PENCARIAN NILAI *k* TERBAIK PADA *k*-NN MENGGGUNAKAN *GENETIC ALGORITHM* (GA) DALAM PENANGANAN *MISSING VALUE* PADA KLASIFIKASI *TUBERCULOSIS*

SKRIPSI

Disusun Oleh :
RUSNIA DYAH WARDANI 180411100018

Dosen Pembimbing 1 : Eka Mala Sari Rochman, S.Kom., M.Kom.
(19840716 200812 2 001)

Dosen Pembimbing 2 : Dr. Bain Khusnul Khotimah, ST., M.Kom.
(19800325 200312 2 002)

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS TRUNOJOYO MADURA

2022

PENCARIAN NILAI *k* TERBAIK PADA *k*-NN MENGGGUNAKAN *GENETIC ALGORITHM* (GA) DALAM PENANGANAN *MISSING VALUE* PADA KLASIFIKASI *TUBERCULOSIS*

SKRIPSI

Diajukan untuk Memenuhi Persyaratan Penyelesaian Studi Strata Satu (S1) dan Memperoleh Gelar Sarjana Komputer (S.Kom) di Universitas Trunojoyo Madua

Oleh :

Rusnia Dyah Wardani

180411100018

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK IFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS TRUNOJOYO MADURA
2022

HALAMAN PENGESAHAN

PENCARIAN NILAI *k* TERBAIK PADA *k*-NN MENGGGUNAKAN *GENETIC ALGORITHM* (GA) DALAM PENANGANAN *MISSING VALUE* PADA KLASIFIKASI *TUBERCULOSIS*

Skripsi diajukan sebagai salah satu syarat untuk penyelesaian Pendidikan program studi S1 Teknik Informatika Universitas Trunojoyo Madura

Oleh :

Nama : Rusnia Dyah Wardani
NIM : 180411100018

Disetujui oleh Tim Penguji Skripsi : **Tanggal Sidang :**

Eka Mala Sari Rochman, S.Kom., M.Kom. **NIP. 19840716 200812 2 001**	_____	**(Pembimbing I)**
Dr. Bain Khusnul Khotimah, ST., M.Kom. **NIP. 19800325 200312 2 002**	_____	**(Pembimbing II)**
Husni, S.Kom., M.Kom. **NIP. 19790722 200312 1 001**	_____	**(Penguji I)**
Dr. Fika Hastarita Rachman, S.T., M.Eng **NIP. 19830305 200604 2 002**	_____	**(Penguji II)**
Dr. Aeri Rachmad, S.T., M.T. **NIP. 19800223 200812 1 001**	_____	**(Penguji III)**

Bangkalan, 23 Februari 2023
Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Yeni Kustiyahningsih, S.Kom., M.Kom.
NIP. 19770921 200812 2 002

HALAMAN PERNYATAAN ORISINILITAS

Saya yang bertanda tangan di bawah ini :

Nama	:	Rusnia Dyah Wardani
NIM	:	180411100018
Judul Tugas Akhir	:	Pencarian Nilai *k* Terbaik pada *k*-NN Mengggunakan *Genetic Algorithm* (GA) dalam Penanganan *Missing Value* pada Klasifikasi *Tuberculosis*

Menyatakan dengan sebenarnya bahwa penulisan tugas akhir ini berdasarkan hasil penelitian, pemikiran dan pemaparan saya sendiri, bukan merupakan karya pihak lain serta belum pernah diajukan untuk mendapatkan gelar akademik Sarjana Komputer baik dalam lingkungan Universitas Trunojoyo Madura maupun di Perguruan Tinggi lain. Jika terdapat karya orang lain sebagai referensi pada penelitian ini, maka saya akan mencantumkan sumber yang jelas.

Dengan pernyataan ini saya buat dengan sesungguhnya dan apabila terbukti tugas akhir ini sebagian atau seluruhnya merupakan hasil plagiasi atau terdapat halhal yang tidak sesuai dengan pernyataan diatas, saya sanggup menerima sanksi akademis sesuai peraturan yang berlaku di Universitas Trunojoyo Madura.

Bangkalan, 23 Februari 2023
Pembuat Penyataan,

Rusnia Dyah Wardani
NIM. 180411100018

PENCARIAN NILAI *k* TERBAIK PADA *k*-NN MENGGGUNAKAN *GENETIC ALGORITHM* (GA) DALAM PENANGANAN *MISSING VALUE* PADA KLASIFIKASI *TUBERCULOSIS*

Rusnia Dyah Wardani
(180411100018)

Dosen Pembimbing I : Eka Mala Sari Rochman, S.Kom., M.Kom.
Dosen Pembimbing II : Dr. Bain Khusnul Khotimah, ST., M.Kom.

ABSTRAK

Tuberculosis atau tuberkulosis paru merupakan salah satu penyakit pernapasan yang berbahaya yang menyerang paru-paru. Selain itu bakteri TB juga dapat menyerang organ tubuh lain yang biasa disebut dengan TB ekstra paru. Upaya agar dapat mengetahui jenis TB yang diderita pasien adalah dengan melalui beberapa pemeriksaan, seperti pendataan umur pasien, jenis kelamin pasien, status HIV pasien, riwayat diabetes pasien, melakukan foto toraks, dan lainnya. Di era teknologi saat ini, proses pengklasifikasian untuk deteksi awal dalam menentukan jenis *tuberculosis* dapat dibantu dengan menggunakan pengenalan Algoritma *Long Short Term Memory* (LSTM). Kelemahan dari dataset adalah adanya kelengkapan data yang hilang dari suatu fitur sehingga mempengaruhi nilai dari hasil klasifikasi. Data yang hilang dapat diatasi dengan melakukan imputasi menggunakan *k*-Nearest Neighboor (*k*-NN). Namun, metode *k*-NN memiliki kelemahan dalam pemilihan nilai *k* yang dapat mempengaruhi kinerja klasifikasi. Peneliti menggunakan *Genetic Algorithm* (GA) sebagai optimasi dalam menentukan nilai *k* terbaik pada *k*-NN untuk penanganan *missing value* pada klasifikasi *tuberculosis*.

Hasil penelitian menunjukkan bahwa nilai terbaik pada parameter GA adalah *crossover rate* 0.7, *mutation rate* 0.04, *population size* 120, dan *max generation* 1600, yang menghasilkan *fitness score* terbaik dalam waktu paling singkat. Sementara itu, kombinasi terbaik dari jumlah *neuron* dan *epoch* berbeda untuk setiap data imputasi *k*-NN. Klasifikasi LSTM dengan imputasi *k*-NN+GA

dianggap lebih baik daripada metode klasifikasi LSTM yang menggunakan imputasi *k*-NN tanpa optimasi GA, karena dapat mencapai akurasi tertinggi yaitu 98,5756% dengan waktu yang lebih singkat.

Kata kunci : *Tuberculosis, Missing Value, k-Nearest Neighboor* (k-NN), *Genetic Algorithm* (GA), *Long Short Term Memory* (LSTM)

KATA PENGANTAR

Segala puji dan syukur selalu penulis panjatkan atas kehadirat Allah SWT yang telah memberi rahmat, hidayat, serta karunia-Nya, tidak lupa shalawat serta salam penulis curahkan kepada nabi besar Muhammad SAW beserta para sahabat dan keluarganya, berkat dorongan dan bantuan dari berbagai pihak yang telah membantu menyelesaikan tugas akhir ini dengan judul "PENCARIAN NILAI *k* TERBAIK PADA *k*-NN MENGGGUNAKAN *GENETIC ALGORITHM* (GA) DALAM PENANGANAN *MISSING VALUE* PADA KLASIFIKASI *TUBERCULOSIS*". Pada kesempatan ini, penulis ingin menyampaikan rasa terimakasih kepada:

1. Keluarga yang selalu mendukung dan selalu mendoakan keberhasilan penulis.
2. Ibu Eka Mala Sari Rochman, S.Kom., M.Kom. dan Ibu Dr. Bain Khusnul Khotimah, ST., M.Kom. selaku dosen pembimbing yang telah memberikan arahan dan motivasi untuk penulis, sehingga penulis bisa menyelesaikan skripsi.
3. Ibu Dr. Fika Hastarita Rachman, S.T., M.Eng selaku Koordinator Prodi Teknik Informatika dan Ibu Dr. Yeni Kustiyahningsih, S.Kom., M.Kom. selaku Ketua jurusan Teknik Informatika.
4. Bapak dan Ibu Dosen Teknik Informatika di Universitas Trunojoyo Madura yang telah memberikan ilmu pengetahuan dalam proses perkuliahan.

Dalam penulisan skripsi ini, penulis menyadari bahwa masih banyak kekurangan. Oleh karena itu, kritik dan saran dari semua pihak sangat diharapkan oleh penulis untuk memberikan perubahan positif agar menjadi lebih baik. Demikian, penulis menyusun laporan skripsi ini dengan harapan dapat bermanfaat bagi semua pihak dan juga bagi penulis sendiri.

Bangkalan, 23 Februari 2023

Rusnia Dyah Wardani
NIM. 180411100018

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR ALGORITMA

DAFTAR KODE PROGRAM

BAB I PENDAHULUAN

1.1 Latar Belakang

Penyakit adalah istilah medis yang digambarkan sebagai gangguan dalam fungsi tubuh yang menyebabkan berkurangnya kapasitas kerja pada organ tubuh. Penyakit dapat terjadi ketika keseimbangan dalam tubuh tidak dapat dipertahankan. Keadaan sakit terjadi pada saat seseorang tidak lagi berada dalam kondisi sehat yang normal [1]. Tingkat keparahan suatu penyakit dapat menyebabkan kematian pada penderitanya. Penyakit dapat menyerang orang tubuh manapun, termasuk dengan organ pernapasan.

Salah satu penyakit pernapasan yang berbahaya adalah penyakit *Tuberculosis* atau tuberkulosis paru. *Tuberculosis* atau dikenal dengan TB Paru merupakan penyakit yang mematikan setelah HIVAIDS [2]. Penyakit tuberkulosis paru adalah penyakit menular langsung yang disebabkan oleh bakteri *Mycobacterium tuberculosis* yang sebagian besar menyerang paru-paru. Selain itu bakteri TB juga dapat menyerang organ tubuh lain yang biasa disebut dengan TB ekstra paru. TB paru adalah bentuk paling umum dari sekitar 80% dari semua penderita. TB yang menyerang jaringan paru-paru merupakan bentuk penyakit yang mudah TB menular. [3]. Oleh karena itu, berdasarkan lokasi atau organ tubuh yang sakit, *Tuberculosis* atau TB dapat diklasifikasikan ke dalam dua *class*, yaitu TB paru dan TB ekstra paru.

Klasifikasi merupakan pengelompokan data yang mana data tersebut mempunyai kelas label ataupun sebuah target. Dengan berkembangnya ilmu pengetahuan, dalam melakukan pengklasifikasian suatu data atau objek, dapat dibantu dengan menggunakan sistem yang menerapkan ilmu olah data. Telah ada beberapa metode yang digunakan untuk melakukan kegiatan klasifikasi, seperti metode *Long Short Term Memory* (LSTM). *Long Short Term Memory* (LSTM) adalah sebuah jenis khusus dari algoritma *Recurrent Neural Network* (RNN) yang mampu mempelajari ketergantungan jangka panjang. Berdasarkan penelitian yang telah dilakukan oleh Arfan dan Lussiana, 2020, LSTM memiliki nilai *loss* yang lebih baik jika dibandingkan dengan SVR (*Support Vector Regression*). Sehingga penggunaan LSTM mampu melakukan prediksi dengan hasil yang lebih akurat [4].

Dalam beberapa kasus yang berhubungan dengan klasifikasi terdapat beberapa permasalahan, salah satunya adalah masalah *missing value* pada data yang digunakan. *Missing value* merupakan suatu permasalahan dimana terdapat ketidak lengkapan nilai dalam suatu data. *Missing value* dapat terjadi karena beberapa faktor. Beberapa faktor tersebut, diantaranya adalah pengumpulan data dari sebuah pengamatan tidak berlangsung baik dan permasalahan responden yang tidak memberikan jawaban dari beberapa pertanyaan tertentu pada saat peninjauan. Adanya *missing value* dapat mempengaruhi hasil keakuratan dari proses oleh data dalam suatu sistem. Penelitian yang dilakukan oleh Achmad Fikri Sallaby dan Azlan, 2021 menjelaskan bahwa klasifikasi dapat dilakukan jika data sudah lengkap. Berdasarkan penelitian tersebut, pengklasifikasian yang melalui proses imputasi terhadap data kosong terlebih dahulu, memiliki nilai akurasi yang lebih tinggi jika dibandingkan dengan proses pengklasifikasian tanpa melalui imputasi [5]. Imputasi merupakan upaya mengatasi *missing value* dengan melakukan perhitungan untuk menentukan nilai pengganti data. Hal ini yang mendorong perlunya mencari metode khusus untuk penanganan imputasi terhadap permasalahan yang berkaitan dengan penanganan *missing value*.

Algoritma *k-Nearest Neighboor* (*k*-NN) merupakan salah satu metode imputasi yang dapat digunakan dalam mengatasi permasalahan *missing value*. Pada penelitian yang dilakukan oleh Taufiq Rizaldi, dkk, 2018 membahas bahwa penangangan *missing value* dengan menggunakan metode *k*-NN memiliki nilai MSE (*Mean Square Error*) yang lebih kecil, jika dibandingkan dengan menggunakan metode *Naïve Bayes*. Dimana, apabila semakin kecil nilai MSE, maka kinerja dari metode tersebut dianggap semakin baik [6]. Langkah penting dalam Algoritma *k-Nearest Neighboor* (*k*-NN) adalah menentukan nilai nilai *k* dan jarak antar satu *instance* dengan seluruh *instance* pada *dataset*. Dimana sejumlah *k* tetang*Gate*rdekat dari *dataset* dijadikan sebagai nilai estimator. Penentuan nilai nilai *k* harus tepat dan benar. Hal tersebut karena, penentuan nilai nilai *k* yang tidak tepat dapat menurukan hasil akurasi

Pada beberapa penelitian yang telah dilakukan sebelumnya, seperti pada penelitian yang dilakukan oleh Susanti, dkk, 2018 [7] dan pada penelitian oleh Achmad Fikri Sallaby dan Azlan, 2021 [5]. Pada penelitian tersebut, penentuan

nilai *k* dalam penggunaan algoritna *K-Nearest Neighboor* (*k*-NN) untuk penanganan *missing value* masih dilakukan secara sembarang. Hal ini berpotensi menyebabkan kurang maksimalnya nilai akurasi dari pengolahan data yang dilakukan.

Untuk mengatasi masalah tersebut, maka diperlukan upaya dalam menentukan nilai *k* yang tepat dengan melakukan optimasi terhadap nilai k pada algoritma *K-Nearest Neighboor* (*k*-NN). Berdasarkan penelitian yang telah dilakukan oleh Abidatul Izzah dan Nurhayatin, 2013, pencarian nilai *k* terbaik pada *k*-NN menggunakan GA menghasilkan kinerja yang lebih baik jika dibandingkan dengan metode *k*-NN yang menggunakan imputasi *mean* dan median [8]. *Genetic Algorithm* (GA) digunakan sebagai upaya dalam memecahkan permasalahan optimasi tersebut. *Genetic Algorithm* (GA) merupakan salah satu dari algoritma komputasi evolusi. Seperti pada penelitian yang dilakukan oleh Ispandi dan Romi Satria Wahon, 2015. Penelitian tersebut melakukan optimasi parameter SVM menggunakan GA dalam kasus prediksi pemasaran langsung [9]. Penelitian lain yang melakukan proses optimasi juga dilakukan oleh Warid Yunus, 2018. Penelitian tersebut melakukan optimasi parameter *k* pada *k*-NN menggunakan PSO dalam kasus prediksi penyakit ginjal kronik [10]. Berdasarkan kedua metode optimasi tersebut (GA dan PSO), optimasi menggunakan GA memiliki akurasi yang lebih unggul 1,64 % jika dibandingkan dengan optimasi mengguanakan PSO. Dengan menggunakan algoritma ini, *Genetic Algorithm* (GA) dapat mencari solusi terbaik dari kemungkinan – kemungkinan solusi yang tersedia. Pada penelitian yang dilakukan ini, solusi terbaik dari *Genetic Algorithm* (GA) digunakan sebagai nilai *k* yang dalam menyelesaikan imputasi *k-Nearest Neighboor* (*k*-NN). Dalam penelitian ini penulis merancang sebuah sistem pencarian nilai *k* terbaik milik Algoritma *k-Nearest Neighboor* (*k*-NN) menggunakan *Genetic Algorithm* (GA) untuk penanganan *missing value* pada data *tuberculosis*.

1.2 Perumusan Masalah

Berdasarkan latar belakang di atas, maka terbentuklah beberapa rumusan masalah yang dijelaskan pada sub bab berikut.

1.2.1 Permasalahan

Hilangnya nilai dari suatu data dapat mempengaruhi hasil akurasi dalam proses pengklasifikasian melalui metode LSTM. Dalam penanganan *missing value, k-Nearest Neighboor* (*k*-NN) merupakan metode yang paling populer. Namun, metode ini memiliki kelemahan pada pemilihan nilai *k* yang dapat mempengaruhi kinerja klasifikasi. *Genetic Algorithm* (GA) digunakan sebagai metode optimasi dalam menentukan nilai *k* terbaik pada *k*-NN. Namun, pemilihan parameter GA juga mempengaruhi solusi yang dihasilkan.

1.2.2 Metode Usulan

Metode usulan adalah dengan menggunakan *Genetic Algorithm* (GA) untuk mencari nilai *k* terbaik pada metode *k-Nearest Neighbor* (*k*-NN) dalam menangani data yang hilang. Namun, sebelumnya dilakukan beberapa percobaan untuk menguji berbagai parameter pada GA agar menghasilkan solusi (nilai *k*) yang terbaik. Dengan menggunakan nilai k yang terbaik, proses imputasi *k*-NN dapat dilakukan untuk mengisi data yang hilang dan memastikan akurasi klasifikasi LSTM yang lebih baik.

1.2.3 Pertanyaan Penelitian

Berikut pertanyaan – pertanyaan penelitian terkait proposal skripsi berjudul "Pencarian Nilai *k* Terbaik pada *k*-NN Mengggunakan *Genetic Algorithm* (GA) dalam Penanganan *Missing Value* pada Klasifikasi *Tuberculosis*" :

1. Berapa nilai terbaik dari parameter, meliputi *crossover rate* (*cr*), *mutation rate* (*mr*), *population size*, dan *maximum generation* pada *Genetic Algorithm*?
2. Berapa nilai *neuron* dan *epoch* terbaik pada *Long Short Term Memory*?
3. Berapa perbandingan nilai evaluasi berupa akurasi dari hasil pengklasifikasian *tuberculosis* menggunakan *k*-NN+GA+LSTM dengan hasil pengklasifikasian *tuberculosis* menggunakan *k*-NN+ LSTM tanpa optimasi?

1.3 Tujuan Dan Manfaat

Berdasarkan rumusan masalah dan usulan solusi pada sub bab sebelumnya, didapatkan tujuan dan manfaat yang dijelaskan pada sub bab berikut.

1.3.1 Tujuan

Adapun tujuan yang dihasilkan dari peneltian ini meliputi :

1. Untuk mengetahui nilai terbaik dari parameter, meliputi *crossover rate (cr)*, *mutation rate* (*mr*), *population size*, dan *maximum generation* pada *Genetic Algorithm*.
2. Untuk mengetahui nilai *neuron* dan *epoch* terbaik pada klaifikasi LSTM melalui varian imputasi *k*-NN.
3. Untuk mengetahui berapa perbandingan nilai evaluasi berupa akurasi dari hasil pengklasifikasian *tuberculosis* menggunakan *k*-NN+GA+LSTM dengan hasil pengklasifikasian *tuberculosis* menggunakan *k*-NN+ LSTM tanpa optimasi.

1.3.2 Manfaat

Manfaat yang didapatkan dalam usulan penelitian yang dilakukan yaitu, dapat menghasilkan sebuah pengetahuan baru. Dengan melakukan optimasi nilai Algoritma *k-Nearest Neighboor* (*k*-NN) menggunakan *Genetic Algorithm* (GA), maka dapat menghasilkan *output* dengan akurasi yang lebih baik.

1.4 Batasan

Beberapa batasan dalam usulan penelitian yang telah dilakukan meliputi :

1. Penelitian dilakukan di wilayah Kabupaten Bangkalan, Provinsi Jawa Timur.
2. Penelitian menggunakan data *tuberculosis* milik RSUD Syarifah Ambami Rato Ebu pada tahun 2017 sampai dengan 2020 yang berjumlah 985 data.
3. Pengklasifikasian data *tuberculosis* menggunakan 6 buah atribut atau parameter, meliputi umur, jenis kelamin, foto toraks, status HIV, riwayat diabetes, dan hasil TCM (Test Cepat Molekuler).
4. Terdapat 741 data yang memiliki nilai *missing value* pada seluruh atribut.
5. Tidak ada korelasi antar atribut.
6. Pengklasifikasian data *tuberculosis* menghasilkan 2 buah *class*, yaitu *class* paru dan ekstra paru.

1.5 Sistematika Proposal

Beberapa sistematika proposal penelitian ini disusun ke dalam tiga bab, dengan rincian sebagai berikut.

BAB I PENDAHULUAN

Bab I mengulas mengenai latar belakang, perumusan masalah, tujuan dan manfaat, batasan masalah, serta sistematika proposal.

BAB II KAJIAN PUSTAKA

Bab II membahas mengenai beberapa ilmu terkait metode usulan pilihan dan beberapa penelitian terkait.

BAB III METODE PENELITIAN

Bab III mengulas mengenai jenis penelitian, langkah-langkah penelitian, perancangan sistem, *dataset* yang digunakan, metode evaluasi, skenario ujicoba, dan perkiraan jadwal penelitian.

BAB IV HASIL DAN PEMBAHASAN

Bab IV mengulas mengenai hasil implementasi sesuai arsitektur sistem dan hasil uji coba sesuai dengan skenario uji coba.

BAB V KESIMPULAN DAN SARAN

Bab V mengulas mengenai kesimpulan dan saran. Kesimpulan didapatkan dari hasil uji coba yang telah dilakukan. Kemudian, saran digunakan untuk pengembangan penelitian terkait selanjutnya.

BAB II TINJAUAN PUSTAKA

2.1 Landasan Teori

2.1.1 *Tuberculosis*

Tuberculosis atau tuberkulosis paru (TB) adalah penyakit menular yang disebabkan oleh *Microbacterium tuberculosis* [3], yang merupakan salah satu dari penyakit saluran pernapasan bagian bawah, yang sebagian besar di jaringan paru dari infeksi paru-paru. Nama Tuberkulosis berasal dari *tuberkel* yang berarti tonjolan kecil dan keras yang terbentuk waktu sistem kekebalan membangun tembok mengelilingi bakteri dalam paru. TB paru ini bersifat menahun dan secara khas ditandai oleh pembentukan *granuloma* dan menimbulkan *nekrosis* jaringan.

Sebagian besar bakteri TB menyerang paru-paru. Selain itu bakteri TB juga dapat menyerang organ tubuh lain yang biasa disebut dengan TB ekstra paru. TB paru adalah bentuk paling umum dari sekitar 80% dari semua penderita. TB yang menyerang jaringan paru-paru merupakan bentuk penyakit yang mudah TB menular. TB ekstra paru merupakan salah satu bentuk penyakit TB yang menyerang organ tubuh selain paru [3]. TB pada dasarnya tidak pandang bulu karena bakteri TB ini dapat menyerang seluruh organ tubuh.

2.1.1.1 *Tuberculosis* Paru

Tuberculosis paru adalah TB yang menyerang jaringan (parenkim) paru, kecuali pleura (selaput paru) dan kelenjar pada hilus. Pada program TB nasional, penemuan bakteri tahan asam (BTA) melalui pemeriksaan dahak mikroskopis merupakan diagnosis utama. Pemeriksaan lain seperti foto toraks, biakan dan uji kepekaan dapat digunakan sebagai penunjang diagnosis sepanjang sesuai dengan indikasinya [11].

Pemeriksaan bakteriologis yang dilakukan adalah dengan pemeriksaan mikroskopis, tes cepat molekuler (TCM) dan biakan. Pemeriksaan TCM (Tes cepat molekuler) merupakan perkembangan alat diagnostik yang dapat digunakan untuk mendeteksi adanya bakteri *mikrobakterium tuberculosis* (MTB) melalui dahak. Selain melakukan pemeriksaan foto toraks, status HIV, dan melakukan tes cepat molekuler, pasien terindikasi TB juga harus melakukan pemeriksaan tambahan berupa pemeriksaan gula darah.

2.1.1.2 *Tuberculosis* Ekstra Paru

TB yang menyerang organ tubuh lain selain paru, misalnya pleura, selaput otak, selaput jantung (*pericardium*), kelenjar *lymfe*, tulang, persendian, kulit, usus, ginjal, saluran kencing, alat kelamin, dan lain-lain. Diagnosis pasti pada pasien TB ekstra paru dilakuan dengan pemeriksaan klinis, bakteriologis dan atau histopatologis dari contoh uji yang diambil dari organ tubuh yang terkena [11].. Selain melakukan pemeriksaan foto toraks, status HIV, dan melakukan tes cepat molekuler, pasien terindikasi TB juga harus melakukan pemeriksaan tambahan berupa pemeriksaan gula darah.

2.1.2 *Data Mining*

Data mining merupakan suatu upaya dalam memperoleh informasi yang berguna, dimana informasi tersebut didapatkan dari basis data yang besar dan perlu dilakukan ekstraksi. Hal tersebut perlu dilakukan agar dapat menjadi informasi baru yang dapat digunakan untuk membantu dalam hal pengambilan keputusan [12]. Dalam upaya mengolah data yang jumlahnya besar menjadi sebuah informasi atau pengetahuan diperlukan suatu teknik yang dinamakan dengan *data mining*. Proses olah data dalam *data mining* memerlukan algoritma – algoritma untuk melakukan ekstraksi menjadi informasi.

2.1.3 *Missing Value*

Missing value merupakan permasalahan dimana terdapat ketidak lengkapan nilai dalam suatu data. Adanya *missing value* dapat mempengaruhi hasil keakuratan dari proses oleh data dalam suatu sistem [7].

Penyebab terjadinya *missing value* dapat disebabkan oleh beberapa faktor. Faktor tersebut antara lain adalah kesalahan pada saat entri data, responden tidak mampu menjawab pertanyaan, dan kurangnya informasi pada saat pengumpulan data.

Missing value hanya berpengaruh kecil dalam hasil keakuratan proses olah data apabila jumlah kasus *missing value* pada data hanya sedikit. Namun apabila dalam suatu data memiliki jumlah kasus *missing value* yang sangat banyak, maka pengaruh keakuratan dari hasil proses olah data semakin berkurang [13]. Oleh

sebab itu, maka dibutuhkanlah penanganan khusus dalam mengatasi kasus *missing value* tersebut.

2.1.3.1 Tipe *Missing Value*

Ketidaklengapan suatu data dapat didefinisian sebagai suatu tipe dalam *missing value*. Terdapat 3 jenis *missing value*, diantaranya adalah sebagai berikut [14].

1. *Missing Completely at Random* (MCAR)

   Missing Completely at Random (MCAR) merupakan hilangnya data secara acak. Hilangnya nilai data milik atribut tidak berkaitan dengan data dari pengamatan.
2. *Missing at Random* (MAR)

   Missing at Random (MAR) merupakan hilangnya nilai data milik atribut yang berkaitan dengan data dari pengamatan.
3. *Not Missing at Random* (NMAR)

   Not Missing at Random (NMAR) merupakan hilangnya nilai data milik atribut yang berkaitan dengan atribut itu sendiri.

Pada penelitian ini, jenis missing value yang terjadi adalah *Missing Completely at Random* (MCAR). *Missing value* pada dataset penelitian ini dapat digolongkan sebagai missing completely at random karena tidak ada pola atau hubungan apapun antara data yang hilang dengan data yang ada dalam dataset. Dalam hal ini, data yang hilang pada atribut foto toraks, status HIV, dan Riwayat diabetes tidak memiliki hubungan dengan atribut lain seperti umur dan jenis kelamin. Sehingga, data yang hilang dianggap sebagai kebetulan semata tanpa ada faktor yang mempengaruhi keberadaannya di dataset.

2.1.3.2 Imputasi *Missing Value* dengan *Most Frequent*

Imputasi adalah tindakan mengganti data yang hilang dengan perkiraan statistik dari nilai yang hilang. *Most frequent* merupakan salah satu upaya atau teknik dalam imputasi *missing value*. *Most frequent* sendiri merupakan Teknik imputasi yang mengacu pada nilai yang paling sering muncul [15]. *Most frequent* dapat digunakan pada kasus data yang berjenis kategorikal karena imputasi

menggunakan mean atau median cenderung merepresentasikan nilai rata – rata dari data.

2.1.4 Normalisasi *Standard Scaling*

Normalisasi dengan *Standard Scaling* merupakan salah satu metode transformasi data yang membantu menghilangkan bias pada data dengan cara mengubah distribusi data menjadi distribusi normal. Normalisasi *Standard Scaling* dibutuhkan ketika fitur-fitur pada data memiliki skala yang berbeda-beda, sehingga data tersebut tidak dapat diolah dengan baik oleh algoritma pembelajaran mesin. Standard scaling dilakukan untuk mengubah nilai-nilai dari suatu fitur ke dalam skala yang sama, sehingga memiliki mean yang sama dan standard deviation yang sama [16].

Dengan demikian, setelah dilakukan standard scaling, setiap fitur dari suatu data dapat memiliki distribusi yang sama. Hal tersebut berguna agar setiap fitur dapat dianggap sama dalam analisis atau model yang dibangun, sehingga tidak ada satu fitur yang dominan dibandingkan fitur lainnya.

Cara kerja *Standard Scaling* adalah dengan menghitung rata-rata (*mean*) dan standar deviasi (*standard deviation*) dari setiap fitur, kemudian mengurangi rata-rata dan membagi dengan standar deviasi dari setiap fitur. Hasil dari transformasi ini adalah fitur-fitur yang memiliki mean 0 dan standar deviasi 1. Berikut adalah persamaan *Standard Scalling* [17].

$$x^* = \frac{x - \mu}{\sigma} \tag{2.1}$$

$$\sigma = \sqrt{\frac{\sum(x-\mu)^2}{n}} \tag{2.2}$$

Keterangan :

x^* = Hasil normalisasi sample data

x = Sample data

μ = Rata – rata dari semua sample data

σ = Standard deviation dari semua sample data

n = Jumlah data

2.1.5 Algoritma *k-Nearest Neighboor*

Algoritma *k-Nearest Neighboor* (*k*-NN) merupakan suatu metode yang digunakan untuk melakukan klasifikasi terhadap obyek berdasarkan beberapa data yang jaraknya paling dekat dengan obyek tersebut [7]. Dalam penggunaan Algoritma *k-Nearest Neighboor* (*k*-NN) memiliki keuntungan berupa algoritma ini dapat digunakan untuk data yang bersifat kulitatif maupun kuantitatif. Berikut adalah penjelasan mengenai cara kerja Algoritma *k-Nearest Neighboor* (*k*-NN).

1. Menentukan nilai *k*

   Langkah penting dalam Algoritma *k-Nearest Neighboor* (*k*-NN) adalah menentukan nilai *k* dimana sejumlah *k* tetang*Gate*rdekat dari *dataset* dijadikan sebagai nilai estimator menentukan jumlah observasi. Penentuan nilai *k* harus tepat dan benar. Hal tersebut karena, penentuan nilai k yang tidak tepat dapat menurukan kinerja klasifikasi [18].

2. Menghitung jarak antar data

   Dalam perhitungan jarak atau dikenal dengan *dissimilarity*, terdapat 4 macam perhitungan yang dapat digunakan, diantaranya adalah perhitungan jarak *Manhattan*, jarak *Euclidean*, jarak *Minkowski*, dan jarak *Chebychev*. Berdasarkan penelitian yang dilakukan oleh M. Nishom, 2019 menjelaskan bahwa tingkat akurasi metode dengan menggunakan perhitungan jarak menggunakan persamaan *Euclidean* memiliki nilai akurasi yang lebih tinggi jika dibanding dengan persamaan *Manhattan* dan *Minkowski* [19]. Oleh karena itu, perhitungan jarak pada penelitian ini menggunakan perhitungan jarak *Euclidean*. Berikut adalah persamaan untuk menghitung jarak *Euclidean*.

$$d_{euc(i,j)} = \sqrt{\sum_{n=1}^{p}\left(x_{in} - x_{jn}\right)^2} \tag{2.3}$$

   Keterangan :

   $d_{euc(i,j)}$ = Jarak *Euclidean* antara data *i* dengan data *j*

   p = Dimensi data

   x_{in} = Atribut ke-*n* data ke *i* (atribut yang memiliki *missing data*)

   x_{jn} = Atribut ke-*n* data ke *j* (atribut yang tidak memiliki *missing data*)

3. Mengurutkan jarak dan menentukan *k* observasi.

Setelah melakukan perhitungan jarak *Euclidean* terhadap seluruh data pada atribut foto toraks dan atribut riwayat diabetes, kemudian dilanjutkan dengan melakukan pengurutan nilai jarak tersebut. Pengurutan jarak dilakukan secara *ascending* (diurutkan dari terkecil ke terbesar). Pengurutan secara *ascending* tersebut dilakukan karena dengan menggunakan nilai jarak terkecil, maka diharapkan memiliki nilai kemiripan yang besar (nilai yang paling mirip). Setelah dilakukan pengurutan jarak, kemudian menggunakan nilai *k* untuk menentukan batas ruang jarak tetang*Gate*rdekat.

