Tugas Besar 2 IF3070 Intelegensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Disusun oleh:

Kelompok 38

Daffa Ramadhan Elengi	18222009/K01
Alessandro Jusack Hasian	18222025/K01
Matthew Nicholas Gunawan	18222058/K02
Viktor Arsidiantoro Siringoringo	18222083/K01

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

BAB I
PENJELASAN ALGORITMA MACHINE LEARNING

1. K-Nearest Neighbor (KNN)

K-Nearest Neighbor (KNN) adalah algoritma machine learning yang digunakan untuk tugas klasifikasi dan regresi. KNN bekerja berdasarkan prinsip bahwa data yang serupa cenderung berada dalam jarak yang dekat di ruang fitur. Berikut adalah cara kerja KNN:

a. Penentuan Parameter K

Parameter K dalam KNN menentukan jumlah tetangga terdekat yang digunakan untuk prediksi. Nilai K yang kecil membuat model lebih sensitif dan cenderung overfit terhadap noise, sementara nilai K yang lebih besar menghasilkan generalisasi yang lebih baik namun dapat mengabaikan pola lokal dalam data.

b. Perhitungan Jarak

Untuk memprediksi data baru, pertama-tama dihitung jarak antara data baru tersebut dan data pelatihan. Beberapa metode yang umum digunakan untuk perhitungan jarak antara lain Euclidean, Manhattan, dan Minkowski. Dalam penerapan ini, kami menggunakan metode Euclidean. Agar tidak ada fitur yang mendominasi jarak, data baru lalu perlu di-scaling.

c. Penentuan Tetangga Terdekat

Setelah jarak dihitung, data pelatihan diurutkan berdasarkan jaraknya terhadap data baru, dari yang terdekat hingga yang terjauh. Kemudian, diambil K data pelatihan terdekat sebagai tetangga terdekat. Nilai K yang digunakan sangat mempengaruhi hasil prediksi.

d. Klasifikasi

Untuk menentukan kelas dari data baru, KNN melihat mayoritas kelas dari K tetangga terdekat. Jika semua tetangga memiliki kelas yang sama, maka data baru diklasifikasikan ke dalam kelas tersebut. Namun, jika terdapat perbedaan kelas,

dilakukan proses voting untuk memilih kelas yang paling sering muncul.

Implementasi algoritma :

```
class KNearestNeighbors:

    def __init__(self, k=3, distance='euclidean', p=2):

        self.k = k

        self.distance = distance

        self.p = p

        self.X_train = None

        self.y_train = None



    def fit(self, X, y):

        self.X_train = np.array(X)

        self.y_train = np.array(y)



    def _compute_distance(self, x_new):

        if self.distance == 'euclidean':

                distances = np.sqrt(np.sum((self.X_train - x_new) ** 2,
axis=1))

        elif self.distance == 'manhattan':

            distances = np.sum(np.abs(self.X_train - x_new), axis=1)

        elif self.distance == 'minkowski':

              distances = np.sum(np.abs(self.X_train - x_new) ** self.p,
axis=1) ** (1 / self.p)

        else:

                raise ValueError("Pilih 'euclidean', 'manhattan', atau
'minkowski'.")
```

```
        return distances


    def predict(self, X):
        X = np.array(X)
        y_pred = []

        for x_new in X:
            distances = self._compute_distance(x_new)
            nearest_indices = np.argsort(distances)[:self.k]
            nearest_labels = self.y_train[nearest_indices]
            labels, counts = np.unique(nearest_labels, return_counts=True)
            majority_label = labels[np.argmax(counts)]

            y_pred.append(majority_label)

        return np.array(y_pred)
```

Untuk penggunaan model KNN teresbut, secara *default* saat memanggil kelas seperti :

```
knn = KNearestNeighbors(k = 3)
```

Tipe *distance* yang dipakai tanpa spesifikasi parameter *distance* adalah *"*euclidean*"*.

