BAB 2
LANDASAN TEORI

2.1 Diabetes Mellitus

Diabetes mellitus merupakan suatu grup penyakit metabolisme yang mempunyai ciri-ciri hyperglycemia yang disebabkan oleh kelainan pada kerja dan/atau produksi insulin. Gejala-gejala diabetes mellitus yang berasal dari hyperglycemia yaitu polyuria, polydipsia, penurunan berat badan, dan penglihatan kabur. Berdasarkan [9], terdapat delapan faktor penentu yang signifikan dalam mendiagnosis diabetes diantaranya:

1. Jumlah Kehamilan

   Jumlah kehamilan merupakan angka yang menunjukan kehamilan yang pernah dialami pasien.
2. Konsentrasi glukosa

   Konsentrasi glukosa adalah kadar glukosa dalam tubuh yang didapat setelah melakukan tes toleransi glukosa. Seseorang dapat didiagnosa diabetes mellitus jika konsentrasi glukosa pada 2 jam test toleransi glukosa bernilai ≥200 mg/dl [10].
3. Tekanan Darah Diastolik (mm Hg)

   Tekanan darah diastolic adalah tekanan darah pada saat jantung sedang berelaksasi dan menerima darah yang Kembali dari seluruh tubuh.
4. Ketebalan Kulit Otot Tricept (mm)

   Ketebalan kulit otot tricept ditentukan dengan mengukur ketebalan kulit pada bagian atas lengan (bagian otot tricept).
5. Insulin

   Kadar insulin setelah 2 jam dari insulin tes merupakan salah satu parameter yang digunakan untuk mendiagnosa diabetes. Jika kadar insulin dalam darah terlalu tinggi setelah tes dilakukan maka pasien tersebut kemungkinan mempunyai *insulin resistance* yang merupakan salah satu ciri diabetes [11].
6. Indeks Massa Tubuh

Indeks massa tubuh adalah nilai yang ditentukan dari berat badan dan tinggi badang seseorang. index massa tubuh (kg/m^2) dapat ditentukan dengan membagi berat badan dengan kuadrat tinggi badan. [12] menunjukan bahwa terdapat hubungan antara index massa tubuh dengan prevalensi diabetes. Semakin tinggi index massa tubuh semakin besar kemungkinan diabetes.

7. Diabetes Pedigree Function

   Diabetes pedigree function merupakan fungsi yang menghasilkan nilai pengaruh riwayat penyakit diabetes pada seseorang.

8. Umur

   Umur merupakan salah satu kriteria pengidentifikasian penyakit diabetes. Seseorang yang berumur 35 hingga 49 tahun memiliki kemungkinan diabetes lebih tinggi dibanding umur di bawahnya [13].

Diabetes mellitus dapat dibagi menjadi tiga kategori, berikut adalah tipe-tipe diabetes mellitus [14]:

1. Diabetes Mellitus Tipe 1

Diabetes mellitus tipe 1 adalah penyakit autoimun kronis yang mempunyai ciri tingginya gula darah(hyperglycemia) yang disebabkan oleh kekurangan insulin karena hilangnya β-cell pada pankreas [15].

2. Diabetes Mellitus Tipe 2

Diabetes mellitus tipe 2 adalah kondisi ketika sel tidak dapat menggunakan insulin untuk menyerap glukosa dengan baik. Hal tersebut menyebabkan meningkatnya tingkat gula darah dalam tubuh [16].

3. Gestational Diabetes Mellitus

Gestaional diabetes mellitus merupakan kondisi ketika intoleransi glukosa muncul pada masa kehamilan [17].

2.2 Pembelajaran Mesin

Pembelajaran mesin (*Machine learning*) adalah pemrograman komputer yang bertujuan untuk mengoptimalkan kinerja prediksi dengan menggunakan

sekumpulan data sehingga dapat menghasilkan model otomatis. Pembelajaran mesin menggunakan teori statistika dalam pembangunan model [18].

