LAPORAN TUGAS BESAR 2

IF3170 INTELIGENSI ARTIFISIAL

Implementasi Algoritma Pembelajaran Mesin KNN, Naive-Bayes, dan ID3

Disusun oleh:

Agil Fadillah Sabri	13522006
Raden Rafly Hanggaraksa Budiarto	13522014
Bastian H Suryapratama	13522034
Moh Fairuz Alauddin Yahya	13522057

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

BAB I

IMPLEMENTASI ALGORITMA

1.1 *K-Nearest Neighbors* (KNN)

K-Nearest Neighbors (KNN) adalah salah satu algoritma pembelajaran mesin yang sederhana, tetapi cukup efektif untuk melakukan klasifikasi. KNN bekerja dengan prinsip bahwa data yang mirip cenderung berjarak dekat satu sama lain dalam ruang fitur.

Algoritma KNN diimplementasikan dengan langkah-langkah berikut:

1. Tentukan Parameter *k*

   Pilih jumlah tetangga terdekat (*k*) yang akan digunakan untuk menentukan hasil klasifikasi atau prediksi.
2. Hitung Jarak

   Untuk setiap data baru yang ingin diprediksi, hitung jarak antara data tersebut dengan semua data pada *training set* menggunakan metrik jarak seperti *Euclidean Distance*, *Manhattan Distance*, atau metrik jarak lainnya.
3. Menentukan k Tetangga Terdekat

   Untuk menentukan *k* tetangga terdekat, pilih *k* data yang memiliki jarak terdekat/minimum
4. *Voting* untuk klasifikasi

   Lakukan *voting* pada target/label dari *k* tetangga terdekat. Label yang paling sering muncul menjadi prediksi untuk data baru.

1.2 *Gaussian Naive-Bayes*

Pada Implementasi *Gaussian Naive-Bayes* menggunakan asumsi bahwa fitur-fitur bersifat kontinu dan memiliki distribusi normal (Gaussian), dengan pendekatan yang berbeda dengan Naive Bayes klasik yang menggunakan frekuensi. Pada Gaussian Naive Bayes menggunakan formula distribusi normal untuk menghitung probabilitas *likelihood* $P(x|y)$ dengan rumus:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma^2 y}} * exp(-\frac{(x_i - \mu y)^2}{(2\sigma^2 y)})$$

dimana μy adalah mean dan $\sigma^2 y$ adalah variance dari fitur x_i untuk kelas y.

Implementasi dilakukan melalui langkah-langkah berikut:

1. Menghitung parameter-parameter distribusi untuk setiap fitur pada setiap kelas.
   a. *Prior probability* $P(y)$ dihitung berdasarkan frekuensi kemunculan setiap kelas dalam data training
   b. *Mean* (μy) dihitung sebagai rata-rata nilai fitur untuk sampel pada kelas tersebut,
   c. Variance ($\sigma^2 y$) dihitung sebagai rata-rata kuadrat selisih nilai fitur dengan mean kelasnya.

d. *Apply variance smoothing factor* (var_smoothing) untuk menghindari masalah pembagian dengan nol dan menghindari masalah *overfitting*, karena jika variansi terlalu kecil bisa menyebabkan model terlalu yakin dengan prediksinya.

2. Prediksi menggunakan formula Naive bayes:
   a. Formula dasar:
   $$P(y|X) \propto P(y) \ * \ \prod P(x_i|y)$$
   b. Transformasi ke domain logaritmik untuk menghindari *underflow* yang terjadi ketika hasil perkalian probabilitas sangat kecil (mendekati nol) melebihi batas presisi *floating-point* komputer
   $$log\ P(y|X) \ = \ log\ P(y) \ + \ \Sigma\ log\ P(x_i|y)$$
   c. Perrhitungan Log *likelihood* untuk mengukur seberapa mungkin sebuah observasi berasal dari sebuah nilai distribusi, menggunakan formula:
   $$log\ P(x\_i|y) \ = \ -\ 0.5\ log(2\pi\sigma^2 y) \ - \ (x_i \ - \ \mu y)^2/(2\sigma^2 y)$$
   d. Normalisasi Probabilitas untuk mencegah *overflow*/*underflow* saat eksponensiasi dengan menggunakan teknik *log-sum-exp* untuk stabilitas numerik:
   $$P(y|X) \ = \ \frac{exp(log\ P(y|X) - max(log\ P(y|X)))}{\Sigma\ exp(log\ P(y|X) - max(log\ P(y|X)))}$$
3. Prediksi akhir
   a. Pilih kelas dengan probabilitas posterior tertinggi:
   $$\hat{y} \ = \ argmax\_y\ P(y|X)$$

1.3 *Iterative Dichotomiser 3* (ID3)

Iterative Dichotomiser 3 (ID3) adalah algoritma pembelajaran mesin untuk membangun *decision tree* yang digunakan dalam klasifikasi. Algoritma ini membagi *dataset* berdasarkan atribut yang memberikan informasi paling tinggi, menggunakan konsep *Information Gain*.

