APLIKASI DIAGNOSIS AWAL PENYAKIT PADA SISTEM URIN MENGGUNAKAN LVQ (*LEARNING VECTOR QUANTIZATION*)

Evaliata Br. Sembiring, Luh Arida Ayu Rahning Putri

Program Studi S2 Ilmu Komputer
Universitas Gadjah Mada Yogyakarta

evaliata@yahoo.com; ariebat@yahoo.com

Abstrak

Salah satu langkah penting dalam mengetahui penyakit yang menyerang organ tubuh manusia adalah dengan melakukan diagnosis awal berdasarkan gejala-gejala yang dialami. Berdasarkan gejala-gejala tersebut dapat diklasifikasi jenis penyakit yang sesuai atau di bidang kedokteran lebih familiar dengan istilah diagnosis penyakit. Dalam penelitian ini terdapat enam jenis gejala yang digunakan untuk melakukan proses klasifikasi penyakit yang menyerang sistem urin (Urinary System Diseases). Penyakit yang diklasifikasi merupakan dua jenis penyakit yaitu Inflammation of urinary bladder dan Nephritis of renal pelvis origin. Berdasarkan enam gejala tersebut dapat dilakukan diagnosis awal, apakah seorang pasien menderita salah satu dari kedua penyakit, menderita keduanya atau bahkan tidak mengalami kedua penyakit tersebut.

Dalam melakukan diagnosis awal penyakit pada sistem urin ini, digunakan sebuah metode jaringan syaraf tiruan. Metode yang dapat menyelesaikan masalah-masalah yang terkait dengan klasifikasi yaitu LVQ (Learning Vector Quantization). Implementasi sistem diagnosis ini menggunakan bahasa pemrograman Delphi 7 dan penyimpanan data-data yang digunakan seperti bobot, data pelatihan dan data pengujiannya menggunakan Microsoft Access. Hasil penelitian yang diperoleh bahwa persentase tingkat akurasi pelatihan dan pengujian dengan rata-rata akurasinya di atas 90% bahkan dapat mencapai 100% untuk learning rate =0.1, dengan delta learning rate = 0.0001 dan kondisi berhenti = 0. Selain itu hasil diagnosis yang diperoleh cukup valid.

Kata Kunci: Sistem urin, diagnosis, LVQ.

1. Pendahuluan

Sistem urin adalah sistem organ yang memproduksi, menyimpan, dan mengalirkan urin. Pada manusia, sistem ini terdiri dari dua ginjal, dua *ureter*, kandung kemih, dua otot *sphincter*, dan uretra (Wikipedia, 2013). Dari beberapa bagian organ pada sistem urin tersebut, sering diserang penyakit-penyakit tertentu. Penyakit yang diklasifikasikan dalam penelitian ini merupakan dua jenis penyakit yang menyerang sistem urin (*Urinary System Diseases*), yaitu *Inflammation of urinary bladder* dan *Nephritis of renal pelvis origin* (UCI, tanpa tahun). *Inflammation of urinary bladder* adalah peradangan/infeksi atau pembengkakan pada kandung kemih sedangkan *Nephritis of renal pelvis* origin adalah peradangan pada ginjal yang sumbernya dari bagian pelvis pada ginjal.

Proses klasifikasi penyakit dilakukan berdasarkan pada enam jenis gejala (*symptom*), yaitu:

a. *Temperature of patient*: suhu atau temperatur pasien
b. *Occurrence of nausea*: timbulnya rasa mual
c. *Lumbar pain*: nyeri pada pinggang
d. *Urine pushing* (*continuous need for urination*): keinginan untuk buang air kecil secara terus-menerus
e. *Micturition pains*: nyeri saat buang air kecil
f. *Burning of urethra*, *itch*, *swelling of urethra outlet*: pembakaran uretra, gatal dan pembengkakan saluran uretra

Berdasarkan enam gejala tersebut dapat dilakukan diagnosis awal apakah seorang pasien menderita salah satu dari kedua penyakit tersebut, menderita keduanya atau bahkan tidak mengalami kedua penyakit tersebut. Aplikasi ini dibuat untuk melakukan klasifikasi atau diagnosa awal terhadap dua jenis penyakit pada sistem urin dengan menggunakan metode *Learning Vector Quantization* (LVQ).

2. Landasan Teori

2.1 LVQ (*Learning Vector Quantization*)

LVQ merupakan salah satu metode dalam jaringan syaraf tiruan yang bisa digunakan untuk penyelesaian masalah klasifikasi. Pada proses klasifikasi setiap unit keluaran mempresentasikan sebuah kelas atau target (Fausett, 1994). Jumlah kelas atau kelompok sebagai target sudah ditentukan sebelumnya.

LVQ sebagai salah satu jaringan syaraf tiruan merupakan algoritma pembelajaran kompetitif terawasi yang merupakan versi dari algoritma *Kohonen Self-Organization Map* (SOM) (Fausett, 1994). Algoritma ini bertujuan untuk mendekati distribusi kelas vektor sehingga dapat mengurangi kesalahan dalam pengklasifikasian.

LVQ melakukan pembelajaran pada lapisan kompetitif yang terawasi (Ranaldhi, 2006). Suatu lapisan kompetitif akan secara otomatis belajar untuk mengklasifikasikan vektor-vektor *input*. Kelas-kelas yang diperoleh sebagai hasil dari lapisan kompetitif ini hanya tergantung pada jarak antara vektor-vektor *input*. Jika 2 (dua) vektor *input* mendekati sama, maka lapisan kompetitif akan meletakkan kedua vektor *input* tersebut ke dalam kelas yang sama, dengan kata lain jaringan LVQ belajar mengklasifikasikan vektor masukan ke kelas target yang ditentukan oleh pengguna.

2.2 Arsitektur LVQ

Learning Vector Quantization (LVQ) merupakan jaringan lapisan tunggal (*single-layer net*) di mana lapisan masukan terkoneksi secara langsung dengan setiap neuron pada keluaran. Koneksi antar neuron tersebut dihubungkan dengan bobot/*weight*. Neuron-neuron keluaran pada LVQ menyatakan suatu kelas atau kategori tertentu. Bobot merupakan nilai matematis dari koneksi yang mentransfer data dari satu lapisan ke lapisan lainnya, yang berfungsi untuk mengatur jaringan sehingga dapat menghasilkan *output* yang diinginkan. Bobot pada LVQ sangat penting, karena dengan bobot ini *input* dapat melakukan pembelajaran dalam mengenali suatu pola. Vektor bobot berfungsi untuk menghubungkan setiap neuron pada lapisan *input* dengan masing-masing neuron pada lapisan *output*.