4. Melakukan imputasi berdasarkan *most frequent*.

   Setelah mengurutkan nilai jarak *Euclidean* dan menentukan batas tetangga, kemudian dilanjutkan pada proses imputasi untuk mengisi nilai atau data yang kosong pada atribut. Pada penelitian ini, proses imputasi dilakukan dengan menggunakan teknik *most frequent*. Teknik most frequent digunakan pada data yang memiliki *categorical feature*s [20]. Nilai berdasarkan teknik *most frequent* merupakan nilai yang memiliki frekuensi kemunculan yang tinggi. Semakin sering nilai pada atribut tersebut keluar, maka nilai tersebut yang digunakan untuk imputasi.

2.1.6 *Genetic Algorithm*

Genetic Algorithm (GA) merupakan salah satu dari algoritma komputasi evolusi. Sistem komputasi evolusi dari *Genetic Algorithm* (GA) adalah dengan mensimulasikan evolusi gen untuk memecahkan suatu permasalahan, dengan cara memilih individu terbaik, dan mengoperasikan operator *Genetic Algorithm* (GA) terhadap gen – gen tersebut. Kemudian individu yang memiliki kualitas yang diinginkan menjadi individu terpilih sebagai solusi permasalahan optimasi [21]. Permasalahan optimasi yang dimaksud merupakan permasalahan yang memiliki banyak kemungkinan solusi.

2.1.6.1 Operasi *Genetic Algorithm*

Operasi *Genetic Algorithm* (GA) berisi beberapa tahapan dalam proses *Genetic Algorithm* (GA). Beberapa tahapan tersebut meliputi inisialisasi populasi dan kromosom, perhitungan nilai *fitness*, *crossover*, mutasi, dan perhitungan nilai

fitness baru [8]. Berikut merupakan gambaran dari operasi *Genetic Algorithm* (GA) yang dapat dilihat pada Gambar 2.1.

Gambar 2.1 Tahap Operasi *Genetic Algorithm*

Berdasarkan Gambar 2.1, berikut adalah penjelasan dari alur tahapan operasi *Genetic Algorithm* (GA) untuk permasalahan optimasi.

1. Inisialisai *n* populasi dan kromosom secara *random*.

   Proses pertama ini digunakan sebagai penentuan bentuk individu yang digunakan untuk merepresentasikan kandidat solusi dari permasalahan optimasi [21]. Kumpulan individu atau kromosom yang dibangkitkan secara *random* ini, biasa disebut dengan populasi. Dalam kromosom terdapat bagian dari gen. Gen dapat direpresentasikan dalam bentuk : bit, bilangan real, daftar aturan permutasi, elemen program atau representasi lainnya yang dapat diimplementasikan untuk operator genetika.

2. Menghitung nilai *fitness* tiap kromosom.

   Nilai fitness pada algoritma genetika adalah ukuran atau skor dari kualitas solusi individu dalam populasi. Sistem kerja nilai *fitness* adalah dengan mengukur seberapa tepat suatu individu atau kromosom menjadi solusi dari permasalahan. Pada penelitian ini, untuk menentukan nilai terbaik k pada k-NN melalui optimasi menggunakan GA, nilai fitness ditentukan dengan menggunakan metrik evaluasi mean squared error (MSE) dari klasifikasi k-NN dengan menggunakan library python pada semua pembagian data. Pembagian data dilakukan dengan menggunakan *KFold* dengan nilai *K*=5. Penggunakan nilai *KFold* tersebut didasari karena, *KFold* dengan nilai *K*=5 pada *k*-NN dapat menghasilkan kinerja komputasi yang lebih baik jika dibandingkan dengan nilai *KFold* lainnya [22].Dimana dataset yang digunakan adalah *real dataset* tanpa *missing value*. Kemudian mencari rata-rata dari MSE tersebut dari hasil klasifikasi *k*-NN dengan nilai *k* yang berbeda. Semakin kecil nilai MSE atau nilai *fitness* suatu individu, maka

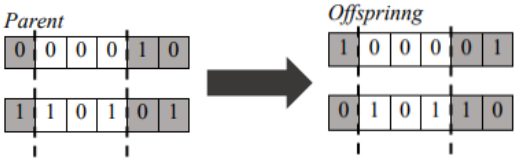
semakin baik performa individu tersebut dalam mencapai tujuan pencarian nilai k terbaik pada *k*-NN menggunakan *GA*.

3. Seleksi *Parents*

   Tahap seleksi *parents* dilakukan dengan tujuan untuk memilih dua kromosom berbeda yang nantinya dilanjutkan ke tahap *crossover*. Pada penelitian ini, proses seleksi parents dilakukan dengan menggunakan *roulette wheel selection*. Metode ini bekerja dengan memperhitungkan nilai fitness dari setiap individu sebagai bobot untuk memilih individu yang dipilih sebagai parent. Metode ini menggunakan teknik acak untuk memastikan bahwa setiap individu memiliki peluang yang sama untuk dipilih sebagai parent.

4. *Crossover*

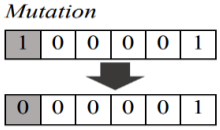
   Pada tahapan ini, pasangan *parents* yang dibangkitkan secara random dari kromosom yang berbeda dilakukan perkawinan silang dengan menukaran gen – gen kromosom dari sepasang *parents* tersebut, sehingga menghasilkan kromosom atau individu baru yang disebut dengan *offspring*. Berikut adaalah contoh proses *crossover* yang dapat dilihat pada Gambar 2.2 [21].



Gambar 2.2 Proses *Crossover*

5. Mutasi

   Kromosom *offspring* yang didapatkan dari proses sebelumnya kemudian digunakan untuk proses mutasi. Proses ini bekerja dengan mengubah gen yang terpilih dari suatu kromosom dengan tujuan menghasilkan individu yang berbeda dari *parents*. Sebagai contoh, representasi kromosom yang digunakan adalah bentuk *string binner* dengan gen ‘0’, maka bermutasi menjadi ‘1’ atau sebaliknya. Proses mutasi digambarkan pada Gambar 2.3 [21].



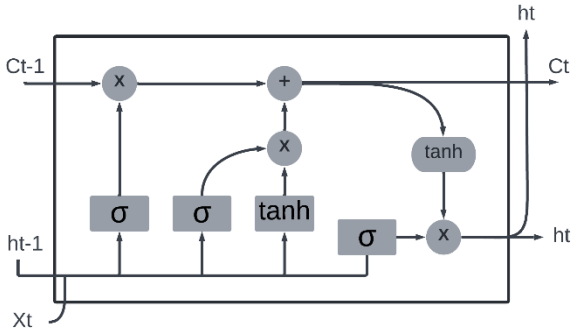
Gambar 2.3 Proses Mutasi

6. *Update* populasi

Proses *update* populasi bertujuan untuk memilih *n* kromosom yang berasal dari gabungan populasi sebelumnya dan *offspring* yang telah dihasilkan. Agar jumlah populasi stabil (sama seperti jumlah populasi awal), maka perlu dilakukan proses seleksi *survivor* berdasarkan nilai *fitness* terbaik dari tiap kromosom tersebut. Lalu, *n* kromosom yang terpilih digunakan sebagai populasi untuk generasi berikutnya. Update populasi dilakukan hingga batas maksimum generasi atau solusi yang didapatkan telah stabil (nilai k tidak berubah).

2.1.7 *Long Short Term Memory*

Long Short Term Memory (LSTM) adalah sebuah jenis khusus dari algoritma *Recurrent Neural Network* (RNN) yang mampu mempelajari ketergantungan jangka panjang. LSTM memiliki empat lapisan jaringan syaraf tunggal. Berikut adalah gambar modul pengulangan LTSM dengan berisi empat *layer* yang dapat dilihat pada Gambar 2.6 [23].



Gambar 2.4 Modul Pengulangan LSTM Empat *Layer*

Pada Gambar 2.4, ht-1 (nilai *hidden state* awal) dan X_t (nilai tiap fitur dalam satu data) digunakan sebagai inputan. LSTM sendiri memiliki empat buah *layer* yang digambarkan dengan kotak berwarna oranye. Ke empat *layer* atau gerbang tersebut antara lain adalah *Forget Gate*, *Input Gate*, *cellstate*, dan *Output Gate*. Kemudian gambar lingkaran adalah mewakili operasi *pointwise*. Penggabungan garis menunjukkan rangkaian (*concatenation*), sementara garis bercabang menunjukkan kontennya disalin dan salinannya masuk ke lokasi yang berbeda. Berikut adalah penjelasan dari tiap gerbang pada LSTM [23].

a. *Forget Gate*

Gerbang pertama atau langkah pertama yaitu *Forget Gate* (f_t). Pada bagian ini informasi yang kurang dibutuhkan atau tidak terlalu memiliki makna terhadap kasus yang diolah dihilangkan menggunakan fungsi sigmoid. Berikut adalah persamaan dalam perhitungan nilai *Forget Gate*.

$$f_t = \sigma(W_{fx}.x_t + W_{fh}.h_{t-1} + b_f) \quad (2.4)$$

Keterangan :

f_t = *Forget Gate*

σ = *Sigmoid*

W_{fx} = Bobot tiap atribut x pada *Forget Gate*

x_t = Nilai tiap atribut x pada data atau individu ke-t

W_{fh} = Bobot *hidden state* pada *Forget Gate*

h_{t-1} = Nilai *hidden state*

b_f = Nilai bias pada *Forget Gate*

b. *Input Gate*

Gerbang kedua yaitu *Input Gate* (i_t). Proses ini melakukan penentuan terhadap informasi tertentu yang diperbarui ke bagian *cell state* dengan menggunakan fungsi aktivasi sigmoid. Pada proses ini juga membentuk kandidat *cell state* baru ($\tilde{C}_t$) menggunakan fungsi aktivasi tanh. Berikut adalag persamaan dalam perhitungan nilai *Input Gate* dan kandidat *cell state* baru.

$$i_t = \sigma(W_{ix}.x_t + W_{ih}.h_{t-1} + b_i) \quad (2.5)$$

Keterangan :

i_t = *Input Gate*

σ = *Sigmoid*

W_{ix} = Bobot tiap atribut x pada *Input Gate*

x_t = Nilai tiap atribut x pada data atau individu ke-t

W_{ih} = Bobot *hidden state* pada *Input Gate*

h_{t-1} = Nilai *hidden state*

b_i = Nilai bias pada *Input Gate*

$$\tilde{C}_t = \tanh(W_{cx}.x_t + W_{ch}.h_{t-1} + b_c) \quad (2.6)$$

Keterangan :

$\tilde{C}_t$ = Kandidat *Cell state*

W_{cx} = Bobot tiap atribut x pada kandidat *cell state*

x_t = Nilai tiap atribut x pada data atau individu ke-t

W_{ch} = Bobot *hidden state* pada kandidat *cell state*

h_{t-1} = Nilai *hidden state*

b_c = Nilai bias pada kandidat *cell state*

c. *Cell State*

Pada tahap ini adalah melakukan *update* atau pembaruan *cell state* lama ke *cell state* baru dengan persamaan sebagai berikut.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \quad (2.7)$$

Keterangan :

C_t = *Cell state*

f_t = *Forget Gate*

C_{t-1} = Nilai *cell state* sebelumnya

i_t = *Input Gate*

$\tilde{C}_t$ = Nilai kandidat *cell state*

d. *Output Gate*

Output Gate terdiri dari *hidden state* sebelumnya dan input sekarang. Pada tahap ini, selain melakukan perhitungan *Output Gate*, juga melakukan perhitungan *hidden state* yang digunakan sebagai *final output*. Berikut adalah persamaan dalam menentukan *Output Gate* dan *hidden state*.

$$o_t = \sigma(W_{ox}.x_t + W_{oh}.h_{t-1} + b_o) \quad (2.8)$$

Keterangan :

o_t = *Output Gate*

σ = *Sigmoid*

W_{ox} = Bobot tiap atribut x pada *Output Gate*

x_t = Nilai tiap atribut x pada data atau individu ke-t

W_{oh} = Bobot *hidden state* pada *Output Gate*

h_{t-1} = Nilai *hidden state*

b_o = Nilai bias pada *Output Gate*

$$h_t = o_t \times \tanh(c_t) \quad (2.9)$$

Keterangan :

h_t = *Hidden state*

o_t = *Output Gate*

C_t = *Cell state*

2.2 Penelitian Terkait

Pada sub bab ini terdapat beberapa penelitian sebelumnya berkaitan dengan penelitian yang sedang dilakukan. Beberapa penelitian tersebut dijabarkan pada Tabel 2.1 berikut.

Tabel 2.1 Penelitian Terkait

Nama Peneliti, Tahun	Metode/Kasus	Hasil
Abidatul Izzah dan Nur Hayatin, 2013 [8]	Algoritma *K-Nearest Neighbor* (*k*-NN) dan Algoritma Genetika (GA)	Pencarian nilai *k* terbaik pada *k*-NN menggunakan GA menghasilkan nilai akurasi yang lebih besar jika dibandingkan dengan imputasi *k-NN*, imputasi mean, dan imputasi median.
Taufiq Rizaldi, Fendik Eko Purnomo, Aji Seto arifianto, 2018 [6]	Algoritma *K-Nearest Neighbor* (*k*-NN) dan *Naïve Bayes*	*k*-NN menghasilkan nilai MSE terkecil senilai 0,1057, lebih baik jika dibandingkan dengan *Naïve Bayes* yang memilliki nilai MSE 2,0876.
Arfan dan Lussiana, 2020 [4]	*Long Short-Term Memory* (LSTM) dan SVR (*Support Vector Regression*)	LSTM memiliki nilai *loss* yang lebih baik jika dibandingkan dengan SVR (*Support Vector Regression*). Sehingga penggunaan LSTM mampu melakukan prediksi dengan hasil yang lebih akurat.
Achmad Fikri Sallaby dan Azlan, 2021 [5]	Algoritma *K-Nearest Neighbor* (*k*-NN)	Penanganan imputasi *missing value* menggunakan *k*-NN pada proses klasifikasi menghasilkan akurasi yang lebih besar 1,3% jika dibandingkan proses klasifikasi tanpa melalui imputasi.
Bain Khusnul Khotimah, Muhammad Syarief, Miswanto, dan Herry Suprajitno, 2021 [24]	*K-Means Clustering* dan Algoritma Genetika (GA)	Optimasi bobot *K-Means Clustering* menggunakan GA untuk penanganan imputasi menghasilkan nilai MSE yang lebih baik dengan nilai MSE terbesar adalah 0,928 dan yang terkecil adalah 0,588.
Saiful Ulya , M.	Algoritma *k-Nearest*	Gabungan *Genetic Algorithm* (GA)

Nama Peneliti, Tahun	Metode/Kasus	Hasil
Arief Soeleman, dan Fikri Budiman, 2021 [25]	*Neighbor* (*k*-NN) dan Algoritma Genetika (GA)	dengan algoritma *k*-NN (GA+*k*-NN) menghasilkan akurasi yag lebih baik sebesar 0,44% jika dibandingkan dengan menggunakan *k*-NN dengan penentuan nilai *k* secara konvensional.

Berdasarkan beberapa penelitian yang telah dilakukan tersebut, dapat disimpulkan bahwa imputasi *missing value* pada data yang diolah cukup berpengaruh terhadap hasil akurasinya. Dalam hal imputasi, metode k-NN merupakan metode yang memiliki kinerja lebih yang baik jika dibandingkan dengan metode *Naïve Bayes* [6]. Kemudian, pencarian nilai *k* terbaik pada *k*-NN juga diperlukan agar dapat menghasilkan akurasi yang lebih maksimal pada proses oleh data nantinya. Pada penelitian ini, metode LSTM digunakan sebagai metode untuk mengolah data dengan melakukan pengklasifikasian letak anatomi *tuberculosis*. Pemilihan metode LSTM tersebut didasarkan karena, kinerja LSTM lebih baik jika dibandingkan dengan SVR [4]. Pengklasifikasian menggunakan LSTM dilakukan untuk menguji bagaimana kinerja dari penerapan imputasi k-NN menggunakan optimasi GA setelah dilakukan proses olah data. Dengan harapan pencarian nilai *k* terbaik PADA *k*-NN dapat menghasilkan kinerja yang maksimal.

BAB III METODE PENELITIAN

3.1 Data

Dataset yang digunakan pada penelitian ini adalah data *Tuberculosis* milik RSUD Syarifah Ambami Rato Ebu pada tahun 2017 sampai dengan 2020. Data *Tuberculosis* yang didapatkan berisi 985 data *record* hasil pemeriksaan pasien. Data yang digunakan dalam penelitian terdiri dari atribut umur, jenis kelamin, foto toraks, status HIV, riwayat diabetes, hasil TCM (Tes Cepat Molekuler), dan dengan *output* adalah lokasi anatomi yang diperlihatkan pada Gambar 3.1.

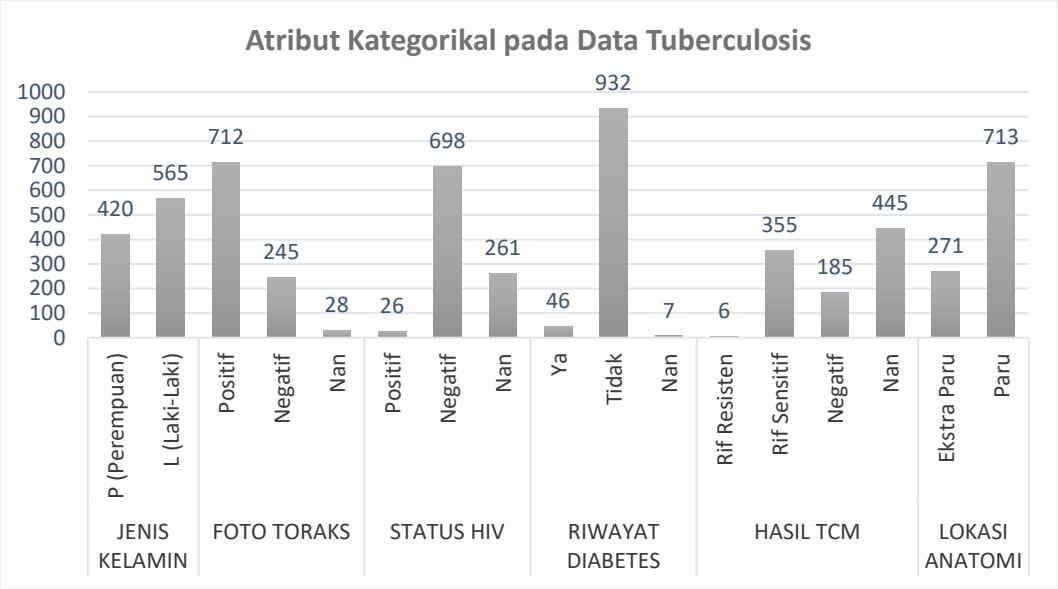
A	B	C	D	E	F	G	H
UMUR	JENIS KELAMIN	KECAMATAN	FOTO TORAKS	STATUS HIV	RIWAYAT DIABETES	HASIL TCM	LOKASI ANATOMI (target/output)
61	P	PULO GADUNG	Positif	Negatif	Ya	Rif Sensitif	Paru
51	P	LABANG	Positif	Negatif	Tidak	Negatif	Paru
81	P	SOCAH	Positif	Negatif	Tidak	Negatif	Paru
48	L	GUNUNG PUTRI	Negatif		Tidak		Ekstra paru
61	L	KAMAL	Positif	Negatif	Tidak	Rif Sensitif	Paru
64	L	GALIS	Positif		Tidak	Rif Sensitif	Paru
52	L	KAMAL	Positif	Negatif	Tidak	Rif Sensitif	Paru
52	L	TANAH MERAH	Negatif		Tidak		Ekstra paru
24	L	TANAH MERAH	Positif	Negatif	Tidak	Negatif	Paru

Gambar 3.1 Contoh Data *Tuberculosis*

Atribut umur merupkan atribut dengan data berbentuk numerik. Atribut kecamatan merupakan atribut dengan tipe data *string*. Atribut jenis kelamin, foto toraks, status HIV, Riwayat diabees, hasil TCM (Tes Cepat Molekuler) merupakan atribut berbentuk kategorikal. Berdasarkan Gambar 3.1 terdapat beberapa data yang kosong atau dikenal sebagai *missing value.* Terdapat 741 data *record* yang memiliki nilai *missing value* pada salah satu atau beberapa atributnya. Berdasarkan jumlah data *record* sebesar 985 dan 6 buah atribut, maka dataset ini memiliki *field* yang berjumlah 5.920, dengan *field* yang mengandung *missing value* berjumlah 741. Berikut adalah rincian dari jumlah *field* yang memuat *missing value* pada setiap atribut nya.

- Atribut umur = 0 data
- Atribut jenis kelamin = 0 data
- Atribut foto toraks = 28 data
- Atribut status HIV = 261 data
- Riwayat Diabetes = 7 data
- Hasil TCM = 445 data

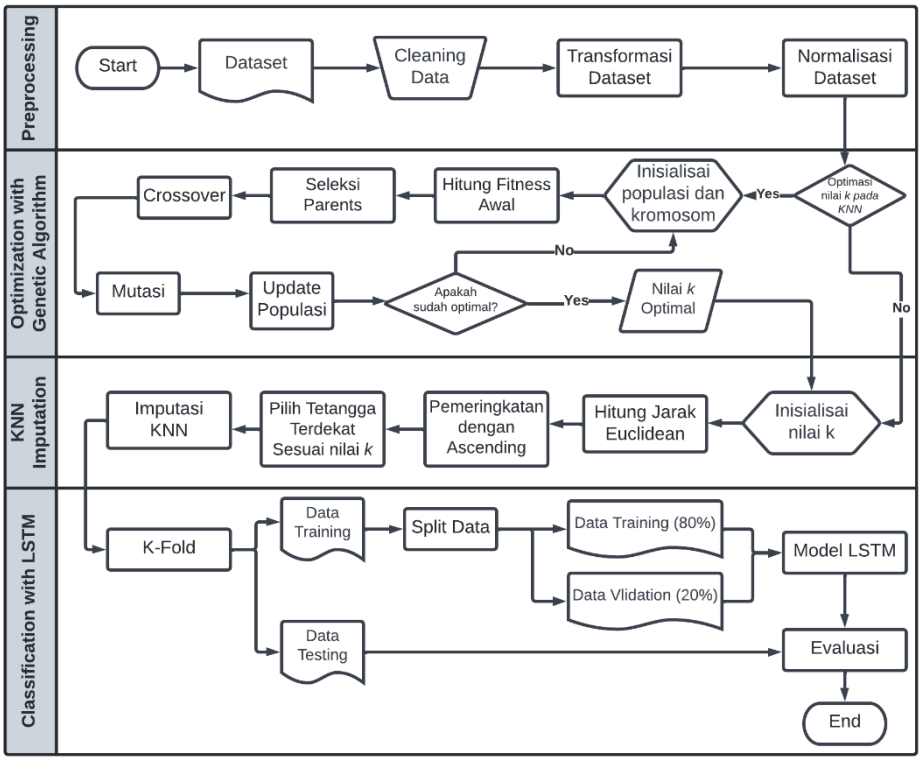
Berikut adalah rincian atribut berbentuk kategorikal yang digunakan pada penelitian ini, meliputi atribut jenis kelamin, foto toraks, status HIV, riwayat diabetes, hasil TCM, dan lokasi anatomi yang ditampilkan pada Gambar 3.2.



Gambar 3.2 Atribut Kategorikal pada Data *Tuberculosis*

3.2 Arsitektur Sistem

Sistem yang dibangun untuk mendukung penelitian ini memiliki 4 buah tahapan. Pertama, sistem melalui tahap *prepare* data terlebih dahulu. Setelah itu, terdapat tahap imputasi *missing value* menggunakan *k*-NN. Pada penelitian ini, penggunaan nilai *k* dalam *k*-NN dapat melalui proses optimasi terlebih dahulu menggunakan *Genetic Algorithm* (GA). Setelah data menjadi penuh atau sudah tidak ada lagi data kosong (*missing value*), kemudian dilanjutkan ke tahap klasifikasi dengan menggunakan algoritma *Long Short-Term Memory* (LSTM). Data diklasifikasikan berdasarkan letak anatomi nya, sehingga dapat diklasifikasikan kedalam *class* TB Paru atau TB Ekstra Paru. Arsitektur sistem dan algoritma dari metode usulan yang digunakan dalam penelitian ini yang diperlihatkan pada Gambar 3.3.



Gambar 3.3 Diagram Alur Arsitekstur Sistem

Berdasarkan alur arsitektur sistem yang ditampilkan pada Gambar 3.3 diatas, penelitian ini memiliki beberapa proses dalam tiap tahapannya. Beberapa proses tersebut dijelaskan pada urutan sub bab di bawah ini.

3.3 *Preproccessing*

Dalam tahap *preprocessing*, data yang digunakan telah dijelaskan pada sub Bab 3.1 sebelumnya. Setelah melalui proses *preprocessing data*, maka data dapat diproses oleh sistem.

3.3.1 Cleaning Data

Pada penelitian ini, cleaning data digunakan untuk menghapus *feature* atau atribut yang tidak digunakan, antara lain atribut kecamatan dan hasil TCM. Atribut kecamatan tidak diperlukan dalam pengolahan data pada penelitian ini karena hanya berisi data kecamatan dari pasien saja dan tidak berhubungan dengan penyakit *tuberculosis*. Atribut hasil TCM juga tidak digunakan karena hampir 50% data rekamannya tidak tersedia (missing value), sehingga dapat mempengaruhi hasil imputasi nantinya [26]. Oleh sebab itu, atribut atau *feature* yang digunakan dalam pengolahan data pada penelitian ini meliputi umur, jenis

kelamin, foto toraks, status HIV, dan riwayat diabetes. Kemudian untuk *output* atau target nya adalah paru dan ekstra paru.

3.3.2 Transformasi Data

Berdasarkan Gambar 3.1 yang telah dijelaskan pada sub bab sebelumnya, transformasi data dilakukan pada data kategori berbentuk *string*. Proses transformasi data tersebut bertujuan untuk mengubah data kategorikal yang berbentuk *string* tersebut menjadi data berbentuk *integer*, sehingga data dapat diproses atau dianalisa oleh sistem. Pada penelitian ini, atribut yang dilakukan proses transformasi data adalah atribut jenis kelamin, foto toraks, status HIV, dan riwayat diabetes, serta output yaitu target lokasi anatomi. Berikut adalah ilustrasi transformasi data pada tiap data atribut dalam penelitia ini yang ditampilkan pada Tabel 3.1, Tabel 3.2, Tabel 3.3, Tabel 3.4, Tabel 3.5, dan Tabel 3.6.

Tabel 3.1 Transformasi Data Atribut Umur.

Umur	***Integer***
≤ 5	0
6 – 11	1
12 – 16	2
17 – 25	3
26 – 35	4
36 – 45	5
46 – 55	6
56 – 65	7
66 ≥	8

Berdasarkan Tabel 3.1, penggolongan umur didasari dari peraturan Kementrian Kesehatan Indonesia, 2009. 0 – 5 tahun, 6 – 11 tahun, 12 – 16 tahun, 17 – 25 tahun, 26 – 35 tahun, 36 – 45 tahun, 46 – 55 tahun, 56 – 65 tahun, lebih dari 66 tahun [27].

Tabel 3.2 Transformasi Data Atribut Jenis Kelamin.

Jenis Kelamin	***Integer***
L	0
P	1

Pada Tabel 3.2 diatas, transformasi data dilakukan terhadap atribut jenis kelamin. Didalam atribut jenis kelamin terdapat dua *value. Value* tersebut adalah bernilai jenis kelamin perempuan dan jenis kelamin laki – laki. Jenis kelamin laki - laki dinotasikan dengan nilai 0 dan jenis kelamin perempuan dinotasikan dengan

nilai 1. Pemberian notasi tersebut didasarkan oleh penelitian yang telah dilakukan oleh Eriani Sahara, 2020. Berdasarkan hasil penelitian yang telah dilakukan tersebut, penderita TB terbesar adalah laki-laki sebesar 70 %. Oleh karena itu, pemberian notasi jenis kelamin dengan nilai 1 (nilai paling besar) diberikan kepada laki – laki.

Tabel 3.3 Transformasi Data Atribut Foto Toraks.

Foto Toraks	***Integer***
Negatif	0
Positif	1

Pada Tabel 3.3 diatas, transformasi data dilakukan terhadap atribut foto toraks. Didalam atribut foto toraks terdapat dua *value*. *Value* tersebut adalah bernilai positif dan negatif. Pemberian notasi mengacu pada nilai *True False*, dengan *True* bernilai 1 dan *False* bernilai 0. Berdasarkan hal tersebut, nilai foto toraks positif dinotasikan dengan nilai 1 dan negatif dinotasikan dengan nilai 0.

Tabel 3.4 Transformasi Data Atribut Status HIV.

Status HIV	***Integer***
Negatif	0
Positif	1

Pada Tabel 3.4 diatas, transformasi data dilakukan terhadap atribut status HIV. Didalam atribut jenis kelamin terdapat dua *value. Value* tersebut adalah bernilai positif dan negatif. Pemberian notasi mengacu pada nilai *True False*, dengan *True* bernilai 1 dan *False* bernilai 0. Berdasarkan hal tersebut, status HIV positif dinotasikan dengan nilai 1 dan negatif dinotasikan dengan nilai 0.

Tabel 3.5 Transformasi Data Atribut Riwayat Diabetes.

Riwayat Diabetes	***Integer***
Tidak	0
Ya	1

Pada Tabel 3.5 diatas, transformasi data dilakukan terhadap atribut riwayat diabetes. Didalam atribut jenis kelamin terdapat dua *value. Value* tersebut adalah bernilai iya dan tidak. Pemberian notasi mengacu pada nilai *True False*, dengan *True* bernilai 1 dan *False* bernilai 0. Berdasarkan hal tersebut, riwayat diabetes dengan nilai iya dinotasikan dengan nilai 1 dan tidak dinotasikan dengan nilai 0.

Tabel 3.6 Transformasi Data Lokasi Anatomi.

Lokasi Anatomi	***Integer***
Ekstra Paru	0

Lokasi Anatomi	*Integer*
Paru	1

Pada Tabel 3.6 diatas, transformasi data dilakukan terhadap target atau *output* lokasi anatomi. Didalam target lokasi anatomi terdapat dua *class*. Berdasarkan hal tersebut, maka bentuk pengklasifikasian yang dilakukan adalah *binary classification*. *Class* tersebut adalah bernilai ekstra paru dan paru. Lokasi anatomi dengan nilai ekstra paru dinotasikan dengan nilai 0 dan paru dinotasikan dengan nilai 1. Berikut adalah ilustrasi dari *dataset* setelah dilakukan normalisasi dan transformasi *real dataset* yang ditampilkan pada Gambar 3.3.

UMUR	JENIS_KELAMIN	FOTO_TORAKS	STATUS_HIV	RIWAYAT_DIABETES	LOKASI_ANATOMI
6	1	1	0	0	1
5	0	0	0	0	0
7	1	1	0	0	1
6	1	1	0	1	1
2	1	0	0		0
6	1		0	0	1
6	0	1	0	0	1
7	1	1	0	0	1
6	0	1	0	0	1

Gambar 3.4 Transformasi Data.

3.3.3 Normalisasi Data

Data yang bernilai integer, kemudian dilakukan normalisasi dengan menggunakan *Standard Scalling*. *Standard Scalling* dilakukan agar data memiliki distribusi yang sama. Proses *Standard Scaling* adalah dengan menghitung rata-rata (*mean*) dan standar deviasi (*standard deviation*) dari setiap fitur, kemudian mengurangi rata-rata dan membagi dengan standar deviasi dari setiap fitur. Berikut adalah contoh gambaran dari proses normalisasi data dengan *Standard Scalling* pada data UMUR, dengan nilai datanya setelah dilakukan transform adalah 6, 5, 7, 6, 2, 6, 6, 7, 6.

Mean = 5,666667

Standar deviasi :

$$\sigma = \sqrt{\frac{(6-5{,}666667)^2+(5-5{,}666667)^2+(7-5{,}666667)^2+\cdots((6-5{,}666667)^2}{9}}$$

$$\sigma = 1{,}414213562$$

Standard scalling :

Pada data umur bernilai 6, maka nilai *Standard Scalling* nya adalah

$x^* = \frac{5 - 5{,}666667}{1{,}414213562} = 0{,}23570226$

Perhitungan normalisasi *standard scaller* dilakukan pada seluruh atribut yang digunakan. Berikut adalah hasil normalisasi data pada tiap data atribut dalam penelitian ini yang ditampilkan pada Tabel 3.7.

Tabel 3.7 Normalisasi *Standard Scaller*

Atribut	Integer	Numerik
Umur	0	1,419981
	1	-2,153292
	2	-1,642824
	3	-1,132357
	4	-0,621890
	5	-0,111422
	6	0,399046
	7	0,909513
	8	1,419981
Jenis Kelamin	0	-1,159844
	1	0,862185
Foto Toraks	0	-1,704735
	1	0,586601
Status HIV	0	-0,193000
	1	5,181327
Riwayat Diabetes	0	-0,222163
	1	4,501208

Hasil normaliasasi yang didapatkan dari *Standard Scalling*, digunakan untuk merepresentasikan label dari hasil proses transformasi ke dalam bentuk numerik. Ilustrasi dari normalisasi *real dataset* ditampilkan pada Gambar 3.5.