Algoritma ID3 diimplementasikan dengan langkah-langkah sebagai berikut:

1. Inisialisasi Dataset

   Siapkan terlebih dahulu dataset yang terdiri dari kumpulan atribut dan label kelas.

2. Hitung Entropi Data Awal

   Entropi digunakan untuk mengukur ketidakpastian dalam data. Semakin rendah entropi, semakin murni data dalam hal klasifikasi. Berikut adalah persamaan yang digunakan untuk menghitung nilai entropi sebuah dataset.

   $$Entropy(S) \equiv \sum_{i=1}^{c} - p_i\ log_2\ p_i$$

   Dengan p_i adalah probabilitas masing-masing kelas di dalam dataset.

3. Hitung *Information Gain* setiap Atribut

   Untuk setiap atribut, hitung *information gain* berdasarkan pengurangan entropi sebelum dan sesudah pemisahan data oleh atribut tersebut. Berikut adalah persamaan yang digunakan untuk menghitung nilai *information gain* suatu atribut.

$$Gain(S,\ A)\ \equiv\ Entropy(S)\ -\ \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

4. Pemilihan Atribut

   Pilih atribut dengan *information gain* tertinggi sebagai node pembagi (*decision node*) pada *decision tree* tahap ini

5. Split Dataset

   Pecah dataset menjadi subset berdasarkan nilai-nilai dari atribut yang dipilih. Setiap subset akan menjadi cabang dari node yang terbentuk pada tahap 4.

6. Ulangi Proses

   Ulangi langkah 1-5 untuk setiap *subset* data hingga:

   - Jika semua *instance* dalam subset termasuk ke dalam kelas yang sama, buat node terminal (*leaf node*) dengan kelas tersebut.
   - Jika subset kosong, gunakan kelas mayoritas dari dataset sebelumnya sebagai prediksi atau node terminal (*leaf node*).
   - Jika atribut habis, gunakan kelas mayoritas dari dataset saat ini sebagai prediksi node terminal (*leaf node*).
   - Jika kedalaman pohon yang diinginkan telah tercapai, gunakan kelas mayoritas dari dataset saat ini sebagai prediksi node terminal (*leaf node*).

7. Validasi Model

   Uji pohon keputusan menggunakan data uji (*testing data*) untuk mengevaluasi akurasi prediksi.

BAB II

DATA CLEANING DAN *PREPROCESSING*

2.1 *Cleaning*

Dalam menangani data yang kosong, kami melakukan imputasi median terhadap data numerik karena median tidak banyak terpengaruh oleh *outlier* dan menjaga distribusi asli data, sehingga cocok untuk data yang tidak berdistribusi normal. Untuk data kategorikal, kami melakukan imputasi konstan dengan *fill value* berupa '-' yang menandakan *none*, karena hal ini akan mempermudah interpretasi data kosong.

Dalam menangani *outlier*, kami mengubah nilai *outlier* menjadi rata-rata dari keseluruhan data karena rata-rata adalah representasi umum dari data, sehingga membantu menjaga konsistensi dan mencegah bias yang disebabkan oleh nilai ekstrim. Pendekatan ini juga memastikan data tetap berada dalam skala yang wajar tanpa menghilangkan sampel.

Kami menangani data duplikat dengan cara menghapusnya untuk menghindari pengaruh negatif pada model pembelajaran mesin, seperti *overfitting* atau *bias*. Penghapusan duplikat juga membantu menjaga keakuratan dan kualitas *dataset*.

Terdapat beberapa pendekatan *Feature Engineering* yang dapat dilakukan untuk meningkatkan kualitas data. fitur-fitur yang terkait dengan waktu dapat diolah lebih lanjut dengan membuat fitur turunan. Misalnya, menghitung rasio *jitter* (*sjit/djit*), waktu antar paket yang dilakukan normalisasi, atau rasio waktu pengaturan koneksi TCP.

