

***DEEP NEURAL NETWORK* UNTUK PREDIKSI STROKE**



TESIS

ANAS FAISAL
14002356

Program Studi Ilmu Komputer (S2)
Universitas Nusa Mandiri
Jakarta
2021

DEEP NEURAL NETWORK UNTUK PREDIKSI STROKE



TESIS

**Diajukan sebagai salah satu syarat untuk memperoleh gelar
Magister Ilmu Komputer (M.Kom)**

**ANAS FAISAL
14002356**

**Program Studi Ilmu Komputer (S2)
Universitas Nusa Mandiri
Jakarta
2021**

SURAT PERNYATAAN ORISINALITAS DAN BEBAS PLAGIARISME

Yang bertanda tangan di bawah ini :

Nama : Anas Faisal
NIM : 14002356
Program Studi : Ilmu Komputer
Jenjang : Strata Dua (S2)
Konsentrasi : Data Mining

Dengan ini menyatakan bahwa tesis yang telah saya buat dengan judul: “*Deep Neural Network* untuk Prediksi Stroke” adalah hasil karya sendiri, dan semua sumber baik yang kutip maupun yang dirujuk telah saya nyatakan dengan benar dan tesis belum pernah diterbitkan atau dipublikasikan dimanapun dan dalam bentuk apapun.

Demikianlah surat pernyataan ini saya buat dengan sebenar-benarnya. Apabila dikemudian hari ternyata saya memberikan keterangan palsu dan atau ada pihak lain yang mengklaim bahwa tesis yang telah saya buat adalah hasil karya milik seseorang atau badan tertentu, saya bersedia diproses baik secara pidana maupun perdata dan kelulusan saya dari Program Studi Ilmu Komputer (S2) Universitas Nusa Mandiri dicabut/dibatalkan.

Jakarta, 20 Agustus 2021

Yang menyatakan

A handwritten signature in black ink is written over a yellow revenue stamp. The stamp features the Garuda Pancasila emblem and the text '1000', 'METERAI TEMPEL', and a unique alphanumeric code 'A1E7AJX377814846'.

Anas Faisal

HALAMAN PERSETUJUAN DAN PENGESAHAN TESIS

Tesis ini diajukan oleh:

Nama : Anas Faisal
NIM : 14002356
Program Studi : Ilmu Komputer
Jenjang : Strata Dua (S2)
Konsentrasi : *Data Mining*
Judul Tesis : Deep Neural Network Untuk Prediksi Stroke

Telah dipertahankan pada periode 2021-1 dihadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh Magister Ilmu Komputer (M.Kom) pada Program Studi Ilmu Komputer (S2) Universitas Nusa Mandiri.

Jakarta, 20 Agustus 2021

PEMBIMBING TESIS

Pembimbing I : Dr. Agus Subekti, M.T



DEWAN PENGUJI

Penguji I : Dr. Lindung Parningotan Manik,
M.T.I



Penguji II : Dr. Hilman Ferdinandus Pardede,
S.T, M.EICT



Penguji III /
Pembimbing I : Dr. Agus Subekti, M.T



KATA PENGANTAR

Alhamdulillah puji syukur kehadiran penulis panjatkan kehadiran Allah, SWT, yang telah melimpahkan rahmat dan karunia-Nya, sehingga pada akhirnya penulis dapat menyelesaikan laporan tesis ini tepat pada waktunya. Dimana laporan tesis ini penulis sajikan dalam bentuk buku yang sederhana.

Adapun judul tesis, yang penulis ambil sebagai berikut “*Deep Neural Network* untuk Prediksi Stroke”. Tujuan penulisan laporan tesis ini dibuat sebagai salah satu untuk mendapatkan gelar Magister Ilmu Komputer (M.Kom) pada Program Studi Ilmu Komputer (S2) Universitas Nusa Mandiri.

Laporan Tesis ini diambil berdasarkan hasil penelitian atau riset mengenai Prediksi Stroke menggunakan *Model Deep Neural Network* dan melakukan optimasi pada beberapa parameter, selanjutnya dilakukan analisis menggunakan *metrics performance model*. Penulis juga lakukan mencari dan menganalisa berbagai macam sumber referensi, baik dalam bentuk jurnal ilmiah, buku-buku literatur, internet, dll yang terkait dengan pembahasan pada laporan tesis ini.

Penulis menyadari bahwa tanpa bimbingan dan dukungan dari semua pihak dalam pembuatan laporan tesis ini, maka penulis tidak dapat menyelesaikan laporan tesis ini tepat pada waktunya. Untuk itu ijinkanlah penulis kesempatan ini untuk mengucapkan terimakasih kepada:

1. Allah SWT yang selalu mencurahkan nikmat dan rahmat-Nya kepada penulis sehingga penulis dapat menyelesaikan tesis ini tepat pada waktunya.
2. Bapak Dr. Agus Subekti, M.T selaku pembimbing tesis yang telah menyediakan waktu, pikiran dan tenaga dalam membimbing penulis dalam menyelesaikan tesis ini.
3. Ayahanda dan Ibunda yang selalu memberikan motivasi dan do’a yang tak pernah terhingga.
4. Haniek Listiyarini, istriku yang selalu menjadi pendamping dalam usaha menyelesaikan studi ini dan yang selalu memberikan motivasi dan kesabarannya, sehingga penulis dapat menyelesaikan penulisan tesis ini.

5. Seluruh Dosen Program Studi Ilmu Komputer (S2) Universitas Nusa Mandiri yang telah memberikan pelajaran yang berarti bagi penulis selama menempuh studi
6. Seluruh staf dan karyawan Studi Ilmu Komputer (S2) Universitas Nusa Mandiri yang telah melayani penulis dengan baik selama kuliah.

Serta semua pihak yang terlalu banyak untuk penulis sebutkan satu persatu sehingga terwujudnya penulisan laporan tesis ini. Penulis menyadari bahwa penulisan laporan tesis ini masih jauh sekali dari sempurna, untuk itu penulis mohon kritik dan saran yang bersifat membangun demi kesempurnaan penulisan karya ilmiah yang penulis hasilkan untuk yang akan datang.

Akhir kata semoga laporan tesis ini dapat bermanfaat bagi penulis khususnya dan bagi para pembaca yang berminat pada umumnya.

Jakarta, 20 Agustus 2021

Yang menyatakan



Anas Faisal

Penulis

SURAT PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Yang bertanda tangan di bawah ini, saya :

Nama : Anas Faisal
NIM : 14002356
Program Studi : Ilmu Komputer
Jenjang : Strata Dua (S2)
Konsentrasi : *Data Mining*
Jenis Karya : Tesis

Demi pengembangan ilmu pengetahuan, dengan ini menyetujui untuk memberikan ijin kepada pihak Program Studi Ilmu Komputer (S2) Universitas Nusa Mandiri **Hak Bebas Royalti Non-Eksklusif** (*Non-exclusive Royalti-Free Right*) atas karya ilmiah kami yang berjudul: “*Deep Neural Network* untuk Prediksi Stroke” beserta perangkat yang diperlukan (apabila ada).

Dengan **Hak Bebas Royalti Non-Eksklusif** ini pihak Universitas Nusa Mandiri berhak menyimpan, mengalih-media atau bentukkan, mengelolanya dalam pangkalan data (*database*), mendistribusikannya dan menampilkan atau mempublikasikannya di *internet* atau media lain untuk kepentingan akademis tanpa perlu meminta ijin dari kami selama tetap mencantumkan nama kami sebagai penulis/pencipta karya ilmiah tersebut.

Saya bersedia untuk menanggung secara pribadi, tanpa melibatkan pihak Universitas Nusa Mandiri, segala bentuk tuntutan hukum yang timbul atas pelanggaran Hak Cipta dalam karya ilmiah saya ini.

Demikian pernyataan ini saya buat dengan sebenarnya.

Jakarta, 20 Agustus 2021
Yang menyatakan,



Anas Faisal

ABSTRAK

Nama : Anas Faisal
NIM : 14002356
Program Studi : Ilmu Komputer
Jenjang : Strata Dua (S2)
Konsentrasi : *Data Mining*
Judul : “*Deep Neural Network* untuk Prediksi Stroke”

Pada Tahun 2019 Organisasi Kesehatan Dunia (WHO) mendudukan stroke sebagai tujuh dari sepuluh penyebab utama kematian. Kementerian Kesehatan menggolongkan stroke sebagai penyakit katastropik karena memiliki dampak luas secara ekonomi dan sosial. Oleh karena itu, diperlukan peran dari teknologi informasi untuk memprediksi risiko stroke guna pencegahan dan perawatan dini. Analisis data yang memiliki kelas tidak seimbang mengakibatkan ketidakakuratan dalam memprediksi stroke. Penelitian ini membandingkan tiga teknik oversampling untuk mendapatkan model prediksi yang lebih baik. Data kelas yang sudah diseimbangkan diuji menggunakan tiga model Arsitektur *Deep Neural Network* (DNN) dengan melakukan optimasi pada beberapa parameter yaitu *optimizer*, *learning rate* dan *epoch*. Hasil paling baik didapatkan teknik *oversampling SMOTETomek* dan Arsitektur DNN dengan lima *hidden layer*, optimasi *Adam*, *learning rate* 0.001 dan jumlah *epoch* 500. Skor akurasi, presisi, *recall*, dan *f-measure* masing-masing mendapatkan 0.96, 0.9614, 0.9608 dan 0.9611.

Kata kunci: Pembelajaran Mendalam, *DNN*, *SMOTETomek*, Stroke, Klasifikasi

ABSTRACT

Name : Anas Faisal
NIM : 14002356
Study of Program : Computer Science
Levels : Strata Dua (S2)
Concentration : Data Mining
Title : “Deep Neural Network for Stroke Prediction”

In 2019 the World Health Organization (WHO) established stroke as seven of the ten leading causes of death. The Ministry of Health classifies stroke as a catastrophic disease because it has a wide impact economically and socially. Therefore, it is necessary to play the role of information technology to predict stroke risk for prevention and early treatment. Analysis of data that has an imbalance class results in inaccuracies in predicting strokes. The study compared three oversampling techniques to get a better prediction model. The balanced class data was tested using three Deep Neural Network (DNN) Architecture models by optimizing several parameters, namely optimizer, learning rate and epoch. The best result obtained SMOTETomek oversampling techniques and DNN Architecture with five hidden layers, Adam optimization, learning rate 0.001 and the number of epoch 500. Accuracy, precision, recall, and f-measure scores were 0.96, 0.9614, 0.9608 and 0.9611, respectively.

Keywords: Deep Learning, DNN, SMOTETomek, Stroke, Classification

DAFTAR ISI

	Halaman
Halaman Sampul	i
Halaman Judul.....	ii
Surat Pernyataan Orisinalitas dan Bebas Plagiarisme	iii
Halaman Persetujuan dan Pengesahan Tesis	iv
Kata Pengantar	v
Lembar Pernyataan Persetujuan Publikasi Karya Ilmiah	vii
Abstrak	viii
Abstract	ix
Daftar Isi.....	x
Daftar Tabel	xii
Daftar Gambar	xiv
Daftar Rumus	xv
Daftar Lampiran	xvi
BAB I PENDAHULUAN	
1.1. Latar Belakang Masalah	1
1.2. Identifikasi Masalah	4
1.3. Tujuan Penelitian	4
1.4. Ruang Lingkup Penelitian	4
1.5. Kontribusi Tesis	5
1.6. Sistematika Penulisan	5
BAB II LANDASAN/KERANGKA PEMIKIRAN	
2.1. Tinjauan Studi	7
2.2. Metode <i>Preprocessing</i>	10
2.2.1. <i>Data Cleaning</i>	10
2.2.2. <i>Feature Encoding</i>	11
2.2.3. <i>Feature Scaling</i>	12
2.2.4. <i>Handling Class Imbalance</i>	13
2.3. Arsitektur dan <i>Hyperparameter Deep Learning</i>	19
2.3.1. <i>Neural Network</i>	19
2.3.2. <i>Backpropagation</i>	21
2.3.3. <i>Loss Functions</i>	22
2.3.4. <i>Hyperparameter Model</i>	23
2.4. <i>Metrics Performance</i>	32
2.4.1. <i>Confusion Matrix</i>	32
2.4.2. <i>Accuracy</i>	33
2.4.3. <i>Precision</i>	33
2.4.4. <i>Recall</i>	34
2.3.5. <i>F1-score</i>	34
2.3.6. <i>Receiver Operating Characteristic Curve-Area Under the Curve</i>	34
2.3.7. <i>Kappa</i>	34

2.3.8. <i>Sensitivity</i>	34
2.3.9. <i>Specificity</i>	35
BAB III METODOLOGI PENELITIAN	
3.1. Deskripsi Masalah dan Dataset	36
3.2. Eksperimen Yang Dilakukan	38
3.2.1. Tahapan Penelitian	38
3.2.2. <i>Data Preparation</i>	39
3.2.3. Pembuatan Model Arsitektur <i>DNN</i>	42
3.3. <i>Metrics Performance</i> Evaluasi Model	45
3.4. Pengembangan Aplikasi	45
BAB IV HASIL PENELITIAN DAN PEMBAHASAN	
4.1. Analisis Statistik Dataset	46
4.1.1. <i>Deskriptif Statistic</i>	46
4.1.2. <i>Feature Importance</i>	47
4.1.3. <i>Missing Values</i>	48
4.1.4. <i>Imbalance Class</i>	49
4.1.5. <i>Feature Distribution</i>	49
4.2. Hasil Eksperimen dan Analisis.....	51
4.2.1. Hasil <i>Data Preparation</i>	52
4.2.2. Hasil Eksperimen Optimasi <i>SGD</i>	55
4.2.3. Hasil Eksperimen Optimasi <i>Adam</i>	64
4.2.4. Hasil Eksperimen Optimasi <i>RMSprop</i>	73
4.3. Model Prediktif Yang Diusulkan.....	86
4.3.1. Arsitektur <i>DNN</i> dan Nilai Parameter	86
4.3.2. <i>Metric Performance Model</i>	88
4.4. Aplikasi Yang Dikembangkan	94
4.4.1. Pengembangan Aplikasi	94
4.4.2. Perbandingan Dengan Penelitian Sebelumnya	96
BAB V PENUTUP	
5.1. Kesimpulan	98
5.2. Saran	99

DAFTAR REFERENSI
DAFTAR RIWAYAT HIDUP
LEMBAR BIMBINGAN TESIS
LAMPIRAN

DAFTAR TABEL

	Halaman
1. Tabel 2.1. Penelitian Terkait dengan Stroke	10
2. Tabel 2.2. <i>Illustration of Label Encoding</i>	12
3. Tabel 2.3. <i>Confusion Matrix</i>	32
4. Tabel 3.1. <i>Description of the Dataset Attribute</i>	37
5. Tabel 3.2. <i>Dataset</i>	38
6. Tabel 3.3. <i>Label Encode</i>	40
7. Tabel 3.4. Arsitektur DNN	42
8. Tabel 3.5. <i>Parameter Model DNN</i>	43
9. Tabel 3.6. <i>Parameter Optimasi Model DNN</i>	44
10. Tabel 3.7. Spesifikasi Komputer	44
11. Tabel 4.1. Deskriptif Statistik	46
12. Tabel 4.2. <i>Feature Important Coefisien</i>	47
13. Tabel 4.3. <i>Missing Values</i>	48
14. Tabel 4.4. <i>Missing Values after Imputation</i>	52
15. Tabel 4.5. <i>Dataset after Feature Encoding</i>	53
16. Tabel 4.6. <i>Dataset after Feature Scaling</i>	54
17. Table 4.7. Perbandingan <i>Accuracy Score SGD ADASYN</i>	56
18. Tabel 4.8. Perbandingan <i>Recall Score SGD ADASYN</i>	56
19. Table 4.9. Perbandingan <i>Precision Score SGD ADASYN</i>	57
20. Tabel 4.10. Perbandingan <i>F1-Score SGD ADASYN</i>	57
21. Tabel 4.11. Perbandingan <i>AUC Score SGD ADASYN</i>	58
22. Tabel 4.12. Perbandingan <i>Accuracy Score SGD BorderlineSMOTE</i>	58
23. Tabel 4.13. Perbandingan <i>Recall Score SGD BorderlineSMOTE</i>	59
24. Tabel 4.14. Perbandingan <i>Precision Score SGD BorderlineSMOTE</i>	60
25. Tabel 4.15. Perbandingan <i>F1-Score SGD BorderlineSMOTE</i>	60
26. Table 4.16. Perbandingan <i>AUC Score SGD BorderlineSMOTE</i>	61
27. Tabel 4.17. Perbandingan <i>Accuracy Score SGD SMOTETomek</i>	61
28. Tabel 4.18. Perbandingan <i>Recall Score SGD SMOTETomek</i>	62
29. Tabel 4.19. Perbandingan <i>Precision Score SGD SMOTETomek</i>	62
30. Tabel 4.20. Perbandingan <i>F1-Score Score SGD SMOTETomek</i>	63
31. Tabel 4.21. Perbandingan <i>AUC Score SGD SMOTETomek</i>	63
32. Tabel 4.22. Perbandingan <i>Accuracy Score Adam ADASYN</i>	64
33. Tabel 4.23. Perbandingan <i>Recall Score Adam ADASYN</i>	65
34. Tabel 4.24. Perbandingan <i>Precision Score Adam ADASYN</i>	65
35. Tabel 4.25. Perbandingan <i>F1-Score Adam ADASYN</i>	66
36. Tabel 4.26. Perbandingan <i>AUC Score Adam ADASYN</i>	66
37. Tabel 4.27. Perbandingan <i>Accuracy Score Adam BorderlineSMOTE</i>	67
38. Tabel 4.28. Perbandingan <i>Recall Score Adam BorderlineSMOTE</i>	68
39. Tabel 4.29. Perbandingan <i>Precision Score Adam BorderlineSMOTE</i>	68
40. Tabel 4.30. Perbandingan <i>F1-Score Adam BorderlineSMOTE</i>	69
41. Tabel 4.31. Perbandingan <i>AUC Score Adam BorderlineSMOTE</i>	69
42. Tabel 4.32. Perbandingan <i>Accuracy Score Adam SMOTETomek</i>	70
43. Tabel 4.33. Perbandingan <i>Recall Score Adam SMOTETomek</i>	71

44. Tabel 4.34. Perbandingan <i>Precision Score</i> Adam SMOTETomek	71
45. Tabel 4.35. Perbandingan <i>F1-Score</i> Adam SMOTETomek	72
46. Tabel 4.36. Perbandingan <i>AUC Score</i> Adam SMOTETomek	72
47. Tabel 4.37. Perbandingan <i>Accuracy Score</i> RMSprop ADASYN	73
48. Tabel 4.38. Perbandingan <i>Recall Score</i> RMSprop ADASYN	74
49. Tabel 4.39. Perbandingan <i>Precision Score</i> RMSprop ADASYN	74
50. Tabel 4.40. Perbandingan <i>F1-Score</i> RMSprop ADASYN	75
51. Tabel 4.41. Perbandingan <i>AUC Score</i> RMSprop ADASYN	75
52. Tabel 4.42. Perbandingan <i>Accuracy Score</i> RMSprop BorderlineSMOTE	76
53. Tabel 4.43. Perbandingan <i>Recall Score</i> RMSprop BorderlineSMOTE	77
54. Tabel 4.44. Perbandingan <i>Precision Score</i> RMSprop BorderlineSMOTE ...	77
55. Tabel 4.45. Perbandingan <i>F1-Score</i> RMSprop BorderlineSMOTE	78
56. Tabel 4.46. Perbandingan <i>AUC Score</i> RMSprop BorderlineSMOTE	78
57. Tabel 4.47. Perbandingan <i>Accuracy Score</i> RMSprop SMOTETomek	79
58. Tabel 4.48. Perbandingan <i>Recall Score</i> RMSprop SMOTETomek	79
59. Tabel 4.49. Perbandingan <i>Precision Score</i> RMSprop SMOTETomek	80
60. Tabel 4.50. Perbandingan <i>F1-Score</i> RMSprop SMOTETomek	80
61. Tabel 4.51. Perbandingan <i>AUC Score</i> RMSprop SMOTETomek	81
62. Tabel 4.52. Rekapitulasi Perbandingan <i>Accuracy Score</i>	82
63. Tabel 4.53. Rekapitulasi Perbandingan <i>Recall Score</i>	82
64. Tabel 4.54. Rekapitulasi Perbandingan <i>Precision Score</i>	83
65. Tabel 4.55. Rekapitulasi Perbandingan <i>F1-Score</i>	84
66. Tabel 4.56. Rekapitulasi Perbandingan <i>AUC Score</i>	85
67. Tabel 4.57. Rekapitulasi Score Tertinggi <i>Metric Performance</i>	86
68. Tabel 4.58. Parameter Arsitektur DNN Yang Diusulkan	87
69. Tabel 4.59. <i>Confusion Matrix</i> Model yang diusulkan	88
70. Tabel 4.60. <i>Classification Report</i>	89
71. Tabel 4.61. Pengujian Aplikasi yang telah di-deployment	96
72. Tabel 4.62. Perbandingan <i>Metrics Performance</i>	97

DAFTAR GAMBAR

	Halaman
1. Gambar 2.1. Penerapan <i>SMOTETomek link</i>	19
2. Gambar 2.2. Arsitektur <i>Neural Network</i>	20
3. Gambar 2.3. <i>Artificial Neuron</i>	23
4. Gambar 2.4. <i>ReLU Function</i>	25
5. Gambar 2.5. <i>Sigmoid Function</i>	26
6. Gambar 2.6. <i>Dropout Layer</i>	27
7. Gambar 3.1. Tahapan Penelitian	38
8. Gambar 4.1. <i>Feature Importance</i> dengan <i>RandomForest</i>	48
9. Gambar 4.2. <i>Class Target Distribution</i>	49
10. Gambar 4.3. Distribusi masing-masing <i>features</i>	50
11. Gambar 4.4. <i>Effect of Body Mass Index, Age and Gender on Stroke</i>	51
12. Gambar 4.5. <i>Effects of Age and Smoking Status on Stroke</i>	51
13. Gambar 4.6. <i>Target Class setelah SMOTE</i>	55
14. Gambar 4.7. Perbandingan <i>Accuracy</i> dan <i>F1-Score</i>	85
15. Gambar 4.8. Arsitektur <i>Deep Neural Network (DNN)</i>	87
15. Gambar 4.9. Kurva <i>ROC</i> Arsitektur <i>DNN</i> yang Diusulkan	91
16. Gambar 4.10. Grafik <i>Accuracy Validation and Training History</i> <i>before Dropout</i>	92
17. Gambar 4.11. Grafik <i>Accuracy Validation and Training History</i> <i>after Dropout</i>	92
18. Gambar 4.12. Grafik <i>Logarithmic Loss Validation and Training History</i> <i>before Dropout</i>	93
19. Gambar 4.13. Grafik <i>Logarithmic Loss Validation and Training History</i> <i>after Dropout</i>	94
20. Gambar 4.14. Aplikasi Prediksi Stroke	95
21. Gambar 4.15. Perbandingan <i>Metrics Performance</i>	97

DAFTAR RUMUS

	Halaman
Rumus 2.1. <i>Mean Imputation</i>	11
Rumus 2.2. <i>Normalizer</i>	12
Rumus 2.3. <i>MinMaxScaler</i>	12
Rumus 2.4. <i>StandarScaler</i>	13
Rumus 2.5. <i>RobustScaler</i>	13
Rumus 2.6. <i>Synthetic Minority Oversampling Technique</i>	14
Rumus 2.7. <i>Class Balance Degree</i>	15
Rumus 2.8. <i>Number of Instance Synthetic Records to be Generated</i>	16
Rumus 2.9. <i>Ratio Euclidean Distance in n Dimension Space</i>	16
Rumus 2.10. <i>Number of Instance Synthetic Records to be Generated for i</i>	16
Rumus 2.11. <i>Instance Synthetic Records to be Generated</i>	16
Rumus 2.12. <i>Borderline SMOTE Algorithm</i>	17
Rumus 2.13. <i>Calculate Error between Target and Output</i>	21
Rumus 2.14. <i>Calculate Error Derivative</i>	21
Rumus 2.15. <i>Reformulate Error Derivative $Y_o \rightarrow Z_o$</i>	22
Rumus 2.16. <i>Reformulate Error Derivative $Z_o \rightarrow y_h$</i>	22
Rumus 2.17. <i>Binary Loss Function</i>	22
Rumus 2.18. <i>Rectified Linear Unit Activation Function</i>	24
Rumus 2.19. <i>Sigmoid Activation Function</i>	25
Rumus 2.20. <i>Sigmoid Derivative Activation Function</i>	25
Rumus 2.21. <i>Stochastic Gradient Descent Algorithm</i>	28
Rumus 2.22. <i>Adam Algorithm</i>	29
Rumus 2.23. <i>Root Mean Squared Propagation Algorithm</i>	30
Rumus 2.24. <i>Accuracy</i>	33
Rumus 2.25. <i>Precision</i>	33
Rumus 2.26. <i>Recall</i>	34
Rumus 2.27. <i>F1-score</i>	34
Rumus 2.28. <i>Kappa</i>	34
Rumus 2.29. <i>Sensitivity</i>	35
Rumus 2.30. <i>Specificity</i>	35
Rumus 3.1. <i>StandarScaler</i>	41
Rumus 4.1. <i>Accuracy</i>	89
Rumus 4.2. <i>Precision Class 0</i>	89
Rumus 4.3. <i>Precision Class 1</i>	89
Rumus 4.4. <i>Weighted Average Precision</i>	90
Rumus 4.5. <i>Recall Class 0</i>	90
Rumus 4.6. <i>Recall Class 1</i>	90
Rumus 4.7. <i>Weighted Average Recall</i>	90
Rumus 4.8. <i>F1-score</i>	91
Rumus 4.9. <i>Kappa</i>	91
Rumus 4.10. <i>Sensitivity</i>	91
Rumus 4.11. <i>Specificity</i>	91

DAFTAR LAMPIRAN

	Halaman
Lampiran 1. <i>Confusion Matrix</i> Optimasi <i>Stochastic Gradient Descent</i>	109
Lampiran 2. <i>Confusion Matrix</i> Optimasi <i>Adam</i>	110
Lampiran 3. <i>Confusion Matrix</i> Optimasi <i>Root Mean Squared Propagation</i>	111
Lampiran 4. <i>Program Code</i>	112

BAB 1

PENDAHULUAN

1.1. Latar Belakang Masalah

Data statistik Organisasi Kesehatan Dunia (WHO) secara global, pada tahun 2019, 7 dari 10 penyebab utama kematian adalah penyakit yang tidak menular. Ketujuh penyebab ini memiliki kontribusi 44% dari semua kematian atau 80% dari 10 besar. Namun, secara keseluruhan semua penyakit tidak menular berkontribusi 74% kematian secara global pada tahun 2019. Penyebab kematian global teratas, dalam rangka jumlah total kematian, dikaitkan dengan tiga topik, yaitu: kardiovaskular (penyakit jantung iskemik, stroke), pernapasan (penyakit paru obstruktif kronis, infeksi pernapasan yang lebih rendah) dan kondisi neonatal - yang meliputi asfiksia kelahiran dan trauma kelahiran, sepsis neonatal dan infeksi, dan komplikasi kelahiran prematur. Pada tahun 2019, secara umum stroke menjadi penyebab utama kematian kedua setelah penyakit jantung iskemik[1].

Menurut Kementerian Kesehatan, karena penyakit stroke mempunyai dampak luas secara ekonomi dan sosial, sehingga digolongkan ke dalam penyakit katastropik. Berbagai jenis penyakit katastropik di Indonesia terjadi akibat gaya hidup yang tidak sehat. Data dari Kementerian Kesehatan, penyakit jantung menempati posisi tertinggi sebagai penyakit katastropik dengan biaya tertinggi, sementara kanker dan stroke berada pada urutan kedua dan ketiga. Berdasarkan data dari Badan Penyelenggara Jaminan Sosial (BPJS) Kesehatan, penyakit stroke pada tahun 2016 menghabiskan biaya pelayanan kesehatan sebesar 1,43 Trilyun, tahun 2017 naik menjadi 2,18 Trilyun dan tahun 2018 mencapai 2,56 Trilyun rupiah[2].

Stroke (*Cerebro-Vascular Accident* atau CVA) adalah sebuah kondisi ketika gangguan mendadak dalam aliran darah ke otak, yang disebabkan oleh penyumbatan (*stroke iskemik*) atau semburan (*hemoragik stroke*) pembuluh darah yang membawa oksigen dan nutrisi[3]. Stroke terjadi ketika darah berhenti mengalir ke bagian mana pun dari otak kita, merusak sel-sel otak. Efek stroke sangat tergantung dari bagian otak yang rusak dan jumlah kerusakan yang terjadi. Mengetahui bagaimana otak kita bekerja dapat membantu kita memahami stroke.

Faktor risiko umum untuk stroke termasuk jangka panjang hipertensi, *hiperglikemia*, *hiperlipidemia* dan tinggi tekanan, stres emosional, dll., dapat menyebabkan stroke. Hal ini diyakini bahwa, di bawah asumsi kedokteran, pencegahan dan deteksi faktor risiko ini sangat membantu untuk perawatan dini[4].

Kurangnya alat yang digunakan untuk menganalisis data penyakit stroke merupakan salah satu tantangan saat ini. [3]. Penyediaan peralatan yang dapat memprediksi risiko stroke akan sangat berkontribusi secara signifikan terhadap pencegahan dan perawatan dini[4]. Alat yang dapat disediakan salah satunya adalah dengan menggunakan *machine learning*. Penggunaan *machine learning* dapat dilakukan dengan memanfaatkan data historis pasien yang pernah mengidap penyakit stroke. Namun, volume data yang tinggi, *heterogenitas* dan kompleksitas yang terdapat pada data medis menjadi tantangan terbesar dalam memprediksi penyakit stroke. Oleh karena itu, penelitian terkait dengan penyakit stroke dalam rangka penyediaan alat yang dapat membantu secara medis akan terus dilakukan untuk mendapatkan hasil yang terbaik.

Seperti penelitian yang dilakukan oleh [4], dengan melakukan studi *feature* yang memiliki pengaruh terhadap prediksi stroke dengan menggunakan algoritma *machine learning*. Penelitian tersebut dilakukan dengan menggunakan *Chi-square* (Chi-2) untuk menemukan faktor afektif atau korelasi dengan penyakit stroke. Hasil yang diperoleh pada penelitian tersebut yaitu Algoritma *Decision Tree* mendapatkan nilai akurasi 72,10% dan nilai *f-measure* sebesar 74,29% untuk menemukan faktor afektif yang mempengaruhi prediksi stroke. S. Ray, K. Alshouli, A. Roy et al [5] juga menggunakan *Chi-square* untuk *feature selection* dan menggunakan 6 *feature* paling atas. Dengan menggunakan *Two-Class Boosted Decision Tree* menghasilkan nilai akurasi sebesar 96,8%. Penelitian lain yang telah dilakukan oleh [6], adalah dengan menggunakan teknik *deep learning* dan beberapa algoritma lainnya. Pada penelitian tersebut membandingkan ketiga model untuk memprediksi penyakit stroke, yaitu menggunakan teknik *deep learning*, *naïve bayes* dan *support vector machine*. Hasil dari penelitian tersebut menunjukkan bahwa penggunaan *deep learning* mendapatkan nilai *Mean Square Error* sebesar 0.2596.