UMUR	JENIS_KELAMIN	FOTO_TORAKS	STATUS_HIV	RIWAYAT_DIABETES	LOKASI_ANATOMI
3,990457	-1,159843996	5,866013328	-1,930007349	4,501207567	1
1,419981	-1,159843996	5,866013328	-1,930007349	-2,221626053	1
3,990457	-1,159843996	5,866013328	-1,930007349	-2,221626053	1
3,990457	-1,159843996	-1,704735302	5,181327421	-2,221626053	0
3,990457	-1,159843996	-1,704735302	-1,930007349	-2,221626053	0
9,095132	-1,159843996	5,866013328	-1,930007349	-2,221626053	1
-6,21889	-1,159843996	5,866013328	-1,930007349	-2,221626053	1
3,990457	-1,159843996	5,866013328	5,181327421	4,501207567	1
-6,21889	8,621849174		-1,930007349	-2,221626053	1
-1,13236	8,621849174		-1,930007349		1
3,990457	8,621849174	5,866013328	-1,930007349	-2,221626053	1

Gambar 3.5 Normalisasi Data.

3.4 *Genetic Algorithm* (GA)

Dalam penelitian ini, *Genetic Algorithm* (GA) digunakan untuk melakukan pencarian nilai *k* terbaik PADA *k*-NN. D*atatoy* yang digunakan dalam perhitungan GA ditampilkan pada Tabel 3.13, Lampiran 5. Algorima GA yang digunakan dalam penelitian ini, dapat ditampilkan pada Algoritma 3.1.

a. ***Input***
Dataset yang akan dilakukan imputasi

b. ***Output***
Nilai *neighbor* terbaik (*k* terbaik)

c. **Inisialisasi**
Populasi, Kromosom

d. **Parameter**
Crossover rate (Cr), *Mutation rate* (*Mr*), *Population Size* (*Pop Size*), Maksimum Generasi (*Max Generation*)

e. **Proses**

- Menghitung nilai *fitness* awal.
- Memilih *parents* (seleksi *parents*).
- Mengawinkan kromosom *parents* (*crossover*).
- Melakukan mutasi terhadap hasil *crossover* (*offspring*).
- Menghitung nilai *fitness offspring*.
- Melakukan *update* populasi dan mengulangi proses dari awal hingga batas maksimum generasi atau solusi yang didapatkan stabil.

Algoritma 3.1 *Genetic Algorithm (* GA *)*

3.4.1 Inisialisasi Populasi dan Kromosom

Tahap pertama dalam melakukan perhitungan GA adalah dengan membangkitkan atau melakukan inisialisasi populasi dan kromosom. Populasi pada GA diibaratkan sebagai kemungkinan solusi. Sedangkan kromosom adalah individu dari populasi. Misal, populasi yang digunakan bernilai 6. Maka secara

random dibangkitkan kromosom atau kemungkinan solusi berupa nilai bilangan bulat *k* sebanyak 6 buah, dengan ketentuan, *k* < data yg digunakan. Sebagai contoh, populasi yang telah dibangkitkan adalah [1, 2, 3, 4, 5, 6].

3.4.2 Menghitung Nilai *Fitness*

Nilai *fitness* dihitung dengan menggunakan rata – rata dari metrik evaluasi yaitu *mean squared error* (MSE) dari hasil klasifikasi *k*-NN menggunakan *library python* dengan pembagian data menggunakan *KFold* dengan nilai *K*=5. Penggunakan nilai *KFold* tersebut didasari karena, *KFold* dengan nilai *K*=5 pada *k*-NN dapat menghasilkan kinerja komputasi yang lebih baik jika dibandingkan dengan nilai *KFold* lainnya [22]. Sebagai contoh, nilai *fitness* dari populasi adalah sebagai berikut.

Populasi = [1, 2, 3, 4, 5, 6]

Fitness scores = [0.2, 0.3, 0.1, 0.1, 0.2, 0.1]

3.4.3 Seleksi *Parents*

Pada penelitian ini, proses seleksi *parents* dilakukan dengan menggunakan *roulette wheel selection*. *Roulette wheel selection* dipilih karena proses seleksi menggunakan probabilitas nilai fitness sehingga nilai fitness yang besar memiliki kesempatan untuk terpilih lebih besar dibandingkan dengan nilai fitness yang memiliki nilai lebih kecil.

a. Perhitungan Probabilitas

Dalam seleksi parents, Langkah pertama adalah menghitung probabilitas *fitness* setiap kromosom dengan cara membagi nilai *fitness* tiap kromosom dengan total nilai *fitness* dalam satu generasi dengan menggunakan persamaan sebagai berikut.

$$Prob_i = \frac{f_i}{(f_1+f_2+f_3+\cdots+f_n)} \quad (3.2)$$

Keterangan :

$Prob_i$ = Probabilitas dari kromosom ke-i

f_i = Nilai *fitness* dari kromosom ke-i

b. Probabilitas Kumulatif

Setelah menentukan nilai probabilitas tiap kromosom, selanjutnya menghitung probabilitas kumulatif. Nilai probabilitas kumulatif didapat

dengan menambahkan nilai probabilitas dari individu sebelumnya dengan nilai probabilitas dari individu yang sedang diproses.

c. Menantukan Daerah Kromosom

Setelah mendapatkan probabilitas kumulatif tiap kromosom, kemudian dilanjutkan dengan menentukan kromosom atau individu mana yang terpilih dengan menggunakan nilai random. Jika nilai random tersebut kurang dari nilai probabilitas kumultaif dari kromosom atau individu tersebut, maka individu tersebut dipilih menjadi *parent*. Namun, jika individu tersebut sama dengan individu sebelumnya, maka individu tersebut dilewati dan proses iterasi dilanjutkan ke individu selanjutnya.

Berikut adalah contoh gambaran dari proses seleksi *parent* pada penelitian ini.

- Populasi = [1, 2, 3, 4, 5, 6]
- *Fitness scores* = [0.2, 0.3, 0.1, 0.1, 0.2, 0.1]
- Total *fitness* = 1
- Probabilitas = [0.2, 0.3, 0.1, 0.1, 0.2, 0.1]
- Probabilitas kumulatif = [0.2, 0.5, 0.6, 0.7, 0.9, 1]
- Ketika dilakukan *random* dan menghasilkan nilai 0.25, maka individu dengan *index* 1 (2) dipilih dan ditambahkan ke dalam daftar parent, karena nilai 0.25 berada di antara probabilitas kumulatif 0.2 dan 0.5.
- Ketika dilakukan *random* dan menghasilkan 0.65, maka individu dengan *index* 2 (3) dipilih dan ditambahkan ke dalam daftar parent, karena nilai 0.65 berada di antara probabilitas kumulatif 0.6 dan 0.7.
- Ketika dilakukan *random* dan menghasilkan 0.8, maka individu dengan *index* 3 (4) dipilih dan ditambahkan ke dalam daftar parent, karena nilai 0.8 berada di antara probabilitas kumulatif 0.7 dan 0.9. Kemudian dilanjutkan hingga individu terakhir.
- Sehingga *parent* yang dihasilkan adalah [2, 3, 4, dan seterusnya].

3.4.4 Crossover

Setelah mendapatkan *parent,* kemudian dilanjutkan ke tahap *crossover*. Kedua nilai binner dari *parent*s tersebut, kemudian dilakukan proses *crossover two point*. *Crossover two point* dilakukan dengan mengawinkan pasangan *parent*

tersebut berdasarkan dua titik *crossover* yang didapatkan secara random agar mendapatkan anak atau individu baru yang disebut dengan *offspring*. Pada proses ini terdapat *crossover rate* (cr). Nilai *crossover rate* (cr) didapatkan dari pengujian yang dilakukan berdasarkan skenario 2. Berikut adalah contoh gambaran dari proses *crossover*.

parent1 = [0, 1, 0, 0, 1, 1]

parent2 = [1, 1, 0, 0, 0, 1]

crossover1 (titik *crossover random*) = 2

crossover2 (titik *crossover random*) = 4

$$Child\ 1 = \text{parent1}[:\text{crossover1}] + \text{parent2}[\text{crossover1}:\text{crossover2}] + \text{parent1}[\text{crossover2}:]$$

$$Child\ 1 = [0,1] + [0,0] + [1,1] = [0,1,0,0,1,1]$$

$$Child\ 2 = \text{parent2}[:\text{crossover1}] + \text{parent1}[\text{crossover1}:\text{crossover2}] + \text{parent2}[\text{crossover2}:]$$

$$Child\ 2 = [1,1] + [0,0] + [0,1] = [1,1,0,0,0,1]$$

$$Offspring = [Child\ 1], [Child\ 2]$$

3.4.5 Mutasi

Proses selanjutnya adalah mutasi. Pada penelitian ini mutasi yang digunakan adalah mengganti nilai biner. Pada proses ini terdapat *mutation rate* (mr). Nilai *mutation rate* (mr) didapatkan dari pengujian yang dilakukan berdasarkan skenario 3. Setiap elemen atau gen dari *offspring* dibangkitkan nilai random. Jika nilai random yang dihasilkan lebih kecil dari *mutation rate* (mr), maka elemen atau gen tersebut di-*flip* dengan cara mengganti nilainya menjadi 1 dikurangi nilai elemen tersebut. Jika nilai elemen atau gen saat ini adalah 0, maka setelah dilakukan *flip* nilainya berganti menjadi 1. Sebaliknya, jika nilai elemen atau gen saat ini adalah 1, maka setelah dilakukan flip nilainya berganti menjadi 0. Berikut adalah contoh gambaran dari proses mutasi.

- Jika offspring saat ini adalah [[0, 1, 0, 0, 1, 1], [1, 1, 0, 0, 0, 1]] dan contoh *mutation rate* = 0.3
- Kemudian, melakukan pemeriksaan nilai *random* untuk setiap elemen dari setiap *offspring*. Jika nilai random yang dihasilkan lebih kecil *dari mutation*

rate (0.3) maka elemen tersebut di-*flip* atau diganti dengan nilai yang berlawanan.

- Misalnya, elemen pertama dari offspring pertama (0) di-flip menjadi 1, elemen ketiga dari offspring kedua (0) di-flip menjadi 1, maka offspring setelah dilakukan mutasi adalah [[1, 1, 0, 0, 1, 1], [1, 1, 1, 0, 0, 1]]

3.4.6 *Update* Populasi

Proses selanjutnya yaitu dengan menggabungkan populasi awal dengan *offspring* baru. Setelah itu, dilakukan perankingan agar mendapatkan populasi baru berdasarkan nilai *fitness* terbaik dengan menghitung kembali nilai *fitness* tiap individu atau kromosom. Sebagai contoh, populasi baru berupa kemungkinan solusi berupa nilai bilangan bulat *k* yang didapatkan adalah [3, 9, 8, 5, 11, 7]. Maka, populasi baru tersebut digunakan pada generasi berikutnya dan dilakukan secara berulang-ulang hingga mencapai batas maksimum generasi. Setelah mencapai batas maksimum generasi atau solusi yang didapatkan stabil (nilai k tidak berubah). Sebagai contoh best individu yang didapatkan adalah 3, maka 3 digunakan sebagai nilai *k* pada saat imputasi *k*-NN.

3.5 *k*-Nearest Neighbor (*k*-NN)

Dalam penelitian ini, Algoritma *k*-NN merupakan algoritma yang digunakan untuk melakukan imputasi *missing value*. Dalam melakukan imputasi *missing value* menggunakan metode *k*-NN, dibutuhkan sejumlah data dengan atribut yang diperlukan adalah berupa atribut umur, jenis kelamin, foto toraks, status HIV, dan riwayat diabetes. Algorima *k*-NN yang digunakan dalam penelitian ini, dapat ditampilkan pada Algoritma 3.2.

a. ***Input***
Dataset yang dilakukan proses imputasi.

b. ***Output***
Nilai imputasi atau nilai pengganti

c. **Inisialisasi**
Nilai *neighbors* (*k*)

d. **Proses**
- Menghitung jarak antar data dengan menggunakan perhitungan *euclidean distance*.
- Mengurutkan jarak
- Menentukan k observasi

- Melakukan imputasi menggunakan teknik *most frequent*.

Algoritma 3.2 *K-Nearest Neighbor*

3.5.1 Inisialisasi nilai *k*

Nilai *k* digunakan untuk menentukan jumlah tetangga terdekat dalam menentukan jumlah observasi dari pengurutan jarak *Euclidean* dari tiap data pada atribut yang mengandung *missing value* dengan ketentuan, nilai *k* < data yg digunakan.

3.5.2 Perhitungan Jarak *Euclidean*

Perhitungan jarak *Euclidean* dilakukan terhadap atribut yang memiliki *missing value.* Perhitungan jarak *Euclidean* dilakukan dengan menghitung perbedaan kuadrat setiap elemen pada setiap kolom (kecuali yang *memiliki missing value*) dari baris yang memiliki *missing value* dengan baris lainnya, lalu hasilnya dijumlahkan. Kemudian, hasil jumlah tersebut diakarkan untuk menghitung *jarak Euclidean* antara kedua baris tersebut. Sebagai contoh pada Gambar 3.5, data yang hilang adalah data pada baris ke-9 dan kolom ke-3 (FOTO_TORAKS). Berikut adalah contoh perhitungan jarak *euclidean* nya.

Jarak antara data hilang dengan data pada baris ke-1 :

$$d_{euc(9,1)} = \sqrt{(((-6{,}21889) - 3{,}990457)^2 + (8{,}621849174 - (-1{,}159843996))^2 + (8{,}621849174 - (-1{,}159843996))^2) + ((-1{,}930007349) - (-1{,}930007349))^2 + ((-2{,}221626053) - 4{,}501207567)^2}$$

$$d_{euc(9,1)} = 15{,}65595028$$

Jarak antara data hilang dengan data pada baris ke-2 :

$$d_{euc(9,2)} = \sqrt{(((-6{,}21889) - 1{,}419981)^2 + (8{,}621849174 - (-1{,}159843996))^2 + (8{,}621849174 - (5{,}866013328))^2) + ((-1{,}930007349) - (-1{,}930007349))^2 + ((-2{,}221626053) - (-2{,}221626053))^2}$$

$$d_{euc(9,2)} = 12{,}41103829$$

Kemudian dilanjutkan hingga data terakhir. Sehingga jarak yang dihasilkan pada baris ke-9 dengan baris ke-1 hingga terakhir, ditampilkan pada Tabel 3.8.

Tabel 3.8 Contoh Perhitungan *Euclidean Distance*

Baris	Baris										
	1	2	3	4	5	6	7	8	9	10	11

9	15,7	12,4	14,1	15,8	14,1	18,2	9,8	17,2	0	5,6	10,1

3.5.3 Pengurutan Jarak dan Menentukan *k* observasi terdekat

Setelah melakukan perhitungan jarak *Euclidean* antar data, kemudian dilanjutkan dengan melakukan pengurutan nilai jarak tersebut secara *ascending*. Kemudian dilanjutkan dengan mengambil observasi yang memiliki jarak terdekat sebanyak *k*, sebagai contoh nilai k=5. Berdasarkan contoh perhitungan dari Tabel 3.7, pengurutan jarak terdekat (terkecil) berdasarkan *k* observasi adalah data dengan baris 7, 11, 2, 3, 5.

3.5.4 Imputasi *k* -NN

Setelah mengurutkan nilai jarak *Euclidean* dan menentukan observasi terdekat, kemudian nilai modus dari observasi tersebut digunakan untuk mengimputasikan nilai yang hilang pada observasi yang dimaksud dengan menggunakan teknik *most frequent* dari nilai label milik baris tersebut. Berdasarkan contoh perhitungan pada Tabel 3.1, berikut adalah imputasi *k*-NN nya.

Baris 7 pada kolom 3 (FOTO_TORAKS), bernilai label 1
Baris 11 pada kolom 3 (FOTO_TORAKS), bernilai label 1
Baris 2 pada kolom 3 (FOTO_TORAKS), bernilai label 0
Baris 3 pada kolom 3 (FOTO_TORAKS), bernilai label 1
Baris 5 pada kolom 3 (FOTO_TORAKS), bernilai label 1

Berdasarkan beberapa nilai label tersebut, 1 (satu) adalah nilai label yang sering muncul. Maka, data yang hilang pada baris ke-9 dan kolom ke-3 (FOTO_TORAKS) digantikan dengan nilai label 1.

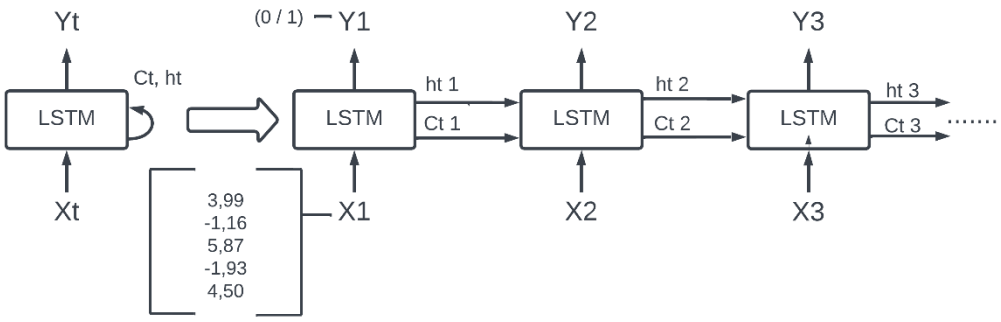
3.6 Pembagian Data

Sebelum melakukan pemodelan untuk melakukan klasifikasi lokasi anatomi, hal yang dilakukan terlebih dahulu adalah melakukan pembagian data menggunakan K-*Fold* untuk menghasilkan *data training* dan *data testing*. K-*Fold* pada penelitian ini menggunakan K bernilai 10. Berdasarkan artikel dari BINUS University, yang ditulis oleh Antoni Wibowo, 2017, nilai 10 fold direkomendasikan untuk pemilihan model terbaik [28]. Selanjutnya, data validasi

didapatkan melalui split data pada *data training*. Data validasi digunakan untuk mengevaluasi kinerja model mesin yang telah dilatih. Berdasarkan penelitian yang dilakukan oleh Ilham Farros, dkk, perbandingan 80:20 dapat merupakan perbandingan yang terbaik dalam melakukan split data [29]. Oleh karena itu, pada penelitian ini menggunakan perbandingan tersebut untuk membagi lagi data training dari hasil KFold menjadi 80% data training dan 20% data validasi.

3.7 *Long Short-Term Memory*

LSTM merupakan suatu algoritma yang memiliki empat buah *layer*. Keempat gerbang tersebut antara lain adalah *Forget Gate*, *Input Gate*, *cellstate*, dan *Output Gate*. Pada penelitian ini, LSTM digunakan dalam pengklasifikasian letak atau jenis *tuberculosis*, sehingga menghasilkan dua buah *class*. *Class* yang dihasilkan dari pengklasifikasian tersebut yaitu, *class* TB paru dan TB ekstra paru. Inputan yang digunakan dalam algoritma ini adalah nilai atribut dari tiap data atau individu. Berikut adalah proses LSTM mulai dari inputannya yang digambarkan pada Gambar 3.6.



Gambar 3.6 Alur *Long Short-Term Memory*

Berdasarkan Gambar 3.6, proses LSTM menghasilkan output yang diwakilkan dengan variabel Y. Output tersebut adalah hasil klasifikasi yang berupa nilai biner (bernilai 0 atau 1) atau yang biasa disebut dengan *binary classification*. Secara umum langkah-langkah membangun sistem yang dibuat terdiri dari inisialisasi parameter, training LSTM Network, dan melakukan uji terhadap *data testing* [23]. Algorima LSTM dari proses training yang digunakan dalam penelitian ini, dapat ditampilkan pada Algoritma 3.3.

a.	***Input***
	Dataset (*data training*)

b. ***Output***
Model

c. **Parameter**
Learning rate, *hidden layer*, jumlah *neuron* pada *hidden layer*, maksimum *epoch*

d. **Proses**

- Menghitung semua fungsi *gates unit* pada setiap *neurons*.
- Menghitung fungsi aktivasi *sigmoid* pada *output layer*.
- Melakukan perulangan sebanyak *epoch* serta melakuan pembaruan bias dan pembaruan bobot menggunakan *adam optimizer*.

Algoritma 3.3 *Training* LSTM

Kemudian, model yang telah didapatkan pada proses training, diuji dengan menggunakan *data testing*, dengan metode akurasi yang digunakan menggunakan persamaan akurasi, presisi, dan *recall*.

3.8 Metode Evaluasi

Evaluasi merupakan tahap akhir setelah sistem dibuat dan telah dilakukan uji coba terhadap sistem tersebut. Evaluasi bertujuan untuk menguji kinerja dari suatu algoritma pengenalan yang digunakan dengan menampilkan tingkat akurasi pada setiap kategorinya [30]. Akurasi merupakan sebuah proses untuk mengukur tingkat keakuratan suatu proses pengujian dengan cara mengidentifikasi jumlah dari data yang diklasifikasi secara benar. *Confusion matrix* digunakan dalam metode evaluasi ini. *Confusion matrix* merupakan tabel yang menunjukkan pengukuran hasil kerja pengenalan yang menghasilkan nilai akurasi. Berikut adalah tabel confusion matrix pada penelitian ini yang ditampilkan pada Tabel 3.9.

Tabel 3.9 *Confusion Matrix*.

		Data Asli	
		Paru	Ekstra Paru
Data Prediksi	Paru	TP	FP
	Ekstra Paru	FN	TN

Keterangan :

TP = Data prediksi berkategori paru dan terprediksi benar oleh sistem

TN = Data prediksi berkategori ekstra paru dan terprediksi benar oleh sistem

FP = Data prediksi berkategori paru namun terprediksi salah oleh sistem, dengan data sebenarnya adalah kategori ekstra paru

FN = Data prediksi berkategori ekstra paru namun terprediksi salah oleh sistem, dengan data sebenarnya adalah kategori paru.

Dalam melakukan pengukuran performa sistem dapat menggunakan perhitungan akurasi, presisi, dan *recall.* Akurasi merupakan sebuah proses untuk mengukur tingkat keakuratan suatu proses pengujian dengan cara mengidentifikasi jumlah dari klasifikasi data berupa benar. Presisi merupakan persentase hasil dari jumlah data positif. Pada penelitian ini, data positif adalah data dengan kategori paru, dimana data positif tersebut terklasifikasikan secara benar dan kemudian dibagi dengan total data yang ju*Gate*rklasifikasi positif. Sedangkan *recall* merupakan persentase hasil dari data berkategori nilai positif. Pada penelitian ini, data positif adalah data dengan kategori paru yang telah terklasifikasikan dengan nilai benar oleh sistem yang kemudian dibandingkan bersama keseluruhan data dari kategori bernilai positif terklasifikasi dengan nilai benar. Berikut adalah persamaan dalam menghitung nilai akurasi, presisi, dan *recall*.

$$Akurasi = \frac{TP+TN}{TP+TN+FP+FN} \times 100\% \quad (3.3)$$

$$Presisi = \frac{TP}{FP+TP} \times 100\% \quad (3.4)$$

$$Recall = \frac{TP}{FN+TP} \times 100\% \quad (3.5)$$

Keterangan :

TP = Data prediksi berkategori *positive* dan terprediksi benar oleh sistem

TN = Data prediksi berkategori *negative* dan terprediksi benar oleh sistem

FP = Data prediksi berkategori *positive* namun terprediksi salah oleh sistem

FN = Data prediksi berkategori *negative* namun terprediksi salah oleh sistem

3.9 Skenario Pengujian

Skenario uji coba sistem dilakukan dengan beberapa tahap. Tahap pertama yaitu melakukan pencarian nilai *k* terbaik didapatkan dari solusi metode GA dengan mengterbaikkan parameter-parameter GA seperti *crossover rate*, *mutation rate*, *population size*, dan *maximum generation*. Tahap kedua, solusi nilai *k* terbaik dari GA digunakan untuk melakukan imputasi pada k-NN + GA.

Kemudian tahap ketiga, melakukan imputasi *missing value* menggunakan metode k-NN tanpa optimasi dan memvariasikan nilai k = 3, 5, 7, 9, 11, 13, dan 15. Terakhir, kinerja klasifikasi dengan menggunakan imputasi k-NN dan k-NN + GA dibandingkan berdasarkan akurasi. Berdasarkan penjelasan tersebut, berikut adalah skenario pengujian yang ditampilkan pada Tabel 3.10.

Tabel 3.10 Skenario Pengujian

No.	Pengujian	Objek Uji Coba	Skenario	Keterangan
1.	Perbandingan nilai *fitness* terkecil dan waktu komputasi paling singkat	Metode GA tanpa imputasi	Skenario 2	Mencari nilai *crossover rate* terbaik dengan varian nilai 0.1 hingga 1.0.
2.			Skenario 3	Mencari nilai *mutation rate* terbaik dengan varian nilai 0.01 hingga 0.1.
3.			Skenario 4	Mencari nilai *population size* terbaik dengan varian nilai 20 hingga 200
4.			Skenario 5	Mencari nilai maksimum generasi terbaik dengan varian nilai 200 hingga 2000
5.	Imputasi *k*NN dengan solusi nilai *k* terbaik	Metode k-NN + GA pada imputasi *missing value*	Skenario 6	Melakukan Imputasi *missing value* nilai *k* adalah *best individu* dari GA pada skenario sebelumnya
6.	Imputasi *k*-NN	Metode *k*-NN tanpa optimasi	Skenario 1	Melakukan Imputasi *missing value* dengan varian nilai *k* = 3, 5 ,7, 9, 11, 13, 15
7.	Perbandingan akurasi terbesar dan *loss* terkecil	Metode GA + kNN + LSTM dengan kNN + LSTM	Skenario 7	Mencari nilai *neuron* terbaik dengan varian nilai 6 hingga 30.
8.			Skenario 8	Mencari nilai *epoch* terbaik dengan varian

				nilai 25 hingga 200.

Setelah dilakukan pengujian dengan skenario seperti pada Tabel 3.9, dapat disimpulkan nilai terbaik dari setiap parameter GA, serta nilai terbaik untuk *neuron* dan *epoch* pada LSTM. Selain itu, juga dapat disimpulkan berhasil atau tidaknya pencarian nilai *k* terbaik pada *k*-NN menggunakan GA dalam meningkatkan akurasi pada klasifikasi LSTM, jika dibandingkan dengan klasifikasi LSTM menggunakan *k*-NN tanpa optimasi.

BAB IV HASIL DAN PEMBAHASAN

4.1 Lingkungan Ujicoba

Pada penelitian ini terdapat hardware dan software yang digunakan, dapat dilihat pada Tabel 4.1.

Tabel 4.1 Lingkungan Ujicoba

No.	Kebutuhan	Jenis Perangkat
1.	Sistem Operasi	*Windows* 10
2.	RAM	8 GB
3.	CPU	*Intel Core I3*
4.	*Tools*	*Visual Studio Code* *Jupyter Notebook*
5.	Bahasa Pemrograman	*Python*
6.	*Library*	*Numpy* *Pandas* *Math* *Random* *Tensorflow* *Keras* *Sklearn*
7.	*User Interface*	*PHP, HTML ,Bootstrap*

4.2 Pembuatan Sistem

Pada sub bab pembuatan sistem ini menjelaskan mengenai *source code* pada tahap *preprocessing* dan penerapan ketiga metode yang digunakan dalam penelitian ini. Metode pertama adalah k-NN untuk penanganan *missing value.* Metode kedua adalah *Genetic Algorithm* yang digunakan untuk melakukan optimasi dalam mencari nilai *k* terbaik untuk *k*-NN. Kemudian metode ketiga adalah *Long Short-Term Memory* yang digunakan untuk melakukan klasifikasi jenis *tuberculosis*, ketika *dataset* yang digunakan sudah melalui tahap imputasi dan sudah tidak mengandung *missing value*.

4.2.1 *Preprocessing*

Tahap *preprocessing* dilakukan sebelum *dataset* melalui proses imputasi. Pada tahap *preprocessing*, berisi proses transformasi data dan normalisasi data.

penjelasan mengenai proses transformasi data dan normalisasi data yang dijelaskan pada sub bab berikutnya.

4.2.1.1 *Cleaning Data*

Pemilihan *cleaning* kolom atribut atau *feature* yang dilakukan dengan manual. Kemudian, proses c*leaning data* diproses ketika membaca file dataset dengan memanggil semua atribut kecuali atribut yang tidak digunakan pada saat proses transformasi data. Atribut yang tidak digunakan dalam penilitian ini adalah atribut kecamatan dan hasil TCM. Pemanggilan kolom tersebut adalah sebagai berikut, *cols_to_transofrm=['UMUR', 'JENIS_KELAMIN', 'FOTO_TORAKS', 'STATUS_HIV','RIWAYAT_DIABETES','LOKASI_ANATOMI']*.

4.2.1.2 Transformasi Data

Proses transformasi data merupakan tahapan yang bertujuan untuk mengubah data kategorikal yang berbentuk *string* menjadi data berbentuk *integer*. Berikut adalah penjelasan dari pembuatan sistem transformasi data.

```
1. cols_to_transofrm=['UMUR','JENIS_KELAMIN','FOTO_TORAKS','STATUS_HIV','RIWAY
   AT_DIABETES','LOKASI_ANATOMI']
2. dataset=pd.read_csv('D:\SIDANG\dataset.csv',usecols=cols_to_transofrm,delim
   iter=";")
```

Kode Program 4.1 Membaca Dataset pada Transformasi Data

Berikut adalah penjelasan dari kode Program 4.1.

1) Baris 1 digunakan untuk mendefinisikan kolom yang digunakan, meliupti umur, jenis kelamin, foto toraks, status JIV, Riwayat diabetes, dan lokasi anatomi yang disimpan ke dalam variable cols_to_transform.
2) Baris 2 digunakan untuk membaca dataset dari *file csv* menggunakan *library pandas*. Dari dataset tersebut, kolom yang dipanggil adalah umur, jenis kelamin, foto toraks, status HIV, riwayat diabetes, dan lokasi anatomi, kemudian disimpan ke dalam variael *dataset*.

```
1. umur=dataset.UMUR.values
2. jenis_kelamin=dataset.JENIS_KELAMIN.values
3. foto_toraks=dataset.FOTO_TORAKS.values
4. status_hiv=dataset.STATUS_HIV.values
5. riwayat_diabetes=dataset.RIWAYAT_DIABETES.values
6. target=dataset.LOKASI_ANATOMI.values
```

Kode Program 4.2 Mengambil Data dari Dataset pada Transformasi Data

Berikut adalah penjelasan dari kode Program 4.2.

1. Baris 1 digunakan untuk memuat data atribut umur dari dataset dan menyimpannya dalam variabel umur.
2. Baris 2 digunakan untuk memuat data atribut jenis kelamin dari dataset dan menyimpannya dalam variabel jenis_kelamin.
3. Baris 3 digunakan untuk memuat data atribut foto toraks dari dataset dan menyimpannya dalam variabel foto_toraks.
4. Baris 4 digunakan untuk memuat data atribut status HIV dari dataset dan menyimpannya dalam variabel status_hiv.
5. Baris 5 digunakan untuk memuat data atribut riwayat diabetes dari dataset dan menyimpannya dalam variabel riwayat_diabetes.
6. Baris 6 digunakan untuk memuat data atribut lokasi anatomi dari dataset dan menyimpannya dalam variabel target.

```
1.  jumlahdata=len(dataset)
2.  datanew=[]
3.  for i in range(jumlahdata):
4.      labelumur= 0 if(umur[i]<=5) else 1 if(umur[i]>= 6 and umur[i]<=11) else
     2 if(umur[i]>= 12 and umur[i]<=16) else 3 if(umur[i]>= 17 and umur[i]<=25)
     else 4 if(umur[i]>= 26 and umur[i]<=35) else 5 if(umur[i]>= 36 and umur[i]
    <=45) else 6 if(umur[i]>= 46 and umur[i]<=55) else 7 if(umur[i]>= 56 and um
    ur[i]<=65) else 8 if(umur[i]>=66) else None
5.      labeljk= int(1) if(jenis_kelamin[i]=="L") else 0
6.      labelft= int(1) if(foto_toraks[i]=="Positif") else int(0) if(foto_torak
    s[i]=="Negatif") else None
7.      labelsh= int(1) if(status_hiv[i]=="Positif") else int(0) if(status_hiv[
    i]=="Negatif") else None
8.      labelrd= int(1) if(riwayat_diabetes[i]=="Ya") else int(0) if(riwayat_di
    abetes[i]=="Tidak") else None
9.      labeltarget= int(1) if(target[i]=="Paru") else int(0)
10.     datanew.append([labelumur,labeljk,labelft,labelsh,labelrd,labeltarget])
```

Kode Program 4.3 *Encoding* Kolom pada Transformasi Data

Berikut adalah penjelasan Kode Program 4.3

1. Baris 1 digunakan untuk menjumlah data dalam dataset dan menyimpannya dalam variabel jumlahdata.
2. Baris 2 digunakan untuk membuat sebuah list kosong dengan nama datanew.
3. Baris 3 merupakan perulangan untuk melakukan iterasi sebanyak jumlahdata kali.

4. Baris 4 hingga 9 digunakan untuk mengubah label tiap atribut sesuai dengan kondisinya..
5. Baris 10 digunakan untuk menambahkan data yang telah dibuat ke dalam list datanew dalam bentuk list dengan urutan label umur, jenis kelamin, foto toraks, status HIV, riwayat diabetes, dan target.

```
1. udtf= pd.DataFrame(datanew,columns=cols_to_transofrm)
2. print("TRANSFORMATION FROM CATEGORICAL DATAS :\n",udtf)
3. udtf.to_csv('D:\SIDANG\csv\preprocessing_transform.csv',index=False)
```

Kode Program 4.4 Menyimpan Hasil Transformasi Data

Berikut adalah penjelasan Kode Program 4.4

1. Baris 1 digunakan untuk membuat objek DataFrame baru udtf dari datanew dengan kolom-kolom yang didefinisikan dalam variabel cols_to_transform.
2. Baris 2 digunakan untuk mencetak teks "TRANSFORMATION FROM CATEGORICAL DATAS :" diikuti oleh isi dari udtf.
3. Baris 3 digunakan untuk menyimpan DataFrame udtf ke dalam file CSV dengan lokasi yang ditentukan dan dengan indeks yang tidak disertakan.

