UNIVERSITAS INDONESIA

Analisis Diagnosis Diabetes Menggunakan Model LVQ-PSO

Tugas Laporan Komputasi Intelegensia

Disusun oleh Kelompok 5:

Ridho Elfapriano Susilo	1906296330
Muhammad Hafidz Agraprana	1906374074
Rachel Octaviani Putri	2006463585
Lutfia Maulidina	2006483151
Miranda Rosely Manullang	2006528710
Safira Raissa Rahmi	2006568891
Marcella Sintauly	2006568916

Dibimbing oleh: Dra. Nora Hariadi, M.Si.

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

PROGRAM STUDI SARJANA MATEMATIKA

DEPOK

DESEMBER 2022

Abstrak

Diabetes adalah kondisi dimana kandungan gula dalam darah melebihi normal dan cenderung tinggi. Penyakit ini ditandai dengan seseorang yang mudah lelah, mengantuk, dan mudah lapar. Tidak hanya orang dewasa, tetapi diabetes juga bisa dialami oleh remaja bahkan anak – anak. Sayangnya, banyak masyarakat yang belum sadar akan bahaya diabetes. Apalagi dengan gaya hidup masyarakat yang cenderung tidak sehat. Dengan mengamati rekam medis, penentuan diagnosis diabetes algoritma *Learning Vector quantiztion* (LVQ). Untuk mengoptimalkan vector bobot awal pada algoritma LVQ, gunakan algoritma *Particle Swarm Optimization* (PSO) untuk mencari vektor bobot awal LVQ terbaik. Untuk mengetahui perbedaan performa, dilakukan pengujian model LVQ-PSO dan LVQ dengan metrik akurasi, presisi, dan *recall*. Hasil pengujian menunjukkan model LVQ-PSO menghasilkan rata-rata akurasi, presisi, dan *recall* tertinggi berturut-turut 75,33%, 65%, dan 55,35% dengan waktu 83,3 detik, sedangkan model LVQ mengasilkan rata-rata akurasi, presisi, dan *recall* tertinggi berturut-turut 75,71%, 65,7%, dan 55,85% dengan waktu 40,2 detik. Parameter-parameter PSO dan LVQ yang digunakan adalahw_{max} 0,6, w_{min} 0,5, banyak partikel 20, maksimal iterasi 100, α 0,1, α_d 0,9. Dari hasil akurasi pengujian tersebut, dapat disimpulkan bahwa algoritma LVQ lebih baik dalam mengklasifikasi diagnosis diabetes daripada LVQ-PSO, tetapi tidak memberikan perbedaan yang signifikan sehingga dapat digunakan algoritma LVQ dengan waktu komputasi yang jauh lebih cepat.

Kata kunci: *Klasifikasi, Diabetes, Learning Vector Quantizer, Particle Swarm Optimization, Optimasi Bobot*.

1. Pendahuluan

1.1 Latar Belakang

Menurut data Kementerian Kesehatan yang diperoleh dari Sample Registration Survey 2014, diabetes menjadi penyebab kematian terbesar nomor 3 di Indonesia dengan persentase sebesar 6,7%, setelah stroke (21,1%), dan penyakit jantung koroner (12,9%). Dari total kasus diabetes, 90% merupakan diabetes tipe 2, dimana umumnya terjadi pada orang dewasa, namun beberapa tahun terakhir juga ditemukan pada anak-anak dan remaja. Diabetes adalah kondisi dimana kandungan gula dalam darah melebihi normal dan cenderung tinggi. Sesuai dengan fakta sebelumnya, diabetes merupakan penyakit

metabolisme yang mampu menyerang siapa saja, tidak hanya orang dewasa. Beberapa gejala diabetes diantaranya adalah kelaparan, keletihan, pandangan yang kabur, dan mati rasa. Diabetes disebabkan karena gaya hidup yang tidak sehat, terlalu banyak konsumsi gula, dan kurangnya aktivitas fisik atau olahraga.

Dari penjabaran ini, maka penting dilakukan klasifikasi untuk menentukan apakah seseorang tergolong menjadi penderita diabetes atau tidak dengan menggunakan faktor-faktor tertentu.

1.2 Definisi Masalah

Penelitian yang dilakukan harapannya dapat menjawab masalah yang didefinisikan sebagai berikut:

1. Masalah yang diajukan pada proyek kali ini adalah Analisis Diagnosis Diabetes menggunakan model LVQ-PSO
2. Dari sudut pandang *machine learning*, masalah analisis diagnosis diabetes menggunakan model LVQ-PSO ini masuk dalam masalah klasifikasi
3. Tipe klasifikasi yang digunakan adalah klasifikasi biner, yaitu antara diabetes (1) atau tidak diabetes (0)

1.3 Tujuan

Tujuan yang ingin dicapai dari penelitian ini adalah sebagai berikut:

1. Mengklasifikasi diagnosis diabetes menggunakan model LVQ-PSO dan LVQ.
2. Mengkaji kemampuan model LVQ-PSO dan LVQ dalam menganalisis diagnosis diabetes.
3. Membandingkan kemampuan model LVQ-PSO dan LVQ dalam menganalisis diagnosis diabetes.

1.4 Metodologi Penelitian

Berikut adalah metodologi dari penelitian ini.

1. Studi Literatur

   Studi literatur dilakukan dari hasil penelitian Awaludin (2017) tentang penggunaan LVQ untuk klasifikasi dan Windi, et al (2018) tentang penggunaan PSO pada LVQ untuk klasifikasi.

2. Pengumpulan Data

   Data yang digunakan pada proyek ini disediakan oleh dosen pengampu mata kuliah Komputasi Intelegensia.

3. Analisis Hasil

   Akan dilihat performa model LVQ-PSO dan LVQ melalui metrik akurasi, presisi, dan recall.

2. Metode

2.1 *Particle Swarm Optimization* (PSO)

Particle Swarm Optimization (PSO) adalah metode optimasi masalah dengan secara iteratif meningkatkan solusi kandidat sehubungan dengan ukuran kualitas (*measure of quality*) yang diberikan. PSO memecahkan masalah dengan memiliki populasi solusi kandidat, yaitu partikel, dan memindahkan partikel-partikel ini di *search-space* sesuai dengan rumus matematika posisi dan kecepatan partikel.