Penelitian lain pada [7], dilakukan perankingan terhadap *features* dengan menggunakan *shaphiro-wilk algorithm* dan *pearson correlation* kemudian dilakukan *Recursive Feature Elimination with Cross Validation*. Hasil tersebut dimasukkan pada beberapa algoritma sebagai estimator untuk memilih *features* kuat yang penting bagi prediksi stroke. Selanjutnya *features* penting yang telah dipilih dilakukan pemodelan dengan menggunakan *Decision Trees-Classifier* dan *simple deep learning*. Hasilnya menunjukkan bahwa dengan menggunakan pendekatan *machine learning* melebihi kemampuan profesional dalam memprediksi stroke. Selain itu, pada penelitian [8], menggunakan data psikologi pasien dan *Artificial Neural Network (ANN)* pada kondisi 1000 kali *cross validation* mendapatkan nilai akurasi 98%. A. Fitri, N. Masruriyah, T. Djatna et al [3], juga menggunakan *ANN* untuk melakukan prediksi penyakit stroke. Hasil penelitian dengan model *ANN* ini mendapatkan nilai akurasi sebesar 94,97%.

Selanjutnya pada penelitian [9], dengan dataset 15099 pasien menggunakan *Principal Component Analysis (PCA)* dan pendekatan *Deep Neural Network (DNN)* untuk mendeteksi stroke berdasarkan catatan medis. Pada penelitian tersebut membandingkan hasil kombinasi *PCA* dan *DNN* dengan lima algoritma lainnya, yaitu *Random Forest*, *AdaBoostClassifier*, *GaussianNB*, *KNeighborsClassifier* dan *Support Vector Classifier*. Metode *PCA* dan *DNN* dapat menghasilkan nilai *Area Under the Curve (AUC)* sebesar 83,48%.

Penelitian terkini yang dilakukan terkait dengan prediksi stroke adalah penelitian [10], dataset yang digunakan terdiri atas 43400 pasien dengan kelas imbalance dan dilakukan preprosesing menggunakan *Synthetic Minority Oversampling Technique (SMOTE)*. Pemodelan dilakukan dengan menggunakan *Naive Bayes*, *Logistic Regression*, *Decision Tree*, *Random Forest* and *Gradient Boosting*. Hasil dari pengujian secara *distributed environment* didapatkan nilai tertinggi menggunakan algoritma *Gradient Boosting* dengan nilai akurasi 94,49%.

Penelitian [10] tidak menggunakan variasi *SMOTE* untuk menangani ketidakseimbangan kelas dan memberikan saran untuk penelitian selanjutnya menggunakan *deep learning model*. Penelitian ini menggunakan beberapa teknik *SMOTE* yaitu *Adaptive Synthetic (ADASYN)*, *BorderlineSMOTE*, dan

SMOTETomek. Penelitian ini menggunakan *deep learning model* atau algoritma *Neural Network (NN)* yang dioptimasi parameternya. Penggunaan teknik *SMOTE* untuk menangani ketidakseimbangan kelas dengan *Tomek link* sebagai metode untuk membersihkan data (*data cleaning method*) dapat meningkatkan *performance* model klasifikasi lebih baik dibandingkan menggunakan teknik *ADASYN* dan *BoderlineSMOTE*.

1.2. Identifikasi Masalah

Berdasarkan analisa pada latar belakang permasalahan tersebut, maka pada penelitian ini mengidentifikasi masalah yang dirumuskan dijadikan sebagai objek penelitian, antara lain:

1. Bagaimana pengaruh penggunaan berbagai variasi *SMOTE* terhadap model algoritma *NN*?
2. Bagaimana pengaruh optimasi *parameter* terhadap *DNN* dalam mendapatkan nilai akurasi untuk klasifikasi?

1.3. Tujuan Penelitian

Adapun tujuan dari penelitian yang dilakukan pada antara lain sebagai berikut:

1. Membuat model arsitektur *DNN* dengan parameter yang tepat sehingga dapat digunakan memprediksi stroke untuk memenuhi kebutuhan medis.
2. Memilih teknik *SMOTE* yang tepat untuk menangani ketidakseimbangan kelas sehingga mampu meningkatkan tingkat akurasi algoritma *NN* untuk memprediksi stroke.

1.4. Ruang Lingkup Penelitian

Penelitian ini membatasi pembahasan atau ruang lingkup yang ada dalam penulisan tesis ini. Ruang lingkup yang dilakukan pada penelitian ini meliputi:

1. Proses klasifikasi stroke hanya menggunakan dataset *healthcare-dataset-stroke-data* yang bersumber dari *kaggle*.

2. Model algoritma yang digunakan pada penelitian ini adalah tiga model *DNN* dengan menggunakan *Python 3.8.8*.
3. Perbandingan untuk teknik *handling imbalance class* hanya menggunakan *Adaptive Synthetic (ADASYN)*, *Borderline SMOTE* dan *SMOTETomek*.
4. Nilai atau *score* yang digunakan untuk perbandingan/evaluasi *performance* antar teknik *handling imbalance class* adalah *score* prediksi *data testing*.

1.5. Kontribusi Tesis

Penanganan ketidakseimbangan kelas sudah menjadi hal umum dengan menggunakan teknik *SMOTE*. Namun, sedikit pekerjaan yang telah dilakukan pada penelitian ini dan pada penelitian sebelumnya, untuk dataset yang sama, belum dipertimbangkan secara komprehensif yaitu melakukan penanganan ketidakseimbangan kelas menggunakan teknik *SMOTETomek* dan menggunakan algoritma *NN*. Penanganan ketidakseimbangan kelas dengan teknik *SMOTETomek* dan pemodelan algoritma *NN* mendapatkan nilai akurasi lebih baik.

Model arsitektur *DNN* yang digunakan terdiri dari 5 *hidden layer* dengan *dropout* 0.2, fungsi aktivasi pada *input* dan *hidden layer* menggunakan *Rectified Linear Unit (ReLU)* dan *Sigmoid* untuk *output layer*, jumlah *epoch* 500 dan *batch size* 96. Optimasi yang digunakan *Adam* dengan *learning rate* 0.001.

1.6. Sistematika Penulisan

Sistematika penulisan tesis mengenai prediksi stroke menggunakan algoritma *NN* dan *SMOTE*, disusun sebagai berikut:

BAB 1: PENDAHULUAN

Pada bab ini berisi uraian fakta-fakta berkaitan permasalahan yang akan diteliti beserta tujuan serta ruang lingkup yang menjadi panduan dalam proses penelitian.

BAB 2: LANDASAN/KERANGKA PEMIKIRAN

Pada bab ini akan dijelaskan tentang landasan-landasan teoritis yang digunakan untuk kegiatan penelitian dan uraian sistematis dari penelitian-penelitian terkait serta kerangka pemikiran yang digunakan dalam melakukan kegiatan penelitian.

BAB 3: METODOLOGI PENELITIAN

Pada bab ini membahas metode yang digunakan dalam penelitian dan penjelasan tahapan yang dilakukan pada kegiatan penelitian. Disamping itu, akan dijelaskan model algoritma *NN* yang digunakan untuk kegiatan eksperimen.

BAB 4: HASIL PENELITIAN DAN PEMBAHASAN

Pada bab ini akan menyajikan dan membahas hasil penelitian dengan berbagai teknik *SMOTE* dan pemodelan menggunakan algoritma *NN*. Model algoritma *NN* yang dipilih akan dikembangkan dalam bentuk aplikasi dan *metrics performance* yang paling baik akan dibandingkan dengan hasil yang telah dilakukan oleh peneliti sebelumnya.

BAB 5: PENUTUP

Pada bab ini berisi tentang kesimpulan dari bab-bab sebelumnya dan saran untuk penelitian selanjutnya.

BAB 2

LANDASAN/KERANGKA PEMIKIRAN

2.1. Tinjauan Studi

Menurut [11], stroke (*Cerebro-Vascular Accident* atau CVA) merupakan sebuah kondisi ketika darah yang mengalir ke otak terganggu secara mendadak, penyebabnya dapat berupa penyumbatan (stroke iskemik) atau semburan (stroke hemoragik) pembuluh darah yang membawa oksigen dan nutrisi. Selain itu, menurut [12], beberapa penyebab stroke lainnya yaitu: jenis kelamin, risiko factor genetic, risiko hipertensi, risiko rokok, risiko sakit jantung, risiko alkohol, risiko hiperkolesterolemia, dan risiko diabetes. Dampak dari stroke dapat menyebabkan gangguan daya pikir, perubahan mental, kelumpuhan, kesadaran, konsentrasi, gangguan komunikasi, gangguan emosional, kehilangan indera rasa, kemampuan belajar dan fungsi intelektual lainnya[13]. Kondisi ini dapat menyebabkan terjadinya kecacatan dalam jangka panjang dan bahkan kematian[14].

Meskipun stroke dapat terjadi pada usia berapapun, seperempat stroke terjadi pada orang yang berusia di bawah 65 tahun. Hipertensi merupakan penyebab utama stroke[5]. Apabila faktor dominan penyakit stroke dapat dideteksi lebih dini, maka penyakit stroke akan dapat dicegah[15]. Oleh karena itu, tersedianya alat untuk melakukan deteksi dini potensi penyakit stroke menjadi sangat penting. Selain itu, peran keluarga menjadi sangat penting dalam mengendalikan penyebab utama terhadap stroke termasuk selama masa penyembuhan apabila sudah terjadi[13].

Dalam penyusunan laporan ini, peneliti telah melakukan studi literatur penelitian-penelitian yang pernah dilakukan sebelumnya terkait dengan prediksi stroke. Pada penelitian-penelitian tersebut digunakan berbagai algoritma klasifikasi dan berbagai metode, baik pada tahapan *pre-processing* maupun melakukan penyetelan *hyperparameter* untuk mendapatkan *performance model* yang lebih baik. Berikut beberapa penelitian yang telah dilakukan terkait dengan prediksi penyakit stroke, sebagaimana disajikan pada Tabel 2.1:

1. Penelitian [6], yang berjudul “*Prediction of Stroke Using Deep Learning Model*”. Penelitian ini dilakukan dengan menggunakan algoritma *Deep*

- LearningTechnique*, *Naïve Bayes* dan *Support Vector Machine*. Penelitian membandingkan ketiga model untuk memprediksi penyakit stroke. *Deep Learning* merupakan teknik yang paling cocok untuk menghasilkan kumpulan data penyakit jantung untuk analisis prediktif penyakit stroke. Penelitian ini menunjukkan nilai *Mean Square Error* (MSE) sebesar 0,2596.
2. Penelitian [9], yang berjudul “*The Use of Deep Learning to Predict Stroke Patient Mortality*”. Penelitian menggunakan *Principal Component Analysis* (PCA) untuk menampilkan penskalaan quartile yang dapat mengekstrak fitur yang relevan dari medical records. Penelitian ini membandingkan PCA dan DNN dengan lima algoritma lainnya yaitu *Random Forest*, *AdaBoostClassifier*, *GaussianNB*, *KNeighborsClassifier* dan *Support Vector Classifier*. Metode PCA dan DNN dapat menghasilkan nilai *sensitivity*, *specificity* dan *AUC* sebesar 64.32%, 85.56% dan 83.48%. Metode ini dapat digunakan tidak hanya untuk memprediksi stroke dengan menggunakan data terbatas, tetapi juga dapat digunakan untuk memprediksi penyakit lainnya.
 3. Penelitian [4], yang berjudul “*A Study of Features Affecting on Stroke Prediction Using Machine Learning*”. *Chi-squared* diadopsi untuk menemukan faktor afektif stroke. Empat faktor stroke yang paling afektif difokuskan untuk mengetahui risiko stroke. Dari penelitian ini, menunjukkan bahwa faktor-faktor tertentu lebih afektif daripada penyakit untuk mendeteksi stroke dan *Decision Tree* adalah pengklasifikasi terbaik dengan *accuracy* dan *f-measure score* sebesar 72.10% dan 74.29%.
 4. Penelitian [7], yang berjudul “*Automated Ischemic Stroke Subtyping Based on Machine Learning Approach*”. Metode yang dilakukan dengan cara meranking *features* dengan *Shapiro-Wilk algorithm* dan *Pearson Correlation* kemudian melakukan *Recursive Feature Elimination with Cross Validation* (RFECV). Pemilihan *feature* yang penting menggunakan *Extra Trees-Classififer* dan pembuatan *training model* menggunakan *deep learning* yang sederhana dapat mendapatkan model yang lebih baik daripada yang dilakukan manusia secara profesional. Penelitian ini menggunakan algoritma *Linear*

SVC, Random-Forest-Classififer, Extra-Trees-Classififer, AdaBoost-Classififer, dan Multinomial-Naïve-Bayes Classififer.

5. Penelitian [8], yang berjudul “*Artificial Neural Network Application to the Stroke Prediction*”. Penelitian menggunakan data fisiologis pasien. Menerapkan *Levenberg Marquardt Algorithm* dan *Scaled Conjugate Gradient (SCG) Algorithm* untuk melatih model. Dalam kondisi 1000 kali *cross-validation* dapat memperoleh 98% *accuracy score*.
6. Penelitian [5], yang berjudul “*Chi-Squared Based Feature Selection for Stroke Prediction using AzureML*”. Metode *Chi-squared* digunakan untuk memilih *feature* dan menggunakan enam *feature* teratas. Metode algoritma yang digunakan *modeling* adalah *Boosted Decision Tree* dan *Ensemble Learning*. Hasil akurasi kedua metode algoritma adalah 96.8%. Hasil penelitian ini menunjukkan bahwa dengan memilih fitur yang tepat, dapat meningkatkan akurasi secara signifikan untuk prediksi stroke, dan juga membutuhkan lebih sedikit waktu untuk melatih model.
7. Penelitian [3], yang berjudul “*Predictive Analytics For Stroke Disease*”. Penelitian ini menggunakan *Artificial Neural Network (ANN)* dengan menyajikan model semata-mata untuk memprediksi penyakit stroke berdasarkan *medical records*. Hasil dari model ANN yang disajikan pada penelitian ini yaitu *accuracy score* sebesar 95.15%.
8. Penelitian [10], yang berjudul “*Stroke Prediction Using Machine Learning in a Distributed Environment*”. Untuk analisisnya, penelitian ini menggunakan lima algoritma yaitu *Naïve Bayes*, *Logistic Regression*, *Decision tree*, *Random Forest* dan *Gradient Boosting*. Algoritma tersebut dilakukan optimasi pada *learning rate*, *depth of tree*, *number of trees* dan *minimum sample split*. Penelitian ini melakukan analisis *features* menggunakan *plotting univariate* dan *multivariate* untuk melihat korelasi diantara berbagai *features*. Penyetelan berbagai *hyperparameter* pada algoritma *Gradient Boosting* mendapatkan *Accuracy*, *Precision*, *Recall* dan *F1-score* sebesar 0.9449, 0.9453, 0.9449 dan 0.9448.

Tabel 2.1. Penelitian Terkait dengan Stroke

No	Penulis	Judul	Metode	Performance
1	Maihul Rajora et all (2021)[10]	<i>Stroke Prediction Using Machine Learning in a Distributed Environment</i>	<i>Gradient Boosting</i>	Accuracy: 0.9449 Precision: 0.9453 Recall: 0.9449 F1-score: 0.9448
2	Anis Fitri Nur Masruriyah et all (2020)[3]	<i>Predictive Analytics for Stroke Disease</i>	<i>Artificial Neural Network</i>	Accuracy: 95.15%
3	Sujan Ray et all (2020)[5]	<i>Chi-Squared Based Feature Selection for Stroke Prediction using AzureML</i>	<i>Two-Class Boosted Decision Tree (Different Dataset)</i>	Accuracy: 96.8%
4	Chun-Cheng Peng et all Liao (2020)[8]	<i>Artificial Neural Network Application to the Stroke Prediction</i>	<i>Artifial Neural Network (Different Dataset and 1000 cross validation)</i>	Accuracy: 98%
5	Panida Songram and Chatklaw Jareanpon (2019)[4]	<i>A Study of Features Affecting on Stroke Prediction Using Machine Learning</i>	<i>Decision tree</i>	Accuracy: 72.10% F-measure: 74.29%
6	Songhee Cheon, Jungyoon Kim and Jihye Lim (2019)[9]	<i>The Use of Deep Learning to Predict Stroke Patient Mortality</i>	<i>Principal Component Analysis dan DNN</i>	AUC: 83.48%
7	Pattanapong Chantamit-o-pas and Madhu Goyal (2017)[6]	<i>Prediction of Stroke Using Deep Learning Model</i>	<i>Deep Learning Technique</i>	Mean Square Error (MSE): 0.2596.

2.2. Metode Preprocessing

2.2.1 Data Cleaning

Tujuan *data cleaning* adalah untuk menghapus *noise*, data yang tidak konsisten dan adanya kesalahan pada *dataset*. Salah satu bentuk *noise* adalah terjadinya nilai yang hilang (*missing values*). Salah satu teknik yang dapat digunakan untuk menangani *missing values* adalah dengan melakukan *imputation*. Beberapa pendekatan *imputation* sebagaimana dijelaskan sebagai berikut[16]:

a. *Simple Regression Imputation*

Simple Regression Imputation lebih banyak menggunakan informasi yang tersedia pada kumpulan data daripada substitusi rata-rata untuk menghasilkan nilai yang digunakan untuk imputasi. *Simple regression* melibatkan pembuatan kuadrat persamaan regresi, di mana observasi variabel yang hilang berfungsi sebagai variabel dependen dan variabel yang relevan dalam kumpulan data digunakan untuk memprediksi nilai yang hilang.

b. *Regression Imputation with Added Error Term*

Regression imputation with an added error menggunakan prosedur yang hampir sama dengan *simple regression* untuk memprediksi Y_{miss} [17]. Pendekatan *mean imputation*, *regression imputation* dan *stochastic regression* dilakukan berdasarkan persamaan:

$$v_i^* = a + X_i b + e_i^*, i = 1, \dots, N_m, \dots\dots\dots (2.1)$$

Dimana v_i^* adalah nilai yang diimputasi untuk hilangnya respons pada variabel v untuk contoh kasus i , X_i adalah vektor baris K -column dari observasi pada prediktor regresi K untuk kasus i dalam model imputasi, b adalah vektor kolom urutan K dari perkiraan koefisien regresi yang sesuai dengan variabel dalam X . e_i^* adalah residu yang diperkirakan dari regresi v pada X dan N_m adalah jumlah *tuple* yang perlu diimputasi.

2.2.2 *Feature Encoding*

Feature encoding merupakan proses mengubah nilai kategorikal pada suatu *feature* yang berbentuk label ke dalam bentuk numerik. *Label encoding* merupakan cara yang sangat sederhana untuk mengonversi nilai kategoris menjadi nilai numerik. *Label Encoding* hanyalah menetapkan nilai bilangan bulat ke setiap nilai yang mungkin dan berbeda dari variabel kategoris[18]. Pendekatan ini sangat sederhana dan melibatkan konversi setiap nilai pada kolom menjadi angka. Label

Encoding mengacu pada konversi label menjadi bentuk numerik sehingga dapat mengubahnya menjadi bentuk yang dapat dibaca mesin. Hal ini merupakan langkah penting *preprocessing* untuk dataset terstruktur dalam *supervised learning*. Tabel 2.1 menunjukkan ilustrasi hasil *Label Encoding*:

Tabel 2.2. *Illustration of Label Encoding*

<i>Smoke Status</i>	<i>Label Encoding</i>
Smoke	2
Never Smoke	1
Formely Smoke	0

2.2.3 Feature Scaling

Feature scaling dilakukan untuk mengubah vektor nilai pada *feature* menjadi format yang lebih cocok untuk *training*. Beberapa teknik *feature scaling* yang paling umum digunakan adalah *Normalizer*, *MinMaxScaler*, *StandardScaler*, dan *RobustScaler*[19].

Normalizer menskalakan data untuk nilai setiap sampel ke norma unit. Nilai yang ditransformasikan untuk *feature* x adalah:

$$z = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}} \dots \dots \dots (2.2)$$

Dimana x_i , y_i dan z_i adalah nilai-nilai pada *feature* x , y dan z .

MinMaxScaler menskalakan data sehingga semua nilai pada dataset antara 0 dan 1.

$$t = \frac{x_{min}}{x_{max} - x_{min}} \dots \dots \dots (2.3)$$

Dimana, t adalah nilai hasil transformasi

x adalah nilai original

x_{min} dan x_{max} adalah nilai minimum dan maksimum pada *feature* x .

StandardScaler mengubah dataset sehingga nilai rata-rata distribusi yang dihasilkan adalah nol dan simpangan bakunya adalah satu. Nilai yang ditransformasi diperoleh dengan mengurangi rata-rata nilai dari nilai asli dan membagi dengan simpangan baku. Rumus yang diberikan di bawah ini digunakan untuk transformasi.

$$z = \frac{x - \mu}{\sigma} \dots\dots\dots (2.4)$$

Dimana, z merupakan nilai *feature* hasil transformasi

x adalah nilai original

μ adalah nilai rata-rata

σ adalah nilai standard deviasi

RobustScaler menghapus *median* dan menskalakan data menurut rentang kuartil yang berkisar dari kuartil ke-25 hingga kuartil ke-75. Nilai yang ditransformasi dari kumpulan data relatif lebih besar dari skalar sebelumnya. Nilai data hasil transformasi di antara rentang $[-2, 3]$. Hasilnya seperti *minmaxscaler* tetapi menggunakan rentang inter kuartil bukan min max. Rumus untuk *feature scaling* dengan *RobustScaler* adalah sebagai berikut:

$$t = \frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)} \dots\dots\dots (2.5)$$

Dimana, t merupakan nilai *feature* hasil transformasi

x adalah nilai original

$Q_1(x)$ dan $Q_3(x)$ adalah *range* inter kuartil

2.2.4 Handling Class Imbalance

Permasalahan ketidakseimbangan kelas dapat diselesaikan dengan dua pendekatan. Kedua pendekatan tersebut adalah pendekatan pada level data dan pendekatan algoritma[20]. Pendekatan pada level data dilakukan dengan

menyeimbangkan distribusi kelas mayoritas dan minoritas melalui metode *under sampling*, *over sampling* atau kombinasi kedua metode (*hybrid*).

2.2.4.1 *Random Over Sampling*

Random Over Sampling merupakan proses penyeimbangan kelas minoritas dan mayoritas, dengan cara menduplikasi kelas minoritas sehingga sama dengan kelas mayoritas[21]. Data yang diduplikasi sama persis dari kelas minoritas sehingga dapat menyebabkan terjadinya *overfitting*, karena model tidak akan mampu mengenali pola data yang baru.

2.2.4.2 *Random Under Sampling*

Random Under Sampling merupakan proses penyeimbangan kelas mayoritas dengan kelas minoritas dengan cara menghapus kelas mayoritas sehingga distribusinya seimbang. Kelemahan *undersampling* yaitu hilangnya informasi yang berguna karena data yang dihapus dari kelas mayoritas[21].

2.2.4.3 *SMOTE*

Metode *SMOTE* diusulkan oleh Chawla et al. (2002)[22] dengan menghasilkan pengamatan sintetik untuk kelas minoritas. Untuk observasi kelas minoritas, observasi sintetik dihasilkan dalam rentang acak antara pengamatan dan *k-nearest neighbor* kelas minoritas. Prosedur ini dilakukan untuk setiap observasi kelas minoritas. Untuk *SMOTE*, jumlah *k-nearest neighbor* diatur ke angka 5. Meskipun *SMOTE* cukup efektif untuk meningkatkan akurasi klasifikasi data minoritas, tetapi masih ada masalah, antara lain terjadinya *overgeneralisasi*. Data sintetis yang dihasilkan oleh *SMOTE* masih dimungkinkan untuk menyebar pada data minoritas dan mayoritas, sehingga akan mengurangi kinerja klasifikasi. Rumus untuk menghasilkan data sintetis oleh *SMOTE* adalah sebagai berikut[20]:

$$x^{synthetic} = x^i + (x^j - x^i) \times \delta \dots\dots\dots (2.6)$$

Dimana $x^{synthetic}$ adalah data sintetis, x^i adalah instan kelas minoritas, x^j adalah instan yang dipilih secara acak dari K -nearest neighbor dari x^i instan minoritas, dan δ adalah vektor di mana setiap elemen adalah angka acak dari $[0,1]$.

a. ADASYN

Konsep yang sangat penting *Adasyn* menggunakan distribusi tertimbang untuk berbagai contoh kelas minoritas berdasarkan tingkat *difficulty learning*. *Adasyn* menghasilkan data sintetis untuk kelas minoritas yang lebih sulit dipelajari[23]. Metode *Adasyn* tidak hanya mengurangi bias pada proses pembelajaran yang disebabkan oleh distribusi ketidakseimbangan dataset sebenarnya, tetapi dapat juga secara adaptif menggeser batas keputusan untuk fokus mempelajari sampel yang sulit[24].

[Algorithm - ADASYN]

Input

(1) Dataset training D_{tr} dengan m sampel $\{X_i, y_i\}$, $i = 1, \dots, m$, dimana x_i adalah instan pada n space dimensi *feature* X dan $y_i \in Y = \{1, -1\}$ adalah identitas kelas label yang diasosiasikan dengan x_i . Definisikan m_s dan m_l sebagai jumlah sampel kelas minoritas dan jumlah sampel kelas mayoritas. Therefore, $m_s \leq m_l$ and $m_s + m_l = m$.

Procedure

(1) Menghitung tingkat keseimbangan kelas:

$$d = \frac{m_s}{m_l} \dots\dots\dots (2.7)$$

dimana $d \in (0, 1]$.

(2) Jika $d < d_{th}$ maka (d_{th} adalah ambang batas yang telah ditetapkan untuk rasio ketidakseimbangan kelas maksimum yang ditoleransi):

(a) Menghitung jumlah sampel data sintetis yang perlu dihasilkan untuk kelas minoritas:

$$G = (ml - ms) \times \beta \dots\dots\dots (2.8)$$

dimana $\beta \in [0,1]$ adalah parameter yang digunakan untuk menentukan tingkat keseimbangan yang diinginkan setelah pembuatan data sintetis. $\beta = 1$ berarti sudah terjadi keseimbangan dataset setelah proses generalisasi.

(b) Untuk setiap sampel $x_i \in minorityclass$, cari K nearest neighbors berdasarkan *Euclidean distance* pada n space dimensi, dan hitung *ratio* r_i yang didefinisikan:

$$r_i = \frac{\Delta_i}{K}, i \dots, m_s \dots\dots\dots (2.9)$$

dimana Δ_i adalah jumlah sampel pada K nearest neighbors dari x_i yang termasuk dalam kelas mayoritas, $r_i \in [0,1]$;

(c) Normalisasi r_i berdasarkan $\dot{r}_i = \frac{r_i}{\sum_{i=1}^{m_s} r_i}$, sehingga \dot{r}_i merupakan kepadatan distribusi ($\sum_i r_{i=1}$)

(d) Menghitung jumlah sampel data sintetis yang perlu dihasilkan untuk setiap sampel minoritas x_i :

$$g_i = \dot{r}_i \times G \dots\dots\dots (2.10)$$

dimana G adalah sampel data sintetis angka total yang perlu dihasilkan untuk kelas minoritas sebagaimana didefinisikan dalam persamaan (2.8).

(e) Untuk setiap sampel data kelas minoritas x_i , menghasilkan g_i sampel data sintetis sesuai dengan langkah-langkah berikut:

Kerjakan perulangan dari 1 ke g_i :

- (i) pilih satu data sampel minoritas secara acak, x_{zi} , dari K *nearest neighbors* untuk data x_i .
- (ii) Menghasilkan data sampel sintetis:

$$s_i = x_i + (x_{zi} - x_i) \times \lambda \dots\dots\dots (2.11)$$

dimana $(x_{zi} - x_i)$ adalah perbedaan vektor pada n *space* dimensi, dan λ adalah jumlah secara acak: $\lambda \in [0, 1]$.

Akhir perulangan

b. *Borderline SMOTE*

Untuk mendapatkan hasil prediksi yang lebih baik, selama proses training algoritma klasifikasi berusaha mempelajari area batas antar kelas. Sampel yang terdapat pada area batas antar kelas yang berdekatan sering kali salah diklasifikasikan dibandingkan sampel yang terdapat pada area yang berjauhan. Padahal, sampel pada area ini mungkin memiliki kontribusi yang besar untuk klasifikasi. Area ini sering kali disebut *borderline*. *Borderline SMOTE* didasarkan pada *SMOTE* untuk menghasilkan sampel sintetis minoritas[25]. Setelah itu, sampel sintetis baru yang dihasilkan di sepanjang garis antara sampel minoritas dan yang ditetapkan sebagai *neighbors* (k). Metode ini memperkuat sampel minoritas pada *borderline*. Secara umum proses yang dilakukan adalah dengan mencari tahu sampel minoritas pada garis perbatasan, kemudian sampel sintetis dihasilkan dari mereka dan ditambahkan ke *dataset original training*. *Pseudo-code Borderline SMOTE* sebagaimana disajikan sebagai berikut[26]:

[*Algorithm – Borderline SMOTE*] (2.12)

Input: T- Dataset training

M- Dataset sampel minoritas

r,k- Jumlah tetangga terdekat

s- Jumlah sampel sintetis yang memperhitungkan jumlah sampel original pada kelas yang diberikan

Output: Dataset sampel minoritas sintetis: M'

```

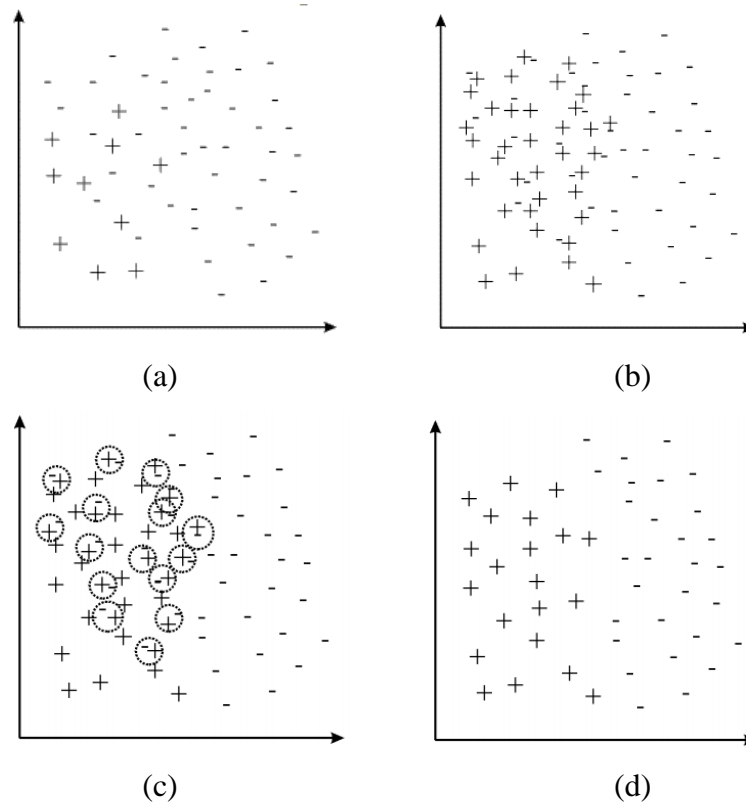
D =  $\phi$  //D merupakan dataset yang berisi sampel pada borderline
for all  $m_i$  in M do
     $N_{m_i} \leftarrow r$  tetangga terdekat dari  $m_i$  in T
    n  $\leftarrow$  jumlah sampel pada  $N_{m_i}$  dan bukan pada M
    if  $r/2 \leq n < r$  then //  $m_i$  merupakan sampel borderline
        add  $m_i$  to D
    end if
end for
 $M' = \phi$  //  $M'$  merupakan dataset yang berisi sampel sintetis
for all  $d_i$  in D do
     $N_{d_i} \leftarrow k$  tetangga terdekat dari  $d_i$  in M
    for i = 1 to s do
        m  $\leftarrow$  pilih sampel secara acak dari  $N_{d_i}$ 
         $d'_i \leftarrow d_i + p * (d_i - m)$  // p merupakan nomor acak
        in(0, 1),  $d'_i$  merupakan sampel sintetis
        add  $d'_i$  to  $M'$ 
    end for
end for
 $M' = M' \cup M$  //  $M'$  merupakan gabungan dari sampel minoritas dan sampel
    sintetis yang berada pada borderline
return  $M'$ 

```

c. *SMOTETomek*

SMOTETomek merupakan teknik penyeimbangan kelas berdasarkan *SMOTE* dengan melakukan pembersihan data yang melibatkan penghapusan tautan *Tomek* untuk pasangan sampel yang berada di area kedekatan satu sama lain tetapi milik kelas berlawanan[27]. Dengan demikian, bukan hanya menghapus sampel kelas mayoritas yang membentuk tautan *Tomek*, sampel *nearest neighbor* dari

kedua kelas dihapus[28]. Penerapan metode *SMOTETomek link* diilustrasikan pada Gambar 2.1. Pertama, yang asli kumpulan data (a) diseimbangkan dengan *SMOTE* (b), lalu *Tomek links* diidentifikasi (c) dan dihapus, sehingga menghasilkan dataset yang seimbang dengan batas klaster yang jelas (d)[29], sehingga dataset yang dihasilkan tidak ada antar kelas yang tumpang tindih.