Jika ingin menggunakan *distance* lain ("manhattan" , "minkowski") bisa melakukan seperti ini

```
knn = KNearestNeighbors(k = 3, distance="minkowski")

knn = KNearestNeighbors(k = 3, distance="manhattan")
```

2. Gaussian Naive Bayes

Gaussian Naive Bayes adalah varian dari algoritma Naive Bayes yang digunakan khusus untuk data kontinu. Algoritma ini mengasumsikan bahwa fitur-fitur dalam dataset memiliki distribusi Gaussian (normal). Hal ini membuatnya cocok untuk tugas klasifikasi dengan data numerik yang kontinu. Gaussian Naive Bayes didasarkan pada Teorema Bayes, yang dirumuskan sebagai berikut:

$$P(C|X) = \frac{P(X|C)\cdot P(C)}{P(X)}$$

Dengan:

$P(C|X)$ Probabilitas posterior kelas C diberikan data X.

$P(X|C)$: Probabilitas fitur X diberikan kelas C (*likelihood*).

$P(C)$: Probabilitas awal dari kelas C (*prior*).

$P(X)$: Probabilitas awal dari data X (*normalizing constant*).

Proses kerja Gaussian Naive Bayes adalah sebagai berikut:

a. Tahap Pelatihan (Fit):

- Menghitung probabilitas prior $P(C)$ untuk setiap kelas, berdasarkan proporsi data dalam kelas terhadap total data.
- Menghitung rata-rata (μ) dan variansi (σ^2) untuk setiap fitur dalam masing-masing kelas.
- Menambahkan nilai kecil (ϵ) pada variansi untuk menghindari pembagian dengan nol saat perhitungan probabilitas.

b. Perhitungan Likelihood dengan Gaussian PDF (Probability Density Function)

- Fungsi Probability Density Function (PDF) Gaussian digunakan untuk menghitung probabilitas *likelihood* $P(X|C)$ setiap fitur.
- Probabilitas *likelihood* dihitung untuk setiap fitur menggunakan nilai rata-rata (μ) dan variansi (σ2) yang diperoleh selama pelatihan.

c. Perhitungan Probabilitas Posterior

- Setelah menghitung prior dan likelihood, hitung probabilitas posterior setiap kelas berdasarkan nilai mean dan variansi yang telah dihitung sebelumnya. Juga, perhitungan dilakukan dalam bentuk log untuk menghindari underflow.

d. Penentuan Prediksi Kelas

- Kelas dengan probabilitas posterior tertinggi akan dipilih.\

Implementasi algoritma :

```
class GaussianNaiveBayes:

    def __init__(self):

        self.classes = None

        self.priors = {}

        self.means = {}

        self.vars = {}

    def fit(self, X, y):

        if isinstance(X, pd.DataFrame):

            X = X.values

        if isinstance(y, pd.Series):

            y = y.values

        self.classes = np.unique(y)

        for cls in self.classes:

            X_cls = X[y == cls]

            self.priors[cls] = X_cls.shape[0] / X.shape[0]

            self.means[cls] = np.mean(X_cls, axis=0)

            self.vars[cls] = np.var(X_cls, axis=0) + 1e-9
```

```
    def pdf(self, x, mean, var):
        coeff = 1.0 / np.sqrt(2.0 * np.pi * var)
        exponent = np.exp(- (x - mean)**2 / (2 * var))
        return coeff * exponent
    def class_likelihood(self, x):
        posteriors = {}
        for cls in self.classes:
            prior = np.log(self.priors[cls])
            pdf_vals = self.pdf(x, self.means[cls], self.vars[cls])
            pdf_vals_clipped = np.clip(pdf_vals, 1e-15, None)
            likelihoods = np.sum(np.log(pdf_vals_clipped))
            posterior = prior + likelihoods
            posteriors[cls] = posterior
        return posteriors
    def predict(self, X):
        if isinstance(X, pd.DataFrame):
            X = X.values
        y_pred = []
        for x in X:
            posteriors = self.class_likelihood(x)
            cls = max(posteriors, key=posteriors.get)
            y_pred.append(cls)
        return np.array(y_pred)
```