Algoritma optimasi PSO adalah seperti berikut.

a. Inisialisasi partikel secara acak/*random* ($\boldsymbol{x}_i$ untuk $i = 1,2, \dots, N$), iterasi maksimum (T), c_1, c_2, r_1, r_2, ω_{min}, ω_{max}, kecepatan awal setiap partikel, *P-best* awal setiap partikel, *G-best* awal, *fitness function* f

b. While (kriteria pemberhentian belum terpenuhi)

   For (setiap $\boldsymbol{x}_i$) do

   Hitung $f(\boldsymbol{x}_i)$, yaitu *fitness* partikel $\boldsymbol{x}_i$

   Perbarui *P-best* jika $f\big(\boldsymbol{x}_i(t)\big) > f(\boldsymbol{P}_i(t-1))$: $\boldsymbol{P}_i(t) = \boldsymbol{x}_i(t)$

   Perbarui *G-best* jika $f\big(\boldsymbol{P}_i(t)\big) > f(\boldsymbol{P}_i(t-1))$: $\boldsymbol{P}_i(t) = \boldsymbol{x}_i(t)$

   End For

   Perbarui bobot inersia (ω): $\omega(t) = \omega_{max} - \left(\frac{\omega_{max} - \omega_{min}}{T}\right)$

   For (setiap $\boldsymbol{x}_i$) do

   Perbarui kecepatan

   $$\boldsymbol{V}_i(t) = \omega \cdot \boldsymbol{V}_i(t-1) + c_1 r_1\big(\boldsymbol{P}_i(t) - \boldsymbol{x}_i(t-1)\big) + c_2 r_2\big(\boldsymbol{G}(t) - \boldsymbol{x}_i(t-1)\big)$$

   Perbarui Posisi

   $$\boldsymbol{x}_i(t) = \boldsymbol{x}_i(t-1) + \boldsymbol{V}_i(t)$$

   End For

   End While

c. Output: *g-best* terakhir $\boldsymbol{G}(T)$

2.2 Learning Vector Quantizer (LVQ)

Learning Vector Quantizer (LVQ) merupakan salah satu algoritma *clustering* (*unsupervised*), tetapi dapat diadaptasi menjadi algoritma klasifikasi (*supervised*). Unit output pada arsitektur/model LVQ merepresentasikan sebuah kelas.

Secara garis besar, input ke-i akan ditempatkan pada unit output yang memiliki jarak Euclidean terkecil dengan vektor input tersebut. Kemudian, akan dibandingkan apakah kelas yang ditempati berdasarkan jarak Euclidean terkecil sama dengan kelas sesungguhnya dari input ke-i. Setelah itu, akan dilakukan pembaruan bobot pada unit output yang berkaitan berdasarkan kesamaan kelas.

Algoritma LVQ untuk klasifikasi adalah seperti berikut.

a. Inisialisasi bobot unit output ($\boldsymbol{w}_j$ untuk $j = 1,2, \dots, K$), iterasi maksimum (T), learning rate (α), penurun learning rate (α_d), learning rate minimum (α_{min})
b. Input: Dataset $\{\boldsymbol{d}_i, t_i\}_{i=1}^{N}$ dengan K buah kelas
c. While (iterasi maksimum belum tercapai) dan ($\alpha > \alpha_{min}$)

    For (setiap $\boldsymbol{d}_i$) do

        Tentukan unit output J sedemikian sehingga

$$\min_{k \in \{1,2,\dots,K\}} \|\boldsymbol{d}_i - \boldsymbol{w}_k\| = \|\boldsymbol{d}_i - \boldsymbol{w}_J\|, \qquad \|\boldsymbol{d}_i - \boldsymbol{w}_k\| = \sqrt{\sum_{p=1}^{P} \left(d_{ip} - w_{kp}\right)^2}$$

        Kelas $\boldsymbol{d}_i = J$

        If (Kelas $\boldsymbol{d}_i = t_i$) do

$$\boldsymbol{w}_J(t+1) = \boldsymbol{w}_J(t) + \alpha \cdot \left(\boldsymbol{d}_i - \boldsymbol{w}_J(t)\right)$$

        Else

$$\boldsymbol{w}_J(t+1) = \boldsymbol{w}_J(t) - \alpha \cdot \left(\boldsymbol{d}_i - \boldsymbol{w}_J(t)\right)$$

        End If

    End For

    Perbarui learning rate: $\alpha(t+1) = \alpha(t) - \alpha_d \cdot \alpha(t)$

End While

d. Output: Bobot-bobot unit output ($\boldsymbol{w}_j$ untuk setiap $j = 1,2, \dots, K$)

2.3 Penggunaan PSO dan LVQ pada Masalah Klasifikasi

Pada masalah ini, PSO akan digunakan sebagai penentuan bobot awal (optimasi) yang akan di gunakan untuk proses LVQ. LVQ akan digunakan sebagai proses klasifikasi dengan menggunakan bobot awal hasil optimasi algoritma PSO.

3. Deskripsi Data

Data yang di gunakan pada proyek ini berasal dari situs *Kaggle* dengan tautan https://www.kaggle.com/code/omerdersu/diabetes-patient-predict-with-classification/data. Terdapat 768 data berjenis numerik yang terdiri atas delapan atribut dan keluaran berupa diagnosis diabetes. Atribut-atribut pada data adalah seperti berikut.

a. *Blood Pressure*, menyatakan pengukuran tekanan darah
b. *Pregnancies*, menyatakan banyak kehamilan yang pernah dialami
c. BMI, mengekspresikan indeks massa tubuh
d. *Glucose*, menyatakan kadar glukosa dalam darah
e. *Skin Thickness*, menyatakan besar ketebalan kulit
f. Insulin, menyatakan kadar insulin dalam darah
g. *Diabetes Pedigree Function*, menyatakan kecenderungan orang menderita diabetes berdasarkan riwayat genetik keluarga nya
h. *Age*, menyatakan umur

