BAB IV
PERANCANGAN DAN IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan yang dijelaskan pada subbab ini adalah implementasi perangkat keras dan lunak yang digunakan. Kemudian inputan berupa data latih dan data uji, yang merupakan data terkait dataset *hepatitis Domain* yang terdiri dari 20 atribut dan 155 *record* data.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam penelitian ini adalah sebagai berikut :

1. *Processor* Intel Pentium Inside Core 2 Duo
2. *Memory* 2 GB
3. *Harddisk* dengan kapasitas 250 GB
4. Monitor Laptop 14.1 *inch.*
5. Keyboard
6. Mouse
7. RAM intel Chipset 512 MB

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian ini adalah sebagai berikut :

1. *Operating System* Windows XP *Black Edition* 32-bit
2. Microsoft Visual Studio C# 2010
3. Microsoft Solver Foundation (solver Quadratic Programming)
4. Microsoft Office Excel 2010

4.2 Implementasi Aplikasi

Berdasarkan pembahasan pada BAB III, proses pada aplikasi terdapat 4 proses utama, yaitu proses input data, proses normalisasi dan kernel, proses *training* SVM dan proses pengujian.

4.2.1 Implementasi *Input* Data

Proses awal yang dilakukan yaitu memasukkan data ke dalam aplikasi untuk kemudian dilakukan proses klasifikasi. Data yang akan dimasukkan adalah data latih dan data uji yang telah dipilih sebelumnya, dimana format data adalah berupa file *.xls atau *.xlsx yang merupakan tipe file Excel. Adapun proses input data ke yaitu dapat dilihat pada *sourcecode 4.1*.

```
private void LoadDataSet()
        {
            if (openFileDialog.ShowDialog(this) == DialogResult.OK)
            {
                string filename = openFileDialog.FileName;
                string extension = Path.GetExtension(filename);
                if (extension == ".xls" || extension == ".xlsx")
                {
                    ExcelReader db = new ExcelReader(filename, true, false);
                    TableSelectDialog t = new
TableSelectDialog(db.GetWorksheetList());
                    if (t.ShowDialog(this) == DialogResult.OK)
                    {
                        DataTable tableSource =
db.GetWorksheet(t.Selection);
                        double[,] sourceMatrix = tableSource.ToMatrix(out
sourceColumns);
                        if (sourceMatrix.GetLength(1) == 19)
                        {
                            MessageBox.Show("Missing class column.");
                        }
                        else
                        {
                            this.dgvDataset.DataSource = tableSource;
                        }
                    }}}}
```

Sourcecode 4.1 Proses input Data

Pada proses input data, data akan membaca file excel, dan nilai dataset tidak boleh ada yang kosong, artinya semuanya memiliki nilai, jika tidak maka tidak akan bisa

dibaca dan dimasukkan ke dalam *gridview* pada aplikasi.Data yang telah dibaca dari file excel akan kemudian ditampung pada matrix berupa array 2 dimensi.

4.2.2 Implementasi Normalisasi

Dalam proses normalisasi, terdapat langkah – langkah sebagai berikut, yaitu mencari nilai *max* dan nilai *min* pada setiap atribut pada data latih dan data uji. Setelah ditemukan nilai *max* dan *min*, baru kemudian dinormalisasi nilai data pada setiap atributnya. Untuk proses pencarin *max* dan *min* dapat dilihat pada *Sourcecode* 4.2. Proses pencarian nilai *max* dan *min* adalah melalui melakukan pencarian nilai data yang minimum dan maksimum pada setiap atribut data. Nilai *max* dan *min* yang telah didapatkan kemudian akan ditampung ke dalam array *max*[ ] dan *min*[ ].

```
  private void Normalisasi(double[,] data, DataGridView grid)
        {
            double[] max = new double[data.GetLength(1) - 1];
            double[] min = new double[data.GetLength(1) - 1];

            for (int j = 0; j < data.GetLength(1) - 1; j++)
            { min[j] = 1000; }

            for (int i = 0; i < data.GetLength(0); i++)
            {
                for (int j = 0; j < data.GetLength(1) - 1; j++)
                {

                    max[j] = Math.Max(data[i, j], max[j]);
                    min[j] = Math.Min(data[i, j], min[j]);
                }
            }
```