Penjelasan singkat mengenai algoritma :

Struktur Kelas

- **self.classes**: Menyimpan semua label unik. Misalnya, jika target memiliki label ["0", "1"], maka self.classes = ["0", "1"].
- **self.priors**: Menyimpan P(y=kelas). Contoh: P("1")=0.3
- **self.means dan self.vars**: Masing-masing berisi array mean dan varians untuk **setiap fitur** pada masing-masing kelas. Dalam Gaussian NB, kita asumsikan setiap fitur mengikuti distribusi normal

Metode **fit**

- **Pisahkan Data Sesuai Kelas:**
  - Contoh: Jika y berisi [0, 1, 0, 1, 0, 0], maka X[y == 0] adalah semua baris di mana label = 0.
- **Hitung Prior P(y=kelas):**
  - prior = (jumlah sampel di kelas/total sampel)
  - Contoh: Jika dari 100 data, 40 adalah "0" , 60 adalah "1" , maka self.priors["0"] = 0.4.
- **Hitung Mean dan Var per Fitur:**
  - Hitung mean dan varians untuk tiap fitur
  - Hasil varians dijumlahkan dengan + 1e-9 . Ini merupakan teknik numerik untuk mencegah pembagian oleh nol saat menghitung PDF nanti.

Hasil akhir→prior, mean, dan var untuk setiap kelas,

Metode **pdf (*Probability Density Function*)**

- Fungsi pdf akan menghitung sebuah nilai dan/atau array x terhadap mean dan varians terhadap mean dan variasn yang sudahdisimpan

Metode **class_likelihood**

- Hitung log prior (*prior* merupakan probabilitas/*likelihood* estimasi awal sebuah tipe kategori dari sebuah kelas)

  e.g. jika jumlah row *phishing* ada 10, row *non-phishing* ada 20. Maka log prior bagi *phising* adalah log(P(phishing) = log(10/(10+20)) = log(0.333)
- Hitung PDF tiap fitur
- Hitung Posterior / log space → (perkalian log prior dengan log(P data | distribusi)

  e.g. menggunakan contoh diatas. Misal log prior dari label phishing adalah log(0.333). Kemudian kita mengetahui misal nilai PDF untuk URLLength yang bernilai 20 adalah 0.999.

  Maka posterior adalah → log(0.333 * 0.999)

  Melalui aturan log bisa diubah menjadi → log(0.333) + log(0.999)

  Jika ada fitur lain maka fitur akan ditambahkan ke samping seperti

  log(prior) + log(PDF fitur 1) + log(PDF fitur 2) dan seterusnya .

Metode **predict**

- Loop untuk setiap sample data
- HItung posterior (melalui **class_likelihood)**
- Pilih posterior terbesar (log posterior tertinggi)
- Kumpulkan prediksi di sebuah variable (y_prd)

BAB II
CLEANING DAN PREPROCESSING

1. Data Cleaning

Untuk data cleaning kami melakukan proses berikut:

a. Handling Missing Data

Kami membuat sebuah kelas `FeatureImputer` untuk menangani proses imputasi data pada dataset, menggunakan SimpleImputer dari Scikit-learn untuk mengisi nilai null pada kolom numerik dan kategorikal dengan mean untuk kolom numerik dan most frequent (modus) untuk kolom kategorikal. Parameter `att` digunakan untuk menentukan daftar kolom khusus (atribut) yang ingin diimputasi secara terpisah, sedangkan label `_col` menunjukkan kolom target yang tidak akan diproses. Metode `fit()` digunakan untuk mempelajari pola dari train, termasuk kolom numerik dan kategorikal yang akan diimputasi, serta mengidentifikasi kolom atribut dan label yang tidak perlu diisi. Metode `transform()` menerapkan proses imputasi ke data train. Kolom yang masuk dalam daftar att diimputasi menggunakan `cat_imputer_att`, kolom numerik umum menggunakan num_imputer, dan kolom kategorikal lainnya menggunakan `cat_imputer_general`. Dengan struktur ini, kode memungkinkan pengisian nilai kosong secara sistematis dan terpisah untuk berbagai jenis kolom dalam dataset.