2.3 Parameter LVQ

Secara umum parameter-parameter yang digunakan pada metode LVQ adalah sebagai berikut:

1. *Alpha* (α) – Parameter ini lebih umum disebut dengan *Learning Rate*. *Alpha* didefinisikan sebagai tingkat pembelajaran. Jika *alpha* terlalu kecil, maka prosesnya akan terlalu lama. Nilai alpha adalah $0 < \alpha < 1$.
2. *Decrement Alpha* ($\Delta\alpha$) – Parameter ini merupakan penurunan tingkat pembelajaran.
3. *Minimal Alpha* (parameter untuk kondisi berhenti, dimana pada aplikasi variabel diberi nama "*stop*") – Parameter ini merupakan minimal nilai tingkat pembelajaran yang masih diperbolehkan.

2.4 Algoritma Pembelajaran LVQ

Algoritma pembelajaran LVQ bertujuan untuk mencari nilai bobot yang sesuai untuk mengelompokkan vektor-vektor ke dalam kelas tujuan yang telah diinisialisasi pada saat pembentukan jaringan LVQ. Sedangkan algoritma pengujiannya adalah untuk menghitung nilai *output* (kelas vektor) yang terdekat dengan vektor *input*, atau dapat disamakan dengan proses pengklasifikasian (pengelompokan). Berikut ini adalah algoritma pembelajaran LVQ:

Langkah 0: inisialisasi vektor referensi (sebagai bobot awal: $x1$, $x2$, ..., $x6$), *learning rate* (α), *decrement alpha* ($\Delta\alpha$), kondisi berhenti, dan jumlah kluster yang diinginkan.

Langkah 1: ketika kondisi berhenti (*stop*) adalah bernilai salah, maka lakukan langkah 2 sampai 6.

Langkah 2: untuk setiap input pelatihan vektor x, lakukan langkah 3 dan 4.

Langkah 3: temukan jarak terpendek (minimum) xi dengan bobot yang ditandai dengan indeks vektor bobot sebagai j.

Langkah 4: perbaharui bobot w sebagai berikut:

Jika *class reference* = *class input*,

maka w(baru) = w(lama) + $\alpha[xi\text{-}w(\text{lama})]$

Jika *class reference* ≠ *class input*,

maka w(lama) - $\alpha[xi\text{-}w(\text{lama})]$

Langkah 5: kurangi *Learning Rate*.

Langkah 6: tes kondisi berhenti (*stop*),

yaitu kondisi yang mungkin menetapkan sebuah jumlah tetap dari iterasi atau rating pembelajaran mencapai nilai kecil yang cukup.

Setelah dilakukan pelatihan, akan diperoleh bobot-bobot akhir (w). Bobot-bobot ini selanjutnya digunakan untuk melakukan pengujian dalam hal ini adalah untuk diagnosis jenis penyakit. Algoritma pengujian LVQ adalah sebagai berikut:

Langkah 1: masukkan input (xi) dan bobot akhir (hasil *training*).

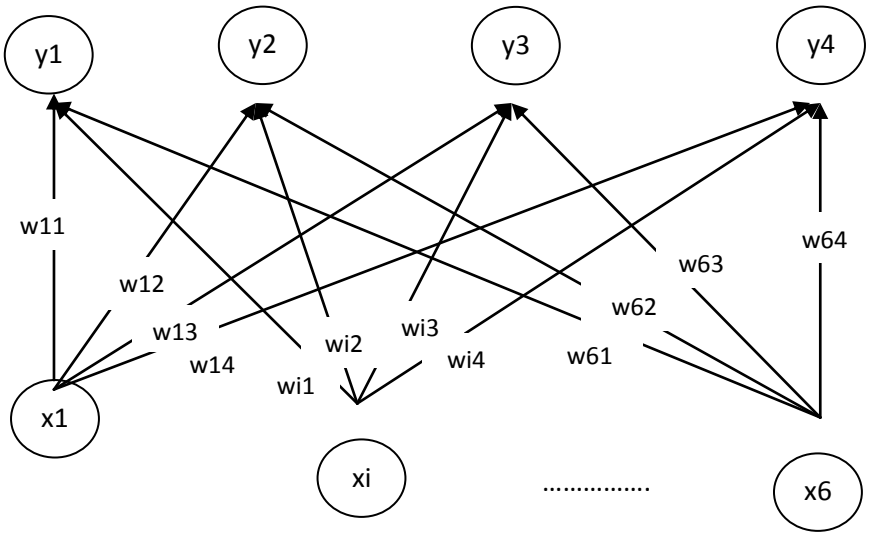
Langkah 2: hitung jarak

Langkah 3: pilih neuron dengan jarak minimum

Langkah 4: memberikan hasil klasifikasi (sebagai diagnosis awal penyakit).

3. Perancangan dan Implementasi

3.1 Perancangan Arsitektur

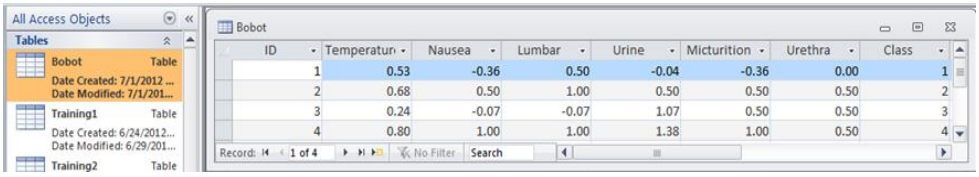


Gambar 1 Arsitektur LVQ Diagnosis Awal Penyakit pada Sistem Urin

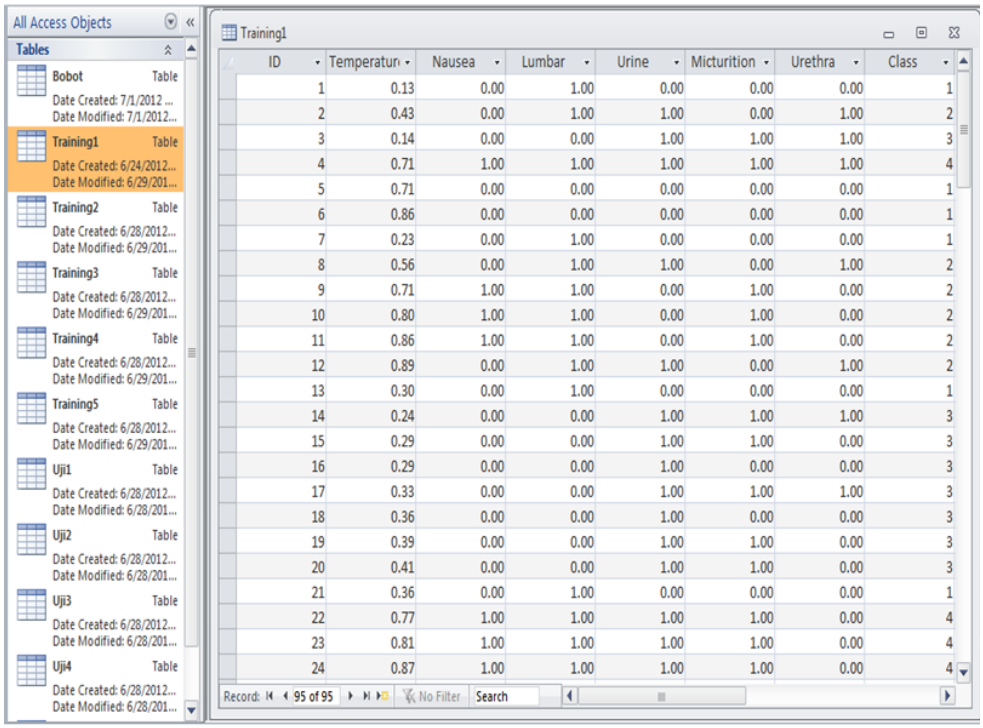
Langkah awal yang dilakukan untuk membangun aplikasi diagnosis awal penyakit pada sistem urin ini adalah merancang arsitekturnya. Secara umum dapat ditetapkan *input* dan *output* yang mempengaruhi proses klasifikasi untuk mendiagnosis penyakit. Terdapat enam variabel dari vektor *input* yaitu gejala yang disimbolkan dengan x = ($x1$, $x2$, $x3$, $x4$, $x5$, $x6$) dan empat jenis diagnosis yang akan menjadi keluarannya yang disimbolkan dengan kelas y = ($y1$, $y2$, $y3$, $y4$). Selain itu terdapat empat vektor bobot yang diwakili oleh variabel w. Arsitektur tersebut ditunjukkan pada Gambar 1.

3.2 Normalisasi *Data Set*

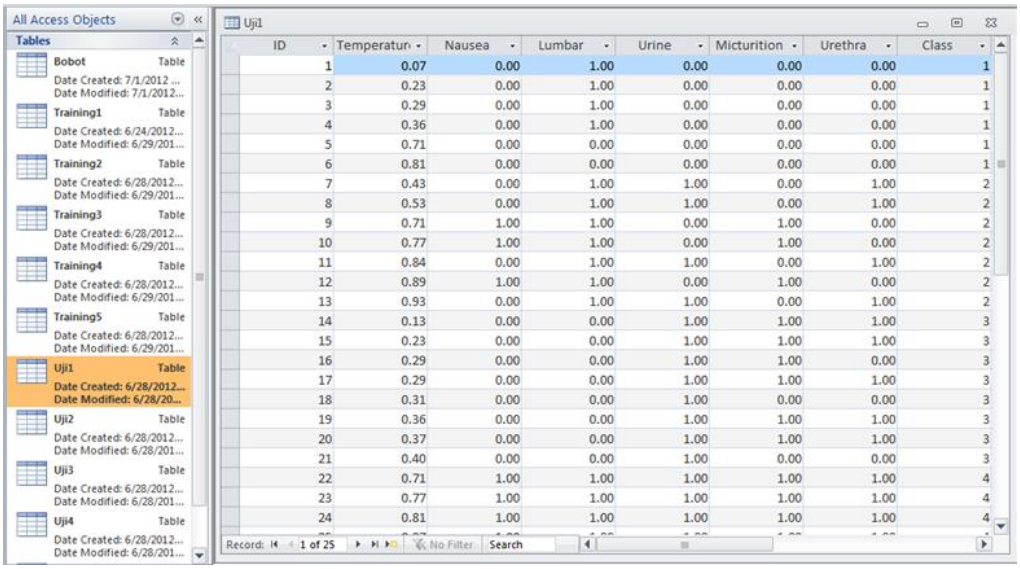
Data gejala yang diperoleh hanya bernilai ”*yes*” dan ”*no*” sehingga sebelum disimpan pada *database* terlebih dahulu dinormalisasi dengan mengubah nilai ”*yes* menjadi 1” dan nilai ”*no* menjadi 0”. Data seluruhnya yang digunakan berjumlah 120 data. Data tersebut dibagi menjadi dua kelompok yaitu data pelatihan dan data pengujian.



Gambar 2(a) Tabel *Update* Bobot



Gambar 2(b) Tabel Data Pelatihan



Gambar 2(c) Tabel Data Pengujian

Masing-masing data disimpan dalam tabel sehingga terdapat 5 tabel untuk data pelatihan (diberi nama *Training1*, *Training2*, *Training3*, *Training4*, *Training5*) dan 5 tabel untuk data pengujian (diberi nama *Uji1*, *Uji2*, *Uji3*, *Uji4*, *Uji5*). Selanjutnya bobot akhir dari pelatihan juga disimpan dalam satu tabel yaitu tabel Bobot. Sehingga terdapat 11 tabel yang digunakan dalam Sistem Diagnosis Awal Penyakit pada Sistem Urin ini dimana masing-masing tabel diwakili oleh satu tabel seperti ditunjukkan pada Gambar 2.