4.2.1.3 Normalisasi Data

Normalisasi data dilakukan dengan menggunakan normalisasi *standard scaller*. Tujuan dari normalisasi data menggunakan Standard Scaler adalah untuk mengubah nilai-nilai atribut pada suatu data menjadi nilai-nilai yang terstandarisasi, yaitu dengan membuat nilai-nilai tersebut memiliki mean yang sama dengan 0 (nol) dan memiliki standard deviation yang sama dengan 1 (satu. Berikut adalah penjelasan dari pembuatan sistem normalisasi data.

```
1. data_awal = pd.read_csv('D:\SIDANG\csv\preprocessing_transform.csv')
2. df = data_awal.drop(columns=['LOKASI_ANATOMI'])
```

Kode Program 4.5 Membaca data pada Normalisasi Data.

Berikut adalah penjelasan Kode Program 4.5

1. Baris 1 digunakan untuk membaca file CSV yang berisi data yang telah di-preprocessing dan menyimpannya dalam objek DataFrame data_awal.

2. Baris 2 digunakan untuk menghapus kolom 'LOKASI_ANATOMI' dari objek DataFrame data_awal dan menyimpan hasilnya dalam objek DataFrame df.

```
1.  # Menentukan standard scaller
2.  scaler = StandardScaler()
3.  # Menjalankan normalisasi
4.  df_normalized = scaler.fit_transform(df)
```

Kode Program 4.6 Melakukan Normalisasi Data.

Berikut adalah penjelasan Kode Program 4.6

1. Baris 2 digunakan untuk membuat objek standard scaler scaler.
2. Baris 4 digunakan untuk melakukan normalisasi data dalam objek DataFrame df menggunakan standard scaler scaler dan menyimpan hasilnya dalam objek DataFrame df_normalized.

```
1.  # Menambahkan kolom lokasi anatomi yang telah di drop sebelumnya
2.  df_normalized = pd.DataFrame(df_normalized, columns=df.columns)
3.  df_normalized['LOKASI_ANATOMI'] = data_awal['LOKASI_ANATOMI']
```

Kode Program 4.7 Menggabungkan Hasil Normalisasi dengan Data Target.

Berikut adalah penjelasan Kode Program 4.7

1. Baris 2 digunakan untuk membuat objek DataFrame baru df_normalized dari hasil normalisasi df dan mengatur nama kolom sesuai dengan kolom-kolom pada objek DataFrame df.
2. Baris 3 digunakan untuk menambahkan kolom 'LOKASI_ANATOMI' dari objek DataFrame data_awal ke objek DataFrame df_normalized.

```
1.  # Menyimpan data yang telah di normalisasi
2.  df_normalized.to_csv('D:\SIDANG\csv\preprocessing_scaled.csv',index=False)
```

Kode Program 4.8 Menyimpan Hasil Normalisasi.

Berdasarkan Kode Program 4.8, baris 2 digunakan untuk menyimpan dataframe yang telah di normalisasi ke dalam file csv dengan nama 'preprocessing_scaled.csv' dan mengatur index menjadi False (tidak menyertakan index pada file csv).

4.2.2 *Genetic Algorithm (GA)*

Pada sub bab ini menjelaskan mengenai proses dari metode *GA* yang digunakan untuk menentukan nilai *k* terbaik pada *k*-NN. Nilai parameter GA ditentukan dari proses pencarian nilai terbaik terhadap parameter GA berdasarkan skenario telah yang dilakukan.

4.2.2.1 Membaca Dataset

Dataset yang digunakan adalah data normalisasi yang tidak memiliki *missing value*. Dataset ini digunakan pada proses penentuan nilai fitness. Berikut adalah source code yang digunakan.

```
1.  # Read the CSV file into a NumPy array
2.  data = np.genfromtxt("D:/SIDANG/csv/dataset_full_scaled.csv", delimiter="," 
    ,usecols=[0,1,2,3,4],skip_header=1)
3.  target = np.genfromtxt("D:/SIDANG/csv/dataset_full_scaled.csv", delimiter="
    ,",usecols=5,skip_header=1)
4.  optimized_k = genetic_algorithm(data,target)
```

Kode Program 4.9 Membaca Dataset pada GA

Berikut adalah penjelasan Kode Program 4.9

1. Baris 2 digunakan untuk membaca file CSV 'dataset_full_scaled.csv' dan menyimpannya ke dalam NumPy array dengan membaca kolom 0 sampai 4 dari file CSV dan menyimpannya ke dalam variabel data.
2. Baris 3 digunakan untuk membaca kolom 5 dari file CSV menggunakan np.genfromtext dan menyimpannya ke dalam variabel target.
3. Baris 4 digunakan untuk memanggil fungsi genetic_algorithm dengan argumen data dan target, dan menyimpan hasilnya ke dalam variabel optimized_k.

4.2.2.2 Inisialiasasi Populasi dan Kromosom

Tahap pertama dalam melakukan perhitungan GA adalah dengan membangkitkan atau melakukan inisialisasi populasi dan kromosom atau individu awal sebagai dasar untuk proses algoritma genetika Dalam implementasinya, pembangkitan populasi awal dapat dilakukan dengan membuat fungsi khusus yang menerima ukuran populasi dan menghasilkan populasi individu dengan menggunakan metode yang ditentukan.. Berikut adalah source code yang digunakan.

```
1.  def generatePopulation(pop_size):
2.      population=[]
3.      while len(population) < pop_size:
```

```
4.          individual = random.choice([i for i in range(3, 225)])
5.          if individual not in population:
6.              population.append(individual)
7.      return population
```

Kode Program 4.10 Inisialisasi Populasi dan Kromosom pada GA

Berikut adalah penjelasan Kode Program 4.10

1. Baris 1 digunakan untuk membuat fungsi bernama generatePopulation dengan satu parameter pop_size.
2. Baris 2 digunakan untuk membuat list kosong dengan nama population.
3. Baris 3 digunakan untuk melakukan loop while dengan syarat panjang population harus kurang dari pop_size.
4. Baris 4 digunakan untuk memilih bilangan integer acak antara 3 dan 225 dan menyimpannya dalam variabel individual.
5. Baris 5 digunakan untuk melakukan pengecekan kondisi, jika individual belum ada dalam population, maka lanjut ke langkah selanjutnya.
6. Baris 6 digunakan untu menambahkan individual ke dalam population.
7. Baris 7 digunakan untuk mengembalikan nilai population dari fungsi generatePopulation.

4.2.2.3 Menentukan Nilai Fitness

Proses penentuan nilai fitness ini bertujuan untuk menentukan tingkat kecocokan setiap individu dalam populasi terhadap solusi yang dicari. Dalam implementasinya, penentuan nilai fitness dapat dilakukan dengan membuat fungsi khusus yang menerima individu dan menghitung nilai fitness dengan menggunakan rumus atau algoritma yang telah ditentukan. Berikut adalah source code yang digunakan.

```
1.  def fitness(individu,data,target):
2.      kf = KFold(n_splits=5, shuffle=False, random_state=None)
3.      mse_scores = []
4.      for train_index, test_index in kf.split(data):
5.          X_train, X_test = data[train_index], data[test_index]
6.          y_train, y_test = target[train_index], target[test_index]
7.          knn = KNeighborsRegressor(n_neighbors=individu)
8.          knn.fit(X_train, y_train)
9.          y_pred = knn.predict(X_test)
10.         mse = mean_squared_error(y_test, y_pred)
11.         mse_scores.append(mse)
12.     return(sum(mse_scores) / len(mse_scores))
```

Kode Program 4.11 Penentuan Nilai Fitness pada GA

Berikut adalah penjelasan Kode Program 4.11

1. Baris 1 digunakan untuk mendefinisikan fungsi 'fitness' yang memiliki 3 parameter masukan yaitu 'individu', 'data', dan 'target'.
2. Baris 2 digunakan untuk membuat objek 'kf' dengan menggunakan fungsi 'KFold' yang melakukan validasi silang dengan 5 lipatan.
3. Baris 3 digunakan untuk membuat list kosong 'mse_scores' yang menampung skor kesalahan kuadrat rata-rata.
4. Baris 4 digunakan untuk melakukan perulangan iterasi sebanyak 'n_splits' menggunakan objek 'kf' dan membagi 'data' menjadi data latih dan data uji.
5. Baris 5 digunakan untuk menetapkan 'X_train' dan 'X_test' sebagai data latih dan data uji untuk variabel 'data'.
6. Baris 6 digunakan untuk menetapkan 'y_train' dan 'y_test' sebagai data latih dan data uji untuk variabel 'target'.
7. Baris 7 digunakan untuk membuat objek 'knn' menggunakan fungsi 'KNeighborsRegressor' dengan parameter 'n_neighbors' yaitu 'individu'.
8. Baris 8 digunakan untuk melatih model menggunakan data latih 'X_train' dan 'y_train' menggunakan fungsi 'fit' pada 'knn'.
9. Baris 9 digunakan untuk melakukan prediksi nilai menggunakan data uji 'X_test' menggunakan fungsi 'predict' pada 'knn'.
10. Baris 10 digunakan untuk menghitung skor kesalahan kuadrat rata-rata (mse) antara 'y_test' dan 'y_pred' menggunakan fungsi 'mean_squared_error'.
11. Baris 11 digunakan untuk menambahkan skor kesalahan kuadrat rata-rata (mse) ke dalam list 'mse_scores'.
12. Baris 12 digunakan untuk mengembalikan rata-rata dari seluruh skor kesalahan kuadrat rata-rata (mse) yang disimpan di dalam list 'mse_scores'.

4.2.2.4 Seleksi Parent

Proses seleksi *parent* ini bertujuan untuk memilih parent individu yang digunakan dalam proses reproduksi. Pada penelitian ini, proses seleksi parent dilakukan dengan metode atau teknik *Roulette Wheel*. Dalam implementasinya, proses seleksi *parent* menggunakan *Roulette Wheel* dapat dilakukan dengan

membuat fungsi khusus yang menerima populasi individu dan menghasilkan parent individu. Berikut adalah *source code* yang digunakan.

```
1.  def select(population,fitness_scores):
2.      total_fitness = sum(fitness_scores)
3.      probabilities = [score/total_fitness for score in
    fitness_scores]
4.      cum_probs = [probabilities[0]]
5.      for i in range(1, len(probabilities)):
6.          cum_probs.append(cum_probs[i-1] + probabilities[i])
7.      selected = []
8.      for i in range(len(population)):
9.          r = random.random()
10.         for j, cum_prob in enumerate(cum_probs):
11.             if r < cum_prob:
12.                 # Skip this element if it is the same as the
    previous element
13.                 if i > 0 and population[j] == selected[-1]:
14.                     continue
15.                 # Otherwise, append the element to the list
16.                 selected.append(population[j])
17.                 break
18.     return selected
```

Kode Program 4.12 Seleksi *Parent* pada GA

Berikut adalah penjelasan Kode Program 4.12

1. Baris 1 digunakan untuk mendefinisikan fungsi 'select' yang memiliki 2 parameter masukan yaitu 'population' dan 'fitness_scores'.
2. Baris 2 digunakan untuk Menghitung total skor kebugaran dengan menjumlahkan semua skor dalam 'fitness_scores'.
3. Baris 3 digunakan untuk menghitung probabilitas untuk setiap individu dalam populasi menggunakan rumus 'score/total_fitness' dan menyimpannya di dalam list 'probabilities'.
4. Baris 4 digunakan untuk membuat list 'cum_probs' dan menetapkan probabilitas pertama dari 'probabilities' ke dalamnya.
5. Baris 5 dan 6 digunakan untuk melakukan perulangan iterasi sebanyak 'len(probabilities)-1' dan menambahkan setiap probabilitas pada 'cum_probs'.
6. Baris 7 digunakan untuk membuat list kosong 'selected' yang menampung individu terpilih.
7. Baris 8 digunakan mengulang iterasi sebanyak jumlah individu dalam populasi.

8. Baris 9 digunakan untuk menghasilkan nilai acak antara 0 dan 1 menggunakan fungsi 'random.random()' dan menetapkannya ke variabel 'r'.
9. Baris 10 digunakan untuk mengulang iterasi sebanyak jumlah probabilitas dalam 'cum_probs' dan menentukan individu terpilih berdasarkan nilai acak 'r' dan probabilitas yang sesuai pada 'cum_probs'.
10. Baris 11 digunakan untuk melakukan pengecekan kondisi, jika nilai acak 'r' lebih kecil dari probabilitas kumulatif pada indeks 'j', maka individu pada indeks 'j' yang dipilih.
11. Baris 13 dan baris 14 digunakan untuk melakukan pengecekan kondisi, jika individu terpilih adalah sama dengan individu terpilih sebelumnya, maka individu tersebut diabaikan dan dilanjutkan ke individu berikutnya.
12. Baris 16 digunakan untuk menambahkan elemen ke dalam list daftar individu terpilih
13. Baris 17 digunakan untuk menghentikan looping setelah menambahkan elemen yang terpilih.
14. Baris 18 digunakan untuk mengembalikan daftar elemen atau individu terpilih dari populasi yang diberikan.

4.2.2.5 Konversi Individu Desimal ke Biner

Pada sub bab ini membahas mengenai bagaimana mengkonversikan representasi individu dalam algoritma genetika dari bentuk desimal menjadi biner, sehingga individu dapat diproses pada tahap *crossover* dan mutasi. Dalam implementasinya, proses konversi individu desimal ke biner dapat dilakukan dengan membuat fungsi khusus yang menerima *parent* individu dalam bentuk desimal dan menghasilkan parent individu dalam bentuk biner. Berikut adalah *source code* yang digunakan.

```
1. def decimal_to_binary(parent):
2.     binary = bin(parent)[2:].zfill(8)
3.     binary_digit = [int(bit) for bit in binary]
4.     return binary_digit
```

Kode Program 4.13 Konversi Individu Desimal ke Biner pada GA

Berikut adalah penjelasan Kode Program 4.13

1. Baris 1 digunakan untu mendefinisikan fungsi dengan memiliki satu argumen yaitu 'parent', yaitu angka desimal yang dikonversi ke dalam representasi biner.
2. Baris 2 digunakan untuk melakukan konversi dari angka desimal ke dalam representasi biner dilakukan dengan menggunakan fungsi 'bin'. Kode '[2:]' berfungsi untuk menghilangkan '0b' yang merupakan awalan pada bilangan biner.
3. Baris 3 digunakan untuk melakukan konversi ke dalam bilangan bulat pada setiap digit biner dan disimpan dalam sebuah list.
4. Baris 4 digunakan untuk mengembalikan representasi biner dari angka desimal dalam bentuk daftar bilangan bulat.

4.2.2.6 *Crossover*

Pada sub bab ini membahas mengenai bagaimana operasi *crossover* bekerja. Operasi *crossover* dilakukan dengan memilih titik potong (*crossover point*) acak pada setiap individu parent dan menukar bagian-bagian dari setiap individu sebelum dan sesudah titik potong, yang kemudian menghasilkan individu baru. Berikut adalah *source code* yang digunakan.

```
1.  def crossover(parent, crossover_rate):
2.      offspring = []
3.      for i in range(0, len(parent)):
4.          parent1 = parent[i]
5.          parent2 = parent[(i+1) % len(parent)]
6.          parent1_binary=decimal_to_binary(parent1)
7.          parent2_binary=decimal_to_binary(parent2)
8.          # check if crossover should occur
9.          random_score=random.random()
10.         if  random_score < crossover_rate:
11.             # print("do co")
12.             # choose two random crossover points
13.             crossover1 = random.randint(1,
    len(parent1_binary) - 2)
14.             crossover2 = len(parent1_binary) - crossover1 - 1
15.             # ensure crossover1 is less than crossover2
16.             if crossover1 > crossover2:
17.                 crossover1, crossover2 = crossover2,
    crossover1
18.             # create children by combining segments of
    parents
19.             child1 = parent2_binary[:crossover1] +
    parent1_binary[crossover1:crossover2] +
    parent2_binary[crossover2:]
20.             child2 = parent1_binary[:crossover1] +
    parent2_binary[crossover1:crossover2] +
    parent1_binary[crossover2:]
21.             offspring.append(child1)
```

```
22.             offspring.append(child2)
23.         else:
24.             # if crossover doesn't occur, append parents to
    offspring list
25.             offspring.append(parent1_binary)
26.             offspring.append(parent2_binary)
27.     return offspring
```

Kode Program 4.14 *Crossover* pada GA

Berikut adalah penjelasan Kode Program 4.14

1. Baris 1 digunakan untuk mendefinisikan sebuah fungsi bernama "crossover" yang menerima dua argumen yaitu "parent" dan "crossover_rate".
2. Baris 2 digunakan untuk melakukan inisialisasi variabel "offspring" sebagai sebuah list kosong.
3. Baris 3 digunakan untuk melakukan iterasi pada range 0 hingga panjang "parent".
4. Baris 4 digunakan untuk Mengambil parent pada index i dari "parent".
5. Baris 5 digunakan untuk Mengambil parent pada index (i+1) % panjang "parent".
6. Baris 6 digunakan untuk Mengonversi parent1 ke dalam bentuk biner dengan menggunakan fungsi "decimal_to_binary".
7. Baris 7 digunakan untuk Mengonversi parent2 ke dalam bentuk biner dengan menggunakan fungsi "decimal_to_binary".
8. Baris 9 digunakan untuk menghasilkan nilai random yang bertipe float antara 0 dan 1 dan menyimpannya di dalam variabel "random_score".
9. Baris 10 digunakan untuk melakukan pegecekan kondisi, jika random_score lebih kecil dari crossover_rate, maka crossover dilakukan.
10. Baris 13 hingga baris 17 digunakan untuk menentukan crossover point untuk membuat dua anak baru.
11. Baris19 digunakan untuk membuat anak pertama dengan menggabungkan segmen dari parent2_binary sebelum crossover1, segmen dari parent1_binary antara crossover1 dan crossover2, dan segmen dari parent2_binary setelah crossover2.
12. Baris 20 digunakan untuk membuat anak kedua dengan menggabungkan segmen dari parent1_binary sebelum crossover1, segmen dari

parent2_binary antara crossover1 dan crossover2, dan segmen dari parent1_binary setelah crossover2.

13. Baris 21 dan 22 digunakan untuk menambahkan kedua anak ke dalam list "offspring".
14. Baris 23 hingga 26 digunakan untuk melakukan perintah apabila random_score lebih besar dari crossover_rate, maka crossover tidak dilakukan dan menambahkan kedua parent ke dalam list "offspring".
15. Baris 27 digunakan untuk mengembalikan list "offspring".

4.2.2.7 Mutasi

Pada sub bab ini membahas mengenai bagaimana operasi mutasi bekerja. Dalam implementasinya, operasi ini mengimplementasikan logika mutasi pada setiap elemen dari individu berdasarkan tingkat mutasi yang telah ditentukan. Berikut adalah *source code* yang digunakan.

```
1.  def mutation(offspring,mutation_rate):
2.      # iterate through each element in the offspring
3.      for i in range(len(offspring)):
4.          # iterate through each element in the current
    offspring
5.          for j in range(len(offspring[i])):
6.              # check if the current element should be mutated
7.              if random.uniform(0, 1) < mutation_rate:
8.                  # mutate the current element by flipping its
    value
9.                  offspring[i][j] = 1 - offspring[i][j]
10.     return offspring
```

Kode Program 4.15 Mutasi pada GA

Berikut adalah penjelasan Kode Program 4.15

1. Baris 1 digunakan mendefinisikan fungsi 'mutation' dengan menerima dua parameter 'offspring' dan 'mutation_rate'.
2. Baris 3 digunakan untuk melakukan iterasi setiap elemen di 'offspring'.
3. Baris 5 digunakan untuk melakukan iterasi setiap elemen di offspring saat ini.
4. Baris 7 digunakan untuk memeriksa apakah elemen saat ini harus dimutasi dengan membandingkan nilai acak antara 0 dan 1 dengan 'mutation_rate'.
5. Baris 9 digunakan jika elemen saat ini harus dimutasi, nilai elemen tersebut diputar dengan membalik nilainya.
6. Baris 10 digunakan untuk mengembalikan offspring yang sudah dimutasi.

4.2.2.8 Konversi Individu Biner ke Desimal

Pada sub bab ini membahas mengenai bagaimana mengkonversikan representasi *offspring* individu dalam algoritma genetika dari bentuk biner menjadi desimal. Dalam implementasinya, proses konversi individu biner ke desimal dapat dilakukan dengan membuat fungsi khusus yang menerima *list offspring* individu dalam bentuk biner dan menghasilkan *list offspring* individu dalam bentuk desimal. Berikut adalah *source code* yang digunakan.

```
1.  def binary_to_decimal(mutate):
2.      decimals = []
3.      for binary in mutate:
4.          decimal = 0
5.          for i, digit in enumerate(binary[::-1]):
6.              decimal += digit * (2 ** i)
7.          if(decimal<=2):
8.              continue
9.          else:
10.             decimals.append(decimal)
11.     return decimals
```

Kode Program 4.16 Konversi Individu Biner ke Desimal pada GA

Berikut adalah penjelasan Kode Program 4.16

1. Baris 1 digunakan untuk mendefinisikan fungsi 'binary_to_decimal' dengan menerima satu parameter 'mutate'.
2. Baris 3 digunakan untuk melakukan iterasi setiap elemen di 'mutate'.
3. Baris 5 digunakan untuk melakukan iterasi setiap digit dalam elemen biner saat ini dari digit terakhir hingga digit pertama.
4. Baris 6 digunakan untuk mengubah digit biner menjadi desimal.
5. Baris 7 digunakan untuk melakukan pengecekan, jika nilai desimal kurang dari atau sama dengan 2, maka dilewati.
6. Baris 10 digunakan untuk menambahkan nilai desimal saat ini ke daftar 'decimals'.
7. Baris 11 digunakan untuk return decimals : mengembalikan daftar desimal yang dihasilkan dari elemen biner di 'mutate'.

4.2.2.9 Update Populasi

Sub bab ini membahas mengenai bagaimana proses memperbarui populasi dalam algoritma genetika. Dalam implementasinya, mewakili bagaimana individu yang lebih baik (yaitu, individu dengan nilai fitness yang lebih tinggi) dipilih

untuk diteruskan ke generasi berikutnya, sementara individu yang kurang baik dibuang. Berikut adalah *source code* yang digunakan.

```
1. def update_population(population,offspring_decode,pop_size,data,target):
2.     population+=offspring_decode
3.     population=list(set(population))
4.     # create a list to store the fitness scores
5.     fitness_scores = []
6.     # calculate the fitness scores for each individual in the population
7.     for individu in population:
8.         score = fitness(individu,data,target)
9.         fitness_scores.append(score)
10.    # pair up each individual with its fitness score
11.    population_with_scores = list(zip(population, fitness_scores))
12.    # sort the population by fitness score
13.    sorted_population_with_scores = sorted(population_with_scores, key=lambda x: x[1])[:pop_size]
14.    # unpack the sorted population and fitness scores
15.    sorted_population, sorted_scores = zip(*sorted_population_with_scores)
16.    return zip(*sorted_population_with_scores)
```

Kode Program 4.17 Update Populasi pada GA

Berikut adalah penjelasan Kode Program 4.17

1. Baris 1 digunakan untuk mendefinisikan fungsi 'update_population' dengan menerima lima parameter 'population', 'offspring_decode', 'pop_size', 'data', dan 'target'.
2. Baris 2 digunakan untuk menambahkan offspring_decode ke dalam population.
3. Baris 3 digunakan untuk menghapus duplikat dalam population dengan mengubahnya menjadi himpunan, kemudian kembali mengubahnya menjadi list.
4. Baris 5 digunakan untuk membuat list kosong 'fitness_scores' untuk menyimpan nilai kebugaran.
5. Baris 7 digunakan untuk melakukan iterasi setiap individu di dalam 'population'.
6. Baris 8 digunakan untuk menghitung skor kebugaran untuk individu saat ini menggunakan fungsi 'fitness'.
7. Baris 9 digunakan untuk menambahkan skor kebugaran individu saat ini ke dalam daftar 'fitness_scores'.

8. Baris 11 digunakan untuk membuat daftar tupel yang berisi pasangan antara setiap individu di 'population' dan skor kebugarannya.
9. Baris 13 digunakan untuk mengurutkan pasangan individu-skor kebugaran berdasarkan skor kebugaran, kemudian membatasi hasilnya hingga jumlah populasi yang diinginkan ('pop_size').
10. Baris 15 digunakan untuk memisahkan populasi yang sudah diurutkan dari skor kebugarannya menjadi dua tupel terpisah.
11. Baris 16 digunakan untuk mengembalikan pasangan individu-skor kebugaran yang sudah diurutkan.

4.2.3 *k-Nearest Neighbor* (*k*-NN)

Setelah melalui tahap *preprocessing*. Tahap selanjutnya adalah membuat sistem imputasi *missing value* menggunakan algoritma *k-Nearest Neighboor* (*k*-NN). Berikut adalah penjelasan dari pembuatan sistem dalam tiap proses pada *k*-NN.

4.2.3.1 Membaca Dataset

Dataset yang digunakan dalam proses imputasi k-NN ini adalah dataset yang telah melalui proses normalisasi. Berikut adalah *source code* yang digunakan.

```
1. # Read the CSV file into a NumPy array
2. scaled_data =
   np.genfromtxt("D:/SIDANG/csv/preprocessing_scaled.csv",
   delimiter=",",usecols=[0,1,2,3,4],skip_header=1)
3. scaled_data_target =
   np.genfromtxt("D:/SIDANG/csv/preprocessing_scaled.csv",
   delimiter=",",usecols=5,skip_header=1)
4. # Create a list containing attribute names to create a data
   frame after imputation
5. cols=['UMUR','JENIS_KELAMIN','FOTO_TORAKS','STATUS_HIV','RIWAYA
   T_DIABETES','LOKASI_ANATOMI']
```

Kode Program 4.18 Membaca Dataset pada *k*-NN

Berikut adalah penjelasan Kode Program 4.18

1. Baris 2 menggunakan NumPy untuk membaca file CSV dari alamat file yang diberikan dan menyimpan datanya dalam variabel "scaled_data". Hanya kolom pertama hingga kelima (indeks 0 hingga 4) yang digunakan. Baris pertama dari file CSV dilewati karena berisi nama kolom.
2. Baris 3 menggunakan NumPy untuk membaca file CSV yang sama seperti baris sebelumnya dan menyimpannya dalam variabel "scaled_data_target".

Hanya kolom keenam (indeks 5) yang digunakan. Baris pertama dari file CSV dilewati karena berisi nama kolom.

3. Baris 5 digunakan untuk membuat variabel "cols" yang berisi daftar nama kolom yang digunakan saat membuat data frame setelah imputasi.

4.2.3.2 Mencari *Missing Value*

Setelah membaca dataset, hal yang selanjutnya dilakukan adalah mencari *index* dari nilai yang hilang. Berikut adalah *source code* yang digunakan.

```
1. # Find the indices of the missing values
2. missing_value_indices = np.where(np.isnan(scaled_data))
```

Kode Program 4.19 Mencari *Missing value* pada *k*-NN

Berdasarkan Kode Program 4.19, baris 2 digunakan untuk untuk menemukan indeks nilai yang hilang (NaN) di dalam "scaled_data" menggunakan np.where dan menyimpannya dalam variabel "missing_value_indices".

4.2.3.3 Inisialisasi Nilai *k*

Inisialisasi nilai k pada k-NN tanpa optimasi atau konvensional, langsung dilakukan dengan mendefinisikan nilai k yang digunakan. Pada penelitian ini, terdapat empat varian nilai *k* konvensional, antara lain 3, 5, 7, dan 9. Berikut adalah *source code* yang digunakan.

```
1. k_value=[3,5,7,9,11,13,15]
2. for k in k_value:
3.     knn(k)
```

Kode Program 4.20 Inisialisasi Nilai *k* pada *k*-NN

Berikut adalah penjelasan Kode Program 4.20

1. Baris 1 digunakan untuk membuat variabel "k_value" yang berisi daftar nilai k yang digunakan untuk KNN.
2. Baris 2 dan 3 digunakan untuk melakukan iterasi pada setiap nilai k yang ada di dalam "k_value" dan memanggil fungsi "knn" dengan nilai k saat ini sebagai argumen.

4.2.3.4 Menentukan Jarak *Euclidean Distance*

Setelah melakukan inisialisasi nilai *k*, proses yang dilakukan selanjutnya adalah melakukan perhitungan jarak *euclidean distance*. Proses *euclidean*

distance dilakukan di dalam fungsi *knn()*. Berikut adalah *source code* yang digunakan.

```
1. def knn(k):
2.     start_time = time.time()
3.     # Iterate through the missing values
4.     for i, j in zip(*missing_value_indices):
5.         distances=[]
6.         not_mv_indices=[]
7.         use_data=[]
8.         for x in range(len(scaled_data)):
9.             if(np.isnan(scaled_data[x][j])==True):
10.                continue
11.            else:
12.                not_mv_indices.append(x)
13.                distance=0
14.                unuse_col1,unuse_col2=[],[]
15.                for y in range(len(scaled_data[0])):
16.                    if(np.isnan(scaled_data[i][y])==True or
   np.isnan(scaled_data[x][y])==True):
17.                        unuse_col1.append(y)
18.                        unuse_col2.append(y)
19.                join=unuse_col1+unuse_col2
20.                unuse_cols=np.unique(join)
21.        # Calculate the distance between the missing value
   and all other points in the data
22.                euc=0
23.                for y in range(len(scaled_data[0])):
24.                    if(y not in unuse_cols):
25.                        euc+=(scaled_data[i][y]-
   scaled_data[x][y])*(scaled_data[i][y]-scaled_data[x][y])
26.                euclidean=math.sqrt(euc)
27.            distances.append(euclidean)
28.            use_data.append(x)
```

Kode Program 4.21 Menentukan Jarak *Euclidean Distance* pada k-NN

Berikut adalah penjelasan Kode Program 4.21

1. Baris 1 digunakan untuk mendefinisikan sebuah fungsi bernama "knn" yang menerima satu argumen yaitu nilai k.
2. Baris 2 digunakan untuk membuat variabel "start_time" untuk mengukur waktu eksekusi kode.
3. Baris 4 digunakan untuk melakukan iterasi pada indeks nilai yang hilang (NaN) yang telah dihitung sebelumnya.
4. Baris 5 digunakan untuk membuat variabel "distances" yang menampung jarak antara nilai yang hilang dan nilai-nilai lain di data.
5. Baris 6 digunakan untuk membuat variabel "not_mv_indices" yang menampung indeks dari baris data yang tidak mengandung nilai yang hilang.

6. Baris 7 digunakan untuk membuat variabel "use_data" yang menampung indeks dari semua baris data.
7. Baris 8 digunakan untuk melakukan iterasi pada semua baris data di "scaled_data".
8. Baris 9 hingga 11 dilakukan untuk melakukan pengecekan, jika nilai di kolom yang sedang diiterasi adalah NaN, maka lewati iterasi saat ini. Jika nilai di kolom yang sedang diiterasi bukan NaN, maka lanjut ke baris 12-27.
9. Baris 12 digunakan untuk menambahkan indeks baris saat ini ke dalam "not_mv_indices".
10. Baris 13 digunakan untuk membuat variabel "distance" yang menampung jarak antara nilai yang hilang dan nilai dari baris saat ini.
11. Baris 14 digunakan untuk membuat variabel "unuse_col1" dan "unuse_col2" untuk menampung indeks kolom yang mengandung NaN untuk nilai yang hilang dan nilai dari baris saat ini, masing-masing.
12. Baris 15 digunakan untuk melakukan iterasi pada semua kolom dalam "scaled_data".
13. Baris 16 digunakann untuk melakukan pengecekan, jika nilai di baris yang hilang atau baris saat ini adalah NaN, tambahkan indeks kolom saat ini ke dalam "unuse_col1" dan "unuse_col2".
14. Baris 19 dan 20 digunakan untuk menggabungkan "unuse_col1" dan "unuse_col2" menjadi satu array dan menghapus duplikatnya. Simpan hasilnya di "unuse_cols".
15. Baris 22 digunakan untuk membuat variabel "euc" yang menampung jarak Euclidean antara nilai yang hilang dan nilai dari baris saat ini.
16. Baris 23 digunakan untuk melakukan iterasi pada semua kolom dalam "scaled_data".
17. Baris 24 digunakan untuk melakukan pengecekan, jika kolom sedang diiterasi bukan termasuk dalam "unuse_cols", lanjut ke baris 25-26.
18. Baris 25 digunakan untuk menghitung jarak Euclidean dan menambahkannya ke "euc".

19. Baris 26 digunakan untuk menghitung akar kuadrat dari "euc" dan simpan hasilnya di "euclidean".
20. Baris 27 dan 28 digunakan untuk menambahkan "euclidean" ke dalam "distances"dan menambahkan indeks baris saat ini ke dalam "use_data".