3.1 Pemeriksaan Data

```
[ ]  diabetes_data.isnull().sum()

Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Gambar 1

Fitur	Jumlah Null
Pregnancies	111
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0

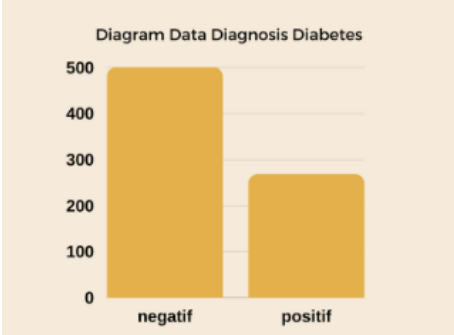
Gambar 2

Dari gambar 1, tidak terdapat *missing value* pada data ini. Namun, terdeteksi banyak data bernilai 0 pada atribut tertentu yang tidak masuk akal jika bernilai 0, yaitu *Glucose, Blood Pressure, Skin Thickness*, Insulin, dan BMI. Banyaknya data yang bernilai 0 pada tiap atribut dapat dilihat pada gambar 2. Oleh karena itu, akan dilakukan prapengolahan data pada bagian 4.

4. Implementasi dan Analisis Hasil

4.1 Pengambilan Data

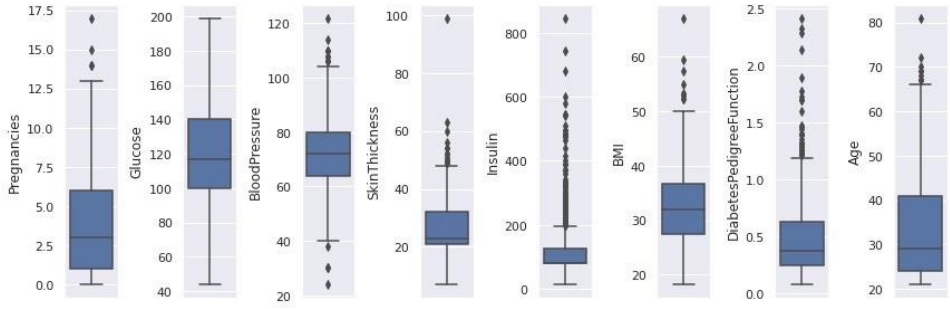
Data yang diambil adalah data terstruktur dengan 8 atribut, dengan target (*outcome*) berupa diabetes (1) atau tidak diabetes (0). Data diambil dari tautan pada deskripsi data di atas. Terdapat 768 data diagnosis yang terdiri dari: diagnosis negatif sebanyak 500 data dan positif sebanyak 268 data.



4.2 Prapengolahan Data

4.2.1 Penggantian Data Bernilai Nol

Terdapat beberapa atribut yang tidak masuk akal jika bernilai nol. Di antaranya adalah *Glucose, Blood Pressure, Skin Thickness*, Insulin, dan BMI. Atribut *Glucose* memiliki 0,6% nilai nol, atribut *Blood Pressure* memiliki 4,5% nilai nol, atribut *Skin Thickness* memiliki 29,5% nilai nol, atribut Insulin memiliki 48,6% nilai nol, dan atribut BMI memiliki 1,4% nilai nol. Presentase banyaknya data bernilai nol dalam atribut-atribut tersebut cukup besar dan terlalu banyak untuk dihapus. Maka dari itu, data bernilai nol dalam atribut-atribut tersebut akan diganti dengan *mean* dari tiap atribut bersangkutan. Berikut *boxplot* dari data setelah diganti nilai nol,



Terlihat pada boxplot bahwa masih banyak *outlier*. Oleh karena itu, akan dilakukan *handling outlier*

4.2.2 *Handling Outlier*

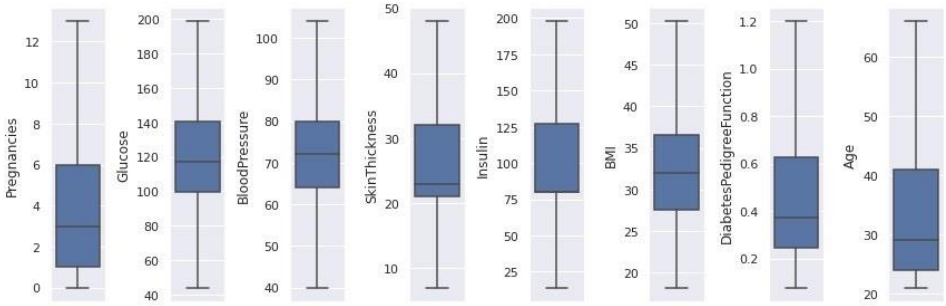
Handling outlier akan dilakukan dengan metode IQR karena semua atribut berjenis numerik. IQR adalah *Interquartile Range* dengan rumus IQR $= Q_3 - Q_1$, dengan Q_3 adalah kuartil ketiga data dan Q_1 adalah kuartil pertama data. Metode IQR dilakukan dengan

mengidentifikasi *outlier* menggunakan rumus batas atas dan batas bawah sebagai berikut untuk setiap atribut:

$$\text{Batas bawah} = Q_1 - 1.5 \cdot \text{IQR}, \ \text{Batas atas} = Q_3 + 1.5 \cdot \text{IQR}$$

Data yang lebih dari batas atas dan kurang dari batas bawah akan dianggap *outlier*. Jadi, nilai yang lebih kecil dari batas bawah akan diganti dengan nilai batas bawah dan nilai yang lebih besar dari batas atas akan diganti dengan batas atas.

Berikut *boxplot* setelah dilakukan *handling outlier*

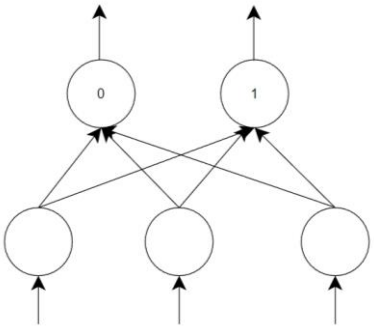


.

Terlihat bahwa sudah tidak ada lagi outlier pada setiap atribut. Maka dari itu, data hasil *handling outlier* akan digunakan untuk klasifikasi.

4.3 Penentuan Partikel PSO

Algoritma PSO akan digunakan untuk menentukan vektor bobot awal LVQ. Oleh karena itu, partikel yang digunakan pada PSO harus merepresentasikan vektor bobot LVQ. Perhatikan arsitektur LVQ berikut.



Arsitektur LVQ di samping memiliki dua unit output dengan vektor bobot untuk masing-masing output adalah

$$\text{Bobot kelas 0: } \boldsymbol{w}_0 = [w_{01}, w_{02}, w_{03}]$$

$$\text{Bobot kelas 1: } \boldsymbol{w}_1 = [w_{11}, w_{12}, w_{13}]$$

Vektor bobot keseluruhan dari arsitektur LVQ dapat ditulis sebagai:

$$\boldsymbol{W} = [\boldsymbol{w}_0 \ \boldsymbol{w}_1] = [w_{01}, w_{02}, w_{03}, w_{11}, w_{12}, w_{13}]$$

Akan dioptimasi vector bobot awal untuk LVQ. Maka dari itu, partikel yang akan dioptimasi berbentuk :

$$x = [x_1, x_2, x_3, x_4, x_5, x_6]$$

dengan $[x_1, x_2, x_3]$ merepresentasikan $\boldsymbol{w}_0$ dan $[x_4, x_5, x_6]$ merepresentasikan $\boldsymbol{w}_1$.

Secara umum, Ketika ada D buah atribut dan K buah kelas, dimensi partikelnya adalah $K \cdot D$ dengan setiap D bagian pada partikel merepresentasikan kelas yang bersesuaian.

4.4 Penentuan *Fitness Function*

Penentuan *fitness function* dilakukan dengan tahap sebagai berikut

1. Ambil data latih untuk optimasi vektor bobot menggunakan PSO;

   Contoh: Berikut data latih yang diambil untuk optimasi vektor bobot.

Glukosa	Insulin	BMI	Kelas
89	96	28,1	0
137	168	43,1	0
78	88	31,0	1

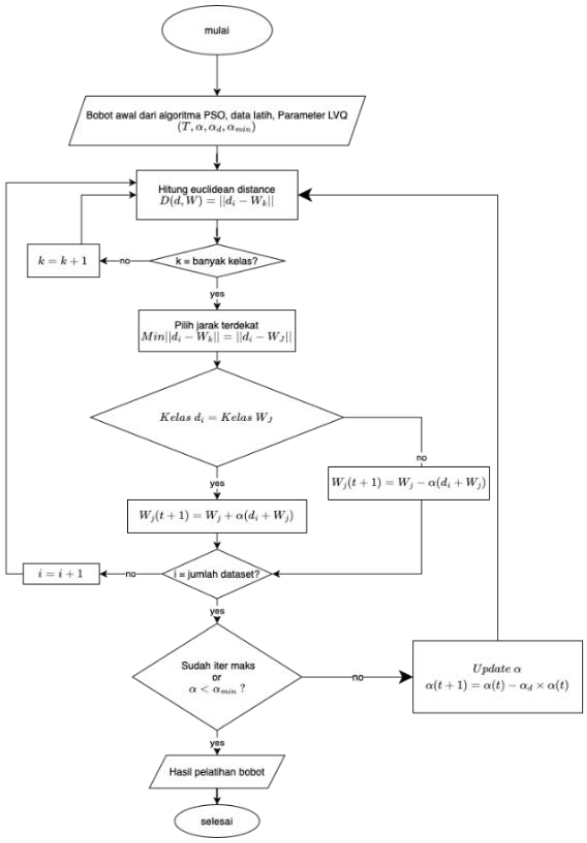
2. Untuk setiap partikel, hitung jarak Euclidean antara setiap data input (pada data latih) dengan vektor bobot setiap kelas yang direpresentasikan pada partikel.

   Contoh: Untuk partikel $\boldsymbol{x}_1 = [99{,}100{,}19.7{,}193{,}130{,}25.9]$ dan data pertama pada data latih, tentukan jarak Euclidean data pertama dengan vektor bobot kelas 0 yang direpresentasikan pada $\boldsymbol{x}_1$, yaitu jarak Euclidean antara $[89{,}94{,}28.1]$ dan $[99{,}100{,}19.7]$. Tentukan pula jarak Euclideannya dengan vektor bobot kelas 1 yang direpresentasikan pada $\boldsymbol{x}_1$

   a. Data input masuk ke kelas yang direpresentasikan pada partikel yang memiliki jarak Euclidean terkecil dengan data input;
   b. Bandingkan hasil kelas target yg diperoleh melalui jarak Euclidean dengan kelas target aslinya (sesuai atau tidak sesuai);

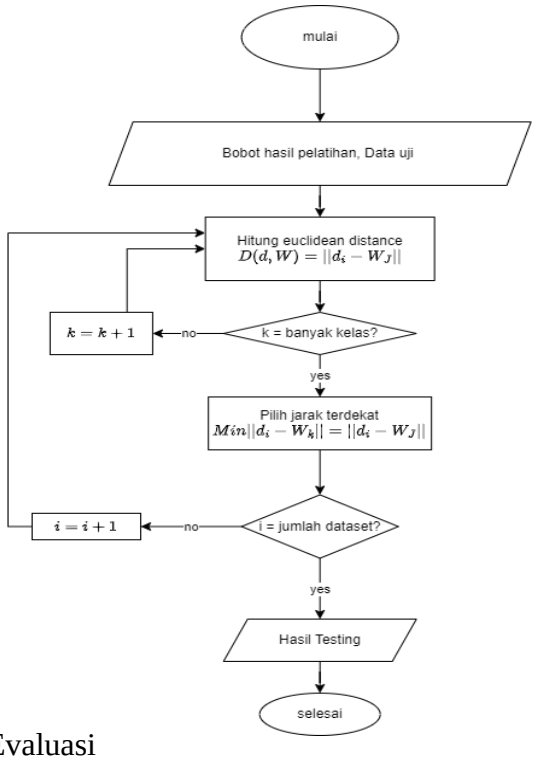
3. Fitness partikel $\boldsymbol{x}_i$ adalah banyak data input yang kelas target prediksinya sesuai dengan kelas target aslinya dibagi dengan ukuran data latih.

