Praktikum Sistem Cerdas

NRP	: 3223600019
Nama	: Muhammad Bimo Fachrizky
Materi	: Membuat Program Aplikasi Learning Vector Quantization
Tanggal	: Senin, 14 April 2025

Praktikum 8

Membuat Program Aplikasi Learning Vector Quantization

I. Tujuan Pembelajaran

- KohonenMahasiswa dapat memahami dan menjelaskan konsep Learning Vector Quantization
- Mahasiswa dapat menjelaskan model Learning Vector Quantization
- Mahasiswa dapat membuat aplikasi Learning Vector Quantization

Software yang di perlukan

- Microsoft Visual C++
- PyCharm

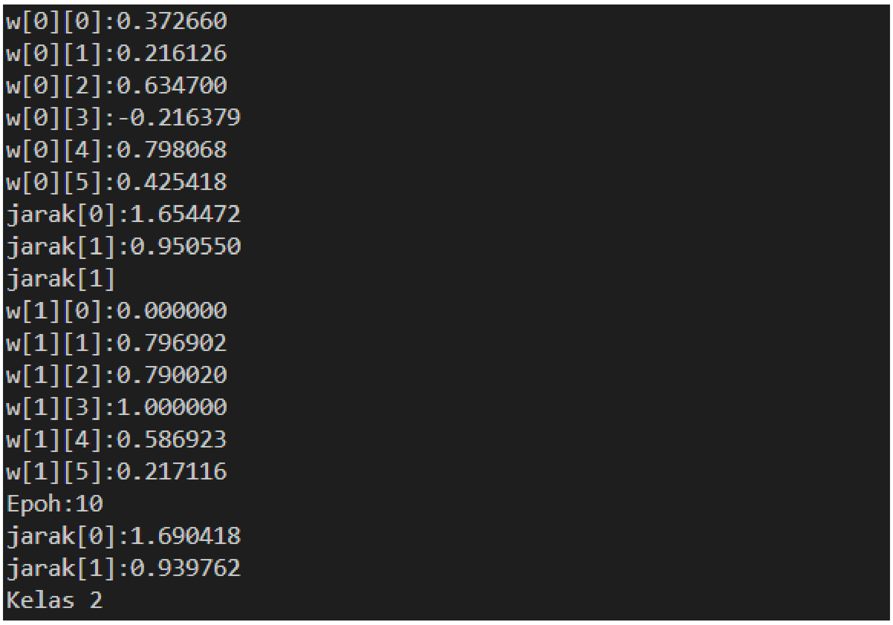
II. Langkah percobaan

1. Program Aplikasi Learning Vector Quantization

```
#include "stdio.h"
#include <conio.h>
#include <math.h>

int main()
{
    int i, j, k, epoh = 0, maxIter = 10;
    float x[10][6] =
    {
        {1, 0, 0, 0, 1, 0},
        {0, 1, 1, 1, 1, 0},
        {0, 0, 1, 0, 0, 1},
        {0, 0, 1, 0, 1, 0},
        {0, 1, 0, 0, 0, 1},
        {1, 0, 1, 0, 1, 1},
        {0, 0, 1, 1, 0, 0},
        {0, 1, 0, 1, 0, 0},
        {1, 0, 0, 1, 0, 1},
        {0, 1, 1, 1, 1, 1}
    };
    int T[10] = {1, 2, 1, 1, 1, 1, 2, 2, 2, 2};
    float w[2][6], jarak[2], alpha = 0.05f;

    // Inisialisasi bobot
    jarak[0] = 0.0;
    jarak[1] = 0.0;
    for (i = 0; i < 6; i++)
```