Kemudian, karakteristik lalu lintas jaringan juga dapat dimanfaatkan dengan membuat fitur interaksi, seperti rasio byte (*sbytes/dbytes*), rasio jumlah paket (*spkts/dpkts*), dan rasio beban (*sload/dload*).

Selanjutnya, *Window Metric* dan *Transmission Metric* juga dapat dihitung seperti pemanfaatan *TCP Window* dan efisiensi transmisi. Contohnya, rasio ukuran jendela terhadap jumlah paket atau perbedaan nomor urut dasar (*base sequence number*). Metrik ini penting untuk mengevaluasi efisiensi pengiriman data pada jaringan.

Terakhir, untuk fitur numerik, transformasi statistik seperti transformasi log pada data yang *skewed*, standardisasi, atau normalisasi dapat diterapkan untuk menyeimbangkan distribusi data. Selain itu, variabel kontinu dapat diubah menjadi kategori melalui proses *binning*. Dengan kombinasi pendekatan ini, kualitas data dapat ditingkatkan sehingga mendukung analisis atau model prediktif secara lebih optimal.

2.2 *Preprocessing*

Dilakukan *scaling* pada data menggunakan *MinMax Scaling*/Normalisasi utama untuk memastikan bahwa data numerik berada dalam rentang nilai tertentu, biasanya antara 0 dan 1. Hal ini penting karena banyak algoritma pembelajaran mesin, terutama *k-nearest neighbors* (KNN) yang akan diimplementasikan pada tugas ini, sangat sensitif terhadap skala fitur. Selain itu, *Min-Max Scaling* membantu meningkatkan efisiensi komputasi. Dengan mereduksi nilai data ke rentang kecil, algoritma dapat berjalan lebih cepat dan stabil.

Penggunaan *One-Hot Encoder* pada data kategorial dilakukan karena seluruh fitur kategorial pada data ini dianggap tidak memiliki bobot atau urutan yang signifikan. Dalam kasus seperti ini, setiap nilai unik dari fitur kategorial direpresentasikan sebagai vektor biner, di mana hanya satu elemen yang bernilai 1 sementara sisanya bernilai 0. Pendekatan ini memastikan bahwa tidak ada nilai yang dianggap lebih besar atau lebih kecil dari nilai lainnya.

BAB III

PERBANDINGAN HASIL PREDIKSI ALGORITMA *FROM SCRATCH* DENGAN *LIBRARY*

3.1 *K-Nearest Neighbors* (KNN)

Berikut ini adalah hasil perbandingan yang diperoleh dari prediksi antara algoritma *from scratch* dengan *library* (atas: *library*, bawah: *from scratch*):

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score

clf = KNeighborsClassifier()
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

[27] ✓ 44.8s

```
F1-Score (Macro Average): 0.44328900211928773
```

```
from scratch.knn import KNearestNeighbors
from sklearn.metrics import f1_score

clf = KNearestNeighbors()
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

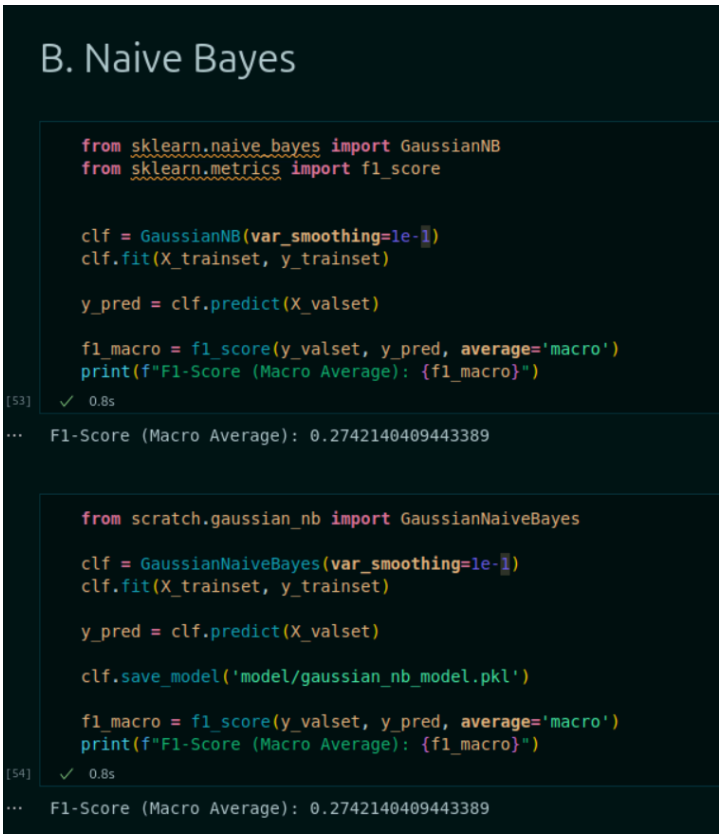
[29] ✓ 8m 29.2s