3.3 Proses Pembelajaran

Implementasi algoritma pembelajaran LVQ dijelaskan mulai dari inisialisasi bobot vektor yaitu dengan mengambil vektor pelatihan pertama dan menggunakannya sebagai bobot vektor. Vektor yang tersisa digunakan untuk pelatihan. Selain vektor bobot, beberapa parameter yang diinisialisasi adalah *learning rate*, perubahan *learning rate* dan kondisi berhenti pelatihan. Berikut ini implementasinya:

```
for i := 1 to 4 do begin // mengambil empat vektor
   for j := 0 to 7 do begin // mengambil 6 vektor input
      GridBobot.Cells[j, i] := GridData.Cells[j, i];
   end;
end;

// nilai parameter dimasukkan oleh user
alpha := StrToFloat(edtLR.Text); // learning rate
delta := StrToFloat(edtDelta.Text); // perubahan learning rate
stop := StrToFloat(edtStop.Text); // kondisi berhenti
```

Langkah berikutnya adalah menemukan jarak minimum vektor *input* dengan bobot. Hal ini dilakukan dengan tujuan untuk merubah bobot.

```
// step 1 kondisi berhenti
epoch := 0;
while alpha >= stop do begin
   // step 2
   for i := 5 to jmldata do begin
      // step 3: hitung jarak data training
      //         ke masing-masing vector bobot
      for j := 1 to 4 do begin
         jarak1 := 0;
         for m := 1 to 6 do begin
            jarak1 := jarak1 + Sqr(
               StrToFloat(GridData.Cells[m, i]) -
               StrToFloat(GridBobot.Cells[m, j])
            );
         end;
         jarak2 := RoundTo(Sqrt(jarak1), -2);
         d[j] := jarak2;
      end;

      // step 4: cari jarak minimum kemudian update bobot
      minD := MinValue(d); // fungsi minimum

      k := 1;
      while k < 5 do begin
         if MinD = d[k] then begin
            indeksD := k; // ambil indeks jarak minimum
            k := 5;
         end
         else k := k + 1;
      end;
```

Ketika jarak minimum sudah ditemukan maka dilakukan perubahan bobot, dengan memeriksa kondisinya terlebih dahulu, dimana vektor *input* sama dengan vektor bobot maka bobot baru ditambah dengan hasil perkalian *learning rate* dengan selisih vektor *input* dan vektor bobot yang lama. Sebaliknya jika tidak sama, maka dilakukan pengurangan.

```
      // update bobot
      // cek class reference dengan class input sama atau tidak
      // jika sama bobot ditambah

      if GridBobot.Cells[7, indeksD] = GridData.Cells[7, i] then
      begin
         edtTrainAcr.Text := '';
         for n := 1 to 6 do begin
            tempB := StrToFloat(GridBobot.Cells[n, indeksD]);
```

```
            tempD := StrToFloat(GridData.Cells[n, i]);
            updB := alpha * (tempD - tempB);
            tempB := RoundTo((tempB + updB), -2);

            GridBobot.Cells[n,indeksD] :=
               FormatFloat('0.00', tempB);
         end;
      end
      // jika berbeda, bobot dikurangi
      else begin
         edtTrainAcr.Text := '';
         for n := 1 to 6 do begin
            tempB := StrToFloat(GridBobot.Cells[n, indeksD]);
            tempD := StrToFloat(GridData.Cells[n, i]);
            updB := alpha * (tempD - tempB);
            tempB := RoundTo((tempB - updB), -2);

            GridBobot.Cells[n, indeksD] :=
               FormatFloat('0.00', tempB);
         end;
      end;
   end;
```

Setelah perubahan bobot dilakukan, maka *learning rate* dikurangi. Sampai suatu saat kondisi berhenti akan bernilai benar yaitu kondisi yang mungkin menetapkan sebuah jumlah tetap dari iterasi atau *rating* pembelajaran mencapai nilai kecil yang cukup.

```
   // step 5: update alpha
   alpha := alpha - delta;
   epoch := epoch + 1;
end; // akhir perulangan while {kondisi STOP}
```

Hasil *training* atau pelatihan adalah bobot akhir, jumlah iterasi selama pelatihan dan juga akurasi pelatihannya.

```
edtEpoch.Text := IntToStr(epoch); // menampilkan jumlah epoch

GridData.ColCount := 9;
GridData.ColWidths[0] := 40;
GridData.ColWidths[1] := 70;
GridData.ColWidths[8] := 55;

for n := 2 to 6 do begin
   GridData.ColWidths[n] := 55;
end;
GridData.Cols[8].Text := 'Diagnosis';

// cek akurasi kemampuan memori
```

```
tracr := 0;
// untuk setiap data training
for i := 1 to jmldata do begin
   // hitung jarak ke masing-masing vektor bobot
   for j := 1 to 4 do begin
      jarak1 := 0;
      for m := 1 to 6 do begin
         jarak1 := jarak1 + Sqr(
            StrToFloat(GridData.Cells[m, i]) -
            StrToFloat(GridBobot.Cells[m, j])
         );
      end;
      jarak2 := RoundTo(Sqrt(jarak1), -2);
      d[j] := jarak2;
   end;

   // cari bobot dengan jarak minimum
   minD := MinValue(d); // fungsi minimum

   k := 1;
   while k < 5 do begin
      if MinD = d[k] then begin
         indeksD := k; // ambil indeks jarak minimum
         k := 5;
      end
      else k := k + 1;
   end;

   // cek apakah hasil diagnosis/prediksi
   // sama dengan class awal
   if GridBobot.Cells[7, indeksD] = GridData.Cells[7, i] then
   begin
      GridData.Cells[8, i] := IntToStr(indeksD);
      tracr := tracr + 1;
   end
   else begin
      GridData.Cells[8, i] := IntToStr(indeksD);
   end;
end;

// akurasi = (jml prediksi benar/jml data)*100%
edtTrainAcr.Text := FormatFloat('0.00', (tracr/jmldata)*100)
   + ' %';
```

Bobot akhir akan disimpan ke tabel Bobot pada *database*, sehingga ketika melakukan diagnosis maka pengguna tidak perlu melakukan proses pelatihan lagi tetapi cukup menggunakan hasil pelatihan yang sudah dilakukan sebelumnya.

