# Laporan Tugas Besar 2 IF3170 Intelegensi Artifisial

## Semester I Tahun Akademik 2024/2025

## Implementasi Algoritma Pembelajaran Mesin


Disusun Oleh

| | |
|---|---|
| Edbert Eddyson Gunawan | 13522039 |
| Vanson Kurnialim | 13522049 |
| Muhammad Syarafi Akmal | 13522076 |
| Fabian Radenta Bangun | 13522105 |

**Sekolah Teknik Elektro dan Informatika**
**Institut Teknologi Bandung**


**2024**

# Pembahasan

Pada tugas ini, Implementasi algoritma dilakukan dalam dua pendekatan, yaitu implementasi manual menggunakan bahasa pemrograman Go dan implementasi menggunakan *library* scikit-learn di Python.

## Implementasi Algoritma K-Nearest Neighbor (KNN)

Pada implementasi algoritma KNN dalam Go, data kategorikal ditangani dengan menghitung jarak menggunakan Hamming Distance, yaitu jarak antara dua titik berdasarkan jumlah atribut yang berbeda nilainya. Proses ini dilakukan melalui fungsi *distance* yang membandingkan setiap elemen antara data uji dan data latih. Selanjutnya, fungsi *predict* digunakan untuk menentukan kelas prediksi berdasarkan mayoritas kelas dari k tetangga terdekat yang dipilih berdasarkan jarak terpendek. *Sorting* dilakukan menggunakan slice untuk mengurutkan tetangga berdasarkan jarak, sementara prediksi akhir ditentukan dengan menghitung modus dari label tetangga tersebut. Implementasi ini dilengkapi dengan paralelisasi menggunakan *goroutine* dan *mutex lock* untuk mempercepat proses evaluasi akurasi melalui fungsi *score*.

Dalam implementasi KNN menggunakan scikit-learn di Python, algoritma KNeighborsClassifier digunakan untuk melatih dan mengevaluasi model. Sebelum pemodelan dilakukan, data kategorikal diproses menggunakan *One-Hot Encoding* untuk mengubah nilai kategorikal menjadi bentuk numerik yang dapat dipahami oleh algoritma KNN. Proses ini dilakukan secara otomatis melalui *library* pandas dan *sklearn.preprocessing*. Dataset kemudian dibagi menjadi *training* dan *test set* dengan perbandingan 80:20 menggunakan fungsi *train_test_split* untuk memisahkan data latih dan data uji. Parameter k, yang menentukan jumlah tetangga terdekat, diberikan nilai awal (misalnya k=3). Model dilatih pada data latih dan dievaluasi menggunakan metrik *accuracy score* untuk menilai akurasi prediksi pada data uji.

## Implementasi Algoritma Naive-Bayes

Pada implementasi Naive Bayes dalam bahasa Go, digunakan perhitungan terhadap probabilitas untuk setiap atribut dan kelas target. Implementasi ini dilakukan dengan tahapan sebagai berikut:

- Inisialisasi Model

  Frekuensi kemunculan setiap atribut dalam setiap kelas dihitung, dan probabilitas awal (*prior probability*) untuk setiap kelas ditetapkan. Ini dilakukan dalam fungsi *fit*, yang membangun struktur data model berbasis map tiga lapis untuk menyimpan frekuensi dan probabilitas setiap atribut.
- Perhitungan Probabilitas

  Untuk memprediksi kelas dari sebuah data uji, fungsi *predict* digunakan untuk mengalikan probabilitas atribut yang muncul dalam setiap kelas.
- Paralelisasi Proses Evaluasi

  Fungsi *score* menghitung akurasi model dengan memprediksi kelas untuk setiap data uji dan membandingkan hasil prediksi dengan label sebenarnya. Proses ini dipercepat menggunakan *goroutine* dan *mutex* untuk menghitung jumlah prediksi benar (true positives) secara paralel.

