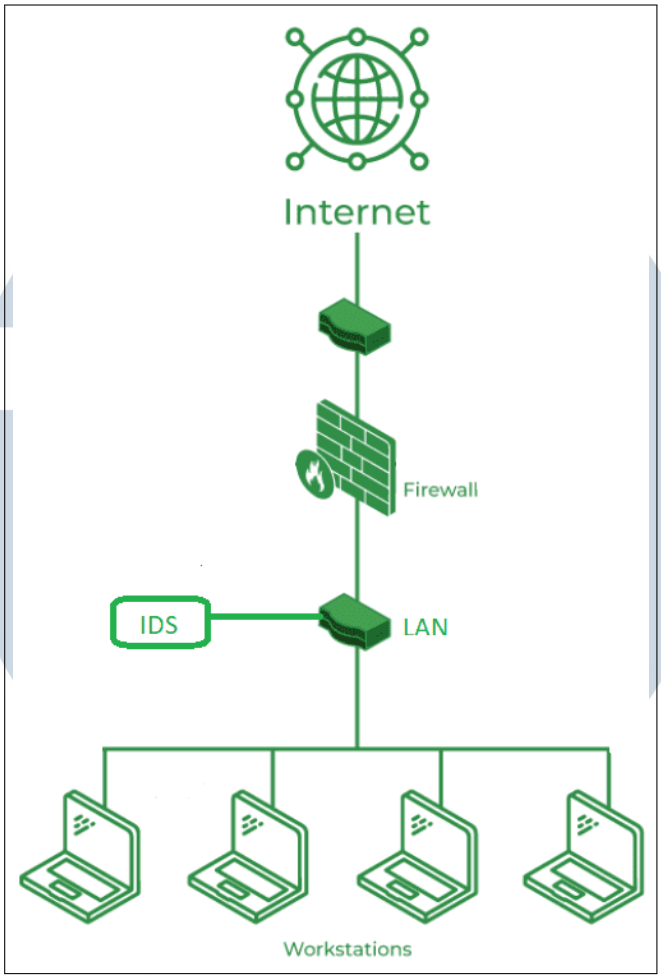
BAB 2
LANDASAN TEORI

2.1 *Ransomware*

Ransomware adalah *software* berbahaya atau *malware* digunakan untuk melakukan *data encryption* pada sistem komputer atau perangkat lainnya lalu pelaku meminta bayaran kepada korban supaya data dikembalikan atau didekripsi. Cara kerja *ransomware* terdiri dari berbagai tahap yaitu pertama, *ransomware* akan masuk kedalam sistem pengguna dari *link* meragukan pada *email*, *website* yang terkena oleh *ransomware*, atau dengan memanfaatkan kelemahan pada sebuah *software* atau sistem operasi. Kemudian, *ransomware* akan melakukan *data encryption* yang menyebabkan data tidak bisa digunakan oleh pemiliknya. lalu, *ransomware* akan menunjukkan pesan bayaran berupa perintah tertuju ke korban untuk melakukan pembayaran dengan bentuk *crypto currency* dengan contoh salah satunya adalah *Bitcoin* sebagai salah satu syarat dalam mendapatkan *decryption key* [13]. *Ransomware* dapat dibagi menjadi beberapa kategori berdasarkan perilaku dan dampaknya, yaitu *crypto ransomware* yang mengenkripsi *file* dan meminta bayaran untuk kunci dekripsi, *locker ransomware* yang mengunci pengguna dari sistem dan umumnya terjadi pada *mobile*, *scareware* yaitu menakuti korban agar membayar tanpa melakukan enkripsi apapun, *doxware* yaitu mengancam untuk membocorkan informasi pribadi korban kecuali uang tebusan dibayar dan yang terakhir *ransomware worms* dimana *ransomware* yang dapat menyebar sendiri diseluruh jaringan tanpa interaksi pengguna [14].

2.2 *Intrusion Detection System*

IDS merupakan salah satu cara dalam mendeteksi serangan yang terjadi pada sebuah komputer atau sebuah server pada suatu jaringan komputer. IDS umumnya ditempatkan di luar infrastruktur jaringan sehingga tidak berada dalam jalur komunikasi antara pengirim dan penerima informasi. Gambar 2.1 menunjukkan *Host-Based* IDS yang ditempatkan di luar infrastruktur jaringan.