2.3 Praproses Data

Praproses data adalah tahapan fundamental pada pembelajaran mesin. Data yang digunakan untuk membangun suatu model pembelajaran mesin dianalisa sedemikian rupa sehingga data tersebut siap untuk digunakan. Beberapa masalah yang sering ada pada data mentah yang akan digunakan adalah nilai yang hilang pada data, duplikasi data, data yang rasio kelas data yang tidak seimbang, dan lain-lain [19]. Adapun beberapa metode praproses data yang dapat menangani masalah pada data mentah, yaitu:

1. *Data Cleansing*

   Data Cleansing biasanya melibatkan dua proses utama yaitu menghapus data yang memiliki nilai yang hilang dan data duplikat.

2. Normalisasi Data

   Normalisasi data dilakukan dengan cara mentransformasi skala setiap fitur yang digunakan terhadap skala tertentu (seperti 0 sampai 1). Metode ini dapat meningkatkan akurasi dan efisiensi algoritma pembelajaran mesin yang melibatkan pengukuran jarak. Rumus untuk normalisasi dapat dilihat pada (2.1) [20].

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad (2.1)$$

Dimana:

X = Data yang akan dinormalisasi

X_{min} = Data terkecil pada fitur yang dinormalisasi

X_{max} = Data terbesar pada fitur yang dinormalisasi

2.4 *Feedforward Neural Network*

Feedforward neural network (FNN) adalah salah satu jenis *neural network* yang sering digunakan. *Feedforward neural network terdiri dari* tiga jenis lapisan neuron yaitu *input layer, hidden layer, dan output layer* yang membentuk *neural network* tanpa siklus. Setiap neuron pada *neural network* saling berhubungan dengan neuron pada layer sebelum atau sesudahnya. Setiap neuron (kecuali pada

input layer) berfungsi untuk menghitung masukan dari neuron pada lapisan sebelumnya dan meneruskan keluaran dari setiap neuron ke neuron pada lapisan berikutnya (*forward pass*) [21]. Setelah melakukan *forward pass, neural network* dapat menghitung nilai perubahan parameter dengan melakukan *backward pass*. Setelah nilai perubahan parameter diketahui, proses pembelajaran dapat dilakukan dengan memperbaharui nilai bobot dan bias dengan menggunakan (2.7), dan (2.8). Contoh arsitektur *feedforward neural network* dapat dilihat pada Gambar 2.1. Untuk setiap hubungan antar neuron, terdapat suatu nilai yang disebut *weight*. Keluaran dari neuron ke-*i* pada *hidden layer* dan *output layer* adalah:

$$a_i = \sigma(z) \tag{2.2}$$

$$z_i = \sum_{j=1}^{N} w_{ij} x_j + b \tag{2.3}$$

a_i = keluaran dari neuron pada neuron ke-i.

σ = fungsi aktivasi.

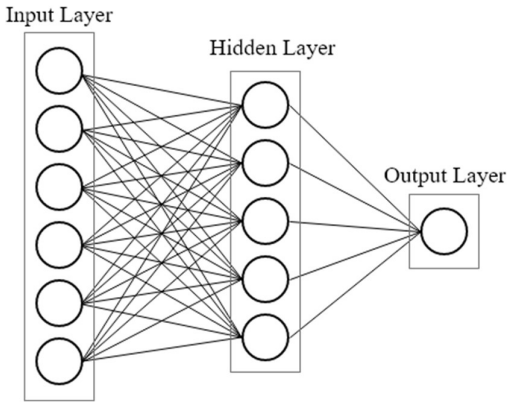
z_i = *weighted sum* pada neuron ke-i.

w_{ij} = bobot pada koneksi neuron ke-i dan neuron ke-j pada lapisan sebelumnya.

x_j = keluaran dari neuron pada neuron ke-j pada lapisan sebelumnya.

b = bobot pada neuron.

dimana x_j adalah keluaran dari neuron ke-j dari layer sebelumnya, b adalah bias untuk setiap neuron, dan σ adalah fungsi aktivasi [22].



