trace_Y'].append(
    closest_prototype['Y'])

        # Calculate quantization error for each epoch.
        hvq = calc_quantization_error(df, prototypes)

        # Update the minimum state if needed.
        if hvq < min_error:
            min_error = hvq
            min_error_prototypes = prototypes.copy()
            min_error_trace = prototype_trace.copy()
        quantization_errors[amt_protos - 1].append(hvq)

    # store values for this amount of prototypes
    final_hvq.append(hvq)
    final_protos.append(prototypes)
    final_proto_trace.append(prototype_trace)
    amt_protos += 1


# Plot learning curve
print("\nMinimum quantization error reached: " + str(min(
    quantization_errors)))
i = 1
for protos, trace in zip(final_protos, final_proto_trace):
    plot_data_points(df, protos, trace, title=f"Final Prototype
    Positions with {i} Prototypes")
    i += 1
    plot_curve(quantization_errors[i - 1], title=f"Learning Curve
    after {len(quantization_errors)} epochs",
               x_lab="Epoch", y_lab="Quantization Error")

# plot final Hvq as function of K
plot_curve(final_hvq, title="Final $H_{vq}$ values for different K
    values", x_lab="K", y_lab="Final Quantization Error",
           ticks=range(1, len(final_hvq) + 1))
plt.show()
```

6.2. CALCULATION.PY

```
import math
import random
import numpy as np
import pandas as pd


def calc_distance(row, prototype):
    return math.sqrt((row['X']-prototype['X'])**2 + (row['Y']-
    prototype['Y'])**2)
```

```
def get_closest_prototype(prototypes, row):
    smallest_dist = float('inf')
    for index, prototype in prototypes.iterrows():
        curr_dist = calc_distance(row, prototype)
        if curr_dist < smallest_dist:
            smallest_dist = curr_dist
            closest_prototype = prototype
            closest_idx = index
    return closest_prototype, closest_idx


def calc_squared_dist(row, winner):
    return calc_distance(row, winner)**2


def calc_quantization_error(df, prototypes):
    Hvq = 0
    # In words: For each data point in df, determine winner.
    # Then compute squared distance between point and winner, and sum
   all these distances up.
    for index, row in df.iterrows():
        closest_prototype, _ = get_closest_prototype(prototypes, row)
        Hvq += calc_squared_dist(row, closest_prototype)
    return Hvq


def create_prototypes_random(amt, df):
    prototypes = pd.DataFrame(columns=['X', 'Y'])
    prototype_trace = pd.DataFrame(columns=['trace_X', 'trace_Y'])
    for i in range(0, amt):
        idx = random.randint(0, len(df.index) - 1)
        proto = df.iloc[[idx]].to_numpy()[0]
        prototypes = prototypes.append({'X': proto[0], 'Y': proto[1]},
    ignore_index=True)
        prototype_trace = prototype_trace.append({'trace_X': [proto
   [0]], 'trace_Y': [proto[1]]}, ignore_index=True)
    return prototypes, prototype_trace


def create_prototypes_subpar(amt, df):
    max_x = df['X'].max()
    max_y = df['Y'].max()
    prototypes = pd.DataFrame(columns=['X', 'Y'])
    prototype_trace = pd.DataFrame(columns=['trace_X', 'trace_Y'])
    for i in range(0, amt):
        # chose values that increase and are definitely far away from
   the data
        x = max_x * 2 + 10 * i
        y = max_y * 2 + 10 * i
        prototypes = prototypes.append({'X': x, 'Y': y}, ignore_index=
    True)
        prototype_trace = prototype_trace.append({'trace_X': [x], '
    trace_Y': [y]}, ignore_index=True)
    return prototypes, prototype_trace
```

6.3. PLOTTING.PY

```
from matplotlib import pyplot as plt


def plot_curve(data, title="", x_lab="", y_lab="", ticks=None):
    plt.figure()
    plt.title(title)
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    if ticks is not None:
        plt.xticks(ticks)
    plt.plot(range(1, len(data) + 1), data)


def plot_data_points(df, prototypes, prototype_trace, title="", x_lab
    ="", y_lab=""):
    color_zero = "gray"
    color_one = "blue"
    edge_color = (0, 0, 0, 0.5)
    trace_color = (0, 0, 0, 0.85)
    plt.figure()
    plt.scatter(df['X'], df['Y'], label='Class 1', color=color_zero,
    edgecolors=edge_color)
    for idx, prototype in prototypes.iterrows():
        plt.plot(prototype_trace.iloc[idx]['trace_X'], prototype_trace.
    iloc[idx]['trace_Y'], color=trace_color)
        plt.scatter(prototype['X'], prototype['Y'], marker='*', color=
    color_one, s=600, edgecolors=edge_color)
    plt.title(title)
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
```