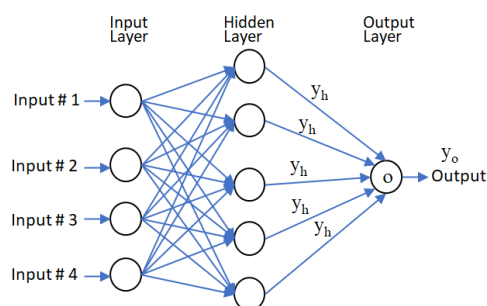
Gambar 2.1. Penerapan *SMOTETomek link*

2.3. Arsitektur dan *Hyperparameter Deep Learning*

2.3.1 *Neural Network (NN)*

NN merupakan sebuah sistem pemrosesan informasi yang memiliki karakteristik serupa dengan jaringan neural pada otak manusia. *NN* adalah pemodelan otak manusia dengan definisi paling sederhana dan blok bangunan Neuron[30]. Prinsip kerja *NN* seperti pada otak manusia. *NN* terdiri dari lapisan-lapisan neuron. Arsitektur paling sederhana *NN* setidaknya memiliki lapisan *input* dan lapisan *output*. Arsitektur *NN* sebagaimana diilustrasikan pada Gambar 2.2.

Pada gambar tersebut terdapat *input*, *input layer*, *hidden layer*, dan *output layer* yang terdiri atas lapisan-lapisan *neuron*. Arsitektur pada gambar tersebut memiliki 3 buah *neuron* pada *input layer*, 5 *neuron* pada *hidden layer* dan 1 buah *neuron* pada *output layer*. *Neuron* pada *input layer* tidak memiliki *activation function*, sedangkan *neuron* pada *hidden layer* dan *output layer* memiliki *activation function* yang bisa jadi berbeda tergantung sesuai dengan problem yang dihadapi. Pada *hidden layer* dan *output layer* biasanya juga terdapat input lain yang dinamakan bias.



Sumber : (Jeff Heaton, 2015)

Gambar 2.2. Arsitektur *Neural Network*

Deep learning merupakan bagian dari *machine learning* yang berbentuk *Artificial Neural Network (ANN)*, algoritma yang digunakan terinspirasi oleh otak manusia, belajar dari sejumlah besar data. *Deep learning* memungkinkan mesin untuk memecahkan masalah yang kompleks, bahkan ketika menggunakan kumpulan data yang sangat beragam, tidak terstruktur, dan saling terkait. Sehingga dapat dikatakan bahwa *deep learning* merupakan pendekatan dalam *Artificial Intelligent (AI)*. [31]. *Deep Neural Networks (DNN)* merupakan *NN* yang memiliki *hidden layer* lebih dari satu [31]. Metode *deep learning* harus menunjukkan hasil yang lebih baik dengan data dan fitur yang dipilih dengan benar. Selain menggunakan data dari satu sumber juga perlu menggunakan data dari beberapa sumber terdekat. Perbandingan jumlah lapisan dan neuron yang berbeda di setiap lapisan serta beberapa fungsi aktivasi di *DNN* untuk mendapatkan akurasi yang lebih tinggi[31].

Proses training pada NN secara garis besar terbagi menjadi 2 (dua), yaitu: *Forward Pass (Forwardpropagation)* dan *Backward Pass (Backpropagation)*. *Forwardpropagation* merupakan proses membawa data pada input melewati tiap neuron pada *hidden layer* sampai kepada *output layer*. Setelah sampai pada *output layer* akan dihitung *error*nya. Hasil hitung *error* akan digunakan untuk meng-*update weight* dan bias dengan learning rate tertentu dengan algoritma *backpropagation*. Kedua proses ini akan berulang sampai mendapatkan nilai *weight* dan bias mendapatkan hasil *output* dengan *error* paling kecil.

2.3.2 Backpropagation

Backpropagation adalah metode inti pembelajaran untuk *deep learning*. Prinsip algoritma *Backpropagation* tidak lain adalah akan menyesuaikan *weight* dan bias berdasarkan hasil hitung *error* pada saat *forwardpropagation*. *Backpropagation of errors* pada dasarnya hanya turunan gradien[32]. Turunan gradien mengacu pada perhitungan gradien pada setiap *weight NN* untuk setiap elemen *training*. Karena *NN* tidak akan mengeluarkan nilai seperti yang diharapkan atau sama persis seperti target, gradien setiap *weight* akan mengindikasikan cara memodifikasi setiap *weight* untuk mencapai *output* yang diharapkan. Jika *NN* mengeluarkan nilai *output* persis apa yang diharapkan atau sama dengan target, maka *gradien* untuk setiap *weight* akan 0. Hal ini menunjukkan bahwa tidak ada perubahan yang diperlukan pada *weight*[33].

Berikut akan diilustrasikan proses *backpropagation* berdasarkan Gambar 2.2. Untuk menjaga eksposisi sejelas mungkin, yang digunakan hanya dua indeks, seolah-olah setiap lapisan hanya memiliki satu *neuron*. Sebagaimana pada Gambar 2.2, subskrip *o* digunakan sebagai *output layer* dan *h* untuk *hidden layer*, *t* merupakan target dan *y* untuk prediksi. Yang perlu diperhatikan bahwa *z* adalah logit. Proses dimulai setelah adanya perhitungan *error* antara *output* dengan target. Yang dihitung sebagai berikut[32]:

$$E = \frac{1}{2} \sum_{o \in \text{Output}} (t_o - y_o)^2 \dots\dots\dots (2.13)$$

Proses pertama yang dilakukan adalah menghitung perbedaan *error* tersebut ke *error derivative*.

$$\frac{\partial E}{\partial y_o} = -(t_o - y_o) \dots\dots\dots (2.14)$$

Selanjutnya melakukan reformulasi *error derivative* sehubungan dengan y_o ke dalam *error derivative* berkaitan dengan z_o . Untuk proses ini digunakan aturan berantai (*chain rule*).

$$\frac{\partial E}{\partial z_o} = \frac{\partial y_o}{\partial z_o} \frac{\partial E}{\partial y_o} = y_o(1 - y_o) \frac{\partial E}{\partial y_o} \dots\dots\dots (2.15)$$

Tahap terakhir yaitu menghitung *error derivative* sehubungan dengan y_h .

$$\frac{\partial E}{\partial y_h} = \sum_o \frac{dz_o}{dy_h} \frac{\partial E}{\partial z_o} = \sum_o w_{ho} \frac{\partial E}{\partial z_o} \dots\dots\dots (2.16)$$

Tahapan perubahan dari $\frac{\partial E}{\partial y_o}$ ke $\frac{\partial E}{\partial y_h}$ merupakan proses *backpropagation*. Proses ini dilakukan pada semua *neuron* dan *layer*.

2.3.3 Loss functions

Terjadinya *error* tidak lain adanya perbedaan antara nilai target aktual dan nilai yang diprediksi. Hal ini juga dinamakan *loss*. Untuk memperkirakan nilai kerugian dengan benar diperlukan fungsi kerugian (*loss function*) yang sesuai. Selama proses pelatihan *NN*, bobot dan bias harus diperbarui pada setiap evaluasi untuk mengurangi nilai kerugian dalam evaluasi berikutnya. *Cost function* merupakan kunci untuk menyesuaikan *weight NN* agar model pembelajaran menjadi lebih baik [34]. *Binary Crossentropy* [15] merupakan *loss function* yang mengidentifikasi kerugian berdasarkan nilai probabilitas. Nilai probabilitas terletak antara 0 dan 1. Model sempurna dengan kerugian nol akan memiliki nilai

probabilitas 0. Berikut merupakan rumus yang digunakan untuk klasifikasi biner[34].

$$J_{bce} = -\frac{1}{M} \sum_{m=1}^M [y_m \times \log(h_{\theta}(x_m)) + (1 - y_m) \log(1 - h_{\theta}(x_m))] \dots\dots\dots (2.17)$$

dimana,

M adalah jumlah sampel training

y_m adalah label target untuk jumlah training m

x_m adalah input untuk sampel training m

h_{θ} adalah model NN dengan *weight* θ

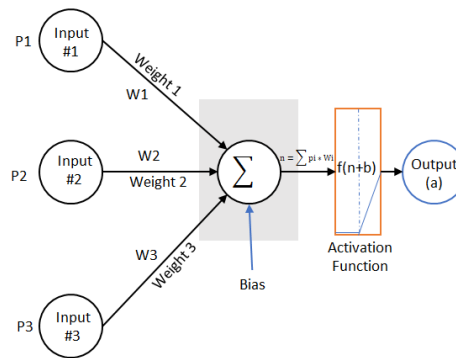
Selama *backpropagation*, turunan parsial dari *lost function* dihitung untuk setiap *weight NN*. *Backpropagation* secara berulang akan menyesuaikan *weight NN* untuk menghasilkan model dengan *loss* yang lebih rendah[34].

2.3.4 Hyperparameter Model

Hyperparameters adalah variabel yang menentukan struktur jaringan dan variabel yang menentukan bagaimana jaringan dilatih.

2.3.4.1 Neuron

Setiap elemen yang memegang input disebut '*neuron*'. *Neuron-neuron* membentuk lapisan yang dinamakan *layer*. Setiap *neuron* dari *input layer* terhubung ke setiap *neuron* dari *hidden layer*, tetapi *neuron* dari layer yang sama tidak saling berhubungan[32]. Proses pembelajaran pada neuron hanyalah modifikasi atau pembaruan *weight* dan bias selama pelatihan dengan *backpropagation*. Proses pembelajaran pada neuron sebagaimana diilustrasikan pada Gambar 2.3[33].



Sumber : (Jeff Heaton, 2015)

Gambar 2.3 *Artificial Neuron*

2.3.4.2 Layer

Meskipun tidak ada ketentuan secara pasti dalam menentukan jumlah layer pada *NN*, namun jumlah *layer* harus dipilih dengan tepat karena jumlah *layer* yang sangat tinggi dapat menjadi masalah seperti masalah *overfitting* dan jumlah yang lebih rendah dapat menyebabkan model memiliki bias tinggi dan mengakibatkan kinerja model menjadi tidak baik. Pemilihan jumlah *layer* sangat tergantung pada ukuran data yang digunakan untuk pelatihan.

2.3.4.3 Activation Function

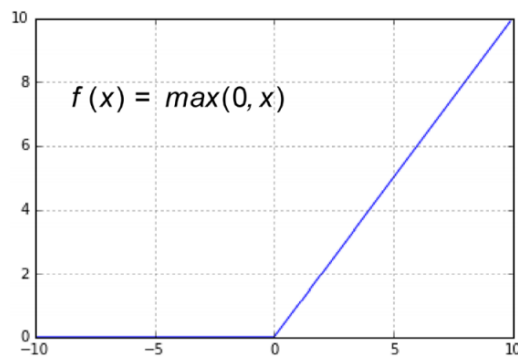
Salah satu yang mempengaruhi tingkat akurasi suatu model *NN* adalah tipe fungsi aktivasi (*activation function*) yang digunakan[35]. Fungsi aktivasi digunakan untuk mengenali pola/jaringan *back-propagation*, prediksi/aproksimasi linear, dan pengambilan keputusan. Fungsi aktivasi yang paling sering digunakan saat ini adalah *Rectified Linear Unit (ReLU)* dan *Sigmoid*.

a. *ReLU*

ReLU merupakan fungsi aktivasi *non-linear* yang banyak digunakan dalam *NN*[35]. Fungsi *ReLU* tidak memiliki turunan di 0. Namun karena konvensi, gradien 0 disubstitusi ketika x adalah 0. Hal ini berarti akan mengembalikan 0 jika inputnya negatif. Hal tersebut secara persamaan dapat disajikan sebagai berikut:

$$f(x) = \max(0, x) \dots \dots \dots (2.18)$$

Persamaan tersebut dalam bentuk grafik sebagaimana disajikan pada Gambar 2.4. Penggunaan fungsi *ReLU* menjadikan model *NN* berkinerja lebih baik daripada fungsi aktivasi lainnya untuk sebagian besar kasus. Fungsi *ReLU* hanya digunakan pada *hidden layer* dan tidak digunakan untuk *outer layer*.



Sumber : (Nikhil Ketkar, 2017)

Gambar 2.4. *ReLU Function*

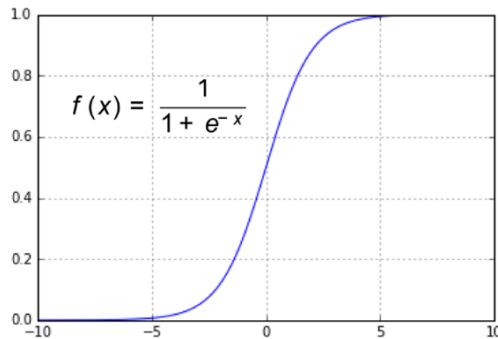
b. *Sigmoid*

Sigmoid merupakan fungsi aktivasi termasuk yang paling banyak digunakan karena *non-linear*. *Sigmoid* digunakan ketika melakukan prediksi dalam bentuk biner. Fungsi *sigmoid* mengubah fungsi nilai dalam rentang 0 sampai dengan 1[35]. Ini dapat didefinisikan sebagai:

$$f(x) = \frac{1}{1+e^x} \dots \dots \dots (2.19)$$

Fungsi *sigmoid* terus diferensiasi dan fungsinya berbentuk S yang *smooth* sebagaimana disajikan pada Gambar 2.5. Fungsi *sigmoid* tidak simetris tentang nol yang berarti bahwa *sign* semua nilai *output neuron* akan sama. Masalah ini dapat diselesaikan dengan meningkatkan fungsi sigmoid. Turunan dari fungsi 2.18 Adalah sebagai berikut:

$$f'(x) = 1 - \text{sigmoid}(x) \dots\dots\dots (2.20)$$



Sumber: (Nikhil Ketkar, 2017)

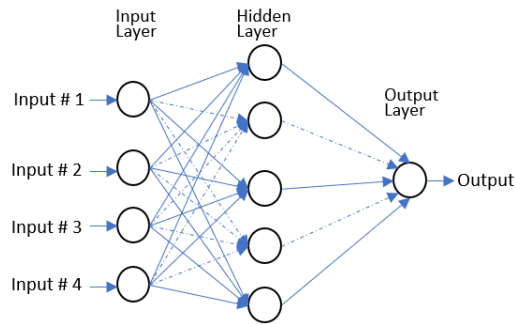
Gambar 2.5. *Sigmoid Function*

2.3.4.4 Batch Normalization

Tingkat pembelajaran (*learning rate*) yang terlalu tinggi dapat mengakibatkan gradien yang naik secara ekstrem atau malah turun secara tajam, serta terjebak dalam kondisi yang buruk. *Batch Normalisasi* membantu mengatasi masalah ini [36]. *Batch Normalization* secara signifikan meningkatkan waktu pelatihan, meningkatkan akurasi hanya dengan menurunkan *initial weights* dan sedikit membantu regularisasi[37].

2.3.4.5 Dropout

Tingginya jumlah parameter membuat sangat rentan terhadap *overfitting*, sehingga dalam praktiknya membutuhkan metode regularisasi. *Dropout* tidak diterapkan ke lapisan output. Probabilitas dapat bervariasi untuk setiap lapisan, (A. Labach et al, 2019) merekomendasikan nilai $p = 0,2$ untuk lapisan input dan $p = 0,5$ untuk lapisan tersembunyi. Neuron di lapisan output tidak di-*dropout*[38]. (C. Garbin et al, 2020) merekomendasikan untuk nilai p 0.1 pada *input layer* dan nilai p antara 0.5 dan 0.8 untuk *hidden layer*. Menambahkan *dropout* dapat menurunkan akurasi[37]. Ilustrasi *dropout layer* sebagaimana disajikan pada Gambar 2.6.



Sumber : (Jeff Heaton, 2015)

Gambar 2.6. *Dropout Layer*

2.3.4.6 Epoch

Epoch pada pembelajaran *back-propagation* merupakan sekali proses pembaruan bobot (*weight update process*) atau sekali proses iterasi. Untuk setiap *epoch*, algoritma pembelajaran *back-propagation* membangun model jaringan dengan sekumpulan bobot (*weight*) yang berbeda[39]. Jumlah *epoch* merupakan berapa kali seluruh *data training* ditampilkan ke model dan memainkan peran penting dalam menentukan seberapa baik model cocok pada *data training*. Jumlah *epoch* yang tinggi dapat menyebabkan masalah generalisasi pada saat validasi. Jumlah *epoch* yang lebih rendah juga dapat membatasi potensi mendapatkan model yang lebih baik.

2.3.4.7 Batch size

Batch Size merupakan jumlah sampel data yang disebar ke *NN*. Berdasarkan pengamatan dalam praktik ketika menggunakan batch yang lebih besar terjadi degradasi dalam kualitas model yang diukur dengan kemampuannya untuk generalisasi. Jika ukuran *batch* kurang, pola akan kurang berulang dan karenanya bobot akan ada di seluruh tempat dan konvergensi akan menjadi sulit[40].

2.3.5 Parameter Optimasi

2.3.5.1 Optimizer

Optimizer merupakan algoritma yang digunakan oleh model untuk memperbarui bobot setiap lapisan setelah setiap iterasi. Di antara variasi pengujian

berbagai optimasi *SGD*, *Adam* dan *RMSprop* merupakan optimasi yang paling populer dan paling baik menghasilkan model [41].

a. *Stochastic Gradient Descent (SGD)*

Gradient Descent merupakan algoritma *optimization* yang digunakan untuk mencari nilai *parameter* (koefisien) dari fungsi untuk meminimalkan nilai *cost function*. Prosedure *Gradient Descent* yaitu dimulai dengan initial value untuk sebuah fungsi. *Cost* dari koefisien tersebut kemudian dievaluasi dengan memasukkan ke dalam sebuah fungsi dan menghitung *cost of function*-nya. *Cost derivative*-nya kemudian dihitung. *Derivative* merupakan konsep dari kalkulus dan mengacu pada kemiringan fungsi pada titik tertentu. Kemiringan fungsi digunakan untuk memindahkan nilai koefisien agar mendapatkan biaya yang lebih rendah pada iterasi berikutnya. Untuk selanjutnya *learning rate* harus ditentukan yang akan mengontrol seberapa besar koefisien dapat berubah pada setiap akan meng-*update*. *SGD* dianggap sebagai algoritma optimasi standar yang efektif untuk klasifikasi model *machine learning* seperti *neural network* dan *logistic regression*. Pada *NN*, *Gradient Descent* merupakan algoritma *optimization* yang digunakan *backpropagation* untuk meminimalkan *Mean Squared Error*[42] sehingga mendapatkan model yang terbaik dengan minima kurva. *Pseudocode SGD* sebagaimana disajikan berikut[41]:

[*Algorithm - SGD*] (2.21)

Inisialisasi η dan W_0

For $t = 1 \dots T$

for $i \in \{1 \dots N\}$ pilih secara acak dan berurutan dalam satu waktu

$W_t = W_{t-1} - \eta \nabla E(W_{t-1}, x_i, y_i)$

#cari kesalahan secara maju

$\bar{E} = 0$

for $j = 1 \dots N$

$E_j = E(W_t, x_j, y_j)$

```

 $\bar{E}_t = \bar{E}_t + E_j$ 
End for
 $\bar{E}_t = \bar{E}_t / N$ 
 $t = t + 1$ 
#periksa kriteria penghentian
End for
End for

```

b. *Adam*

Adam merupakan turunan dari *adaptive moment estimation*, yaitu metode untuk optimasi *stochastic* yang efisien yang hanya memerlukan gradien urutan pertama dengan sedikit persyaratan memori. Metode ini menghitung tingkat pembelajaran adaptif individu untuk parameter yang berbeda dari perkiraan momen pertama dan kedua dari gradien. Berikut merupakan Algoritma dari Adam yang diusulkan oleh (D. P. Kingma and J. L. Ba) [43].

[*Algorithm - Adam*] (2.22)

g_t^2 mengindikasikan kuadrat elementwise $g_t \odot g_t$. Nilai setting yang baik $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ dan $\epsilon = 10^{-8}$.

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1]$: tingkat pembusukan eksponensial untuk perkiraan saat ini

Require: $f(\theta)$: Tujuan fungsi stochastic dengan parameter θ

Require: θ_0 Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$V_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (mengambil gradien w.r.t tujuan stochastic pada waktu t)

```

 $m_t \leftarrow \beta_1.m_{t-1} + (1-\beta_1).g_t$  (update bias estimasi first-moment)
 $v_t \leftarrow \beta_2.v_{t-1} + (1-\beta_2).g_t^2$  (update bias estimasi second-row-moment)
 $\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$  (perkiraan komputasi first-moment bias yang dikoreksi)
 $\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$  (perkiraan komputasi second-row-moment bias yang dikoreksi)
 $\theta_t \leftarrow \theta_{t-1} - \alpha.\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (mengubah parameter)

end while
return  $\theta_t$  (Hasil parameter)

```

c. Root Mean Squared Propagation (RMSprop)

RMSprop berusaha mengatasi keterbatasan yang terdapat pada AdaGrad yaitu menghindari efek dari tingkat pembelajaran yang menurun secara monoton dan menggunakan rata-rata yang menurun atau rata-rata bergerak dari turunan parsial daripada jumlah dalam perhitungan tingkat pembelajaran untuk setiap parameter. Masalah yang terjadi pada AdaGrad adalah bahwa dapat memperlambat pencarian terlalu banyak, menghasilkan tingkat pembelajaran yang sangat kecil untuk setiap parameter atau dimensi pencarian pada akhir proses. Hal ini memiliki efek menghentikan pencarian terlalu cepat, sebelum nilai minimal dapat ditemukan. RMSProp dirancang untuk mempercepat proses pengoptimalan. Algoritma *pseudocode RMSprop* sebagaimana disajikan berikut[41]:

[*Algorithm - RMSprop*] (2.23)

Inisialisasi η , ρ , T dan W_0

For $t = 1 \dots T$

$\forall \psi_k^+ = 0 | k \in \{1 \dots \rho\}$

$\forall \psi_k^- = 0 | k \in \{1 \dots \rho\}$

$\beta = 0.9$

for $i \in \{1 \dots N\}$ pilih secara acak dan berurutan dalam satu waktu

$r_t = \beta r_{t-1} + (1-\beta) \nabla E(W_{t-1}, x_i, y_i)^2$

$W_t = W_{t-1} - \eta \nabla E(W_{t-1}, x_i, y_i) / \sqrt{r_t} + \epsilon$

#cari kesalahan secara maju

```

 $\bar{E} = 0$ 
for  $j = 1 \dots N$ 
     $E_j = E(W_t, x_j, y_j)$ 
     $\bar{E}_t = \bar{E}_t + E_j$ 
     $[\psi^+, \psi^-] = \text{collectInconsistentInstances}(i, j, \rho, E_j, \bar{E}_{t-1}, \psi^+, \psi^-)$ 
End for
 $\bar{E}_t = \bar{E}_t / N$ 
 $[\psi^+, \psi^-, \omega_t] = \text{extractConsistentInstances}(i, \rho, \psi^+, \psi^-, \bar{E}_t, \bar{E}_{t-1})$ 
// penyempurnaan lebih lanjut dari  $w_t$  dengan data yang konsisten
For each  $k \in \omega_t$ 
     $W_t = W_t - \eta \nabla E(W_{t-1}, x_k, y_k) / \sqrt{r_t + \epsilon}$  // RMSprop weight update
End for
 $t = t + 1$ 
#periksa kriteria penghentian
End for
End for

```

2.3.5.2 Learning rate

Learning rate adalah *hyperparameter* yang mengontrol seberapa banyak perubahan model dalam menanggapi kesalahan yang diperkirakan setiap kali bobot model diperbarui. Memilih learning rate merupakan tantangan karena nilai yang terlalu kecil dapat mengakibatkan proses pelatihan yang lama dan dapat macet, sedangkan nilai yang terlalu besar dapat mengakibatkan rangkaian pembelajaran mendapatkan bobot yang kurang optimal atau proses pelatihan terlalu cepat yang tidak stabil[44].

Nilai *learning rate* sangat tergantung pada *optimizer* yang digunakan. Untuk SGD umumnya menggunakan 0,1, sedangkan pada Adam umumnya bekerja dengan baik untuk nilai 0.001 atau 0.01, tetapi perlu untuk mencoba semua nilai dari kisaran di atas. Umumnya lebih baik menggunakan algoritma tingkat pembelajaran adaptif seperti Adam daripada menggunakan tingkat pembelajaran

yang lebih buruk. (S. L. Smith et al, 2018) merekomendasi agar menaikkan nilai *batch size* daripada membuat *learning rate* lebih buruk. Ketika kita meningkatkan koefisien momentum, kita meningkatkan akumulasi skala waktu yang diperlukan untuk melupakan gradien lama[45].

2.3.5.3 Momentum

Momentum dapat memperlancar perkembangan algoritma pembelajaran yang juga dapat mempercepat proses pelatihan[44]. Nilai *momentum* yang digunakan umumnya pada *range* antara 0 sampai dengan 1, nilai yang paling optimal adalah 0,9[46].

2.4. Metrics Performance

Metric Performance merupakan suatu nilai yang digunakan untuk mengukur kinerja suatu model. Berikut beberapa *metric performance* yang dapat digunakan untuk mengukur suatu model *machine learning*:

2.4.1 Confusion Matrix

Confusion Matrix merupakan salah satu cara untuk melihat kinerja pada *classifier/supervised learning*[47]. *Confusion Matrix* dapat memberikan nilai akurasi dari validasi algoritma pada *dataset* yang ada[48]. *Metric performance* yang pertama *confusion matrix* yaitu metode standar untuk menyajikan jumlah *True Positive* (TP), *False Positive* (FP), *True Negative* (TN) dan *False Negative* (FN) dengan cara yang lebih visual [32]. *Confusion Matrix* ditunjukkan oleh tabel, dimana setiap kolom pada table tersebut mewakili kelas yang diprediksi dengan kelas sebenarnya atau kelas aktual. Untuk klasifikasi dua kelas (klasifikasi biner), tampilan *confusion matrix* sebagaimana disajikan pada Tabel 2.1[49].

Tabel 2.3. *Confusion Matrix*

<i>Predict</i>	<i>Actual</i>	
	<i>Positive</i>	<i>Negative</i>
<i>Positive</i>	<i>TP</i>	<i>FP</i>
<i>Negative</i>	<i>FN</i>	<i>TN</i>

Keterangan:

TP = *True Positive* TN = *True Negative*

FP = *False Positive* FN = *False Negative*

True Positives menunjukkan hasil prediksi model untuk jumlah kelas aktual positif yang tepat diprediksi sebagai kelas positif. *True Negative* menunjukkan hasil prediksi model untuk jumlah kelas aktual negatif yang tepat diprediksi sebagai kelas negatif. *False Positives* menunjukkan hasil prediksi model untuk jumlah kelas aktual negatif yang keliru diprediksi sebagai kelas positif. *False Negatives* menunjukkan hasil prediksi model untuk jumlah kelas aktual positif yang keliru diprediksi sebagai kelas negatif[50]. Berdasarkan *confusion matrix* ini dapat dihitung dan diinterpretasikan *metric performance* lainnya.

2.4.2 Accuracy

Accuracy merupakan salah satu *metric performance* yang paling umum digunakan untuk kinerja klasifikasi, dan didefinisikan sebagai rasio antara sampel yang diklasifikasikan dengan benar ke jumlah total Sampel[51]. *Accuracy* dapat dihitung dengan menggunakan rumus sebagai berikut[49]:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \dots\dots\dots (2.24)$$

2.4.3 Precision

Precision adalah rasio sampel berkategori positif yang diklasifikasi dengan benar dibandingkan dengan total sampel yang diklasifikasi sebagai positif.

$$Precision = \frac{TP}{TP+FP} \dots\dots\dots (2.25)$$

2.4.4 Recall

Recall adalah rasio sampel yang diprediksi positif yang diidentifikasi dengan benar.

$$Recall = \frac{TP}{TP+FN} \dots\dots\dots (2.26)$$

2.4.5 F1-score

F1-Score rata-rata harmonic antara *precision* dan *recall*.

$$F1 - score = \frac{2x (Precision \times Recall)}{(Precision+Recall)} \dots\dots\dots (2.27)$$

2.4.6 Receiver Operating Characteristic Curve - Area Under the Curve (ROC-AUC)

ROC merupakan grafik hubungan dua dimensi antara *True Positive Rate* (y-axis) dengan *False Positive Rate* (x-axis). Metode penilaian grafis seperti *ROC* dan kurva *Precision-Recall* memberikan interpretasi yang berbeda dari kinerja klasifikasi[51].

2.4.7 Kappa

Kappa intraclass digunakan untuk penilaian *agreement* atau keandalan (*reliability*) antara dua atau pengukuran lebih dari satu kategoris dan *weighted kappa* digunakan untuk penilaian korelasi antara dua tindakan kategoris yang tidak berurutan, dan estimasi sampelnya direview[52].

$$k = \frac{2 \times (TP \times TN - FN \times FP)}{(TP+FP) \times (FP+TN) + (TP+FN) \times (FN+TN)} \dots\dots\dots (2.28)$$

2.4.8 Sensitivity

Sensitivity merupakan pecahan dari sampel yang relevan yang diambil. *Sensitivity* dihitung dengan menggunakan rumus sebagai berikut[49]:

$$Sensitivity = \frac{TP}{TP+FN} \dots\dots\dots (2.29)$$

2.4.9 *Specificity*

Specificity menghitung proporsi positif aktual yang diidentifikasi dengan benar. *Specificity* dihitung menggunakan rumus sebagai berikut[49]:

$$Specificity = \frac{TN}{TP+FP} \dots\dots\dots (2.30)$$

BAB 3 METODOLOGI PENELITIAN

3.1. Deskripsi Masalah dan Dataset

Stroke merupakan kondisi dimana kematian sel terjadi karena kurangnya sirkulasi darah ke otak. Stroke merupakan keadaan darurat medis yang perlu segera membutuhkan perhatian medis. Selama orang menderita penyakit stroke, otak tidak menerima cukup oksigen atau nutrisi yang mengakibatkan sel-sel mati. Stroke dapat terjadi pada usia berapa pun dan hampir seperempat kasus stroke terjadi pada orang yang berusia di bawah 65 tahun. Stroke juga dapat menyerang laki-laki maupun perempuan. Jumlah pasien stroke yang tinggi menyebabkan biaya yang dikeluarkan untuk perawatan sangat tinggi, produktivitas menurun dan berdampak pada kematian.

