BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam sub bab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan perangkat lunak ini adalah sebagai berikut :

1. Intel® Core™ 2 CPU P7350 @ 2,00GHz
2. Memori 3 GB
3. Harddisk 200 GB
4. Monitor 21”

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan perangkat lunak ini adalah sebagai berikut :

1. Sistem Operasi Windows 7 Ultimate 32 bit
2. Notepad++ v6.3
3. Microsoft Visual C# 2010 Express

4.2 Implementasi Program

Berdasarkan analisa dan perancangan proses yang terdapat pada bab III, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Implementasi *Clustering*

Proses *clustering* pada aplikasi yang dibangun menggunakan algoritma *K-Modes*. Salah satu proses yang ada pada *clustering K-Modes* adalah perhitungan jarak data dengan pusat *cluster*. Proses perhitungan tersebut menggunakan metode *new dissimilarity measure* dan *K-Modes* Konvensional.

4.2.2 Penentuan Pusat *Cluster* Awal

Sesuai dengan proses pada bab II dan bab III, penentuan *cluster* ditentukan secara *random*. *Source code* penentuan *cluster* awal ditunjukkan **Source Code 4.1**

```
Random r = new Random();
List<int> iC = new List<int>();
do//penentuan pusat klaster awal
{
    int rand = r.Next(0, record);
    if (!iC.Contains(rand))
    {
        iC.Add(rand);
    }
}
while (iC.Count <= k);

string[][] initCentroid = new string[k][];
for (int a = 0; a < k; a++)
{
    initCentroid[a] = new string[attribute];
    for (int b = 0; b < attribute; b++)
    {
        initCentroid[a][b] = dataset[iC[a]][b];
    }
}
```

Source Code 4.1 Menentukan pusat *cluster* awal

4.2.3 Penentuan Jarak

Perhitungan jarak *Dissimilarity Measure* adalah dengan cara menghitung perbedaan nilai dari suatu *record* dengan *centroid*. Untuk setiap nilai yang berbeda pada *K-Modes* Konvensional, variabel total bertambah satu. Perulangan dilakukan hingga seluruh *record* dibandingkan jaraknya.

Proses penggantian *centroid* yang baru dilakukan dari kolom per kolom untuk tiap-tiap *cluster*. Untuk tiap kolom, dicari nilai apa saja yang ada dan dimasukkan dalam *List*. Frekuensi kemunculan tiap nilai per kolom disimpan

dalam *array*. Kemudian, cari frekuensi nilai dalam kolom yang paling sering muncul yang disebut juga dengan modus. Lakukan langkah ini berulang-ulang hingga seluruh baris *centroid* untuk semua *cluster* telah terpenuhi. *Source code* jarak akan ditunjukkan **Source Code 4.2**

```
double[][] distance = new double[record][];
        for (int a = 0; a < record; a++)
        {
            distance[a] = new double[k];
        }

        double[][] vFr = new double[attribute][];
        string[][] value = new string[attribute][];
        for (int a = 0; a < attribute; a++)
        {
            List<string> val = new List<string>();
            for (int b = 0; b < record; b++)
            {
                if (!val.Contains(dataset[b][a]))
                {
                    val.Add(dataset[b][a]);
                }
            }
            vFr[a] = new double[val.Count];
            value[a] = new string[val.Count];

            for (int b = 0; b < val.Count; b++)
            {
                value[a][b] = val[b];
            }

            for (int b = 0; b < val.Count; b++)
            {
                for (int c = 0; c < record; c++)
                {
                    if (value[a][b] == dataset[c][a])
                    {
```

```
                            vFr[a][b]++;
                        }
                    }
                }
            }

            for (int a = 0; a < record; a++)
            {
                for (int b = 0; b < k; b++)
                {
                    for (int c = 0; c < attribute; c++)
                    {
                        if (dataset[a][c] != initCentroid[b][c])
                        {
                            distance[a][b]++;
                        }
                        else
                        {
                            for (int d = 0; d < value[c].Length;
d++)
                            {
                                if (dataset[a][c] == value[c][d])
                                {//rumus jarak langkah awal
                                    distance[a][b] += 1 -
(vFr[c][d] / record);

                                    break;
                                }
                            }
                        }
                    }
                }
            }

            //penentuan klaster
            int[] ixCl = new int[record];
            for (int a = 0; a < record; a++)
            {
                double temp = 99999;
```

```
            int index = 0;
            for (int b = 0; b < k; b++)
            {
                if (distance[a][b] < temp)
                {
                    temp = distance[a][b];
                    index = b;
                }
            }
            ixCl[a] = index;
        }

        //untuk mengetahui jumlah klaster
        int[] sumCl = new int[k];
        for (int a = 0; a < record; a++)
        {
            for (int b = 0; b < k; b++)
            {
                if (b == ixCl[a])
                {
                    sumCl[b]++;
                    break;
                }
            }
        }

        //mengelompokkan nilai record hasil clustering
        string[][][] cluster = new string[k][][];
        for (int a = 0; a < k; a++)
        {
            cluster[a] = new string[sumCl[a]][];
            for (int b = 0; b < sumCl[a]; b++)
            {
                cluster[a][b] = new string[attribute];
            }
        }

        int[][] indexCl = new int[k][];
```

```
for (int a = 0; a < k; a++ )
{
    indexCl[a] = new int[sumCl[a]];
    int c = 0;
    for (int b = 0; b < record; b++)
    {
        if (ixCl[b] == a)
        {
            indexCl[a][c] = b;
            c++;
        }
    }
}

for (int a = 0; a < k; a++)
{
    for (int b = 0; b < record; b++)
    {
        if (a == ixCl[b])
        {
            int d = 0;
            for (int c = 0; c < attribute; c++)
            {
                cluster[a][d][c] = dataset[b][c];
            }
            d++;
        }
    }
}

double[][][] clvFr = new double[k][][];
string[][][] clvalue = new string[k][][];
for (int z = 0; z < k; z++)
{
    clvFr[z] = new double[attribute][];
    clvalue[z] = new string[attribute][];
    for (int a = 0; a < attribute; a++)
    {
```

```
                clvFr[z][a] = new double[vFr[a].Length];
                clvalue[z][a] = new string[value[a].Length];

                for (int b = 0; b < value[a].Length; b++)
                {
                    clvalue[z][a][b] = value[a][b];
                }

                for (int b = 0; b < value[a].Length; b++)
                {
                    for (int c = 0; c < sumCl[z]; c++)
                    {
                        if (clvalue[z][a][b] ==
dataset[indexCl[z][c]][a])
                        {
                            clvFr[z][a][b]++;
                        }
                    }
                }
            }
        }

        //menentukan centroid/pusat klaster baru
        string[][] newCentroid = new string[k][];
        for (int a = 0; a < k; a++)
        {
            newCentroid[a] = new string[attribute];
            for (int b = 0; b < attribute; b++)
            {
                double temp = 0;
                for (int c = 0; c < clvFr[a][b].Length; c++)
                {
                    if (clvFr[a][b][c] > temp)
                    {
                        newCentroid[a][b] = clvalue[a][b][c];
                    }
                }
```

```
            }
        }

        bool konvergen = false;
        int iteration = 0;

        int[] newCluster = new int[record];
        int[] oldCluster = new int[record];
        for (int a = 0; a < record; a++)
        {
            newCluster[a] = ixCl[a];
        }

        do
        {
            iteration++;
            for (int a = 0; a < record; a++)
            {
                oldCluster[a] = newCluster[a];
            }

            newCluster = new int[record];
            distance = new double[record][];
            for (int a = 0; a < record; a++)
            {
                distance[a] = new double[k];
            }

            for (int a = 0; a < record; a++)
            {
                for (int b = 0; b < k; b++)
                {
                    for (int c = 0; c < attribute; c++)
                    {
                        if (dataset[a][c] !=
newCentroid[b][c])
                        {
                            distance[a][b]++;
```

```
                        }
                        else
                        {
                            for (int d = 0; d <
value[c].Length; d++)
                            {
                                if (dataset[a][c] ==
value[c][d])
                                {
                                    distance[a][b] += 1 -
((clvFr[b][c][d] / sumCl[b]) * (vFr[c][d] / record));
                                    if
(Double.IsNaN(distance[a][b]))
                                    {
                                        distance[a][b] = 0;
                                    }
                                    break;
                                }
                            }
                        }
                    }
                }

                newCluster = new int[record];
                for (int a = 0; a < record; a++)
                {
                    double temp = 0;
                    int index = 0;
                    for (int b = 0; b < k; b++)
                    {
                        if (distance[a][b] > temp)
                        {
                            temp = distance[a][b];
                            index = b;
                        }
                    }
                    newCluster[a] = index;
```

```
}

sumCl = new int[k];
for (int a = 0; a < record; a++)
{
    for (int b = 0; b < k; b++)
    {
        if (b == newCluster[a])
        {
            sumCl[b]++;
            break;
        }
    }
}

cluster = new string[k][][];
for (int a = 0; a < k; a++)
{
    cluster[a] = new string[sumCl[a]][];
    for (int b = 0; b < sumCl[a]; b++)
    {
        cluster[a][b] = new string[attribute];
    }
}

indexCl = new int[k][];
for (int a = 0; a < k; a++)
{
    indexCl[a] = new int[sumCl[a]];
    int c = 0;
    for (int b = 0; b < record; b++)
    {
        if (newCluster[b] == a)
        {
            indexCl[a][c] = b;
            c++;
        }
    }
```

```
                }

                for (int a = 0; a < k; a++)
                {
                    for (int b = 0; b < record; b++)
                    {
                        if (a == newCluster[b])
                        {
                            int d = 0;
                            for (int c = 0; c < attribute; c++)
                            {
                                cluster[a][d][c] = dataset[b][c];
                            }
                            d++;
                        }
                    }
                }

                clvFr = new double[k][][];
                clvalue = new string[k][][];
                for (int z = 0; z < k; z++)
                {
                    clvFr[z] = new double[attribute][];
                    clvalue[z] = new string[attribute][];
                    for (int a = 0; a < attribute; a++)
                    {

                        clvFr[z][a] = new double[vFr[a].Length];
                        clvalue[z][a] = new
string[value[a].Length];

                        for (int b = 0; b < value[a].Length; b++)
                        {
                            clvalue[z][a][b] = value[a][b];
                        }

                        for (int b = 0; b < value[a].Length; b++)
                        {
```

```
                        for (int c = 0; c < sumCl[z]; c++)
                        {
                            if (clvalue[z][a][b] ==
dataset[indexCl[z][c]][a])
                            {
                                clvFr[z][a][b]++;
                            }
                        }
                    }
                }
            }

            newCentroid = new string[k][];
            for (int a = 0; a < k; a++)
            {
                newCentroid[a] = new string[attribute];
                for (int b = 0; b < attribute; b++)
                {
                    double temp = 0;
                    for (int c = 0; c < clvFr[a][b].Length;
c++)
                    {
                        if (clvFr[a][b][c] > temp)
                        {
                            newCentroid[a][b] =
clvalue[a][b][c];
                        }
                    }
                }
            }
```

Source Code 4.2 Menentukan jarak

4.2.4 Penentuan *Purity Measure*

Proses perhitungan *purity measure* yaitu penjumlahan dari objek di *class* yang frekuensinya paling dominan di tiap *cluster* kemudian di bagi dengan jumlah data/*object/record* yang ada. Sedangkan error yang dihasilkan merupakan pengurangan 1 dengan *purity* yang di hasilkan.

```
        double[] dominant = new double[k];
        double[][] dominate = new double[k][];
                string[][] clClass = new string[k][];
                for (int a = 0; a < k; a++)
                {
                    clClass[a] = new string[sumCl[a]];
                    dominate[a] = new double[k];
                    for (int b = 0; b < sumCl[a]; b++)
                    {
                        clClass[a][b] =
dataset[indexCl[a][b]][attribute - 1];
                    }
                }

                for (int a = 0; a < k; a++)
                {
                    for (int b = 0; b < sumCl[a]; b++)
                    {
                        for (int c = 0; c < k; c++)
                        {
                            if (clClass[a][b] == kelas[c])
                            {
                                dominate[a][c]++;
                            }
                        }
                    }
                }

                for (int a = 0; a < k; a++)
                {
                    double temp = 0;
                    for (int b = 0; b < k; b++)
                    {
                        if (dominate[b][a] > temp)
                        {
                            temp = dominate[b][a];
                            dominant[a] = dominate[b][a];
                        }
```

```
                }
            }

            double purity = dominant.Sum() / record;
            textBox5.Text = Convert.ToString(purity);
            double error = 1 - purity;
            textBox6.Text = Convert.ToString(error);
```

Source Code 4.3 Menghitung hasil *Purity Measure*

4.2.5 Evaluasi *F-Measure*

Awal dari proses evaluasi *f-measure* adalah memasukkan nilai-nilai seperti *record*, *cluster* dan kelas ke dalam *List* dataEval bertipe struct. Kemudian, cari banyak kelas pada *dataset* dan simpan pada *List* class*List*. Alokasikan jumlah anggota suatu kelas yang ada pada *cluster*, simpan dalam *array* dataM. Selanjutnya, cari *truePositive* dengan cara melihat jumlah kelas anggota klaster, yang dominan dijadikan *truePositive cluster* tersebut. *falseNegative* didapat dari setiap data yang dominan di *cluster* a, namun tidak di *cluster* b, jika memenuhi syarat, maka *falseNegative* akan bertambah. *falsePositive* diambil dari anggota *cluster* a yang tidak masuk dominan, jika memenuhi syarat, maka nilai *falsePositive* akan ditambah. Langkah selanjutnya adalah perhitungan *precision*, *recall*, dan *f-measure* masing-masing *cluster*. Jika sudah dihitung, maka ambil rata-rata *f-measure* untuk dijadikan nilai akhir. *Source code* evaluasi *f-measure* ditunjukkan **Source Code 4.4**.

```
List<string> classList = new List<string>();
            for (int a = 0; a < record; a++)
            {
                if (!classList.Contains(dataEval[a].kelas))
                {
                    classList.Add(dataEval[a].kelas);
                }
            }

            int[][] dataM = new int[k][];
            for (int a = 0; a < k; a++)
            {
                dataM[a] = new int[classList.Count];
            }

            for (int a = 0; a < record; a++)
```