3.4 Proses Pengujian

Proses selanjutnya adalah pengujian terhadap hasil pelatihan yang sudah selesai dilakukan. Dalam hal ini menggunakan data pengujian yang sudah disimpan pada Tabel *Uji1*, *Uji2*, dst. Selanjutnya akan dihasilkan persentase akurasi.

```
tracr := 0;
// untuk setiap data training
for i := 1 to jmldata do begin
   // hitung jarak ke masing-masing vektor bobot
   for j := 1 to 4 do begin
      jarak1 := 0;
      for m := 1 to 6 do begin
         jarak1 := jarak1 + Sqr(
            StrToFloat(GridData.Cells[m, i]) –
            StrToFloat(GridBobot.Cells[m, j])
         );
      end;
      jarak2 := RoundTo(Sqrt(jarak1), -2);
      d[j] := jarak2;
   end;

   // cari bobot dengan jarak minimum
   minD := MinValue(d); // fungsi minimum

   k := 1;
   while k < 5 do begin
      if MinD = d[k] then begin
         indeksD := k; // ambil indeks jarak minimum
         k := 5;
      end
      else k := k + 1;
   end;

   // cek apakah hasil diagnosis/prediksi
   // sama dengan class awal
   if GridBobot.Cells[7, indeksD] = GridData.Cells[7, i] then
   begin
      GridData.Cells[8, i] := IntToStr(indeksD);
      tracr := tracr + 1;
   end
   else begin
      GridData.Cells[8, i] := IntToStr(indeksD);
   end;
end;

// akurasi = (jml prediksi benar/jml data)*100%
edtTestAcr.Text := FormatFloat('0.00', (tracr/jmldata)*100)
   + ' %';
```

3.5 Proses Diagnosis

Akhir dari proses pada sistem adalah melakukan diagnosis terhadap penyakit berdasarkan gejala-gejala yang sudah di-*training* oleh sistem jaringan syaraf tiruan menggunakan metode LVQ ini. Data masukan yang diberikan oleh *user* berdasarkan pertanyaan gejala-gejala yang dialami hanya dengan memilih jawaban Ya dan Tidak, kecuali *temperature* memang wajib diisi.

```
for i := 1 to 7 do input[i] := 0;
input[1] := RoundTo(((StrToFloat(edtSuhu.Text) - 35)/7), -2);

if UpperCase(cmbNausea.Text) = 'YA' then input[2] := 1
else input[2] := 0;

if UpperCase(cmbLumbar.Text) = 'YA' then input[3] := 1
else input[3] := 0;

if UpperCase(cmbUrine.Text) = 'YA' then input[4] := 1
else input[4] := 0;

if UpperCase(cmbMict.Text) = 'YA' then input[5] := 1
else input[5] := 0;

if UpperCase(cmbUrtra.Text) = 'YA' then input[6] := 1
else input[6] := 0;

with ADOTable3 do begin
   Active := True;
   for i := 1 to RecordCount do begin
      bobot[1, i] := FieldByName('Temperature').AsFloat;
      bobot[2, i] := FieldByName('Nausea').AsFloat;
      bobot[3, i] := FieldByName('Lumbar').AsFloat;
      bobot[4, i] := FieldByName('Urine').AsFloat;
      bobot[5, i] := FieldByName('Micturition').AsFloat;
      bobot[6, i] := FieldByName('Urethra').AsFloat;
      bobot[7, i] := FieldByName('Class').AsFloat;
      Next;
   end;
end;
```

Selanjutnya menghitung jarak masing-masing vektor bobot dan mencari bobot dengan jarak minimum.

```
// hitung jarak ke masing-masing vektor bobot
for j := 1 to 4 do begin
   jarak1 := 0;
   for m := 1 to 6 do begin
      jarak1 := jarak1 + Sqr(input[m] - bobot[m, j]);
   end;
   jarak2 := RoundTo(Sqrt(jarak1), -2);
```

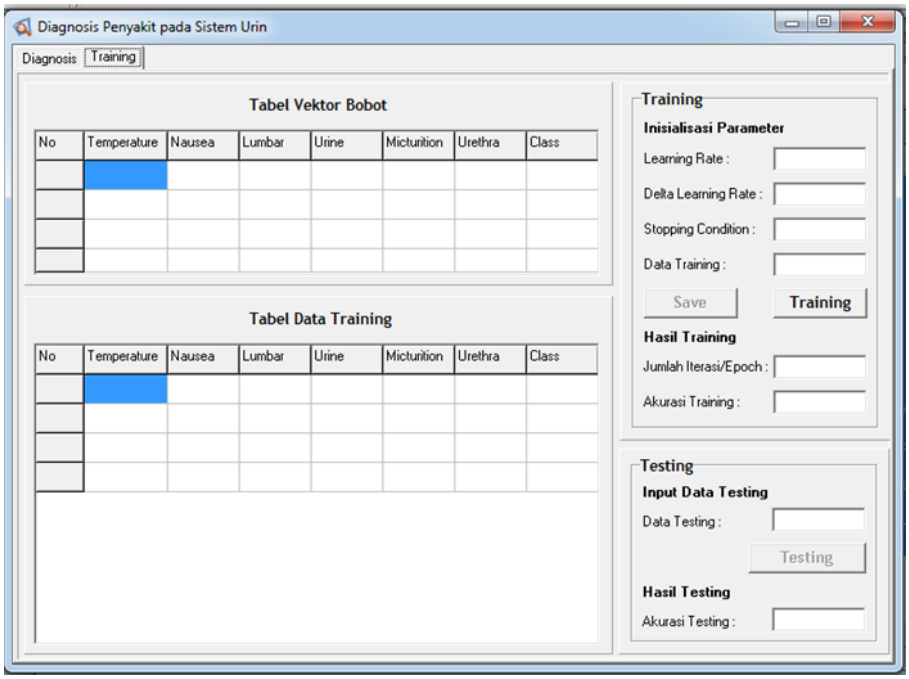
```
   d[j] := jarak2;
end;

// cari bobot dengan jarak minimum
minD := MinValue(d); // fungsi minimum

k := 1;
while k < 5 do begin
   if MinD = d[k] then begin
      indeksD := k; // ambil indeks jarak minimum
      k := 5;
   end
   else k := k + 1;
end;
```

Terakhir adalah menampilkan hasil diagnosis.