Permasalahan dan dampak yang diakibatkan dari stroke perlu diambil tindakan untuk menyelesaikannya. Hal ini sesuai dengan adanya dukungan sistem keputusan klinis yang dapat digunakan untuk melakukan prediksi. Oleh karena itu, prediksi risiko stroke dapat berkontribusi secara signifikan karena masalah sudah meningkat pada tingkat yang mengkhawatirkan. Kebutuhan yang muncul untuk memeriksa data kesehatan dan mengembangkan sistem yang dapat memprediksi apakah seseorang cenderung menderita stroke atau tidak. Di bawah asumsi kedokteran, pencegahan dan deteksi faktor risiko akan sangat membantu untuk perawatan dini. Data medis pencegahan dan perawatan dini untuk penyakit yang lain juga telah dilakukan, sehingga sangat relevan jika langkah ini juga dilakukan untuk penyakit stroke.

Dataset yang digunakan untuk penelitian ini merupakan data sekunder atau data publik yang diambil dari *Kaggle*. Dataset memiliki 12 *feature* dan 5.110 *instance* dengan ukuran file 310 Kb. Semua *features* yang terdapat pada dataset tersebut merupakan catatan elektronik dari pasien, kecuali id, terutama berdasarkan data fisiologis dasar pasien, penyakit historis dan lingkungan. Dataset yang telah diambil dari *Kaggle*, selanjutnya disimpan ke dalam *google drive*. Karakteristik dataset memiliki enam atribut *binary* yaitu *gender*, *hypertension*, *heart disease*, *ever married*, *residence type* dan *stroke*, delapan atribut dalam bentuk *categorical*

yaitu *gender*, *hypertension*, *heart disease*, *ever married*, *work type*, *residence type*, *smoking status* dan *stroke* dan enam atribut dalam bentuk numerik yaitu *age*, *hypertension*, *heart disease*, *avg glucose level*, *bmi* dan *stroke*. Atribut yang *binary*, ada yang berbentuk numerik dan juga ada yang berbentuk *text*. Atribut dataset yang digunakan dan penjelasannya disajikan pada Table 3.1[8].

Tabel 3.1. *Description of the Dataset Attribute*

Attribute	Description
ID	Nomor id pasien
<i>Gender</i>	Jenis kelamin pasien (<i>Male, Female</i>)
<i>Age</i>	Usia pasien
<i>Hypertension</i>	0 – tidak hipertensi 1 – hipertensi
<i>Heart disease</i>	0 – tidak memiliki riwayat penyakit jantung 1 – memiliki riwayat penyakit jantung
<i>Ever Married</i>	Marital Status Pasien Ya - Menikah Tidak - Tidak menikah
<i>Work-type</i>	Jenis pekerjaan pasien (<i>Private, Self-employed, Govt_job, Children, Never-Worked</i>)
<i>Residence area</i>	Wilayah tempat tinggal pasien (<i>Urban, Rural</i>)
<i>Avg-glukose</i>	Rata-rata level glukosa dalam darah yang diukur setelah makan
<i>BMI</i>	<i>Body Mass Index</i> pasien
<i>Smoking status</i>	Status merokok pasien (<i>Never Smoked, Formely Smoked, Smokes</i>)
<i>Stroke</i>	0 – Tidak stroke 1 – Stroke

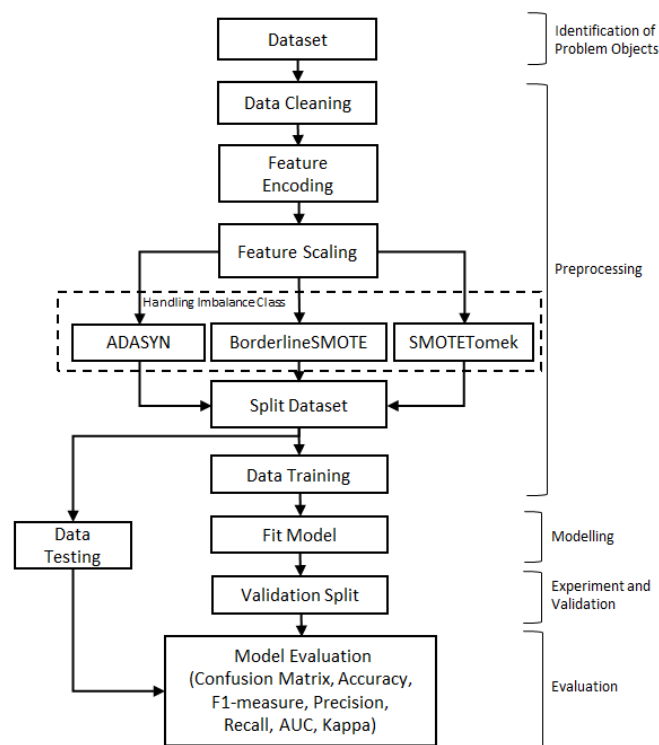
Sepuluh instans pertama dari dataset tersebut disajikan pada Tabel 3.2. Pada tabel tersebut menginformasikan karakteristik awal dataset bahwa terdapat data yang memiliki nilai dalam bentuk *numeric* dan *text*, ada yang berbentuk *Categorical text* dan berbentuk *Categorical numeric*. Disamping itu juga memberikan informasi bahwa skala data pada beberapa atribut berbeda dengan yang lainnya. Informasi awal lainnya yang dapat diperoleh yaitu adanya *missing values* pada atribut *BMI* yang berisikan nilai ‘*NaN*’ dan pada atribut *smoking_status* yang berisikan nilai ‘*Unknown*’.

Tabel 3.2. *Dataset*

id	gender	age	hypertension	Heart disease	Avg glucose level	bmi	Smoking status	stroke
9046	Male	67.0	0	1	228.69	36.6	formerly smoked	1
51676	Female	61.0	0	0	202.21	NaN	never smoked	1
31112	Male	80.0	0	1	105.92	32.5	never smoked	1
60182	Female	49.0	0	0	171.23	34.4	smokes	1
1665	Female	79.0	1	0	174.12	24.0	never smoked	1
56669	Male	81.0	0	0	186.21	29.0	formerly smoked	1
53882	Male	74.0	1	1	70.09	27.4	never smoked	1
10434	Female	69.0	0	0	94.39	22.8	never smoked	1
27419	Female	59.0	0	0	76.15	NaN	Unknown	1
60491	Female	78.0	0	0	58.57	24.2	Unknown	1

3.2. Eksperimen Yang Dilakukan

3.2.1. Tahapan Penelitian



Gambar 3.1. Tahapan Penelitian

Penelitian dilakukan sesuai dengan tahapan-tahapan sebagaimana disajikan pada Gambar 3.1. Penelitian dimulai dari melakukan identifikasi objek permasalahan sampai dengan melakukan evaluasi model arsitektur DNN yang diusulkan. Penjelasan atas setiap tahapan sebagai berikut:

3.2.2. *Data Preparation*

Tahapan persiapan data dilakukan untuk memastikan bahwa dataset yang digunakan untuk *training* dan *testing* merupakan data yang berkualitas. Jika dataset yang digunakan masih memiliki *noise* maka model yang dihasilkan juga tidak akan berkualitas dan memiliki *bias*. Pada tahapan persiapan data dilakukan hal-hal sebagai berikut:

3.2.2.1 *Data Cleaning*

Model DNN akan bekerja secara *garbage in – garbage out*. Jika data yang dimasukkan tidak bersih atau banyak mengandung *noise* maka model yang akan dihasilkan juga akan bias dan kinerjanya kurang baik. Oleh karena itu sangat penting untuk membersihkan data sebelum melatih model. Kegiatan *data cleaning* akan sangat tergantung dari hasil identifikasi permasalahan dengan *exploratory data analysis (EDA)*. Pada dataset yang sering terjadi adalah adanya *missing values* yang harus ditangani. Teknik imputasi menurut [53] terbagi menjadi dua yaitu *Statistical Techniques* dan *Machine Learning Techniques*. Ada beberapa metode pada Teknik *Statistical Techniques* yang dapat digunakan untuk menangani *missing values*, yaitu dengan melakukan *imputasi* menggunakan nilai rata-rata (*mean*), *imputasi* dengan nilai tengah (*median*) atau dengan nilai yang memiliki frekuensi paling sering (*mode*). Metode ini sudah pernah diperbandingkan oleh Folino and Pisani pada penelitian [54]. Pada penelitian ini, untuk menangani *missing values* akan dilakukan teknik *imputasi* dengan menggunakan nilai *mean* untuk atribut *bmi* dan nilai *mode* untuk *smoking status*.

3.2.2.2 Feature Encoding

Pengkodean adalah proses konversi data dari satu formulir ke formulir lainnya. Sebagian besar algoritma mesin pembelajaran tidak dapat menangani nilai kategorikal kecuali kita mengonversinya menjadi nilai numerik[18]. Ada banyak cara untuk mengubah variabel kategorikal menjadi numerik. Beberapa di antaranya adalah *Label Encoding*, *One Hot Encoding (OHE)*, *Ordinal Encoding*, dan *Frequency Encoding*. Pada penelitian ini untuk melakukan konversi nilai pada variable dalam bentuk kategorikal ke dalam bentuk numerik menggunakan teknik *Label Encoding*.

Pendekatan ini sangat sederhana dan melibatkan konversi setiap nilai pada kolom menjadi angka. *Label Encoding* mengacu pada konversi label menjadi bentuk numerik sehingga dapat mengubahnya menjadi bentuk yang dapat dibaca mesin. Algoritma *machine learning* kemudian dapat memutuskan dengan cara yang lebih baik tentang bagaimana label tersebut harus dioperasikan. Hal ini merupakan langkah penting *preprocessing* untuk dataset terstruktur dalam *supervised learning*. Data pada Tabel 3.3 merupakan hasil *encode* label dataset yang diproses menggunakan teknik *label encoding library sklearn python*.

Tabel 3.3. *Label Encode*

ID	Label Tanpa Encoding	Label Encoding
Data 1	formerly smoked	2
Data 2	never smoked	1
Data 3	never smoked	1
Data 4	smokes	0
..
Data 166	never smoked	1
Data 167	formerly smoked	2
Data 168	never smoked	1
Data 169	smokes	0
Data 170	formerly smoked	2

3.2.2.3 Feature Scaling

Standardisasi atau *feature scaling* merupakan salah satu tahapan selama data *pre-processing*. Hal ini dilakukan untuk menormalkan data dalam rentang tertentu. *Feature scaling* bertujuan untuk membantu dalam mempercepat perhitungan dalam algoritma. *Feature scaling* data adalah persyaratan umum untuk eksperimen yang dilakukan menggunakan *Keras*, *Scikit-learn* dan *Deep Learning*. Ada banyak *scaler* berbeda yang tersedia untuk *feature scaling* dataset. Yang paling umum digunakan adalah *Normalizer*, *RobustScaler*, *MinMaxScaler*, dan *StandardScaler* [19]. Penelitian ini akan menggunakan *StandardScaler* untuk *feature scaling* dataset.

StandardScaler akan mengubah dataset sedemikian rupa sehingga nilai rata-rata distribusi yang dihasilkan adalah nol dan simpangan bakunya satu[55]. Nilai yang ditransformasi diperoleh dengan mengurangi nilai rata-rata dari nilai original dan membagi dengan simpangan baku. Rumus untuk proses transformasi adalah sebagai berikut:

$$z = \frac{x-\mu}{\sigma} \dots\dots\dots (3.1)$$

dimana, z merupakan nilai hasil transformasi

x adalah nilai original

μ adalah nilai rata-rata

σ adalah nilai standard deviasi

3.2.2.4 SMOTE

Pada tahapan ini akan dilakukan teknik menangani *imbalance class*. Beberapa teknik *SMOTE* akan diuji untuk mendapatkan hasil akurasi yang paling baik untuk menangani *imbalance* sesuai dengan dataset yang digunakan. Teknik *SMOTE* yang digunakan pada penelitian ini yaitu: *ADASYN*, *BorderlineSMOTE* dan *SMOTETomek*. Tujuan dari *ADASYN* adalah mengurangi bias dan pembelajaran adaptif[24]. *Borderline* lebih tepat untuk *misclassification* disepanjang garis border dibandingkan untuk yang jauh dari border, dan jadi lebih penting untuk digunakan dalam melakukan klasifikasi[25]. *SMOTETomek* menggunakan dataset yang telah seimbangkan dengan teknik *SMOTE*, kemudian mengidentifikasi *Tomek* dan menghapusnya, sehingga menghasilkan kumpulan

data seimbang dengan kluster kelas yang terdefinisi dengan baik[29]. Proses *SMOTE* menggunakan *imbalanced-learn libraries*.

3.2.3 Pembuatan Model Arsitektur DNN

Model Arsitektur DNN dibuat dalam tiga model algoritma berdasarkan jumlah *hidden layer* yaitu Model A, Model B dan Model C. Model A terdiri 4 *hidden layer*, Model B terdiri 5 *hidden layer* dan Model C terdiri 6 *hidden layer*. Penentuan jumlah *hidden layer* menggunakan *Keras Tuner* dan teknik *random search*, karena jumlah parameter yang dipertimbangkan sangat tinggi dan besarnya pengaruh tidak seimbang. Untuk mengatasi bias yang dihasilkan karena proses *random search* diperlukan *cross validation*. Pada beberapa implementasi memerlukan jumlah layer yang lebih banyak untuk meningkatkan kinerja model[56]. Semua model yang dibuat menerapkan prinsip kerja *learning back-propagation algorithm* [6]. Eksperimen dilakukan dengan beberapa variasi *setting hyperparameter*. Evaluasi *performance model* dilakukan dengan membandingkan hasil skor *metric performance* yang didapatkan berdasarkan:

- optimasi dan besarnya *learning rate* yang digunakan
- jumlah *epoch* yang digunakan pada setiap variasi model
- jumlah *hidden layer* pada model *neural network*.
- teknik *handling imbalance class*.

Tabel 3.4. Arsitektur DNN

Model DNN		Jumlah		
Model	Teknik Handling Imbalance Class	Input Neuron	Hidden Layer	Hidden Neuron
A	ADASYN	10	4	320,256,32,96
	BLSMOTE	10		320,256,32,64
	SMOTETomek	10		320,256,32,32
B	ADASYN	10	5	320,288,32,320,96
	BLSMOTE	10		320,288,32,320,64
	SMOTETomek	10		320,288,32,320,32
C	ADASYN	10	6	320,224,128,32,320,96
	BLSMOTE	10		320,224,128,32,320,64
	SMOTETomek	10		320,224,128,32,320,32

Eksperimen akan dilakukan terhadap ketiga model dengan masing-masing mengimplementasikan teknik *handling imbalance class*. Arsitektur model DNN dan jumlah neuron pada setiap *hidden layer* sebagaimana disajikan pada Tabel 3.1. Setiap Model memiliki input neuron sebanyak 10 sesuai dengan jumlah *feature predictor* pada dataset. Penentuan jumlah neuron pada *hidden layer* dilakukan secara *random* dengan pola dari besar-kecil-besar-kecil dengan menggunakan kelipatan angka 32[57]. Semua *hidden layer* pertama dimulai dengan jumlah neuron 320 yang merupakan perkalian dari jumlah input neuron dengan kelipatan yang digunakan untuk penentuan jumlah neuron pada *hidden layer* dan semua neuron antar layer *fully connected*[56].

Fungsi aktivasi pada input layer dan *hidden layer* menggunakan *ReLU* sedangkan untuk output layer menggunakan *Sigmoid*. Pada *hidden layer*, kecuali *hidden layer* terakhir, menggunakan *batch normalization* dan *dropout* sebesar 0,2 untuk regularisasi dan mengurangi *overfitting*. Eksperimen masing-masing akan dilakukan dengan jumlah epoch 400, 500, dan 600 dengan *batch size* 96. Variasi Model DNN dengan parameter dan nilainya sebagaimana pada Tabel 3.5.

Tabel 3.5. *Parameter Model DNN*

Parameter	Nilai Parameter
Aktivasi Input Layer	<i>ReLU (Rectified Linear Unit)</i>
Aktivasi Hidden Layer	<i>ReLU (Rectified Linear Unit)</i>
Aktivasi Output	<i>Sigmoid</i>
Dropout	0,2
Epoch	400,500,600
Batch Size	96

Kombinasi parameter optimasi Model DNN sebagaimana disajikan pada Tabel 3.6. Berdasarkan komposisi pada tabel tersebut, eksperimen akan dilakukan terhadap ketiga model dengan optimasi SGD, Adam dan RMSprop. Untuk optimasi *SGD* menggunakan nilai *learning rate* 0.1, 0.01 dan 0.001 dan *momentum* 0.9, sedangkan untuk *Adam* dan *RMSprop* menggunakan nilai *learning rate* sebesar

0,01, 0,001 dan 0,0001 dengan nilai *momentum default*. Pemodelan dilakukan dengan menggunakan *keras* dan *back-end tensorflow*.

Tabel 3.6. *Parameter Optimasi Model DNN*

Optimasi	Learning Rate	Momentum
SGD (<i>Stochastic Gradient Descent</i>)	0.1	0.9
	0.01	0.9
	0.001	0.9
Adam (<i>Adaptive Moments Estimation</i>)	0,01	<i>Default</i>
	0,001	<i>Default</i>
	0,0001	<i>Default</i>
RMSprop (<i>Root Mean Square Root</i>)	0,01	<i>Default</i>
	0,001	<i>Default</i>
	0,0001	<i>Default</i>

3.2.3.1. Eksperimen dan Validasi

Eksperimen dilakukan menggunakan aplikasi *Jupyter Notebook* versi 6.3.0 dengan Bahasa pemrograman *Python 3.8.8*. Spesifikasi komputer yang digunakan untuk *running program* sebagaimana disajikan pada Tabel 3.7.

Tabel 3.7. Spesifikasi Komputer

<i>Computer Spesification</i>	
Sistem Operasi	<i>Windows 10 Pro 64-Bit Version 20H2 (Build 19042.1110)</i>
<i>Processor</i>	<i>Intel(R) Core(TM) i5-9400T CPU @ 1.80GHz 1.80 GHz</i>
RAM	8GB DDR4 (2666MHz)
<i>Harddisk</i>	1TB 5400RPM 3.5" SATA III + 256GB NVMe M.2 SSD
Aplikasi <i>Data Mining</i>	<i>Jupyter Notebook 6.3.0</i>

Teknik validasi model DNN yang diimplementasikan pada penelitian ini menggunakan *validation split*. Prinsip kerja *validation split* membagi *data training* menjadi dua bagian. Bagian pertama sebagai *data training* dan x% bagian akhir

sebagai data validasi (*hold-out to validate on while training*). Set validasi yang sama digunakan untuk semua *epoch*. Pada penelitian ini nilai x disetting sebesar 0,2 sehingga validasi menggunakan 20% bagian akhir data. Sehingga secara keseluruhan dataset akan dibagi menjadi tiga bagian, yaitu:

- a. *Data training*, bagian dari dataset untuk melatih model.
- b. *Data validation*, bagian dari dataset untuk validasi model pada saat pelatihan.
- c. *Data testing*, bagian dari dataset untuk evaluasi *performance* model.

3.3. Metrics Performance Evaluasi Model

Setelah dilakukan eksperimen dan validasi atas model selama proses *training*, selanjutnya akan dilakukan evaluasi atas setiap model. Evaluasi model menggunakan *data testing*. Proses evaluasi menggunakan *scikit-learn libraries*. *Metrics performance* yang digunakan untuk mengevaluasi model yaitu *accuracy*, *f1-score*, *precision*, *recall*, *AUC* dan *kappa*. Disamping itu, untuk memastikan bahwa model DNN yang dipilih tidak *over-fitting*, proses eksperimen juga melakukan *plotting training dan validasi history accuracy dan negative logarithmic loss*. Pemilihan model yang diusulkan dilakukan dengan menganalisis dan membandingkan hasil dari *metrics performance* tersebut.

3.4. Pengembangan Aplikasi

Model dengan *metrics performance* yang paling baik telah dipilih dan dibuat menjadi arsitektur DNN yang diusulkan. Model tersebut kemudian dikembangkan menjadi sebuah Aplikasi Prediksi Stroke. *Code Editor* yang digunakan untuk membuat program menggunakan *Visual Studio Code versi 1.57.1*. Aplikasi Stroke dikembangkan dan di-*deployment* dengan menggunakan bahasa pemrograman *Python version 3.7.6* dan *Flask version 1.1.2*.

BAB 4

HASIL PENELITIAN DAN PEMBAHASAN

4.1. Analisis Statistik Dataset

Tahap pertama yang dilakukan dalam eksperimen adalah menganalisis statistik dataset dengan melakukan proses *Exploratory Data Analysis (EDA)*. Berikut merupakan hasil dari kegiatan *EDA* dan penjelasannya:

4.1.1 Deskriptif Statistic

Data Hasil *EDA* untuk *deskriptif statistic* sebagaimana disajikan pada Table 4.1.

Tabel 4.1. Deskriptif Statistik

	Age	Hypertension	Heart disease	Avg_glucose level	BMI	Stroke
count	5110.00	5110.0000	5110.0000	5110.0000	4909.0000	5110.0000
mean	43.22	0.0974	0.0540	106.1477	28.8932	0.0487
std	22.61	0.2966	0.2261	45.2836	7.8541	0.2153
min	0.08	0.0000	0.0000	55.1200	10.3000	0.0000
25%	25.00	0.0000	0.0000	77.2450	23.5000	0.0000
50%	45.00	0.0000	0.0000	91.8850	28.1000	0.0000
75%	61.00	0.0000	0.0000	114.0900	33.1000	0.0000
max	82.00	1.0000	1.0000	271.7400	97.6000	1.0000

Data yang disajikan pada Tabel 4.1 dapat dianalisis bahwa dataset memiliki 5110 *instance*. Data usia pasien minimal 0.08 tahun dan usia maksimal 82 tahun dengan standar deviasi 22.61. Rata-rata dua puluh lima persen data pasien berusia 25 tahun, rata-rata lima puluh persen pasien berusia 45 tahun, dan rata-rata tujuh puluh lima persen pasien berusia 61 tahun. Sedangkan rata-rata usia pasien secara keseluruhan adalah 43 tahun. Data rata-rata level glukosa pasien nilai terendah 55,12 dan nilai tertinggi 271,74. Rata-rata dua puluh lima persen pasien memiliki level glukosa 77,245, rata-rata lima puluh persen pasien memiliki level glukosa 91,88 dan rata-rata tujuh puluh lima persen pasien memiliki level glukosa 114,09. Sedangkan rata-rata level glukosa pasien secara keseluruhan 106,15 dengan standar deviasi 45,28.

Jumlah *Body Mass Index (BMI)* pasien sebanyak 4909 dari 5110 jumlah *instance*. Hal ini mengindikasikan adanya data pasien yang tidak memiliki nilai

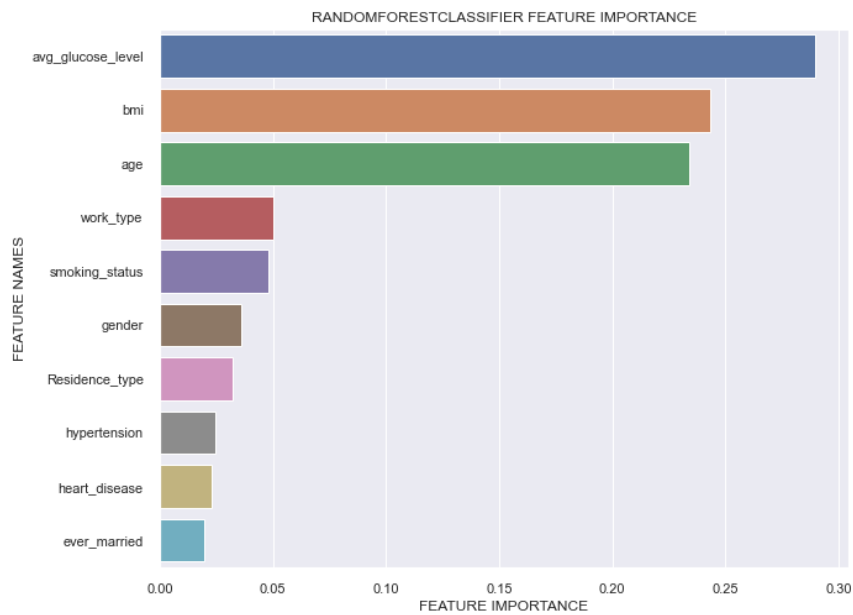
BMI, sehingga perlu dianalisis lebih lanjut. Dari 4909 pasien tersebut nilai *BMI* paling rendah 10,3 dan nilai paling tinggi 97,6. Rata-rata dua puluh lima persen pasien memiliki *BMI* 23,5, rata-rata lima puluh persen pasien memiliki *BMI* 28,1 dan rata-rata tujuh puluh lima persen pasien memiliki *BMI* 33,1. Sedangkan rata-rata *BMI* pasien secara keseluruhan 28,89 dengan standar deviasi 7,85.

4.1.2 Feature Importance

Feature yang terdapat pada dataset memiliki tingkat pengaruh yang berbeda-beda pada stroke. Oleh karena itu, perlu menggunakan algoritma *machine learning* untuk mengidentifikasi *feature* yang terpenting untuk membantu mengetahui terjadinya stroke dari *feature* tersebut. Dengan demikian, prediksi stroke yang dilakukan dapat lebih akurat. Dari kesepuluh *feature predictor*, tidak ada *feature* yang memiliki pengaruh kuat stroke seperti ditunjukkan pada Tabel 4.2 terkait dengan *important coefisien value* masing-masing *feature*. Namun demikian terdapat tiga *feature* yang memiliki pengaruh dominan atau lebih tinggi dibandingkan *feature* lainnya. Ketiga *feature* tersebut adalah rata-rata level glukosa, *BMI* dan usia sebagaimana disajikan pada Gambar 4.1 yang diidentifikasi menggunakan *Random Forest Feature Importance*.

Tabel 4.2. *Feature Important Coefisien*

<i>Feature</i>	<i>Coefisien Important</i>
<i>gender</i>	0.03594
<i>age</i>	0.23384
<i>hypertension</i>	0.02475
<i>heart_disease</i>	0.02295
<i>ever_married</i>	0.01961
<i>work_type</i>	0.05005
<i>Residence_type</i>	0.03223
<i>avg_glucose_level</i>	0.28967
<i>bmi</i>	0.24302
<i>smoking_status</i>	0.04794



Gambar 4.1. *Feature Importance* dengan *RandomForest*

4.1.3 Missing Values

Tabel 4.3. *Missing Values*

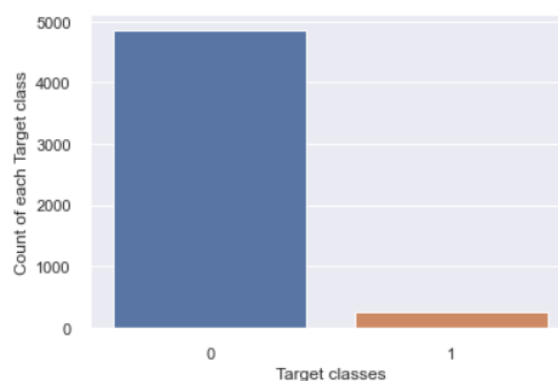
<i>Features</i>	<i>Missing Values</i>
<i>id</i>	0
<i>gender</i>	1
<i>age</i>	0
<i>hypertension</i>	0
<i>heart_disease</i>	0
<i>ever_married</i>	0
<i>work_type</i>	0
<i>Residence_type</i>	0
<i>avg_glucose_level</i>	0
<i>bmi</i>	201
<i>smoking_status</i>	1544
<i>stroke</i>	0

Berdasarkan observasi, pada *feature smoking status* terdapat nilai ‘unkown’. Untuk nilai ini dilakukan *treatment* sehingga menjadi *missing values*. Oleh karena itu, dataset memiliki *missing values* sebagaimana disajikan pada Tabel 4.3. *Missing values* terdapat pada tiga atribut, yaitu atribut *gender*, *BMI* dan *smoking status*. Pada

feature gender hanya terdapat satu *instance* dan secara intuisi langsung dilakukan imputasi menggunakan *mode imputation*, *feature BMI*, jumlah *missing values* sebanyak 201 (3,9%) dan jumlah *missing values* pada *smoking status* sebanyak 1544 (30,2%). Berdasarkan Gambar 4.1, *feature BMI* merupakan *feature* penting dibandingkan dengan *feature* lainnya, hal ini mengindikasikan bahwa meskipun jumlahnya sangat kecil, namun kemungkinan terdapat informasi penting pada *feature* tersebut, sehingga *instance* yang terdapat *missing values* BMI tidak dihapus. Sedangkan *missing values* untuk atribut *smoking status* karena prosentasenya signifikan juga tidak dihapus.

4.1.4 Imbalance Class

Dataset memiliki dua *target class* dengan komposisi untuk *class 0* (tidak-stroke) sebanyak 4861 *instance* (95,13%) dan *class 1* (stroke) sebanyak 249 *instance* (4,87%). Gambar 4.2 menunjukkan bahwa kedua *class target* tidak seimbang (*high imbalance class*).

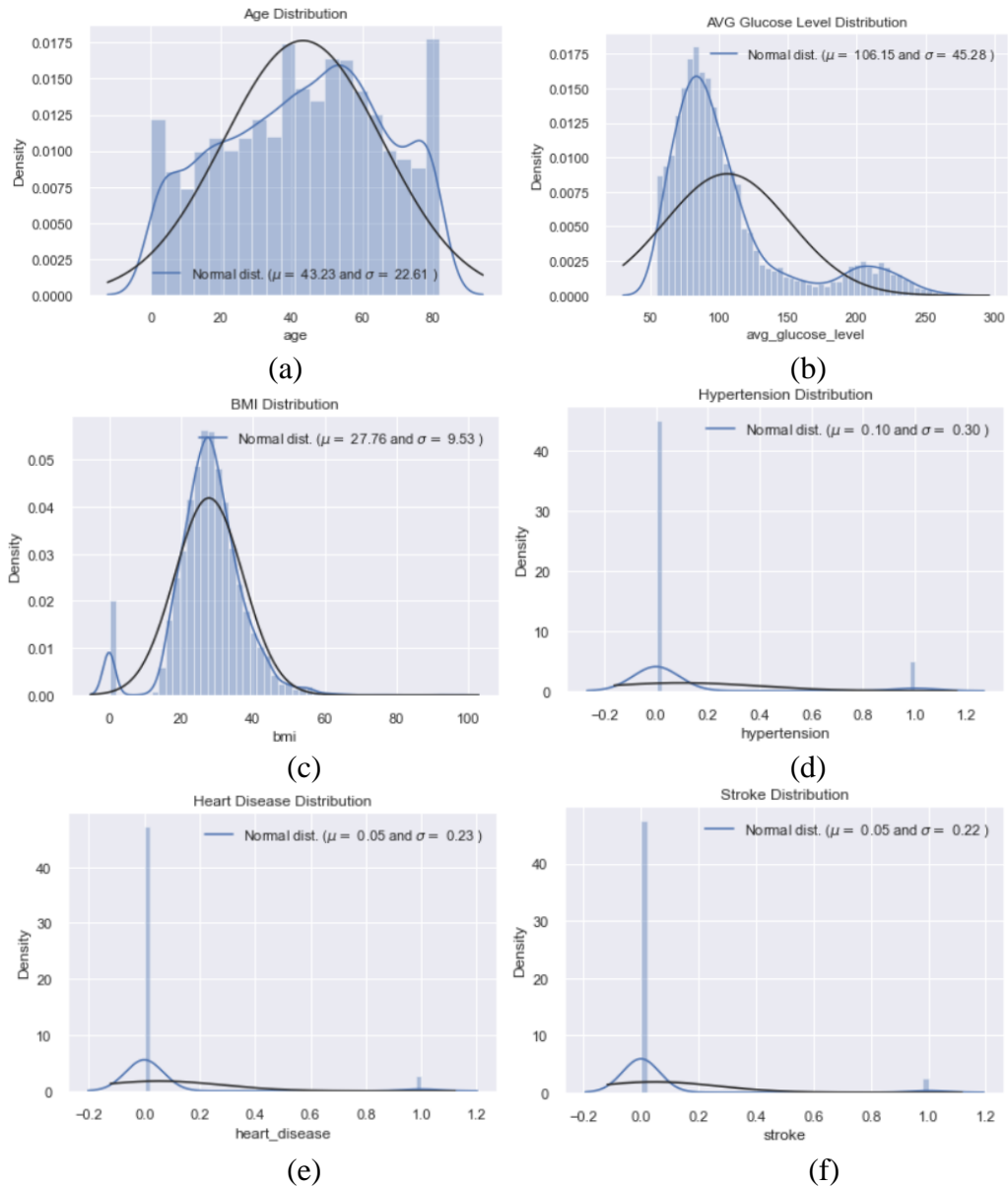


Gambar 4.2. Class Target Distribution

4.1.5 Feature Distribution

Pada Gambar 4.3 menunjukkan bahwa data usia, rata-rata level glukosa dan *BMI* sebagai faktor yang harus dipertimbangkan dalam memprediksi kemungkinan stroke. Sedangkan untuk hipertensi dan penyakit jantung menunjukkan bahwa sebagian besar data pasien tidak memiliki riwayat hipertensi dan penyakit jantung. Pada *feature* usia berbentuk *unimodal* dan rata-rata usia pada dataset 43 tahun dan

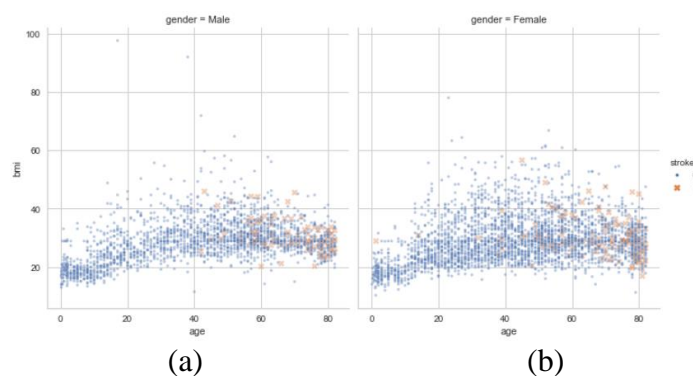
hampir rata di semua usia kecuali terjadinya lonjakan yang tajam pada usia anak dan orang tua. Untuk rata-rata level glukosa berbentuk *bimodal* dengan sedikit *skew-right*. Sedangkan untuk *BMI* lebih berbentuk *unimodal* dan sedikit miring ke kiri (*skew-left*).