Sementara pada implementasinya dengan Python, digunakan *library* scikit-learn di Python. Tahapan yang dilakukan pada implementasi ini adalah:

- Preprocessing Data

  Data kategorikal diproses menggunakan *One-Hot Encoding* untuk mengubah atribut kategorikal menjadi representasi numerik, sehingga kompatibel dengan model *GaussianNB* dari *scikit-learn*.
- Pemodelan dan Pelatihan

  Model Gaussian Naive Bayes dilatih menggunakan data latih dengan metode *fit*.
- Evaluasi Model

  Model dievaluasi dengan membandingkan prediksi pada data uji dengan label sebenarnya menggunakan metrik *accuracy score*. Hal ini memberikan gambaran performa model dalam memprediksi kelas target.

## Implementasi Algoritma ID3

Pada implementasi menggunakan bahasa Go, *Decision Tree* dibangun dengan menghitung *entropy* dan *information gain* sebagai dasar pemilihan atribut terbaik untuk memisahkan data. Proses dimulai dengan perhitungan *entropy* kelas melalui fungsi *entropyClass* untuk mengukur ketidakpastian data, kemudian dilanjutkan dengan perhitungan *entropy* atribut pada setiap nilai unik atribut menggunakan fungsi *entropyAttribute*. Selisih antara *entropy*

kelas dan *entropy* atribut digunakan untuk menghitung *information gain*, yang kemudian dimanfaatkan oleh fungsi *infoGain* untuk menentukan atribut dengan informasi pemisahan terbaik. *Decision tree* dibangun secara rekursif menggunakan fungsi *recursiveBuildID3*, di mana atribut dipilih satu per satu, dan dataset dibagi berdasarkan nilai-nilai unik dari atribut tersebut hingga kondisi berhenti tercapai, seperti ketika semua data memiliki kelas yang sama atau atribut telah habis. Untuk prediksi, fungsi *predict* digunakan untuk menelusuri pohon keputusan berdasarkan input fitur hingga mencapai kelas hasil. Evaluasi model dilakukan dengan menghitung akurasi melalui fungsi *score* yang memprediksi setiap data uji secara paralel menggunakan *goroutine* untuk mempercepat pemrosesan.

Sementara itu, implementasi ID3 di Python menggunakan *library scikit-learn* dilakukan dalam fungsi *ID3s*. Proses ini diawali dengan *preprocessing data*, atribut kategorikal diubah menjadi format numerik menggunakan *One-Hot Encoding* agar kompatibel dengan model *decision tree*. Dataset kemudian dibagi menjadi data latih dan data uji dengan rasio 80:20 menggunakan fungsi *train_test_split*. Model *decision tree* dilatih menggunakan *DecisionTreeClassifier* dengan parameter criterion='entropy', yang memastikan bahwa atribut dipilih berdasarkan perhitungan *information gain*. Evaluasi model dilakukan dengan mengukur akurasi prediksi menggunakan metrik *accuracy score* yang membandingkan hasil prediksi dengan label sebenarnya pada data uji.

## Data Cleaning dan Preprocessing

Pada tahap *Data Cleaning*, dilakukan langkah sebagai berikut:

1. *Handling Missing Data*
    a. Mengisi nilai NaN pada kolom *is_sm_ips_ports*, *is_ftp_login*, dan *ct_flw_http_mthd* dengan 0.
    b. Mengisi nilai NaN pada kolom *tcprtt* dengan penjumlahan *synack* dan *ackdat*
    c. Mengisi nilai NaN pada kolom *smean* dengan pembagian *sbytes* oleh *spkts*
    d. Mengisi nilai NaN pada kolom *dmean* dengan pembagian *dbytes* oleh *dpkts*
2. *Feature Engineering*