b. Dealing with Outliers

Kelas `FeatureTransformator` untuk mengurangi *skewness* pada fitur numerik. Dengan parameter `skew_threshold`, mendeteksi kolom numerik yang memiliki *skewness* signifikan dan menerapkan transformasi sesuai jenis *skewness*. *Skewness* positif tinggi diubah menggunakan log1p, sedangkan skewness negatif menggunakan `PowerTransformer()`. Metode `fit()` mempelajari pola *skewness*, sementara `transform()` menerapkan transformasi

ke dataset baru, memastikan distribusi data menjadi lebih normal.

c. Removing Duplicates

Kami membuat kelas `DuplicateRemover` untuk penghapusan data yang terduplikasi.

d. Feature Engineering

Kami membuat feature engineering yang membantu proses data cleaning dengan menciptakan fitur baru yang mengurangi redundansi. `FeatureCreator` menambahkan fitur seperti `resources` untuk menghitung total sumber daya atau `WeightedScore` untuk memberikan bobot pada relevansi domain dan URL, dan fitur - fitur lainnya sehingga model dapat menangkap pola lebih baik. `FeatureSelector` membantu membersihkan data dengan membuang kolom yang kurang relevan, seperti kolom identitas atau detail teknis yang tidak signifikan untuk analisis. Sementara itu, `FeatureObjectDropper` secara otomatis menghapus kolom bertipe objek yang tidak nyambung dengan sebagian besar algoritma *machine learning*. Dengan langkah-langkah ini, proses feature engineering tidak hanya meningkatkan kualitas data tetapi juga memastikan dataset yang lebih terstruktur dan optimal untuk digunakan dalam model.

2. Preprocessing

Untuk tahap *preprocessing* kami melakukan proses berikut:

a. Feature Scaling

Kami melakukan proses *feature scaling* dengan tujuan agar setiap fitur dapat berkontribusi setara terhadap proses pelatihan model sehingga algoritma *machine learning* dapat bekerja secara efektif dengan data.

Pada tahap ini, kami menggunakan metode robust scaling pada kelas `FeatureScaler()`, yang memanfaatkan median dan interquartile range (IQR) untuk menskalakan data. Pendekatan ini memiliki keunggulan dalam mengatasi

outlier karena tidak bergantung pada rata-rata dan simpangan baku seperti metode scaling lainnya.

Proses scaling dilakukan dengan langkah-langkah berikut:

- Deteksi outlier

  Kami menggunakan aturan IQR untuk mendeteksi outlier pada setiap kolom numerik. Kolom yang memiliki persentase outlier lebih besar dari ambang batas yang ditentukan (`outlier_threshold`) dipilih untuk proses scaling. Aturan IQR menggunakan batas bawah $(Q1 - 1.5 \times IQR)$ dan batas atas $(Q1 + 1.5 \times IQR)$ untuk mendefinisikan outlier. Proses ini dilakukan pada fungsi `_check_outliers_iqr()`.

- Pemilihan fitur

  Hanya fitur-fitur numerik dengan persentase outlier yang melebihi ambang batas yang diskalakan. Fitur ini dipilih secara otomatis berdasarkan distribusi datanya. Proses ini dilakukan pada fungsi `fit()`.