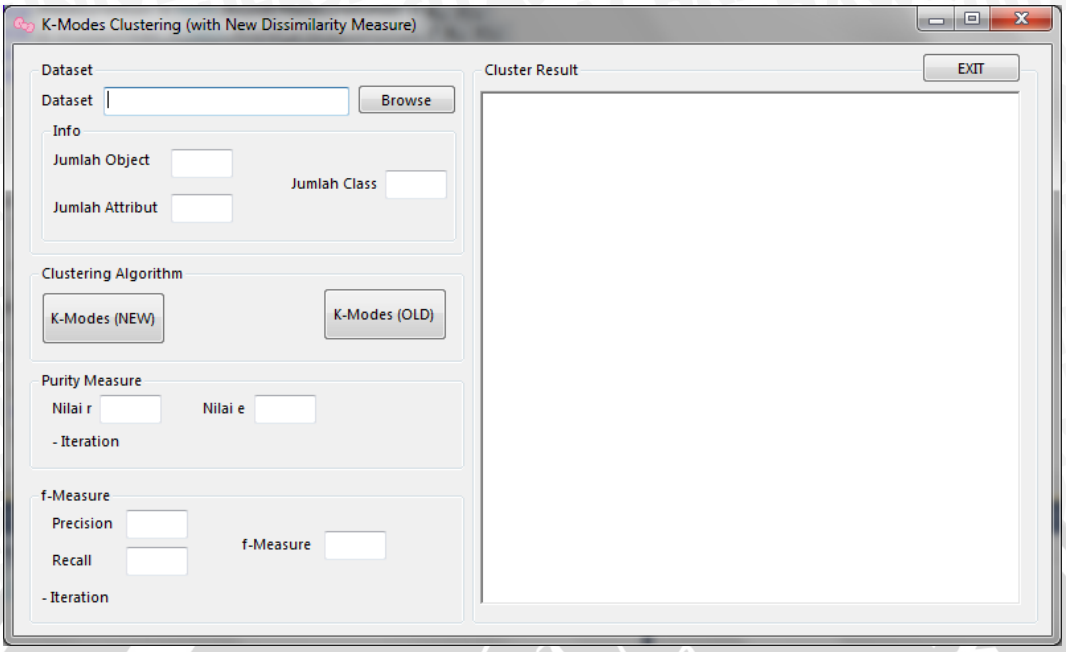
```
            {
                for (int b = 0; b < k; b++)
                {
                    for (int c = 0; c < classList.Count; c++)
                    {
                        if ((dataset[a][attribute - 1] == classList[c])
&& (dataEval[a].cluster == b))
                        {
                            dataM[b][c]++;
                            break;
                        }
                    }

                }
            }

            int[] truePositive = new int[k];
            int[] trueNegative = new int[k];
            int[] falseNegative = new int[k];
            int[] falsePositive = new int[k];
            double[] accuracy = new double[k];
            double[] recall = new double[k];
            double[] precision = new double[k];
            double[] fmeasure = new double[k];
            int[] dominan = new int[k];
            for (int a = 0; a < k; a++)
            {
                dominan[a] = 0;
                int temporary = 0;
                for (int b = 0; b < classList.Count; b++)
                {
                    if (dataM[a][b] > temporary)
                    {
                        temporary = dataM[a][b];
                        dominan[a] = b;
                    }
                }
                truePositive[a] = dataM[a][dominan[a]];
            }

            for (int a = 0; a < k; a++)
            {
                for (int b = 0; b < k; b++)
                {
                    if (dominan[a] != dominan[b])
                    {
                        falseNegative[a] += dataM[b][dominan[a]];
                    }
                }
            }

            for (int a = 0; a < k; a++)
            {
                for (int b = 0; b < k; b++)
                {
                    if (dominan[a] != dominan[b])
                    {
                        trueNegative[a] = dataM[b][dominan[b]];
                    }
```

```
                }
            }

            for (int a = 0; a < k; a++)
            {
                for (int b = 0; b < classList.Count; b++)
                {
                    if (b != dominan[a])
                    {
                        falsePositive[a] += dataM[a][b];
                    }
                }
            }


            for (int a = 0; a < k; a++)
            {
                accuracy[a] = ((0.5 * Convert.ToDouble(truePositive[a]))
/ ((Convert.ToDouble(truePositive[a]) +
Convert.ToDouble(falseNegative[a])))) + ((0.5 *
Convert.ToDouble(trueNegative[a])) / ((Convert.ToDouble(trueNegative[a])
+ Convert.ToDouble(falsePositive[a]))));
                recall[a] = Convert.ToDouble(truePositive[a]) /
(Convert.ToDouble(truePositive[a]) + Convert.ToDouble(falseNegative[a]));
                precision[a] = Convert.ToDouble(truePositive[a]) /
(Convert.ToDouble(truePositive[a]) + Convert.ToDouble(falsePositive[a]));
                fmeasure[a] = 2 * recall[a] * precision[a] / (recall[a] +
precision[a]);
            }

            double fmeasuretotal = 0;
            double precisiontotal = 0;
            double recalltotal = 0;
            double accuracytotal = 0;
            for (int a = 0; a < k; a++)
            {
                fmeasuretotal += fmeasure[a];
                precisiontotal += precision[a];
                recalltotal += recall[a];
                accuracytotal += accuracy[a];
            }
            double fmeasureMean = Math.Round(fmeasuretotal / k, 3);
            double precisionMean = Math.Round(precisiontotal / k, 3);
            double recallMean = Math.Round(recalltotal / k, 3);
            double accuracyMean = Math.Round(accuracytotal / k, 3);
```