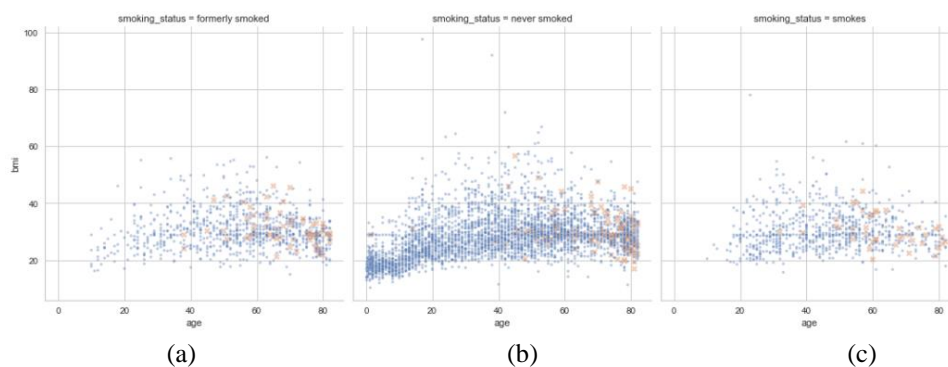
Gambar 4.3. *Features Distribution*

Pada Gambar 4.4 (a) dan 4.4 (b) menunjukkan bahwa pada usia di atas 40 tahun dan kurang dari 80 tahun lebih berpotensi terkena stroke baik pada jenis

kelamin laki-laki maupun perempuan. Sedangkan pada Gambar 4.5 dapat dianalisis bahwa status merokok juga menjadi faktor yang sangat mempengaruhi terhadap stroke. Sebagian besar yang tidak stroke adalah yang tidak pernah merokok dan sebagian besar pada usia dibawah 40 tahun. Sedangkan usia diatas 60 tahun meskipun tidak pernah merokok juga berpotensi terkena stroke. Sebagian besar yang pernah merokok berpotensi terkena stroke dan hanya sedikit yang pernah merokok tidak terkena stroke. Sedangkan yang masih merokok mulai terkena stroke pada usia di atas 50 tahun.



Gambar 4.4. *Effect of Body Mass Index, Age and Gender on Stroke*



Gambar 4.5. *Effects of Age and Smoking Status on Stroke*

4.2. Hasil Eksperimen dan Analisis

Eksperimen dilakukan sesuai dengan Tahapan Penelitian. Tahap pertama penelitian telah dijelaskan pada bab sebelumnya. Hasil dari tahap kedua dan selanjutnya akan dijelaskan pada bab ini.

4.2.1. Hasil *Data Preparation*

Sebelum dilakukan eksperimen untuk pemodelan, terlebih dahulu dilakukan tahap *pre-processing* atau *data preparation*. Proses *pre-processing* terdapat beberapa sub tahapan yang akan dijelaskan sebagai berikut:

4.2.1.1 *Data Cleaning*

Berdasarkan observasi *EDA* menunjukkan bahwa dataset memiliki *missing values* pada dua *feature* yaitu *BMI* dan *smoking status*. *Missing values* pada kedua *feature* tersebut ditangani dengan dilakukan *imputasi*. Teknik *imputasi* yang dilakukan adalah dengan menggunakan nilai *mean* untuk *feature BMI* dan nilai *modus* untuk *feature smoking status*. Hasil pengecekan *missing values* setelah dilakukan proses *imputasi* disajikan pada Tabel 4.5.

Tabel 4.4. *Missing Values after Imputation*

<i>Features</i>	<i>Missing Values</i>
<i>id</i>	0
<i>gender</i>	0
<i>age</i>	0
<i>hypertension</i>	0
<i>heart_disease</i>	0
<i>ever_married</i>	0
<i>work_type</i>	0
<i>Residence_type</i>	0
<i>avg_glucose_level</i>	0
<i>bmi</i>	0
<i>smoking_status</i>	0
<i>stroke</i>	0

Tabel 4.4 menunjukkan hasil pengecekan bahwa sudah tidak terdapat *missing values* karena telah dilakukan imputasi pada *feature BMI* dan *smoking status*.

4.2.1.2 Feature Encoding

Feature yang memiliki nilai dalam bentuk kategorikal harus diubah terlebih dahulu ke dalam bentuk numerik. Untuk *feature* kategorikal yang tidak ada hubungan ordinal, pengkodean ke dalam numerik dalam bentuk *integer* tidak cukup, Hal ini akan menghasilkan model yang kurang baik. Namun, hal ini akan sangat tergantung dengan dataset dan model yang digunakan, karena membuat model mengasumsikan membuat urutan secara alami antara kategori dapat mengakibatkan *performance* yang tidak terduga. Pada tahapan ini, semua pengkodean nilai *feature* kategorikal diimplementasikan dengan *Label Encoder*. Hasil dari *preprocessing Label Encoder* sebagaimana disajikan pada Tabel 4.5.

Tabel 4.5. *Dataset setelah Feature Encoding*

id	gender	age	hypertension	Heart disease	avg_glucose level	bmi	Smoking status	stroke
9046	1	67.0	0	1	228.69	36.6	0	1
51676	0	61.0	0	0	202.21	28.9	1	1
31112	1	80.0	0	1	105.92	32.5	1	1
60182	0	49.0	0	0	171.23	34.4	2	1
1665	0	79.0	1	0	174.12	24.0	1	1
56669	1	81.0	0	0	186.21	29.0	0	1
53882	1	74.0	1	1	70.09	27.4	1	1
10434	0	69.0	0	0	94.39	22.8	1	1
27419	0	59.0	0	0	76.15	28.9	1	1
60491	0	78.0	0	0	58.57	24.2	1	1

4.2.1.3 Feature Scaling

Dataset yang telah dilakukan proses standardisasi *standardscaler* sebagaimana pada Tabel 4.6. Beberapa *feature* memiliki skala data yang berbeda. Ini adalah teknik untuk menstandarkan variabel x (fitur) yang ada dalam data dalam rentang tetap. Hal ini perlu dilakukan sebelum melatih model. Dua teknik paling populer untuk menskalakan data numerik sebelum pemodelan adalah normalisasi dan standardisasi. Normalisasi menskalakan setiap variabel input secara terpisah ke rentang 0-1, yang merupakan rentang untuk nilai *floating-point* di mana kita memiliki presisi paling tinggi[55]. Standardisasi menskalakan setiap variabel input

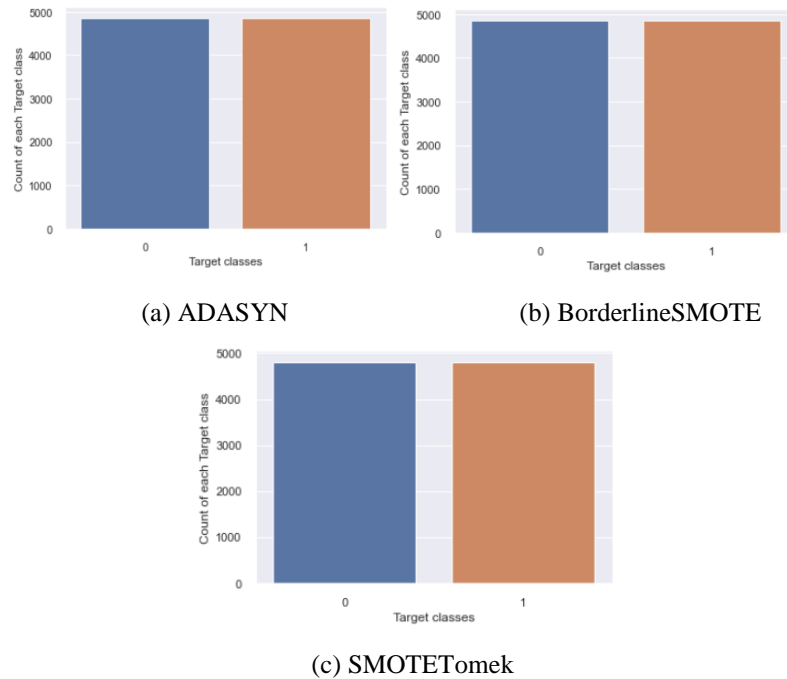
secara terpisah dengan mengurangi rata-rata (disebut pemusatan) dan membagi dengan simpangan baku untuk menggeser distribusi agar memiliki rata-rata nol dan simpangan baku satu. Proses Standardisasi dilakukan dengan menggunakan *sklearn libraries*.

Tabel 4.6. *Dataset setelah Feature Scaling*

	gender	age	hypertension	Heart disease	...	avg_glucose level	bmi	smoking_status
9046	1.1899	1.0514	-0.328602	4.185032	...	2.706375	1.0012e+00	-1.715261
51676	-0.8403	0.7860	-0.328602	-0.238947	...	2.121559	1.3846e-15	0.032841
31112	1.1899	1.6263	-0.328602	4.185032	...	-0.005028	4.6857e-01	0.032841
60182	-0.8403	0.2553	-0.328602	-0.238947	...	1.437358	7.1541e-01	1.780943
1665	-0.8403	1.5821	3.043196	-0.238947	...	1.501184	-6.3571e-01	0.032841
56669	1.1899	1.6706	-0.328602	-0.238947	...	1.768195	1.3870e-02	-1.715261
53882	1.1899	1.3610	3.043196	4.185032	...	-0.796342	-1.9399e-01	0.032841
10434	-0.8403	1.1398	-0.328602	-0.238947	...	-0.259671	-7.9161e-01	0.032841
27419	-0.8403	0.6976	-0.328602	-0.238947	...	-0.662506	1.3846e-15	0.032841
60491	-0.8403	1.5379	-0.328602	-0.238947	...	-1.050764	-6.0972e-01	0.032841

4.2.1.4 SMOTE

Dataset yang digunakan untuk penelitian memiliki *target class* yang tidak seimbang dengan perbandingan antara *class 0* (tidak stroke) sebesar 95,13% dan *class 1* (stroke) sebesar 4,87%. Ketidakseimbangan kelas ini akan menjadi masalah ketika dilakukan pemodelan. Oleh karena itu, perlu dilakukan penyeimbangan pada dataset agar model yang dihasilkan nantinya tidak bias. Teknik penyeimbangan kelas dilakukan dengan teknik *oversampling* menggunakan *SMOTE*. Penelitian tidak menggunakan Random Oversampling yang hanya menduplikasi data minoritas, karena hanya model juga cenderung tidak akan mengenali data baru diluar data pelatihan. Pada eksperimen dilakukan pengujian dengan menggunakan tiga teknik *SMOTE*, yaitu: *ADASYN*, *BorderlineSMOTE*, dan *SMOTETomek*. Karena hal ini seperti menambahkan data baru ke dalam dataset yang diproses dari tetangga terdekat. Proses tersebut dilakukan dengan menggunakan *imbalance-learn libraries*. Gambar 4.7 menunjukkan *target class* sesudah dilakukan proses *oversampling SMOTE*.



Gambar 4.6. *Target Class* setelah *SMOTE*

4.2.1.5 Split Dataset

Tahapan terakhir data preparation adalah melakukan *split dataset* menjadi *data training* dan *data testing*. Proses *split dataset* menggunakan *library sklearn*, data dibagi menjadi *data training* dan *data testing* dengan proporsi 80% untuk *data training* dan 20% untuk *data testing*. *Data training* digunakan untuk *training model*, sedangkan *data testing* digunakan untuk *evaluasi model*.

4.2.2. Hasil Eksperimen Optimasi SGD

Eksperimen pada setiap model dilakukan secara bertahap berdasarkan teknik *handling imbalance class* yang diimplementasikan. Tahap pertama menggunakan teknik *handling imbalance class* dengan *ADASYN*, tahap kedua dengan *BorderlineSMOTE* dan tahap ketiga menggunakan *SMOTETomek*. Setiap tahapan penggunaan teknik *handling imbalance class*, model akan dioptimasi dengan *SGD*, *Adam* dan *RMSprop*. Hasil eksperimen dalam bentuk *Confusion Matrix* terdapat pada lampiran. Rekapitulasi hasil eksperimen disajikan dalam bentuk tabel perbandingan setiap *metric performance* yaitu *Accuracy*, *Recall*,

Precision, *F1-Score* dan Nilai *AUC* berdasarkan optimasi dan Teknik *handling imbalance class* yang diimplementasikan.

Tabel 4.7. Perbandingan *Accuracy Score SGD ADASYN*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,1	0,01	0,001
ADASYN	A	400	93.70%	92.93%	91.50%
		500	93.70%	94.52%	92.93%
		600	93.70%	94.62%	92.63%
	B	400	94.01%	94.01%	92.63%
		500	94.93%	94.01%	93.04%
		600	94.83%	94.01%	93.50%
	C	400	94.47%	93.14%	91.71%
		500	94.47%	93.14%	92.47%
		600	94.21%	93.96%	92.83%

Pada Tabel 4.7 merupakan perbandingan *accuracy score* hasil eksperimen dataset yang telah dilakukan *oversampling* dengan teknik ADASYN. Optimasi model dengan nilai *learning rate* 0.1, 0.01, dan 0.001. Jumlah *epoch* masing-masing model 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *accuracy score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 dan optimasi *learning rate* 0.1 yaitu sebesar 94.93%.

Tabel 4.8. Perbandingan *Recall Score SGD ADASYN*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,1	0,01	0,001
ADASYN	A	400	98.95%	99.27%	99.37%
		500	98.95%	99.27%	99.06%
		600	98.95%	98.95%	99.06%
	B	400	99.27%	99.27%	99.48%
		500	99.06%	99.27%	99.37%
		600	98.95%	89.60%	99.27%
	C	400	99.37%	99.48%	99.69%
		500	99.37%	99.48%	99.58%
		600	98.64%	99.48%	99.58%

Pada Tabel 4.8 merupakan perbandingan *Recall score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.1, 0.01, dan 0.001

dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400 dan optimasi *learning rate* 0.001 yaitu sebesar 99.69%.

Tabel 4.9. Perbandingan *Precision Score* SGD ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,1	0,01	0,001
ADASYN	A	400	89.32%	87.86%	85.57%
		500	89.32%	90.46%	88.00%
		600	89.32%	90.87%	87.51%
	B	400	89.60%	89.60%	87.24%
		500	91.31%	89.60%	87.95%
		600	91.22%	89.60%	88.76%
	C	400	90.29%	88.04%	85.69%
		500	90.29%	88.04%	86.93%
		600	90.40%	89.37%	87.49%

Pada Tabel 4.9 merupakan perbandingan *Precision score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 dan optimasi *learning rate* 0.1 yaitu sebesar 91.31%.

Tabel 4.10. Perbandingan *F1-Score* SGD ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,1	0,01	0,001
ADASYN	A	400	93.89%	93.22%	91.96%
		500	93.89%	94.66%	93.20%
		600	93.89%	94.74%	92.93%
	B	400	94.19%	94.19%	92.95%
		500	95.03%	94.19%	93.31%
		600	94.93%	94.19%	93.72%
	C	400	94.62%	93.41%	92.16%
		500	94.62%	93.41%	92.83%
		600	94.34%	94.15%	93.14%

Pada Tabel 4.10 merupakan perbandingan *F1-score* hasil eksperimen *oversampling* dengan teknik *ADASYN*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 dan optimasi *learning rate* 0.1 yaitu sebesar 95.03%.

Tabel 4.11. Perbandingan *AUC Score SGD ADASYN*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>ADASYN</i>	A	400	0,979	0,964	0,971
		500	0,979	0,978	0,975
		600	0,979	0,98	0,976
	B	400	0,977	0,975	0,973
		500	0,974	0,975	0,973
		600	0,977	0,975	0,976
	C	400	0,976	0,972	0,972
		500	0,976	0,972	0,973
		600	0,976	0,976	0,975

Pada Tabel 4.11 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *ADASYN*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model A dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.01 yaitu sebesar 0.98.

Tabel 4.12. Perbandingan *Accuracy Score SGD BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>BorderlineSMOTE</i>	A	400	94.76%	94.29%	92.85%
		500	94.76%	94.29%	93.42%
		600	94.76%	94.29%	93.42%
	B	400	94.29%	94.40%	93.37%
		500	94.29%	94.40%	93.78%
		600	94.29%	94.40%	93.78%
	C	400	95.27%	94.34%	92.70%
		500	95.27%	94.34%	93.16%
		600	95.27%	94.34%	93.16%

Pada Tabel 4.12 merupakan perbandingan *Accuracy score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Accuracy score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400,500, atau 600 dan optimasi *learning rate* 0.1 yaitu sebesar 95.27%. Hal ini mengindikasikan bahwa pada jumlah *epoch* 500 dan 600 sudah tidak ada *loss improve* lagi sehingga mendapatkan *accuracy score* yang sama dengan jumlah *epoch* 400.

Tabel 4.13. Perbandingan *Recall Score SGD BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>BorderlineSMOTE</i>	A	400	97.63%	97.94%	97.73%
		500	97.63%	97.94%	97.73%
		600	97.63%	97.94%	97.73%
	B	400	98.14%	97.73%	97.84%
		500	98.14%	97.73%	98.04%
		600	98.14%	97.73%	98.04%
	C	400	98.14%	98.35%	98.04%
		500	98.14%	98.35%	98.45%
		600	98.14%	98.35%	98.45%

Pada Tabel 4.13 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 500 dan 600 dan optimasi *learning rate* 0.001 yaitu sebesar 95.45%.

Pada Tabel 4.14 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. *Precision Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400,500 atau 600 dan optimasi *learning rate* 0.1 yaitu sebesar 92.79%.

Tabel 4.14. Perbandingan *Precision Score SGD BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>BorderlineSMOTE</i>	A	400	92.30%	91.26%	89.01%
		500	92.30%	91.26%	89.94%
		600	92.30%	91.26%	89.94%
	B	400	91.10%	91.59%	89.78%
		500	91.10%	91.59%	90.31%
		600	91.10%	91.59%	90.31%
	C	400	92.79%	91.03%	88.55%
		500	92.79%	91.03%	89.00%
		600	92.79%	91.03%	89.00%

Tabel 4.15. Perbandingan *F1-Score SGD BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>BorderlineSMOTE</i>	A	400	94.89%	94.48%	93.17%
		500	94.89%	94.48%	93.68%
		600	94.89%	94.48%	93.68%
	B	400	94.49%	94.56%	93.64%
		500	94.49%	94.56%	94.02%
		600	94.49%	94.56%	94.02%
	C	400	95.39%	94.55%	93.05%
		500	95.39%	94.55%	93.49%
		600	95.39%	94.55%	93.49%

Pada Tabel 4.15 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.1 yaitu sebesar 95.39%. Hal ini mengindikasikan bahwa selama penambahan *epoch* tidak terjadi peningkatan nilai *f1-score*, yang kemungkinan sudah dimulai pada *score* tersebut sebelum pada *epoch* 400.

Tabel 4.16. Perbandingan nilai *AUC SGD BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>BorderlineSMOTE</i>	A	400	0,978	0,978	0,974
		500	0,978	0,978	0,977
		600	0,978	0,978	0,977
	B	400	0,978	0,981	0,976
		500	0,978	0,981	0,977
		600	0,978	0,981	0,977
	C	400	0,977	0,979	0,973
		500	0,977	0,979	0,974
		600	0,977	0,979	0,974

Pada Tabel 4.16 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model B dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 0.981.

Tabel 4.17. Perbandingan *Accuracy Score SGD SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>SMOTETomek</i>	A	400	94.07%	94.23%	93.14%
		500	93.97%	94.80%	93.66%
		600	94.96%	94.91%	93.76%
	B	400	94.54%	94.23%	92.41%
		500	94.18%	94.80%	93.14%
		600	94.18%	94.80%	93.50%
	C	400	94.07%	94.18%	92.10%
		500	94.85%	94.18%	92.36%
		600	94.85%	94.28%	93.14%

Pada Tabel 4.17 merupakan perbandingan *Accuracy Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa *Accuracy Score* terbaik didapatkan oleh model A dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.1 yaitu sebesar 94.96%.

Tabel 4.18. Perbandingan *Recall Score* SGD *SMOTETomek*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,1	0,01	0,001
<i>SMOTETomek</i>	A	400	98.83%	98.94%	99.15%
		500	98.62%	99.04%	99.25%
		600	98.94%	99.36%	99.25%
	B	400	99.15%	98.94%	98.83%
		500	98.19%	99.15%	99.25%
		600	98.19%	99.15%	99.36%
	C	400	98.94%	99.04%	99.15%
		500	98.30%	99.25%	98.83%
		600	98.30%	99.47%	99.15%

Pada Tabel 4.18 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.01 yaitu sebesar 99.47%.

Tabel 4.19. Perbandingan *Precision Score* SGD *SMOTETomek*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,1	0,01	0,001
<i>SMOTETomek</i>	A	400	90.01%	90.19%	88.25%
		500	89.99%	91.09%	89.02%
		600	91.44%	91.02%	89.19%
	B	400	90.56%	90.19%	87.30%
		500	90.66%	91.01%	88.17%
		600	90.66%	91.01%	88.69%
	C	400	89.93%	90.03%	86.60%
		500	91.75%	89.87%	87.22%
		600	91.75%	89.89%	88.25%

Pada Tabel 4.19 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik didapatkan oleh

model C dengan jumlah *epoch* 500 dan 600, dengan optimasi *learning rate* 0.1 yaitu sebesar 91.75%.

Tabel 4.20. Perbandingan *F1-Score* SGD *SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>SMOTETomek</i>	A	400	94.21%	94.36%	93.38%
		500	94.11%	94.90%	93.86%
		600	95.04%	95.01%	93.95%
	B	400	94.66%	94.36%	92.71%
		500	94.27%	94.90%	93.39%
		600	94.27%	94.90%	93.72%
	C	400	94.22%	94.32%	92.45%
		500	94.91%	94.33%	92.66%
		600	94.91%	94.44%	93.38%

Pada Tabel 4.20 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model A dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.1 yaitu sebesar 95.04%.

Tabel 4.21. Perbandingan nilai *AUC* SGD *SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,1	0,01	0,001
<i>SMOTETomek</i>	A	400	0,978	0,98	0,973
		500	0,977	0,98	0,976
		600	0,98	0,979	0,977
	B	400	0,978	0,978	0,971
		500	0,978	0,979	0,975
		600	0,978	0,979	0,978
	C	400	0,974	0,977	0,975
		500	0,98	0,977	0,975
		600	0,98	0,979	0,976

Pada Tabel 4.21 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.1, 0.01, dan 0.001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch*

dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model A dan C dengan jumlah *epoch* untuk Model A sebesar 400 dan 500 dengan optimasi *learning rate* 0.01 dan jumlah *epoch* 600 dengan *learning rate* 0.1. Sedangkan untuk Model C jumlah *epoch* 500 dan 600 dengan *learning rate* 0.1. Nilai *AUC* sebesar 0.98.

4.2.3. Hasil Eksperimen Optimasi Adam

Eksperimen tahap kedua menggunakan optimasi Adam. Hasil rekapitulasi optimasi Adam untuk masing-masing teknik *handling imbalance class* dijelaskan sebagai berikut:

Tabel 4.22. Perbandingan Accuracy Score Adam ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
ADASYN	A	400	94.62%	94.62%	93.19%
		500	94.62%	94.62%	94.06%
		600	94.62%	94.62%	94.06%
	B	400	94.16%	94.62%	93.19%
		500	94.16%	94.62%	93.55%
		600	94.16%	94.62%	93.55%
	C	400	93.60%	95.08%	92.68%
		500	93.60%	95.08%	92.83%
		600	94.73%	95.08%	94.27%

Pada Tabel 4.22 merupakan perbandingan *accuracy score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *accuracy score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 dan 600 dengan optimasi *learning rate* 0.001 yaitu sebesar 95.08%. Hal ini mengindikasikan bahwa pada jumlah *epoch* 500 dan 600 sudah tidak terjadi *loss improve* ketika *training*, sehingga mendapatkan *accuracy score* yang sama dengan jumlah *epoch* 400. Selain itu juga mengindikasikan bahwa *accuracy score* tersebut sudah didapatkan sebelum *epoch* 400.

Tabel 4.23. Perbandingan *Recall Score* Adam ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
ADASYN	A	400	97.38%	98.95%	98.95%
		500	97.38%	98.95%	99.06%
		600	97.38%	98.95%	99.06%
	B	400	99.48%	99.16%	99.27%
		500	99.48%	99.16%	99.37%
		600	99.48%	99.16%	99.37%
	C	400	99.06%	98.85%	99.69%
		500	99.06%	98.85%	99.58%
		600	97.91%	98.85%	99.58%

Pada Tabel 4.23 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400 dan optimasi *learning rate* 0.0001 yaitu sebesar 99.69%.

Tabel 4.24. Perbandingan *Precision Score* Adam ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
ADASYN	A	400	92.08%	90.87%	88.48%
		500	92.08%	90.87%	89.84%
		600	92.08%	90.87%	89.84%
	B	400	89.71%	90.71%	88.27%
		500	89.71%	90.71%	88.77%
		600	89.71%	90.71%	88.77%
	C	400	89.08%	91.74%	87.18%
		500	89.08%	91.74%	87.49%
		600	91.85%	91.74%	89.80%

Pada Tabel 4.24 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik didapatkan oleh

model A dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 92.08%.

Tabel 4.25. Perbandingan *F1-Score* Adam ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
ADASYN	A	400	94.66%	94.74%	93.43%
		500	94.66%	94.74%	94.22%
		600	94.66%	94.74%	94.22%
	B	400	94.34%	94.75%	93.45%
		500	94.34%	94.75%	93.77%
		600	94.34%	94.75%	93.77%
	C	400	93.80%	95.16%	93.01%
		500	93.80%	95.16%	93.14%
		600	94.78%	95.16%	94.44%

Pada Tabel 4.25 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.001 yaitu sebesar 95.16%.

Tabel 4.26. Perbandingan nilai AUC Adam ADASYN

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
ADASYN	A	400	0,976	0,978	0,971
		500	0,976	0,978	0,974
		600	0,976	0,978	0,974
	B	400	0,976	0,972	0,972
		500	0,976	0,972	0,974
		600	0,976	0,972	0,974
	C	400	0,974	0,981	0,97
		500	0,974	0,981	0,973
		600	0,977	0,981	0,974

Pada Tabel 4.26 merupakan perbandingan nilai AUC hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.01, 0.001, dan

0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.001 yaitu sebesar 0.981. Nilai ini sangat dipengaruhi oleh *precision* dan *recall* yang merupakan *True Positive Rate* dan *False Positive Rate*, dan juga komposisi hasil prediksi pada *confusion matrix*.

Tabel 4.27. Perbandingan *Accuracy Score* Adam *BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	95.32%	95.17%	93.88%
		500	95.63%	95.17%	93.88%
		600	95.63%	95.17%	93.88%
	B	400	95.73%	94.86%	93.78%
		500	95.73%	94.86%	93.78%
		600	95.73%	94.86%	93.78%
	C	400	94.86%	95.01%	93.78%
		500	94.86%	95.01%	93.78%
		600	94.86%	95.01%	93.78%

Pada Tabel 4.27 merupakan perbandingan *Accuracy Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Accuracy Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.73%. Hal ini mengindikasikan bahwa pada jumlah *epoch* 500 dan 600 sudah tidak terjadi *loss improve* ketika *training*, sehingga mendapatkan *accuracy score* yang sama dengan jumlah *epoch* 400. Hal ini juga mengindikasikan kemungkinan *accuracy score* sudah didapatkan pada *epoch* sebelum 400. Namun hal ini perlu dilakukan pengujian lebih lanjut dengan cara menurunkan jumlah *epoch* secara bertahap untuk mengetahui jumlah *epoch* yang tepat pada *accuracy score* tersebut.

Tabel 4.28. Perbandingan *Recall Score* Adam *BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	97.53%	98.25%	97.84%
		500	97.84%	98.25%	97.84%
		600	97.84%	98.25%	97.84%
	B	400	98.14%	97.63%	97.73%
		500	98.14%	97.63%	97.73%
		600	98.14%	97.63%	97.73%
	C	400	97.11%	98.45%	98.25%
		500	97.11%	98.45%	98.25%
		600	97.11%	98.45%	98.25%

Pada Tabel 4.28 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.001 yaitu sebesar 98.45%.

Tabel 4.29. Perbandingan *Precision Score* Adam *BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	93.39%	92.52%	90.64%
		500	93.68%	92.52%	90.64%
		600	93.68%	92.52%	90.64%
	B	400	93.61%	92.48%	90.54%
		500	93.61%	92.48%	90.54%
		600	93.61%	92.48%	90.54%
	C	400	92.90%	92.09%	90.16%
		500	92.90%	92.09%	90.16%
		600	92.90%	92.09%	90.16%

Pada Tabel 4.29 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik

didapatkan oleh model A dengan jumlah *epoch* 500 dan optimasi *learning rate* 0.01 yaitu sebesar 93.68%.

Tabel 4.30. Perbandingan *F1-Score* Adam *BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	95.41%	95.30%	94.10%
		500	95.71%	95.30%	94.10%
		600	95.71%	95.30%	94.10%
	B	400	95.82%	94.98%	94.00%
		500	95.82%	94.98%	94.00%
		600	95.82%	94.98%	94.00%
	C	400	94.96%	95.17%	94.03%
		500	94.96%	95.17%	94.03%
		600	94.96%	95.17%	94.03%

Pada Tabel 4.30 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.82%.

Tabel 4.31. Perbandingan nilai AUC Adam *BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	0,978	0,98	0,977
		500	0,979	0,98	0,977
		600	0,979	0,98	0,977
	B	400	0,981	0,98	0,977
		500	0,981	0,98	0,977
		600	0,981	0,98	0,977
	C	400	0,979	0,978	0,975
		500	0,979	0,978	0,975
		600	0,979	0,978	0,975

Pada Tabel 4.31 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model B dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 0.981. Disamping itu, data tersebut juga menunjukkan bahwa hampir pada setiap penambahan jumlah epoch tidak terjadi kenaikan nilai *AUC* kecuali hanya pada Model A dengan *learning rate* 0.01. Hal ini menunjukkan ratio True Positive Rate dengan False Positive Rate tidak mengalami perubahan.

Tabel 4.32. Perbandingan *Accuracy Score Adam SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	95.63%	95.48%	93.66%
		500	95.63%	95.53%	94.39%
		600	95.63%	95.53%	95.06%
	B	400	94.59%	94.96%	93.30%
		500	94.59%	96.00%	93.66%
		600	94.59%	96.00%	93.87%
	C	400	94.59%	95.32%	93.19%
		500	94.59%	95.58%	93.66%
		600	95.11%	95.58%	94.18%

Pada Tabel 4.32 merupakan perbandingan *Accuracy Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Accuracy Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.001 yaitu sebesar 96.00%. Hal ini mengindikasikan bahwa pada jumlah *epoch* 500 sudah tidak terjadi *loss improve* ketika *training*, sehingga mendapatkan *accuracy score* yang sama dengan jumlah *epoch* 400.

Tabel 4.33. Perbandingan *Recall Score* Adam *SMOTETomek*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	98.51%	99.57%	99.25%
		500	98.51%	98.83%	99.25%
		600	98.51%	98.83%	99.25%
	B	400	99.15%	99.15%	99.15%
		500	99.15%	99.36%	99.25%
		600	99.15%	99.36%	99.04%
	C	400	98.94%	98.72%	88.26%
		500	98.94%	98.51%	99.25%
		600	98.30%	98.51%	99.25%

Pada Tabel 4.33 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model A dengan jumlah *epoch* 400 dan optimasi *learning rate* 0.001 yaitu sebesar 99.57%.

Tabel 4.34. Perbandingan *Precision Score* Adam *SMOTETomek*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	92.96%	91.85%	89.02%
		500	92.96%	92.52%	90.22%
		600	92.96%	92.52%	91.37%
	B	400	90.65%	91.27%	88.50%
		500	90.65%	92.93%	89.02%
		600	90.65%	92.93%	89.51%
	C	400	90.81%	92.24%	88.26%
		500	90.81%	92.87%	89.02%
		600	92.21%	92.87%	89.87%

Pada Tabel 4.34 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik didapatkan oleh model A dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 92.96%.

Tabel 4.35. Perbandingan *F1-Score* Adam *SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	95.66%	95.55%	93.86%
		500	95.66%	95.57%	94.52%
		600	95.66%	95.57%	95.15%
	B	400	94.71%	95.05%	93.52%
		500	94.71%	96.04%	93.86%
		600	94.71%	96.04%	94.03%
	C	400	94.70%	95.37%	93.43%
		500	94.70%	95.61%	93.86%
		600	95.15%	95.61%	94.33%

Pada Tabel 4.35 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.001 yaitu sebesar 96.04%.

Tabel 4.36. Perbandingan nilai *AUC* Adam *SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	0,982	0,981	0,978
		500	0,982	0,981	0,978
		600	0,982	0,981	0,979
	B	400	0,978	0,981	0,977
		500	0,978	0,982	0,978
		600	0,978	0,982	0,98
	C	400	0,981	0,979	0,977
		500	0,981	0,979	0,978
		600	0,98	0,979	0,98

Pada Tabel 4.36 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model A dan B dengan jumlah *epoch* untuk model A sebesar 400, 500 atau 600 dan untuk

model B dengan jumlah epoch 500 atau 600. Optimasi *learning rate* untuk model A sebesar 0.01, sedangkan untuk model B sebesar 0.001. Nilai AUCnya sebesar 0.982.

4.2.4. Hasil Eksperimen Optimasi *RMSprop*

Eksperimen tahap ketiga menggunakan optimasi *RMSprop*. Hasil rekapitulasi optimasi *RMSprop* untuk masing-masing teknik *handling imbalance class* dengan ketiga optimasi *learning rate* dijelaskan sebagai berikut:

Tabel 4.37. Perbandingan *Accuracy Score RMSprop ADASYN*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
ADASYN	A	400	95.08%	94.67%	93.29%
		500	95.08%	94.67%	93.65%
		600	94.83%	94.67%	77.52%
	B	400	95.08%	94.47%	93.60%
		500	95.29%	94.47%	94.42%
		600	95.29%	94.47%	94.42%
	C	400	95.39%	94.78%	93.29%
		500	95.49%	94.78%	93.65%
		600	95.49%	94.78%	94.16%

Pada Tabel 4.37 merupakan perbandingan *Accuracy Score* hasil eksperimen *oversampling* dengan teknik ADASYN, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Accuracy Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.49%. Hal ini mengindikasikan bahwa pada jumlah *epoch* 600 sudah tidak terjadi *loss improve* ketika *training*, sehingga mendapatkan *accuracy score* yang sama dengan jumlah *epoch* 500. Selain itu, untuk mengetahui jumlah *epoch* yang tepat pada *accuracy score* tersebut, harus dilakukan pengujian dengan menambahkan epoch secara bertahap dari *epoch* 400.

Tabel 4.38. Perbandingan *Recall Score RMSprop ADASYN*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>ADASYN</i>	A	400	97.80%	98.43%	99.27%
		500	97.80%	98.43%	99.16%
		600	96.96%	98.43%	90.37%
	B	400	96.86%	98.22%	99.27%
		500	98.64%	98.22%	99.48%
		600	98.64%	98.22%	99.48%
	C	400	96.86%	98.85%	99.37%
		500	97.91%	98.85%	99.16%
		600	97.91%	98.85%	99.37%

Pada Tabel 4.38 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *ADASYN*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.0001 yaitu sebesar 99.48%.

Tabel 4.39. Perbandingan *Precision Score RMSprop ADASYN*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>ADASYN</i>	A	400	92.57%	91.35%	88.43%
		500	92.57%	91.35%	89.09%
		600	92.79%	91.35%	71.32%
	B	400	93.34%	91.16%	88.93%
		500	92.26%	91.16%	90.13%
		600	92.26%	91.16%	90.13%
	C	400	93.91%	91.21%	88.36%
		500	93.22%	91.21%	89.09%
		600	93.22%	91.21%	89.78%

Pada Tabel 4.39 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *ADASYN*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik didapatkan oleh

model C dengan jumlah *epoch* 400 dan optimasi *learning rate* 0.01 yaitu sebesar 93.91.

Tabel 4.40. Perbandingan *F1-Score RMSprop ADASYN*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>ADASYN</i>	A	400	95.11%	94.76%	93.54%
		500	95.11%	94.76%	93.86%
		600	94.83%	94.76%	79.72%
	B	400	95.07%	94.56%	93.81%
		500	95.34%	94.56%	94.57%
		600	95.34%	94.56%	94.57%
	C	400	95.36%	94.87%	93.54%
		500	95.51%	94.87%	93.86%
		600	95.51%	94.87%	94.33%

Pada Tabel 4.40 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *ADASYN*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.51%.

Tabel 4.41. Perbandingan nilai *AUC RMSprop ADASYN*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>ADASYN</i>	A	400	0,978	0,973	0,973
		500	0,978	0,973	0,974
		600	0,974	0,973	0,846
	B	400	0,982	0,973	0,974
		500	0,981	0,973	0,976
		600	0,981	0,973	0,976
	C	400	0,981	0,973	0,97
		500	0,983	0,973	0,973
		600	0,983	0,973	0,974

Pada Tabel 4.41 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *ADASYN*, optimasi *learning rate* 0.01, 0.001, dan

0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model C dengan jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 0.983.

Tabel 4.42. Perbandingan *Accuracy Score RMSprop BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	93.98%	95.12%	93.37%
		500	93.98%	95.12%	94.04%
		600	93.98%	95.12%	94.04%
	B	400	94.76%	95.17%	94.24%
		500	94.76%	95.17%	94.24%
		600	94.76%	95.17%	94.24%
	C	400	95.53%	94.81%	94.04%
		500	95.53%	94.81%	94.04%
		600	95.53%	94.81%	94.04%

Pada Tabel 4.42 merupakan perbandingan *Accuracy Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Accuracy Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.53%. Hal ini mengindikasikan bahwa pada jumlah *epoch* 500 dan 600 sudah tidak terjadi *loss improve* ketika *training*, sehingga mendapatkan *accuracy score* yang sama dengan jumlah *epoch* 400.

Pada Tabel 4.43 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.0001 yaitu sebesar 98.35%.

Tabel 4.43. Perbandingan *Recall Score RMSprop BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	96.29%	97.73%	97.84%
		500	96.29%	97.73%	97.94%
		600	96.29%	97.73%	97.94%
	B	400	97.84%	97.11%	98.35%
		500	97.84%	97.11%	98.35%
		600	97.84%	97.11%	98.35%
	C	400	98.14%	98.14%	98.04%
		500	98.14%	98.14%	98.04%
		600	98.14%	98.14%	98.04%

Tabel 4.44. Perbandingan *Precision Score RMSprop BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	92.02%	92.85%	89.78%
		500	92.02%	92.85%	90.82%
		600	92.02%	92.85%	90.82%
	B	400	92.14%	93.45%	90.86%
		500	92.14%	93.45%	90.86%
		600	92.14%	93.45%	90.86%
	C	400	93.24%	91.98%	94.04%
		500	93.24%	91.98%	90.74%
		600	93.24%	91.98%	90.74%

Pada Tabel 4.44 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.001 yaitu sebesar 93.45%. Data pada tabel tersebut juga menunjukkan bahwa hampir semua penambahan epoch tidak berpengaruh pada precision, kecuali pada Model A dan B dengan learning rate 0.0001.

Tabel 4.45. Perbandingan *F1-Score RMSprop BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	94.11%	95.23%	93.64%
		500	94.11%	95.23%	94.25%
		600	94.11%	95.23%	94.25%
	B	400	94.90%	95.25%	94.46%
		500	94.90%	95.25%	94.46%
		600	94.90%	95.25%	94.46%
	C	400	95.63%	94.96%	94.25%
		500	95.63%	94.96%	94.25%
		600	95.63%	94.96%	94.25%

Pada Tabel 4.45 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.63%.

Tabel 4.46. Perbandingan nilai *AUC RMSprop BorderlineSMOTE*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>BorderlineSMOTE</i>	A	400	0,975	0,978	0,975
		500	0,975	0,978	0,977
		600	0,975	0,978	0,977
	B	400	0,981	0,979	0,976
		500	0,981	0,979	0,976
		600	0,981	0,979	0,976
	C	400	0,982	0,978	0,974
		500	0,982	0,978	0,974
		600	0,982	0,978	0,974

Pada Tabel 4.46 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *BorderlineSMOTE*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik

didapatkan oleh model C dengan jumlah *epoch* 400, 500 atau 600 dan optimasi *learning rate* 0.01 yaitu sebesar 0.982.

Tabel 4.47. Perbandingan *Accuracy Score RMSprop SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	95.32%	95.43%	93.81%
		500	95.17%	95.43%	94.23%
		600	95.17%	95.43%	94.13%
	B	400	95.06%	95.11%	93.35%
		500	95.43%	95.11%	93.35%
		600	95.63%	95.11%	93.92%
	C	400	95.22%	95.06%	92.98%
		500	95.22%	95.06%	93.66%
		600	95.37%	95.06%	94.07%

Pada Tabel 4.47 merupakan perbandingan *Accuracy Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa *Accuracy Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.63%.

Tabel 4.48. Perbandingan *Recall Score RMSprop SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	98.40%	99.04%	99.15%
		500	98.72%	99.04%	99.04%
		600	98.72%	99.04%	98.83%
	B	400	99.36%	98.51%	99.04%
		500	97.55%	98.51%	99.04%
		600	98.40%	98.51%	99.36%
	C	400	98.30%	99.25%	99.15%
		500	98.30%	99.25%	99.25%
		600	98.94%	99.25%	99.25%

Pada Tabel 4.48 merupakan perbandingan *Recall Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan

0.0001 dan jumlah *epoch* 400, 500, dan 600. Dari data tersebut dapat dianalisis bahwa *Recall Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 400 dan optimasi *learning rate* 0.01 yaitu sebesar 99.36%.

Tabel 4.49. Perbandingan *Precision Score RMSprop SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	92.49%	92.17%	89.35%
		500	91.96%	92.17%	90.12%
		600	91.96%	92.17%	90.10%
	B	400	91.29%	92.04%	88.66%
		500	93.37%	92.04%	88.66%
		600	93.05%	92.04%	89.37%
	C	400	92.39%	91.37%	88.00%
		500	92.39%	91.37%	89.02%
		600	92.16%	91.37%	89.70%

Pada Tabel 4.49 merupakan perbandingan *Precision Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *Precision Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 500 dan optimasi *learning rate* 0.01 yaitu sebesar 93.37%.

Tabel 4.50. Perbandingan *F1-Score RMSprop SMOTETomek*

<i>Technique Handling Imbalance Class</i>	<i>Model</i>	<i>Epoch</i>	<i>Learning Rate</i>		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	95.36%	95.48%	93.99%
		500	95.22%	95.48%	94.37%
		600	95.22%	95.48%	94.26%
	B	400	95.16%	95.16%	93.56%
		500	95.42%	95.16%	93.56%
		600	95.65%	95.16%	94.10%
	C	400	95.25%	95.15%	93.24%
		500	95.25%	95.15%	93.86%
		600	95.43%	95.15%	94.24%

Pada Tabel 4.50 merupakan perbandingan *F1-Score* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa *F1-Score* terbaik didapatkan oleh model B dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.01 yaitu sebesar 95.65%.

Tabel 4.51. Perbandingan nilai *AUC RMSprop SMOTETomek*

Technique Handling Imbalance Class	Model	Epoch	Learning Rate		
			0,01	0,001	0,0001
<i>SMOTETomek</i>	A	400	0,979	0,98	0,977
		500	0,976	0,98	0,979
		600	0,976	0,98	0,978
	B	400	0,98	0,979	0,976
		500	0,981	0,979	0,976
		600	0,983	0,979	0,978
	C	400	0,98	0,977	0,977
		500	0,98	0,977	0,978
		600	0,981	0,977	0,978

Pada Tabel 4.51 merupakan perbandingan nilai *AUC* hasil eksperimen *oversampling* dengan teknik *SMOTETomek*, optimasi *learning rate* 0.01, 0.001, dan 0.0001 dan jumlah *epoch* 400, 500, dan 600. Dengan membandingkan jumlah *epoch* dan *learning rate*, dapat dianalisis bahwa nilai *AUC* terbaik didapatkan oleh model B dengan jumlah *epoch* 600 dan optimasi *learning rate* 0.01 yaitu sebesar 0.983.

Hasil analisa berdasarkan masing-masing optimasi dan teknik *handling imbalance class* dipilih satu model yang paling baik untuk setiap *metric performance*. Pada Tabel 4.52 merupakan rekapitulasi perbandingan hasil *accuracy score* berdasarkan kombinasi optimasi dan teknik *handling imbalance class*. Data pada tabel tersebut dapat dianalisis bahwa *score accuracy* terbaik didapatkan oleh Model B dengan optimasi *Adam*, teknik *handling imbalance class SMOTETomek*, jumlah *epoch* 500 atau 600 dan optimasi *learning rate* 0.001. *Accuracy Score* yang didapatkan Model B sebesar 96.00%. Hal tersebut menunjukkan bahwa dari hasil eksperimen dengan ketiga teknik *handling imbalance class* dan ketiga optimasi dan *learning rate* tidak ada yang mendapatkan *accuracy score* paling tinggi yang sama.

Tabel 4.52. Rekapitulasi Perbandingan *Accuracy Score*

Optimasi	<i>SMOTE</i>	Model	<i>Epoch</i>	LR	<i>Accuracy</i>
<i>SGD</i>	<i>ADASYN</i>	B	500	0,1	94.93%
	<i>BorderlineSMOTE</i>	C	400, 500, 600	0,1	95.27%
	<i>SMOTETomek</i>	A	600	0,1	94.96%
<i>Adam</i>	<i>ADASYN</i>	C	400, 500, 600	0,001	95.08%
	<i>BorderlineSMOTE</i>	B	400, 500, 600	0,01	95.73%
	<i>SMOTETomek</i>	B	500, 600	0,001	96.00%
<i>RMSprop</i>	<i>ADASYN</i>	C	500, 600	0,01	95.49%
	<i>BorderlineSMOTE</i>	C	400, 500, 600	0,01	95.53%
	<i>SMOTETomek</i>	B	600	0,01	95.63%

Pada Tabel 4.53 merupakan rekapitulasi perbandingan hasil *recall score* berdasarkan kombinasi optimasi dan teknik *handling imbalance class* dapat dianalisis bahwa *score* terbaik didapatkan oleh model C dengan optimasi SGD atau Adam dengan teknik *handling imbalance class* *ADASYN* dan jumlah epoch 400. Untuk *setting* nilai *learning rate* optimasi *SGD* sebesar 0.001, sedangkan untuk *setting* nilai *learning rate* optimasi *Adam* sebesar 0.0001. *Recall score* yang didapatkan model C sebesar 99.69%.

Tabel 4.53. Rekapitulasi Perbandingan *Recall Score*

Optimasi	<i>SMOTE</i>	Model	<i>Epoch</i>	LR	<i>Recall</i>
<i>SGD</i>	<i>ADASYN</i>	C	400	0,001	99.69%
	<i>BorderlineSMOTE</i>	C	500,6	0,001	98.45%
	<i>SMOTETomek</i>	C	600	0,01	99.47%
<i>Adam</i>	<i>ADASYN</i>	C	400	0,0001	99.69%
	<i>BorderlineSMOTE</i>	C	400,500,600	0,001	98.45%
	<i>SMOTETomek</i>	A	400	0,001	99.57%
<i>RMSprop</i>	<i>ADASYN</i>	B	500, 600	0,0001	99.48%
	<i>BorderlineSMOTE</i>	B	400,500,600	0,0001	98.35%
	<i>SMOTETomek</i>	B	400	0,01	99.36%

Pada Tabel 4.54 merupakan rekapitulasi perbandingan hasil *precision score* berdasarkan kombinasi optimasi dan teknik *handling imbalance class* dapat dianalisis bahwa model C dengan optimasi *RMSprop*, teknik *handling imbalance*

class ADASYN, jumlah epoch 400 dan *learning rate* 0.01 mendapatkan *score* terbaik sebesar 93.91%.

Tabel 4.54. Rekapitulasi Perbandingan *Precision Score*

Optimasi	SMOTE	Model	Epoch	LR	Precision
SGD	ADASYN	B	500	0,1	91.31%
	BorderlineSMOTE	C	400,500,600	0,1	92.79%
	SMOTETomek	C	500,6	0,1	91.75%
Adam	ADASYN	A	400,500,600	0,01	92.08%
	BorderlineSMOTE	A	600	0,01	93.68%
	SMOTETomek	A	400,500,600	0,01	92.96%
RMSprop	ADASYN	C	400	0,01	93.91%
	BorderlineSMOTE	B	400,500,600	0,001	93.45%
	SMOTETomek	B	500	0,01	93.37%

Pada Tabel 4.55 merupakan rekapitulasi perbandingan hasil *F1-score* berdasarkan kombinasi optimasi dan teknik *handling imbalance class*. Data pada tabel tersebut dapat dianalisis bahwa model B dengan optimasi *Adam*, teknik *handling imbalance class SMOTETomek*, jumlah epoch 500 atau 600, dan *learning rate* 0.001 mendapatkan *F1-score* paling baik sebesar 96.04%. Hasil eksperimen menunjukkan bahwa dengan berbagai variasi teknik *handling imbalance class* dan optimasi, hanya satu model yang mendapatkan *f1-score* paling baik.

Perbedaan yang sangat mendasar antara *precision* dengan *recall* yaitu adanya variabel *False Positif* pada *precision* dan *False Negatif* pada *recall*. Dengan semakin kecilnya variabel tersebut akan membuat masing-masing *metric* memiliki nilai yang semakin besar. Data pada Tabel 4.53 menginformasikan bahwa *recall* paling tinggi didapatkan model C dengan optimasi *SGD* dan *Adam* dengan teknik *SMOTE Adasyn*, jumlah epoch 400 dan *learning rate* 0.001 dan 0.0001. Sedangkan berdasarkan analisis Tabel 4.54, *precision* paling tinggi didapatkan Model C dengan optimasi *RMSprop* dan teknik *SMOTE Adasyn*, jumlah epoch 400 dan *learning rate* 0.01.

Memprediksi pasien terkena stroke padahal tidak stroke akan menimbulkan biaya untukantisipasi perawatan supaya kondisinya tidak menjadi lebih parah.

Memprediksi pasien tidak stroke padahal terkena stroke juga berisiko pada pasien, dan akan sangat fatal jika ternyata kondisi stroke pasien sudah parah. Terkait dengan kasus ini, menggunakan *recall* sebenarnya lebih relevan karena nilai *False Negative*-nya kecil sehingga tingkat *error*-nya juga kecil. Namun, model yang mendapatkan *recall* tinggi justru mendapatkan nilai *precision* sangat kecil, yang berarti tingkat *error* mengklasifikasikan pasien terkena stroke padahal tidak stroke menjadi sangat tinggi. Kesalahan ini akan mengakibatkan biaya yang juga tinggi untukantisipasi karena perlunya perawatan dini.

Berdasarkan analisis *precision* dan *recall*, model yang mendapatkan *recall* paling tinggi justru mendapatkan *precision* yang sangat rendah begitu juga sebaliknya untuk model yang mendapatkan *precision* yang paling tinggi mendapatkan *recall* yang rendah. Hal ini merupakan *trade-off* antara *precision* dan *recall*. Memilih model berdasarkan salah satu dari keduanya harus berani mengambil risiko atau menanggung biaya yang timbul dari tingkat *error* klasifikasi yang mungkin terjadi.

Tabel 4.55. Rekapitulasi Perbandingan *F1-Score*

Optimasi	<i>SMOTE</i>	Model	<i>Epoch</i>	LR	<i>F1-Score</i>
<i>SGD</i>	<i>ADASYN</i>	B	500	0,1	95.03%
	<i>BorderlineSMOTE</i>	C	400, 500, 600	0,1	95.39%
	<i>SMOTETomek</i>	A	600	0,1	95.04%
<i>Adam</i>	<i>ADASYN</i>	C	400, 500, 600	0,001	95.16%
	<i>BorderlineSMOTE</i>	B	400, 500, 600	0,01	95.82%
	<i>SMOTETomek</i>	B	500, 600	0,001	96.04%
<i>RMSprop</i>	<i>ADASYN</i>	C	500, 600	0,01	95.51%
	<i>BorderlineSMOTE</i>	C	400, 500, 600	0,01	95.63%
	<i>SMOTETomek</i>	B	600	0,01	95.65%

Pada Tabel 4.56 merupakan rekapitulasi perbandingan hasil nilai *AUC* berdasarkan kombinasi optimasi dan teknik *handling imbalance class* dapat dianalisis bahwa model B atau C dengan optimasi learning rate 0.01 mendapatkan nilai *AUC* terbaik sebesar 0.983. Untuk model B menggunakan teknik *handling imbalance class SMOTETomek* dengan jumlah *epoch* 600, sedangkan model C

menggunakan teknik *handling imbalance class ADASYN* dengan jumlah epoch 500 atau 600. Nilai *AUC* ini sangat dipengaruhi *True Positive Rate* dan *Falsa Positive Rate*.

Tabel 4.56. Rekapitulasi Perbandingan nilai *AUC*

Optimasi	<i>SMOTE</i>	Model	<i>Epoch</i>	LR	<i>AUC</i>
<i>SGD</i>	<i>ADASYN</i>	A	600	0.01	0,98
	<i>BorderlineSMOTE</i>	B	400,500,600	0.01	0,981
	<i>SMOTETomek</i>	A, C	400,500,600	0.1, 0.01	0,98
<i>Adam</i>	<i>ADASYN</i>	C	400,500,600	0.001	0,981
	<i>BorderlineSMOTE</i>	B	400,500,600	0.01	0,981
	<i>SMOTETomek</i>	A, B	400,500,600	0.01, 0.001	0,982
<i>RMSprop</i>	<i>ADASYN</i>	C	500, 600	0.01	0,983
	<i>BorderlineSMOTE</i>	C	400,500,600	0.01	0,982
	<i>SMOTETomek</i>	B	600	0.01	0,983

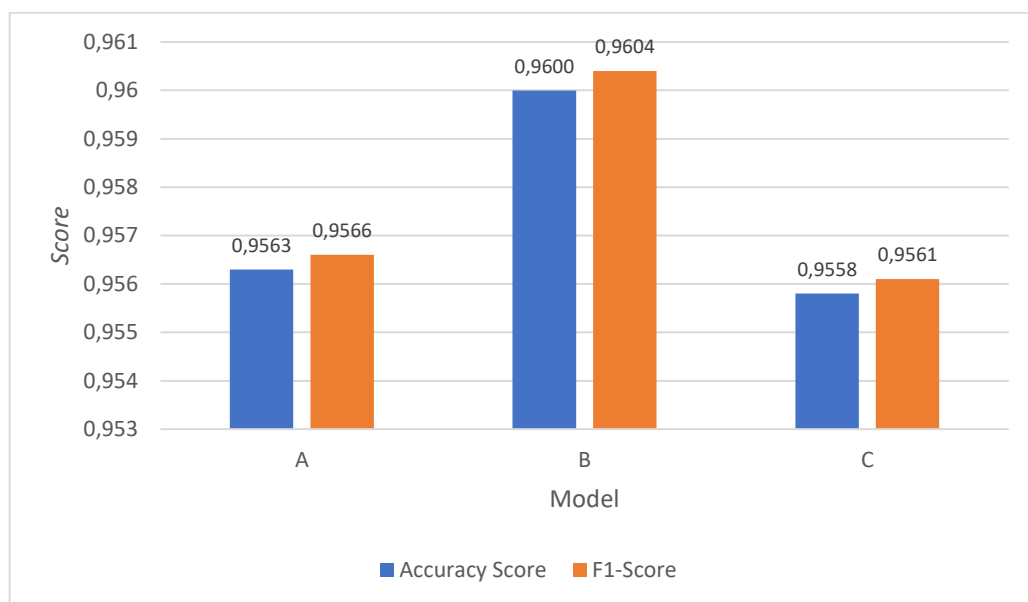
Pada Tabel 4.57 merupakan rekapitulasi hasil analisis *score* terbaik setiap *metric performance*. Agar dapat dibandingkan dengan studi literatur yang dijadikan rujukan penelitian ini, maka pemilihan model usulan prediktif berdasarkan *accuracy score*. Namun, karena dataset yang digunakan merupakan dataset yang *imbalance*, *metric accuracy score* kurang relevan karena akan menimbulkan bias, dan menggunakan salah satu dari *precision* atau *recall* untuk memilih model juga harus berani mengambil risiko yang diakibatkan dari tingkat *error* klasifikasi yang mungkin terjadi. Oleh karena itu, untuk pemilihan model selain menggunakan *accuracy score*, juga menggunakan kombinasi antara *precision* dan *recall score*, yaitu menggunakan *metric F1-score* atau *f-measure* yang merupakan nilai rata-rata harmonik antara *precision* dan *recall*.

Berdasarkan observasi pada Tabel 4.57, *score* paling baik untuk *accuracy* dan *f-measure* didapatkan oleh Model B dengan optimasi *Adam*, *learning rate* 0.001, jumlah *epoch* 500 dan teknik *handling imbalance class* menggunakan *SMOTETomek*. Sehingga model yang dipilih sebagai model prediktif stroke adalah Model B.

Tabel 4.57. Rekapitulasi *Score* Tertinggi *Metric Performance*

<i>Metric Performance</i>	Model	<i>Epoch</i>	LR	Optimasi	<i>SMOTE</i>	<i>Score</i>
Akurasi	B	500, 600	0,001	Adam	SMOTETomek	96.00%
Precision	C	400	0,01	RMSprop	ADASYN	93.91%
Recall	C	400	0,001 atau 0,0001	SGD, Adam	ADASYN	99.69%
F1-measure	B	500, 600	0,001	Adam	SMOTETomek	96.04%
AUC	B, C	500, 600	0,01	RMSprop	ADASYN, SMOTETomek	0,983

Masing-masing *score* tertinggi pada setiap model dipilih kemudian dibandingkan *accuracy* dan *f1-score*nya. Hasil perbandingan *Accuracy score* dan *f1-score* atau *f-measure* untuk ketiga model sebagaimana disajikan pada Gambar 4.7.

Gambar 4.7. Perbandingan *Accuracy* dan *F1-Score*

4.3 Model Prediktif yang diusulkan

4.3.1. Arsitektur *DNN* dan Nilai Parameter

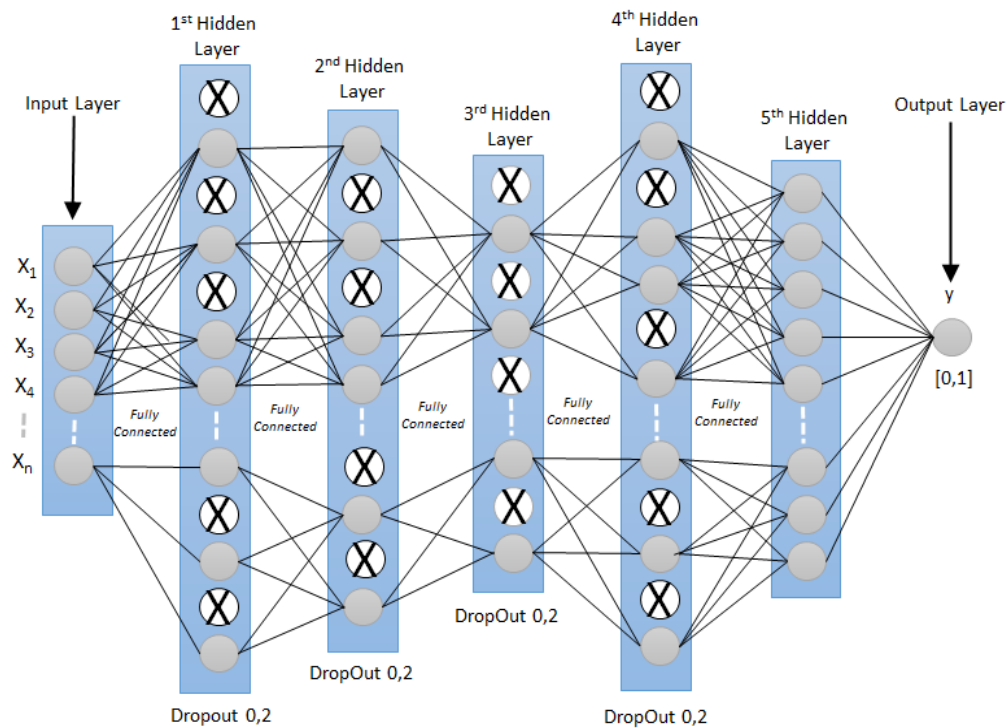
Berdasarkan analisis hasil eksperimen didapatkan model yang dipilih untuk diusulkan sebagai model prediktif adalah model B. Parameter Arsitektur model

DNN yang dipilih sebagaimana disajikan pada Tabel 4.58 dengan jumlah input *neuron* sebanyak 10 unit sesuai dengan *feature* dataset. Jumlah *hidden layer* sebanyak 5 dengan jumlah *neuron* masing-masing *layer* 320, 288, 32, 320, 32 unit. Fungsi Aktivasi *input layer* dan *hidden layer* menggunakan *ReLU*, sedangkan untuk *output layer* menggunakan *Sigmoid*. Untuk mengurangi *over-fitting* diimplementasikan dropout sebesar 0.2 pada setiap hidden layer, kecuali hidden layer terakhir. Optimasi yang digunakan Adam dengan learning rate 0.001.

Tabel 4.58. Parameter Arsitektur DNN Yang Diusulkan

Parameter	Nilai Parameter
<i>Input Neuron</i>	10
<i>Hidden Layer</i>	5
<i>Hidden Neuron</i>	320,288,32,320,32
<i>Epoch</i>	500, 600
<i>Batch Size</i>	96
Fungsi Aktivasi <i>Input dan Hidden Layer</i>	<i>ReLU</i>
Fungsi Aktivasi <i>Output Layer</i>	<i>Sigmoid</i>
<i>Dropout</i>	0.2
Optimasi	Adam (lr = 0.001)

Tahap selanjutnya yaitu membangun Arsitektur Model DNN berdasarkan parameter model yang telah dipilih dan diusulkan sebagai model prediktif. Bangunan Arsitektur Model DNN berdasarkan data Tabel 4.58 sebagaimana disajikan pada Gambar 4.8.



Gambar 4.8. Arsitektur *Deep Neural Network (DNN)*

4.3.2. *Metric Performance Model*

Berdasarkan hasil eksperimen, berikut merupakan *Score Metric Performance* Arsitektur *DNN* yang diusulkan.

4.3.2.1. *Confusion Matrix*

Pada Tabel 4.59 merupakan *confusion matrix* yang dihasilkan dari arsitektur model yang diusulkan.

Tabel 4.59. *Confusion Matrix* Model yang diusulkan

Model	Epoch	LR	TP	FP	TN	FN
B	500	0.001	933	71	914	6

4.3.2.2. *Classification Report*

Pada Tabel 4.60 merupakan *classification report* yang dihasilkan dari arsitektur model yang diusulkan.

Tabel 4.60. *Classification Report*

	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>
<i>Class 0</i>	0,99	0,93	0,96
<i>Class 1</i>	0,93	0,99	0,96
<i>Accuracy</i>			0,96
<i>Macro avg</i>	0,96	0,96	0,96
<i>Weighted Avg</i>	0,96	0,96	0,96

4.3.2.3. Accuracy

Berdasarkan data *confusion matrix* diatas, maka *Accuracy score* dapat dihitung sebagai berikut:

$$\begin{aligned}
 \text{Accuracy} &= \frac{TP + TN}{TP + FP + TN + FN} \dots\dots\dots (4.1) \\
 &= \frac{933 + 914}{933 + 71 + 914 + 6} \\
 &= 0.96
 \end{aligned}$$

Accuracy score sebesar 0.96 yang di dapat dari model prediktif yang diusulkan diinterpretasikan bahwa model mampu memprediksi dengan tingkat akurasi 96%.

4.3.2.4. Precision

Berdasarkan data *confusion matrix* diatas, maka *Precision score* dapat dihitung sebagai berikut:

a. *Class 0 (non stroke)*

$$\begin{aligned}
 \text{Precision} &= \frac{TN}{(FN + TN)} \dots\dots\dots (4.2) \\
 &= \frac{914}{(6 + 914)} \\
 &= 0.9935
 \end{aligned}$$

b. *Class 1 (stroke)*

$$\begin{aligned}
 \text{Precision} &= \frac{TP}{(FP + TP)} \dots\dots\dots (4.3) \\
 &= \frac{933}{(71 + 933)}
 \end{aligned}$$

$$= 0.9293$$

c. *Weighted Average Precision*

$$\begin{aligned} \text{Precision} &= \frac{\text{Class 0} + \text{Class 1}}{2} \dots\dots\dots (4.4) \\ &= \frac{0.9935 + 0.9293}{2} \\ &= 0.9614 \end{aligned}$$

4.3.2.5. *Recall*

Berdasarkan data *confusion matrix* diatas, maka *Recall score* dapat dihitung sebagai berikut:

a. *Class 0 (non stroke)*

$$\begin{aligned} \text{Recall} &= \frac{TN}{(TN+FP)} \dots\dots\dots (4.5) \\ &= \frac{914}{(914+71)} \\ &= 0.9279 \end{aligned}$$

b. *Class 1 (stroke)*

$$\begin{aligned} \text{Recall} &= \frac{TP}{(TP+FN)} \dots\dots\dots (4.6) \\ &= \frac{933}{(933+6)} \\ &= 0.9936 \end{aligned}$$

c. *Weighted Average Recall*

$$\begin{aligned} \text{Recall} &= \frac{\text{Class 0} + \text{Class 1}}{2} \dots\dots\dots (4.7) \\ &= \frac{0.9279 + 0.9936}{2} \\ &= 0.9608 \end{aligned}$$

4.3.2.6. *F1-score atau F-measure*

Berdasarkan hasil dari *Precision score* dan *Recall score*, *F1-score* dapat dihitung sebagai berikut:

$$\begin{aligned}
 F1 - Score &= \frac{2 \times (Precision \times Recall)}{(Precision + Recall)} \dots\dots\dots (4.8) \\
 &= \frac{2 \times (0.9614 \times 0.9608)}{(0.9614 + 0.9608)} \\
 &= 0.9611
 \end{aligned}$$

F1-score merupakan rata-rata harmonik antara *precision* dengan *recall* dan lebih relevan digunakan untuk mengukur *performance model* yang menggunakan dataset tidak seimbang untuk pelatihan model.

4.3.2.7. Kappa

Kappa untuk menghitung koefisien sebagai evaluasi persetujuan antara 2 rater[58].

$$\begin{aligned}
 k &= \frac{2 \times (TP \times TN - FN \times FP)}{(TP + FP) \times (FP + TN) + (TP + FN) \times (FN + TN)} \dots\dots\dots (4.9) \\
 &= \frac{2 \times (933 \times 914 - 6 \times 71)}{(933 + 71) \times (71 + 914) + (933 + 6) \times (6 + 914)} \\
 &= 0.92004
 \end{aligned}$$

Indeks kappa sebesar 0.92004, sehingga dapat diinterpretasikan bahwa tingkat *agreement* antara kedua *rater* adalah *excellent*.

4.3.2.8. Sensitivity

Sensitivity merupakan proporsi positif yang diidentifikasi oleh model dengan benar, atau kemungkinan test positif[59].

$$\begin{aligned}
 Sensitivity &= \frac{TP}{(TP + FN)} \dots\dots\dots (4.10) \\
 &= \frac{933}{(933 + 6)} \\
 &= 99.36\%
 \end{aligned}$$

4.3.2.9. Specificity

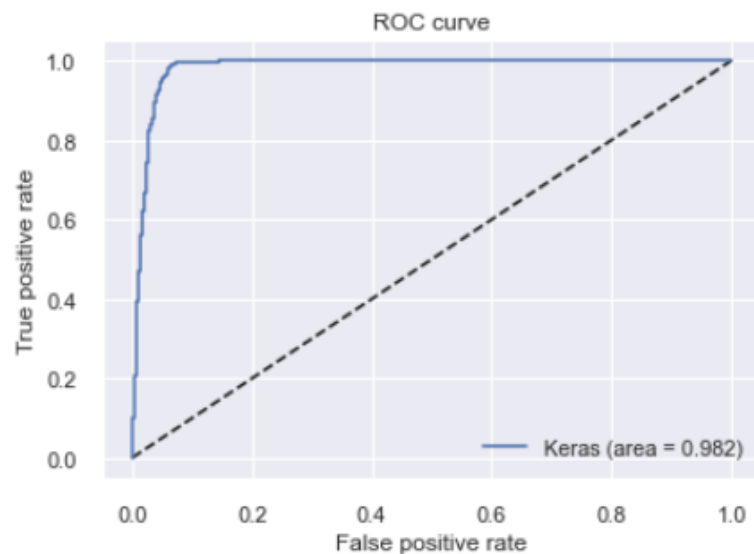
Specificity merupakan proporsi negatif yang diidentifikasi oleh model dengan benar, atau kemungkinan test negatif[59].

$$Specificity = \frac{TN}{(TN + FP)} \dots\dots\dots (4.11)$$

$$\begin{aligned}
 &= \frac{914}{(914+71)} \\
 &= 92.79\%
 \end{aligned}$$

4.3.2.10. ROC-AUC

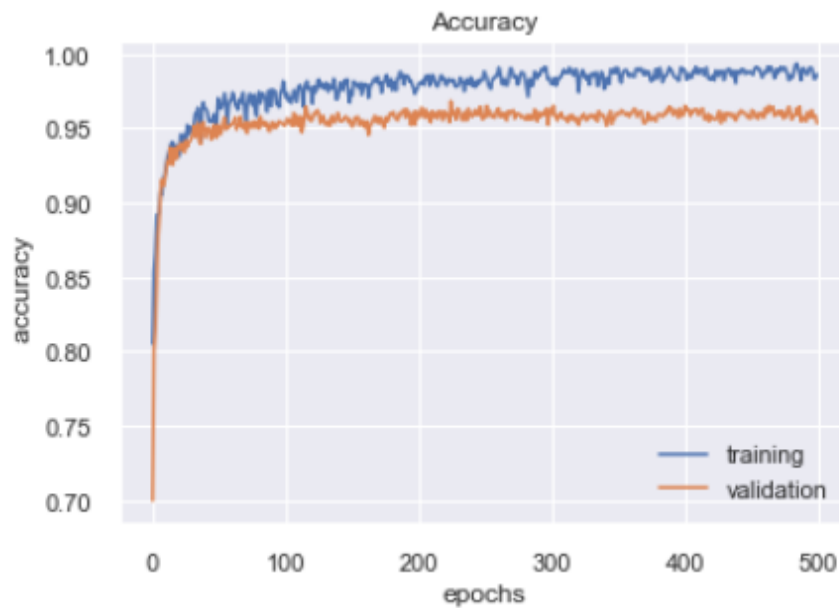
Hasil kurva ROC dan nilai AUC Arsitektur DNN yang diusulkan sebagaimana disajikan pada Grafik 4.9. Nilai AUC sebesar 0.982 atau 98.2% hampir mendekati 100% *sensitivity*. Nilai AUC dapat diinterpretasikan untuk menentukan seberapa bagus model dapat memisahkan antar kelas.



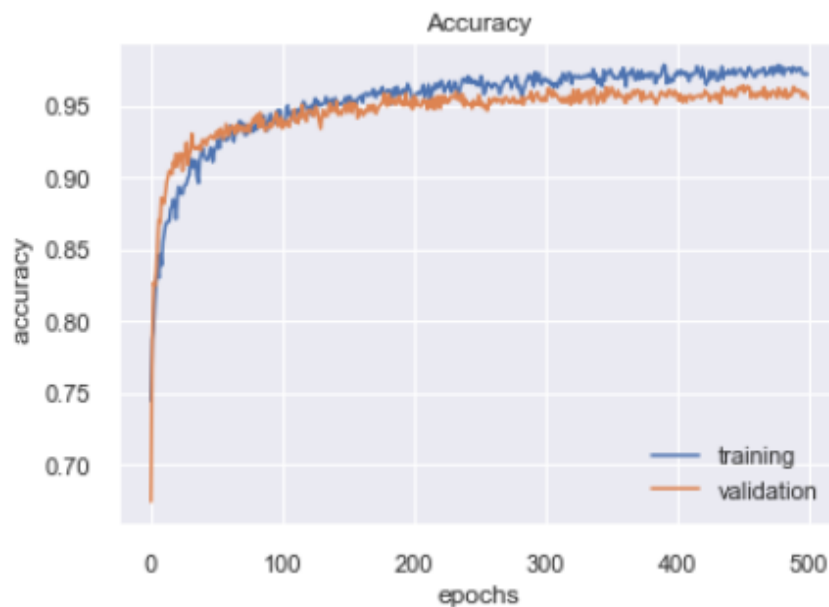
Gambar 4.9. Kurva ROC Arsitektur DNN yang Diusulkan

Berdasarkan eksperimen, untuk memastikan bahwa model yang telah dipilih tidak *over-fitting*, maka dilakukan analisis terhadap grafik *history training* dan *validation accuracy* dan *logarithmic loss*. Grafik 4.10 menunjukkan pergerakan *history training* dan *validation accuracy* sebelum diberikan nilai *Dropout*. Pada Grafik tersebut menunjukkan *accuracy score* untuk training yang terus meningkat, namun ketika dilakukan validasi selama proses *training accuracy score*-nya bergerak turun. Hal ini akan mengakibatkan model lemah dalam memprediksi data diluar pola dataset training. Grafik 4.11 menunjukkan pergerakan *history training* dan *validation accuracy* setelah diberikan nilai *Dropout*. Grafik *accuracy training*

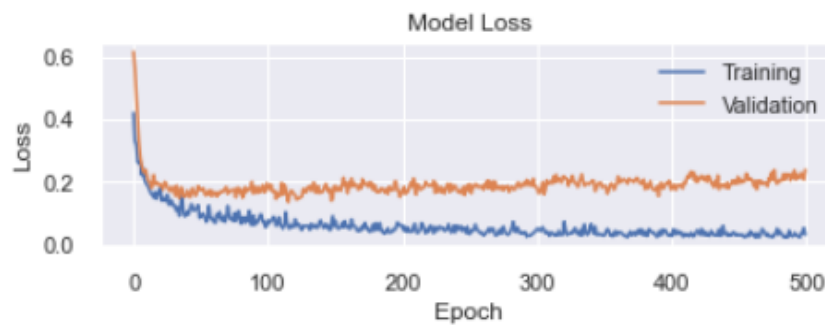
bergerak naik dan disertai dengan naiknya *accuracy validation*, hal ini mengindikasikan bahwa model tidak *over-fitting*.



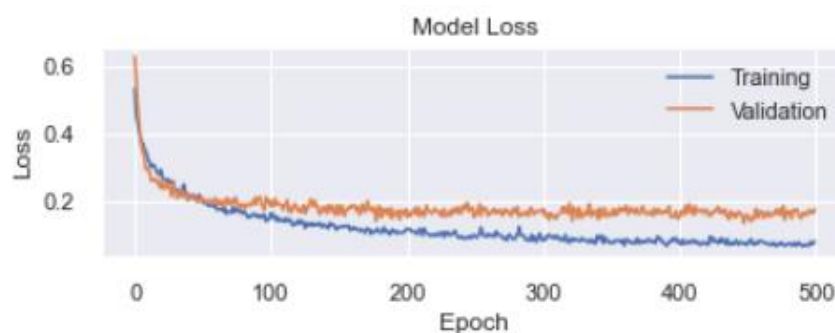
Gambar 4.10. Grafik *Accuracy Validation and Training History before Dropout*



Gambar 4.11. Grafik *Accuracy Validation and Training History after Dropout*



Gambar 4.12. Grafik *Logarithmic Loss Validation and Training History before Dropout*



Gambar 4.13. Grafik *Logarithmic Loss Validation and Training History after Dropout*

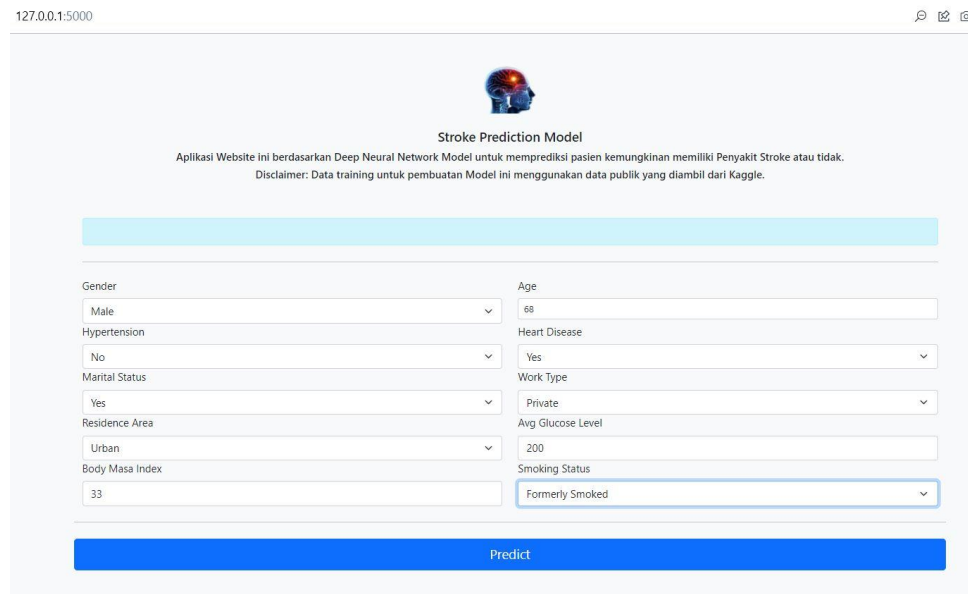
Grafik 4.4 merupakan grafik histori *logarithmic loss* selama proses *training* dan *logarithmic loss* validasi selama proses *training* sebelum diberikan nilai *dropout*. Pada grafik tersebut menunjukkan bahwa *logarithmic loss training* bergerak turun ke bawah dan nilai nya semakin kecil, namun untuk *logarithmic loss* validasi justru bergerak ke atas dan nilainya semakin besar yang mengindikasikan terjadinya *over-fitting*. Sedangkan untuk Grafik 4.5, *logarithmic loss training* bergerak turun ke bawah dan nilainya semakin kecil disertai dengan Bergeraknya *logarithmic loss* validasi yang juga bergerak dengan nilai semakin kecil.

4.4 Aplikasi Yang Dikembangkan

4.4.1. Pengembangan Aplikasi

Arsitektur model yang diusulkan telah dikembangkan menjadi sebuah aplikasi Prediksi Stroke menggunakan *Python version 3.7.6* dan *Flask version*

1.1.2. Gambar 4.14 merupakan tampilan aplikasi Prediksi Stroke yang telah dikembangkan dan dapat digunakan untuk memprediksi stroke.



127.0.0.1:5000

Stroke Prediction Model

Aplikasi Website ini berdasarkan Deep Neural Network Model untuk memprediksi pasien kemungkinan memiliki Penyakit Stroke atau tidak.
Disclaimer: Data training untuk pembuatan Model ini menggunakan data publik yang diambil dari Kaggle.

Gender: Male

Age: 68

Hypertension: No

Heart Disease: Yes

Marital Status: Yes

Work Type: Private

Residence Area: Urban

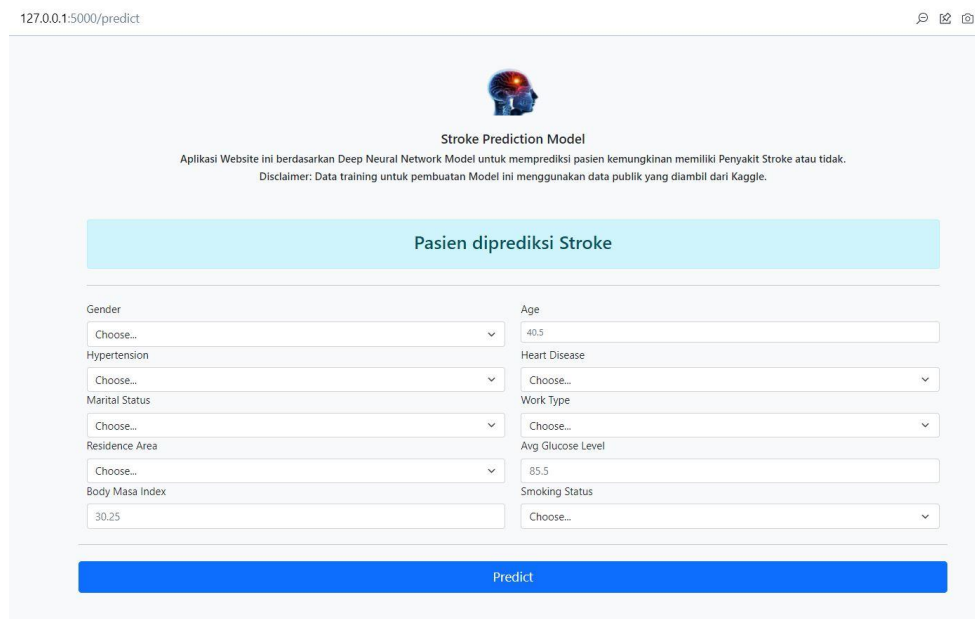
Avg Glucose Level: 200

Body Masa Index: 33

Smoking Status: Formerly Smoked

Predict

(a) Input data



127.0.0.1:5000/predict

Stroke Prediction Model

Aplikasi Website ini berdasarkan Deep Neural Network Model untuk memprediksi pasien kemungkinan memiliki Penyakit Stroke atau tidak.
Disclaimer: Data training untuk pembuatan Model ini menggunakan data publik yang diambil dari Kaggle.

Pasien diprediksi Stroke

Gender: Choose...

Age: 40.5

Hypertension: Choose...

Heart Disease: Choose...

Marital Status: Choose...

Work Type: Choose...

Residence Area: Choose...

Avg Glucose Level: 85.5

Body Masa Index: 30.25

Smoking Status: Choose...

Predict

(b) Hasil setelah dijalankan *predict*

Gambar 4.14. Aplikasi Prediksi Stroke

Untuk memastikan aplikasi yang telah di-*deployment* berfungsi dengan baik, maka perlu dilakukan percobaan dan pengujian terlebih dahulu. Pengujian sebaiknya menggunakan data yang sebenarnya terjadi pada rumah sakit karena akan sangat menggambarkan kondisi sebenarnya. Namun karena adanya kendala ketika mendapatkan data pada rumah sakit yang relevan terhadap data ini, sehingga pengujian dilakukan menggunakan data secara acak sebagaimana pada Tabel 4.61. Dari hasil pengujian tersebut terdapat data yang diprediksi stroke dan juga terdapat data yang diprediksi tidak stroke.

Tabel 4.61. Pengujian Aplikasi yang telah di-*deployment*

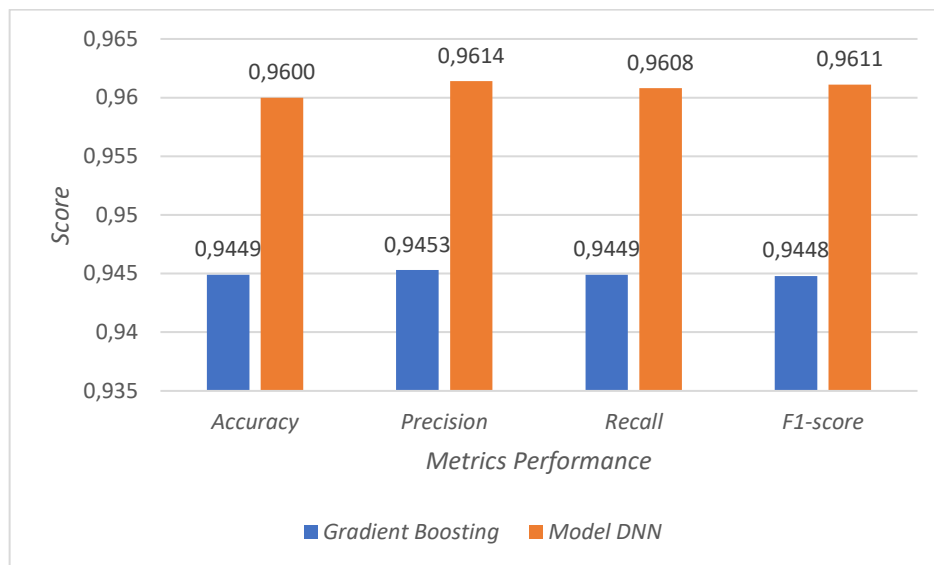
gender	age	hypertension	heart disease	ever married	Work type	Residence type	Avg glucose level	bmi	smoking status	Hasil Prediksi
Male	68.0	0	1	Yes	Private	Urban	200	33.0	formerly smoked	Stroke
Female	62.0	1	0	Yes	Self-employed	Rural	200.15	24.0	never smoked	tidak Stroke
Male	75.0	0	1	Yes	Private	Rural	103.6	24.0	never smoked	Stroke
Female	55.0	0	0	Yes	Private	Urban	170.0	32.5	smokes	Stroke
Female	40.0	1	1	Yes	Self-employed	Rural	163.4	34.5	never smoked	tidak Stroke
Female	13.00	0	0	No	children	Rural	125.5	18.2	never smoked	tidak Stroke
Male	18.00	0	0	No	Private	Urban	82.85	46.9	smokes	tidak Stroke

4.4.2. Perbandingan Dengan Penelitian Sebelumnya

Perbandingan *metrics performance Model DNN* yang diusulkan dengan penelitian prediksi stroke yang telah dilakukan sebelumnya pada penelitian [10] menggunakan algoritma *Gradient Boosting* sebagaimana disajikan pada Tabel 4.62 dan Gambar 4.15. Pada tabel dan gambar tersebut menunjukkan bahwa *score* yang didapatkan oleh Model DNN untuk *accuracy*, *precision*, *recall* dan *f1-score* lebih tinggi yaitu 0.96, 0.9614, 0.9608 dan 0.9611.

Tabel 4.62. Perbandingan *Metrics Performance*

	<i>Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>
<i>Gradient Boosting</i>	0,9449	0,9453	0,9449	0,9448
<i>Model DNN</i>	0,9600	0,9614	0,9608	0,9611

Gambar 4.15. Perbandingan *Metrics Performance*

BAB 5

PENUTUP

5.1. Kesimpulan

Berdasarkan analisis dan pembahasan hasil penelitian, dapat disimpulkan:

1. Arsitektur *Model DNN* dapat digunakan untuk memprediksi stroke.
2. Penggunaan berbagai variasi teknik *SMOTE* untuk menangani ketidakseimbangan kelas dan optimasi pada variasi *Model DNN* mendapatkan *score metrics performance* yang berbeda-beda.
3. Teknik *SMOTE Tomek link* untuk menangani ketidakseimbangan kelas mampu meningkatkan *performance* Algoritma *Neural Network*.
4. Tingkat akurasi paling baik sebesar 96% didapatkan arsitektur:
 - a. Model DNN
 - 1) Jumlah *Input Layer* sebanyak 10.
 - 2) Jumlah *Hidden Layer* sebanyak 5 dengan 320, 288, 32, 320, 32 neuron.
 - 3) Jumlah *Output Layer* sebanyak 1.
 - b. *Parameter* Model DNN
 - 1) Aktivasi *Input Layer* dan *Hidden Layer* menggunakan *ReLU*.
 - 2) Aktivasi *Output Layer* menggunakan *Sigmoid*.
 - 3) *Dropout rate*: 0.2.
 - 4) Jumlah *epoch*: 500 atau 600.
 - 5) *Batch size*: 96
 - c. *Parameter* optimasi Model DNN menggunakan *Adam* dengan *learning rate* 0.001.
5. Model *deep learning* dengan algoritma *Neural Network* yang diusulkan mendapatkan *score metric performance* lebih baik dibandingkan dengan *score metric performance* yang didapatkan oleh Algoritma *Gradient Boosting* yang dilakukan pada penelitian[10].

5.2. Saran

Peneliti menyadari bahwa dalam kegiatan penelitian selalu ada ruang untuk *improvement* baik dalam dataset maupun metode yang digunakan, oleh karena itu, untuk penelitian selanjutnya, peneliti memberikan saran:

1. Untuk mendapatkan model DNN prediksi stroke yang lebih baik dan akurat, sebaiknya menambahkan jumlah data dan atribut pada *dataset*, karena pada dunia medis memiliki data yang sangat kompleks.
2. Supaya dapat diperbandingkan Model DNN yang sudah diusulkan, agar dibuat lebih banyak variasi arsitektur model DNN dan melakukan kombinasi parameter model maupun parameter optimasi.
3. Peneliti berharap untuk penelitian prediksi stroke selanjutnya dapat dilakukan dengan menggunakan model *deep learning* lainnya seperti *Convolutional Neural Network (CNN)*, *Recurrent Neural Network (RNN)*, ataupun yang lainnya.

DAFTAR REFERENSI

- [1] WHO, “The Top 10 Causes of Death - Factsheet,” *WHO reports*, no. December 2020, pp. 1–9, 2020, [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>.
- [2] P2PTM Kemenkes RI, “Hari Stroke Sedunia 2019 : Otak Sehat, SDM Unggul,” *Kementerian Kesehatan Republik Indonesia*, 2019. <http://www.p2ptm.kemkes.go.id/artikel-sehat/hari-stroke-sedunia-2019-otak-sehat-sdm-unggul> (accessed Jun. 03, 2021).
- [3] A. Fitri, N. Masruriyah, T. Djatna, M. K. D. Hardhienata, H. H. Handayani, and D. Wahiddin, “Predictive Analytics for Stroke Disease,” pp. 13–16, 2020, doi: 10.1109/ICIC47613.2019.8985716.
- [4] P. Songram and C. Jareanpon, “A Study of Features Affecting on Stroke Prediction Using Machine Learning,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11909 LNAI, pp. 216–225, 2019, doi: 10.1007/978-3-030-33709-4_19.
- [5] S. Ray, K. Alshouli, A. Roy, A. Alghamdi, and D. P. Agrawal, “Chi-Squared Based Feature Selection for Stroke Prediction Using AzureML,” *2020 Intermt. Eng. Technol. Comput. IETC 2020*, 2020, doi: 10.1109/IETC47856.2020.9249117.
- [6] P. Chantamit-O-Pas and M. Goyal, “Prediction of Stroke Using Deep Learning Model,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10638 LNCS, pp. 774–781, 2017, doi: 10.1007/978-3-319-70139-4_78.
- [7] G. Fang, P. Xu, and W. Liu, “Automated Ischemic Stroke Subtyping Based on Machine Learning Approach,” *IEEE Access*, vol. 8, pp. 118426–118432, 2020, doi: 10.1109/ACCESS.2020.3004977.
- [8] C. C. Peng, S. H. Wang, S. J. Liu, Y. K. Yang, and B. H. Liao, “Artificial Neural Network Application to the Stroke Prediction,” *2nd IEEE Eurasia Conf. Biomed. Eng. Healthc. Sustain. 2020, ECBIOS 2020*, pp. 130–133, 2020, doi: 10.1109/ECBIOS50299.2020.9203638.

- [9] S. Cheon and J. Kim, "The Use of Deep Learning to Predict Stroke Patient Mortality," 2019, doi: doi.org/10.3390/ijerph16111876.
- [10] M. Rajora, M. Rathod, and N. S. Naik, "Stroke Prediction Using Machine Learning in a Distributed Environment," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12582 LNCS, pp. 238–252, 2021, doi: 10.1007/978-3-030-65621-8_15.
- [11] v. M. buyanov, "High Spatial Resolution Diffusion-Weighted Imaging (DWI) of Ischemic Stroke and Transient Ischemic Attacks," *Angew. Chemie Int. Ed.* 6(11), 951–952., 1967.
- [12] F. Susilawati and N. SK, "Faktor Resiko Kejadian Stroke," *J. Ilm. Keperawatan Sai Betik*, vol. 14, no. 1, p. 41, 2018, doi: 10.26630/jkep.v14i1.1006.
- [13] D. Y. Handayani and D. E. Dewi, "Analisis Kualitas Hidup Penderita dan Keluarga Pasca Serangan Stroke (dengan gejala sisa)," *Psycho Idea*, vol. 7, no. 1, pp. 35–44, 2009.
- [14] A. Imanda, S. Martini, and K. D. Artanti, "Post Hypertension and Stroke: A Case Control Study," *Kesmas*, vol. 13, no. 4, pp. 164–168, 2019, doi: 10.21109/kesmas.v13i4.2261.
- [15] L. Ghani, L. K. Mihardja, and D. Delima, "Faktor Risiko Dominan Penderita Stroke di Indonesia," *Bul. Penelit. Kesehat.*, vol. 44, no. 1, pp. 49–58, 2016, doi: 10.22435/bpk.v44i1.4949.49-58.
- [16] P. Jeatrakul, K. W. Wong, and C. C. Fung, "Using Misclassification Analysis for Data Cleaning," *IWACIII 2009 - Int. Work. Adv. Comput. Intell. Intell. Informatics*, vol. 14, no. 3, 2009.
- [17] J. Blair and M. G. Lacy, "The SAGE Social Science Collections," *Ann. Am. Acad. Pol. Soc. Sci.*, vol. 503, no. 1, pp. 122–136, 1993.
- [18] J. T. Hancock and T. M. Khoshgoftaar, "Survey on Categorical Data for Neural Networks," *J. Big Data*, vol. 7, no. 1, 2020, doi: 10.1186/s40537-020-00305-w.
- [19] T. D. K. Thara, P. S. Prema, and F. Xiong, "Auto-Detection of Epileptic Seizure Events Using Deep Neural Network with Different Feature Scaling

- Techniques,” *Pattern Recognit. Lett.*, vol. 128, pp. 544–550, 2019, doi: 10.1016/j.patrec.2019.10.029.
- [20] K. A. N. and B. S. B Santoso, H Wijayanto, “Synthetic Over Sampling Methods for Handling Class Imbalanced Problems: A Review,” *Water (Switzerland)*, vol. 26, no. 2, pp. 1–72, 2017, doi: doi:10.1088/1755-1315/58/1/012031.
- [21] K. E. Saputro, S. Fadli, K. E. Saputro, and S. Fadli, “K-means-SMOTE untuk Menangani Ketidakseimbangan Kelas dalam Klasifikasi Penyakit Diabetes dengan C4 . 5 , SVM , dan Naive Bayes K-means-SMOTE for Handling Class Imbalance in The Classification of Diabetes,” vol. 8, no. February, pp. 89–93, 2020, doi: 10.14710/jtsiskom.8.2.2020.89-93.
- [22] B. Kovács, F. Tinya, C. Németh, and P. Ódor, “SMOTE: Synthetic Minority Over-sampling Technique,” *Ecol. Appl.*, vol. 30, no. 2, pp. 321–357, 2020, doi: 10.1002/eap.2043.
- [23] H. A. Gameng, B. D. Gerardo, and R. P. Medina, “A Modified Adaptive Synthetic SMOTE Approach in Graduation Success Rate Classification A Modified Adaptive Synthetic SMOTE Approach in Graduation Success Rate Classification,” no. December 2019, pp. 4–9, 2020, doi: 10.30534/ijatcse/2019/63862019.
- [24] H. He, Y. Bai, E. A. Garcia, and S. Li, “ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning,” *Proc. Int. Jt. Conf. Neural Networks*, no. 3, pp. 1322–1328, 2008, doi: 10.1109/IJCNN.2008.4633969.
- [25] T. Gupta, J. Yadav, S. Chaudhary, and U. Agarwal, “Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning,” *Adv. Intell. Syst. Comput.*, vol. 683, pp. 232–242, 2018, doi: 10.1007/978-3-319-68385-0_20.
- [26] S. Smiti and M. Soui, “Bankruptcy Prediction Using Deep Learning Approach Based on Borderline SMOTE,” *Inf. Syst. Front.*, vol. 22, no. 5, pp. 1067–1083, 2020, doi: 10.1007/s10796-020-10031-6.
- [27] G. Goel, L. Maguire, Y. Li, and S. Mcloone, “Evaluation of Sampling Methods for Learning,” pp. 392–401, 2013.

- [28] I. Tomek, “Two Modifications of CNN,” *IEEE Trans. Syst. Man Sybernetics*, pp. 769–772, 1976.
- [29] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, “A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data,” *ACM SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 20–29, 2004, doi: 10.1145/1007730.1007735.
- [30] M. M. Mijwil, “Artificial Neural Networks Advantages and Disadvantages,” *Linkedin*, no. March, pp. 1–2, 2018, [Online]. Available: <https://www.researchgate.net/publication/323665827>.
- [31] A. M. Kairbekov, R. I. Mukhamediev, and Y. N. Yergaliyev, “Deep Neural Networks in Classification of Lithological Layers and Determining Uranium Deposits,” pp. 40–41, 2018.
- [32] I. Mackie, *Introduction to Deep Learning*. Springer, 2018.
- [33] J. Heaton, *Artificial Intelligent for Humans*. 2015.
- [34] Y. Ho and S. Wookey, “The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling,” *IEEE Access*, vol. 8, pp. 4806–4813, 2020, doi: 10.1109/ACCESS.2019.2962617.
- [35] S. Sharma and S. Sharma, “Activation Functions in Neural Networks,” vol. 4, no. 12, pp. 310–316, 2020.
- [36] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.
- [37] C. Garbin, X. Zhu, and O. Marques, “Dropout vs . Batch Normalization : An Empirical Study of Their Impact to Deep Learning,” pp. 12777–12815, 2020.
- [38] A. Labach, H. Salehinejad, and S. Valaee, “Survey of Dropout Methods for Deep Neural Networks,” 2019, [Online]. Available: <http://arxiv.org/abs/1904.13310>.
- [39] J. G. Carney and P. Cunningham, “The Epoch Interpretation of Learning,” p. 5, 1998, [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.5940&rep=rep1&type=pdf>.

- [40] G. E. G. Ap, S. H. M. Inima, J. Nocedal, P. Tak, and P. Tang, “On Large-Batch Training For Deep Learning: Generalization Gap and Sharp Minima,” pp. 1–16, 2017.
- [41] A. Sharma, “Guided Stochastic Gradient Descent Algorithm for Inconsistent Datasets,” pp. 1–26, 2018.
- [42] L. Bottou, “Stochastic Gradient Learning in Neural Networks,” *Tutorial2*, 2012.
- [43] D. P. Kingma and J. L. Ba, “Adam: A Method for Stochastic Optimization,” *Adam A Method Stoch. Optim.*, pp. 1–15, 2015.
- [44] I. J. Brownlee and A. Optimizer, “Understand the Impact of Learning Rate on Neural Network Performance,” 2021. <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/> (accessed Jul. 20, 2021).
- [45] S. L. Smith, P. Kindermans, C. Ying, Q. V Le, and G. Brain, “Don’t Decay The Learning Rate, Increase The Batch Size,” no. 2017, pp. 1–11, 2018.
- [46] F. A. Iskandarianto, “Penerapan Metode Back Propagation Neural Network pada Pendeteksian Kelainan Otak Ischemic Cerebral Infraction dengan Bahasa Pemrograman Delphi,” pp. 1–6, 2010.
- [47] W. S. Eka Putra, “Klasifikasi Citra Menggunakan Convolutional Neural Network (CNN) pada Caltech 101,” *J. Tek. ITS*, vol. 5, no. 1, 2016, doi: 10.12962/j23373539.v5i1.15696.
- [48] A. Suryanto, I. Alfarobi, and T. A. Tutupoly, “Komparasi Algoritma C4.5, Naive Bayes Dan Random Forest Untuk Klasifikasi Data Kelulusan Mahasiswa Jakarta,” *Mitra dan Teknol. Pendidik.*, vol. iv nomor 1, pp. 2–14, 2018, [Online]. Available: <https://www.publikasiilmiah.com/jurnal-mitra-dan-teknologi-pendidikan-volume-iv-nomer-1-februari-2018/>.
- [49] T. Djatna, M. Kusuma, D. Hardhienata, A. Fitri, and N. Masruriyah, “An intuitionistic Fuzzy Diagnosis Analytics for Stroke Disease,” *J. Big Data*, 2018, doi: 10.1186/s40537-018-0142-7.
- [50] Muhammad Ni’man Nasir and Irwan Budiman, “Perbandingan Pengaruh Nilai Centroid Awal Pada Algoritma K-Means Dan K-Means ++ Confusion

- Matrix,” *Semin. Nas. Ilmu Komput.*, vol. 1, pp. 118–127, 2017.
- [51] A. Tharwat, “Classification Assessment Methods,” *Appl. Comput. Informatics*, vol. 17, no. 1, pp. 168–192, 2018, doi: 10.1016/j.aci.2018.08.003.
- [52] H. C. Kraemer, “Kappa Coefficient,” *Wiley StatsRef Stat. Ref. Online*, pp. 1–4, 2015, doi: 10.1002/9781118445112.stat00365.pub2.
- [53] C. Tsai, M. Li, and W. Lin, “A Class Center Based Approach for Missing Value Imputation,” *A Cl. Cent. Based Approach Missing Value Imput.*, 2018, doi: 10.1016/j.knosys.2018.03.026.
- [54] G. Folino and F. S. Pisani, “Evolving Meta-Ensemble of Classifiers for Handling Incomplete and Unbalanced Datasets in The Cyber Security Domain,” *Appl. Soft Comput. J.*, vol. 47, pp. 179–190, 2016, doi: 10.1016/j.asoc.2016.05.044.
- [55] “Preprocessing Data.” <https://scikit-learn.org/stable/modules/preprocessing.html#normalization> (accessed Mar. 17, 2021).
- [56] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, “An Efficient Approach for Neural Network Architecture,” *2018 25th IEEE Int. Conf. Electron. Circuits Syst.*, pp. 745–748, 2018.
- [57] A. L. Invernizzi, J. Long, F. Chollet, T. O. Malley, and H. Jin, “Getting Started with KerasTuner Start the Search,” 2021. https://keras.io/guides/keras_tuner/getting_started/ (accessed May 20, 2021).
- [58] M. of L. and Ministry of Forestry, “Studi Validitas Dan Realibilitas Faktor Sukses Implementasi E-Government Berdasarkan Pendekatan Kappa,” *مجلة اسيوط للدراسات البيئية*, vol. العدد الحا, no. Mm, p. 43, 2003.
- [59] J. V. Carter, J. Pan, S. N. Rai, and S. Galandiuk, “ROC-ing along: Evaluation and Interpretation of Receiver Operating Characteristic Curves,” *Surg. (United States)*, vol. 159, no. 6, pp. 1638–1645, 2016, doi: 10.1016/j.surg.2015.12.029.

DAFTAR RIWAYAT HIDUP

A. Biodata Mahasiswa

NIM : 14002356
Nama Lengkap : Anas Faisal
Tempat, Tanggal Lahir : Sragen, 9 Mei 1978
Alamat Lengkap : Puri Serang Blok L-2 No. 5
Banjarsari, Cipocok Jaya, Kota Serang, Banten

B. Riwayat Pendidikan Formal & Non Formal

Pendidikan Formal

1. SD Negeri Plosokerep III, Lulus Tahun 1992
2. SMP Negeri I Sragen, Lulus Tahun 1995
3. SMU Negeri I Sragen, Lulus Tahun 1998
4. Sekolah Tinggi Akuntansi Negara Jakarta, Lulus Tahun 2001
5. Universitas Mercu Buana, Lulus Tahun 2005

Pendidikan Non Formal (Training)

1. Linux Fundamental, Brainmatics, 2008
2. Unified Modelling Language, Sciencom, 2009
3. PHP-Mysql, Brainmatics, 2009
4. Java Fundamental, Brainmatics, 2013
5. Kotlin, Kementerian Komunikasi dan Informatika dan Dicoding Indonesia, 2020
6. Android Fundamental, Kementerian Pariwisata dan Ekonomi Kreatif dan Dicoding Indonesia Dicoding, 2020
7. Data Scientist, Kementerian Komunikasi dan Informatika dan DqLab, 2021

C. Pengalaman Bekerja

1. Kanwil X Direktorat Jenderal Anggaran Serang (2002 – 2007) – Pelaksana
2. Direktorat Sistem Perbendaharaan, Direktorat Jenderal Perbendaharaan (2007 – 2012) – *Developer*
3. Direktorat Sistem Perbendaharaan, Direktorat Jenderal Perbendaharaan (2013 – 2015) – *Analyst*
4. Kantor Pelayanan Perbendaharaan Negara Marisa (2016 - 2017) – Kepala Seksi Verifikasi Akuntansi dan Kepatuhan Internal
5. Direktorat Pembinaan Pengelolaan Keuangan Badan Layanan Umum (2017 - Sekarang) – Kepala Seksi Informasi BLU

D. Kemampuan

- ✓ Software
Windows, Linux, Microsoft Office, Open Office
- ✓ Bahasa Pemrograman & Database
PHP, JAVA, Node.js, Python, Oracle, MySQL

E. Projek Aplikasi

- Aplikasi Bendahara Umum
- Aplikasi *I-Account* APBN
- Aplikasi Sistem Informasi Kredit Program (SIKP)
- BLU *Integrated Online Sistem* (BIOS)



Jakarta, 20 Agustus 2021

A handwritten signature in black ink, consisting of stylized, overlapping loops and lines.

Anas Faisal



LEMBAR BIMBINGAN TESIS

Universitas Nusa Mandiri

NIM : 14002356
Nama Lengkap : Anas Faisal
Dosen Pembimbing : Dr. Agus Subekti, M.T
Judul Tesis : *Deep Neural Network* untuk Prediksi Stroke

No	Tanggal Bimbingan	Materi Bimbingan	Paraf Dosen Pembimbing
1	6 April 2021	Bimbingan Perdana	
2	13 April 2021	Pembahasan Paper Terkait dan Fokus Eksperimen	
3	4 Juni 2021	Penyampaian Hasil Eksperimen	
4	8 Juni 2021	Pembahasan Hasil Eksperimen	
5	13 Juni 2021	Analisis Hasil Eksperimen	
6	27 Juni 2021	Drafting Paper	
7	5 Juli 2021	Perhitungan dan Analisa Data	
8	20 Juli 2021	Pengajuan Bab 3	
9	22 Juli 2021	Persetujuan Bab 3	
10	23 Juli 2021	Pengajuan Bab 4	
11	24 Juli 2021	Pengajuan 5	
12	25 Juli 2021	Persetujuan Bab 4 dan 5, serta Pengajuan Bab 1	
13	26 Juli 2021	Persetujuan Bab 1	
14	30 Juli 2021	Pengajuan dan Persetujuan Bab 2	
15	1 Agustus 2021	Persetujuan Semua Bab	

Bimbingan Tesis

- Dimulai pada tanggal : 6 April 2021
- Diakhiri pada tanggal : 1 Agustus 2021
- Jumlah pertemuan bimbingan : 15 (lima belas) kali pertemuan

Disetujui oleh,
Dosen Pembimbing

Dr. Agus Subekti, M.T

Lampiran 1. *Confusion Matrix* Optimasi *Stochastic Gradient Descent*

MODEL	Epoch	LR	ADASYN				BLSMOTE				SMOTETomek			
			TP	FP	TN	FN	TP	FP	TN	FN	TP	FP	TN	FN
A	400	0,1	945	113	885	10	947	79	896	23	928	103	882	11
		0,01	948	131	867	7	950	91	884	20	929	101	884	10
		0,001	949	160	838	6	948	117	858	22	931	124	861	8
	500	0,1	945	113	885	10	947	79	896	23	926	103	882	13
		0,01	948	100	898	7	950	91	884	20	930	91	894	9
		0,001	946	129	869	9	948	106	869	22	932	115	870	7
	600	0,1	945	113	885	10	947	79	896	23	929	87	898	10
		0,01	945	95	903	10	950	91	884	20	933	92	893	6
		0,001	946	135	863	9	948	106	869	22	932	113	872	7
B	400	0,1	948	110	888	7	953	93	882	18	931	97	888	8
		0,01	948	110	888	7	948	87	888	22	929	101	884	10
		0,001	950	139	859	5	948	108	867	21	928	135	850	11
	500	0,1	946	90	908	9	952	93	882	18	922	95	890	17
		0,01	948	110	888	7	948	87	888	22	931	92	893	8
		0,001	949	130	868	6	951	102	873	19	932	125	860	7
	600	0,1	945	91	907	10	952	93	882	18	922	95	890	17
		0,01	948	110	888	7	948	87	888	22	931	92	893	8
		0,001	948	120	878	7	951	102	873	19	933	119	866	6
C	400	0,1	949	102	896	6	952	74	901	18	929	104	881	10
		0,01	950	129	869	5	954	94	881	16	930	103	882	9
		0,001	952	159	839	3	951	123	852	19	931	144	841	8
	500	0,1	949	102	896	6	952	74	901	18	923	83	902	16
		0,01	950	129	869	5	954	94	881	16	932	105	880	7
		0,001	951	143	855	4	955	118	857	15	928	136	849	11
	600	0,1	942	100	898	13	952	74	901	18	923	83	902	16
		0,01	950	113	885	5	954	94	881	16	934	105	880	5
		0,001	951	136	862	4	955	118	857	15	931	124	861	8

Lampiran 2. *Confusion Matrix* Optimasi Adam

MODEL	Epoch	LR	ADASYN				BLSMOTE				SMOTETomek			
			TP	FP	TN	FN	TP	FP	TN	FN	TP	FP	TN	FN
A	400	0,01	930	80	918	25	946	67	908	24	925	70	915	14
		0,001	945	95	903	10	953	77	898	17	935	83	902	4
		0,0001	945	123	875	10	949	98	877	21	932	115	870	7
	500	0,01	930	80	918	25	949	64	911	21	925	70	915	14
		0,001	945	95	903	10	953	77	898	17	928	75	910	11
		0,0001	946	107	891	9	949	98	877	21	932	101	884	7
	600	0,01	930	80	918	25	949	64	911	21	925	70	915	14
		0,001	945	95	903	10	953	77	898	17	928	75	910	11
		0,0001	946	107	891	9	949	98	877	21	932	88	897	7
B	400	0,01	950	109	889	5	952	65	910	18	931	96	889	8
		0,001	947	97	901	8	947	77	898	23	931	89	896	8
		0,0001	948	126	872	7	948	99	876	22	931	121	864	8
	500	0,01	950	109	889	5	952	65	910	18	931	96	889	8
		0,001	947	97	901	8	948	77	898	23	933	71	914	6
		0,0001	949	120	878	6	948	99	876	22	932	115	870	7
	600	0,01	950	109	889	5	952	65	910	18	931	96	889	8
		0,001	947	97	901	8	947	77	898	23	933	71	914	6
		0,0001	949	120	878	6	948	99	876	22	930	109	876	9
C	400	0,01	946	116	882	9	942	72	903	28	929	94	891	10
		0,001	944	85	913	11	955	82	893	15	927	78	907	12
		0,0001	952	140	858	3	953	104	871	17	932	124	861	7
	500	0,01	946	116	882	9	942	72	903	28	929	94	891	10
		0,001	944	85	913	11	955	82	893	15	925	71	914	14
		0,0001	951	136	862	4	953	104	871	17	932	115	870	7
	600	0,01	935	83	915	20	942	72	903	28	923	78	907	16
		0,001	944	85	913	11	955	82	893	15	925	71	914	14
		0,0001	951	108	890	4	953	104	871	17	932	105	880	7

Lampiran 3. *Confusion Matrix Optimasi Root Mean Squared Propagation*

MODEL	Epoch	LR	ADASYN				BLSMOTE				SMOTETomek			
			TP	FP	TN	FN	TP	FP	TN	FN	TP	FP	TN	FN
A	400	0,01	934	75	923	21	934	81	894	36	924	75	910	15
		0,001	940	89	909	15	948	73	902	22	930	79	906	9
		0,0001	948	124	874	7	949	108	867	21	931	111	874	8
	500	0,01	934	75	923	21	934	81	894	36	927	81	904	12
		0,001	940	89	909	15	948	73	902	22	930	79	906	9
		0,0001	947	116	882	8	950	96	879	20	930	102	883	9
	600	0,01	926	72	926	29	934	81	894	36	927	81	904	12
		0,001	940	89	909	15	948	73	902	22	930	79	906	9
		0,0001	863	347	651	92	950	96	879	20	928	102	883	11
B	400	0,01	925	66	932	30	949	81	894	21	933	89	896	6
		0,001	938	91	907	17	942	66	909	28	925	80	905	14
		0,0001	948	118	880	7	954	96	879	16	930	119	866	9
	500	0,01	942	79	919	13	949	81	894	21	916	65	920	23
		0,001	938	91	907	17	942	66	909	28	925	80	905	14
		0,0001	950	104	894	5	954	96	879	16	930	119	866	9
	600	0,01	945	79	919	13	949	81	894	21	924	69	916	15
		0,001	938	91	907	17	942	66	909	28	925	80	905	14
		0,0001	950	104	894	5	954	96	879	16	933	111	874	6
C	400	0,01	925	60	938	30	952	69	906	18	923	76	909	16
		0,001	944	91	907	11	952	83	892	18	932	88	897	7
		0,0001	949	125	873	6	951	97	878	19	931	127	858	8
	500	0,01	935	68	930	20	952	69	906	18	923	76	909	16
		0,001	944	91	907	11	952	83	892	18	932	88	897	7
		0,0001	947	116	882	8	951	97	878	19	932	115	870	7
	600	0,01	935	68	930	20	952	69	906	18	929	79	906	10
		0,001	944	91	907	11	952	83	895	18	932	88	897	7
		0,0001	949	108	890	6	951	97	878	19	932	107	878	7

Lampiran 4. *Program Code*

1. Input Parameter

```
# Input file name with path
input_file_name =
'https://drive.google.com/file/d/10Ln7fjwImgs9D1Z3sQeHzgt0rxHxSjdd/view?usp=sharing'

# Target class name
input_target_class = "stroke"

# Columns to be removed
input_drop_col = "id"

# Col datatype selection
# use auto if you don't want to provide column names by data type else
use 'manual'

input_datatype_selection = 'auto'

# Categorical columns
input_cat_columns =
['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'
]

# Numerical columns
input_num_columns =
['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi']

# Encoding technique
# choose the encoding technique from 'LabelEncoder', 'OneHotEncoder',
'OrdinalEncoder' and 'FrequencyEncoder'

input_encoding = 'LabelEncoder'

# Handle missing value
# choose how to handle missing values from 'drop', 'inpute' and
'ignore'

input_treat_missing_value = 'inpute'

#which methode we choose, impute technique using 'mean' or 'mode'

input_missing_value = 'mean'

#which methode for handling imbalance
#choose methode handling imbalance using 'ADASYN', 'BLSMOTE',
'SMOTETomek'

input_handling_imbalance = 'SMOTETomek'

#which model we choose, 'A', 'B', 'C'

input_model = 'B'

#whats number of epoch we use, '400', '500', '600'
```



```

input_epoch= '500'

#which methods of optimizer we use, 'SGD', 'Adam', 'RMSprop'

input_optimasi = 'Adam'

#what is the value of the learning rate we use, '01', '001', '0001',
'00001'

input_lr = '0001'

```

2. Load library

```

# Import libraries
# Data Manipulation
import numpy as np
import pandas as pd
import tensorflow as tf
from pandas import DataFrame
from numpy.random import seed

seed(1)
import tensorflow
tensorflow.random.set_seed(2)

# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
from scipy.stats import norm, skew, kurtosis

# Machine Learning
from sklearn.preprocessing import LabelEncoder, StandardScaler,
OrdinalEncoder
from sklearn.ensemble import RandomForestClassifier
from yellowbrick.model_selection import FeatureImportances

from imblearn import FunctionSampler
from imblearn.over_sampling import BorderlinesMOTE, ADASYN
from imblearn.combine import SMOTETomek
from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from sklearn.metrics import confusion_matrix , classification_report,
accuracy_score, roc_auc_score, plot_roc_curve
from sklearn.metrics import roc_curve, recall_score, auc,
precision_score, f1_score, cohen_kappa_score
from sklearn.metrics import plot_confusion_matrix

import pickle

```

```
# Maths
import math

# Set the options
pd.set_option('display.max_rows', 800)
pd.set_option('display.max_columns', 500)
%matplotlib inline
sns.set_theme()
```

3. Data Preparation

#Missing Values

```
df['bmi'].replace(0.0, np.nan, inplace=True)
df['smoking_status'].replace('Unknown', np.nan, inplace=True)
df['gender'].replace('Other', np.nan, inplace=True)
df.loc[df.age < 10, "smoking_status"] = "never smoked"

if input_missing_value == 'mean':
    #mean
    df['bmi'].fillna(df['bmi'].mean(), inplace=True)

elif input_missing_value == 'mode':
    #modus
    b=df.filter(['bmi']).mode()
    df[['bmi']]=df[['bmi']].fillna(value=b.iloc[0])

l=df.filter(['smoking_status']).mode()
df[['smoking_status']]=df[['smoking_status']].fillna(value=l.iloc[0])

l=df.filter(['gender']).mode()
df[['gender']]=df[['gender']].fillna(value=l.iloc[0])

# Select how you wish to treat missing values according to the input
provided
if input_treat_missing_value == 'drop':

    # drop rows with missing values
    df.dropna(inplace=True)
    print(df.shape)

elif input_treat_missing_value == 'impute':

    # Impute missing values
    for col in numerical_columns:
        df[col] = df[col].fillna(df[col].mean())
    for col in categorical_columns:
        df[col] = df[col].fillna("Unknown")
elif input_treat_missing_value == 'ignore':
    print("Ignore missing values")
```

#Feature Encoding

```
# Select the encoding technique according to the input provided
if input_encoding == "LabelEncoder":
    # Use LabelEncoder function from sklearn
    le = LabelEncoder()
    df[categorical_columns] = df[categorical_columns].apply(lambda
col: le.fit_transform(col))
```

```

elif input_encoding == "OneHotEncoder":
    # Use pandas get dummies function to one hot encode
    df = pd.get_dummies(df, columns=categorical_columns)

elif input_encoding == "OrdinalEncoder":
    # Use OrdinalEncoder function from sklearn
    oe = OrdinalEncoder()
    df[categorical_columns] =
oe.fit_transform(df[categorical_columns])
elif input_encoding == "FrequencyEncoder":
    # Frequency encode
    for variable in categorical_columns:
        # group by frequency
        fq = df.groupby(variable).size()/len(df)
        # mapping values to dataframe
        df.loc[:, "{}".format(variable)] = df[variable].map(fq)

```

#Feature Scaling

```

def scale_data(data):

    scaler = StandardScaler()

    # transform data
    scaled_data = scaler.fit_transform(data)
    scaled_data = DataFrame(scaled_data)

    scaled_data.columns = data.columns

    return scaled_data

```

#Handling Imbalance Class Technique

```

if input_handling_imbalance == 'ADASYN':
    oversample = ADASYN(random_state=42)
    X_ros, y_ros = oversample.fit_resample(scaled_X, y)

elif input_handling_imbalance == 'BLSMOTE':
    oversample = BorderlineSMOTE(random_state=42)
    X_ros, y_ros = oversample.fit_resample(scaled_X, y)

elif input_handling_imbalance == 'SMOTETomek':
    oversample = SMOTETomek(random_state=42)
    X_ros, y_ros = oversample.fit_resample(scaled_X, y)

# Split the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X_ros, y_ros,
test_size = 0.2, random_state = 42, shuffle=True)

```

4. Modeling

Create Model

```

if input_model == 'A' and input_handling_imbalance == 'ADASYN':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=256, activation='relu'))
    model.add(BatchNormalization())

```

```

model.add(Dropout(rate=0.2))
model.add(Dense(units=32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.2))
model.add(Dense(units=96, activation='relu'))
# Output layer
model.add(Dense(1, activation='sigmoid'))
model.summary()
elif input_model == 'A' and input_handling_imbalance == 'BLSMOTE':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=64, activation='relu'))
    # Output layer
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
elif input_model == 'A' and input_handling_imbalance == 'SMOTETomek':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    # Output layer
    model.add(Dense(1, activation='sigmoid'))
    model.summary()

elif input_model == 'B' and input_handling_imbalance == 'ADASYN':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=288, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=320, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=96, activation='relu'))
    # Output layer
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
elif input_model == 'B' and input_handling_imbalance == 'BLSMOTE':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=288, activation='relu'))
    model.add(BatchNormalization())

```

```

model.add(Dropout(rate=0.2))
model.add(Dense(units=32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.2))
model.add(Dense(units=320, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.2))
model.add(Dense(units=64, activation='relu'))
# Output layer
model.add(Dense(1, activation='sigmoid'))
model.summary()
elif input_model == 'B' and input_handling_imbalance == 'SMOTETomek':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=288, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=320, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    # Output layer
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
elif input_model == 'C' and input_handling_imbalance == 'ADASYN':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=224, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=320, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=96, activation='relu'))
    # Output layer
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
elif input_model == 'C' and input_handling_imbalance == 'BLSMOTE':
    model = Sequential()
    model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=224, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(rate=0.2))

```

```

        model.add(Dense(units=320, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(rate=0.2))
        model.add(Dense(units=64, activation='relu'))
        # Output layer
        model.add(Dense(1, activation='sigmoid'))
        model.summary()
    elif input_model == 'C' and input_handling_imbalance == 'SMOTETomek':
        model = Sequential()
        model.add(Dense(units=320, input_dim=X_train.shape[1], activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(rate=0.2))
        model.add(Dense(units=224, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(rate=0.2))
        model.add(Dense(units=128, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(rate=0.2))
        model.add(Dense(units=32, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(rate=0.2))
        model.add(Dense(units=320, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(rate=0.2))
        model.add(Dense(units=32, activation='relu'))
        # Output layer
        model.add(Dense(1, activation='sigmoid'))
        model.summary()

#Set optimizer

if input_lr == '01':
    lr=0.1
elif input_lr == '001':
    lr=0.01
elif input_lr == '0001':
    lr=0.001
elif input_lr == '00001':
    lr=0.0001

if input_optimasi == 'SGD':
    opt = SGD(learning_rate=lr, momentum=0.9)
elif input_optimasi == 'Adam':
    opt = Adam(learning_rate=lr)
elif input_optimasi == 'RMSprop':
    opt = RMSprop(learning_rate=lr)

# Compile model
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=[accuracy])

callback_a = ModelCheckpoint(filepath = 'my_best_model.hdf5',
monitor='val_loss', verbose=1, save_best_only=True)
callback_b = EarlyStopping(monitor='val_loss', mode='min', patience=20,
verbose=1)

if input_epoch == '400':
    n_epochs = 400
elif input_epoch == '500':
    n_epochs = 500
elif input_epoch == '600':
    n_epochs = 600

batch_size = 96

```

```
history = model.fit(X_train.values, y_train, batch_size=batch_size,
epochs=n_epochs, validation_split=0.2, callbacks=[callback_a], verbose=0,
shuffle=True)
```

5. Evaluation

```
#Classification report, Accuracy, Precision, Recall, F1-score, Cohens
kappa
```

```
from sklearn.metrics import confusion_matrix ,
classification_report, accuracy_score, roc_auc_score, plot_roc_curve
from sklearn.metrics import roc_curve, recall_score, auc,
precision_score, f1_score, cohen_kappa_score
prediction = model.predict(X_test.values)
y_preds = np.round(prediction)
accuracy = accuracy_score(y_test, prediction.round())
precision = precision_score(y_test, prediction.round())
recall = recall_score(y_test, prediction.round())
f1_score = f1_score(y_test, prediction.round())
#print(prediction)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
print("Precision: %.2f%%" % (precision * 100.0))
print("Recall: %.2f%%" % (recall * 100.0))
print("F1-score: %.2f%%" % (f1_score * 100.0))
kappa = cohen_kappa_score(y_test, y_preds)
print('Cohens kappa: %f' % kappa )
print("\n Confusion Matrix : \n ",confusion_matrix(y_test, y_preds))
print("Classification Report: \n", classification_report(y_test,
y_preds))
```

```
#Plotting ROC-AUC
y_pred = model.predict(X_test).ravel()
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred)
auc_keras = auc(fpr_keras, tpr_keras)
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Keras (area =
{:.3f})'.format(auc_keras))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

```
#Plotting training and validation accuracy history
plt.plot(np.arange(len(history.history['accuracy'])),
history.history['accuracy'], label='training')
plt.plot(np.arange(len(history.history['val_accuracy'])),
history.history['val_accuracy'], label='validation')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy ')
plt.legend(loc=0)
plt.show()
```

```
#Plotting training and validation loss history
plt.subplot(212)
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.tight_layout()
plt.show()

#clear session
tf.keras.backend.clear_session()
```