4.2.3.5 Pengurutan Jarak dan Menentukan *k* Observasi Terdekat

Setelah mendapatan nilai *euclidean distance*, kemudian dilanjutkan dengan mengurutkan nilai jarak yang didapatkan dan menentukan *k* obeservasi terdekat berdasarkan jarak tersebut. Proses tersebut masih dilakukan di dalam fungsi *knn()* dan masih dalam perulangan *loop* melalui nilai-nilai yang hilang pada *missing_value_indices*. Berikut adalah *source code* yang digunakan.

```
1. # Find the indices of the k nearest neighbors
2.         nearest_neighbors = np.argpartition(distances, k)[:k]
```

Kode Program 4.22 Pengurutan Jarak dan Menentukan *k* Observasi Terdekat pada *k*-NN

Berdasarkan Kode Program 4.22, baris 2 digunakan untuk menghitung k tetang*Gate*rdekat dengan menggunakan argpartition dari numpy untuk menghasilkan indeks dari array distances yang telah diurutkan sehingga kita dapat mengambil k tetang*Gate*rdekat.

4.2.3.6 Imputasi k-NN

Setelah mendapatkan tetang*Gate*rdekat, kemudian dilanjutkan dengan melakukan imputasi dengan target berdasarkan nilai label pada tetang*Gate*rsebut. Proses imputasi masih dilakukan di dalam fungsi *knn()* dan masih dalam perulangan *loop* melalui nilai-nilai yang hilang pada *missing_value_indices*. Berikut adalah *source code* yang digunakan.

```
1. # Calculate the mode of the target variable for the k nearest
   neighbors
2.         most_target=[]
3.         for label in nearest_neighbors:
4.             most_target.append(scaled_data[use_data[label],
   j])
5.         mode_target_value = mode(most_target)[0][0]
6.         scaled_data[i][j] = mode_target_value
```

Kode Program 4.23 Imputasi pada *k*-NN

Berikut adalah penjelasan Kode Program 4.23

1. Baris 2 digunakan untuk membuat list kosong untuk menampung nilai target dari k tetang*Gate*rdekat.

2. Baris 3 digunakan untuk melakukan iterasi melalui indeks dari k tetang*Gater*dekat.
3. Baris 4 digunakan untuk menambahkan nilai target (scaled_data_target) dari k tetang*Gater*dekat ke dalam daftar most_target.
4. Baris 5 digunakan untuk menghitung nilai modus dari most_target menggunakan fungsi mode dari scipy.
5. Baris 6 digunakan untuk mengganti nilai yang hilang pada baris ke-i dan kolom ke-j dalam array scaled_data dengan nilai modus dari most_target.

4.2.3.7 Menyimpan Dataset Imputasi

Setelah dataset di imputasi dan sudah tidak ada data yang *missing*, dataset tersebut kemudian disimpan ke dalam file csv. Proses tersebut masih dilakukan di dalam fungsi *knn()*. Berikut adalah *source code* yang digunakan.

```
1.  # Print the imputed data into csv
2.      to_df1=[]
3.      for i in range(len(scaled_data)):
4.          x=np.append(scaled_data[i],scaled_data_target[i])
5.          to_df1.append(x)
6.      udtf1= pd.DataFrame(to_df1,columns=cols)
7.  udtf1.to_csv('D:/SIDANG/csv/skenario1_k'+str(k)+'.csv',index=Fa
    lse)
8.      end_time = time.time()
9.      total_time = round(end_time - start_time, 2)
10.     print("     time required for imputation
    :",total_time,"second....")
```

Kode Program 4.24 Menyimpan Dataset yang Telah Melalui Imputasi *k*-NN

Berikut adalah penjelasan Kode Program 4.24

1. Baris 2 digunakan untuk mendeklarasikan sebuah variabel dengan nama "to_df1" yang berisi sebuah list kosong. List ini digunakan untuk menyimpan data yang diimputasi.
2. Baris 3 digunakan untuk memulai perulangan for sebanyak data yang ada di dalam "scaled_data". "len(scaled_data)" berfungsi untuk menghitung jumlah data yang ada pada variabel "scaled_data".
3. Baris 4 digunakan untuk menggabungkan setiap data pada "scaled_data" dengan data pada "scaled_data_target" menggunakan fungsi "np.append()". Data yang telah digabungkan kemudian disimpan dalam variabel x.

4. Baris 5 digunakan untuk menambahkan data yang telah digabungkan ke dalam list "to_df1" dengan menggunakan fungsi "append()".
5. Baris 6 digunakan untuk membuat DataFrame dengan menggunakan list "to_df1" sebagai data dan "cols" sebagai kolom pada DataFrame. DataFrame yang telah dibuat kemudian disimpan dalam variabel "udtf1".
6. Baris 7 digunakan untuk menyimpan DataFrame "udtf1" ke dalam format csv dengan menggunakan fungsi "to_csv()". Lokasi dan nama file yang digunakan untuk menyimpan file csv diatur dalam argumen pada fungsi "to_csv()".
7. Baris 8 digunakan untuk mendeklarasikan sebuah variabel dengan nama "end_time" dan mengisi variabel tersebut dengan waktu saat ini.
8. Baris 9 digunakan untuk menghitung selisih waktu dari waktu awal program dijalankan hingga waktu akhir program dijalankan menggunakan variabel "start_time" dan "end_time". Hasil perhitungan kemudian dibulatkan menjadi 2 digit angka dibelakang koma dan disimpan dalam variabel "total_time".
9. Baris 10 digunakan untuk mencetak output running yang berisi waktu yang diperlukan untuk melakukan imputasi data. Pesan ini muncul setelah program selesai dijalankan.

4.2.4 *Long Short-Term Memory (LSTM)*

Pada sub bab ini menjelaskan mengenai *pseudecode* dari metode LSTM yang digunakan untuk menentukan klasifikasi *tuberculosis*. Penggunaan parameter LSTM meliputi nilai *neuron* dan *epoch*, didapatkan dari varian pada skenario pengujian 7 dan skenario pengujian 8. Kemudian, untuk parameter lainnya didasari oleh penelitian yang telah dilakukan oleh Pramita Sree Muhuri, dkk, 2020 [31]. Nilai tiap parameter yang digunakan dapat dilihat pada Tabel 4.2

Tabel 4.2 Nilai Parameter LSTM.

No.	***Hyper-Parameter***	**Nilai**
1.	*Learning Rate*	0.01
2.	*Dropout*	0.3
3.	*Activation Function*	*Sigmoid* dan *Tanh*
4.	*Optimizer*	*Adam*

No.	*Hyper-Parameter*	Nilai
5.	*Batch Size*	50
6.	*Hidden Layer*	2
7.	*Neuron*	Varian 6, 12, 18, 24, 30
8.	*Epoch*	Varian 25, 50, 75, 100, 125, 150, 175, 200

Setelah menentukan nilai parameter yang digunakan pada model, berikut adalah penjelasan dari pembuatan sistem dalam tiap proses pada LSTM.

4.2.4.1 Membaca *Dataset*

Pada sub bab ini menjelaskan bagaimana untuk mempersiapkan data yang digunakan oleh model. *Dataset* diambil dan dibaca menggunakan *library pandas*. Kemudian, data tersebut dibagi menjadi data atribut dan data target untuk mempermudah proses pembuatan model. Berikut adalah *source code* yang digunakan.

```
1. data = pd.read_csv("D:/SIDANG/csv/skenario6_k3.csv")
2. x = data.drop(["LOKASI_ANATOMI"], axis=1)
3. y = data["LOKASI_ANATOMI"]
4. x=np.array(x)
5. y=np.array(y)
```

Kode Program 4.25 Membaca *Dataset* pada LSTM

Berikut adalah penjelasan Kode Program 4.25

1. Baris 1 digunakan untuk membaca file csv yang berisi data dan menyimpannya ke dalam variabel "data".
2. Baris 2 digunakan untuk menghapus kolom "LOKASI_ANATOMI" pada data dan menyimpan hasilnya ke dalam variabel "x".
3. Baris 3 digunakan untuk mengambil kolom "LOKASI_ANATOMI" pada data dan menyimpan hasilnya ke dalam variabel "y".
4. Baris 4 digunakan untuk mengubah data pada variabel "x" menjadi numpy array dan menyimpan hasilnya kembali ke variabel "x".
5. Baris 5 digunakan untuk mengubah data pada variabel "y" menjadi numpy array dan menyimpan hasilnya kembali ke variabel "y".

4.2.4.2 Pembagian Data

Sebelum melakukan klasifikasi menggunakan metode LSTM, dilakukan pembagian data terhadap dataset yang telah diimputasi terlebih dahulu.

Pembagian data dilakukan untuk menghasilkan *data training*, *data testing*, dan *data validation*. Berikut adalah *source code* yang digunakan.

```
1. fold = 0
2. kfold = KFold(n_splits=10, shuffle=False, random_state=None)
3. # Iteration for each K-Fold Split
4. for train, test in kfold.split(x, y):
5.  fold+=1
6.  # Split the data into test sets, train sets, val sets
7. X_train, X_test = x[train], x[test]
8. y_train, y_test = y[train], y[test]
9.  X_val,y_val =
   X_train[:int(X_train.shape[0]*0.2)],y_train[:int(y_train.shape[
   0]*0.2)]
```

Kode Program 4.26 Pembagian Data

Berikut adalah penjelasan Kode Program 4.26

1. Baris 1 digunakan untuk mendeklarasikan variabel "fold" dengan nilai awal 0.
2. Baris 2 digunakan untuk mendeklarasikan variabel "kfold" sebagai objek KFold dengan jumlah split sebanyak 10, tidak melakukan shuffle, dan nilai random state diatur ke None.
3. Baris 4 digunakan untuk menjalankan perulangan for dengan mengiterasi objek "kfold.split(x,y)". Variabel "train" dan "test" yang berisi indeks untuk data train dan test yang digunakan pada setiap split.
4. Baris 5 digunakan untuk menambahkan 1 ke nilai variabel "fold" setiap kali perulangan for dijalankan.
5. Baris 7 digunakan untuk membagi data pada variabel "x" menjadi data train dan data test sesuai dengan indeks yang didapatkan dari variabel "train" dan "test". Data train disimpan dalam variabel "X_train" dan data test disimpan dalam variabel "X_test".
6. Baris 8 digunakan untuk membagi data pada variabel "y" menjadi data train dan data test sesuai dengan indeks yang didapatkan dari variabel "train" dan "test". Data train disimpan dalam variabel "y_train" dan data test disimpan dalam variabel "y_test".
7. Baris 9 digunakan untuk membagi data train lagi menjadi data train dan data validation (val) dengan cara mengambil 20% dari data train. Data train disimpan dalam variabel "X_train" dan data validation disimpan dalam variabel "y_train".

4.2.4.3 Membuat LSTM *Model*

Sub bab ini bertujuan untuk menciptakan sebuah model LSTM yang digunakan untuk melakukan klasifikasi. Dalam sub bab ini, terdapat beberapa parameter seperti jumlah layer LSTM, jumlah unit pada setiap layer, dan tipe aktivasi yang ditentukan. Selain itu, dalam sub bab ini juga ditentukan optimizer dan *loss* function yang digunakan untuk melatih model. Berikut adalah *source code* yang digunakan.

```
1.  def lstmModel(X_train,y_train,X_val,y_val):
2.      batch=50
3.      epoch=25
4.      output_size=1
5.      neurons=24
6.      activ_func='sigmoid'
7.      dropout=0.3
8.      loss='binary_crossentropy'
9.      optimizer = optimizers.Adam(learning_rate=0.01)
10.     #Build the Model
11.     model = Sequential()
12.     model.add(LSTM(neurons, input_shape=(X_train.shape[1],
    1),return_sequences=True))
13.     model.add(Dropout(dropout))
14.     model.add(LSTM(neurons,return_sequences=False))
15.     model.add(Dropout(dropout))
16.     model.add(Dense(output_size, activation=activ_func))
17.     # Compile the model
18.     model.compile(optimizer=optimizer, loss=loss,
    metrics=['accuracy'])
19.     # Train the model
20.     model.fit(X_train, y_train,validation_data=(X_val,y_val),
    epochs=epoch, batch_size=batch,shuffle=False,verbose=0)
21.     return model
```

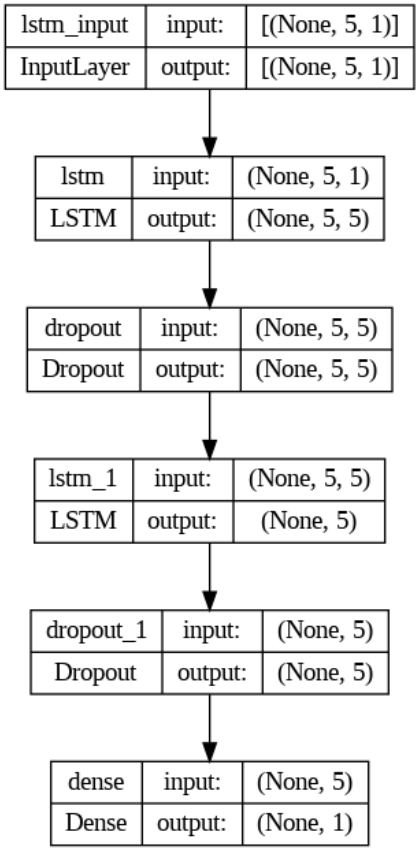
Kode Program 4.27 Membuat LSTM *Model*

Berikut adalah penjelasan Kode Program 4.27

1. Baris 1 digunakan untuk membuat sebuah fungsi dengan nama "lstmModel" dan memasukkan parameter X_train, y_train, X_val, y_val.
2. Baris 2 hingga 9 digunakan untuk inisiasi beberapa parameter untuk model seperti batch size, epoch, output size, neuron, activation function, dropout, loss function, dan optimizer.
3. Baris 11 digunakan untuk membangun model menggunakan Sequential() dari Keras.
4. Baris 12 digunakan untuk menambahkan layer LSTM dengan jumlah neuron yang sudah di set sebelumnya dan dimensi input yang diambil dari data X_train.

5. Baris 13 digunakan untuk menambahkan dropout layer pada model.
6. Baris 14 digunakan untuk menambahkan layer LSTM kedua dengan jumlah neuron dan tanpa mengembalikan hasil dalam bentuk sequences.
7. Baris 15 digunakan untuk menambahkan dropout layer lagi pada model.
8. Baris 16 digunakan untuk menambahkan layer Dense pada model untuk menghasilkan output dengan activation function yang telah di set sebelumnya.
9. Baris 18 digunakan untuk melakukan compile model dengan menggunakan optimizer, loss function, dan metrics untuk evaluasi model.
10. Baris 20 digunakan untuk melakukan training model dengan menggunakan data X_train dan y_train, serta memasukkan data validation X_val dan y_val dengan jumlah epoch dan batch size yang sudah di set sebelumnya.
11. Baris 21 digunakan untuk mengembalikan model yang sudah dibangun.

Struktur model LSTM dapat diperoleh setelah melakukan proses pelatihan. Struktur tersebut berisi informasi mengenai *layer*, *input shape*, dan *output shape*. *Plot* struktur LSTM yang dihasilkan dan digunakan dapat dilihat pada Gambar 4.1.



Gambar 4.1 Struktur Model LSTM

Struktur LSTM yang digunakan untuk membandingkan kinerja *k-NN*+LSTM dan *k-NN*+GA+LSTM adalah sama. Struktur di atas digunakan untuk semua skenario pengujian.

4.2.4.4 Membuat LSTM *Model Plot*

Pada sub bab ini membahas mengenai bagaimana proses visualisasi hasil dari model yang dibuat. Performa model dalam memprediksi data dapat dilihat melalui plot grafik. Pada implementasinya, plot ini menampilkan grafik performa dari *data training* dan data validasi. Berikut adalah *source code* yang digunakan.

```
1.  def modelPlot(do_model):
2.      # Plot the training accuracy and loss
3.      # Create a figure with 2 subplots
4.      fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,3))
5.      # Plot the training accuracy in the first subplot
6.      ax1.plot(do_model.history.history['accuracy'])
7.      ax1.plot(do_model.history.history['val_accuracy'])
8.      ax1.set_title('Model Accuracy')
9.      ax1.set_xlabel('Epoch')
10.     ax1.set_ylabel('Accuracy')
11.     ax1.legend(['Train', 'Validation'], loc='upper left')
12.     # Plot the training loss in the second subplot
13.     ax2.plot(do_model.history.history['loss'])
```

```
14.     ax2.plot(do_model.history.history['val_loss'])
15.     ax2.set_title('Model Loss')
16.     ax2.set_xlabel('Epoch')
17.     ax2.set_ylabel('Loss')
18.     ax2.legend(['Train', 'Validation'], loc='upper left')
19.     # Show the plot
20.     plt.show()
```

Kode Program 4.28 Membuat LSTM *Model Plot*

Berikut adalah penjelasan Kode Program 4.28

1. Baris 1 digunakan untuk mendefinisikan sebuah fungsi dengan nama modelPlot yang menerima parameter do_model.
2. Baris 4 digunakan untuk menginisialisasi objek fig dan ax1 serta ax2 yang masing-masing merepresentasikan gambar dan subplot pertama dan kedua. Kemudian menetapkan ukuran gambar dengan figsize.
3. Baris 6 digunakan untuk memanggil metode plot pada ax1 dengan parameter history dari model dan metrik accuracy untuk data latih.
4. Baris 7 digunakan untuk memanggil metode plot pada ax1 dengan parameter history dari model dan metrik accuracy untuk data validasi.
5. Baris 8 digunakan untuk memberikan judul "Model Accuracy" pada subplot pertama.
6. Baris 9 digunakan untuk memberikan label "Epoch" pada sumbu x pada subplot pertama.
7. Baris 10 digunakan untuk memberikan label "Accuracy" pada sumbu y pada subplot pertama.
8. Baris 11 digunakan untuk menambahkan legenda pada subplot pertama untuk masing-masing plot data latih dan data validasi.
9. Baris 13 digunakan untuk memanggil metode plot pada ax2 dengan parameter history dari model dan metrik loss untuk data latih.
10. Baris 14 digunakan untuk memanggil metode plot pada ax2 dengan parameter history dari model dan metrik loss untuk data validasi.
11. Baris 15 digunakan untuk memberikan judul "Model Loss" pada subplot kedua.
12. Baris 16 digunakan untuk memberikan label "Epoch" pada sumbu x pada subplot kedua.
13. Baris 17 digunakan untuk memberikan label "Loss" pada sumbu y pada subplot kedua.

14. Baris 18 digunakan untuk menambahkan legenda pada subplot kedua untuk masing-masing plot data latih dan data validasi.
15. Baris 20 digunakan untuk menampilkan plot hasil visualisasi.

4.2.4.5 Melakukan Evaluasi LSTM *Model*

Dalam evaluasi model, hal pertama yang perlu dilakukan memanggil model LSTM dan menjalankannya. Berikut adalah source code yang digunakan.

```
1.  # Do Model
2.      do_model = lstmModel(X_train,y_train,X_val,y_val)
3.      print("Fold",fold,":","\nPlot Model Training")
4.      modelPlot(do_model)
```

Kode Program 4.29 Melatih Model pada LSTM

Berikut adalah penjelasan Kode Program 4.29

1. Baris 2 digunakan untuk menampung hasil dari fungsi lstmModel yang memodelkan data dengan menggunakan parameter X_train, y_train, X_val, dan y_val.
2. Baris 3 digunakan untuk mencetak hasil dari pengujian fold yang sedang berjalan.
3. Baris 4 digunakan untuk memanggil fungsi yang memplot model yang telah dibuat pada baris sebelumnya.

Setelah melatih model, kemudian menentukan akurasi dan *loss* dari setiap model yang dilatih. Berikut adalah source code yang digunakan.

```
1.  # Get Accuracy the training
2.      average_train_acc =
    np.mean(do_model.history.history['accuracy'])
3.      train_acc_each_fold.append(average_train_acc* 100)
4.      # Get Loss the training
5.      average_train_loss =
    np.mean(do_model.history.history['loss'])
6.      train_loss_each_fold.append(average_train_loss)
```

Kode Program 4.30 Menentukan Akurasi dan *Loss* Setiap Model pada LSTM

Berikut adalah penjelasan Kode Program 4.30.

1. Baris 2 digunakan untuk menghitung rata-rata akurasi dari data latih dengan menggunakan np.mean dan memanggil nilai akurasi pada history.
2. Baris 3 digunakan untuk menambahkan hasil akurasi data latih setiap fold ke dalam list train_acc_each_fold.
3. Baris 5 digunakan untuk menghitung rata-rata loss dari data latih dengan menggunakan np.mean dan memanggil nilai loss pada history.

4. Baris 6 digunakan untuk menambahkan hasil loss data latih setiap fold ke dalam list train_loss_each_fold.

Setelah membuat dan melatih model, selanjutnya dilakukan evaluasi dan prediksi terhadap *data testing*. Untuk melakukan evaluasi dan prediksi *data testing*, pertama dilakukan dengan membuat model baru yang sama dengan model yang sudah dibuat sebelumnya dan dilatih dengan data seluruh data latih. Setelah itu dilakukan evaluasi dengan memasukkan *data testing* dan melihat akurasi model dalam menentukan hasil. Kemudian dilakukan prediksi dengan memasukkan input dari *data testing* dan melihat hasil yang diperoleh dari model. Berikut adalah source code yang digunakan.

```
1.      #Evaluate Testing
2.      test_loss, test_acc = do_model.evaluate(X_test,
    y_test,verbose=0)
3.      acc_each_fold.append(test_acc * 100)
4.      loss_each_fold.append(test_loss)
5.      # Get Prediction
6.      y_predict=do_model.predict(X_test,verbose=0)
7.      y_predict = (y_predict > 0.5).astype(int)
8.      global_y_true.append(y_test)
9.      global_y_predict.append(y_predict)
```

Kode Program 4.31 Evaluasi dan Prediksi *Testing Data* pada LSTM

Berikut adalah penjelasan Kode Program 4.31.

1. Baris 2 digunakan untuk menghitung loss dan akurasi dari data uji dengan menggunakan evaluate pada model do_model.
2. Baris 3 digunakan untuk menambahkan hasil akurasi data uji setiap fold ke dalam list acc_each_fold.
3. Baris 4 digunakan untuk menambahkan hasil loss data uji setiap fold ke dalam list loss_each_fold.
4. Baris 6 digunakan untuk memprediksi label pada data uji dengan menggunakan model do_model.
5. Baris 7 digunakan untuk mengubah prediksi menjadi 0 atau 1 berdasarkan threshold 0.5.
6. Baris 8 digunakan untuk menambahkan nilai sebenarnya pada data uji ke dalam list global_y_true.
7. Baris 9 digunakan untuk menambahkan hasil prediksi pada data uji ke dalam list global_y_predict.

Kemudian, proses selanjutnya yaitu menghitung rata-rata *global* akurasi dan *loss* pada *data testing* dan *training*. *Source code* yang digunakan adalag sebagai berikut.

```
1. test_acc=np.mean(acc_each_fold)
2. test_loss=np.mean(loss_each_fold)
3. train_acc=np.mean(train_acc_each_fold)
4. train_loss=np.mean(train_loss_each_fold)
```

Kode Program 4.32 Menentukan *Global* Akurasi dan *Loss* pada LSTM

Berikut adalah penjelasan Kode Program 4.32

1. Baris 1 digunakan untuk menghitung rata-rata akurasi dari seluruh fold pada data uji dan menampungnya dalam variabel test_acc.
2. Baris 2 digunakan untuk menghitung rata-rata loss dari seluruh fold pada data uji dan menampungnya dalam variabel test_loss.
3. Baris 3 digunakan untuk menghitung rata-rata akurasi dari seluruh fold pada data latih dan menampungnya dalam variabel train_acc.
4. Baris 4 digunakan untuk menghitung rata-rata loss dari seluruh fold pada data latih dan menampungnya dalam variabel train_loss.

Setelah mendapatkan akurasi dan *loss*, langkah yang dilakukan selanjutnya adalah menentukan *confusin matrix* dari seluruh hasil prediksi *data testing*. Dalam menentukan *confusion matrix* memerlukan kumpulan data hasil prediksi dan data target asli yang digunakan sebagai acuan. Dalam kode program, data hasil prediksi dan data target dari setiap *fold* dikumpulkan dalam variabel *global_y_predict* dan *global_y_true*. Kemudian, dari kedua data tersebut dilakukan perhitungan *confusion matrix* dengan menggunakan fungsi *confusion_matrix()* dari *library scikit-learn*. Hasil dari *confusion matrix* memberikan informasi mengenai performa model dalam menentukan nilai positif atau negatif. Berikut adalah source code yang digunakan.

```
1. # Concatenate the fold's global_y_true and global_y_predict
   of test data
2. global_y_true= np.concatenate(global_y_true)
3. global_y_predict=np.concatenate(global_y_predict)
4. # Calculate the confusion matrix of global test data
5. cm_test = confusion_matrix(global_y_true, global_y_predict)
6. # Convert the confusion matrix to a dataframe of test data
7. df_test = pd.DataFrame(cm_test, columns=["Predicted Ekstra
   Paru", "Predicted Paru"],
8. index=["Actual Ekstra Paru", "Actual Paru"])
```

Kode Program 4.33 Menentukan *Global Confusion Matrix* pada LSTM

Berikut adalah penjelasan Kode Program 4.33.

1. Baris 2 digunakan untuk menggabungkan seluruh nilai sebenarnya dari seluruh fold pada data uji dan menampungnya dalam variabel global_y_true.
2. Baris 3 digunakan untuk menggabungkan seluruh prediksi dari seluruh fold pada data uji dan menampungnya dalam variabel global_y_predict.
3. Baris 5 digunakan untuk menghitung confusion matrix dari seluruh data uji dengan menggunakan nilai sebenarnya dan prediksi yang telah digabungkan pada baris sebelumnya.
4. Baris 7 digunakan untuk mengubah confusion matrix menjadi dataframe dengan menggunakan library pandas. Kolom diisi dengan "Predicted Ekstra Paru" dan "Predicted Paru", sedangkan barisnya diisi dengan "Actual Ekstra Paru" dan "Actual Paru".

4.3 Hasil Implementasi

Hasil implementasi dari penelitian ini, dibagi menjadi dua macam. Pertama, penelitian ini mengerjakan uji coba pada tiap skenario dan dijelaskan pada sub bab hasil implementasi skenario. Kemudian, untuk hasil implementasi tiap skenario tersebut ditampilkan menggunakan *user interface*, yang dijelaskan pada sub bab hasil implementasi *user interface*.

4.3.1 Hasil Implementasi Skenario 1

Skenario 1 melakukan pengujian dengan membandingkan nilai *fitness* pada *GA* untuk mencari nilai *crossover rate* terbaik dari varian *crossover rate* bernilai 0.1 hingga 1.0. Dalam menentukan varian terbaik dapat dilakukan dengan mempertimbangkan beberapa hal, seperti *best fitness* (nilai *fitness* terkecil), nilai konvergensi pada generasi dan *time required* paling singkat [32]. Berikut adalah hasil uji coba skenario 2 yang ditampilkan pada Tabel 4.2.

Tabel 4.2 Hasil Uji Coba Skenario 1

Varian Parameter				**Hasil Uji Coba**			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.1	0.01	20	200	46	0.0033	Ke - 13	22.05
0.2	0.01	20	200	3	0.0011	Ke - 18	26.66
0.3	0.01	20	200	4	0.0021	Ke - 14	24.99

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.4	0.01	20	200	3	0.0011	Ke - 20	31.12
0.5	0.01	20	200	4	0.0021	Ke - 15	24.30
0.6	0.01	20	200	47	0.0035	Ke - 12	21.47
0.7	0.01	20	200	3	0.0011	Ke - 14	22.91
0.8	0.01	20	200	3	0.0011	Ke - 14	24.99
0.9	0.01	20	200	3	0.0011	Ke - 18	31.78
1.0	0.01	20	200	3	0.0011	Ke - 15	27.30

Berdasarkan tabel di atas, varian *Cr* dengan nilai terbaik adalah yang memiliki *fitness score* terkecil dan konvergensi generasi tercepat dengan waktu yang paling singkat. Dari tabel tersebut, varian *Cr* dengan *fitness score* terkecil adalah 0.0011, yang ditemukan pada varian *Cr* dengan nilai 0.2, 0.4, 0.7, 0.8, 0.9, dan 1.0. Namun, jika dilihat dari konvergensi generasi dan waktu, varian *Cr* dengan nilai terbaik adalah yang memiliki konvergensi generasi tercepat dan waktu paling singkat, yaitu varian *Cr* dengan nilai 0.7. Varian *Cr* dengan nilai 0.7 memiliki *fitness score* terkecil yaitu 0.0011, konvergensi generasi ke-14, dan waktu yang dibutuhkan 22.91 detik. Oleh karena itu, varian *Cr* dengan nilai 0.7 dapat dianggap sebagai varian terbaik dalam hal ini.

4.3.2 Hasil Implementasi Skenario 2

Skenario 2 melakukan pengujian dengan membandingkan nilai *fitness* pada *GA* untuk mencari nilai *mutation rate* terbaik dari varian *mutation rate* bernilai 0.01 hingga 0.1. Dalam menentukan varian terbaik dapat dilakukan dengan mempertimbangkan beberapa hal, seperti *best fitness* (nilai *fitness* terkecil), nilai konvergensi pada generasi dan *time required* paling singkat [32]. Berikut adalah hasil uji coba skenario 3 yang ditampilkan pada Tabel 4.3.

Tabel 4.3 Hasil Uji Coba Skenario 2

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.7	0.01	20	200	3	0.0011	Ke - 14	32.5
0.7	0.02	20	200	3	0.0011	Ke - 20	37.64
0.7	0.03	20	200	4	0.0021	Ke - 20	35.43
0.7	0.04	20	200	3	0.0011	Ke - 14	26.00

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.7	0.05	20	200	3	0.0011	Ke - 18	31.94
0.7	0.06	20	200	3	0.0011	Ke - 20	37.29
0.7	0.07	20	200	3	0.0011	Ke - 20	38.76
0.7	0.08	20	200	3	0.0011	Ke - 15	29.85
0.7	0.09	20	200	3	0.0011	Ke - 15	30.03
0.7	0.10	20	200	3	0.0011	Ke - 14	29.43

Berdasarkan tabel di atas, varian *Mr* dengan nilai terbaik adalah yang memiliki *fitness score* terkecil dan konvergensi generasi tercepat dengan waktu yang paling singkat. Dari tabel tersebut, varian *Mr* dengan *fitness score* terkecil adalah 0.0011, yang ditemukan pada varian *Mr* dengan nilai 0.01, 0.02, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, dan 0.10. Namun, jika dilihat dari konvergensi generasi dan waktu, varian *Mr* dengan nilai terbaik adalah yang memiliki konvergensi generasi tercepat dan waktu paling singkat, yaitu varian *Mr* dengan nilai 0.04. Varian *Mr* dengan nilai 0.04 memiliki *fitness score* terkecil yaitu 0.0011, konvergensi generasi ke-14, dan waktu yang dibutuhkan 26.00 detik. Oleh karena itu, varian *Mr* dengan nilai 0.04 dapat dianggap sebagai varian terbaik dalam hal ini.

4.3.3 Hasil Implementasi Skenario 3

Skenario 3 melakukan pengujian dengan membandingkan nilai *fitness* pada *GA* untuk mencari nilai *population size* terbaik dari varian *population size* bernilai 20, 40, 60, 80, 100, 120, 140, 160, 180, 200. Dalam menentukan varian terbaik dapat dilakukan dengan mempertimbangkan beberapa hal, seperti *best fitness* (nilai *fitness* terkecil), nilai konvergensi pada generasi dan *time required* paling singkat [32]. Berikut adalah hasil uji coba dari skenario 4 yang ditampilkan pada Tabel 4.4.

Tabel 4.4 Hasil Uji Coba Skenario 3

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.7	0.04	20	200	3	0.0011	Ke - 12	22.11
0.7	0.04	40	200	3	0.0011	Ke - 15	48.87
0.7	0.04	60	200	3	0.0011	Ke - 13	65.37
0.7	0.04	80	200	3	0.0011	Ke - 13	85.62

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.7	0.04	100	200	3	0.0011	Ke - 12	99.43
0.7	0.04	120	200	3	0.0011	Ke - 11	110.19
0.7	0.04	140	200	3	0.0011	Ke - 16	196.23
0.7	0.04	160	200	3	0.0011	Ke - 11	149.16
0.7	0.04	180	200	3	0.0011	Ke - 13	195.92
0.7	0.04	200	200	3	0.0011	Ke - 11	189.26

Berdasarkan tabel di atas, varian *Pop Size* dengan nilai terbaik adalah yang memiliki *fitness score* terkecil dan konvergensi generasi tercepat dengan waktu yang paling singkat. Dari tabel tersebut, seluruh varian *Pop Size* memiliki nilai *fitness score* 0.0011. Namun, jika dilihat dari konvergensi generasi dan waktu, varian *Pop Size* dengan nilai terbaik adalah yang memiliki konvergensi generasi tercepat dan waktu paling singkat, yaitu varian *Pop Size* dengan nilai 120. Varian *Pop Size* dengan nilai 120 memiliki *fitness score* terkecil yaitu 0.0011, konvergensi generasi ke-11, dan waktu yang dibutuhkan 110.19 detik. Oleh karena itu, varian *Pop Size* dengan nilai 120 dapat dianggap sebagai varian terbaik dalam hal ini.

4.3.4 Hasil Implementasi Skenario 4

Skenario 5 melakukan pengujian dengan membandingkan nilai *fitness* pada *GA* untuk mencari nilai *maximum generation* terbaik dari varian *maximum generation* bernilai 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000. Dalam menentukan varian terbaik dapat dilakukan dengan mempertimbangkan beberapa hal, seperti *best fitness* (nilai *fitness* terkecil), nilai konvergensi pada generasi dan *time required* paling singkat [32]. Berikut adalah hasil uji coba skenario 5 yang ditampilkan pada Tabel 4.5.