Gambar 2.1. Penempatan *Host-Based Intrusion Detection System* [1]

IDS memiliki prinsip yaitu memonitor lalu lintas jaringan lalu mendeteksi serangan atau intrusi dari luar sistem yang berasal dari internet ke dalam suatu sistem internal [15]. IDS sendiri memiliki berbagai macam jenis berdasarkan pendekatannya seperti *network-based* (NIDS) dimana NIDS memantau semua lalu lintas jaringan yang mengalir ke perangkat-perangkat dan membuat keputusan berdasarkan konten paket dan metadata. Selanjutnya adalah *host-based* (HIDS) yaitu memantau infrastruktur komputer dimana HIDS tersebut dipasang dan melakukan analisis lalu lintas, mencatat aktivitas berbahaya dan memberi tahu otoritas yang dipilih. Lalu terdapat IDS juga *protocol-based* (PIDS) memantau dan menganalisis protokol antara perangkat dan server, *application protocol-based* (APIDS) yang dapat melacak protokol yang spesifik untuk aplikasi tertentu dan yang terakhir adalah *Hybrid* IDS dimana sistem menggabungkan dua atau lebih pendekatan deteksi intrusi. IDS juga dapat dibagi berdasarkan cara mendeteksinya yaitu *signature-based* atau mendeteksi dengan cara memonitor lalu lintas jaringan

yang masuk mencari pola dan urutan serangan yang cocok dengan signaturenya. Setelah itu, terdapat *anomaly-based* yang digunakan untuk mendeteksi serangan yang tidak diketahui dan umumnya menggunakan *machine learning*. Akan tetapi, *anomaly-based* dapat menyebabkan *false positives* yang lebih tinggi dibandingkan *signature-based*.

2.3 *Ensemble Learning*

Ensemble Learning yaitu *machine learning* yang menggabungkan model-model yang dilatih untuk menyelesaikan permasalahan serupa sehingga mendapatkan hasil yang lebih optimal. *Ensemble Learning* dapat dibagi menjadi 3 yaitu *Bagging*, *Boosting*, dan *Stacking* [16]. *Bagging* adalah memilih beberapa model untuk dilatih pada *subset* yang berbeda dari data pelatihan lalu setiap *subset* diambil sampel secara berulang dan prediksi dari masing-masing model digabungkan dengan cara rata-rata untuk regresi atau voting untuk klasifikasi. Selanjutnya adalah *boosting, boosting* adalah melatih beberapa model secara berurutan dan berusaha untuk memperbaiki kesalahan yang dibuat oleh model sebelumnya. Kemudian yang terakhir adalah *stacking, stacking* menggunakan beberapa model yang melakukan pembelajaran secara pararel pada setiap modelnya dan digabungkan menggunakan salah satu algoritma *meta learning* agar mendapatkan hasil kombinasi model [16].

2.4 *Decision Tree Learning*

Decision Tree Learning merupakan metode dalam pembelajaran mesin dipakai untuk membuat model prediksi dalam bentuk *decision tree*. Pohon ini digunakan untuk mengklasifikasikan data atau memprediksi nilai berdasarkan keputusan yang diambil. Dengan menggunakan node yang saling berhubungan seperti bentuk pohon, Decision Tree akan mencari solusi dari sebuah permasalah. Setiap pohon memiliki cabang yang mewakili semua *attribute* yang diperlukan untuk melanjutkan ke tahap berikutnya yaitu pada cabang sampai selesai di daun atau ketika tidak ada lagi cabang[17]. Algoritma C4.5 adalah merupakan contoh algoritma yang pakai untuk membangun *decision tree*. Langkah kerja algoritma C4.5 untuk memilih fitur yaitu [18] :

1. **Menghitung *Entropy* Setiap Atribut:**

$$Entropy(S) = \sum_{i=1}^{n} -p_i \log_2(p_i) \tag{2.1}$$

Dimana:

- S = Himpunan kasus,
- n = total partisi dalam S,
- p_i = Probabilitas yang dihasilkan oleh jumlah kelas yang dibagi dengan total kasus.

2. **Menghitung *Information Gain* dari Setiap Atribut:**

$$Gain(S,A) = Entropy(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} \cdot Entropy(S_i) \tag{2.2}$$

Dimana:

- S = himpunan kasus,
- A = *Attribute*,
- n = total partisi dalam S,
- $|S_i|$ = total kasus dalam partisi ke-i,
- $|S|$ = total kasus total dalam S.