Gambar 2.1 *Arsitektur Feedforward Neural Network*

Terdapat lebih dari satu fungsi aktivasi yang biasa digunakan pada *neural network*. Pemilihan fungsi aktivasi bergantung pada masalah yang akan diselesaikan oleh *neural network* yang akan dibangun. Berikut adalah beberapa contoh fungsi aktivasi yang biasa digunakan:

a. *Binary Step Function*

$$y = \begin{cases} 0; jika\ x \leq 0 \\ 1\ jika\ x > 0 \end{cases} \qquad (2.4)$$

b. *Rectified Linear Unit (ReLU)*

$$y = max\ (0, x) \qquad (2.5)$$

c. Fungsi *Sigmoid*

$$y = \frac{1}{1 + \ e^{-x}} \qquad (2.6)$$

Dimana x adalah masukan fungsi aktivasi *y* yang merupakan keluaran dari suatu neuron seperti yang dapat dilihat pada (2.2) dan (2.3).

Proses-proses yang terlibat pada fase pelatihan dapat dilihat pada Tabel 2.1.

Tabel 2.1 *Pseudocode algoritma pelatihan neural network*

Algoritma Pelatihan Neural Network

Require *epochs*; μ; β_1; β_2; *threshold*; *min_error*: Hyperparameter(epochs, learning rate, beta 1, beta 2, threshold classifier, minimum error).
Require data_latih: Data latih yang digunakan.

w = 3d-array dengan nilai random dari [-1, 1]; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa; dimensi ke-3 memetakan koneksi ke berapa.

b = 2d-array dengan nilai random dari [-1, 1]; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa.

for i = 1 to *epochs* **do**
    error = 0

    delta_w = 3d-array dengan nilai 0; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa; dimensi ke-3 memetakan koneksi ke berapa.

    delta_b = 2d-array dengan nilai 0; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa.

    for each *data* in *data_latih* **do**
        z, a = feedforward(*data*, *w, b*)
        tmp_delta_b, *tmp_delta_w*, *tmp_error* = backprop(*w*, *b*, *z*, *a*)
        delta_b = *delta_b* + *tmp_delta_b*/length(*data_latih*)
        delta_w = *delta_w* + *tmp_delta_w*/length(*data_latih*)
        error = *error* + *error*/length(*data_latih*)
    end for

    if *error* <= *min_error* **then**
        break
    end if

    w, b = update_parameter(*w, b*, *delta_b*, *delta_w*)
end for

2.4.1 *Forward Pass*

Tahap pelatihan *feedforward neural network* dapat dibagi menjadi dua bagian yaitu *forward pass* dan *backward pass*. Pada bagian *forward pass*, data masukan diteruskan melalui input layer hingga output layer. Aturan perambatan tersebut dapat dilihat pada (2.2) dan (2.3).

Tabel 2.2 *Pseudocode algorima feedforward*

Algoritma *Feedforward*

Require data; *w*; *b*: Data yang di*feedforward*, bobot, dan bias.

```
z = empty array
a = array of data

for i = 0 to length(w) do
        tmp_z_layer = empty array
        tmp_a_layer = empty array
        a_cursor = a[i]
        for j = 0 to length(w[i]) do
                tmp_z = 0
                for k = 0 to length(w[i][j]) do
                        tmp_z = tmp_z + a_cursor[k]*w[i][j][k]
                end for
                tmp_z = tmp_z + b[j]
                push(tmp_z_layer, tmp_z)
                tmp_a = sigmoid(a)
                push(tmp_a_layer, tmp_a)
        end for
        push(z, tmp_z_layer)
        push(a, tmp_a_layer)
end for
```

2.4.2 *Backward Pass*

Backward Pass adalah salah fase pada proses pembelajaran *neural network*. Pada tahap ini *error output* digunakan sebagai acuan untuk mengubah bobot (*weight*) dan bias pada masing-masing neuron [23]. Proses pembaharuan bobot dan bias pada *neural network* melibatkan metode yang disebut *Gradient Descent*. *Gradient descent* dapat dilakukan setelah tahap perambatan maju (*forward pass*) selesai. Aturan pembaharuan bobot dan bias dapat dilihat pada (2.7), dan (2.8).