Sourcecode 4.2 Proses Pencarian Nilai *Max* dan *Min*

Setelah itu didapatkan nilai *max* dan *min*, maka kemudian proses normalisasi data dapat dilihat pada *Sourcecode 4.3*

```
for (int i = 0; i < data.GetLength(0); i++)
            {
                for (int j = 0; j < data.GetLength(1) - 1; j++)
                {
```

```
                    data[i, j] = Math.Round(((data[i, j] - min[j]) / (max[j]
- min[j])) * (1 + 0) + (0), 3);
                }
            }
```

Sourcecode 4.3 Proses Normalisasi Data

Pada *sourcecode* di atas dijelaskan bahwa nilai hasil normalisasi akan disimpan ke dalam array. Proses yang dilalui untuk melakukan normalisasi data latih dan data uji adalah sama, sehingga *method* Normalisasi pada *Sourcecode* 4.2. Nilai data yang telah dibaca dari file excel sebelmunya yang telah ditampung pada array 2 dimensi, dinormalisasi melalui *method* Normalisasi. Hasil normalisasi dari data uji maupun data latih akan ditampilkan pada *gridview* data uji dan data latih.

4.2.3 Proses Kernel Data Uji dan Data Latih

Untuk proses kernel pada data uji dan data latih yaitu dengan mengambil nilai data dari hasil normalisasi data latih dan data uji, yang selanjutnya akan dimasukkan ke dalam array 2 dimensi untuk setiap data latih maupun data uji. Adapun proses kernel secara linear dapat ditunjukkan oleh *Sourcecode 4.4*.

```
private void kernel(double[,] data, DataGridView grid)
        {
            for (int i = 0; i < data.GetLength(0); i++)
            {
                for (int j = 0; j < data.GetLength(1) - 1; j++)
                {
                    data[i, j] = Math.Round(data[i, j] * data[i, j], 3);
                }
            }

            for (int i = 0; i < data.GetUpperBound(1) + 1; i++)
            {
                grid.Columns.Add(sourceColumns[i], sourceColumns[i]);
            }

            for (int i = 0; i < data.GetUpperBound(0) + 1; i++)
            {
                var row = new DataGridViewRow();
                for (int j = 0; j < data.GetUpperBound(1) + 1; j++)
                {
                    row.Cells.Add(new DataGridViewTextBoxCell()
                    {
                        Value = data[i, j].ToString()
```

```
                    }
                );
            }
            grid.Rows.Add(row);
        }
        return;
    }
```

Sourcecode 4.4 Proses Kernel Data Latih dan Data Uji.

Pada *sourcecode* di atas proses kernel yaitu dengan mengambil data yang telah di normalisasi sebelumnya yang telah ditampung pada array 2 dimensi yaitu data[,]. Proses kernel dilakukan secara linear dengan implementasikan rumus kernel linear, dimana kernel linear nilai setiap atribut data akan dikuadratkan sehingga kemudian diperoleh hasil kernel. Hasil kernel akan ditampilkan pada *gridview* kernel data latih dan uji pada aplikasi.

4.2.4 *Quadratic Programming* (Optimasi)