4.5 Algoritma Pelatihan LVQ-PSO



Flowchart pelatihan LVQ dapat dilihat pada Gambar di samping. Proses PSO merupakan proses pertama yang berjalan untuk mendapatkan partikel terbaik yang akan dijadikan vektor bobot awal LVQ. Hitung jarak Euclidean dan *update* vektor bobot pada sebanyak data latih. Setelah itu, *update* α. Jika nilai $\alpha \leq \alpha_{min}$, iterasi berhenti. Jika belum, lakukan pengulangan sampai sebanyak maksimal iterasi atau sampai nilai $\alpha \leq \alpha_{min}$ (syarat berhenti terpenuhi). Output dari pelatihan LVQ adalah vektor bobot yang akan digunakan pada proses pengujian.

4.6 Algoritma Pengujian LVQ-PSO



Siklus pengujian LVQ dapat dilihat pada Gambar di samping. Vektor bobot yang diperoleh dari siklus pelatihan LVQ-PSO akan dijadikan vektor bobot awal pada siklus pengujian LVQ-PSO. Hitung jarak Euclidean antara data uji dengan tiap vektor bobot awal dari tiap kelas. Data uji masuk ke kelas yang vektor bobotnya memiliki jarak Euclidean terkecil dengan data uji tersebut. Lakukan pengulangan untuk semua data uji. Output dari pelatihan LVQ adalah hasil prediksi kelas dari tiap data uji.

4.7 Evaluasi

4.7.1 Parameter

Berikut adalah parameter-parameter pada PSO dan LVQ beserta nilai yang digunakan.

Parameter	Nilai	Keterangan
Number of Particle	20	Banyaknya partikel yang dipakai di PSO
Dimension	16	Dimensi partikel
Iteration	100	Banyaknya iterasi PSO
Number of Training	20	Ukuran data latih PSO
w_{min}	0.5	Batas bawah bobot inersia
w_{max}	0.6	Batas atas bobot inersia
c_1	1.5	Koefisien pengaruh *P-best position* tiap partikel
c_2	1.5	Koefisien pengaruh *G-best position*
Maximum iteration	50	Iterasi maksimum LVQ
Learning rate	0.1	*Learning rate* awal
Penurun *learning rate*	0.9	pengurangan *Iearning rate* di tiap iterasi
Learning rate minimum	20	Batas minimum *learning rate*

4.7.2 5-*Cross Validation*

Berikut langkah-langkah dari 5-*Cross Validation* yang dilakukan:

a. Buat vektor bobot awal LVQ dengan PSO sebanyak 50 kali
b. Pilih vektor bobot awal yang tidak memiliki bobot negatif
c. Lakukan splitting data untuk pelatihan dan pengujian dengan rasio 80:20
d. Gunakan tiap bobot awal untuk pelatihan dan pengujian klasifikasi dengan LVQ
e. Diperoleh akurasi, presisi, dan recall untuk setiap bobot awal. Pilih kombinasi akurasi, precision, recall yang memiliki akurasi terbesar

Lakukan prosedur di atas sebanyak 5 kali

4.7.2.1 Evaluasi LVQ-PSO

Berikut adalah hasil 5-*cross validation* dari model LVQ-PSO. (RT: 83,3 detik)

Running	Akurasi	Presisi	Recall
1	0,792208	0,74539	0,568663
2	0,746753	0,608696	0,571429
3	0,74026	0,5	0,575

4	0,753247	0,72973	0,490909
5	0,733766	0,666667	0,561404
Rata – Rata	0,7532468	0,6500966	0,553481
Standar Deviasi	0,020534376	0,089375486	0,03533313

4.7.2.2 Evaluasi LVQ

Berikut adalah hasil *5-cross validation* dari model LVQ. (RT: 40,2 detik)

Running	Akurasi	Presisi	Recall
1	0,785714	0,725	0,568627
2	0,746753	0,608696	0,571429
3	0,746753	0,510638	0,6
4	0,753247	0,72973	0,490909
5	0,753247	0,711111	0,561404
Rata -Rata	0,7571428	0,657035	0,5584738
Standar Deviasi	0,014577818	0,08555321	0,036246209

4.8 Analisis Hasil

Berikut analisis dari hasil performa model LVQ-PSO dan LVQ

Model	Akurasi	Presisi	Recall
LVQ-PSO	0.753247 ± 0.020534	0.650097 ± 0.089375	0.553481 ± 0.035333
LVQ	0.757143 ± 0.014578	0.657035 ± 0.085553	0.558474 ± 0.036246

5. Kesimpulan

Algoritma optimasi PSO dapat digunakan untuk mengoptimasi vektor bobot awal LVQ. Urutan implementasi model LVQ-PSO adalah optimasi vektor bobot awal menggunakan PSO, pelatihan (training) LVQ, dan kemudian pengujian (testing) LVQ. Berdasarkan hasil evaluasi, model LVQ-PSO menghasilkan rata-rata akurasi, presisi, dan *recall* tertinggi berturut-turut 75,33%, 65%, dan 55,35% dengan waktu 83,3 detik, sedangkan model LVQ mengasilkan rata-rata akurasi, presisi, dan recall tertinggi berturut-turut 75,71%, 65,7%, dan 55,85% dengan waktu 40,2 detik. Terlihat bahwa didapat nilai akurasi, presisi, dan recall yang lebih baik oleh model LVQ, tetapi tidak menunjukkan perbedaan yang signifikan dibandingkan dengan model LVQ-PSO. Karena LVQ juga memiliki *running time* yang lebih cepat, dapat dikatakan bahwa model LVQ mengklasifikasi diagnosis diabetes lebih baik daripada model LVQ-PSO.

DAFTAR PUSTAKA

Analisis Akurasi Learning Vector Quantization (LVQ) dan Particle Swarm Optimization. Dongoran, Awaluddin . 2017.

Computational Intelligence, An Introduction. Engelbrecht. Andries, P. John Wiley & Sons, 2007.

Data Clustering using Particle Swarm Optimization. DW van der Merwe. 2003.

Lindungi Keluarga Dari Diabetes (2018) DIREKTORAT PENCEGAHAN DAN PENGENDALIAN PENYAKIT TIDAK MENULAR DIREKTORAT JENDERAL PENCEGAHAN DAN PENGENDALIAN PENYAKIT. Diakses melalui: https://p2ptm.kemkes.go.id/post/lindungi-keluarga-dari-diabetes (Diakses pada: November 30, 2022).

Optimasi Vektor Bobot Pada Learning Vector Quantization Menggunakan Particle Swarm Optimization Untuk Klasifikasi Jenis Attention Deficit Hyperactivity Disorder (ADHD) Pada Anak Usia Dini. Setyowati, Windi A dan Mahmudy, Wayan F. 2018.

Confusion Matrix, Accuracy, Precision, Recall, F1 Score. Harikrishnan N B. Diakses melalui: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd.

LAMPIRAN

Data Exploration