- Scaling data

  Data yang terpilih lalu dilakukan scaling dengan formula berikut (sklearn):

$$X_{scaled} = \frac{X - Median(X)}{IQR(X)}$$

  Formula ini memastikan bahwa data akan disesuaikan dengan median dan IQR sehingga tetap *robust* terhadap outlier. Proses ini dilakukan pada fungsi `transform()`.

b. Feature Encoding

Kami melakukan proses *feature encoding* dengan tujuan agar jenis data non-numerik dapat diubah menjadi numerik sehingga bisa digunakan sebagai masukan untuk algoritma *machine learning*.

Pada tahap ini, kami menggunakan dua metode feature encoding pada kelas `FeatureEncoder()`, yaitu:

- Target Encoding dengan Top-N Categories

  Pada kolom target tertentu (`target_column`), data non-numerik yang tidak termasuk dalam Top-N berdasarkan frekuensi akan diubah menjadi label `else`. Metode ini digunakan dengan tujuan mengurangi jumlah kategori untuk menghindari *dimensionality explosion* dikarenakan jumlah kategori yang terlalu besar saat menggunakan metode one-hot encoding.

  Kami memilih nilai N yaitu 4 untuk target kolom TLD sehingga hanya 4 TLD dengan jumlah frekuensi terbanyak yang lalu akan ditampilkan, sisanya akan diganti menjadi satu kategori gabungan.

- One-Hot Encoding

  Setelah nilai pada kolom `target_column` diubah, semua kolom dengan tipe data object lalu diproses menggunakan one-hot encoding. Penerapan metode ini akan mengonversi setiap data non-numerik menjadi representasi biner di setiap kolom terpisah.

Penerapan kedua metode ini dilakukan pada fungsi `transform()`.

c. Handling Imbalanced Dataset

Kami melakukan proses *handling imbalanced dataset* dikarenakan data yang tidak seimbang dapat mengakibatkan beberapa permasalahan yang berdampak negatif terhadap performa dan *reliability* dari model *machine learning*, seperti terjadinya bias, akurasi yang *misleading*, maupun generalisasi yang buruk dikarenakan ada beberapa kelas yang *underrepresented* pada data.

Pada tahap ini kami menyiapkan resampling method yaitu SMOTE, SMOTEENN, dan SMOTETomek.

- SMOTE (Synthetic Minority Oversampling Technique)

Metode ini menghasilkan sampel sintetis untuk kelas minoritas dengan cara interpolasi linear. Data baru dibuat antara titik-titik data minoritas yang ada sehingga menyeimbangkan proporsi kelas dalam dataset.

Metode ini memiliki keunggulan yaitu menambah data sintetis hanya untuk kelas minoritas saja dan menghindari penggandaaan data asli secara langsung.

- SMOTEENN (SMOTE + Edited Nearest Neighbors)

  SMOTEEN adalah metode kombinasi antara SMOTE dengan Edited Nearest Neighbor (ENN). Kombinasi dengan ENN akan menghapus sampel yang dianggap salah klasifikasi atau noise berdasarkan jarak terdekatnya untuk membersihkan dataset setelah *oversampling*.

  Metode ini memiliki keunggulan yaitu dapat membersihkan noise dan kesalahan dari dataset sehingga cocok untuk dataset yang mengandung banyak noise pada kelas mayoritas.

- SMOTETomek (SMOTE + Tomek Links)

  SMOTETomek adalah metode kombinasi antara SMOTE dengan Tomek Links. Kombinasi dengan Tomek Links akan menghapus pasangan sampel yang saling bertetangga dengan label berbeda dengan tujuan mengurangi *overlap* antara kelas.

  Metode ini memiliki keunggulan yaitu dapat mengurang *overlap* antara kelas mayoritas dan minoritas setelah *oversampling* sehingga mampu mengurangi ambiguitas data pada perbatasan kelas.

Kami mengimplementasikan ketiga metode tersebut pada fungsi `my_smote()`, `my_smotenn()`, dan `my_smotetomek()`. Namun, pada penggunaannya kami memutuskan hanya memakai metode SMOTE saja dikarenakan pada permasalahan berikut berhasil menghasilkan performa yang jauh lebih baik dibanding kedua metode sisanya.