```
F1-Score (Macro Average): 0.3721352880835775
```

Berdasarkan gambar tersebut, algoritma KNN *from scratch* menghasilkan *F1-Score* yang lebih kecil daripada algoritma KNN dari *library* scikit-learn. Hal utama yang menyebabkan perbedaan *F1-Score* tersebut adalah karena adanya langkah efisiensi yang dilakukan pada algoritma *from scratch*, yaitu melakukan *skip*/pengurangan data *training* yang dimasukkan menjadi hanya 1/1000 dari data *training* semula. Pengurangan data *training* ini menyebabkan penurunan kualitas prediksi yang dihasilkan oleh algoritma KNN *from scratch*. Alasan dari pengurangan data *training* ini adalah demi mempercepat proses pencarian saja.

3.2 *Gaussian Naive-Bayes*

Berikut ini adalah hasil perbandingan yang diperoleh dari prediksi antara algoritma *from scratch* dengan *library* (atas: *library*, bawah: *from scratch*):



Pada hasil F1-*score* tersebut dapat dilihat bahwa kedua implementasi menghasilkan F1-*score* yang identik dan presisi hingga mencapai 17 angka desimal. Hal tersebut menunjukkan bahwa implementasi *from scratch* telah berhasil mereplikasi fungsi GaussianNB dari *library* dengan sempurna. Hal tersebut juga membuktikan bahwa implementasi *from scratch* telah berhasil menerapkan semua komponen inti dari algoritma *Gaussian Naive Bayes* dengan benar, termasuk perhitungan *probabilitas prior*, *estimasi likelihood Gaussian*, *smoothing variance*, perhitungan *log probability* untuk stabilitas numerik, dan klasifikasi *maximum likelihood* sesuai dengan referensi dari dokumentasi yang ada.

3.3 *Iterative Dichotomiser 3* (ID3)

Berikut ini adalah hasil perbandingan yang diperoleh dari prediksi antara algoritma *from scratch* dengan *library* (atas: *library*, bawah: *from scratch*):

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score

clf = DecisionTreeClassifier(random_state=1, criterion='entropy', max_depth=20)
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

✓ 7.5s

```
F1-Score (Macro Average): 0.5604856860060033
```

```python
from scratch.id3 import IterativeDichotomiser3
from sklearn.metrics import f1_score

clf = IterativeDichotomiser3(20)
clf.target_attribute = 'attack_cat'
clf.load_tree("id3_tree.json")  # the tree is from the previous run)

y_pred = clf.predict(val_set_transform)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

✓ 5.2s

```
Tree loaded from id3_tree.json
F1-Score (Macro Average): 0.5230438580456244
```

Berdasarkan gambar di atas, terdapat perbedaan *F1-Score* antara algoritma ID3 *from scratch* dan ID3 dengan *library*. Salah satu penyebabnya adalah penggunaan *mean* sebagai *breakpoint*.

Pada algoritma ID3 *from scratch*, nilai *mean* dari atribut numerik digunakan sebagai titik pemecahan (*split point*). Pendekatan ini tidak optimal karena tidak mempertimbangkan pemisahan yang memberikan *information gain* maksimal, sehingga pohon keputusan yang dihasilkan kurang akurat. Alasan pendekatan ini digunakan hanya demi mempercepat proses pembentukan model.

Pada DecisionTreeClassifier (scikit-learn), pemilihan *breakpoint* dilakukan secara iteratif di seluruh nilai potensial atribut numerik untuk mencari pemisahan terbaik. Selain itu, scikit-learn sudah mengimplementasikan optimisasi komputasi dan algoritma yang efisien untuk pemilihan atribut, perhitungan *entropy*, dan pemilihan *split point* terbaik. Hal ini memberikan akurasi yang lebih tinggi dibandingkan dengan implementasi sederhana *from scratch*.

KONTRIBUSI

NIM	**Kontribusi**
13522006	Implementasi ID3
13522014	*Data cleaning* dan *preprocessing*
13522034	Implementasi KNN
13522057	Implementasi Naive-Bayes

REFERENSI

https://scikit-learn.org/1.5/modules/naive_bayes.html#gaussian-naive-bayes

https://scikit-learn.org/1.5/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/