```
from mlxtend.plotting import plot_decision_regions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
diabetes_data = pd.read_csv('../content/diabetes_klasifikasi.csv')
```

```
diabetes_data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

```
diabetes_data.info(verbose=True)
```

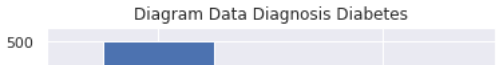
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
diabetes_data.shape
```

```
(768, 9)
```

```
d=diabetes_data.copy()
for i in range(len(d)):
  if d['Outcome'][i]==0:
    d['Outcome'][i]='Negatif'
  else:
    d['Outcome'][i]='Positif'
d['Outcome'].value_counts().plot.bar(rot=0, title='Diagram Data Diagnosis Diabetes',xlabel='Diagnosis Pasien')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbc3829970>
```



Data Preprocessing

```
diabetes_data.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
diabetes_data.duplicated().sum()
```

```
0
```

```
numerical_columns = ['Pregnancies','Glucose','BloodPressure','SkinThickness',
                     'Insulin','BMI','DiabetesPedigreeFunction','Age']
```

```
nul = []
for i in range(len(numerical_columns)):
  bnyk = diabetes_data.loc[diabetes_data[numerical_columns[i]] == 0]
  nul.append(len(bnyk))
data = {'Fitur': numerical_columns,
        'Jumlah Null': nul}  #buat kerangka data frame Y

dataxy = pd.DataFrame(data) #bentuk data frame X dan Y
dataxy.style.hide_index()
```

Fitur	Jumlah Null
Pregnancies	111
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0

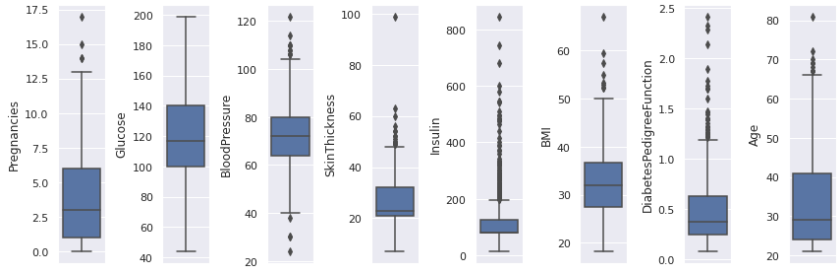
```
ganti_0 = ['Glucose','BloodPressure','SkinThickness',
                     'Insulin','BMI']
```

```
for i in range(len(ganti_0)):
  mean_df = np.mean(diabetes_data[ganti_0[i]])
  if i != 4:
    mean_df = np.ceil(mean_df)
    diabetes_data[ganti_0[i]] = diabetes_data[ganti_0[i]].replace(0,mean_df)
  else:
    mean_df = float(np.ceil(mean_df))
    diabetes_data[ganti_0[i]] = diabetes_data[ganti_0[i]].replace(0.0,mean_df)
```

```
diabetes_data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	80	33.6	0.627	50	1
1	1	85	66	29	80	26.6	0.351	31	0
2	8	183	64	21	80	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...

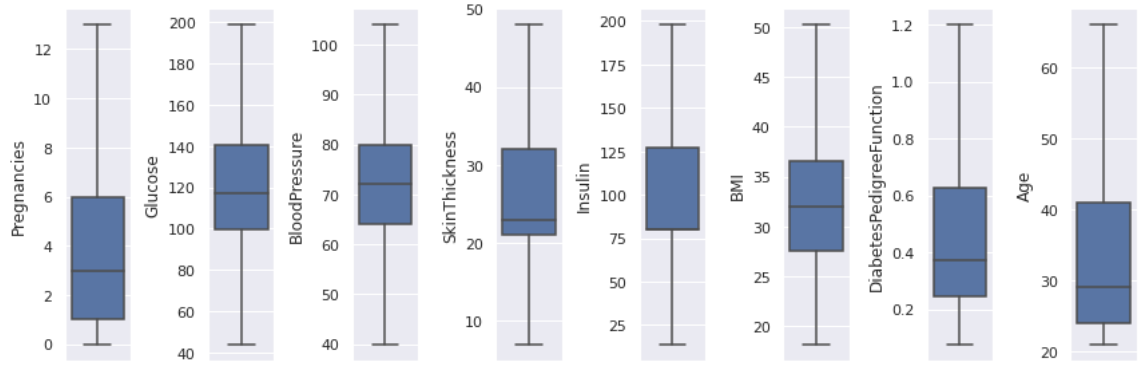
```
plt.figure(figsize=(12,4))
for i in range(0,len(numerical_columns)):
    plt.subplot(1,len(numerical_columns),i+1)
    sns.boxplot(y=diabetes_data[numerical_columns[i]])
    plt.tight_layout()
```




```
for i in range(len(numerical_columns)):
  Q1 = np.quantile(diabetes_data[numerical_columns[i]], .25)
  Q3 = np.quantile(diabetes_data[numerical_columns[i]], .75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR

  #cek data outlier
  outmax=diabetes_data.loc[diabetes_data[numerical_columns[i]] > upper_bound]
  outmin=diabetes_data.loc[diabetes_data[numerical_columns[i]] < lower_bound]
  if len(outmax)!=0:
    for j in range(len(outmax.index)):
      diabetes_data[numerical_columns[i]][outmax.index[j]] = upper_bound
  if len(outmin)!=0:
    for j in range(len(outmin.index)):
      diabetes_data[numerical_columns[i]][outmin.index[j]] = lower_bound
```

```
plt.figure(figsize=(12,4))
for i in range(0,len(numerical_columns)):
    plt.subplot(1,len(numerical_columns),i+1)
    sns.boxplot(y=diabetes_data[numerical_columns[i]])
    plt.tight_layout()
```




```
diabetes_data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPed
0	6	148	72	35	80	33.6	
1	1	85	66	29	80	26.6	
2	8	183	64	21	80	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	80	36.8	
765	5	121	72	23	112	26.2	

```
diabetes_data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.834635	121.682292	72.266927	26.589844	103.944010	32.389063	0.458914	33.194010	0.348958
std	3.336808	30.435999	11.707333	8.969585	46.629117	6.667627	0.285596	11.611715	0.476951
min	0.000000	44.000000	40.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	21.000000	80.000000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	80.000000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	13.000000	199.000000	104.000000	48.000000	198.000000	50.250000	1.200000	66.000000	1.000000

Algoritma PSO

```
import random
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation
from scipy.spatial import distance as dist
```

Men-generate data latih PSO

```
def datalatihPSO(dataset,num):
  idx = random.sample(range(0,len(dataset)),num)
  datalatih = []
  for i in range(num):
    datalatih.append(dataset.loc[idx[i]].values.tolist())
  return datalatih
```

Fitness Function

```
# Fitness function
# data latih
# banyak kelas: 2
# banyak atribut: 6

def fitness(particle,datalatihPSO):
  cost = 0
  yk = len(datalatihPSO)
  dl = len(datalatihPSO[0])-1
  for i in range(yk):
    if dist.minkowski(datalatihPSO[i][0:dl],particle[0:dl], 2) <= dist.minkowski(datalatihPSO[i][0:dl],particle[dl:2*dl], 2):
      if int(datalatihPSO[i][-1]) != 0:
        cost = cost + 1
    else:
      if int(datalatihPSO[i][-1]) != 1:
        cost = cost + 1
  acc = (yk-cost)/yk
  return acc
```

Update Velocity (Kecepatan)

```
def update_velocity(particle, velocity, pbest, gbest, w, c1, c2):
  # Initialise new velocity array
  num_particle = len(particle)
  new_velocity = np.array([0.0 for i in range(num_particle)])
  # Randomly generate r1, r2 and inertia weight from normal distribution
  r1 = 0.5
  r2 = 0.5
  # Calculate new velocity
  for i in range(num_particle):
    new_velocity[i] = w*velocity[i] + c1*r1*(pbest[i]-particle[i])+c2*r2*(gbest[i]-particle[i])
  return new_velocity
```

Update Position (Posisi)

```
def update_position(particle, velocity):
  # Move particles by adding velocity
  new_particle = particle + velocity
  return new_particle
```

Men-generate partikel

```
def get_particle(num_particle,dataset):
  lastcol = dataset.columns[-1]
  dataset = dataset.drop(lastcol, axis = 'columns')
  part = []
  for i in range(num_particle):
    idx = random.sample(range(0,len(dataset)),2)
    ptc = []
    for j in range(2):
      ptc = ptc + dataset.loc[idx[j]].values.tolist()
    part.append(ptc)
  return part
```