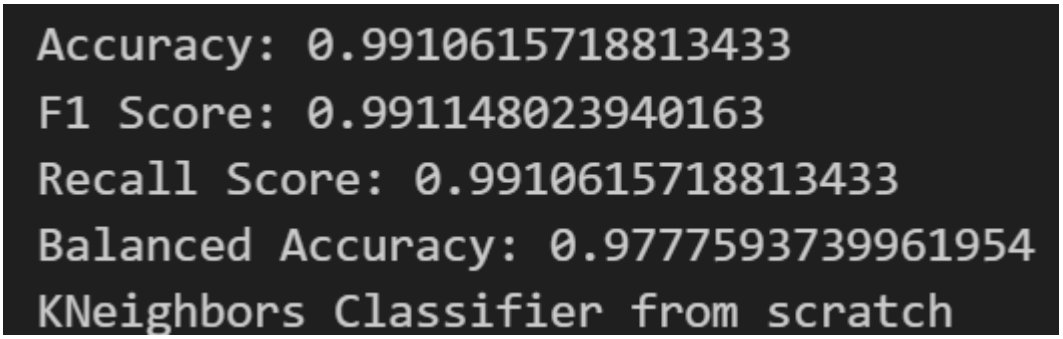
BAB III
HASIL DAN PEMBAHASAN

1. Perbandingan Hasil Implementasi dan Hasil yang Didapatkan dari Pustaka

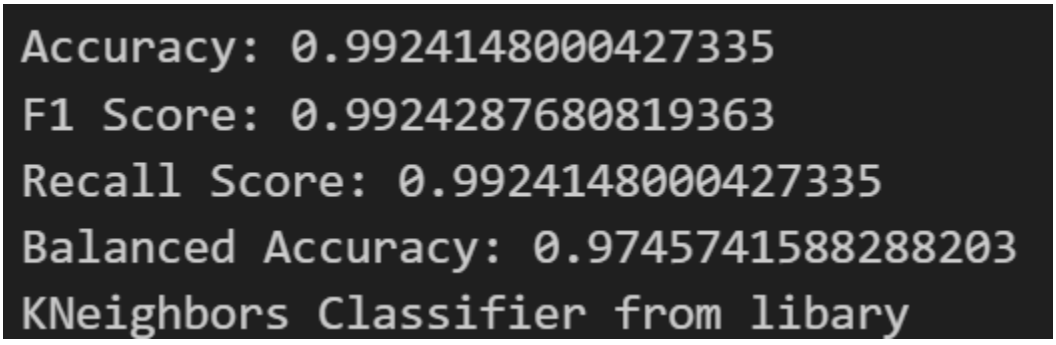
Kami menggunakan fungsi accuracy_score dari sklearn.metrics untuk menghitung akurasi prediksi.

a. KNN

Berikut merupakan akurasi yang didapat dari implementasi KNN yang kami buat.



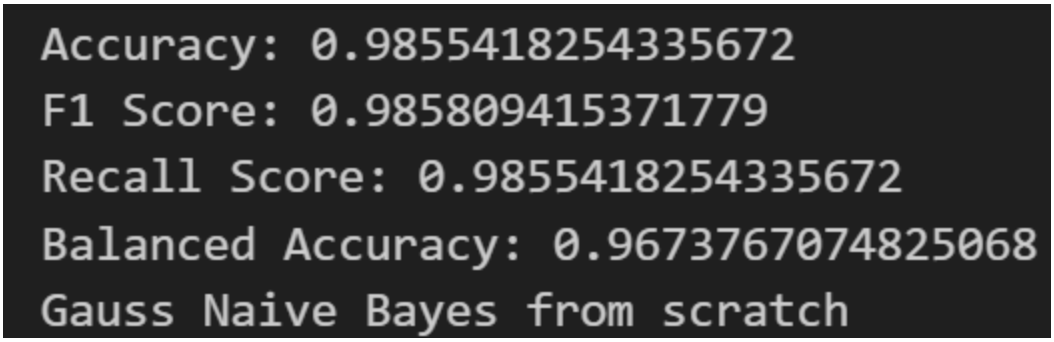
Berikut merupakan akurasi knn dari pustaka.