3. **Menghitung *Split Gain* dari Setiap Atribut:**

$$SplitInfo(S,A) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \log_2 \left(\frac{|S_i|}{|S|} \right) \tag{2.3}$$

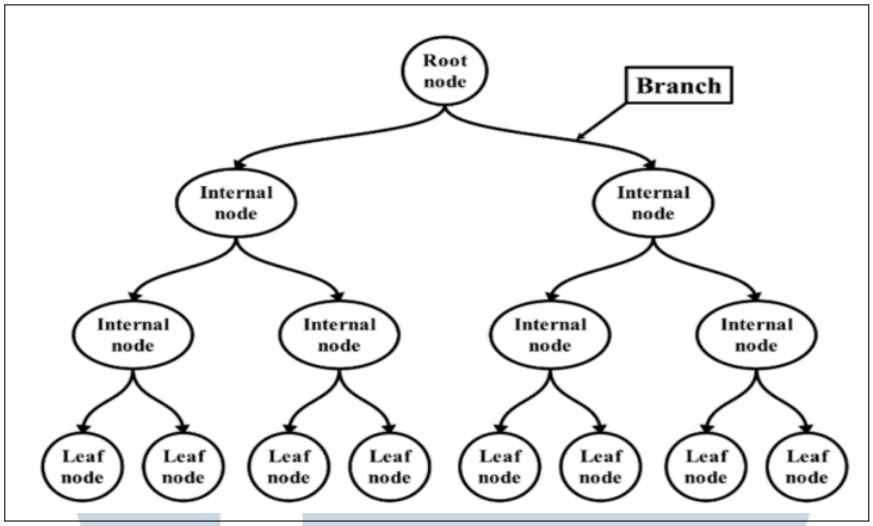
Dimana:

- S = Ruang sampel data dipakai sebagai training,
- A = *Attribute*,
- $|S_i|$ = total sampel *attribute* i.

4. **Menghitung *Gain Ratio* dari Setiap Atribut:**

$$GainRatio(S,A) = \frac{Information\ Gain(S,A)}{SplitInfo(S,A)} \tag{2.4}$$

Gambar 2.2 merepresentasikan sebuah *decision tree*



Gambar 2.2. Representasi *Decision Tree* [2]

2.5 *Random Forest*

Random Forest (RF) merupakan jenis algoritma klasifikasi meggunakan *decision tree* yang bergantung dengan *value* vektor acak sampel dan dibagikan dengan sama kepada seluruh pohon. Metode tersebut berupa salah satu metode yang bisa dipakai untuk mengatasi variabel dengan input yang sangat besar tanpa menyebabkan *overfitting*, serta membantu menghilangkan korelasi antara *decision tree* seperti sifat khas metode *ensemble* [19]. Algoritma RF umumnya menggunakan teknik *Bagging (Bootstrap Aggregating)*. Tahapan dalam algoritma RF yaitu diawali dengan pembentukan pohon keputusan. Algoritma RF akan membuat banyak pohon keputusan yang akan dipakai untuk membuat keputusan dalam mengklasifikasi data. Selanjutnya adalah melakukan proses *bootstrapping* dan melakukan pemilihan *subset* dari seluruh fitur yang dimiliki secara acak. Tahap yang terakhir adalah pengambilan hasil keputusan dari setiap pohon yang akan digabungkan berdasarkan mayoritas untuk permasalahan klasifikasi dan rata-rata untuk permasalahan regresi. Berikut adalah penjelasan algoritma RF secara detail dengan beberapa tahapan [20]:

1. **Pada Tahap Pembagian Data :**

$$D_{\text{bootstrap}} = \{D_i\}_{i=1}^{m} \tag{2.5}$$

$D_{\text{bootstrap}}$ merupakan hasil *bootstrap* sample yang didapatkan serta memiliki

hasil yang sama dengan jumlah data asli. D_i merupakan *bootstrap sample* yang diambil dari dataset asli. m merupakan jumlah model atau *estimator* yang ingin dibangun

2. **Pada Tahap Pembangunan Pohon Menggunakan *Bootstrap Sample*:**

$$T_j = T(D_j, X_j, \theta_j) \tag{2.6}$$

T_j merupakan pohon keputusan yang didapatkan dari *bootstrap sample* D_j, variabel prediktor X_j, dan parameter acak θ_j.

3. **Pada Tahap Penggabungan Pohon:**

$$F(X) = \frac{1}{m} \sum_{j=1}^{m} T_j(X) \tag{2.7}$$

Dilakukan penggabungan prediksi dari masing-masing pohon agar menghasilkan fungsi prediksi akhir. $F(X)$ adalah fungsi prediksi yang didapatkan dari penggabungan seluruh pohon.