$$w_{ij}^{(k+1)} = w_{ij}^{k} - \mu \left(\frac{\partial C}{\partial w_{ij}} \right)^{k} \qquad (2.7)$$

$$b_{i}^{(k+1)} = b_{i}^{k} - \mu \left(\frac{\partial C}{\partial b_{i}} \right)^{k} \qquad (2.8)$$

Dimana $w_{ij}^{(k+1)}$, $b_i^{(k+1)}$ adalah bobot dan bias pada iterasi selanjutnya. μ merupakan laju pembelajaran ($\mu > 0$) dan $\frac{\partial C}{\partial w_{ij}}$, $\frac{\partial C}{\partial b_i}$ adalah turunan parsial *cost function* (2.9) terhadap bobot dan bias secara berurutan pada iterasi ke-*k*. Pada *vanilla gradient descent* atau bentuk *gradient descent* yang paling dasar $\frac{\partial C}{\partial w_{ij}}$ dan $\frac{\partial C}{\partial b_i}$ merupakan jumlah nilai pembaharuan pada tiap baris data dibagi dengan jumlah data.

Tabel 2.3 *Pseudocode algoritma backpropagation*

Algoritma *backpropagation*
Require *w*; *b*; *z*; *a*: bobot, bias, nilai z, dan nilai aktivasi.

```
nabla_b = empty array of b shape
nabla_w = empty array of w shape

for i = length(w)-1 to 0 do
        for j = 0 to length(w[i]) do
                delta = cost derivative of current node * sigmoid_prime(z[i][j])
                nabla_b[i][j] = delta
                for k = 0 to length(w[i][j]) do
                        nabla_w[i][j][k] = delta * w[i][j][k]
                end for
        end for
end for
```

2.4.3 *Gradient Descent*

Neural network melakukan pembelajaran dengan cara menyesuaikan bobot dan bias pada neuron-neuron. Penyesuaian bobot dan bias tersebut merupakan hasil dari tahapan *gradient descent. Gradient descent* bertujuan untuk meminimalisir nilai *cost function* (2.9) pada *neural network* dengan cara mencari tahu gradien dari *cost function* dan mengubah bobot-bobot pada *neural network* berlawanan arah dengan besar gradien [24].

$$C = \frac{1}{2}(y - \hat{y})^2 \qquad (2.9)$$

Dimana,

$C =$ nilai *cost.*

$y =$ label pada data yang bersangkutan.

$\hat{y} =$ nilai yang diprediksi.

Nilai gradien yang digunakan untuk memperbaharui bobot dan bias dapat dilihat pada (2.10), dan (2.11).

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} \quad (2.10)$$

$$\frac{\partial C}{\partial b_i} = \frac{\partial C}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} \quad (2.11)$$

Nilai pembaharuan bobot dihitung per-koneksi bobot. Pada (2.10) diperlihatkan rumus untuk mencari nilai pembaharuan bobot pada koneksi bobot yang menghubungkan neuron layer dengan indeks terbesar dari koneksi bobot pada posisi ke-i dan neuron layer dengan indeks terkecil dari koneksi bobot pada posisi ke-j. Nilai pembaharuan bobot dicari menggunakan turunan partial dari *cost function* yang digunakan terhadap koneksi bobot yang dicari.

Sama halnya dengan nilai pembaharuan bobot, nilai pembaharuan bias dicari menggunakan rumus yang sama hanya saja *cost function* diturunkan terhadap bias yang bersangkutan.

Untuk pembaharuan bobot dan bias pada *neural network* dapat dilihat pada (2.7) dan (2.8).

2.4.3.1 Mini-batch Training

Terdapat beberapa strategi yang tersedia pada penerapan algoritma *gradient descent*. Strategi tersebut melibatkan pemilihan jumlah *batch* yang akan digunakan pada saat pelatihan. Umumnya, terdapat tiga strategi yang dapat digunakan yaitu *stochastic gradient descent*, *mini-batch gradient descent,* dan *batch gradient descent*. Masing-masing strategi ditentukan dengan jumlah *batch* yang digunakan.

(Jika m = 1, maka *gradient descent* yang digunakan adalah *stochastic gradient descent*; jika 1 < m < besar data latih, maka *gradient descent* yang digunakan adalah *mini-batch gradient descent*; jika m = besar data latih, maka *gradient descent* yang digunakan adalah *batch gradient descent*.). Rumus untuk *gradient descent* menggunakan jumlah *batch* sebesar m dapat dilihat pada (2.12) [24].

$$\frac{\partial C}{\partial w_{ij}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}} \qquad (2.12)$$

Sehingga, proses pelatihan pada neural network dapat digambarkan dengan *pseudocode* pada Tabel 2.1 menjadi:

Tabel 2.4 *Pseudocode algoritma pelatihan neural network dengan mini batch*

Algoritma Pelatihan Neural Network dengan Mini Batch

Require *epochs*; μ; β_1; β_2; *threshold*; *min_error*: Hyperparameter(epochs, learning rate, beta 1, beta 2, threshold classifier, minimum error).
Require *data_latih*: Data latih yang digunakan.

w = 3d-array dengan nilai random dari [-1, 1]; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa; dimensi ke-3 memetakan koneksi ke berapa.

b = 2d-array dengan nilai random dari [-1, 1]; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa.

mini_batches = make_mini_batch(*data_latih*).

for i = 1 to *epochs* **do**
    error = 0

    delta_w = 3d-array dengan nilai 0; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa; dimensi ke-3 memetakan koneksi ke berapa.

    delta_b = 2d-array dengan nilai 0; dimensi ke-1 memetakan layer ke berapa; dimensi ke-2 memetakan node ke berapa.

    for each *mini_batch* in *mini_batches* **do**
        for each *data* in *mini_batch* **do**
            z, a = feedforward(*data*, *w, b*)
            tmp_delta_b, *tmp_delta_w*, *tmp_error* = backprop(*w*, *b*, *z*, *a*)
            delta_b = delta_b + *tmp_delta_b*/length(*mini_batch*)

```
        delta_w = delta_w + tmp_delta_w/length(mini_batch)error = error +
        error/length(mini_batch)
    end for
    if error <= min_error then
        break
    end if
    w, b = update_parameter(w, b, delta_b, delta_w)
end for

if error <= min_error then
    break
end if

end for
```

2.4.4 Momentum

Pada umumnya, skema pembelajaran pada neural network bisa memerlukan waktu yang cukup lama. Penggunaan momentum pada proses pembelajaran *neural network* dapat mempercepat laju konvergensi secara dramatis [25]. Dengan memanfaatkan momentum terhadap rumus (2.7), rumus pembaharuan bobot dengan momentum menjadi [26]:

$$v^k = \gamma v^{(k-1)} + \mu \left(\frac{\partial C}{\partial w_{ij}} \right)^k \qquad (2.13)$$

$$w_{ij}^{(k+1)} = w_{ij}^k - v^k \qquad (2.14)$$

Dimana:

$w_{ij}^{(k+1)}$ = Bobot pada iterasi selanjutnya (iterasi ke-*k+1*).

v^k = Nilai pembaharuan bobot dengan momentum.

γ = Parameter momentum (yang menentukan seberapa besar momentum akan mempengaruhi pembaharuan bobot)

2.4.5 *Nesterov Accelerated Gradient*

Nesterov accelerated gradient (NAG) merupakan salah satu turunan metode momentum yang memiliki waktu konvergensi lebih cepat daripada *gradient descent* pada masalah konveks [27]. Inti dari NAG adalah proses "lihat kedepan" yang terhadap parameter bobot yang akan menjadi bobot pada iterasi selanjutnya. Jika

pada iterasi selanjutnya gradien melebihi titik minima maka pembaharuan bobot akan menjadi lebih kecil, jika sebaliknya pembaharuan bobot akan menjadi lebih besar [28]. Aturan pembaharuan parameter pada NAG adalah:

$$v^k = \gamma v^{(k-1)} - \mu \left(\frac{\partial C}{\partial (w_{ij} - \gamma v^{(k-1)})} \right)^k \quad (2.15)$$

$$w_{ij}^{(k+1)} = w_{ij}^k - v^k \quad (2.16)$$

$w_{ij}^{(k+1)}$ = Bobot pada iterasi selanjutnya (iterasi ke-*k+1*).

v^k = Nilai pembaharuan bobot dengan momentum.

γ = Parameter momentum (yang menentukan seberapa besar momentum akan mempengaruhi pembaharuan bobot)

Pada (2.15), pencarian nilai pembaharuan bobot dengan momentum berbeda dengan rumus yang ada pada (2.13). Hal tersebut disebabkan karena pada (2.15) proses "lihat kedepan"(*look ahead)* yang menjadi inti dari *Nesterov Accelerated Gradient* sudah diterapkan. Berbeda dengan (2.13) yang menggunakan gradient dari *cost function* terhadap koneksi bobot pada iterasi yang sedang berjalan, pada NAG *cost function* diturunkan terhadap bobot yang sudah dikurangi dengan nilai pembaharuan bobot pada iterasi sebelumnya. Sehingga, jika gradient *cost function* terhadap bobot yang telah dikurangi dengan nilai pembaharuan sebelumnya berlawanan arah(melebihi titik optima) maka nilai pembaharuan bobot tersebut akan mengecil namun jika tidak maka nilai pembaharuan bobot tersebut akan semakin besar.

2.4.6 *Adaptive Moment Estimation*

Adaptive moment estimation (Adam) merupakan algoritma untuk *first-order gradient-based optimization* fungsi objektif stokastik, berdasarkan perkiraan adaptif momen orde rendah. Metode ini mudah diterapkan, efisien secara komputasi, membutuhkan sedikit memori, tidak berubah dari penskalaan diagonal gradien, dan cocok untuk masalah yang besar dalam hal data dan / atau parameter [29]. Aturan pembaharuan parameter pada Adam yaitu sebagai berikut:

$$m^k = \beta_1 m^{(k-1)} + (1 - \beta_1) g_k \quad (2.17)$$

$$v^k = \beta_2 v^{(k-1)} + (1 - \beta_2) g_k^2 \quad (2.18)$$

$$\widehat{m}^k = \frac{m^k}{1 - \prod_{i=1}^{k} \beta_1^k} \quad (2.19)$$

$$\hat{v}^k = \frac{v^k}{1 - \prod_{i=1}^{k} \beta_2^k} \quad (2.20)$$

$$w^{(k+1)} = w^k - \frac{\mu}{\sqrt{\hat{v}^k} + \epsilon} \widehat{m}^k \quad (2.21)$$

Dimana:

m^k = nilai estimasi moment ke-satu.

v^k = nilai estimasi moment ke-dua.

β_1, β_2 = laju peluruhan(*decay rate,* biasanya bernilai 0.9, dan 0.999 secara berurutan).

$\widehat{m}^k$ = nilai estimasi moment ke-satu setelah dilakukan *bias-correction.*

$\hat{v}^k$ = nilai estimasi moment ke-dua setelah dilakukan *bias-correction*.

ϵ = nilai pengkoreksi yang berguna untuk mencegah pembagian pada (2.21) tidak menjadi pembagian dengan 0 (biasanya bernilai 10^{-8}).

g_k = nilai pembaharuan bobot atau bias.

Aturan pada pencarian nilai moment m dan v merupakan inti dari *adaptive moment estimation*. Nilai g_k yang digunakan adalah gradient dari *cost function* terhadap parameter yang akan diperbaharui(bisa jadi bobot atau bias). Setelah kedua moment tersebut diketahui, akan diterapkan *bias correction* terhadap moment m dan v. Hal tersebut dikarenakan pada inisialisasi moment m dan v bernilai 0 dan karena itu moment m dan v bias terhadap nilai 0 terutama pada iterasi awal dan *decay rate* yang kecil. Untuk menangani hal tersebut, penulis yang mencetuskan *adaptive*

moment estimation menambahkan *bias correction*. Setelah kedua nilai moment diketahui, nilai parameter baru dapat diketahui dengan menggunakan (2.21).

2.4.7 *Nesterov accelerated—Adaptive moment estimation*

NAdam merupakan metode yang muncul dari hasil penggabungan metode Adam dan NAG. Seperti yang bisa dilihat pada (2.21), jika $\widehat{m}^k$ dijabarkan akan terlihat seperti:

$$w^{(k+1)} = w^k - \frac{\mu}{\sqrt{\hat{v}^k} + \epsilon}\left(\frac{\beta_1 m^{(k-1)} + (1 - \beta_1) g_k}{1 - \prod_{i=1}^{k} \beta_1^k}\right) \qquad (2.22)$$

Untuk dapat memodifikasi Adam dengan memanfaatkan karakteristik "lihat kedepan" dari NAG, $m^{(k-1)}$ pada (2.22) akan diganti dengan $m^{(k)}$ sehingga menjadi [8]:

$$w^{(k+1)} = w^k - \frac{\mu}{\sqrt{\hat{v}^k} + \epsilon}\left(\frac{\beta_1 m^{(k)} + (1 - \beta_1) g_k}{1 - \prod_{i=1}^{k} \beta_1^k}\right) \qquad (2.23)$$

Aturan yang sama pada pembaharuan bobot di (2.23) juga berlaku pada pembaharuan bias

$$b^{(k+1)} = w^k - \frac{\mu}{\sqrt{\hat{v}^k} + \epsilon}\left(\frac{\beta_1 m^{(k)} + (1 - \beta_1) g_k}{1 - \prod_{i=1}^{k} \beta_1^k}\right) \qquad (2.24)$$

2.5 Pengujian Performa Model

Pengujian performa model dilakukan dengan mencari nilai akruasi, presisi, dan *recall*. Akurasi adalah nilai yang menunjukan seberapa sukses klasifikasi. Presisi merupakan nilai yang menunjukan seberapa sukses klasifikasi kelas positif. Teakhir, *recall* adalah nilai yang menunjukan seberapa lengkap klasifikasi kelas positif. Persamaan untuk menghitung ketiga nilai tersebut dapat dilihat pada (2.25), (2.26), dan (2.27).

$$\text{Akurasi} = \frac{TP + TN}{TP + TN + FP + FN} * 100\% \qquad (2.25)$$

$$\text{Presisi} = \frac{TP}{TP + FP} * 100\% \quad (2.26)$$

$$Recall = \frac{TP}{TP + FN} * 100\% \quad (2.27)$$

Keterangan:

TP (*True Positive*) = Data terklasifikasi positif dan merupakan data positif.
TN (*True Negative*) = Data terklasifikasi negatif dan merupakan data positif.
FP (*False Positive*) = Data terklasifikasi positif dan merupakan data negatif.
FN (*False Negative*) = Data terklasifikasi negatif dan merupakan data positif.

Untuk kemudahan, data hasil prediksi model dapat disajikan menggunakan *confusion matrix.*

Tabel 2.5 *Confusion matrix*

		Kelas Sebenarnya	
		Positif	Negatif
Kelas Prediksi	Positif	TP	FP
	Negatif	FN	TN

Baris pada *confusion matrix* merupakan contoh kelas yang diprediksi, dan kolom pada *confusion matrix* adalah contoh kelas yang sebenarnya [30].

2.6 K-fold Cross Validation

K-fold cross validation merupakan suatu metode untuk mengevaluasi model *machine learning* pada dataset yang jumlahnya terbatas. Tujuan utama digunakan k-fold cross validation adalah untuk memberikan estimasi performa suatu model pada data yang belum terlihat. Metode k-fold cross validation dimulai dengan membuat subset dari dataset sebanyak *k* bagian secara acak. Lalu, proses traning dan testing akan dilakukan sebanyak *k* dengan subset dataset ke-*k* sebagai data uji dan subset dasaset sisanya sebagai data latih [31].