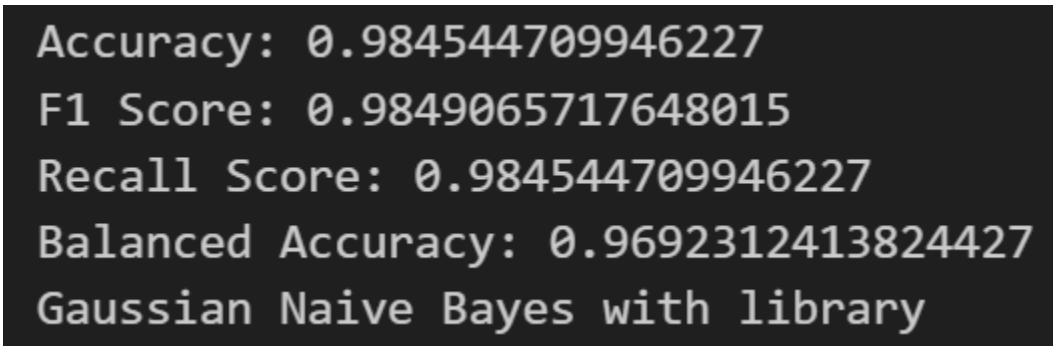
Dapat dilihat bahwa akurasi implementasi cukup dekat, kurang sedikit dari akurasi *library*. Waktu yang dibutuhkan untuk implementasi dari scratch memerlukan hampir 10 menit sedangkan dari library hanya membutuhkan beberapa detik saja.

b. Naive Bayes

Berikut merupakan akurasi dari implementasi Naive Bayes kami.



Berikut merupakan akurasi implementasi Naive Bayes menggunakan library sklearn



Dapat dilihat bahwa akurasi dan waktu yang dibutuhkan dari kedua implementasi baik dari library ataupun scratch sudah hampir sama.

BAB IV
KESIMPULAN DAN SARAN

Dari semua algoritma yang kami terapkan, algoritma KNN memberikan akurasi yang paling besar, dapat lebih besar jika menggunakan library. Lalu, algoritma Gaussian Naive Bayes memberikan nilai akurasi terbesar kedua dengan nilai scratch yang hampir sama dengan penggunaan library. Selain pengembangan model dan pemilihan model terbaik, tahap data cleaning and preprocessing juga sangat penting, karena semakin sesuai data yang disiapkan untuk pemrosesan machine learning, maka semakin optimal juga hasil model yang telah kami kembangkan.

PEMBAGIAN TUGAS

Daffa Ramadhan Elengi	- Menyusun template dokumen - Menyusun dokumentasi laporan
Alessandro Jusack Hasian	- Membagi training set dan validation set - Melakukan preprocessing - Membangun pemodelan Gaussian Naive Bayes scratch - Dokumen penjelasan algoritma
Matthew Nicholas Gunawan	- Membangun pemodelan KNN scratch - Menyusun dokumentasi laporan
Viktor Arsindiantoro Siringoringo	- Membagi training set dan validation set - Membuat bagian transformasi data - Menyusun imputer untuk data cleaning - Membuat feature engineering - Melakukan proses scaling - Membangun pemodelan KNN scratch

REFERENSI

- https://www.ibm.com/topics/knn
- https://builtin.com/artificial-intelligence/gaussian-naive-bayes
- https://builtin.com/artificial-intelligence/gaussian-naive-bayes
- https://scikit-learn.org/1.5/modules/neighbors.html
- https://scikit-learn.org/1.5/modules/naive_bayes.html