Source Code 4.4 Evaluasi *F-Measure*

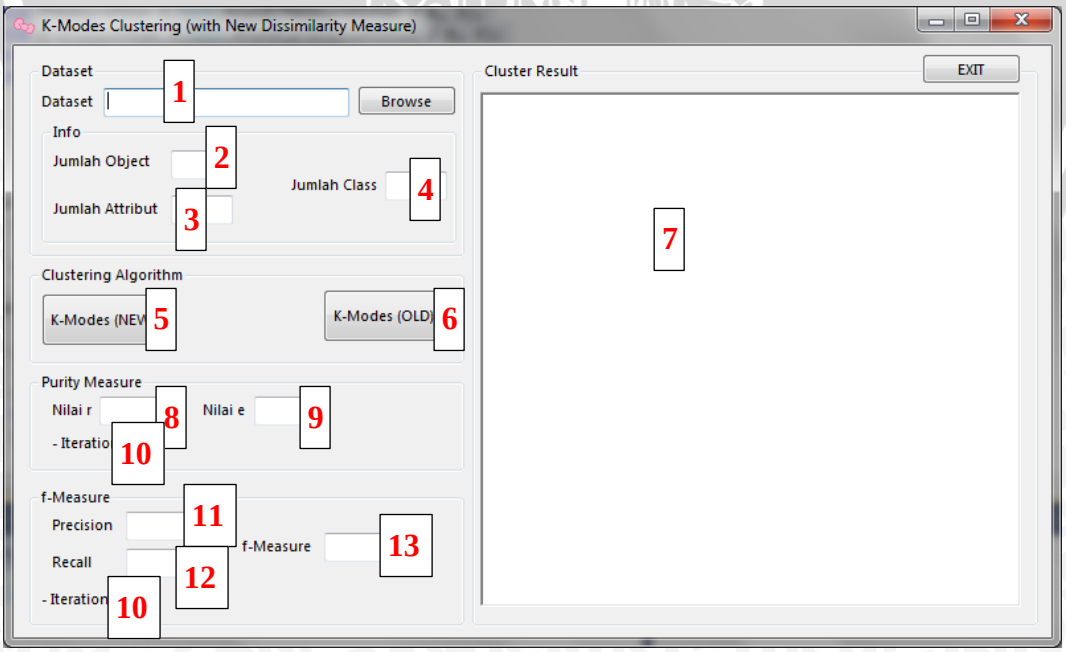
4.3 Implementasi Antarmuka

Implementasi antarmuka pada *clustering* data kategori menggunakan *K-Modes New Dissimilarity Measure*, seperti yang telah dijelaskan di subbab 3.4, rancangan antarmuka yang digunakan terdiri dari satu *form*.



Gambar 4.1 Antarmuka *system K-Modes Clustering*

Pada satu *form* tersebut terdiri dari 4 bagian. Pada bagian pertama berisi *input file*, kedua berisi *button* untuk memilih algoritma yang di gunakan, bagian ketiga berisi informasi hasil *clustering*, dan bagian yang keempat berisikan *purity measure*. Antarmuka *system* akan ditunjukkan pada Gambar 4.2 berikut.



Gambar 4.2 Informasi Antarmuka *system K-Modes Clustering*

Keterangan :

1. *Browse* pada *Dataset* adalah *button* yang akan memunculkan *browse text box* dimana user akan memilih *dataset* yang akan digunakan.
2. *Text box* Jumlah *Object* adalah tempat dimana hasil pembacaan *dataset* menunjukkan banyaknya *object* di *dataset*.
3. *Text box* Jumlah Attribut adalah tempat dimana hasil pembacaan *dataset* menunjukkan banyaknya attribut di *dataset*.
4. *Text box* Jumlah *Class* adalah tempat dimana hasil pembacaan *dataset* menunjukkan banyaknya *class* di *dataset*, yang juga akan berfungsi sebagai jumlah *cluster* yang digunakan dalam pengkasteran *dataset* tersebut.
5. *Button process K-Modes* (*NEW*) adalah *button* yang berfungsi untuk melakukan komputasi *New K-Modes* atau *K-Modes New Dissimilarity Measure*.
6. *Button process K-Modes* (*OLD*) adalah *button* yang berfungsi untuk melakukan komputasi *Old K-Modes* atau *K-Modes* Konvensional.
7. *Rich text box cluster result* details menampilkan frekuensi *class* di setiap *cluster* yang dihasilkan oleh *system* dan nilai *Precision*, *Recall*, dan *F-Measure*.
8. *Text box* Nilai *r* adalah tempat dimana menunjukkan hasil *purity measure* yang dihasilkan oleh *system*.
9. *Text box* Nilai *e* adalah tempat dimana menunjukkan hasil *error* yang dihasilkan oleh *system*.
10. *Label* keterangan adalah banyaknya iterasi yang dilakukan dalam perhitungan *clustering* pada *system*.
11. *Text box* Nilai *Precision* adalah tempat dimana menunjukkan hasil *Precision* yang dihasilkan oleh *system.*
12. *Text box* Nilai *Recall* adalah tempat dimana menunjukkan hasil *Recall* yang dihasilkan oleh *system.*
13. *Text box* Nilai *F-Measure* adalah tempat dimana menunjukkan hasil *F-Measure* yang dihasilkan oleh *system.*