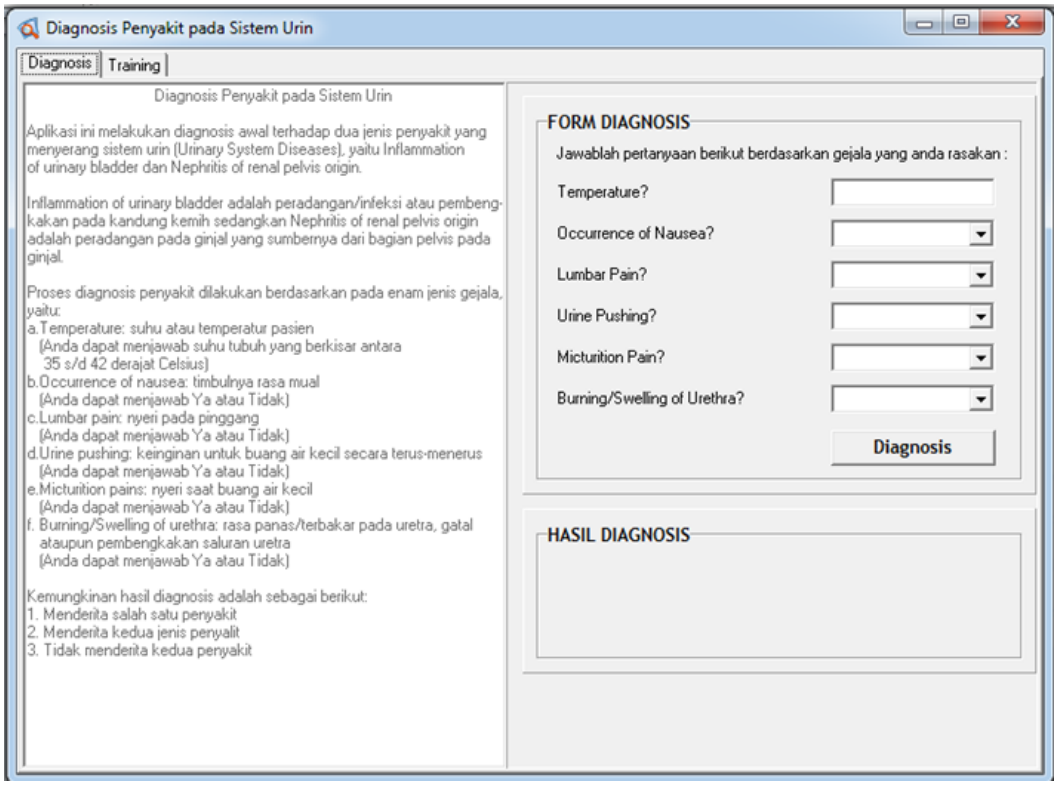
```
if indeksD = 1 then form1.lblHasil.Caption :=
   'Anda tidak menderita kedua jenis penyakit'
else if indeksD = 2 then form1.lblHasil.Caption :=
   'Anda menderita Inflammation of Urinary Bladder'
else if indeksD = 3 then form1.lblHasil.Caption :=
   'Anda menderita Nephritis of Renal Pelvis Origin'
else form1.lblHasil.Caption :=
   'Anda menderita kedua jenis penyakit';
```

4. Hasil dan Pembahasan

4.1 Antarmuka Sistem



Gambar 3(a) Antarmuka Proses *Training*



Gambar 3(b) Antarmuka Proses Diagnosis

Antar muka sistem yang seperti ditunjukkan pada Gambar 3 dibagi menjadi dua bagian yaitu antarmuka untuk proses *training* atau pelatihan dan yang kedua adalah antarmuka untuk proses diagnosis.

4.2 Hasil Uji Coba

Vektor Referensi sebagai bobot awal yang digunakan adalah empat data awal pada data *training* yang disajikan pada Gambar 4.

Training1

ID	Temperatur	Nausea	Lumbar	Urine	Micturition	Urethra	Class
1	0.13	0.00	1.00	0.00	0.00	0.00	1
2	0.43	0.00	1.00	1.00	0.00	1.00	2
3	0.14	0.00	0.00	1.00	1.00	1.00	3
4	0.71	1.00	1.00	1.00	1.00	1.00	4

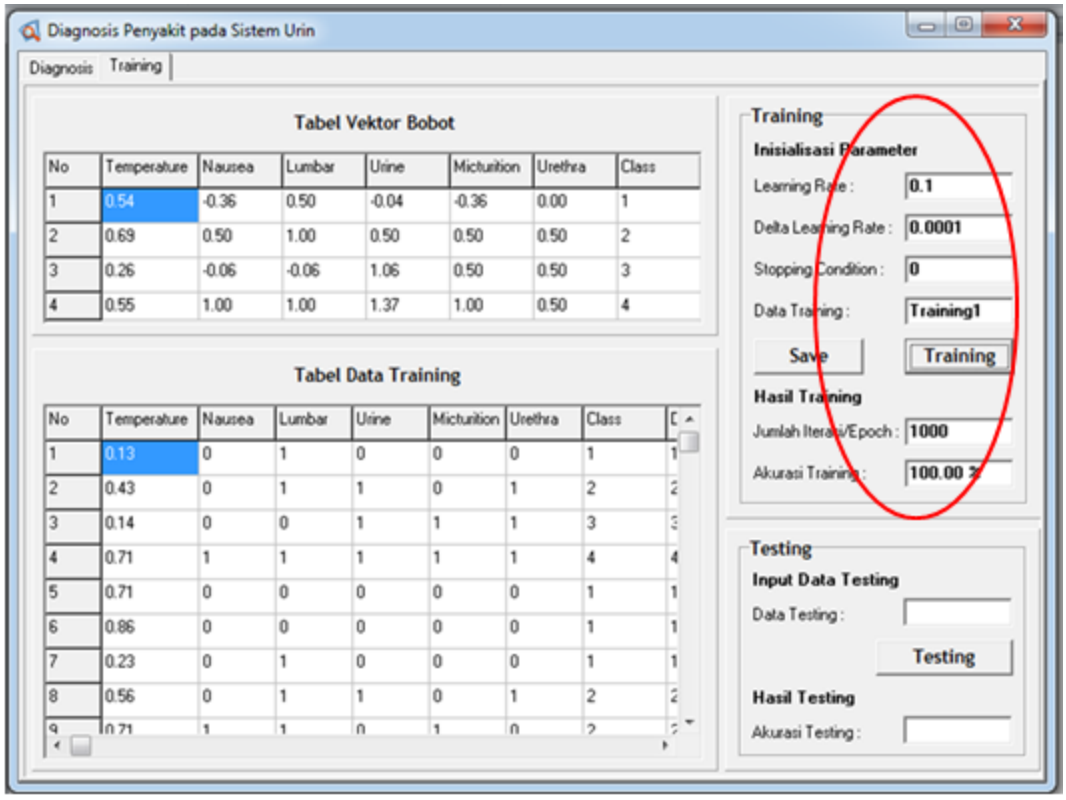
Gambar 4 Bobot Awal Pembelajaran menggunakan LVQ