    Untuk meningkatkan kualitas data, ditambahkan fitur-fitur baru yang dapat meningkatkan kinerja model prediksi. Berikut adalah penjelasan mengenai proses yang telah dilakukan:

a. Perhitungan Rasio

Beberapa rasio antara fitur sumber dan tujuan dihitung untuk memahami hubungan antara ukuran dan frekuensi data yang dikirim dan diterima. Rasio yang dihitung meliputi:

i. byte_ratio: Rasio antara jumlah byte yang dikirim dan diterima.
ii. pkt_ratio: Rasio antara jumlah paket yang dikirim dan diterima.
iii. load_ratio: Rasio antara jumlah load yang dikirim dan diterima.
iv. jit_ratio: Rasio antara jitter yang dikirim dan diterima.
v. inter_pkt_ratio: Rasio antara interval paket yang dikirim dan diterima.
vi. tcp_setup_ratio: Rasio antara waktu setup TCP (tcprtt) dan jumlah sinyal yang dipertukarkan dalam TCP setup.

b. Fitur Agregat

Beberapa fitur agregat dihitung dengan menggabungkan data sumber dan tujuan untuk memperoleh gambaran lebih komprehensif tentang aktivitas jaringan secara keseluruhan. Fitur agregat yang dibuat meliputi:

i. total_bytes: Jumlah total byte yang dikirim dan diterima.
ii. total_pkts: Jumlah total paket yang dikirim dan diterima.
iii. total_load: Jumlah total load yang dikirim dan diterima.
iv. total_jitter: Jumlah total jitter yang dikirim dan diterima.
v. total_inter_pkt: Jumlah total interval paket yang dikirim dan diterima.
vi. total_tcp_setup: Jumlah total waktu setup TCP yang melibatkan SYN, ACK, dan datagram.

c. Fitur Interaksi

Interaksi antar fitur dihitung untuk mengekstrak hubungan yang lebih mendalam antara berbagai aspek komunikasi. Fitur interaksi yang dihitung meliputi:

i. byte_pkt_interaction_src dan byte_pkt_interaction_dst: Interaksi antara jumlah byte dan jumlah paket yang dikirim/diperoleh oleh sumber dan tujuan.
ii. load_jit_interaction_src dan load_jit_interaction_dst: Interaksi antara load dan jitter yang dikirim/diperoleh oleh sumber dan tujuan.
iii. pkt_jit_interaction_src dan pkt_jit_interaction_dst: Interaksi antara jumlah paket dan jitter yang dikirim/diperoleh oleh sumber dan tujuan.

d. Fitur Statistik

Dua fitur statistik ditambahkan untuk menangkap pola dalam komunikasi:

i. mean_pkt_size: Rata-rata ukuran paket yang dikirim dan diterima.

ii. tcp_seq_diff: Selisih antara sequence number TCP yang dikirim dan diterima.

Setelah fitur-fitur baru ini dihasilkan, dilakukan analisis korelasi antar fitur untuk mendeteksi fitur-fitur yang memiliki korelasi tinggi (>= 0.75). Fitur-fitur dengan korelasi tinggi dianggap redundant, sehingga satu fitur dari setiap pasangan fitur yang berkorelasi tinggi dipilih untuk dihapus, dengan tujuan untuk mengurangi multikolinearitas dan meminimalkan overfitting pada model. Fitur-fitur yang dipilih untuk dihapus dimasukkan ke dalam daftar features_to_drop, dan akhirnya, kolom-kolom yang tidak relevan dihapus dari dataframe.

Preprocessing yang dilakukan mencakupi beberapa hal atau metode yang dilakukan yaitu :

1. Feature Scaling

   Digunakan metode Z-score yang merupakan teknik untuk mengubah skala fitur dalam dataset agar memiliki distribusi dengan rata-rata 0 dan standar deviasi 1. Proses ini dilakukan dengan mengurangi setiap nilai fitur dengan nilai rata-ratanya, kemudian membaginya dengan standar deviasi dari fitur tersebut. Dengan kata lain, Z-score menghitung seberapa jauh setiap nilai fitur dari rata-rata dalam satuan standar deviasi. Hal ini penting karena beberapa algoritma machine learning, seperti K-Nearest Neighbors (KNN) dan Support Vector Machine (SVM), sensitif terhadap skala fitur. Fitur dengan skala yang lebih besar dapat mendominasi model, sehingga mempengaruhi hasil prediksi. Standard Scaler mengatasi masalah ini dengan menyamakan skala semua fitur, yang memungkinkan algoritma untuk memberikan bobot yang lebih adil pada setiap fitur selama pelatihan model.

2. Feature Encoding

   Digunakan metode LabelEncoder atau metode untuk mengkonversi setiap kategori dalam fitur kategorik menjadi angka unik. Setiap kategori diberi label numerik secara berurutan. Misalnya, jika sebuah fitur memiliki tiga kategori seperti "Merah", "Hijau", dan "Biru", maka LabelEncoder akan mengubahnya menjadi angka seperti 0, 1, dan 2.

3. Imbalanced Dataset

Digunakan SMOTE (Synthetic Minority Over-sampling Technique) adalah sebuah teknik untuk mengatasi masalah class imbalance pada dataset yang digunakan dalam pembelajaran mesin. Masalah class imbalance terjadi ketika satu kelas (label) dalam dataset memiliki jumlah contoh yang jauh lebih sedikit dibandingkan kelas lainnya, yang dapat menyebabkan model machine learning cenderung bias terhadap kelas mayoritas. SMOTE adalah salah satu teknik yang populer untuk menangani masalah ini dengan menghasilkan contoh data sintetis (buatan) untuk kelas minoritas.

4. Dimensionality Reduction

   Digunakan Principal Component Analysis (PCA). Teknik yang digunakan untuk mengurangi jumlah fitur dalam dataset sambil mempertahankan sebanyak mungkin informasi yang ada. PCA sering digunakan dalam analisis data untuk meningkatkan efisiensi komputasi dan mengatasi masalah curse of dimensionality, yang terjadi ketika jumlah fitur dalam dataset sangat besar. Walaupun kehilangan interpretability, PCA dapat mengurangi dimensi namun tetap mempertahankan data relevan sebanyak mungkin.

Data Normalization tidak diperlukan lagi karena sudah adanya feature scaling dan feature encoding. Kedua metode tersebut dirasa sudah cukup untuk menormalisasi dataset.

## Perbandingan

| Algoritma | Pendekatan | Akurasi (%) | Waktu Eksekusi (s) |
|---|---|---|---|
| KNN | Go | 77 | 251 |
| | Scikit-learn | | |
| Naive Bayes | Go | 76 | 0.2 |
| | Scikit-learn | | |
| ID3 | Go | 79 | 9.08 |
| | Scikit-learn | | |

# Pembagian Tugas

| **Task** | **NIM** |
|---|---|
| EDA | 13522039 |
| Split Training Set and Validation Set | 13522039, 13522049, 13522076, 13522105 |
| Data Cleaning and Preprocessing | 13522039, 13522105, 13522049 |
| Compile Preprocessing Pipeline | 13522049 |
| Modeling and Validation | 13522076 |
| Error Analysis | |
| Dokumen Laporan | 13522039, 13522049, 13522076, 13522105 |

# Referensi


Institut Teknologi Bandung. (n.d.). *Data Understanding* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/storages/files/1730818009007_IF3170-Data-Understanding.pdf

Institut Teknologi Bandung. (n.d.). *Data Preparation* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/297373-Data-Preparation/1730818790714_IF-3170-Data-Preparation.pdf

Institut Teknologi Bandung. (n.d.). *k-Nearest Neighbor* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250331293_IF3170_Materi09_Seg01_AI-kNN.pdf

Institut Teknologi Bandung. (n.d.). *Naive Bayes* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250380758_IF3170_Materi09_Seg02_AI-NaiveBayes.pdf

Institut Teknologi Bandung. (n.d.). *Decision Tree Learning* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/298936-Modeling-Decision-Tree-Learning-DTL/120485-Modul-Introduction-to-DTL/1731401412073_IF3170_SupervisedLearning_DTL.pdf