**BAB IV**

**IMPLEMENTASI DAN UJI COBA**

Bab 4 ini merupakan pembahasan mengenai hasil implementasi dan uji coba yang dilakukan. Pada bab ini juga akan ditampilkan hasil dari pembuatan klasifikasi penyakit diabetes yang di implementasikan kedalam GUI yang telah dirancang pada bab sebelumnya. Total data yang digunakan adalah 614 data *train* dan 154 data uji hasil pra-pemrosesan.

**4.1 Dataset Hasil Pra-pemrosesan**

Pada bab sebelumnya dilakukan pra-pemrosesan untuk datasets. Sebelum dilakukan *modelling* menggunakan algorima *logistic regression binary*, data dibagi menjadi dua kategori yaitu kategori 0 ( tidak terindikasi ) dan kategori 1 ( terindikasi ). Maka hasil dari pra-pemrosesan yang dijadikan data latih sebanyak 614 data. Pada tabel 4.1 dibawah ini terdapat hasil data setelah dilakukan pra-pemrosesan.

Tabel 4.1 Data Latih

|  |  |
| --- | --- |
| Jumlah Data |  |
| Latih | 614 |
| Uji | 154 |

Total data untuk data latih sebanyak 614 data yang berada pada 2 kelas *diabetes prediction*. Sedangkan data uji sebanyak 154 yang tersebar pada 2 kelas *diabetes prediction*.

**4.2 Pelatihan Model *Logistic Regression Binary***

Proses latih pada model *Logistic Regression Binary* terdiri atas memuatdata kemudian membagi data dan selanjutnya diproses untuk dilatih.

**4.2.1 Memuat Datasets**

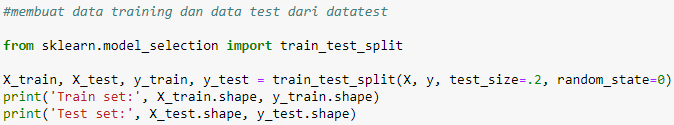
Datasets hasil pra-pemrosesan akan dimuat dan dilakukan normalisasi menggunakan *min-max* pada persamaan 2.12 melalui skrip dengan tujuan modifikasi nilai dalam variabel sehingga dapat diukur dalam skala umum. Berikut adalah skrip fungsi pada python untuk memuat *datasets* sekaligus melakukan normalisasi pada gambar 4.1.



Gambar 4.1 Pra-pemrosesan data dan Normalisasi

**4.2.2 Split Datasets**

Pelatihan model *Logistic Regression Binary* menggunakan *datasets* latih hasil pra-pemrosesan yang kemudian dibagi dengan cara perbandingan 80% untuk data latihdan 20% untuk data validasi dari setiap kelas menggunakan pustaka atau *library* *sklearn*. Berikut adalah *listing code* untuk melakukan pemisahan datasets menggunakan *library sklearn* pada python.

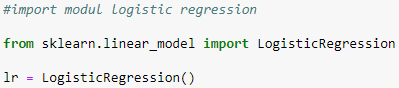


Gambar 4.2 Membagi data *train* dan data *test*

Jumlah data latih yang didapat yaitu sejumlah 614 data dan sejumlah 154 data untuk data validasi .

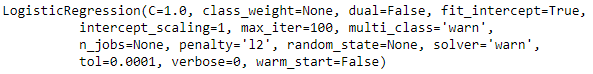
**4.2.3 Train Model**

Model yang dilatih untuk mempelajari fitur-fitur dan kelas pada data *diabetes prediction*. Tujuan dari latihan model ini agar supaya model dapat mengenali dan mempelajari cirri-ciri untuk melakukan klasifikasi penyakit diabetes. Kelas tersebut berasal dari data kategorikal yaitu 1 ( terindikasi ) dan 0 ( tidak terindikasi ). Kemudian pada tahap ini dilakukan pelatihan dengan *datasets* yang telah dimuat dan dipisahkan terhadap model yang dibuat. Berikut adalah *listing code* untuk melakukan pelatihan model.



Gambar 4.3 Memanggil Model *Logistic Regression Binary*

Dalam proses latih, dilakukan *modelling* menggunakan algoritma *Logistic Regression Binary* dengan nama penalty = ‘l2’. Berikut contoh gambar proses latih model *Logistic Regression Binary.*



Gambar 4.4 Proses Latih Model *Logistic Regression Binary*

**4.3 *Accuracy and Loss Validation***

Proses latih dilakukan dengan algoritma *logistic regression binary*. Proses tersebut melatih sebanyak 614 data latih yang merupakan 80% dari jumlah keseluruhan *datasets* dan dilakukan proses validasi dengan 154 data yang didapatkan dari 20% jumlah *datasets*. Dari percobaan yang telah dilakukan, didapatkan hasil *accuracy* dan *loss* pada model. Tabel 4.2 dibawah ini adalah hasil dari *accuracy* dan *loss*.

Tabel 4.2 Accuracy dan Loss

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | **Accuracy in percentage** | **Loss** | **Loss in percentage** |
| 0.8181818181818182 | 81,8 % | 0.181818181818181 | 18,2 % |

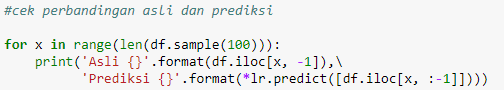
**4.4 Pengujian Klasifikasi Dengan Data Test**

Data *test* merupakan data yang sudah dipisahkan sebelumnya pada tahap pra-pemrosesan. Data yang akan di uji berjumlah 154 data yang ada pada setiap kelas. Sedangkan model yang digunakan adalah model *Logistic Regression Binary* yang merupakan model terbaik dalam penelitian ini. Untuk melakukan klasifikasi, langkah-langkah yang harus dilakukan adalah memuat semua data *test* kemudian di uji menggunakan model yang sudah dilatih. Berikut adalah *listing code* untuk memuat data *test* dan melakukan klasifikasi pada program.



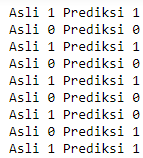
Gambar 4.5 Pengujian data *test*

Dalam kasus ini penggunaan kelas kategorikal dikhususkan untuk membandingkan hasil prediksi dan kelas aslinya. Berikut adalah *listing code* untuk menampilkan hasil prediksi data *test.*



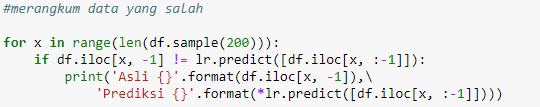
Gambar 4.6 Cek perbandingan data asli dan prediksi

*Listing code* diatas merupakan *listing code* untuk melakukan klasifikasi data masukan secara masal. Dari skrip diatas akan menghasilkan output sebagai berikut.



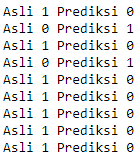
Gambar 4.7 Hasil klasifikasi menggunakan data acak

Gambar 4.7 merupakan sebagian hasil dari prediksi menggunakan data acak. Dari 200 data yang diujikan pada model yang telah dilatih, terdapat 47 data yang mengalami kegagalan dalam melakukan klasifikasi. Untuk menampilkan output berupa data yang salah dalam melakukan klasifikasi menggunakan *listing code* seperti gambar 4.8.



Gambar 4.8 *listing code* untuk output klasifikasi yang salah

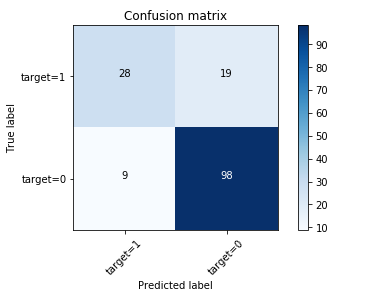
Pada dasarnya *listing code* diatas sama dengan *listing code* sebelumnya, namun pada *listing code* diatas hanya akan menampilkan *output* hasil klasifikasi yang salah. Berikut contoh *output* pada gambar 4.9.