Persentase tingkat akurasi *training* dan *testing* berdasarkan nilai parameter-parameter yang digunakan disajikan pada Tabel 1. Salah satu tampilan antarmuka proses *training* atau pelatihan dan pengujiannya ditunjukkan pada Gambar 5.

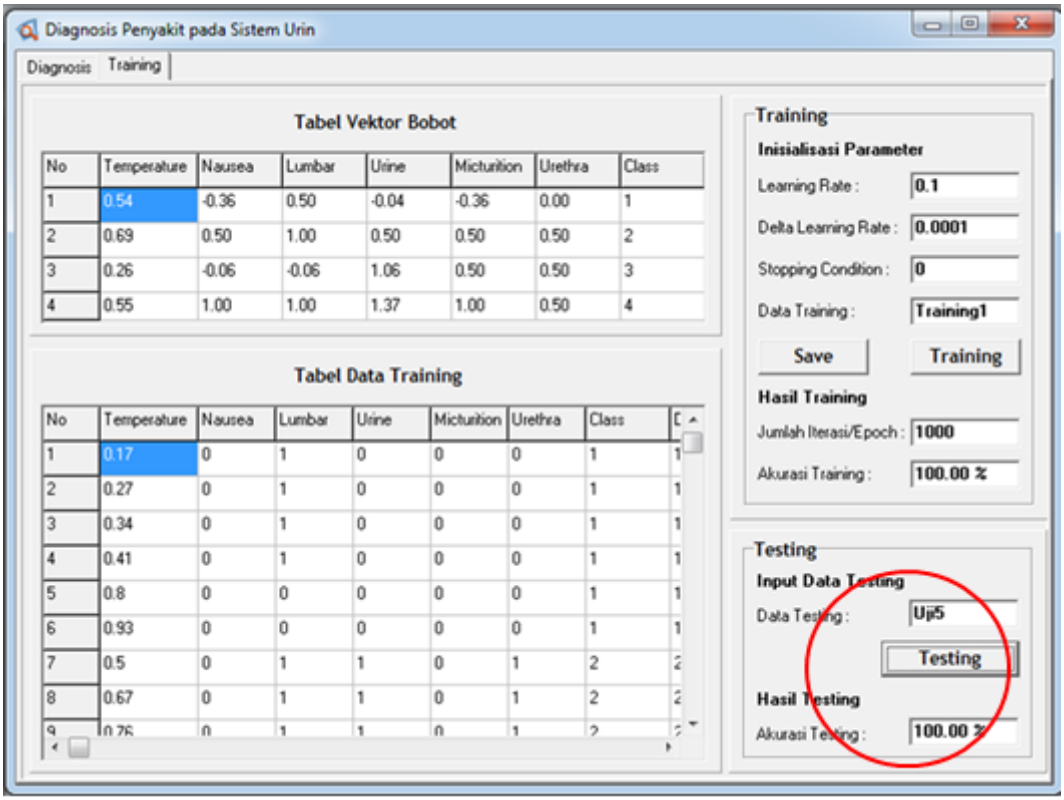
Bobot akhir yang dihasilkan akan disimpan pada tabel Bobot, dengan menekan tombol *Save* dan akan ditampilkan pesan verifikasi bahwa data bobot sudah berhasil disimpan seperti ditunjukkan pada Gambar 6.

Tabel 1 Persentase Tingkat Akurasi Pelatihan dan Pengujian

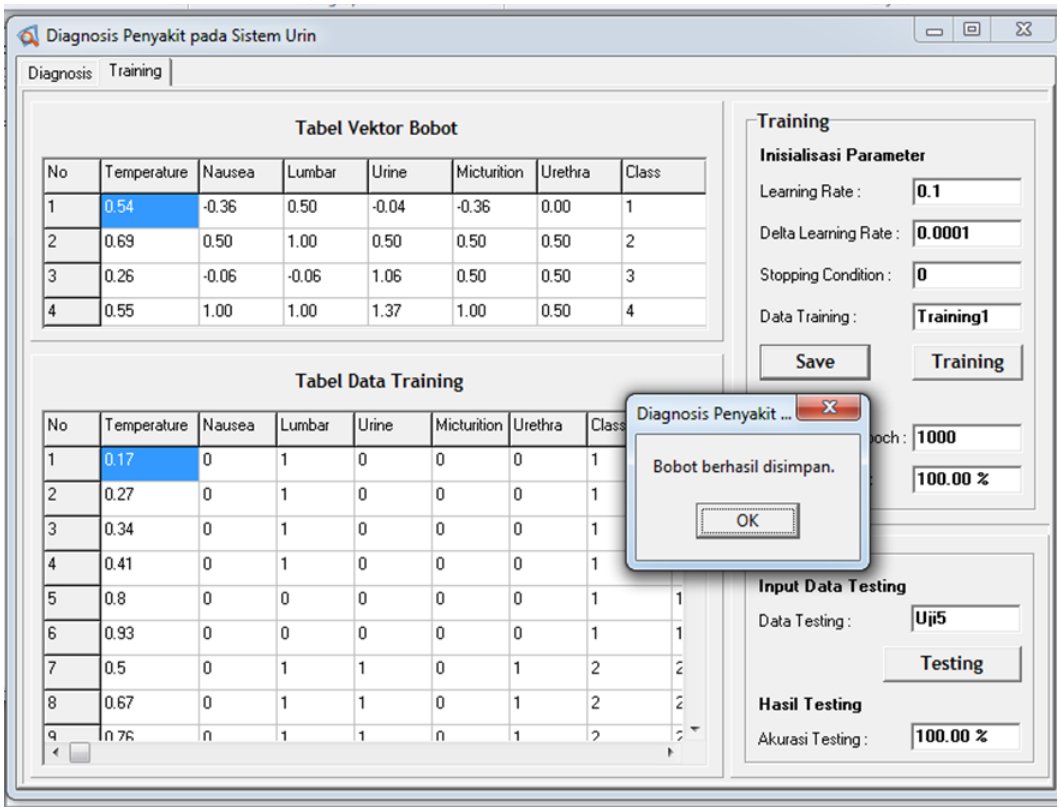
Parameter	Nilai	Data	Akurasi (%)	
			Training	Testing
Learning Rate Delta Stop Condition	0.1 0.0001 0	1 2 3 4 5	100 100 100 100 100	100 100 100 100 100
Rata-rata			100	100
Learning Rate Delta Stop Condition	0.1 0.0001 0	1 2 3 4 5	92.63 92.71 100 95.83 100	88 87.5 100 91.67 100
Rata-rata			96.234	93.434
Learning Rate Delta Stop Condition	0.1 0.0001 0	1 2 3 4 5	92.63 92.71 90.63 92.71 89.69	88 87.5 95.83 87.5 100
Rata-rata			91.674	91.766



Gambar 5(a) Antarmuka Hasil Pelatihan

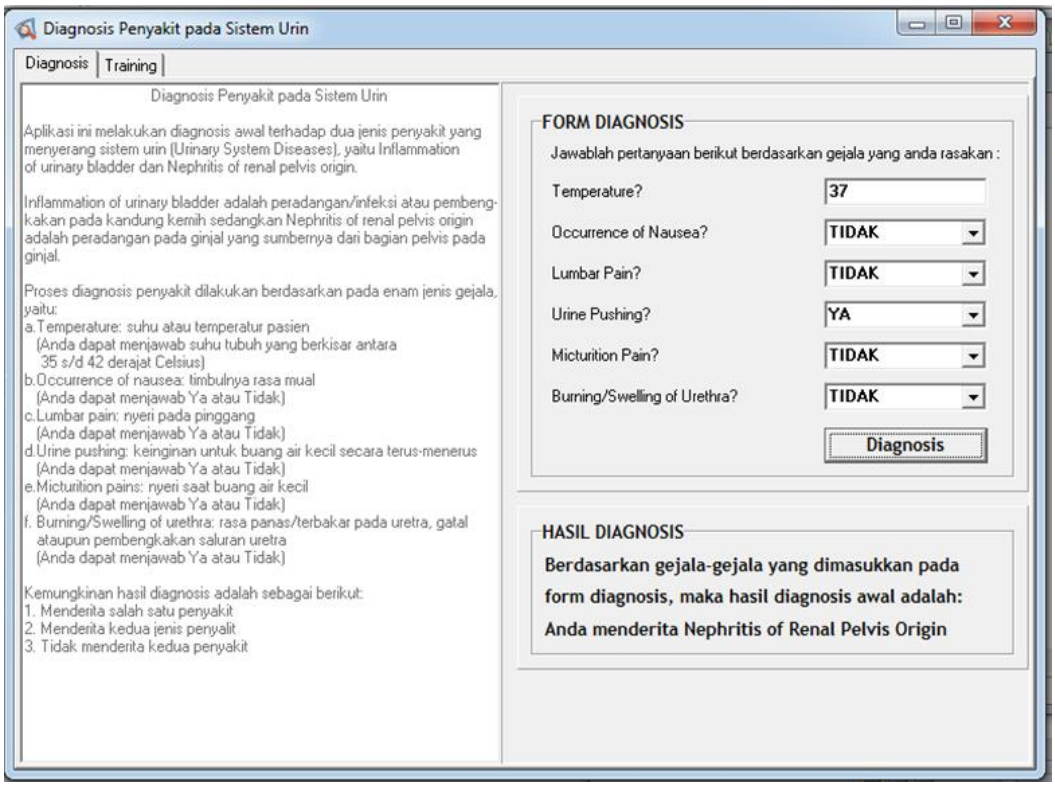


Gambar 5(b) Antarmuka Hasil Pengujian



Gambar 6 Antarmuka Penyimpanan Bobot Akhir ke Tabel Bobot

Dalam pengujian diagnosis ini, diberikan temperatur dengan nilai 37 dan gejala-gejala yang dialami hanya *Urine Pushing*, maka hasil diagnosis seperti pada Gambar 7 yang menunjukkan bahwa proses diagnosis berhasil dilakukan dengan benar atau sesuai.



Gambar 7 Hasil Diagnosis

5. Kesimpulan

Berdasarkan hasil implementasi yang sudah diuraikan sebelumnya bahwa persentase tingkat akurasi pelatihan dan pengujian menggunakan metode LVQ untuk kasus Diagnosis penyakit pada sistem Urin cukup tinggi. Hal ini berdasarkan hasil rata-rata akurasinya di atas 90% bahkan dapat mencapai 100% untuk *learning rate* = 0.1 dengan *delta learning rate* = 0.0001 dan kondisi berhenti = 0. Selain itu hasil diagnosis juga dapat dihasilkan dengan benar. Oleh karena itu metode LVQ ini dianggap cocok digunakan untuk masalah diagnosis, selain prosesnya cukup cepat dengan arsitektur yang sederhana dan juga hasil cukup memuaskan.

Daftar Pustaka

Fausett, L.V., 1994, *Fundamentals of Neural Networks Architecture, Algorithms, and Applications*, Prentice Hall Publishers, New Jersey.

Ranaldhi, D., 2006, Implementasi Learning Vector Quantization (LVQ) untuk Pengenalan Pola Sidik Jari pada Sistem Informasi Narapidana LP Wirogunan, *Jurnal Media Informatika*, Vol.4, No.1, hal. 51-65.

UCI, tanpa tahun, Data Sets, *http://archive.ics.uci.edu/ml/datasets.html*, diakses pada tanggal 11 Mei 2012.

Wikipedia, 2013, Sistem Urin, *http://id.wikipedia.org/wiki/Sistem_urin*, diakses pada tanggal 11 Mei 2012.