Tabel 4.5 Hasil Uji Coba Skenario 4

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.7	0.04	120	200	3	0.0011	Ke - 11	120.05
0.7	0.04	120	400	3	0.0011	Ke - 11	101.44
0.7	0.04	120	600	3	0.0011	Ke - 11	100.40

Varian Parameter				Hasil Uji Coba			
Cr	***Mr***	***Pop Size***	***Max Generation***	***Best Individu***	***Fitness Score***	**Konvergensi Generasi**	***Time ('s)***
0.7	0.04	120	800	3	0.0011	Ke - 11	104.17
0.7	0.04	120	1000	3	0.0011	Ke - 11	107.90
0.7	0.04	120	1200	3	0.0011	Ke - 11	102.06
0.7	0.04	120	1400	3	0.0011	Ke - 11	96.18
0.7	0.04	120	1600	3	0.0011	Ke - 11	92.72
0.7	0.04	120	1800	3	0.0011	Ke - 11	103.79
0.7	0.04	120	2000	3	0.0011	Ke - 11	101.07

Berdasarkan tabel di atas, varian *Max Generation* dengan nilai terbaik adalah yang memiliki *fitness score* terkecil dan konvergensi generasi tercepat dengan waktu yang paling singkat. Dari tabel tersebut, seluruh varian *Max Generation* memiliki nilai *fitness score* 0.0011. Namun, jika dilihat dari konvergensi generasi dan waktu, varian *Max Generation* dengan nilai terbaik adalah yang memiliki konvergensi generasi tercepat dan waktu paling singkat, yaitu varian *Max Generation* dengan nilai 1600. Varian *Max Generation* dengan nilai 1600 memiliki *fitness score* terkecil yaitu 0.0011, konvergensi generasi ke-11, dan waktu yang dibutuhkan 92.72 detik. Oleh karena itu, varian *Max Generation* dengan nilai 1600 dapat dianggap sebagai varian terbaik dalam hal ini.

4.3.5 Hasil Implementasi Skenario 5

Skenario 5 melakukan uji coba imputasi *missing value* dengan nilai *k* yang didapatan dari *best individu* GA pada skenario sebelumnya. Berdasarkan skenario sebelumnya, menghasilkan *best individu* atau solusi nilai *k* terbaik pada *k*-NN untuk imputasi, yaitu *k* = 3. Nilai *k* tersebut digunakan untuk melakukan imputasi *missing value* pada *k*-NN.

4.3.6 Hasil Implementasi Skenario 6

Skenario 6 melakukan uji coba imputasi *missing value* dengan variasi nilai *k* = 3, 5, 7, 9, 11, 13, dan 15. Contoh hasil uji coba skenario 6 berupa data imputasi, dapat dilihat pada Tabel 4.6, Lampiran 1.

4.3.7 Hasil Implementasi Skenario 7

Pada skenario 7, dilakukan ujicoba model LSTM dengan varian *neuron* pada *hidden layer* bernilai 6, 12, 18, 24, dan 30 menggunakan dataset yang telah melalui imputasi *k*-NN dan *k*-NN+GA.

Proses kompilasi dan proses pembelajaran terhadap *data training* dapat dilakukan setelah membuat struktur *model*. Setelah *model* hasil pembelajaran dihasilkan, selanjutnya dilakukan prediksi dan evaluasi pada *data testing* dengan menggunakan *model* hasil pembelajaran tersebut. Kemudian untuk melihat performa LSTM, dilakukan dengan menghitung rata – rata akurasi evaluasi LSTM pada seluruh *fold*.

4.3.7.1 LSTM Menggunakan Imputasi *k*-NN+GA, *k*=3

Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 3, yang dapat dilihat pada Tabel 4.7.

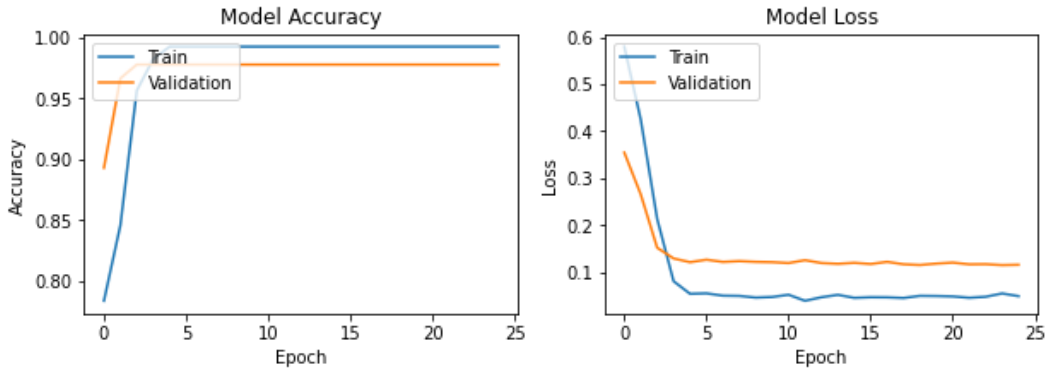
Tabel 4.7 Performa LSTM Menggunakan *k*-NN+GA, *k* = 3 pada Skenario 7.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	6	25	95,4661	0,158	98,5756	0,0815	349,33
2.	12	25	96,5599	0,1292	98,5756	0,0778	348,38
3.	18	25	96,8207	0,1207	98,5756	0,0771	347,96
4.	24	25	97,0075	0,1154	98,5756	0,0794	349,7
5.	30	25	97,0689	0,1137	98,5756	0,0775	353,12

Berdasarkan Tabel 4.7, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *neuron*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,0771 adalah varian dengan jumlah *neuron* sebanyak 18. Hal tersebut menunjukkan

bahwa varian dengan jumlah *neuron* 18 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 18 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.2.



Gambar 4.2 Hasil Pembelajaran LSTM Menggunakan *k*-NN+GA, *k* = 3pada Varian *Neuron* 18

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.8 berikut.

Tabel 4.8 Performa LSTM Menggunakan *k*-NN+GA, *k* = 3 pada Varian *Neuron* 18.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.8 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.2 LSTM Menggunakan Imputasi *k*-NN, *k*=3

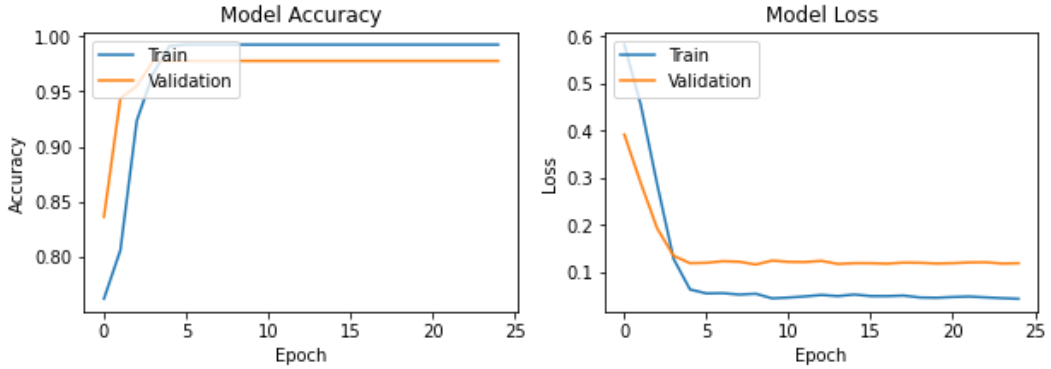
Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 3, yang dapat dilihat pada Tabel 4.9.

Tabel 4.9 Performa LSTM Menggunakan *k*-NN, *k* = 3 pada Skenario 7.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	6	25	95,3195	0,1631	98,5756	0,0793	190,11
2.	12	25	96,5626	0,1302	98,5756	0,0790	191,46
3.	18	25	96,7932	0,1213	98,5756	0,0784	194,35
4.	24	25	97,0233	0,1152	98,5756	0,0785	194,89
5.	30	25	97,0756	0,1141	98,5756	0,0788	191,97

Berdasarkan Tabel 4.9, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *neuron*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,0784 adalah varian dengan jumlah *neuron* sebanyak 18. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 18 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 18 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.3.



Gambar 4.3 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 3 pada Varian *Neuron* 18

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.10 berikut.

Tabel 4.10 Performa LSTM Menggunakan *k*-NN, *k* = 3 pada Varian *Neuron* 18.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.10 dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.3 LSTM Menggunakan Imputasi *k*-NN, *k*=5

Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 5, yang dapat dilihat pada Tabel 4.11.

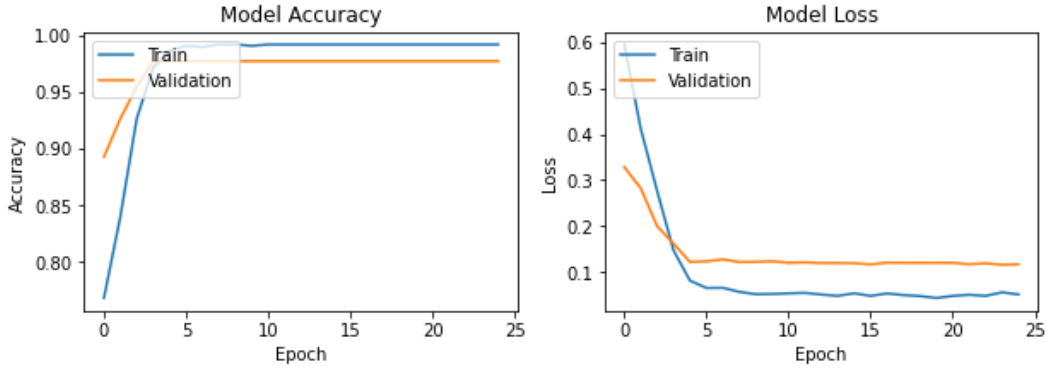
Tabel 4.11 Performa LSTM Menggunakan *k*-NN, *k* = 5 pada Skenario 7.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	6	25	95,5957	0,157	98,5756	0,0816	200,93
2.	12	25	96,5351	0,1297	98,5756	0,0779	203,6
3.	18	25	96,8009	0,1218	98,5756	0,0782	203,98
4.	24	25	96,9236	0,1172	98,5756	0,0788	204,00
5.	30	25	97,0324	0,1138	98,5756	0,078	198,35

Berdasarkan Tabel 4.11, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *neuron*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan

mempertimbangkan beberapa hal, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,0779 adalah varian dengan jumlah *neuron* sebanyak 12. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 12 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 12 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.4.



Gambar 4.4 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 5 pada Varian *Neuron* 12

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.12 berikut.

Tabel 4.12 Performa LSTM Menggunakan *k*-NN, *k* = 5 pada Varian *Neuron* 12.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.12 dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.4 LSTM Menggunakan Imputasi *k*-NN, *k*=7

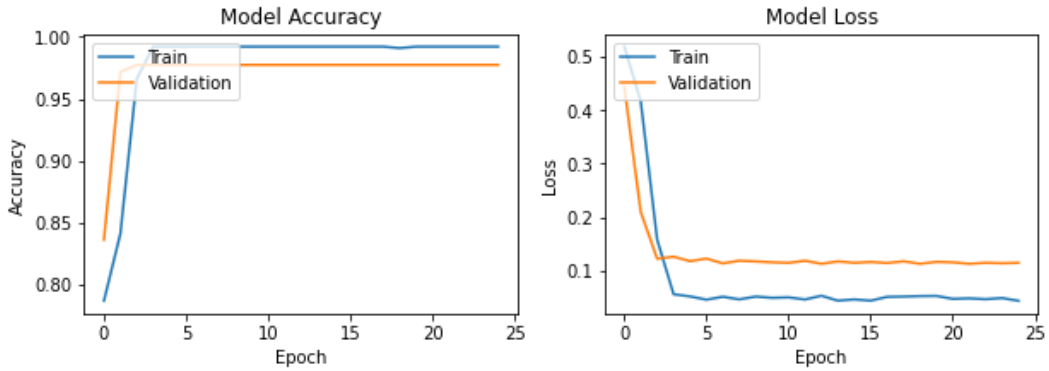
Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 7, yang dapat dilihat pada Tabel 4.13.

Tabel 4.13 Performa LSTM Menggunakan *k*-NN, *k* = 7 pada Skenario 7.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	6	25	95,5352	0,1584	98,5756	0,0808	196,86
2.	12	25	96,5563	0,1283	98,5756	0,078	197,59
3.	18	25	96,8139	0,1209	98,5756	0,0795	201,87
4.	24	25	96,9335	0,1179	98,5756	0,0783	199,8
5.	30	25	97,0905	0,1135	98,5756	0,077	195,84

Berdasarkan tabel Tabel 4.13, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *neuron*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan beberapa hal, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,077 adalah varian dengan jumlah *neuron* sebanyak 30. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 30 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 30 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.5.



Gambar 4.5 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 7 pada Varian *Neuron* 30

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.14 berikut.

Tabel 4.14 Performa LSTM Menggunakan *k*-NN, *k* = 7 pada Varian *Neuron* 30.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.14 dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.5 LSTM Menggunakan Imputasi *k*-NN, *k*=9

Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 9, yang dapat dilihat pada Tabel 4.15.

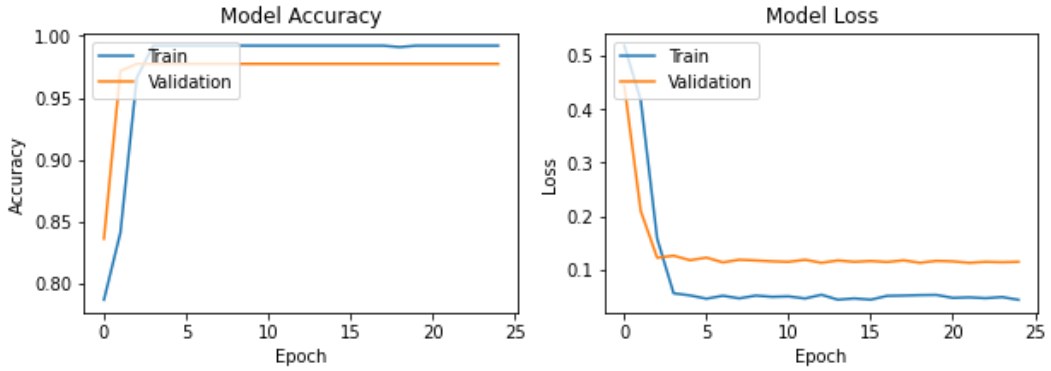
Tabel 4.15 Performa LSTM Menggunakan *k*-NN, *k* = 9 pada Skenario 7.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	6	25	95,5447	0,1596	98,5756	0,0785	202,79

No.	Neuron	Epoch	Data training		Data testing		Time (Second)
			Accuracy (%)	Loss	Accuracy (%)	Loss	
2.	12	25	96,6244	0,1288	98,5756	0,0801	194,75
3.	18	25	96,8703	0,1198	98,5756	0,0791	196,55
4.	24	25	97,0143	0,1155	98,5756	0,0785	200,09
5.	30	25	97,0756	0,1134	98,5756	0,0774	193,72

Berdasarkan Tabel 4.15, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *neuron*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan beberapa hal, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,0774 adalah varian dengan jumlah *neuron* sebanyak 30. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 30 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 30 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.6.



Gambar 4.6 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 9 pada Varian *Neuron* 30

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.16 berikut.

Tabel 4.16 Performa LSTM Menggunakan *k*-NN, *k* = 9 pada Varian *Neuron* 30.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.16dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.6 LSTM Menggunakan Imputasi *k*-NN, *k*=11

Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 11, yang dapat dilihat pada Tabel 4.17.

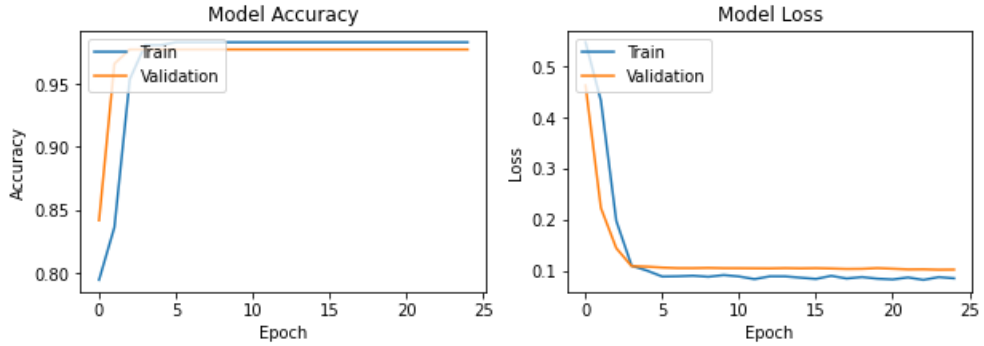
Tabel 4.17 Performa LSTM Menggunakan *k*-NN, *k* = 11 pada Skenario 7.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	6	25	95,3926	0,1626	98,4735	0,0855	194,89
2.	12	25	96,5148	0,1321	98,4735	0,0879	193,08
3.	18	25	96,6925	0,1255	98,4735	0,0864	193,09
4.	24	25	96,9015	0,1197	98.3725	0,0854	195,11
5.	30	25	96,9845	0,1177	98,4735	0,0819	192,48

Berdasarkan Tabel 4.17, terlihat bahwa nilai *accuracy testing* tertinggi selalu stabil pada 98,4735% untuk varian *neuron* 6,12,18, dan 30. nilai *accuracy testing* terendah pada 98.3725 untuk varian *neuron* 24. Hal ini menunjukkan bahwa model hampir memiliki performa yang baik dan stabil dalam

mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,0819 adalah varian dengan jumlah *neuron* sebanyak 30. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 30 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 30 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.7.



Gambar 4.7 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 11 pada Varian *Neuron* 30

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.18 berikut.

Tabel 4.18 Performa LSTM Menggunakan *k*-NN, *k* = 11 pada Varian *Neuron* 30.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.18 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.7 LSTM Menggunakan Imputasi *k*-NN, *k*=13

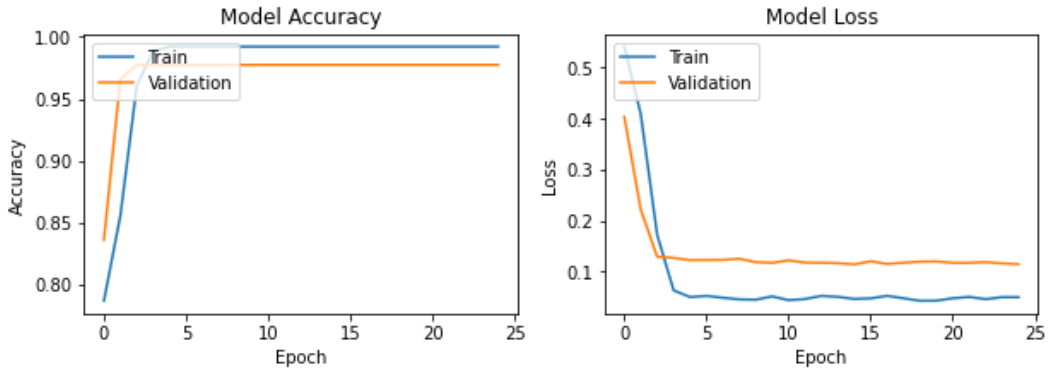
Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 13, yang dapat dilihat pada Tabel 4.19.

Tabel 4.19 Performa LSTM Menggunakan *k*-NN, *k* = 13 pada Skenario 7.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	6	25	95,4074	0,1628	98,4735	0,0872	200,21
2.	12	25	96,5161	0,1317	98,4735	0,0865	195,66
3.	18	25	96,7179	0,1248	98,4735	0,0836	197,00
4.	24	25	96,887	0,1203	98,4735	0,083	195,89
5.	30	25	96,9592	0,1177	98,4735	0,0855	199,29

Berdasarkan Tabel 4.19, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,4735% untuk semua varian *neuron*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,083 adalah varian dengan jumlah *neuron* sebanyak 24. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 24 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 24 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.8.



Gambar 4.8 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 13 pada Varian *Neuron* 24

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.20 berikut.

Tabel 4.20 Performa LSTM Menggunakan *k*-NN, *k* = 13 pada Varian *Neuron* 24.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.20 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.8 LSTM Menggunakan Imputasi *k*-NN, *k*=15

Berikut adalah seluruh hasil performa LSTM dengan varian *neuron* 6, 12, 24, 18, dan 30 menggunakan data imputasi *k*-NN dengan nilai *k* = 15, yang dapat dilihat pada Tabel 4.21.

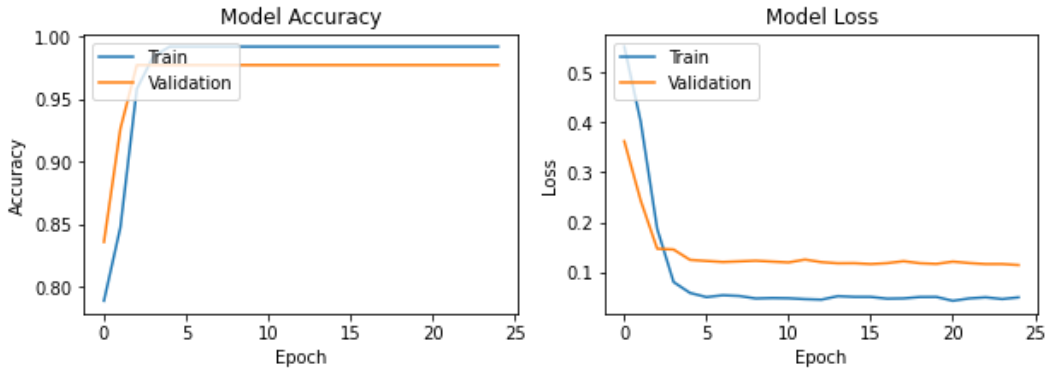
Tabel 4.21 Performa LSTM Menggunakan *k*-NN, *k* = 15 pada Skenario 7.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	6	25	95,3619	0,1644	98,4735	0,0889	203,05

No.	Neuron	Epoch	Data training		Data testing		Time (Second)
			Accuracy (%)	Loss	Accuracy (%)	Loss	
2.	12	25	96,5292	0,131	98,4735	0,0865	199,42
3.	18	25	96,6596	0,1257	98,4735	0,0841	201,56
4.	24	25	96,9172	0,1198	98,4735	0,0851	200,69
5.	30	25	96,9624	0,1177	98,3725	0,0860	197,01

Berdasarkan Tabel 4.21, terlihat bahwa nilai *accuracy testing* tertinggi selalu stabil pada 98,4735% untuk varian *neuron* 6,12,18, dan 24. nilai *accuracy testing* terendah pada 98.3725 untuk varian *neuron* 30. Hal ini menunjukkan bahwa model hampir memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *neuron* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *neuron* dengan nilai *loss testing* terkecil senilai 0,0841 adalah varian dengan jumlah *neuron* sebanyak 18. Hal tersebut menunjukkan bahwa varian dengan jumlah *neuron* 18 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan 18 *neuron*, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.9.



Gambar 4.9 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 15 pada Varian *Neuron* 18

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.22 berikut.

Tabel 4.22 Performa LSTM Menggunakan *k*-NN, *k* = 15 pada Varian *Neuron* 18.

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.22 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.7.9 Hasil Perbandingan Akurasi pada Skenario 7

Setelah dilakukan pembelajaran dan evaluasi dengan varian *neuron* menggunakan dataset yang telah melalui imputasi *k*-NN dan *k*-NN+GA, diperoleh perbandingan akurasi yang ditampilkan pada Tabel 4.23 sebagai berikut.

Tabel 4.23 Perbandingan Akurasi Skenario 7.

No.	**Klasifikasi LSTM**	**Evaluasi *Testing***		***Time (Second)***
		Accuracy	***Loss***	
1.	*k*-NN, *k* = 3	98,5756	0,0784	194,35
2.	*k*-NN, *k* = 5	98,5756	0,0779	203,6
3.	*k*-NN, *k* = 7	98,5756	0,0770	195,84
4.	*k*-NN, *k* = 9	98,5756	0,0774	193,72
5.	*k*-NN, *k* = 11	98,4735	0,0819	192,48
6.	*k*-NN, *k* = 13	98,4735	0,0830	195,89
7.	*k*-NN, *k* = 15	98,4735	0,0841	201,56
Total Time (Second)				2754,88
8.	*k*-NN + GA, *k* = 3	98,5756	0,0771	347,96

Berdasarkan Tabel 4.23, terlihat bahwa nilai *accuracy testing* tertinggi selalu stabil pada 98,5756% untuk varian klasifikasi LSTM dengan *k-NN* yang bernilai *k* = 3, *k-NN* yang bernilai *k* = 5, *k-NN* yang bernilai *k* = 7, *k-NN* yang bernilai *k* = 9, dan *k-NN*+GA yang bernilai *k* = 3. Sedangkan akurasi terendah selalu stabil pada 98,4735% untuk varian klasifikasi LSTM dengan *k-NN* yang bernilai *k* = 11, *k-NN* yang bernilai *k* = 13, dan *k-NN* yang bernilai *k* = 15. Hal tersebut menunjukkan bahwa pada skenario 7, klasifikasi LSTM dengan *k-NN*+GA berhasil menghasilkan akurasi tertinggi. Meskipun mencapai akurasi tertinggi, namun klasifikasi LSTM dengan *k-NN*+GA memiliki waktu komputasi yang lebih lama jika dibandingkan dengan varian klasifikasi LSTM lainnya, karena klasifikasi LSTM dengan *k-NN*+GA memiliki waktu komputasi selama 347,96 *second* sedangkan klasifikasi LSTM dengan *k-NN* memerlukan waktu dengan total 2754,88 second untuk mencari akurasi terbaik dari k = 3, 5, 7, 9, 11, 13, 15.

4.3.8 Hasil Implementasi Skenario 8

Setelah dilakukan ujicoba pada skenario7, didapatkan masing – masing nilai *neuron* terbaik untuk setiap datasetnya. Kemudian pada skenario 8, dilakukan ujicoba model LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan dataset yang telah melalui imputasi *k*-NN dan *k*-NN+GA dan masing- masing nilai *neuron* tersebut.

Proses kompilasi dan proses pembelajaran terhadap *data training* dapat dilakukan setelah membuat struktur *model*. Setelah *model* hasil pembelajaran dihasilkan, selanjutnya dilakukan prediksi dan evaluasi pada *data testing* dengan menggunakan *model* hasil pembelajaran tersebut. Kemudian untuk melihat performa LSTM, dilakukan dengan menghitung rata – rata akurasi evaluasi LSTM pada seluruh *fold*.

4.3.8.1 LSTM Menggunakan Imputasi *k*-NN+GA, *k*=3

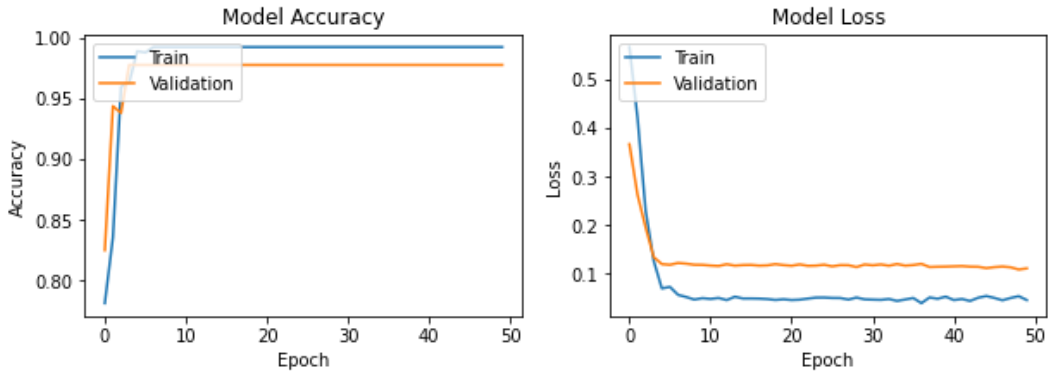
Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 5, yang dapat dilihat pada Tabel 4.24.

Tabel 4.24 Performa LSTM Menggunakan *k*-NN+GA, *k* = 3 pada Skenario 8.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	18	25	96,8433	0,1207	98,5756	0,0802	503,83
2.	18	50	97,6995	0,0975	98,5756	0,0777	553,89
3.	18	75	97,9793	0,0892	98,5756	0,0834	602,57
4.	18	100	98,1182	0,0846	98,5756	0,0796	649,48
5.	18	125	98,2115	0,0817	98,5756	0,0788	706,71
6.	18	150	98,2748	0,0792	98,5756	0,0795	748,95
7.	18	175	98,3155	0,0778	98,5756	0,0812	799,25
8.	18	200	98,3547	0,0759	98,5756	0,0836	848,64

Berdasarkan Tabel 4.24, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *epoch*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *epoch* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *epoch* dengan nilai *loss testing* terkecil senilai 0,0777 adalah varian dengan jumlah *epoch* sebanyak 50. Hal tersebut menunjukkan bahwa varian dengan jumlah *epoch* 50 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan *epoch* 50, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.10.



Gambar 4.10 Hasil Pembelajaran LSTM Menggunakan *k*-NN+GA, *k* = 3 pada Varian 50 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.25 berikut.

Tabel 4.25 Performa LSTM Menggunakan *k*-NN+GA, *k* = 3 pada Varian *Epoch 50*

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan tabel Tabel 4.25 dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.2 LSTM Menggunakan Imputasi *k*-NN, *k*=3

Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 3, yang dapat dilihat pada Tabel 4.26.

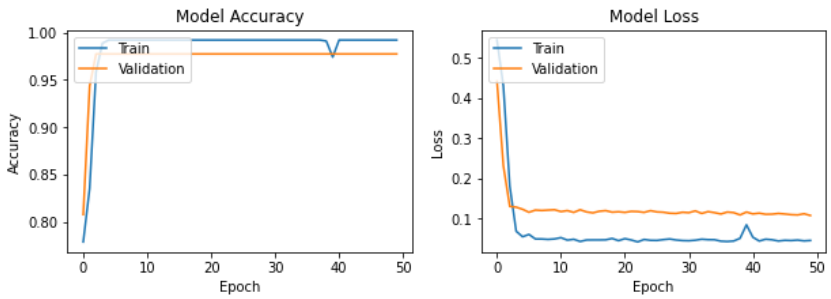
Tabel 4.26 Performa LSTM Menggunakan *k*-NN, *k* = 3 pada Skenario 8.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	

No.	Neuron	Epoch	Data training		Data testing		Time (Second)
			Accuracy (%)	Loss	Accuracy (%)	Loss	
1.	18	25	97,0116	0,1159	98,5756	0,0786	364,27
2.	18	50	97,7576	0,0949	98,5756	0,0757	422,19
3.	18	75	98,0041	0,0877	98,5756	0,0768	473,5
4.	18	100	98,1494	0,0832	98,5756	0,0785	519,45
5.	18	125	98,2536	0,0795	98,5756	0,0769	568,56
6.	18	150	98,3062	0,0777	98,4735	0,0870	625,29
7.	18	175	98,3323	0,0767	98,4735	0,0820	686,74
8.	18	200	98,3667	0,0752	98,5756	0,0805	751,60

Berdasarkan Tabel 4.26, terlihat bahwa nilai *accuracy testing* tertinggi selalu stabil pada 98,5756% untuk varian *epoch* 25, 50, 75, 100, 125, dan 200. Kemudian untuk *accuracy testing* terendah selalu stabil pada 98,4735% untuk varian *epoch* 150 dan 175. Kemudian, untuk menentukan varian *epoch* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *epoch* dengan nilai accuracy test tertinggi dan nilai *loss testing* terkecil senilai 0,0757 adalah varian dengan jumlah *epoch* sebanyak 50. Hal tersebut menunjukkan bahwa varian dengan jumlah *epoch* 50 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan *epoch* 50, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.11.



Gambar 4.11 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 3 pada Varian 50 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.27 berikut.

Tabel 4.27 Performa LSTM Menggunakan *k*-NN, *k* = 3 pada Varian *Epoch* 50

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.27 dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.3 LSTM Menggunakan Imputasi *k*-NN, *k*=5

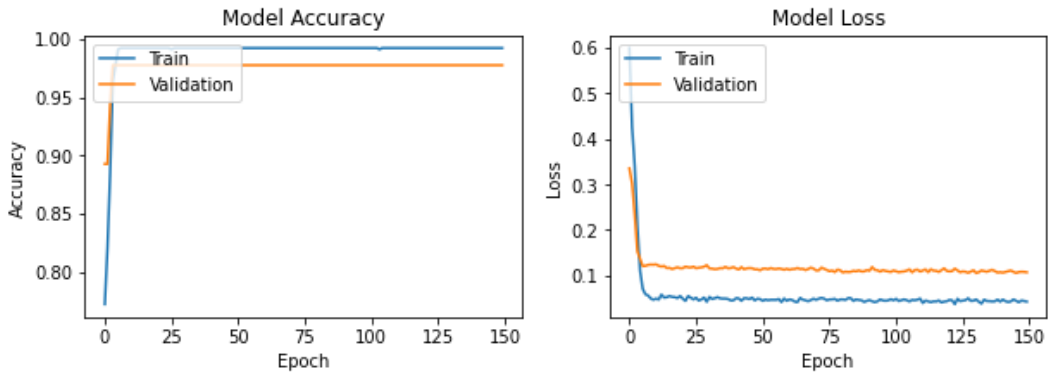
Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 5, yang dapat dilihat pada Tabel 4.28.

