Introduction to Machine Learning
Assignment 1
Learning Vector Quantization

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

November 1, 2021

Contents

Learning Vector Quantization

1. Introduction

Classification problems arise in many areas, ranging from diagnosing illnesses to recognition of speech or images. For example, say we collect and arrange lots of fruit and have the near-perfect examples of all fruits. In other words, we have prototypes for all fruits. When we are a given a new fruit, this fruit can be compared with all the prototypes that we had collected and arranged. Then the prototype which is most similar to the new fruit will be that fruit's type. This process of creating (training) prototype examples from data and then using these prototypes to classify new data into it's correct classes is seen in classification algorithms, specifically the one that is implemented in this lab, the learning vector quantization algorithm.

In this lab, we implement and analyze LVQ which is a prototype-based data classification algorithm. It does multiple iterations (epochs) of comparisons between prototypes and random data points in order to achieve trained prototypes. The LVQ algorithm is a supervised machine learning algorithm. This means that class labels of the training data have to be known a priori.

Our data set contains 100 data points with two feature dimensions. In this analysis we will refer to the dimensions of a data point p as $p.x$ and $p.y$.

2. Method

2.1. General

The goal of the LVQ algorithm is to create a set of prototypes for each class in the data set. These prototypes should represent these classes as well as possible. Novel data points can then be classified using the label of the nearest prototype.

2.2. Classification

To measure the distance between two data points p_1, p_2 a distance function $d(p_1, p_2)$ needs to be defined. In this assignment we used the squared Euclidean distance. Since our data points have two dimensions we get the following definition of d.

$$d(p_1, p_2) = (p_1.x - p_2.x)^2 + (p_1.y - p_2.y)^2$$

Given a set of prototypes W with $|W| = K$ and a set of class labels S we can classify the class $S(p)$ of a data point p the following way. To classify the data point p, with unknown class label $S(p)$, it's distance to all prototypes $w_i \in W$ needs to be determined. The winner w^* is defined as the prototype closest to p. Then p is assigned the class $S_i = S(w^*)$. This can be expressed by the following equation.

$$S(p) = S(w^*) = S(\underset{w_j}{\operatorname{argmin}}\{d(w_j, p)\})_{j=1}^{K}$$

2.3. Training

The LVQ algorithm uses an iterative training procedure to find optimal positions for all prototypes w_i. Initially, all prototypes w_i are initialized at the coordinates of a random data point p_j, for which $S(w_i) = S(p_j)$. Now in each training epoch every data point p in the Training set P_{train} is evaluated in a random order. For each data point $p_j \in P_{train}$ we find the closest prototype w^* and update its position according to the class labels of p_j and w^*, based to the following formula.

$$w^* \leftarrow w^* - \eta * \Phi(S(w^*), S(p_j)) * (p_j - w^*)$$

where

$$\Phi(S(p_1), S(p_2)) = \begin{cases} 1 & \text{if } S(p_1) = S(p_2) \\ -1 & \text{if } S(p_1) \neq S(p_2) \end{cases}$$

$$\eta: \text{ learning rate}$$

From these equations we can see that the position of the prototype w^* is moved closer to the data point p_j if they have the same class and moved further away if they do not have the same class. The magnitude of this position shift is determined by two factors. The first factor is the distance between w^* and p_j. The second factor is the learning rate η. The learning rate is a hyper-parameter that has to be set before the execution of this algorithm. From preliminary experiments we have determined that $\eta = 0.002$ to give satisfactory results. Another hyper-parameter that has to be set before the algorithm can be applied is the amount of prototypes per class. While too few prototypes might not be able to properly represent the nature of the data, too many prototypes might lead to over-fitting.

2.4. Different Training Possibilities

In the basic LVQ algorithm, which is implemented in this report, the learning rate is a constant. However, the learning rate can be adjusted throughout the training process to achieve different results. If the learning rate is defined as a function of time (the current epoch) then the magnitude of positional change of a prototype can decrease as training proceeds. This LVQ version uses the assumption that prototypes are expected to reach a more optimal version the longer the training process goes on for. This means that the positional change should be less significant in later training epochs.

Another adjustment that can be made to the LVQ is that more than one prototype's position can be adjusted in each training epoch. If we define w_k^* as the kth closest prototype to the current data point p, then we can adjust it's position by using the following formula.

$$w_k^* \leftarrow w_k^* - \eta * \frac{\alpha}{k} * \Phi(S(w^*), S(p_j)) * (p_j - w^*)$$

This ensures that the prototypes further from p also get adjusted but with a lower magnitude. In this case α is another hyper-parameter that has to be set. The maximal amount of k for which w_k^* should be updated must also be defined.

3. Results

We asses the accuracy of our classifier by using the error rate of the classifications. This means we divide the amount of errors of the classifier by the total amount of points in the data set. The process of the training can be visualized using a learning curve.
A learning curve is a graph showing the progression of the error rate of the classifier after each training epoch. Our training ends after a maximum of $t_{max} = 100$ epochs or when a no decrease in training error is seen after 10 consecutive epochs.

To ensure consistent results, we have defined a seed for all randomized parts of the classification process.

For all of the training presented in this section a learning rate of 0.002 has been used, and prototypes are initialized at a random position of a data point of the same class.

3.1. A) One Prototype per Class

First we used only one prototype for the classification. The Learning curve for this section can be seen in figure 1. The final position and trajectory of the prototypes can be seen in figure 2.

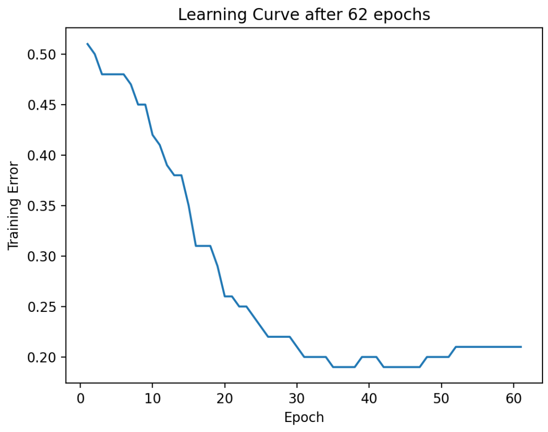

Figure 1: *Learning curve of the LVQ classifier using one prototype per class.*

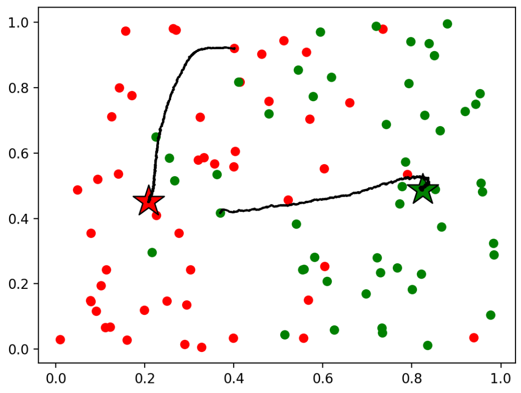

Figure 2: *Scatter plot of all data points and the two prototypes. Prototypes are marked as stars. Black lines represent the prototype's trajectory during the training process.*

We can see that the training error decreases quite steeply in the first 30 epochs. It reaches its minimum of 0.19 between epoch 30 and 40 and again between epoch 40 and 50. Finally, after 62 epochs the training error stabilizes at 0.21. The prototype's positions change quite a lot during the training process.

3.2. B) Two Prototypes per Class

Now we increased the prototypes per class to two for the classification. The Learning curve for this section can be seen in figure 3. The final position and trajectory of the prototypes can be seen in figure 4.

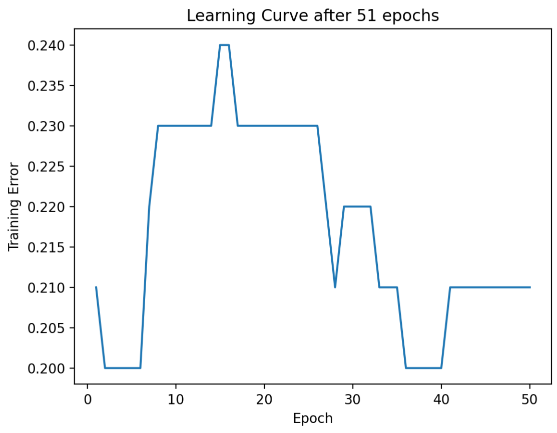

Figure 3: *Learning curve of the LVQ classifier using two prototypes per class.*

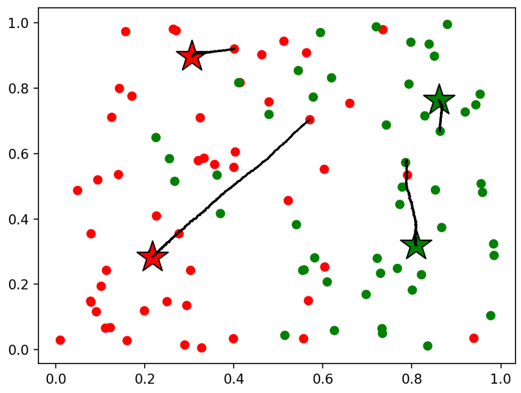

Figure 4: *Scatter plot of all data points and the four prototypes. Prototypes are marked as stars. Black lines represent the prototype's trajectory during the training process.*

We can see that the training error starts out fairly low and reaches its minimum of 0.20 in the first 10 epochs. Afterwards the training error increases until reaching its maximum at 0.24 after 15 epochs. After that it decreases again reaching its minimum of 0.2 between epoch 30 and 40. Finally, after 51 epochs the training error stabilizes at 0.21. Two of the prototypes' positions change quite a lot during the training process, while the other two stay somewhat close to their initial position.

3.3. Three Prototypes per Class

For further comparison we rerun the training process of the classifier with three prototypes per class. The Learning curve for this section can be seen in figure 5. The final position and trajectory of the prototypes can be seen in figure 6.

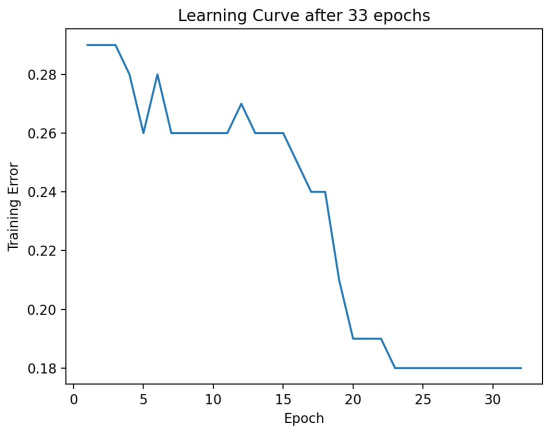

Figure 5: *Learning curve of the LVQ classifier using three prototypes per class.*

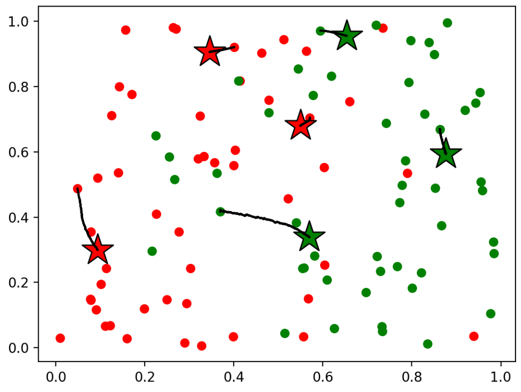

Figure 6: *Scatter plot of all data points and the six prototypes. Prototypes are marked as stars. Black lines represent the prototype's trajectory during the training process.*

We can see that the training error decreases quite steeply in the first 20 epochs. It reaches its minimum of 0.18 between epoch 20 and 25. The training error stabilizes at that minimal value of 0.18 after 33 epochs. Four of the prototypes' positions change very minimally during the training process, while the other two move further from their initial position.

3.4. Four Prototypes per Class

Finally, we executed the training process one more time with four prototypes per class. The Learning curve for this section can be seen in figure 7. The final position and trajectory of the prototypes can be seen in figure 8.

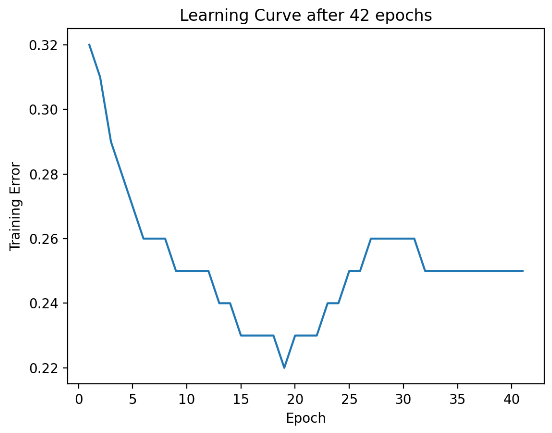

Figure 7: *Learning curve of the LVQ classifier using four prototypes per class.*

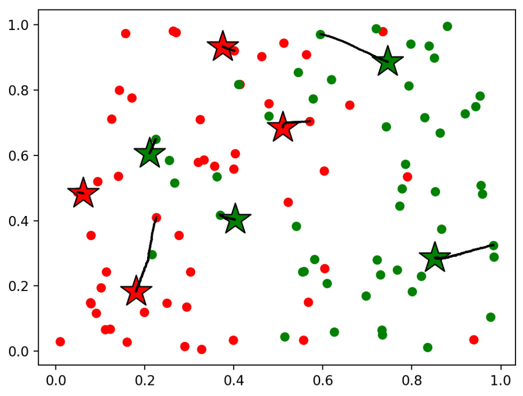

Figure 8: *Scatter plot of all data points and the eight prototypes. Prototypes are marked as stars. Black lines represent the prototype's trajectory during the training process.*

We can see that the training error decreases quite steeply in the first 20 epochs. It reaches its minimum of 0.22 around epoch 20. The training error then increases until epoch 30, and then stabilizes at a value of 0.25 after 42 epochs. Three of the red prototypes' and two of the green prototypes' positions change very minimally during the training process, while the other three move further from their initial position.

4. Discussion

4.1. One Per Class

4.1.1. Learning Curve and Trajectory

The learning curve of this case, Figure 1, is decisive and it can be said that the training error always decreased over time except for a single epoch. This gives the conclusion that the one prototype per class approach always improved the prototype with each epoch. As seen in their trajectory lines in Figure 2, with no other prototypes to compete and intervene with, these solo prototypes always improve their position and thus, have a lower error rate.

4.1.2. Prototype Results

The trained prototypes in this case end up in approximately the middle point of all points in their respective class. Looking at Figure 2 it can be seen that both the red and green prototypes positioned themselves in the middle spot with respect to their own data points x and y coordinates. This result of ending up in the mean, can be tied to the fact that there is only one prototype for each class. Unlike in two prototypes per class, the prototypes do not get pushed to the edges, but rather maintain an average position to compensate for the lack of other prototypes.

4.2. Two Per Class

4.2.1. Learning Curve and Trajectory

Figure 3 shows a very different learning curve compared to the one per class example. The training error starts off very low, meaning that the two randomly selected positions, by chance, are already good to produce a low error rate. The learning curve shows a rapid increase in epochs 10 to 30, meaning that the prototypes go through very bad positions in terms of error. This can be tied into the fact that there are four prototypes total and they intervene with each other, leading to a high error and sharp highs and lows during the middle of the training. As seen in towards the final epochs in Figure 3, the prototypes return back to the initial training error of 0.21. Looking at the trajectories, it can be seen that the prototypes start off really close to each other, but after training they position themselves further away from each other.

4.2.2. Prototype Results

Since there are two prototypes for each class, they evenly space out between each other at the end. As seen in Figure 4, the prototype markers have a position which is the lower average and higher average of their respective class points. It can be seen that both for the red and green points, the prototypes place themselves in the middle of their upper and lower spread with respect to the y-axis. With the learning trajectory of the prototypes, it can be seen how they gain their positions by moving towards the lower and upper clusters of points.

4.3. Three Per Class

4.3.1. Learning Curve and Trajectory

As seen in Figure 5, this trial of LVQ shows a stable error for the first 20 epochs and then a sharp, immense decrease in error after that. The prototypes then reach their equilibrium at an error rate of 0.18, which is the lowest training error achieved out of all K-Prototype approaches that were run, from K = 1 to K = 4. Based on this and the results from subsection 4.4, it can be said that a three prototype per class LVQ worked better for this data set then one, two or four per class. It can be proposed that setting the number of prototypes per class to more than three might lead to over-fitting, which is discussed in the conclusion, subsection 4.5.

4.3.2. Prototype Results

In this scatter plot, Figure 6, the prototypes are not influenced by outliers and maintain a better overall position with respect to their class. What is meant by this is that, looking at the positions around the coordinate (0.3,0.5), some green points can be seen amongst the red data points. As much as they are green points, they act as outliers because they are far away from the green cluster towards x = 0.8, and are among red points. As seen by the prototype markers, the green prototypes are not affected by these outliers and avoid over-fitting by staying at positions where $x >= 0.6$, which is where the majority of green points are located at.

4.4. Four Per Class

4.4.1. Learning Curve and Trajectory

The learning curve in Figure 7 shows how the protoypes improved well after 42 epochs. Even though it reaches a very low error rate of 0.22 during the middle of the training, the error rates increases a tiny bit towards the end. Overall, a significant reduction in error is observed still.

4.4.2. Prototype Results

The prototypes final positions as seen in Figure 8 show a glimpse of overfitting, which in short means that a prototypes positions and trains itself too extensively, resulting in it not acting as an overall representative of the class but rather on a particular data point(s) which may be outliers. Observing the green prototype at $x = 0.2$, it is placed too far away from the green clusters around $x = 0.8$. It can be said that this green prototype is compensating for the few green points around $x = 0.2$. It can also be argued that it has positioned itself too far towards the red points and it has overfitted in training. In our analysis, it is more probable that this is not considered overfitting, but it is a glimpse of it and how overfitting prototypes will occur as we increase the number of prototypes per class further.

4.5. Conclusion

As seen from the discussion of learning curves, trajectories and final positions of prototypes, overall trends and patterns are seen. In each K-prototype trial where K is the number of prototypes per class, the error rate decreases significantly meaning prototypes position themselves better as a representative of their corresponding class. For all scatter plots showing the data points and prototypes on a 2D graph, the observation was that for $K = 1, 2, 3$, the prototypes positioned themselves very well with respect to their data points spread. The prototypes cover the middle points of majority of its points and are not affected by outlying points far away from data clusters. At $K = 4$, Figure 8, some suspicious prototypes which show a glimpse of overfitting are observed. These kind of overfitting prototypes are not necessarily fitting for majority of data points but rather too specifically focused on particular points of their classes, which may be considered as outliers. A good example of this was the green prototype star at $x = 0.2$ in Figure 8. So the conclusion is supporting the pattern that for LVQ1 with $K > 3$, the prototypes start over-training themselves and over-fit their positions. Such case should be avoided because over-fitting prototypes may fail to fit new data and may present a wrong prototype for it's class. Early detection and labelling of outliers is thus helpful to compensate for over-fitting prototypes.
A final and minor pattern that is observed is how the length of trajectory lines in the visualizations decreased. This is expected and trivial since each training is at maximum t_{max} which is the limit number of epochs and only one prototype is updated at each epoch. So with $K = 2$, the two prototypes have near equal length trajectories, suggesting a half split of moves (updates) for each prototype as seen in Figure 4. So approximately both prototypes move equal times. Meanwhile, looking at Figure 8, it can be seen how some prototypes trajectories are so short that it is hard to see without zooming in to the plot. It is not a critical point of analysis but the pattern is worth mentioning. If this shortening of trajectories is unwanted, another variation of LVQ can be implemented in which multiple prototypes are trained in each epoch.
Specific patterns regarding the training and results were discussed and recorded in the discussion section, section 4.
In conclusion, it can be said with confidence that this implementation of the LVQ1 algorithm has worked as intended and through analysis has shown it's purpose, advantages and short-comings such as over-fitting.

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. lvq.py

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


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

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

# Adding label to the data frame
labels = np.concatenate((np.full((50, 1), 0, dtype=int), np.full((50,
    1), 1, dtype=int)), axis=None)
df["Label"] = labels

# Setting learning variables
learning_rate = 0.002
t_max = 100
threshold = 10
amt_protos = 1 # adjust for different prototype amount per class
random.seed(123)

# Creating the prototypes
# prototypes is also a pandas dataframe
prototypes, prototype_trace = create_prototypes_random(amt_protos, df)

# Algorithm
# epoch loop
training_errors = []
```

```
count = 0
for i in range(0, t_max):
    for index, row in df.iterrows():
        # randomize
        df = df.sample(frac=1).reset_index(drop=True)
        # for current point check closest prototype
        closest_prototype, closest_idx = get_closest_prototype(
   prototypes, row)
        # update the prototype
        function = (lambda x, y: x + y) if int(closest_prototype['
   Label']) == int(row['Label']) else (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'])
    training_error = calc_training_error(df, prototypes)
    if i != 0 and training_error == training_errors[i - 1]:
        count += 1
    else:
        count = 0
    if count == threshold:
        break
    training_errors.append(training_error)

plot_data_points(df, prototypes, prototype_trace)
plot_learning_curve(training_errors)
```

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_training_error(df, prototypes):
    fail = 0
    for index, row in df.iterrows():
        closest_prototype, _ = get_closest_prototype(prototypes, row)
        fail += int(closest_prototype['Label']) != int(row['Label'])
    return fail / df.shape[0]


def create_prototypes_random(amt_per_class, df):
    prototypes = pd.DataFrame(columns=['X', 'Y', 'Label'])
    prototype_trace = pd.DataFrame(columns=['trace_X', 'trace_Y'])
    for i in range(0, amt_per_class * 2):
        if i < amt_per_class:
            idx = random.randint(0, 50)
        else:
            idx = random.randint(50, 100)
        proto = df.iloc[[idx]].to_numpy()[0]
        prototypes = prototypes.append({'X': proto[0], 'Y': proto[1], '
   Label': proto[2]}, ignore_index=True)
        prototype_trace = prototype_trace.append({'trace_X': [proto
   [0]], 'trace_Y': [proto[1]]}, ignore_index=True)
    return prototypes, prototype_trace
```

6.3. PLOTTING.PY

```
from matplotlib import pyplot as plt


def plot_learning_curve(training_errors):
    plt.title(f"Learning Curve after {len(training_errors) + 1} epochs
   ")
    plt.xlabel("Epoch")
    plt.ylabel("Training Error")
    plt.plot(range(1, len(training_errors) + 1), training_errors)
    plt.show()


def plot_data_points(df, prototypes, prototype_trace):
    color_zero = "red"
    color_one = "green"
    df_zero = df[df['Label'] == 0]
    df_one = df[df['Label'] == 1]
    plt.scatter(df_zero['X'], df_zero['Y'], label='Class 1', color=
   color_zero)
    plt.scatter(df_one['X'], df_one['Y'], label='Class 2', color=
   color_one)
    for idx, prototype in prototypes.iterrows():
        plt.plot(prototype_trace.iloc[idx]['trace_X'], prototype_trace.
    iloc[idx]['trace_Y'], color='black')
        color = color_zero if prototype['Label'] == 0 else color_one
        plt.scatter(prototype['X'], prototype['Y'], marker='*', color=
    color, s=600, edgecolors='black')
    plt.show()
```