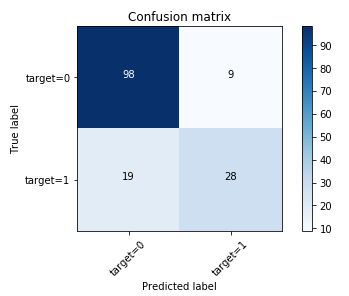
Gambar 4.9 output hasil klasifikasi yang salah

Ketepatan model tersebut dapat diukur dengan beberapa metode pengukuran *Confussion Matrix* diantaranya pengukuran *accuracy, loss, recall, F1-score,* dan pengukuran *precision.*

Langkah awal untuk melakukan pengukuran adalah melakkukan pemetaan hasil klasifikasi data *test* pada suatu *confussion matrix*. Hasil ukur menggunakan *confussion matrix* menggunakan data *test* yang terdiri dari 154 data dan tersebar secara acak pada dua kelas.



Gambar 4.10: *Confussion Matrix* untuk kelas 1



Gambar 4.11: Confussion *Matrix* untuk kelas 0

**4.4.1 Nilai Akurasi**

Pengukuran nilai akurasi dilihat berdasarkan dengan seberapa sering hasil klasifikasi bernilai benar secara keseluruhan. Nilai akurasi dapat dihitung dengan persamaan 2.8.

**4.4.2 *Error Rate***

Pengukuran nilai *loss* atau *error* dapat diukur dengan persamaan 2.9 pada bab 2.

**4.4.3 Nilai *Positive Predictive Value (Precision)***

Pengukuran nilai presisi bisa diukut dengan persamaan 2.11. Berikut adalah contoh perhitungan untuk nilai *precision* pada kelas 1:

Perhitungan diatas menghitung nilai *precision* pada kelas 1, untuk melihat hasil nilai *precision* kedua kelas dapat dilihat pada tabel 4.3.

Tabel 4.3: Nilai *precision* untuk kelas 0 dan 1

|  |  |
| --- | --- |
| Kelas | *Precision* |
| 0 | 0.84 |
| 1 | 0.76 |

**4.4.4 Nilai True Positive Rate (*Recall*)**

Pengukuran *recall* dapat dihitung menggunakan persamaan 2.10. Berikut contoh perhitungan *recall* pada kelas 1:

Perhitungan diatas hanya untuk menghitung nilai *recall* pada kelas 1, untuk melihat hasil kedua kelas dapat dilihat pada tabel 4.4.

Tabel 4.4: Nilai *Recall* untuk kelas 0 dan 1

|  |  |
| --- | --- |
| Kelas | Recall |
| 0 | 0.92 |
| 1 | 0.60 |

**4.4.5 Nilai F1*-Score***

Untuk membandingkan model dengan menggunakan hasil pengukuran dari nilai *recall* dan *precision* tidaklah mudah atau terbilang cukup sulit. Maka agar kedua nilai pengukuran tersebut sebanding maka dilakukan pengukuran nilai F1-*score*. Pengukuran ini merupakan harmonisasi dari metode pengukurann *recall* dan *precisionn* yang merupakan hasil perbandingan dari persamaan 2.10 dan 2.11. Maka perhitungan F1-*score* dapat dimodelkan secara matematis dengan persamaan berikut:

Perhitungan nilai F1-*score* diatas hanya menghitung nilai pada kelas 1, untuk melihat nilai F1-*score* kedua kelas dapat dilihat pada tabel 4.5.

Tabel 4.5: Tabel nilai f1-*score* untuk kelas 0 dan 1

|  |  |
| --- | --- |
| Kelas | F1*-score* |
| 0 | 0.88 |
| 1 | 0.67 |

Hasil rata-rata seluruh nilai dikalikan dengan 100% untuk menyajikan nilai akhir dalam bentuk persen.

Tabel 4.6: Hasil akhir nilai pengukuran *confussion matrix* menggunakan data *test*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Accuracy* | *Error Rate* | *Precision* | *Recall* | F1*-score* |
| 81.8 % | 18.1% | 84% | 83% | 84% |

**4.5 Aplikasi GUI (*Graphical User Interface)***

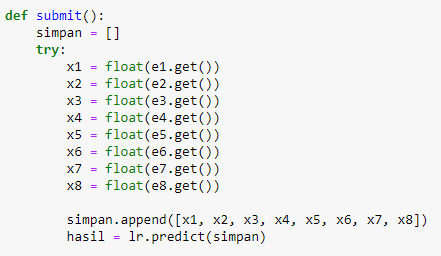
Aplikasi GUI yang dibuat mengikuti rancangan pada bab sebelumnya. Tujuan dari pembahasan aplikasi ini adalah untuk melakukan prediksi menggunakan model yang sudah diuji dan dilatih. Pada pembuatan aplikasi menggunakan pustaka GUI python bernama Tkinter, Berikut adalah *listing code* yang diperlukan untuk membuat GUI dan memanggil pustaka atau *library* yang diperlukan.



Gambar 4.12: memanggil modul Tkinter

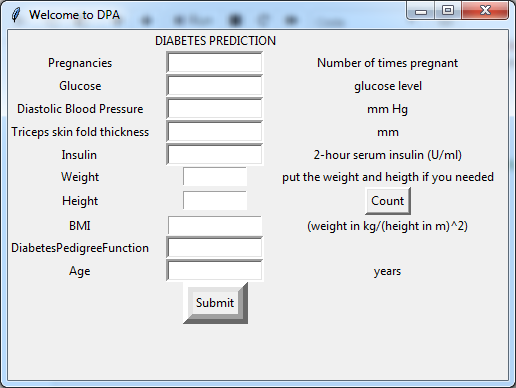
Pada contoh *listing code* diatas merupakan cara memanggil modul Tkinter untuk digunakan sebagai rancangan GUI pada program python.

Untuk melakukan klasifikasi dengan menggunakan GUI, model yang telah dilatih sebelumnya harus dimuat terlebih dahulu. Berikut *listing code* untuk memuat model yang telah dilatih.



Gambar 4.13: *Listing code* memuat model kedalam GUI

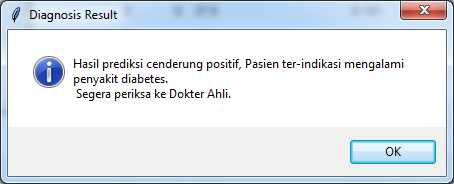
Pada gambar diatas merupakan proses klasifikasi GUI dengan kode lr.predict(new\_input) program dapat melakukan prediksi. Output dari aplikasi GUI yang telah dibuat dapat dilihat pada gambar dibawah ini.



Gambar 4.14: Tampilan GUI aplikasi prediksi diabetes

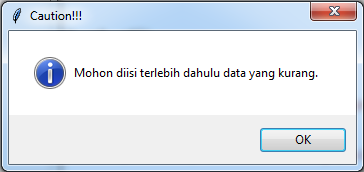
Aplikasi yang telah dirancang dan dibuat bertujuan untuk mempermudah proses memprediksi proses klasifikasi menggunakan model yang telah dilatih yaitu model *Logistic Regression Binary*. Untuk menentukan suatu data yang input dapat dilakukan hanya dengan melakukan *input data* pada *form* yang telah disediakan menggunakan aplikasi GUI. Berdasarkan tujuan tersebut maka pengujian aplikasi hanya dapat menghasilkan hasil klasifikasi sesuai dengan model yang dibuat sebelumnya.

Percobaan yang dilakukan yaitu uji menggunakan data acak yang ada pada dataset *diabetes prediction* yang sebelumnya data tersebut telah dilakukan uji coba melalui *Jupyter Notebook.* Berikut gambar-gambar hasil output ketika melakukan klasifikasi dengan data acak.



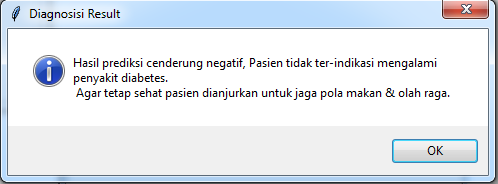
Gambar 4.15: Klasifikasi dengan hasil output terindikasi atau kelas 1

Gambar 4.15 diatas merupakan tampilan output hasil dari aplikasi prediksi diabetes jika melakukan input data yang terdiagnosa diabetes.



Gambar 4.16: *Messagebox* hasil output jika beberapa data kosong

Gambar 4.15 diatas merupakan tampilan output hasil dari aplikasi prediksi diabetes jika ada data input yang kosong.



Gambar 4.17: Klasifikasi dengan hasil output tidak terindikasi atau kelas 0

Gambar 4.15 diatas merupakan tampilan output hasil dari aplikasi prediksi diabetes jika melakukan input data yang tidak terdiagnosa diabetes.

**4.6 Uji Coba Black Box**

Uji coba pada aplikasi adalah proses untuk memastikan apakah semua fungsi sistem bekerja dengan baik, dan mencari apakah masih ada kesalahan pada sistem. Pengujian atau *testing* aplikasi sangat penting untuk dilakukan. Pengujian ini bertujuan untuk menjamin kualitas aplikasi, dan juga menjadi peninjauan terakhir terhadap spesifikasi, design dan pengkodean.

Pada tahap ini dilakukan uji coba dengan menggunakan metode black box. Uji coba tersebut dilakukan untuk mengetahui apakah aplikasi dapat berfungsi dengan baik dan berjalan sebagaimana mestinya. Uji coba ini dilakukan untuk aplikasi berbasis desktop pendeteksi penyakit diabetes secara dini. Berikut adalah tabel dari uji coba black box:

Tabel 4.7 Uji Coba Aplikasi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Pengujian** | **Hasil yang diharapkan** | **Hasil Pengujian** | **Kesimpulan** |
| 1 | Menjalankan Aplikasi | Menjalankan Aplikasi dan menampilkan tampilan berupa form aplikasi | Aplikasi berjalan dan menampilkan form aplikasi dengan baik tanpa error dan hambatan | Berhasil |
| 2 | Tampilan input data | Menginput data-data seperti pregnancies, Glucose, Diastolic Blood Pressure, Triceps Skin, BMI, Diabetes Pedigree Function, Age dengan baik | Aplikasi menampilkan tampilan wadah input data berupa textbox , label,dan button sesuai urutan data data yang dibutuhkan seperti pregnancies, Glucose, Diastolic Blood Pressure, Triceps Skin, BMI, Diabetes Pedigree Function, Age dengan baik | Berhasil |
| 4 | Hitung Hasil BMI | Melakukan perhitungan untuk parameter BMI dengan baik | BMI dapat dihitung dengan baik dengan memasukkan weigth dan height pada masing masing textbox | Berhasil |
| 3 | Proses Klasifikasi | Melakukan proses klasifikasi terhadap data yang input dengan baik dan benar | Aplikasi dapat melakukan proses klasifikasi terhadap data yang telah diinput dengan baik tanpa kendala atau error. | Berhasil |
|  |  |  |  |
|  |  |  |  |
| 4 | Tampilan Output *MessageBox* | Menampilkan hasil output berupa *message box* yang informasi hasil klasifikasi dengan baik | Aplikasi dapat dengan baik menampilkan output berupa *message box* sebagai informasi detail hasil klasifikasi tanpa error | Berhasil |

Setelah melakukan uji coba dengan metode black box sesuai dengan tabel 4.2, maka dapat disimpulkan bahwa aplikasi untuk user dan admin dapat menjalankan fungsi-fungsinya dengan baik dan berjalan sesuai dengan apa yang diharapkan.