Tabel 4.28 Performa LSTM Menggunakan *k*-NN, *k* = 5 pada Skenario 8.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	12	25	96,5789	0,1288	98,5756	0,0788	359,43
2.	12	50	97,5566	0,1036	98,5756	0,0799	407,02
3.	12	75	97,8865	0,0933	98,5756	0,0792	455,84
4.	12	100	98,0452	0,0882	98,5756	0,0794	489,01
5.	12	125	98,1462	0,0850	98,5756	0,0790	540,93
6.	12	150	98,2111	0,0826	98,5756	0,0773	586,23
7.	12	175	98,2808	0,0802	98,5756	0,0818	634,88
8.	12	200	98,3242	0,0785	98,5756	0,0788	686,9

Berdasarkan Tabel 4.28, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk semua varian *epoch*. Hal ini menunjukkan bahwa model memiliki performa yang baik dan stabil dalam mengklasifikasikan *data testing*. Kemudian, untuk menentukan varian *epoch* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *epoch* dengan nilai *loss testing* terkecil senilai 0,0773 adalah varian dengan jumlah *epoch* sebanyak 150. Hal tersebut menunjukkan bahwa varian dengan jumlah *epoch* 150 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan *epoch* 150, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.12.



Gambar 4.12 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 5 pada Varian *150 Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.29 berikut.

Tabel 4.29 Performa LSTM Menggunakan *k*-NN, *k* = 5 pada Varian *Epoch 150*

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.29 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.4 LSTM Menggunakan Imputasi *k*-NN, *k*=7

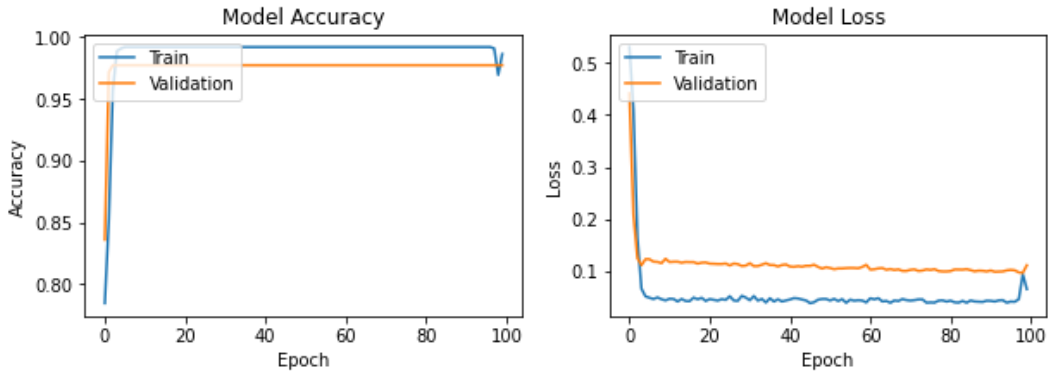
Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 7, yang dapat dilihat pada Tabel 4.30.

Tabel 4.30 Performa LSTM Menggunakan *k*-NN, *k* = 7 pada Skenario 8.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	30	25	97,0883	0,1130	98,5756	0,0783	357,87
2.	30	50	97,8080	0,0933	98,5756	0,0785	411,82
3.	30	75	98,0424	0,0863	98,5756	0,0793	464,63
4.	30	100	98,1654	0,0822	98,5756	0,0766	510,8
5.	30	125	98,2558	0,0787	98,4735	0,0805	573,52
6.	30	150	98,3003	0,0772	98,0653	0,0892	659,2
7.	30	175	98,3535	0,0753	98,4735	0,0878	691,06
8.	30	200	98,3627	0,0745	98,5756	0,0826	749,51

Berdasarkan Tabel 4.30, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk varian *epoch* 25, 50, 75, 100,dan 200. Kemudian, untuk menentukan varian *epoch* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *epoch* dengan nilai *accuracy testing* tertinggi dan *loss testing* terkecil senilai 0,0766 adalah varian dengan jumlah *epoch* sebanyak 100. Hal tersebut menunjukkan bahwa varian dengan jumlah *epoch* 100 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan *epoch* 100, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.13.



Gambar 4.13 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 7 pada Varian 100 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.31 berikut.

Tabel 4.31 Performa LSTM Menggunakan *k*-NN, *k* = 7 pada Varian *Epoch* 100

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.31 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.5 LSTM Menggunakan Imputasi *k*-NN, *k*=9

Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 9, yang dapat dilihat pada Tabel 4.32.

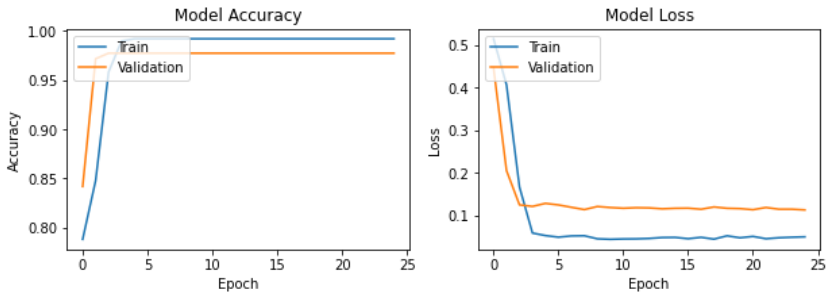
Tabel 4.32 Performa LSTM Menggunakan *k*-NN, *k* = 9 pada Skenario 8.

No.	***Neuron***	***Epoch***	***Data training***	***Data testing***	***Time***

			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	*(Second)*
1.	30	25	97,0576	0,1134	98,5756	0,0770	362,12
2.	30	50	97,7910	0,0938	98,5756	0,0783	411,05
3.	30	75	98,0450	0,0861	98,5756	0,0798	466,03
4.	30	100	98,1663	0,0825	98,5756	0,0843	508,91
5.	30	125	98,2333	0,0795	98,4735	0,0842	569,79
6.	30	150	98,3078	0,0772	98,5756	0,0846	622,73
7.	30	175	98,3404	0,0757	98,3725	0,0859	682,62
8.	30	200	98,3744	0,0741	98,5756	0,0861	775,77

Berdasarkan Tabel 4.32, terlihat bahwa nilai *accuracy testing* selalu stabil pada 98,5756% untuk varian *epoch* 25, 50, 75, 100, 150, dan 200. Kemudian, untuk menentukan varian *epoch* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *epoch* dengan nilai *accuracy testing* tertinggi dan *loss testing* terkecil senilai 0,0770 adalah varian dengan jumlah *epoch* sebanyak 25. Hal tersebut menunjukkan bahwa varian dengan jumlah *epoch* 25 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan *epoch* 25, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.14.



Gambar 4.14 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 9 pada Varian 25 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.33 berikut.

Tabel 4.33 Performa LSTM Menggunakan *k*-NN, *k* = 9 pada Varian *Epoch* 25

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	257	14
***Actual* Paru**	0	714

Berdasarkan Tabel 4.33 dijelaskan bahwa, 257 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 14 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.6 LSTM Menggunakan Imputasi *k*-NN, *k*=11

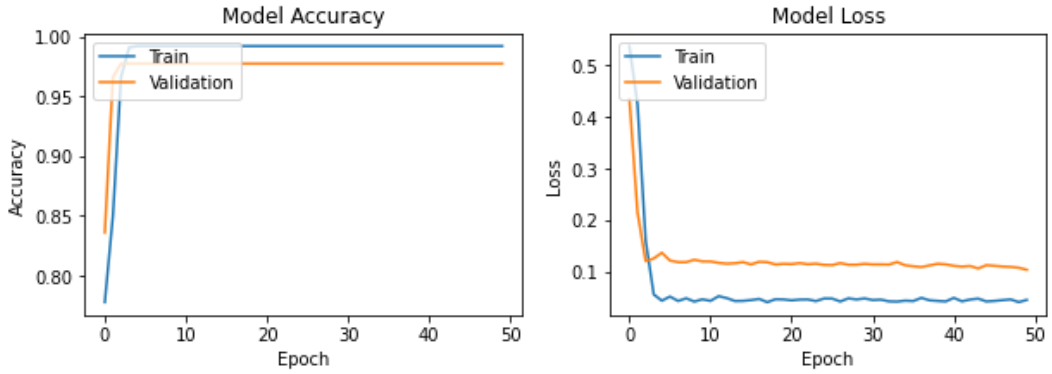
Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 11, yang dapat dilihat pada Tabel 4.34.

Tabel 4.34 Performa LSTM Menggunakan *k*-NN, *k* = 11 pada Skenario 8.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	30	25	96,9845	0,1172	98,3725	0,0854	192,48
2.	30	50	97,6943	0,0970	98,4735	0,0855	571,27
3.	30	75	97,9542	0,0890	98,3725	0,0870	459,5
4.	30	100	98,0852	0,0838	98,3725	0,0953	501,61
5.	30	125	98,1591	0,0815	98,2705	0,1032	565,1
6.	30	150	98,1979	0,0795	98,3725	0,1007	635,25
7.	30	175	98,2347	0,0779	98,2705	0,0930	708,13
8.	30	200	98,2674	0,0764	98,3725	0,0979	730,94

Berdasarkan Tabel 4.34, terlihat bahwa varian *epoch* dengan nilai *accuracy testing* tertinggi dimiliki oleh varian *epoch* 50 dengan nilai 98.47%. Kemudian, untuk varian *epoch* lainnya memiliki nilai accuracy testing yang lebih rendah jika dibandingkan varian *epoch* 50.

Pada performa LSTM menggunakan *epoch* 50, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.15.



Gambar 4.15 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 11 pada Varian 50 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.35 berikut.

Tabel 4.35 Performa LSTM Menggunakan *k*-NN, *k* = 11 pada Varian *Epoch 50*

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.35 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.7 LSTM Menggunakan Imputasi *k*-NN, *k*=13

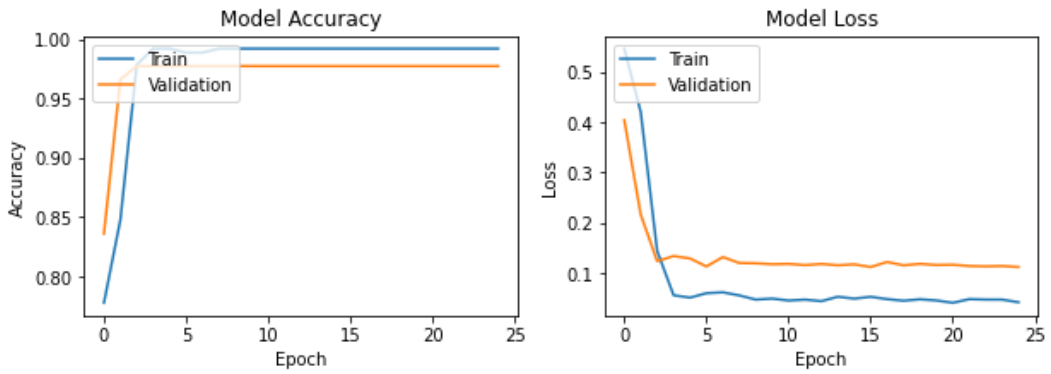
Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 5, yang dapat dilihat pada Tabel 4.36.

Tabel 4.36 Performa LSTM Menggunakan *k*-NN, *k* = 13 pada Skenario 8.

No.	*Neuron*	*Epoch*	*Data training*		*Data testing*		*Time (Second)*
			Accuracy (%)	*Loss*	*Accuracy (%)*	*Loss*	
1.	24	25	96,9010	0,1198	98,4735	0,0852	353,65
2.	24	50	97,6320	0,0997	98,3725	0,0875	408,41
3.	24	75	97,9303	0,0906	98,3725	0,0933	460,35
4.	24	100	98,0485	0,0859	98,3725	0,0967	495,92
5.	24	125	98,1444	0,0824	98,3725	0,0990	550,44
6.	24	150	98,1967	0,0802	98,2705	0,0879	608,54
7.	24	175	98,2364	0,0786	98,3725	0,1036	677,6
8.	24	200	98,2668	0,0768	98,3725	0,1015	740,9

Berdasarkan Tabel 4.36, terlihat bahwa varian *epoch* dengan nilai *accuracy testing* tertinggi dimiliki oleh varian *epoch* 25 dengan nilai 98,4735%. Kemudian, untuk varian *epoch* lainnya memiliki nilai accuracy testing yang lebih rendah jika dibandingkan varian *epoch* 25.

Pada performa LSTM menggunakan *epoch* 25, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.16.



Gambar 4.16 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 13 pada Varian 25 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.37 berikut.

Tabel 4.37 Performa LSTM Menggunakan *k*-NN, *k* = 13 pada Varian *Epoch* 25

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.37 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.8 LSTM Menggunakan Imputasi *k*-NN, *k*=15

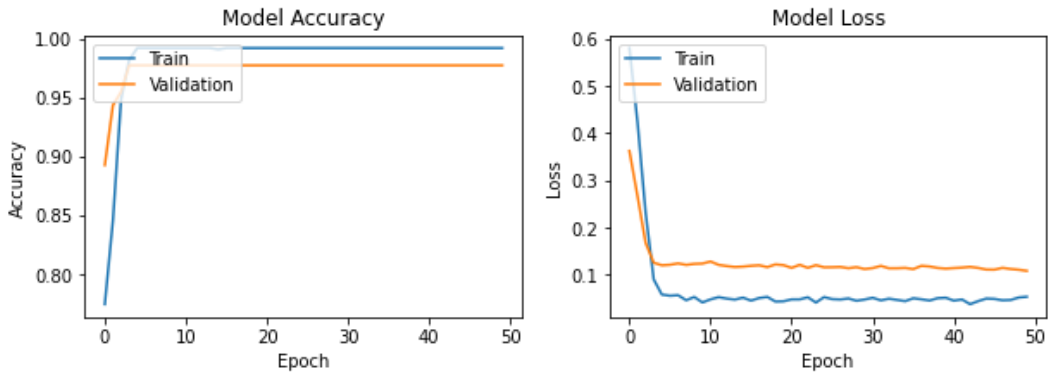
Berikut adalah hasil performa LSTM dengan varian *epoch* bernilai 25, 50, 75, 100, 125, 150, 175, dan 200 menggunakan data imputasi *k*-NN dengan nilai *k* = 5, yang dapat dilihat pada Tabel 4.38.

Tabel 4.38 Performa LSTM Menggunakan *k*-NN, *k* = 15 pada Skenario 8.

No.	***Neuron***	***Epoch***	***Data training***		***Data testing***		***Time (Second)***
			Accuracy (%)	***Loss***	***Accuracy (%)***	***Loss***	
1.	18	25	96,6614	0,1258	98,4735	0,0847	358,69
2.	18	50	97,5948	0,1014	98,4735	0,0821	408,75
3.	18	75	97,8627	0,0926	98,4735	0,0850	457,97
4.	18	100	98,0147	0,0873	98,3725	0,0856	493,97
5.	18	125	98,1082	0,0844	98,3725	0,0889	542,71
6.	18	150	98,1590	0,0822	98,3725	0,0943	592,19
7.	18	175	98,2052	0,0806	98,3725	0,0996	645,88
8.	18	200	98,2506	0,0786	98,3725	0,0934	718,22

Berdasarkan Tabel 4.38, terlihat bahwa nilai *accuracy testing* tertinggi selalu stabil pada 98,4735% untuk varian *epoch* 25, 50, dan 75. Kemudian untuk *accuracy testing* terendah selalu stabil pada 98,3725% untuk varian *epoch* 100, 125, 150, 175, dan 200. Kemudian, untuk menentukan varian *epoch* terbaik, dapat dilakukan dengan mempertimbangkan hal lain, seperti nilai *loss testing*. Nilai *loss* mengindikasikan seberapa besar *error* dalam melakukan klasifikasi data. Semakin kecil nilai *loss*, maka semakin baik performa model dalam melakukan klasifikasi. Berdasarkan hasil uji coba, varian *epoch* dengan nilai accuracy test tertinggi dan nilai *loss testing* terkecil senilai 0,0821 adalah varian dengan jumlah *epoch* sebanyak 50. Hal tersebut menunjukkan bahwa varian dengan jumlah *epoch* 50 memiliki performa terbaik dalam hal melakukan klasifikasi *data testing*.

Pada performa LSTM menggunakan *epoch* 50, hasil pembelajaran terbaik terdapat pada *fold* ke – 6. Berikut adalah hasil proses pembelajar *fold* ke – 6 yang digambarkan dalam bentuk grafik, Gambar 4.17.



Gambar 4.17 Hasil Pembelajaran LSTM Menggunakan *k*-NN, *k* = 15 pada Varian 50 *Epoch*

Setelah melakukan pembelajaran, model dari hasil pembelajaran tersebut dapat digunakan untuk melakukan prediksi. Hasil prediksi dengan menggunakan fungsi *confusion_matrix* dapat dilihat pada Tabel 4.39 berikut.

Tabel 4.39 Performa LSTM Menggunakan *k*-NN, *k* = 15 pada Varian *Epoch* ….

	***Predicted* Ekstra Paru**	***Predicted* Paru**
***Actual* Ekstra Paru**	256	15
***Actual* Paru**	0	714

Berdasarkan Tabel 4.39 dijelaskan bahwa, 256 pasien yang sebenarnya Ekstra Paru diidentifikasi dengan benar sebagai Ekstra Paru oleh model, 15 pasien yang sebenarnya Ekstra Paru diidentifikasi sebagai Paru oleh model, 0 pasien yang sebenarnya Paru diidentifikasi sebagai Ekstra Paru oleh model dan 714 pasien yang sebenarnya Paru diidentifikasi dengan benar sebagai Paru oleh model.

4.3.8.9 Hasil Perbandingan Akurasi pada Skenario 8

Setelah dilakukan pembelajaran dan evaluasi dengan varian *neuron* menggunakan dataset yang telah melalui imputasi *k*-NN dan *k*-NN+GA, diperoleh perbandingan akurasi yang ditampilkan pada Tabel 4.40 sebagai berikut.

Tabel 4.40 Perbandingan Akurasi Skenario 8.

No.	**Klasifikasi LSTM**	**Evaluasi *Testing***		***Time (Second)***
		Accuracy	***Loss***	
1.	*k*-NN, *k* = 3	98,5756	0,0757	789,83
2.	*k*-NN, *k* = 5	98,5756	0,0773	789,83
3.	*k*-NN, *k* = 7	98,5756	0,0766	706,64
4.	*k*-NN, *k* = 9	98,5756	0,0770	555,84
5.	*k*-NN, *k* = 11	98,4735	0,0855	763,75
6.	*k*-NN, *k* = 13	98,4735	0,0852	549,54
7.	*k*-NN, *k* = 15	98,4735	0,0821	610,31
Total Time (Second)				4765,74
8.	*k*-NN + GA, *k* = 3	98,5756	0,0777	901,85

Berdasarkan Tabel 4.40, terlihat bahwa nilai *accuracy testing* tertinggi selalu stabil pada 98,5756% untuk varian klasifikasi LSTM dengan *k-NN* yang bernilai *k* = 3, *k-NN* yang bernilai *k* = 5, *k-NN* yang bernilai *k* = 7, *k-NN* yang bernilai *k* = 9, dan *k-NN*+GA yang bernilai *k* = 3. Sedangkan akurasi terendah selalu stabil pada 98,4735% untuk varian klasifikasi LSTM dengan *k-NN* yang bernilai *k* = 11, *k-NN* yang bernilai *k* = 13, dan *k-NN* yang bernilai *k* = 15. Hal tersebut menunjukkan bahwa pada skenario 8, klasifikasi LSTM dengan *k-NN*+GA berhasil menghasilkan akurasi tertinggi. Meskipun mencapai akurasi tertinggi, klasifikasi LSTM dengan *k-NN*+GA memiliki waktu komputasi yang lebih lama jika dibandingkan dengan varian klasifikasi LSTM lainnya, karena

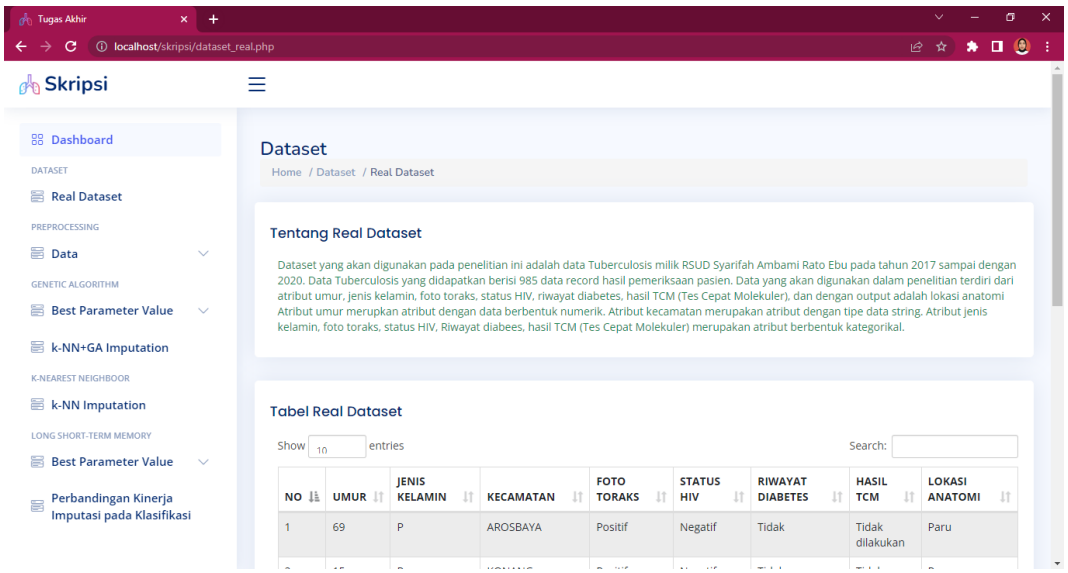
klasifikasi LSTM dengan *k-NN*+GA memiliki waktu komputasi selama 901,85 *second*.

4.3.9 Hasil Implementasi *User Interface*

Setelah mendapatkan hasil dari setiap skenario, hasil tersebut kemudian ditampilkan ke dalam *user interface*. Implementasi user interface dibuat menggunakan *PHP*, *HTML*, dan *template* yang berasal dari *Bootstrap*. Tampilan *user interface* dijelaskan pada sub bab di bawah ini.

4.3.9.1 Tampilan *Dashboard*

Tampilan dashboard adalah tampilan awal atau inde halaman. Berikut adalah tampilan dari Halaman dashboard yang ditampilkan pada Gambar 4.18.

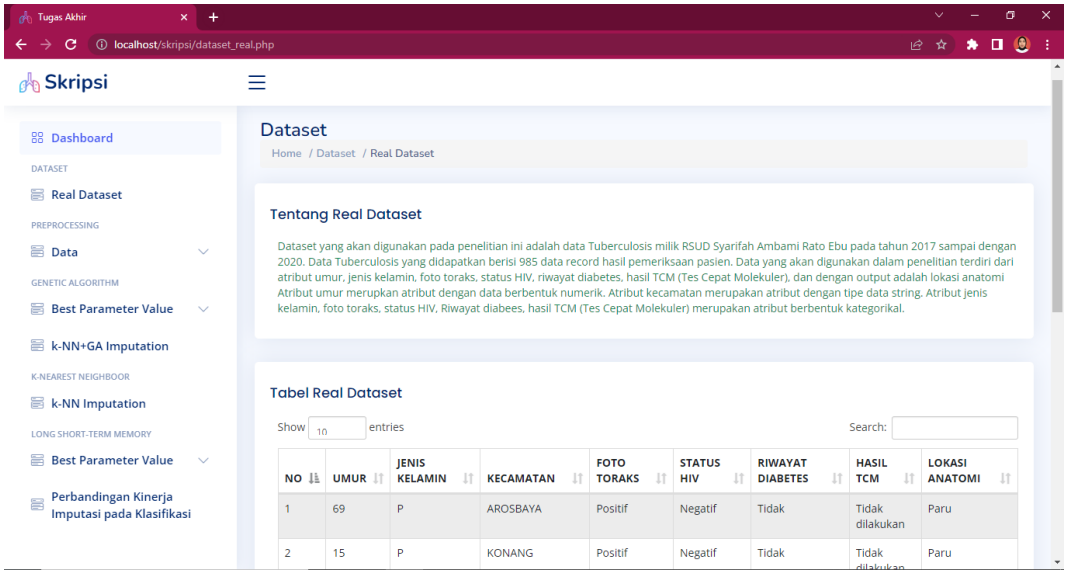


Gambar 4.18 Tampilan Dashboard

Berdasarkan Gambar 4.18, halaman *dashboard* menampilkan informasi berupa judul peneilitian, penjelasan mengenai topik, masalah dan solusi dari penelitian yang dilakukan.

4.3.9.2 Tampilan Dataset

Pada sub bab ini menjelaskan mengenai tampilan dataset. Berikut adalah tampilan dari halaman dashboard yang ditampilkan pada Gambar 4.19.

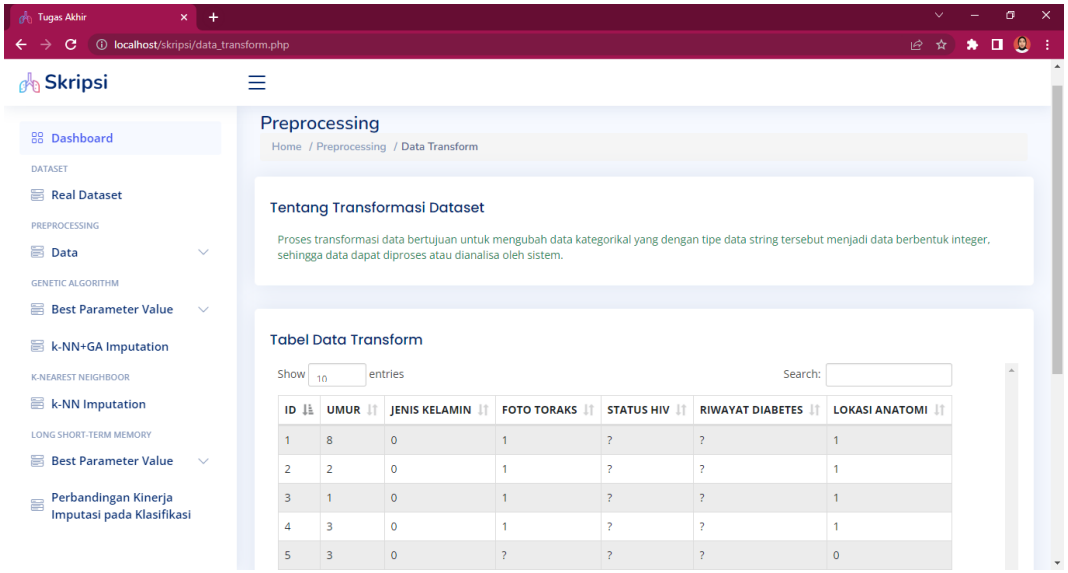


Gambar 4.19 Tampilan *Real Dataset*

Berdasarkan Gambar 4.19, tampilan dataset menampilkan penjelasan mengenai real dataset pada penelitian ini. Hal yang dijelaskan meliputi asal dataset, atribut pada dataset dan jenis datanya.

4.3.9.3 Tampilan Preprocessing

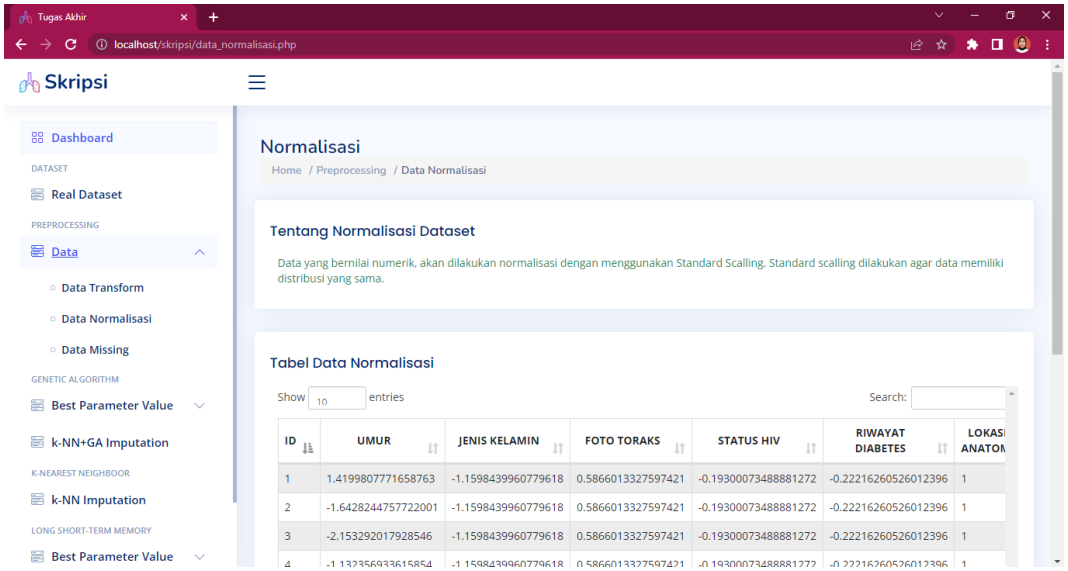
Pada sub bab ini menjelaskan mengenai tampilan *preprocessing*. Tampilan *preprocessing* dapat dilihat pada Gambar 4.20 hingga Gambar 4.23.



Gambar 4.20 Tampilan *Data Transform*

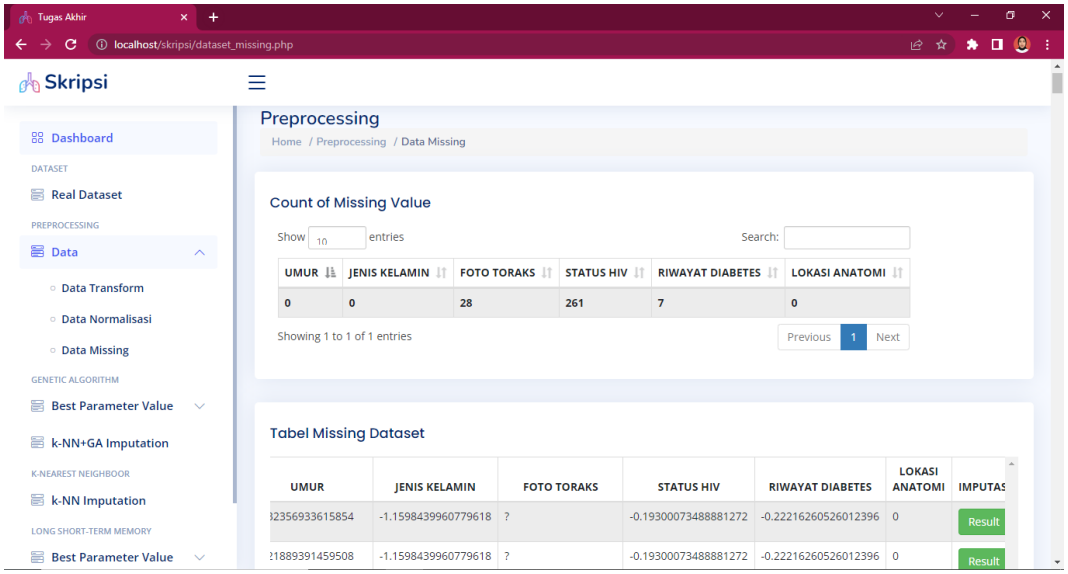
Berdasarkan Gambar 4.20, pada halaman tampilan *data transform* menampilkan informasi mengenai penjelasan singkat proses transformasi dan

dataset yang telah dilakukan transformasi. Kolom yang mengandung *missing value* pada tiap atribut data record nya ditandai dengan tanda baca tanya (?).

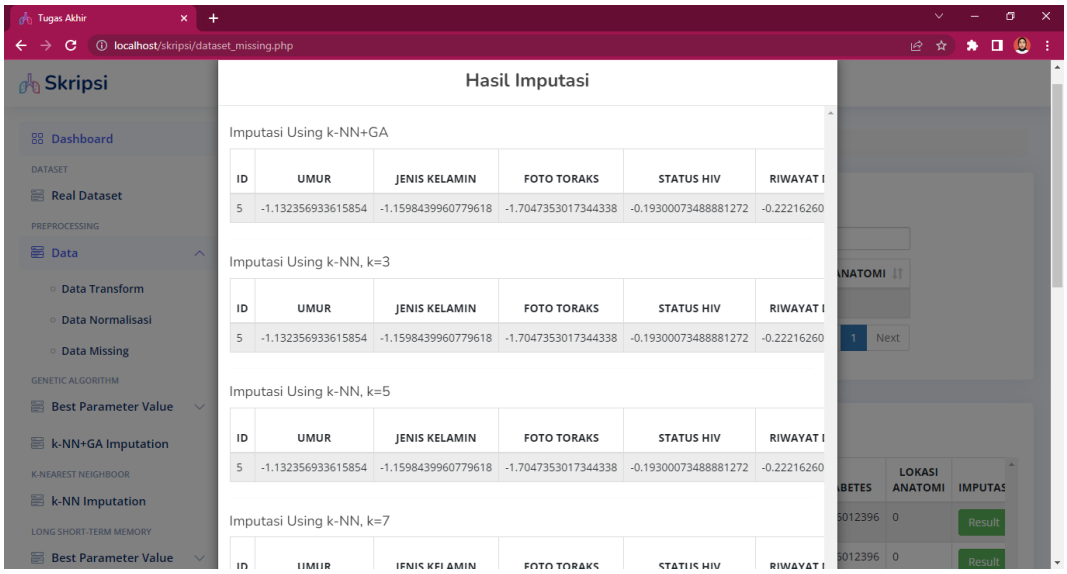


Gambar 4.21 Tampilan *Data Normalization*

Berdasarkan Gambar 4.21, pada halaman tampilan *data normalization* menampilkan informasi mengenai penjelasan singkat tujuan proses normalisasi dan dataset yang telah dilakukan normalisasi. Kolom yang mengandung *missing value* pada tiap atribut data record nya ditandai dengan tanda baca tanya (?).