Algoritma Utama

```
def PSO(num_part, dim, iter, dataset, num_tr, fitness_criterion, wmin, wmax, c1, c2):
  # Initialisation
  dlatih = datalatihPSO(dataset,num_tr)
  # Population
  particles = get_particle(num_part,dataset)
  # Particle's best position
  pbest_position = particles
  # Fitness
  pbest_fitness = [fitness(p,dlatih) for p in particles]
  # Index of the best particle
  gbest_index = np.argmax(pbest_fitness)
  # Global best particle position
  gbest_position = pbest_position[gbest_index]
  # Velocity (starting from 0 speed)
  velocity = [[0.0 for j in range(dim)] for i in range(num_part)]

  # Loop for the number of generation
  for t in range(iter):
    # Stop if the average fitness value reached a predefined success criterion
    if np.average(pbest_fitness) >= fitness_criterion:
      break
    else:
      w = wmax - ((wmax - wmin)/(iter))*t
      for n in range(num_part):
        # Update the velocity of each particle
        velocity[n] = update_velocity(particles[n], velocity[n], pbest_position[n], gbest_position, w ,c1, c2)
        # Move the particles to new position
        particles[n] = update_position(particles[n], velocity[n])
    # Calculate the fitness value
    pbest_fitness = [fitness(p,dlatih) for p in particles]
    # Find the index of the best particle
    gbest_index = np.argmax(pbest_fitness)
    # Update the position of the best particle
    gbest_position = pbest_position[gbest_index]
```

```
  # Print the results
  return gbest_position.tolist(), max(pbest_fitness)
```

Coba-coba

```
# PSO(num_part, dim, iter, dataset, num_tr, fitness_criterion, wmin, wmax, c1, c2)
salin = diabetes_data.copy()
a = PSO(20, 16, 100, salin, 20, 0.9, 0.5, 0.6, 1.5, 1.5)
a
```

```
     ([3.1323847103518574,
       91.05297551019757,
       34.670696744374204,
       22.61995827163706,
       87.12463457590853,
       26.13213032053955,
       0.8721764966496548,
       25.254950728122427,
       -5.711563829074178,
       99.38094981501311,
       75.02626393127167,
       31.246659158287343,
       122.439384629355,
       19.360423158000156,
       -0.11241314820837617,
       35.87920801281211],
      0.9)
```

Algoritma LVQ

Pelatihan Klasifikasi dengan LVQ

```
def LVQtr(dtratr, dtrlabel,itmax, lrate, declrate, minlrate, bwl):
  it = 0
  dtratr.reset_index(drop=True, inplace = True)
  dtrlabel.reset_index(drop=True, inplace = True)
  b0, b1 = bwl[0:int(len(bwl)/2)], bwl[int(len(bwl)/2):len(bwl)]
  while it <= itmax and lrate > minlrate:
    for i in range(len(dtratr)):
      dtlist = dtratr.loc[i].values.tolist()
      if dist.minkowski(dtlist,b0, 2) <= dist.minkowski(dtlist,b1, 2):
        kls = 0
        if kls == dtrlabel[i]:
          for j in range(len(b0)):
            b0[j] = b0[j] + lrate*(dtlist[j]-b0[j])
        else:
          for j in range(len(b0)):
            b0[j] = b0[j] - lrate*(dtlist[j]-b0[j])
      else:
        kls = 1
        if kls == dtrlabel[i]:
          for j in range(len(b1)):
            b1[j] = b1[j] + lrate*(dtlist[j]-b1[j])
        else:
          for j in range(len(b1)):
            b1[j] = b1[j] - lrate*(dtlist[j]-b1[j])
    lrate = lrate - declrate*lrate
    it += 1
  return b0, b1
```

Pengujian Klasifikasi dengan LVQ

```
def LVQts2(dtsatr, dtslabel,bts):
  dtsatr.reset_index(drop=True, inplace = True)
  dtslabel.reset_index(drop=True, inplace = True)
  bobot0, bobot1 = bts[0], bts[1]
  klspred = []
  cost = 0
  for i in range(len(dtsatr)):
    dtslist = dtsatr.loc[i].values.tolist()
    if dist.minkowski(dtslist,bobot0, 2) <= dist.minkowski(dtslist,bobot1, 2):
      kls = 0
    else:
```

```
      kls = 1
    klspred.append(kls)
  return klspred
```

```
def confmatrix(dtslabellist, pred):
  banding = {'Benar': klsbenar, 'Prediksi': pred}
  bandingdf = pd.DataFrame(banding)
  confmix = pd.crosstab(bandingdf['Benar'], bandingdf['Prediksi'], rownames=['True'], colnames=['Pred'])
  lst = confmix.values.tolist()
  TN = lst[0][0]
  FP = lst[0][1]
  FN = lst[1][0]
  TP = lst[1][1]
  acc = (TN+TP)/(TN+FP+FN+TP)
  pres = (TP)/(TP+FP)
  rec = (TP)/(TP+FN)
  return [acc, pres, rec]
```

5-Cross Validation

```
from sklearn.model_selection import train_test_split
```

Random State = 101

Bobot Awal

```
bawal = []
fness = []
salin2 = diabetes_data.copy()

for i in range(50):
  a = PSO(20, 16, 100, salin2, 20, 0.8, 0.5, 0.6, 1.5, 1.5)
  ind = 0
  for j in range(len(a[0])):
    if a[0][j] < 0:
      ind = 1
      break
  if ind == 1:
    continue
  else:
    bawal.append(a[0])
    fness.append(a[1])
```

Splitting data

```
X = diabetes_data.drop('Outcome', axis=1) # Independent Values
Y = diabetes_data['Outcome'] # Targeted Values Selection
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)
```

LVQ

```
eval = []
for i in range(len(bawal)):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, bawal[i])
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval.append(ev)
dframe = pd.DataFrame(eval, columns = ['Akurasi','Presisi','Recall'])
dframe.loc[dframe.idxmax()[0]]
```

```
Akurasi    0.792208
Presisi    0.743590
Recall     0.568627
Name: 2, dtype: float64
```

```
eval2 = []
for i in range(25):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  idx = random.sample(range(0,len(salin2)),2)
  botwl = []
  for j in range(2):
    botwl = botwl + salin2.loc[idx[j]].values.tolist()
    botwl.pop(-1)
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, botwl)
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval2.append(ev)
dframe2 = pd.DataFrame(eval2, columns = ['Akurasi','Presisi','Recall'])
dframe2.loc[dframe2.idxmax()[0]]
```

```
Akurasi    0.785714
Presisi    0.725000
Recall     0.568627
Name: 0, dtype: float64
```