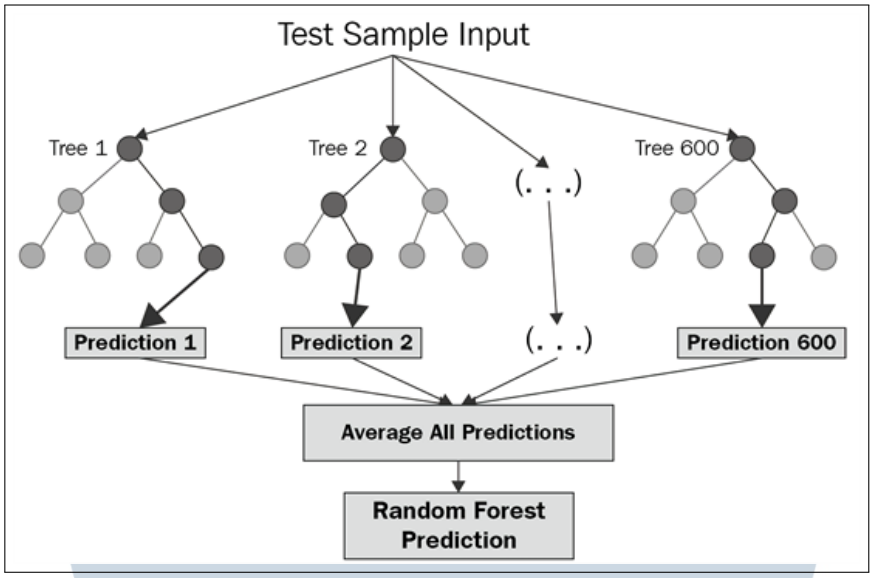
4. **Proses Tahap Prediksi:**

$$y = \arg\max_{k} F(X) \tag{2.8}$$

Prediksi menggunakan fungsi prediksi yang didapatkan. Prediksi dilakukan dengan mendapatkan kelas k yang memiliki nilai terbesar dalam fungsi prediksi. y merupakan kelas yang diprediksi berdasarkan voting mayoritas dari semua pohon.

Gambar 2.3 merupakan representasi dari algoritma RF

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 2.3. Representasi Algoritma *Random Forest* [3]

2.6 *Particle Swarm Optimization*

PSO merupakan sebuah algoritma optimasi berdasarkan perilaku sebuah kumpulan ikan atau kumpulan burung. PSO umumnya mempunyai tiga komponen utama yaitu *particle*, *cognitive component*, *social component*, dan kecepatan *particle*. Setiap *particle* mewakilkan sebuah solusi penyelesaian [21]. PSO bekerja dengan menginisialisasi sekumpulan *particle* yang tersebar secara acak dalam ruang pencarian dan memiliki dua atribut yaitu posisi dan kecepatan. *particle* tersebut bergerak dalam ruang pencarian untuk mencari solusi yang dapat memperbaruhi posisi dan kecepatan menurut pengalaman terbaik sebuah *particle*(pBest) atau pengalaman terbaik seluruh *particle* (gBest). Berikut adalah penjelasan tahapan mengenai algoritma particle swarm optimization secara detail [22]:

1. **Inisialisasi**

   - Pada iterasi ke-0, ketika nilai kecepatan awal semua *particle* adalah 0, melakukan inisialisasi kecepatan awal dan posisi awal *particle*. Posisi awal *particle* diambil dari persamaan berikut :

$$x = x_{\min} + \text{rand}[0,1] \times (x_{\max} - x_{\min}) \tag{2.9}$$

     Dimana:

     – x posisi *particle*

- $x_{\min}$ = posisi *minimum particle*
- $x_{\max}$ = posisi *maximum particle*
- rand$[0,1]$ = nilai *random* antara 0 dan 1 yang berdistribusi *uniform* dalam interval [0, 1]

- Inisialisasi *pBest* dan *gBest* pada iterasi ke-0. Nilai *pBest* disamakan oleh posisi awal *particle* akan tetapi *gBest* dipilih dari salah satu *pBest* dengan *fitness* tertinggi.