Gambar 4.22 Tampilan *Data Missing Value*

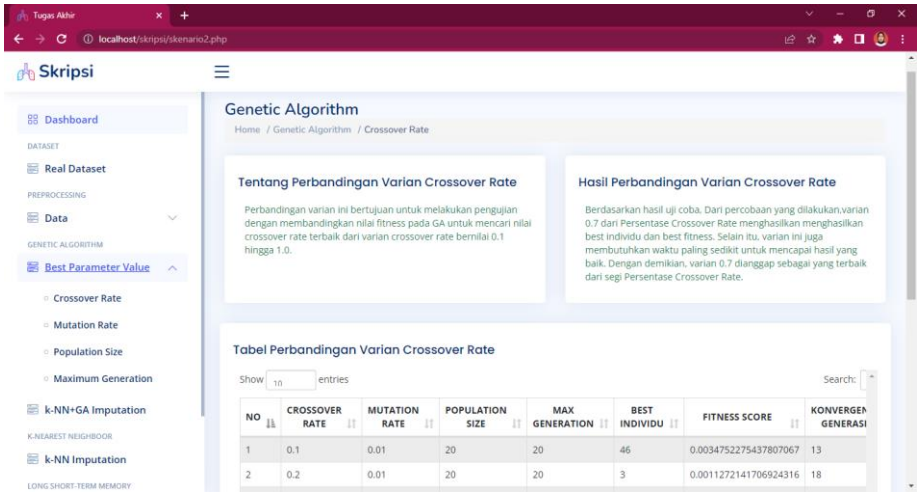


Gambar 4.23 Tampilan *Modal* Hasil Imputasi *Missing Value*

Berdasarkan Gambar 4.22 dan Gambar 4.23, pada halaman tampilan *data missing value* menampilkan informasi mengenai jumlah data yang hilang dalam tiap atributnya dan tabel yang berisi dataset dengan *missing value*. Kolom yang mengandung *missing value* pada tiap atribut data record nya ditandai dengan tanda baca tanya (?). Pada setiap *data record*, terdapat aksi untuk melihat hasil dari imputasi *missing value* yang ditampilkan dalam bentuk *modal*. Dalam modal tersebut menampilkan hasil imputasi dengan berbagai varian imputasi *k*-NN.

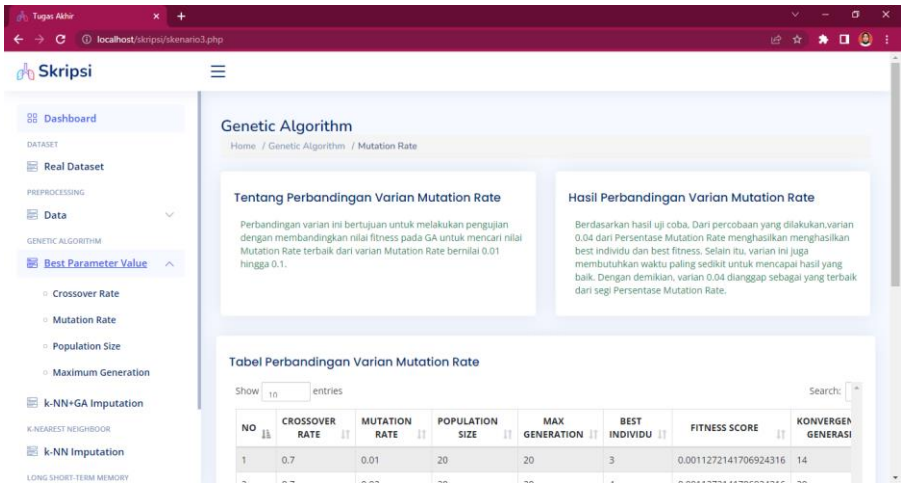
4.3.9.4 Tampilan Genetic Algorithm

Pada sub bab ini menjelaskan mengenai tampilan Genetic Algorithm. Tampilan tersebut dapat dilihat pada Gambar 4.24 hingga Gambar 4.28.



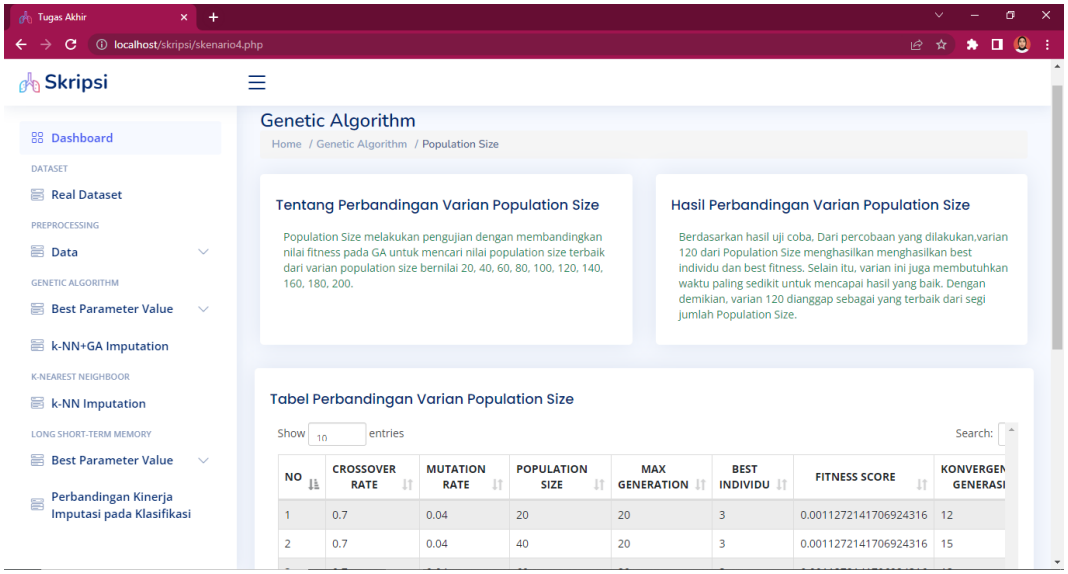
Gambar 4.24 Tampilan *Genetic Algorithm* : *Crossover Rate*

Berdasarkan Gambar 4.24, halaman ini menyajikan penjelasan singkat mengenai perbandingan varian *crossover rate*, yang mencakup tabel dan hasil perbandingan untuk memberikan informasi yang lebih terperinci.



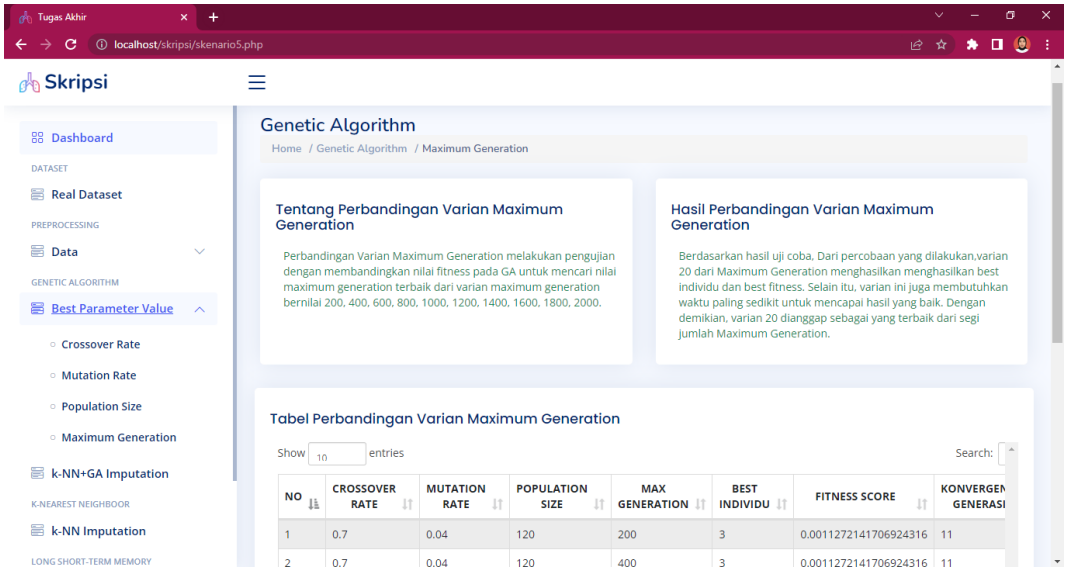
Gambar 4.25 Tampilan *Genetic Algorithm* : *Mutation Rate*

Berdasarkan Gambar 4.25, halaman ini menyajikan penjelasan singkat mengenai perbandingan varian *mutation rate*, yang mencakup tabel dan hasil perbandingan untuk memberikan informasi yang lebih terperinci.



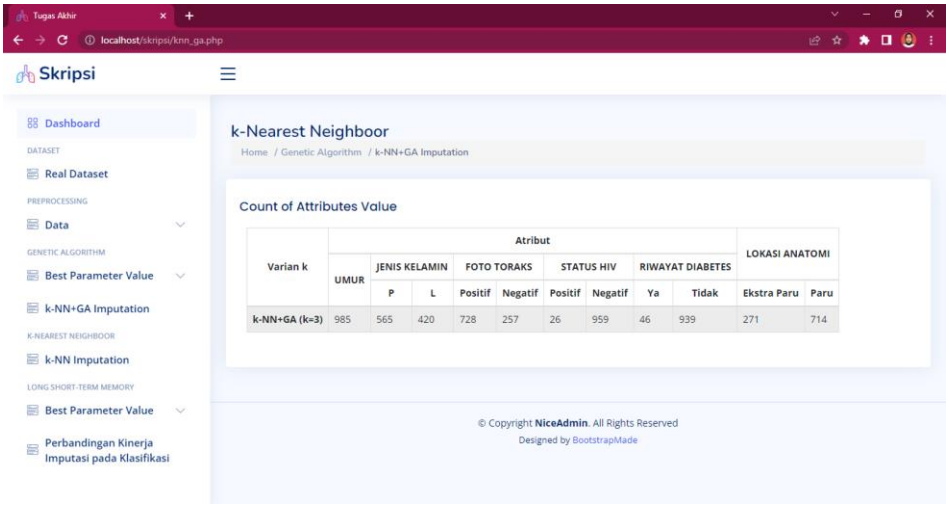
Gambar 4.26 Tampilan *Genetic Algorithm* : *Population Size*

Berdasarkan Gambar 4.26, halaman ini menyajikan penjelasan singkat mengenai perbandingan varian *population size*, yang mencakup tabel dan hasil perbandingan untuk memberikan informasi yang lebih terperinci.



Gambar 4.27 Tampilan *Genetic Algorithm* : *Maximum Generation*

Berdasarkan Gambar 4.27, halaman ini menyajikan penjelasan singkat mengenai perbandingan varian *maximum generation*, yang mencakup tabel dan hasil perbandingan untuk memberikan informasi yang lebih terperinci.

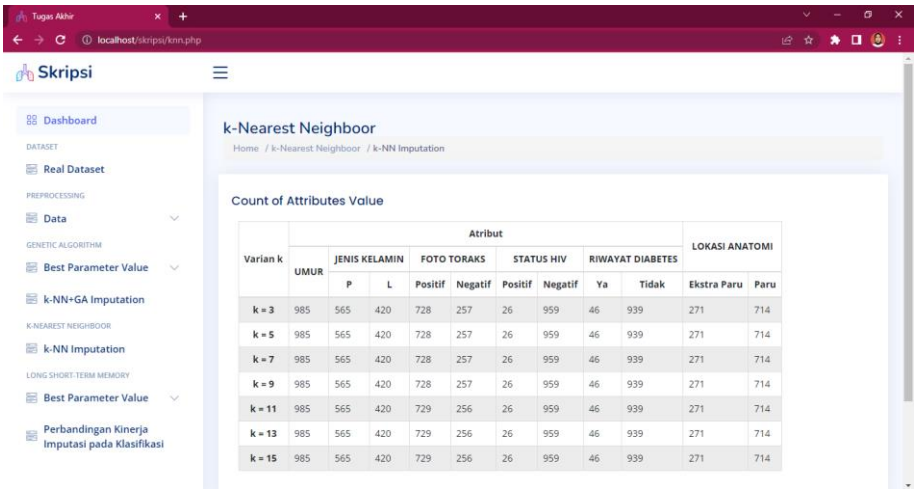


Gambar 4.28 Tampilan *Genetic Algorithm* : *k*-NN+GA *Imputation*

Berdasarkan Gambar 4.28, halaman ini memberikan informasi tentang jumlah data pada setiap nilai atribut setelah dilakukan imputasi missing value menggunakan *k*-NN+GA. Halaman ini bertujuan untuk mengetahui distribusi data pada setiap atribut.

4.3.9.5 Tampilan *k-Nearest Neighboor*

Pada sub bab ini menjelaskan mengenai tampilan *k-Nearest Neighboor*. Tampilan halaman *k-Nearest Neighboor* dapat dilihat pada Gambar 4.29.



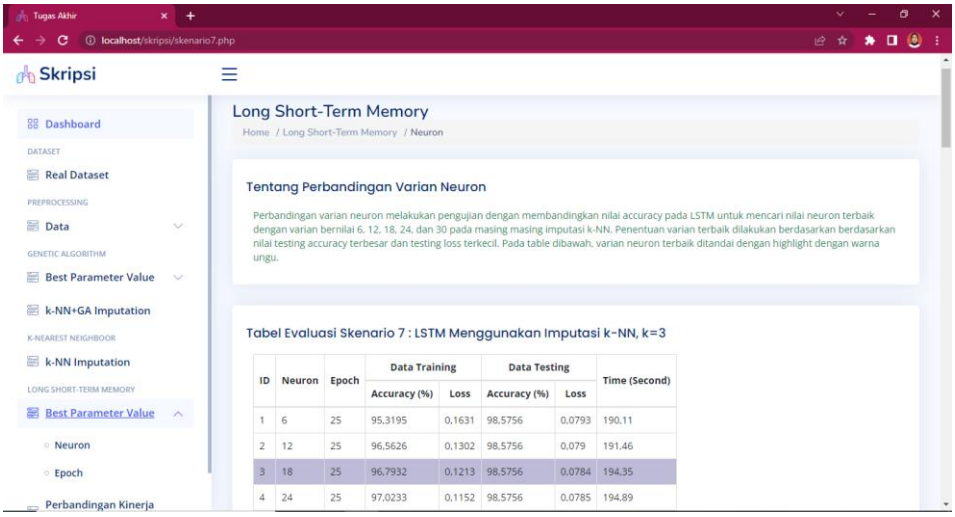
Gambar 4.29 Tampilan *k - Nearest Neighboor* : *k*-NN *Imputation*

Berdasarkan Gambar 4.29, halaman ini memberikan informasi tentang jumlah data pada setiap nilai atribut setelah dilakukan imputasi *missing value*

menggunakan varian imputasi k-NN tanpa optimasi. Halaman ini bertujuan untuk mengetahui distribusi data pada setiap atribut.

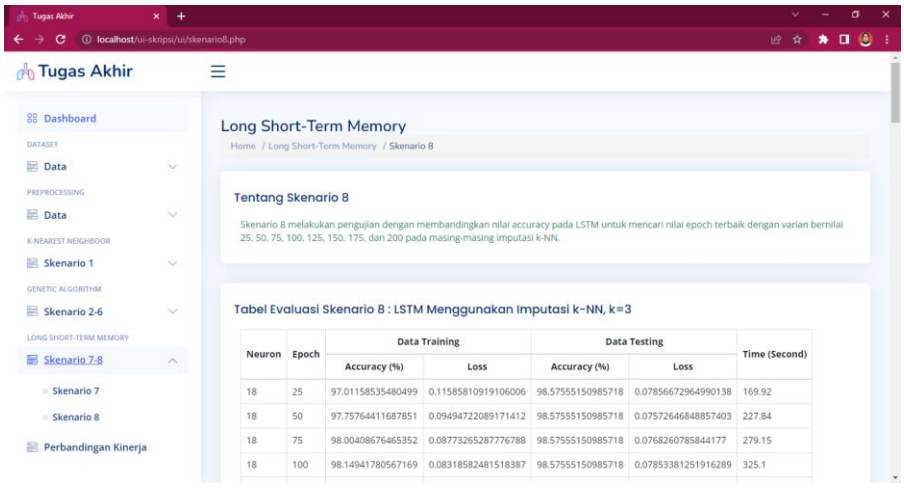
4.3.9.6 Tampilan Long Short-Term Memory

Pada sub bab ini menjelaskan mengenai tampilan *Long Short-Term Memory*. Tampilan tersebut dilihat pada Gambar 4.30 hingga Gambar 4.32.



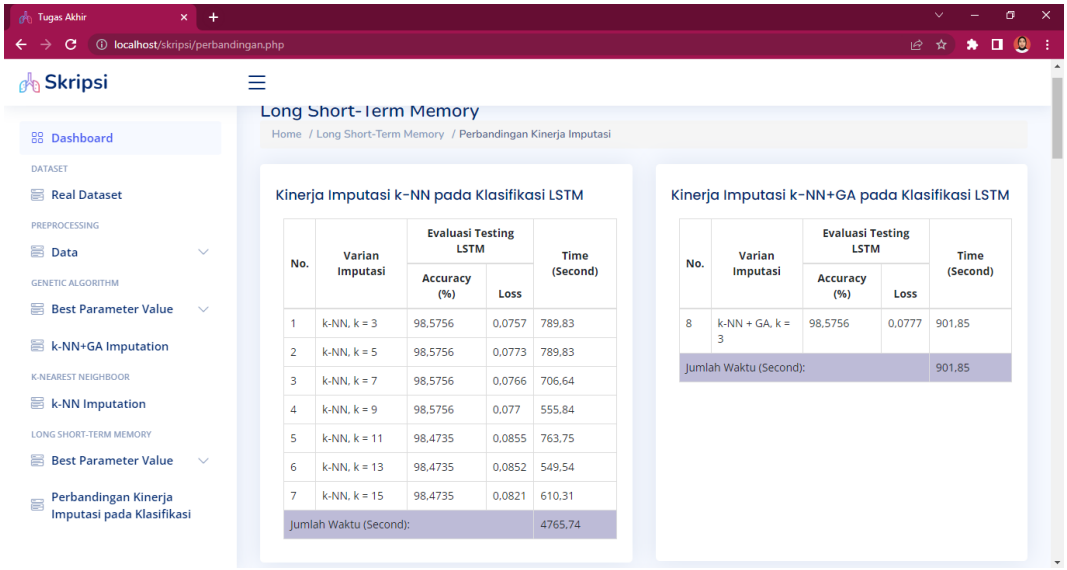
Gambar 4.30 Tampilan *Long Short-Term Memory* : *Neuron*

Berdasarkan Gambar 4.30, halaman ini menyajikan penjelasan singkat mengenai perbandingan varian *neuron*, yang mencakup tabel dan hasil perbandingan untuk memberikan informasi yang lebih terperinci.



Gambar 4.31 Tampilan *Long Short-Term Memory* : *Epoch*

Berdasarkan Gambar 4.31, halaman ini menyajikan penjelasan singkat mengenai perbandingan varian *epoch*, yang mencakup tabel dan hasil perbandingan untuk memberikan informasi yang lebih terperinci.



Gambar 4.32 Tampilan *Long Short-Term Memory* Perbandingan Kinerja

Berdasarkan Gambar 4.32, halaman ini menyajikan perbandingan kinerja klasifikasi LSTM menggunaka *k*-NN tanpa optimasi dan *k*-NN+GA, yang mencakup tabel dan kesimpulan hasil kinerja antar kedua nya untuk memberikan informasi yang lebih terperinci.

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

1. Hasil pencarian nilai terbaik pada parameter GA menunjukkan bahwa nilai *crossover rate* terbaik adalah 0.7. Nilai *mutation rate* terbaik yaitu 0.04. *Population size* terbaik sebesar 120. Kemudian, varian *max generation* terbaik adalah 1600. Seluruh nilai parameter tersebut dapat dianggap sebagai varian terbaik karena mampu menghasilkan *fitness score* terbaik (*fitness score* terkecil) yaitu 0,0011 dengan waktu yang paling singkat.
2. Hasil pencarian nilai neuron dan epoch terbaik untuk klasifikasi LSTM, menunjukkan bahwa kombinasi terbaik dari jumlah *neuron* dan *epoch* berbeda untuk setiap data imputasi *k*-NN. Terdapat beberapa kombinasi terbaik, seperti dengan imputasi *k*-NN, *k* = 3, jumlah *neuron* terbaik adalah 18 dan jumlah *epoch* terbaik adalah 50. Imputasi *k*-NN, *k* = 5, jumlah *neuron* terbaik adalah 12 dan jumlah *epoch* terbaik adalah 150. Imputasi *k*-NN, *k* = 7, jumlah *neuron* terbaik adalah 30 dan jumlah *epoch* terbaik adalah 100. Imputasi *k*-NN, *k* = 9, jumlah *neuron* terbaik adalah 30 dan jumlah *epoch* terbaik adalah 25. Imputasi *k*-NN, *k* = 11, jumlah *neuron* terbaik adalah 30 dan jumlah *epoch* terbaik adalah 50. Imputasi *k*-NN, *k* = 13, jumlah *neuron* terbaik adalah 24 dan jumlah *epoch* terbaik adalah 25. Imputasi *k*-NN, *k* = 15, jumlah *neuron* terbaik adalah 18 dan jumlah *epoch* terbaik adalah 50. Sementara dengan imputasi *k*-NN + GA, *k* = 3, jumlah *neuron* dan *epoch* terbaik yaitu 18 dan 50.
3. Hasil perbandingan akurasi pada klasifikasi LSTM menunjukkan bahwa nilai akurasi tertinggi selalu sama, yaitu 98,5756%, untuk varian klasifikasi LSTM dengan imputasi *k*-NN bernilai *k* = 3, 5, 7, 9, dan *k*-NN+GA dengan *k* = 3. Sedangkan, nilai akurasi terendah stabil pada 98,4735% untuk varian klasifikasi LSTM dengan imputasi *k*-NN bernilai *k* = 11, 13, dan 15. Namun untuk mencapai akurasi tertinggi tersebut, klasifikasi LSTM dengan imputasi *k*-NN tanpa optimasi GA memerlukan waktu yang lebih lama, yaitu 4765,74 detik. Hal ini disebabkan karena percobaan imputasi *k*-NN dengan varian nilai *k* dilakukan secara berulang dan manual. Sedangkan,

klasifikasi LSTM dengan *k*-NN+GA hanya memerlukan waktu 901,85 detik untuk mencapai akurasi tertinggi. Oleh karena itu, klasifikasi LSTM dengan imputasi *k*-NN+GA dianggap lebih baik, karena dapat mencapai akurasi tertinggi dengan waktu yang lebih singkat dibandingkan dengan metode klasifikasi LSTM yang menggunakan imputasi *k*-NN tanpa optimasi GA.

5.2 Saran

Hasil Penelitian tentang pencarian nilai *k* terbaik pada *k*-NN menggunakan *Genetic Algorithm* (GA) untuk imputasi data *tuberculosis* pada klasifikasi LSTM sudah menunjukkan hasil yang lebih baik dibandingkan dengan *k*-NN tanpa optimasi. Terdapat beberapa hal yang mempengaruhi kinerja GA, salah satunya adalah dalam penentukan ukuran *fitness score*. Ukuran *fitness score* yang digunakan harus mampu mengukur seberapa baik suatu individu dalam populasi, sesuai dengan kriteria yang ditentukan. Untuk pengembangan penelitian selanjutnya, dapat menggunakan upaya penentuan ukuran *fitness score* yang berbeda.

REFERENSI

[1] Irwan, Etika dan Perilaku Kesehatan, Yogyakarta: CV. Absolute Media, 2017.

[2] Iwan Setia Budi, Yustini Ardillah, Indah Purnama Sari, Dwi Septiawati, "Analisis Faktor Risiko Kejadian penyakit Tuberculosis Bagi Masyarakat Daerah Kumuh Kota Palembang," *Jurnal Kesehatan Lingkungan Indonesia,* pp. 87 - 94, 2018.

[3] Hiswani, "Tuberkulosis Merupakan Penyakit Infeksi yang Masih Menjadi Masalah Kesehatan Masyarakat," *Skripsi, Universitas Sumatera Utara.,* 2008.

[4] Arfan. A, Lussiana E.T.P., "Perbandingan Algoritma Long Short-Term Memory dengan SVR Pada Prediksi Harga Saham di Indonesia," *Petir: Jurnal Pengkajian dan Penerapan Teknik Informatika,* vol. 13(1), pp. 33-43, 2020.

[5] A. Achmad Fikri Sallaby, "Analysis of Missing Value Imputation Application with K-Nearst Neighbor (KNN) Algorithm in Dataset," *The IJICS (International Journal of Informatics and Computer Science),* vol. V, pp. 141-144, 2021.

[6] Taufiq Rizaldi, Fendik Eko Purnomo, Aji Seto arifianto, "Perbandingan Metode K-NN dan Bayes pada Missing Imputation," *Jurnal Teknologi Informatika dan Terapan,* vol. v, 2018.

[7] Susanti, Shantika Martha, Evy Sulistianingsih, "K-Nearest Neighbor Dalam Imputasi Missing Data," *Buletin Ilmiah Math. Stat. dan Terapannya (Bimaster),* vol. VII, pp. 9-14, 2018.

[8] N. H. Abidatul Izzah, "Imputasi Missing Data Menggunakan Metode K-Nearest Neighbour Dengan Optimasi Algoritma Genetika," vol. II, 2013.

[9] Ispandi, Romi Satrio Wahono, "Penerapan Algoritma Genetika untuk Optimasi Parameter pada Support Vector Machine untuk Meningkatkan Prediksi Pemasaran Langsung," *Journal of Intelligent Systems,* vol. I, 2015.

[10] W. Yunus, "Algoritma K-Nearest Neighbor Berbasis Particle Swarm Optimization Untuk Prediksi Penyakit Ginjal Kronik," *Jurnal Teknik Elektro CosPhi,* vol. II, 2018.

[11] KEMENKES, "Penanggulangan Tuberculosis". Indonesia Patent 364/Menkes/SK/V/2009, 2016.

[12] Suntoro, Data Mining Algoritme dan Implementasi Menggunakan Bahasa Pemrogaman PHP, 2019.

[13] Moch. Lutfi, Mochamad Hasyim, "Penanganan Data Missing Value Pada Kualitas Produksi Jagung Dengan Menggunakan Metode K-NN Imputation Pada Algoritma C4.5," *JURNAL RESISTOR*, vol. II, 2019.

[14] Farhangfar, A. Kurgan, "Impact of imputation of missing values on classification," *Pattern Recognition*, vol. XLI, pp. 3692-3705, 2008.

[15] M. Ramuka, "www.analyticsvidhya.com," Analytics Vidhya, 21 Juni 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/defining-analysing-and-implementing-imputation-techniques/#:~:text=Frequent%20Category%20Imputation,referred%20to%20as%20Mode%20Imputation.. [Accessed 25 August 2022].

[16] S. Mulani, "Using StandardScaler() Function to Standardize Python Data," DigitalOcean, 4 August 2022. [Online]. Available: https://www.digitalocean.com/community/tutorials/standardscaler-function-in-python. [Accessed 25 December 2022].

[17] J. Hale, "Scale, Standardize, or Normalize with Scikit-Learn," Towards Data Science, 04 March 2019. [Online]. Available: https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02. [Accessed 25 12 2022].

[18] d. Acuna, "The treatment of missing values and its effect on classifier accuracy, Classification, Clustering, and Data Mining Ahallications," *Springer Berlin Heidelberg*, pp. 639-647, 2004.

[19] M. Nishom, "Perbandingan Akurasi Euclidean Distance, Minkowski Distance, dan Manhattan Distance pada Algoritma KMeans Clustering berbasis Chi-Square," *Jurnal Informatika: Jurnal Pengembangan IT (JPIT)*, vol. IX, 2019.

[20] K. Makwana, "medium.com," Geek Culture, 31 May 2021. [Online]. Available: https://medium.com/geekculture/frequent-category-imputation-missing-data-imputation-technique-

4d7e2b33daf7#:~:text=Mode%20imputation%20consists%20of%20replacing,value%20or%20most%20frequent%20category.. [Accessed 14 October 2022].

[21] I. A. Siradjudin, Kecerdasan Komptasional dan Aplikasinya dengan Menggunakan Python, Yogyakarta: Teknosain, 2018.

[22] Mutiara Ayu Banjarsari, H. Irwan Budiman, Andi Farmadi, "Penerapan K-Optimal Pada Algoritma Knn untuk Prediksi Kelulusan Tepat Waktu Mahasiswa Program Studi Ilmu Komputer Fmipa Unlam Berdasarkan IP Sampai Dengan Semester 4," *Kumpulan jurnaL Ilmu Komputer (KLIK),* vol. 02, 2015.

[23] Muhammad Wildan Putra Aldi, Jondri, Annisa Aditsania, "Analisis dan Implementasi Long Short Term Memory Neural Network untuk Prediksi Harga Bitcoin," *e-Proceeding of Engineering*, vol. 5, p. 3548, 2018.

[24] Bain Khusnul Khotimah, Muhammad Syarief, Miswanto, Herry Suprajitno, "Optimasi Bobot K-Means Clustering Untuk Mengatasi Missing Value Dengan Menggunakan Algoritma Genetika," *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK),* vol. VIII, pp. 745-752, 2021.

[25] Saiful Ulya , M. Arief Soeleman, dan Fikri Budima, "Optimasi Parameter K Pada Algoritma K-NN Untuk Klasifikasi Prioritas Bantuan Pembangunan Desa," *Techno.COM,* vol. 20, pp. 83-96, 2021.

[26] Muhammad Yasir, Khuram Shahzad, Yasir Arfat, and Naveed Ahmad, "Analysis of Critical Success Factors for Enterprise Resource Planning (ERP) Implementation in Public Sector Organizations," *Sustainability,* vol. 11, no. 21, 2019.

[27] DEPKES, "Kesehatan". Indonesia Patent Undang-Undang No. 36 Tahun 2009, 2009.

[28] A. Wibowo, "BINUS," BINUS UNIVERSITY GRADUATE PROGRAM, 24 Nov 2017. [Online]. Available: https://mti.binus.ac.id/2017/11/24/10-fold-cross-validation/. [Accessed 18 Oct 2022].

[29] Ilham Farros, Deni Mahdiana, Ani Dijah Rahajoe, "Penerapan Algoritma K-Nearest Neighbor Untuk Analisis Sentimen Ulasan SiCepat Ekspres Pada Twitter," *Seminar Nasional Mahasiswa Fakultas Teknologi Informasi*

(SENAFTI), pp. 1723-1730, 2022.

[30] J. H. a. M. Kamber, Data mining; Concepts and Techniques Second Edition, 2006.

[31] Pramita Sree Muhuri , Prosenjit Chatterjee, Xiaohong Yuan, Kaushik Roy and Albert Esterline, "Using a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) to Classify Network Attacks," *MDPI*, vol. 11, p. 243, 2020.

[32] Desak Made Dwi Utami Putra, Subanar, "Penerapan Algoritma Genetika Untuk Menyelesaikan Permasalahan Penjadwalan Perawat Dengan Fuzzy Fitness Function," *IJCCS*, vol. 6, pp. 11-12, 2012.

[33] D. H. Rahm Erhard, " Data Cleaning:Problems and Current Approaches," *IEEE Data Engineering Bulletin*, pp. 3-13, 2000.

[34] S. Notoatmojo, Promosi Kesehatan & Ilmu Perilaku, Rineka Cipta, 2007.

[35] N. K. Manaswi, Deep Learning with Applications Using Python, Apress, 2018.

[36] KEMENKES, "Petunjuk teknis Pemeriksaan TB Menggunakan Tes Cepat Molekuler". Indonesia September 2017.

[37] Rizki Rino Pratama, Rintho Rante Rerung, Aditya Erfina, "Penyelesaian Travelling Salesman Problem Menggunakan Algoritma Genetika," *JURSISTEKNI (Jurnal Sistem Informasi dan Teknologi Informasi)*, vol. 2, pp. 10 - 18, 2020.

[38] Darnisa Azzahra Nasution, Hidayah Husnul Khotimah, Nurul Chamidah, "Perbandingan Normalisasi Data Untuk Klasifikasi Wine Menggunakan Algoritma K-NN," *CESS (Journal of Computer Engineering System and Science)*, vol. IV, 2019.

LAMPIRAN

Lampiran 1

Tabel 4.6 Hasil Skenario 6.

Sampel Data Sebelum Imputasi						
No.	UMUR	JENIS_KELAMIN	FOTO_TORAKS	STATUS_HIV	RIWAYAT_DIABETES	LOKASI_ANATOMI
7	-0.621890	-1.159844	?	-0.1930007	-0.222163	0
13	-0.621890	-1.159844	0.586601	?	-0.222163	1
15	-0.621889	-1.159844	?	-0.1930007	-0.222163	0
18	0.909513	-1.159844	0.586601	?	-0.222163	1
27	0.909513	-1.159844	?	-0.1930007	-0.222163	0
Sampel Data Setelah Imputasi *k*-NN dengan *k* = 3						
No.	UMUR	JENIS_KELAMIN	FOTO_TORAKS	STATUS_HIV	RIWAYAT_DIABETES	LOKASI_ANATOMI
7	-0.621890	-1.159844	0.586601	-0.1930007	-0.222163	0
13	-0.621890	-1.159844	0.586601	-0.1930007	-0.222163	1
15	-0.621889	-1.159844	0.586601	-0.1930007	-0.222163	0
18	0.909513	-1.159844	0.586601	-0.1930007	-0.222163	1
27	0.909513	-1.159844	0.586601	-0.1930007	-0.222163	0