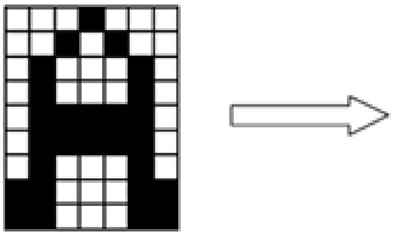
```
{
   w[0][i] = x[0][i];
   w[1][i] = x[1][i];
}

// Training
for (i = 0; i < 10; i++)
{
   for (j = 2; j < 10; j++)
   {
      jarak[0] = 0.0;
      jarak[1] = 0.0;
      for (k = 0; k < 6; k++)
      {
         jarak[0] = jarak[0] + (x[j][k] - w[0][k]) * (x[j][k] - w[0][k]);
         jarak[1] = jarak[1] + (x[j][k] - w[1][k]) * (x[j][k] - w[1][k]);
      }

      jarak[0] = sqrt(jarak[0]);
      jarak[1] = sqrt(jarak[1]);

      printf("jarak[0]:%f\n", jarak[0]);
      printf("jarak[1]:%f\n", jarak[1]);

      if (jarak[0] <= jarak[1])
      {
         printf("jarak[0]\n");
         if (T[j] == 1)
         {
            for (k = 0; k < 6; k++)
            {
               w[0][k] = w[0][k] + alpha * (x[j][k] - w[0][k]);
               printf("w[0][%d]:%f\n", k, w[0][k]);
            }
         }
         else
         {
            for (k = 0; k < 6; k++)
            {
               w[0][k] = w[0][k] - alpha * (x[j][k] - w[0][k]);
               printf("w[0][%d]:%f\n", k, w[0][k]);
            }
         }
```

```
        }
        else
        {
            printf("jarak[1]\n");
            if (T[j] == 2)
            {
                for (k = 0; k < 6; k++)
                {
                    w[1][k] = w[1][k] + alpha * (x[j][k] - w[1][k]);
                    printf("w[1][%d]:%f\n", k, w[1][k]);
                }
            }
            else
            {
                for (k = 0; k < 6; k++)
                {
                    w[1][k] = w[1][k] - alpha * (x[j][k] - w[0][k]);
                    printf("w[1][%d]:%f\n", k, w[1][k]);
                }
            }
        }
    }

    alpha = alpha - 0.1 * alpha;
    epoh++;
    printf("Epoh:%d\n", epoh);
}

// Running
x[0][0] = 0;
x[0][1] = 1;
x[0][2] = 0;
x[0][3] = 1;
x[0][4] = 1;
x[0][5] = 0;

jarak[0] = 0.0;
jarak[1] = 0.0;

for (k = 0; k < 6; k++)
{
    jarak[0] = jarak[0] + (x[0][k] - w[0][k]) * (x[0][k] - w[0][k]);
    jarak[1] = jarak[1] + (x[0][k] - w[1][k]) * (x[0][k] - w[1][k]);
    }

    jarak[0] = sqrt(jarak[0]);
    jarak[1] = sqrt(jarak[1]);

    printf("jarak[0]:%f\n", jarak[0]);
    printf("jarak[1]:%f\n", jarak[1]);

    if (jarak[0] <= jarak[1])
    {
        printf("Kelas 1\n");
    }
    else
    {
        printf("Kelas 2\n");
    }
}
```



```
w[0][0]:0.372660
w[0][1]:0.216126
w[0][2]:0.634700
w[0][3]:-0.216379
w[0][4]:0.798068
w[0][5]:0.425418
jarak[0]:1.654472
jarak[1]:0.950550
jarak[1]
w[1][0]:0.000000
w[1][1]:0.796902
w[1][2]:0.790020
w[1][3]:1.000000
w[1][4]:0.586923
w[1][5]:0.217116
Epoh:10
jarak[0]:1.690418
jarak[1]:0.939762
Kelas 2
```

2. Tugas
   Akan dicoba untuk mengenali huruf A,B, atau H yang direpresentasikan dengan menggunakan kode 0 dan 1 pada matriks berukuran 9x7 seperti pada gambar



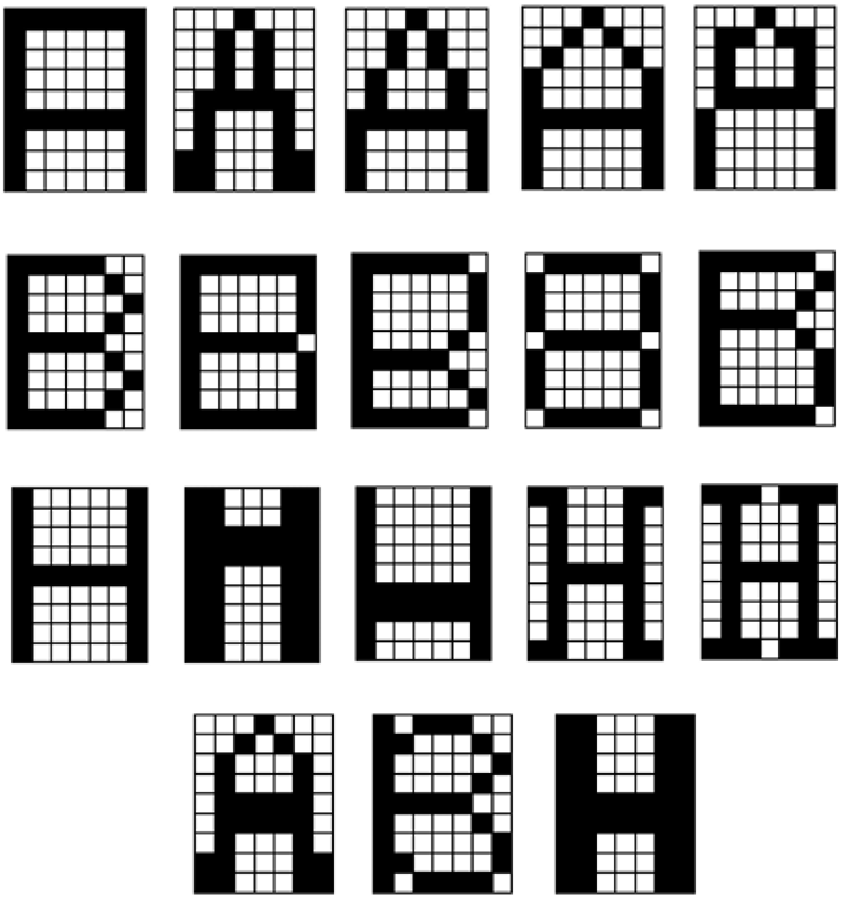
$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Pada gambar tersebut,kode 1 menunjukkan suatu kotak berwarna hitam, sedangkan kode 0 menunjukkan suatu kotak berwarna putih.

- Untuk mempermudah dalam implementasi, matriks 9x7 tersebut dibawa ke bentuk vektor 63 kolom. Pada Gambar tersebut vektor yang bersesuaian adalah:

000100000101000100010010001001111100111110010001011000111100011

- Misalkan ada 15 data yang akan dilatih, yaitu 5 data A, 5 data B, dan 5 data H, sebagai berikut



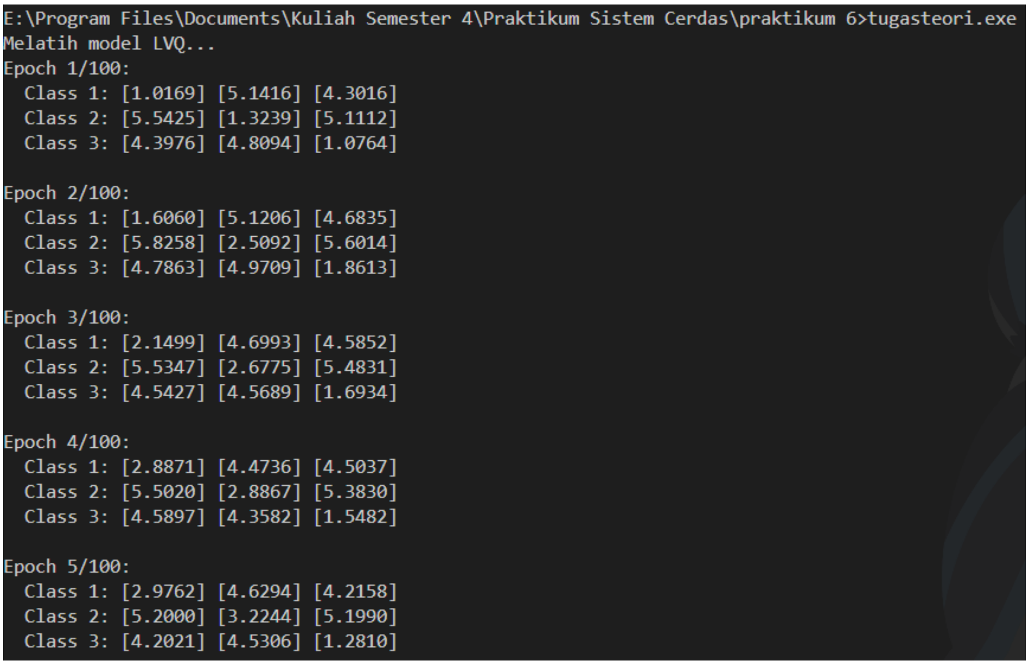
- Program

```
#include <iostream>
#include <vector>
#include <cmath>
#include <string>
#include <cstdio>
using namespace std;
const int VECTOR_SIZE = 63;
const int NUM_CLASSES = 3;
const int NUM_SAMPLES_PER_CLASS = 5;
const int NUM_EPOCHS = 100;
const double INITIAL_LEARNING_RATE = 0.1;

double euclideanDistance(const vector<double>& v1, const
vector<double>& v2) {
    double sum = 0.0;
    for (int i = 0; i < VECTOR_SIZE; ++i) {
        sum += pow(v1[i] - v2[i], 2);
    }
    return sqrt(sum);
}

class LVQ {
public:
    LVQ() {
        weightVectors.push_back(vector<double>(trainingData[0].begin(),
trainingData[0].end()));
        weightVectors.push_back(vector<double>(trainingData[5].begin(),
trainingData[5].end()));

        weightVectors.push_back(vector<double>(trainingData[10].begin(),
trainingData[10].end()));
    }

    void train() {
        double learningRate = INITIAL_LEARNING_RATE;
        for (int epoch = 0; epoch < NUM_EPOCHS; ++epoch) {
            for (int i = 0; i < trainingData.size(); ++i) {
                const vector<double>& sample = trainingData[i];
                const string& trueClass = trainingLabels[i];

                int closestIndex = findClosestWeightVector(sample);
                string closestClass = classLabels[closestIndex];
                if (trueClass == closestClass) {
                    for (int j = 0; j < VECTOR_SIZE; ++j) {
                        weightVectors[closestIndex][j] += learningRate *
(sample[j] - weightVectors[closestIndex][j]);
                    }
                } else {
                    for (int j = 0; j < VECTOR_SIZE; ++j) {
                        weightVectors[closestIndex][j] -= learningRate * (sample[j]
- weightVectors[closestIndex][j]);
                    }
                }
            }

            printf("Epoch %d/%d:\n", epoch + 1, NUM_EPOCHS);
            for (int c = 0; c < NUM_CLASSES; ++c) {
                int repIndex = c * NUM_SAMPLES_PER_CLASS;
                printf("  Class %d:", c + 1);
                for (int w = 0; w < NUM_CLASSES; ++w) {
                    double dist = euclideanDistance(trainingData[repIndex],
weightVectors[w]);
                    printf(" [%.4f]", dist);
                }
                printf("\n");
            }

            learningRate *= 0.95;
            printf("\n");
        }


        printf("Bobot Final:\n");
        for (int c = 0; c < NUM_CLASSES; ++c) {
            printf("  Class %d: ", c + 1);
            for (int j = 0; j < VECTOR_SIZE; ++j) {
                printf("%.4f ", weightVectors[c][j]);
            }
            printf("\n");
        }
        printf("\nPelatihan selesai.\n\n");
    }
    string classify(const vector<double>& input, vector<double>&
distances) {
        distances.clear();
        for (int i = 0; i < NUM_CLASSES; ++i) {
            distances.push_back(euclideanDistance(input, weightVectors[i]));
        }
        int closestIndex = 0;
        double minDistance = distances[0];
        for (int i = 1; i < NUM_CLASSES; ++i) {
            if (distances[i] < minDistance) {
                minDistance = distances[i];
                closestIndex = i;
            }
        }
        return classLabels[closestIndex];
    }

private:
    vector<vector<double>> weightVectors;
    vector<string> classLabels = {"A", "B", "H"};

    vector<vector<double>> trainingData = {
        // 5 Dataset A
        {
            1,1,1,1,1,1,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,1,1,1,1,1,1,

            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1
        },
        {
            0,0,0,1,0,0,0,
            0,0,1,0,1,0,0,
            0,0,1,0,1,0,0,
            0,0,1,0,1,0,0,
            0,1,1,1,1,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            1,1,0,0,0,1,1,
            1,1,0,0,0,1,1
        },
        {
            0,0,0,1,0,0,0,
            0,0,1,0,1,0,0,
            0,0,1,0,1,0,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            1,1,1,1,1,1,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1
        },
        {
            0,0,0,1,0,0,0,
            0,0,1,0,1,0,0,
            0,1,0,0,0,1,0,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,1,1,1,1,1,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1
        },
        {
            0,0,0,1,0,0,0,
            0,1,1,0,1,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,1,1,1,1,0,

            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1
        },
         // 5 Dataset B
         {
            1,1,1,1,1,0,0,
            1,0,0,0,0,1,0,
            1,0,0,0,0,0,1,
            1,0,0,0,0,1,0,
            1,1,1,1,1,0,0,
            1,0,0,0,0,1,0,
            1,0,0,0,0,0,1,
            1,0,0,0,0,1,0,
            1,1,1,1,1,0,0
         },
         {
            1,1,1,1,1,1,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,1,1,1,1,1,0,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,1,1,1,1,1,1
         },
         {
            1,1,1,1,1,1,0,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,1,0,
            1,1,1,1,1,0,0,
            1,0,0,0,0,1,0,
            1,0,0,0,0,0,1,
            1,1,1,1,1,1,0
         },
         {
            0,1,1,1,1,1,0,
            1,0,0,0,0,0,1,

            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            0,1,1,1,1,1,0,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            1,0,0,0,0,0,1,
            0,1,1,1,1,1,0
         },
         {
            1,1,1,1,1,1,0,
            1,0,0,0,0,0,1,
         1,0,0,0,0,1,0,
         1,1,1,1,1,0,0,
         1,0,0,0,0,1,0,
         1,0,0,0,0,0,1,
         1,0,0,0,0,0,1,
         1,0,0,0,0,0,1,
         1,1,1,1,1,1,0
},
// 5 Dataset H
{
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,1,1,1,1,1,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1
},
{
    1,1,0,0,0,1,1,
    1,1,0,0,0,1,1,
    1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,
    1,1,0,0,0,1,1,
    1,1,0,0,0,1,1,
    1,1,0,0,0,1,1,
    1,1,0,0,0,1,1,
    1,1,0,0,0,1,1
},
{
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1,
    1,0,1,1,1,0,1,
    1,0,1,1,1,0,1,
    1,0,0,0,0,0,1,
    1,0,0,0,0,0,1
},
{
```

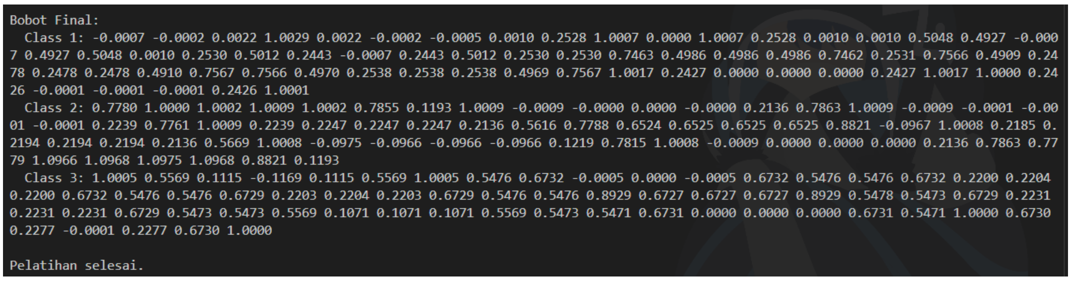
```
            1,1,0,0,0,1,1,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,1,1,1,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            1,1,0,0,0,1,1
        },
        {
            1,1,1,0,1,1,1,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,1,1,1,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            0,1,0,0,0,1,0,
            1,1,1,0,1,1,1
        }
    };

    vector<string> trainingLabels = {
        "A", "A", "A", "A", "A",
        "B", "B", "B", "B", "B",
        "H", "H", "H", "H", "H"
    };

    int findClosestWeightVector(const vector<double>& sample) {

        double minDistance = euclideanDistance(sample, weightVectors[0]);
        int closestIndex = 0;
        for (int i = 1; i < NUM_CLASSES; ++i) {
            double distance = euclideanDistance(sample, weightVectors[i]);
            if (distance < minDistance) {
                minDistance = distance;
                closestIndex = i;
            }
        }
        return closestIndex;
    }
};
```

```
void testLVQ(LVQ& lvq, float testing_data[3][VECTOR_SIZE], const
vector<string>& expectedLabels) {
    for (int i = 0; i < 3; ++i) {
        vector<double> sample(testing_data[i], testing_data[i] +
VECTOR_SIZE);
        vector<double> distances;
        string predictedClass = lvq.classify(sample, distances);
        printf("Sampel %d:\n", i + 1);
        printf("  Diharapkan: '%s'\n", expectedLabels[i].c_str());
        printf("  Dengan jarak:");
        for (int j = 0; j < NUM_CLASSES; ++j) {
            printf(" [%.4f]", distances[j]);
        }
        printf("\n");
        printf("  Hasil Prediksi: '%s'\n\n", predictedClass.c_str());
    }
}

int main() {
    LVQ lvq;
    printf("Melatih model LVQ...\n");
    lvq.train();

    float testing_data[3][VECTOR_SIZE] = {
        // Huruf H
        {1,1,0,0,0,1,1,
         1,1,0,0,0,1,1,
         1,1,0,0,0,1,1,
         1,1,0,0,0,1,1,
         1,1,1,1,1,1,1,

         1,1,1,1,1,1,1,
         1,1,0,0,0,1,1,
         1,1,0,0,0,1,1,
         1,1,0,0,0,1,1},
        // Huruf A
        {0,0,0,1,0,0,0,
         0,0,1,0,1,0,0,
         0,1,0,0,0,1,0,
         0,1,0,0,0,1,0,
         0,1,1,1,1,1,0,
         0,1,1,1,1,1,0,
         0,1,0,0,0,1,0,
         0,1,0,0,0,1,1,
         1,1,0,0,0,1,1},
        // Huruf B
        {1,0,1,1,1,0,0,
         1,0,0,0,0,1,0,
         1,0,0,0,0,0,1,
         1,0,0,0,0,1,0,
         1,1,1,1,1,0,0,
         1,0,0,0,0,1,0,
         1,0,0,0,0,0,1,
         1,1,0,0,0,0,1,
         1,0,1,1,1,1,0}
    };

    vector<string> expectedLabels = {"H", "A", "B"};

    printf("Hasil pengujian:\n");
    testLVQ(lvq, testing_data, expectedLabels);

    return 0;
}
```

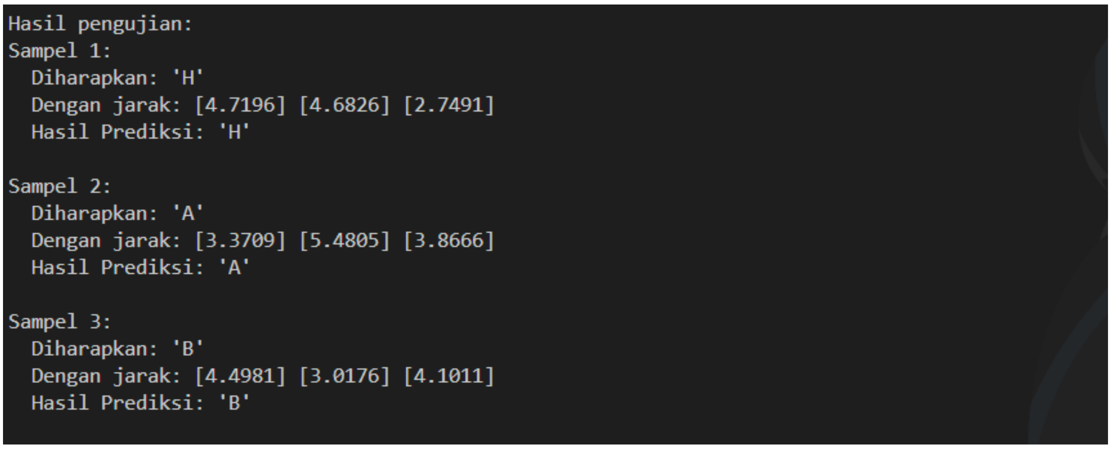
III. Hasil Percobaan



```
E:\Program Files\Documents\Kuliah Semester 4\Praktikum Sistem Cerdas\praktikum 6>tugasteori.exe
Melatih model LVQ...
Epoch 1/100:
  Class 1: [1.0169] [5.1416] [4.3016]
  Class 2: [5.5425] [1.3239] [5.1112]
  Class 3: [4.3976] [4.8094] [1.0764]

Epoch 2/100:
  Class 1: [1.6060] [5.1206] [4.6835]
  Class 2: [5.8258] [2.5092] [5.6014]
  Class 3: [4.7863] [4.9709] [1.8613]

Epoch 3/100:
  Class 1: [2.1499] [4.6993] [4.5852]
  Class 2: [5.5347] [2.6775] [5.4831]
  Class 3: [4.5427] [4.5689] [1.6934]

Epoch 4/100:
  Class 1: [2.8871] [4.4736] [4.5037]
  Class 2: [5.5020] [2.8867] [5.3830]
  Class 3: [4.5897] [4.3582] [1.5482]

Epoch 5/100:
  Class 1: [2.9762] [4.6294] [4.2158]
  Class 2: [5.2000] [3.2244] [5.1990]
  Class 3: [4.2021] [4.5306] [1.2810]
```



```
Bobot Final:
  Class 1: -0.0007 -0.0002 0.0022 1.0029 0.0022 -0.0002 -0.0005 0.0010 0.2528 1.0007 0.0000 1.0007 0.2528 0.0010 0.0010 0.5048 0.4927 -0.000
7 0.4927 0.5048 0.0010 0.2530 0.5012 0.2443 -0.0007 0.2443 0.5012 0.2530 0.2530 0.7463 0.4986 0.4986 0.4986 0.7462 0.2531 0.7566 0.4909 0.24
78 0.2478 0.2478 0.4910 0.7567 0.7566 0.4970 0.2538 0.2538 0.2538 0.4969 0.7567 1.0017 0.2427 0.0000 0.0000 0.0000 0.2427 1.0017 1.0000 0.24
26 -0.0001 -0.0001 -0.0001 0.2426 1.0001
  Class 2: 0.7780 1.0000 1.0002 1.0009 1.0002 0.7855 0.1193 1.0009 -0.0009 -0.0000 0.0000 -0.0000 0.2136 0.7863 1.0009 -0.0009 -0.0001 -0.00
01 -0.0001 0.2239 0.7761 1.0009 0.2239 0.2247 0.2247 0.2247 0.2136 0.5616 0.7788 0.6524 0.6525 0.6525 0.6525 0.8821 -0.0967 1.0008 0.2185 0.
2194 0.2194 0.2194 0.2136 0.5669 1.0008 -0.0975 -0.0966 -0.0966 -0.0966 0.1219 0.7815 1.0008 -0.0009 0.0000 0.0000 0.0000 0.2136 0.7863 0.77
79 1.0966 1.0968 1.0975 1.0968 0.8821 0.1193
  Class 3: 1.0005 0.5569 0.1115 -0.1169 0.1115 0.5569 1.0005 0.5476 0.6732 -0.0005 0.0000 -0.0005 0.6732 0.5476 0.5476 0.6732 0.2200 0.2204
0.2200 0.6732 0.5476 0.5476 0.6729 0.2203 0.2204 0.2203 0.6729 0.5476 0.5476 0.8929 0.6727 0.6727 0.6727 0.8929 0.5478 0.5473 0.6729 0.2231
0.2231 0.2231 0.6729 0.5473 0.5473 0.5569 0.1071 0.1071 0.1071 0.5569 0.5473 0.5471 0.6731 0.0000 0.0000 0.0000 0.6731 0.5471 1.0000 0.6730
0.2277 -0.0001 0.2277 0.6730 1.0000

Pelatihan selesai.
```



```
Hasil pengujian:
Sampel 1:
  Diharapkan: 'H'
  Dengan jarak: [4.7196] [4.6826] [2.7491]
  Hasil Prediksi: 'H'

Sampel 2:
  Diharapkan: 'A'
  Dengan jarak: [3.3709] [5.4805] [3.8666]
  Hasil Prediksi: 'A'

Sampel 3:
  Diharapkan: 'B'
  Dengan jarak: [4.4981] [3.0176] [4.1011]
  Hasil Prediksi: 'B'
```

IV. Analisa

Praktikum ini menggunakan program implementasi sederhana dari algoritma Learning Vector Quantization (LVQ) dalam bahasa C, yang digunakan untuk melakukan klasifikasi data menjadi dua kelas. Dataset terdiri dari 10 data dengan 6 fitur biner, disimpan dalam array dua dimensi `x[10][6]`, sementara array `T[10]` menyimpan label kelas untuk masing-masing data, yaitu kelas 1 dan kelas 2. Bobot awal atau vektor representasi kelas (`w[2][6]`) diinisialisasi dari dua data pertama dalam dataset. Algoritma ini bekerja dengan menghitung jarak Euclidean antara tiap data pelatihan dengan kedua bobot, kemudian menyesuaikan bobot yang paling dekat dengan data tersebut. Jika bobot terdekat mewakili kelas yang sesuai, maka bobot didekati ke data tersebut menggunakan rumus pembaruan `w = w + alpha * (x - w)`. Namun, jika bobot terdekat salah kelas, maka bobot dijauhkan dari data dengan rumus `w = w - alpha * (x - w)`. Nilai `alpha` sebagai learning rate akan menurun 10% setiap epoh untuk memperhalus proses pembelajaran. Setelah proses pelatihan selesai, dilakukan pengujian terhadap sebuah data input baru, yang disimpan pada `x[0]`, dengan menghitung kembali jarak ke masing-masing bobot dan menentukan kelas berdasarkan bobot yang memiliki jarak terdekat. Program ini mencetak jarak, bobot hasil pelatihan, dan prediksi kelas akhir dari data uji tersebut. Pendekatan ini cukup efektif untuk klasifikasi dua kelas sederhana,

V. Kesimpulan

Program ini menunjukkan bagaimana algoritma Learning Vector Quantization (LVQ) dapat diterapkan untuk mengklasifikasikan data biner menjadi dua kelas secara sederhana. Melalui proses pelatihan berulang, bobot (vektor representatif kelas) disesuaikan berdasarkan kedekatannya dengan data pelatihan menggunakan jarak Euclidean dan nilai learning rate yang dikurangi secara bertahap. Hasil akhirnya adalah bobot yang merepresentasikan masing-masing kelas, yang kemudian digunakan untuk mengklasifikasikan data baru.