Proses optimasi atau proses quadratic programming adalah proses untuk mencari nilai parameter SVM, yaitu nilai *w* (*weight*) , nilai *b* (bias), dan *slack variabel*. Proses Quadratic Programming pada source code ini menggunakan bantuan solver dari *Microsoft Solver Foundation* yang merupakan *library* dari *C#*. Adapun prosesnya yaitu dapat dilihat pada *Sourcecode 4.5*.

```
    private void optimasi(double [,] data, double [,] data2)
        {
            double[] w = new Double[data.GetLength(1) - 1];
            double[] y = new Double[data.GetLength(0)];
            double[] t = new Double[data.GetLength(0)];
            double b = 0.0;

            var solver = SolverContext.GetContext();
            solver.ClearModel();
            var model = solver.CreateModel();
            Term[] yi = new Term[data.GetLength(0)];
            Term[,] data1 = new Term[data.GetLength(0),
data.GetLength(1)];
            Term[] sum = new Term[data.GetLength(0)];
            var wi = new Decision[data.GetLength(1) - 1];
            var ti = new Decision[data.GetLength(0)];
            Term C = Convert.ToDouble(tbAlpha.Text);
```

```
            Decision bi = new Decision(Domain.Real, null);
            model.AddDecision(bi);
            for (var i = 0; i < data.GetLength(0); i++)
            {
                for (var j = 0; j < data.GetLength(1); j++)
                {
                    data[i, j] =
Convert.ToDouble(dgvNorm1.Rows[i].Cells[j].Value);
                    data1[i, j] = data[i, j];
                    yi[i] = data1[i, data.GetLength(1) - 1];
                }

            }

            for (int i = 0; i < data2.GetLength(0); i++)
            {

                for (int j = 0; j < data2.GetLength(1); j++)

                {

                    data2[i, j] =
Convert.ToDouble(dgvKernel2.Rows[i].Cells[j].Value);

                }
            }

            for (var i = 0; i < data.GetLength(1) - 1; i++)
                model.AddDecision(wi[i] = new Decision(Domain.Real,
null));

            for (var i = 0; i < data.GetLength(0); i++)
                model.AddDecision(ti[i] = new Decision(Domain.Real,
null));

            for (var i = 0; i < data.GetLength(0); ++i)
            {
                for (var j = 0; j < data.GetLength(1) - 1; j++){
                   //temp[i, j] = data1[i, j] * w[j];
                    //sum[i]+=temp[i,j];
                    model.AddConstraints(null, yi[i] *
((Model.Sum(data1[i, j] * wi[j])) + bi) + ti[i] - 1 >= 0 , ti[i] >= 0 );
                }
}

            Term wei = 0.0;
            Term jum = 0.0;


            Term goal = 0.0;

for (int i = 0; i < data.GetLength(0); i++)
            {
                jum += (ti[i]*C);
```

```
            }
                for (int j = 0; j < data.GetLength(1) - 1; j++)
                {
                    wei += (wi[j] * wi[j]);

                }
            goal = (0.5 * wei) + jum;
            model.AddGoal(null, GoalKind.Minimize, goal);
            var solution = solver.Solve();
            solver.PropagateDecisions();}
```

Sourcecode 4.5. Proses Quadratic Programming

Pada *sourcecode* di atas adalah merupakan proses *Quadratic Programming* untuk mencari nilai parameter SVM. Proses pencarian nilai tersebut adalah dengan membuat model formulasi SVM seperti pada bab sebelumnya. *Term* merupakan bentuk variabel yang dikenali oleh solver, jadi untuk inisialisasi variabel menggunakan *Term* dan *var*. Pertama adalah menginisialisasi variabel *decision* berupa nilai *w,* nilai *b* dan *slack variabel*. Selanjutnya adalah membuat *constraint* seperti pada formulasi SVM, dan kemudian membuat nilai goal atau fungsi objective sesuai dengan formulasi SVM. Hasil dari proses optimasi ini nantinya kan menghasilkan nilai dari variabel *decision* yang telah dinisialisasi sebelumnya yaitu nilai *w* (*weight*), nilai *b* (bias) dan *slack variabel*. Kemudian nilai parameter SVM yang telah didapatkan akan ditampung ke dalam array untuk diproses pada proses berikutnya.

4.2.5 Proses Pengujian SVM

Untuk proses pengujian yaitu dengan membandingkan nilai kelas yang diperoleh melalui hasil metode SVM, dengan dataset kelas asli. Berikut ini adalah *sourcecode* untuk perhitungan nilai constraint dan score pada data uji.

```
private void SetScore1(double[,] data1, double[] w, double b, double[] t)
        {
            double b1 = b;
            double[] Constraint = new Double[data1.GetLength(0)];
            double[] Score = new Double[data1.GetLength(0)];
            double[] Class = new Double[data1.GetLength(0)];
            double[] y = new Double[data1.GetLength(0)];
            double[,] temp = new Double[data1.GetLength(0),
data1.GetLength(1) - 1];
```

```
            double[] te = new Double[data1.GetLength(0)];
            double[] sum = new Double[data1.GetLength(0)];

            for (int i = 0; i < data1.GetLength(0); i++)
            {
                y[i] = data1[i, data1.GetLength(1) - 1];

            }

            for (int i = 0; i < data1.GetLength(0); i++)
            {
                for (int j = 0; j < data1.GetLength(1) - 1; j++)
                {
                    temp[i, j] = data1[i, j] * w[j];
                    sum[i] += temp[i, j];
                }
            }


            for (int i = 0; i < data1.GetLength(0); i++)
            {
                Constraint[i] = y[i] * (sum[i] + b1);
                Score[i] = sum[i] + b1;
                Class[i] = Math.Sign(Score[i]);
            }
            DisplayData(Constraint, Score, Class, dgvModelUji);
        }
```

Sourcecode 4.6. Perhitungan *constraint,score* untuk klasifikasi

Perhitungan nilai constraint dan nilai score menggunakan parameter SVM yang telah didapatkan yaitu nilai *w* (weight) dan nilai *b* (bias). Nilai *constraint* akan ditampung pada array sebanyak jumlah record data uji yang digunakan, begitu pula dengan nilai *score*. Untuk mengetahui kelas baru yang didapatkan pada dataset, maka kemudian digunakan fungsi *sign* pada nilai *score* untuk mengetahui tanda dari nilai score, apakah kategori 1 atau -1.

Selanjutnya untuk melakukan proses pengujian data dapat dilihat pada *Sourcecode* 4.5 berikut

```
private void UjiData()
        {
            double[,] temp = new Double[dgvModelUji.RowCount,
dgvModelUji.ColumnCount];
            double[,] dataset = new Double[dgvDataset.RowCount,
dgvDataset.ColumnCount];
```

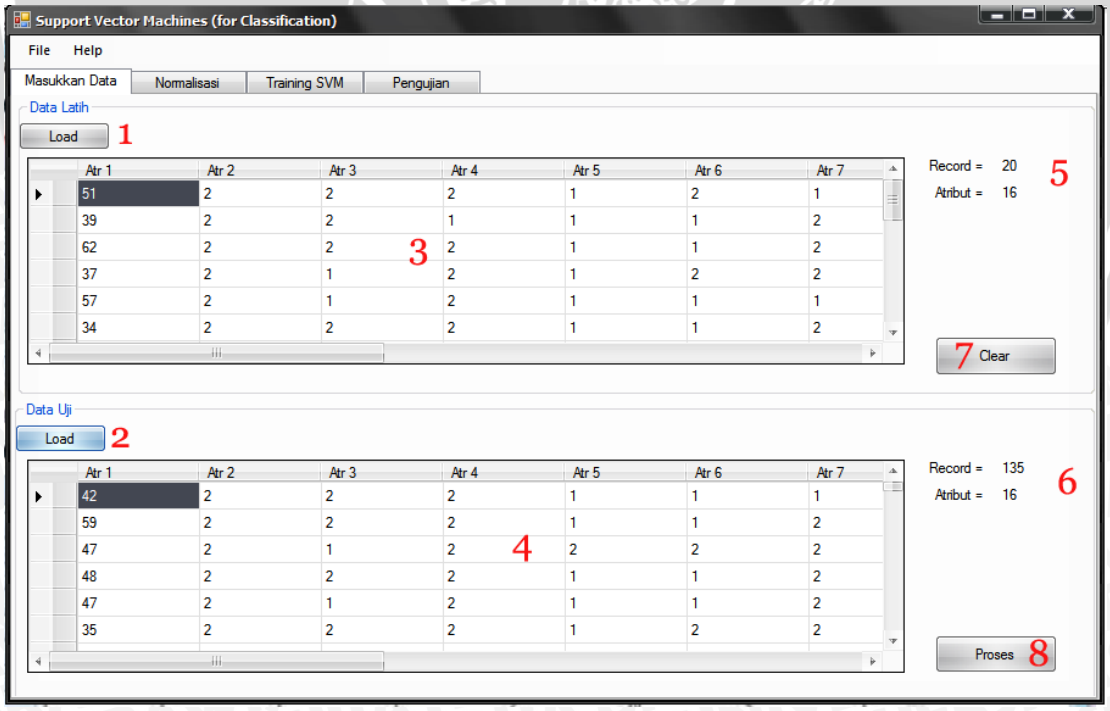
```
            double[] y = new Double[temp.GetLength(0)];
            double[] y2 = new Double[dataset.GetLength(0)];
            String[] dat = new String[temp.GetLength(0)];
            for (int i = 0; i < temp.GetLength(0) - 1; i++)
            {
                for (int j = 0; j < temp.GetLength(1); j++)
                {
                    temp[i, j] =
Convert.ToDouble(dgvModelUji.Rows[i].Cells[j].Value);
                    y[i] = temp[i, temp.GetLength(1) - 1];
                }
            }
            DisplayInfo(y, dgvInfo);

            for (int i = 0; i < dataset.GetLength(0) - 1; i++)
            {
                for (int j = 0; j < dataset.GetLength(1); j++)
                {
                    dataset[i, j] =
Convert.ToDouble(dgvDataset.Rows[i].Cells[j].Value);
                    y2[i] = dataset[i, dataset.GetLength(1) - 1];
                }
            }

            for (int i = 0; i < temp.GetLength(0) - 1; i++)
            {
                if ((y[i] == 1) && (y[i] == y2[i])) { dat[i] = "a"; }
                if ((y[i] == 1) && (y[i] != y2[i])) { dat[i] = "b"; }
                if ((y[i] == -1) && (y[i] == y2[i])) { dat[i] = "c"; }
                if ((y[i] == -1) && (y[i] != y2[i])) { dat[i] = "d"; }
            }

            double a1 = 0, a2 = 0, a3 = 0, a4 = 0;
            double akurasi = 0;
            double sensiti = 0;
            double spesiti = 0;
            for (int i = 0; i < temp.GetLength(0) - 1; i++)
            {
                a1 = dat.Count(s => s == "a");
                a2 = dat.Count(s => s == "b");
                a3 = dat.Count(s => s == "c");
                a4 = dat.Count(s => s == "d");
                Console.Write(" " + dat[i]);
            }
            akurasi = Math.Round(((a1 + a3) / (a1 + a2 + a3 + a4)) * 100,
3);
            sensiti = Math.Round(a1 / (a1 + a4), 3);
            spesiti = Math.Round(a3 / (a3 + a2), 3);
}
```

Sourcecode 4.7. Proses Pengujian Data

Untuk proses pengujian data yaitu dengan membandingkan nilai kelas pada dataset dengan kelas baru yang telah didapatkan dari proses SVM. Proses perbandingan yaitu dengan menggunakan kriteria *confusion table* dimana nilai a adalah True Positive, nilai b adalah False Positive, nilai c adalah True Negative, dan nilai d adalah False Negative. Kriteria ini nantinya akan dicari dengan membandingkan data uji dengan kelas asli dengan data uji kelas yang baru didapatkan dari hasil pelatihan. Setelah hasilnya didapatkan kemudian dapat diukur akurasinya, dengan rumus akurasi, dan juga dicari nilai *spesitivity* dan *sensitivity*.

4.3 Implementasi Antarmuka

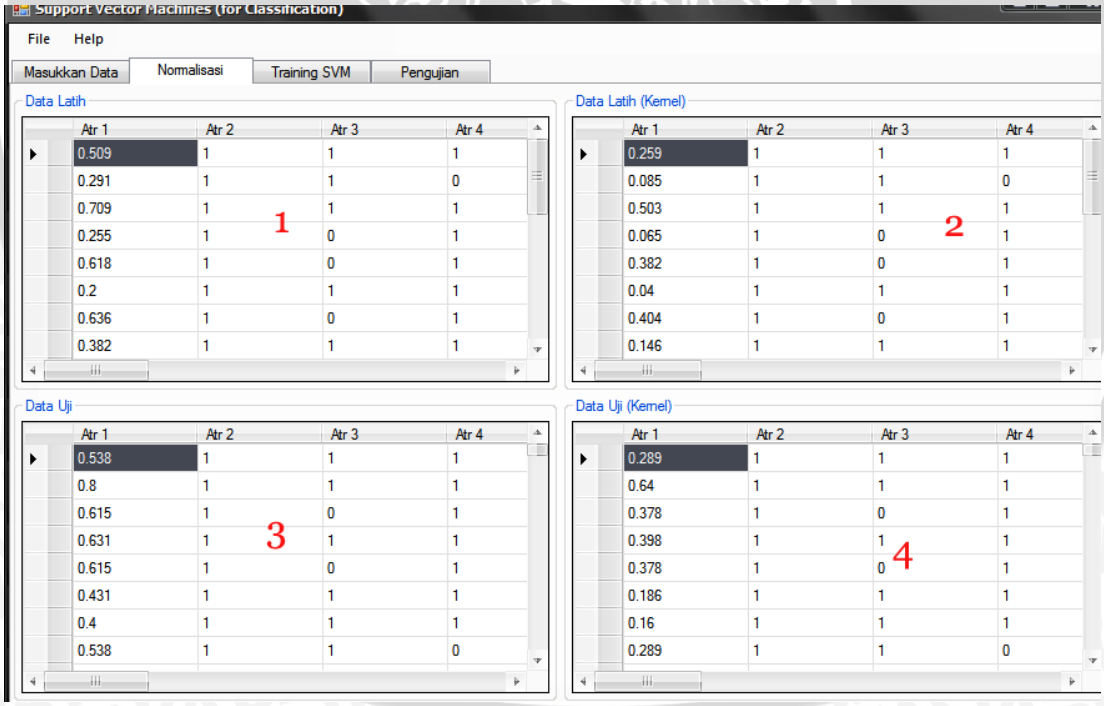
Seperti yang dijelaskan pada bab sebelumnya bahwa dalam sistem terdapat 4 bagian utama dari sistem yaitu, proses input data, proses normalisasi data, proses training data. Proses klasifikasi adalah dengan menggunakan metode *support vector machine*. Adapun proses tersebut dapat ditunjukkan pada tab – tab berikut.



Gambar 4.1 Form Input Data

Keterangan gambar 4.1 :

1. Tombol *Load* berfungsi untuk memasukkan data latih
2. Tombol *Load* berfungsi untuk memasukkan data uji
3. *DataGridView* untuk menampilkan data latih yang telah dipilih
4. *DataGridView* untuk menampilkan data uji yang telah dipilih
5. *Label* yang menunjukkan jumlah *record* dan *atribut* pada data latih
6. *Label* yang menunjukkan jumlah *record* dan *atribut* pada data uji
7. Tombol *Clear* untuk mengosongkan *DataGridView* untuk data latih maupun data latih
8. Tombol *Proses* untuk melakukan proses Normalisasi dan Kernel pada data uji dan data latih, dan hasilnya ditampilkan pada tab Normalisasi.

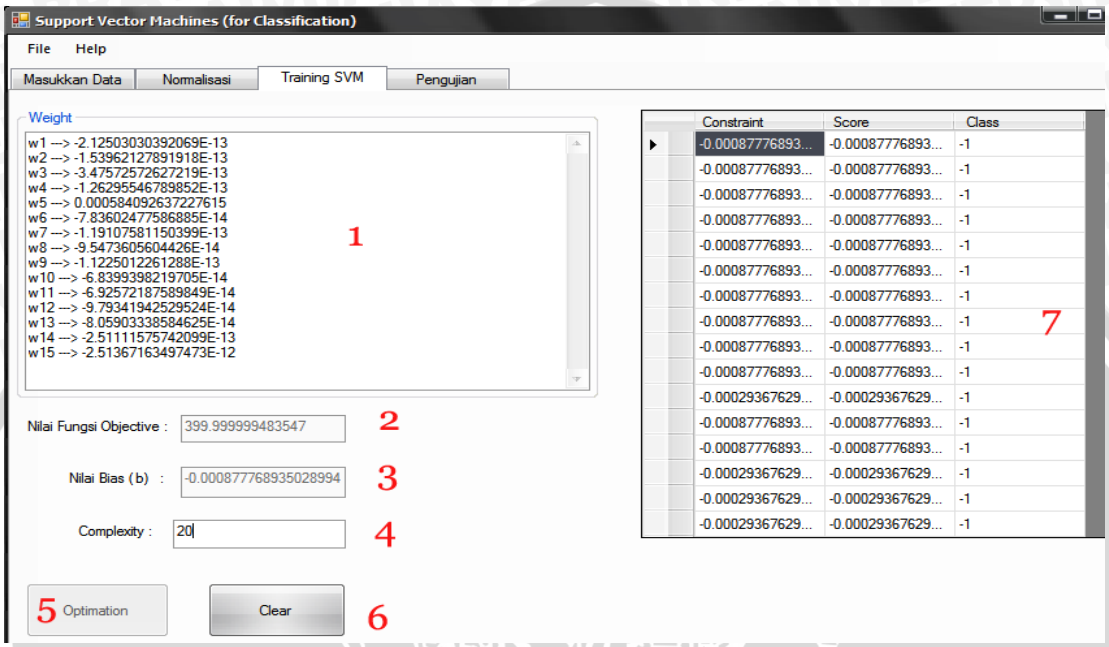


Gambar 4.2 Form Normalisasi Data

Keterangan gambar 4.2

1. *DataGridView* untuk menampilkan hasil Normalisasi data latih

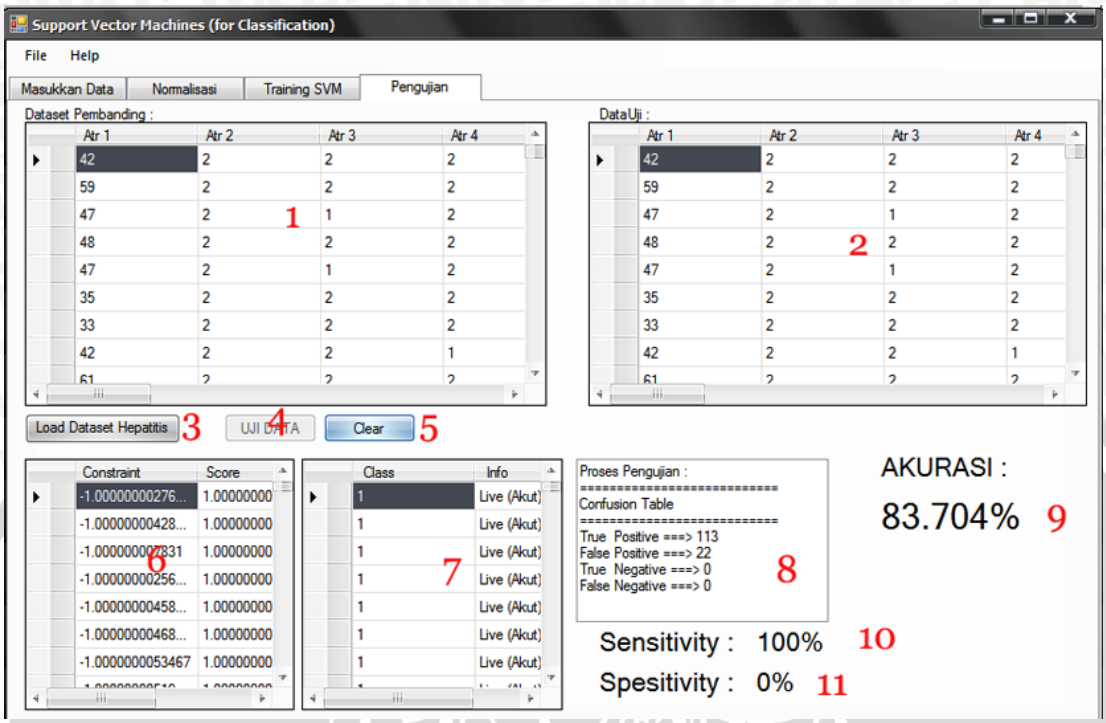
2. *DataGridView* untuk menampilkan hasil Normalisasi data uji
3. *DataGridView* untuk menampilkan hasil Kernel data latih
4. *DataGridView* untuk menampilkan hasil Kernel data uji



Gambar 4.3 Form Training SVM

Keterangan gambar 4.3

1. *TextBox* untuk menampilkan nilai *w* (*weight*) hasil proses Quadratic Programming
2. *TextBox* untuk menampilkan nilai fungsi *objective*
3. *TextBox* untuk menampilkan nilai bias
4. *TextBox* untuk menginputkan nilai *Complexity*
5. Tombol *Optimasi* untuk melakukan proses Quadratic Programming untuk mendapatkan parameter SVM
6. Tombol *Clear* untuk mengosongkan semua *TextBox* dan *Datagridview*
7. *DataGridview* untuk menampilkan hasil perhitungan *constraint* dan *score* pada data latih.



Gambar 4.4 Form Pengujian

Keterangan gambar 4.4

1. *DataGridView* untuk menampilkan data uji yang berisi kelas asli
2. *DataGridView* untuk menampilkan data uji tanpa kelas
3. Tombol *Load Dataset* untuk menginputkan dataset dengan kelas asli
4. Tombol Uji Data untuk melakukan proses pengujian data dan pengukuran akurasi.
5. Tombol *Clear* untuk mengosongkan semua *DataGridView* dan *TextBox*
6. *DataGridView* untuk menampilkan nilai *constraint* dan *score* hasil klasifikasi pada data uji
7. *DataGridView* untuk menampilkan nilai kelas dari data uji dan informasi nilainya.
8. *TextBox* untuk menampilkan *confusion table* klasifikasi
9. Label untuk menampilkan tingkat akurasi klasifikasi

10. Label untuk menampilkan tingkat akurasi *sensitivity*
11. Label untuk menampilkan tingkat akurasi *spesitivity*

4.4 Implementasi Pengujian

Pengujian dilakukan terhadap *dataset Hepatitis Domain*, dimana terdiri dari 155 *record* data dengan 20 atribut termasuk kelas. Dataset ini kemudian dibagi menjadi 20,40,80,100,dan 155 untuk keperluan Data Latih, sedangkan untuk Data Uji digunakan 135 data. Kemudian proses pelatihan dengan SVM, terdapat nilai C (*Complexity*) adalah suatu parameter *user* yang digunakan untuk mengukur tingkat akurasi klasifikasi dari metode SVM.

4.4.1 Lingkungan Pengujian

Lingkungan pengujian pada proses klasifikasi penderita penyakit hepatitis dengan menggunakan metode SVM akan dilakukan dengan beberapa kali pengujian terhadap data uji sebanyak 75 *record* data. Adapun jumlah atribut untuk proses pengujian yaitu dengan menggunakan sebanyak 19 atribut data dan 15 atribut data. Data latih yang digunakan adalah sebanyak 20,40,60, dan 80 dengan lingkungan pengujian sebagai berikut :

- Nilai C : 10,20,30,40,…100

Nilai C (*Complexity*) merupakan parameter yang diinputkan untuk mengukur tingkat akurasi dari metode SVM. Tujuan adanya nilai *C* (*Complexity*) adalah untuk meminimalkan *error* dan memperkecil nilai *slack variabel*.

(*Halaman ini sengaja dikosongkan*)