Random State = 27

Bobot Awal

```
bawal = []
fness = []
salin2 = diabetes_data.copy()

for i in range(50):
  a = PSO(20, 16, 100, salin2, 20, 0.8, 0.5, 0.6, 1.5, 1.5)
  ind = 0
  for j in range(len(a[0])):
    if a[0][j] < 0:
      ind = 1
      break
  if ind == 1:
    continue
  else:
    bawal.append(a[0])
    fness.append(a[1])
```

Splitting data

```
X = diabetes_data.drop('Outcome', axis=1) # Independent Values
Y = diabetes_data['Outcome'] # Targeted Values Selection
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=27)
```

LVQ

```
eval = []
for i in range(len(bawal)):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, bawal[i])
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
```

```
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval.append(ev)
dframe = pd.DataFrame(eval, columns = ['Akurasi','Presisi','Recall'])
dframe.loc[dframe.idxmax()[0]]
```

```
Akurasi    0.746753
Presisi    0.608696
Recall     0.571429
Name: 3, dtype: float64
```

```
eval2 = []
for i in range(25):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  idx = random.sample(range(0,len(salin2)),2)
  botwl = []
  for j in range(2):
    botwl = botwl + salin2.loc[idx[j]].values.tolist()
    botwl.pop(-1)
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, botwl)
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval2.append(ev)
dframe2 = pd.DataFrame(eval2, columns = ['Akurasi','Presisi','Recall'])
dframe2.loc[dframe2.idxmax()[0]]
```

```
Akurasi    0.746753
Presisi    0.608696
Recall     0.571429
Name: 3, dtype: float64
```

Random State = 56

Bobot Awal

```
bawal = []
fness = []
salin2 = diabetes_data.copy()

for i in range(50):
  a = PSO(20, 16, 100, salin2, 20, 0.8, 0.5, 0.6, 1.5, 1.5)
  ind = 0
  for j in range(len(a[0])):
    if a[0][j] < 0:
      ind = 1
      break
  if ind == 1:
    continue
  else:
    bawal.append(a[0])
    fness.append(a[1])
```

Splitting data

```
X = diabetes_data.drop('Outcome', axis=1) # Independent Values
Y = diabetes_data['Outcome'] # Targeted Values Selection
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=56)
```

LVQ

```
eval = []
for i in range(len(bawal)):
  xtr = X_train.copy()
  ytr = Y_train.copy()
```

```
  xts = X_test.copy()
  yts = Y_test.copy()
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, bawal[i])
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval.append(ev)
dframe = pd.DataFrame(eval, columns = ['Akurasi','Presisi','Recall'])
dframe.loc[dframe.idxmax()[0]]
```

```
Akurasi    0.74026
Presisi    0.50000
Recall     0.57500
Name: 0, dtype: float64
```

```
eval2 = []
for i in range(25):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  idx = random.sample(range(0,len(salin2)),2)
  botwl = []
  for j in range(2):
    botwl = botwl + salin2.loc[idx[j]].values.tolist()
    botwl.pop(-1)
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, botwl)
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval2.append(ev)
dframe2 = pd.DataFrame(eval2, columns = ['Akurasi','Presisi','Recall'])
dframe2.loc[dframe2.idxmax()[0]]
```

```
Akurasi    0.746753
Presisi    0.510638
Recall     0.600000
Name: 3, dtype: float64
```

Random State = 72

Bobot Awal

```
bawal = []
fness = []
salin2 = diabetes_data.copy()

for i in range(50):
  a = PSO(20, 16, 100, salin2, 20, 0.8, 0.5, 0.6, 1.5, 1.5)
  ind = 0
  for j in range(len(a[0])):
    if a[0][j] < 0:
      ind = 1
      break
  if ind == 1:
    continue
  else:
    bawal.append(a[0])
    fness.append(a[1])
```

Splitting data

```
X = diabetes_data.drop('Outcome', axis=1) # Independent Values
Y = diabetes_data['Outcome'] # Targeted Values Selection
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=72)
```

LVQ

```
eval = []
for i in range(len(bawal)):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, bawal[i])
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval.append(ev)
dframe = pd.DataFrame(eval, columns = ['Akurasi','Presisi','Recall'])
dframe.loc[dframe.idxmax()[0]]
```

```
Akurasi    0.753247
Presisi    0.729730
Recall     0.490909
Name: 2, dtype: float64
```

```
eval2 = []
for i in range(25):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  idx = random.sample(range(0,len(salin2)),2)
  botwl = []
  for j in range(2):
    botwl = botwl + salin2.loc[idx[j]].values.tolist()
    botwl.pop(-1)
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, botwl)
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval2.append(ev)
dframe2 = pd.DataFrame(eval2, columns = ['Akurasi','Presisi','Recall'])
dframe2.loc[dframe2.idxmax()[0]]
```

```
Akurasi    0.753247
Presisi    0.729730
Recall     0.490909
Name: 3, dtype: float64
```

Random State = 7

Bobot Awal

```
bawal = []
fness = []
salin2 = diabetes_data.copy()

for i in range(50):
  a = PSO(20, 16, 100, salin2, 20, 0.8, 0.5, 0.6, 1.5, 1.5)
  ind = 0
  for j in range(len(a[0])):
    if a[0][j] < 0:
      ind = 1
      break
  if ind == 1:
    continue
  else:
    bawal.append(a[0])
    fness.append(a[1])
```

Splitting data

```
X = diabetes_data.drop('Outcome', axis=1) # Independent Values
Y = diabetes_data['Outcome'] # Targeted Values Selection
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=7)
```

LVQ

```
eval = []
for i in range(len(bawal)):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, bawal[i])
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval.append(ev)
dframe = pd.DataFrame(eval, columns = ['Akurasi','Presisi','Recall'])
dframe.loc[dframe.idxmax()[0]]
```

```
Akurasi    0.727273
Presisi    0.659574
Recall     0.543860
Name: 1, dtype: float64
```

```
eval2 = []
for i in range(25):
  xtr = X_train.copy()
  ytr = Y_train.copy()
  xts = X_test.copy()
  yts = Y_test.copy()
  idx = random.sample(range(0,len(salin2)),2)
  botwl = []
  for j in range(2):
    botwl = botwl + salin2.loc[idx[j]].values.tolist()
    botwl.pop(-1)
  bbt = LVQtr(xtr, ytr, 50, 0.1, 0.9, 10e-16, botwl)
  evmod = LVQts2(xts, yts, bbt)
  if (1 in evmod) == False or (0 in evmod) == False:
    continue
  else:
    klsbenar = yts.values.tolist()
    ev = confmatrix(klsbenar, evmod)
    eval2.append(ev)
dframe2 = pd.DataFrame(eval2, columns = ['Akurasi','Presisi','Recall'])
dframe2.loc[dframe2.idxmax()[0]]
```

```
Akurasi    0.753247
Presisi    0.711111
Recall     0.561404
Name: 20, dtype: float64
```