2. **Memperbarui Kecepatan Partikel** dapat memperbarui kecepatan *particle* dengan menggunakan rumus:

$$v_{i,j}^{t+1} = w \cdot v_{i,j}^{t} + c_1 \cdot r_1 \cdot (pBest_{i,j}^{t} - x_{i,j}^{t}) + c_2 \cdot r_2 \cdot (gBest_{g,j}^{t} - x_{i,j}^{t}) \quad (2.10)$$

Dimana:

- $v_{i,j}^{t+1}$ = kecepatan *particle i* di *dimension j* pada *iteration* $t+1$
- w = bobot inersia
- $v_{i,j}^{t}$ = kecepatan *particle i* di *dimension j* pada *iteration t*
- c_1, c_2 = konstanta kecepatan
- r_1, r_2 = nilai acak antara 0 dan 1
- $pBest_{i,j}^{t}$ = *personal best* dari *particle i* di *dimension j* pada *iteration t*
- $gBest_{g,j}^{t}$ = *global best* dari *particle g* di *dimension j* pada *iteration t*
- $x_{i,j}^{t}$ = posisi *particle i* di *dimension j* pada *iteration t*

3. **Memperbarui Posisi dan Hitung *Fitness*** dapat memperbarui posisi partikel dengan rumus:

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1} \quad (2.11)$$

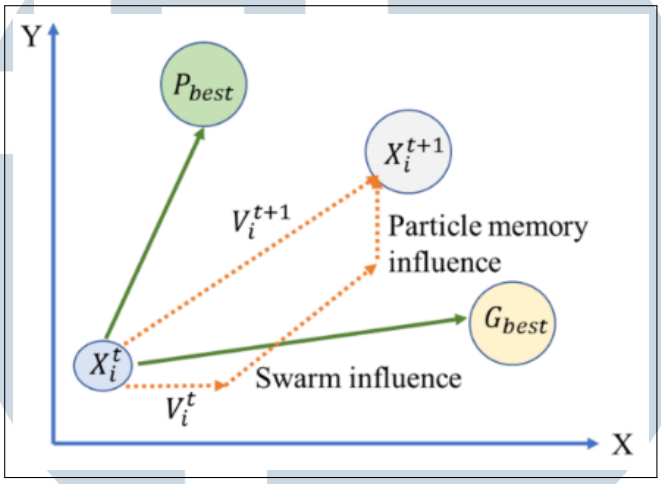
Dimana:

- $x_{i,j}^{t+1}$ = posisi *particle i* di *dimension j* pada *iteration* $t+1$
- $x_{i,j}^{t}$ = posisi *particle i* di *dimension j* pada *iteration t*
- $v_{i,j}^{t+1}$ = kecepatan *particle i* di *dimension j* pada *iteration* $t+1$

Setelah posisi diperbaharui, dapat menghitung nilai *fitness* berdasarkan akurasi model optimal.

4. **Memperbarui** *pBest* **dan** *gBest***:** Melakukan perbandingan *personal best* dari iterasi sebelumnya menggunakan hasil dari pembaruan posisi. *pBest* terbaru dengan nilai *fitness* terbaik dapat menjadi *gBest* terbaru.

Gambar 2.4 adalah representasi dari model PSO



Gambar 2.4. Representasi Model PSO [4]

2.7 *Principal Component Anaylsis*

PCA merupakan teknik yang dipakai dalam melakukan *dimensionality reduction*, yang bertujuan untuk menyederhanakan dataset dengan mengurangi jumlah variabel yang dianalisis tanpa kehilangan informasi penting. PCA membuat dimensi-dimensi baru lalu akan diurutkan menurut varian datanya. PCA mendapatkan hasil berupa Komponen Utama yang dihasilkan dari dekomposisi *eigen value* dan *eigen vector* dari *covariance matrix* [23]. Tahapan yang dimiliki oleh algortima PCA adalah [23] :

1. **Menghitung rata-rata ($\bar{X}$) data pada tiap dimensi:**

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{2.12}$$

Dengan:

- n = total data sampel
- X_i = data sampel

2. **Menghitung *covariance matrix* (C_X):**

$$C_X = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})^T \tag{2.13}$$

Dengan:

- n = total data sampel
- X_i = data sampel
- $\bar{X}$ = rata-rata

3. **Menghitung *eigenvector* (v_m) dan *eigenvalue* (λ_m) dari *covariance matrix*:**

$$C_X v_m = \lambda_m v_m \tag{2.14}$$

4. **mengurutkan *eigenvalue* secara menurun. Komponen Utama adalah deretan *eigenvector* sesuai dengan urutan *eigenvalue* yang didapatkan**

5. **Menghasilkan dataset baru.**

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA