Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial Pencarian Implementasi Algoritma Pembelajaran Mesin

Disusun oleh:

Kelompok 21

Muhammad Rifa Ansyari	/ 18222004
Justin Lawrance	/ 18222006
Axelius Davin	/ 18222016
Natanael Steven Simangunsong	/ 18222054

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

BAB I

Penjelasan KNN

1.1 Implementasi KNN Dengan Scikit-Learn

Implementasi KNN dengan Scikit-Learn dilakukan dengan mengimpor library sklearn.metrics untuk evaluasi dan sklearn.neighbors untuk inisialisasi model KNN melalui KNeighborsClassifier. Model KNN dikonfigurasi dengan parameter n_neighbors=3 untuk menentukan jumlah tetangga terdekat yang digunakan. Selanjutnya, model dilatih menggunakan data training yang sudah di-resample (X_resampled dan y_resampled) dengan metode fit. Prediksi dilakukan pada data validasi yang telah diproses (X_val_transformed) menggunakan predict, menghasilkan prediksi label (y_pred). Evaluasi performa model mencakup perhitungan akurasi dengan accuracy_score, precision menggunakan precision_score dengan rata-rata berbobot, confusion matrix menggunakan confusion_matrix untuk melihat distribusi prediksi benar dan salah, serta classification report melalui classification_report untuk menampilkan precision, recall, F1-score, dan support per kelas. Berikut adalah code-nya.

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_resampled, y_resampled)
y_pred = knn.predict(X_val_transformed)
print(y_pred)
accuracy = accuracy_score(y_val, y_pred)
print(f"Accuracy: {accuracy:.4f}")

precision = precision_score(y_val, y_pred, average='weighted', zero_division=0)
print(f"Precision Score: {precision:.4f}")

conf_matrix = confusion_matrix(y_val, y_pred)
print(f"Confusion Matrix:\n{conf_matrix}")

class_report = classification_report(y_val, y_pred, zero_division=0)
print(f"Classification Report:\n{class_report}")
```

1.2 Implementasi KNN From Scratch

Implementasi KNN dari scratch bisa dilakukan dengan membuat class KNN terlebih dahulu. Lalu membuat 4 method, yang pertama yaitu __init__ yang menerima input parameter seperti berapa neighbor terdekat yang diambil, metric apa yang digunakan, dan jika menggunakan metric minkowski, berapa norma p yang digunakan.

Lalu, dalam method kedua yaitu fit, dalam method ini hanya mengubah data menjadi array. Untuk method ketiga yaitu _distance, method ini melihat metric apa yang dipakai lalu memasukkan jarak kedalam algoritma ke metric yang digunakan. Untuk method terakhir, yaitu predict, method ini menghitung jarak setiap baris yang lalu digabungkan dengan index agar bisa dihubungkan dengan label nantinya. Setelah itu, jarak tersebut diurutkan dari yang paling kecil kemudian diambil beberapa neighbor tersebut berdasarkan k yang sudah ditentukan di parameter. Terakhir, total label dihitung berdasarkan neighbor yang diambil dan menentukan apakah prediksinya benar atau salah berdasarkan jumlah label paling banyak. Metode save dan load memungkinkan model disimpan dan dimuat dari file. Berikut adalah code-nya.

```python
class KNN:
    def __init__(self, k=3, metric='euclidean', p=3):
        self.k = k
        self.metric = metric
        self.p = p

    def fit(self, X_train, y_train):
        self.X_train = np.array(X_train) #mengubah menjadi numpy array
        self.y_train = np.array(y_train)

    def _distance(self, point1, point2):
        if self.metric == 'euclidean':
            return np.sqrt(np.sum((point1 - point2) ** 2))
        elif self.metric == 'manhattan':
            return np.sum(np.abs(point1 - point2))
        elif self.metric == 'minkowski':
            return np.sum(np.abs(point1 - point2) ** self.p) ** (1 / self.p)

    def predict(self, X_test):
        predictions = []
        for test_point in X_test:
            distances = []
            for i, train_point in enumerate(self.X_train):
                distance = self._distance(test_point, train_point) #menghitung distance dengan metric yang sudah ditentukan
                distances.append((distance, i)) #menjadikan hasil ke distance beserta indexnya agar bisa diambil labelnya

            distances.sort(key=lambda x: x[0]) #sort dari distance terkecil

            k_nearest_neighbors = [self.y_train[i] for _, i in distances[:self.k]] #memilih 7 data dengan distance terdekat

            vote = Counter(k_nearest_neighbors).most_common(1) #menghitung jumlah label
            predictions.append(vote[0][0]) #memilih label dengan jumlah terbanyak

        return np.array(predictions)

    def save(self, filename):
        """Save the model to a file."""
        with open(filename, 'wb') as file:
            pickle.dump(self, file)

    @staticmethod
    def load(filename):
        """Load the model from a file."""
        with open(filename, 'rb') as file:
            return pickle.load(file)
```

BAB II

Penjelasan Naive-Bayes

2.1 Implementasi Naive-Bayes Dengan Scikit-Learn

Implementasi Gaussian Naive Bayes dengan Scikit-Learn dilakukan dengan mengimpor library sklearn.naive_bayes untuk inisialisasi model GaussianNB, kemudian melatih model menggunakan data training yang telah di-resample dengan gnb.fit. Prediksi dilakukan pada data validasi yang telah diproses menggunakan pipeline dengan gnb.predict. Evaluasi model dilakukan dengan menghitung confusion matrix menggunakan confusion_matrix untuk melihat distribusi prediksi benar dan salah, serta menghitung akurasi menggunakan accuracy_score untuk mengevaluasi performa keseluruhan. Selain itu, classification report yang mencakup precision, recall, dan F1-score dihitung menggunakan classification_report untuk mendapatkan gambaran lebih rinci mengenai performa model pada setiap kelas target. Berikut adalah code-nya.

```python
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X_resampled, y_resampled)

y_predb = gnb.predict(X_val_transformed)

conf_matrix = confusion_matrix(y_val, y_predb)
print(f"Confusion Matrix:\n{conf_matrix[0]}")
print(conf_matrix[1])

accuracy = accuracy_score(y_val, y_predb)
print(f"\nAccuracy: {accuracy:.4f}")

class_report = classification_report(y_val, y_predb, zero_division=0)
print(f"Classification Report:\n{class_report}")
```

2.2 Implementasi Naive-Bayes From Scratch

Implementasi Naive Bayes dari awal dilakukan dengan membuat kelas NaiveBayes, yang memiliki beberapa metode utama. Metode pertama adalah init, yang bertugas menginisialisasi variabel-variabel untuk menyimpan probabilitas kelas, rata-rata, dan varians dari fitur-fitur data. Metode kedua adalah fit, yang digunakan untuk menghitung probabilitas prior dari setiap kelas, serta menghitung rata-rata dan varians fitur berdasarkan label target. Metode ketiga, _prediction, menghitung likelihood menggunakan distribusi Gaussian dan mengalikan dengan prior untuk menentukan kelas mana yang memiliki probabilitas lebih tinggi. Metode keempat, predict, menghasilkan prediksi label untuk seluruh dataset berdasarkan hasil dari _prediction. Selain itu, metode predict_proba digunakan untuk menghitung probabilitas masing-masing kelas, sedangkan metode save dan load memungkinkan model disimpan dan dimuat dari file. Berikut adalah code-nya.

```python
class NaiveBayes :
    def __init__(self) :
        self.label_probs = {}
        self.mean = {}
        self.var = {}

    def fit(self,data,target) :
        # menggunakan label 0 dan 1
        # label 0
        data_0 = data[target == 0]
        self.label_probs[0] = data_0.shape[0]/data.shape[0]
        self.mean[0] = np.mean(data_0, axis=0)
        self.var[0]  = np.var(data_0, axis=0)
        #label 1
        data_1 = data[target == 1]
        self.label_probs[1] = data_1.shape[0]/data.shape[0]
        self.mean[1] = np.mean(data_1, axis=0)
        self.var[1]  = np.var(data_1, axis=0)

    def predict(self,data) :
        predictions = [self._prediction(x) for x in data]
        return np.array(predictions)

    def _prediction(self,x) :
        like_exp_0 = -((x-self.mean[0])**2)/(2*self.var[0]+(2*1e-5))
        likelihood_0 = np.exp(like_exp_0)/np.sqrt(2*np.pi*self.var[0]+(2*1e-5))
        prior_0 = self.label_probs[0]
        pred_0 = prior_0*likelihood_0.prod()

        like_exp_1 = -((x-self.mean[1])**2)/(2*self.var[1]+(2*1e-5))
        likelihood_1 = np.exp(like_exp_1)/np.sqrt(2*np.pi*self.var[1]+(2*1e-5))
        prior_1 = self.label_probs[1]
        pred_1 = prior_1*likelihood_1.prod()
        if pred_0 > pred_1 :
            return 0
        else :
            return 1

    def predict_proba(self, data):
        probabilities = []
        for x in data:
            prob_0 = self.label_probs[0] * (np.exp(-((x - self.mean[0]) ** 2) /
                                                   (2 * self.var[0] + 1e-5)) /
                                            np.sqrt(2 * np.pi * self.var[0] + 1e-5)).prod()
            prob_1 = self.label_probs[1] * (np.exp(-((x - self.mean[1]) ** 2) /
                                                   (2 * self.var[1] + 1e-5)) /
                                            np.sqrt(2 * np.pi * self.var[1] + 1e-5)).prod()
            probabilities.append([prob_0, prob_1])
        return np.array(probabilities)

    def save(self, filename):
        """Save the model to a file."""
        with open(filename, 'wb') as file:
            pickle.dump(self, file)

    @staticmethod
    def load(filename):
        """Load the model from a file."""
        with open(filename, 'rb') as file:
            return pickle.load(file)
```

BAB III

Data Cleaning and Preprocessing

3.1 Data Cleaning

Data cleaning diperlukan karena data mentah sering mengandung kesalahan seperti duplikasi, data hilang (missing value), atau kesalahan format yang dapat mengurangi akurasi analisis. Dengan melakukan data cleaning, kita memastikan bahwa data yang digunakan dalam proses pengambilan keputusan adalah valid, relevan, dan bebas dari error. Proses ini meningkatkan kualitas data, sehingga menghasilkan wawasan yang lebih andal dan mendukung pengambilan keputusan yang lebih baik. Adapun tahapan data cleaning yang kami lakukan adalah handling missing data, dealing with outliers, remove duplicates, dan feature engineering.

3.1.1 Handling Missing Data

Kami menggunakan dua metode imputasi, yaitu most_frequent dan mean, untuk menangani nilai yang hilang (missing values) pada dataset. Imputasi dengan strategi most_frequent digunakan untuk kolom bertipe object dan boolean, karena tipe data ini biasanya berisi nilai kategori atau True/False, sehingga pengisian dengan nilai yang paling sering muncul (modus) adalah pendekatan yang masuk akal. Sementara itu, strategi mean digunakan untuk kolom bertipe numerik, yaitu float64 dan Int64, karena pengisian nilai yang hilang dengan rata-rata dapat menjaga distribusi data numerik tetap konsisten. Berikut adalah code-nya.

```python
def missingDataHandler(dataset) :
    dataset_copy = dataset.copy()

    imputer_modus = SimpleImputer(strategy='most_frequent')
    imputer_mean  = SimpleImputer(strategy='mean')

    dataset_object = dataset.select_dtypes(include='object').columns
    boolean_dataset = dataset.select_dtypes(include='boolean').columns
    float_dataset = dataset.select_dtypes(include='float64').columns
    int_dataset = dataset.select_dtypes(include='Int64').columns

    dataset[dataset_object]  = imputer_modus.fit_transform(dataset[dataset_object])
    dataset[int_dataset]     = np.ceil(imputer_mean.fit_transform(dataset[int_dataset]))
    dataset[float_dataset]   = imputer_mean.fit_transform(dataset[float_dataset])
    dataset[boolean_dataset] = imputer_modus.fit_transform(dataset[boolean_dataset])

    #mengubah kembali datatype menjadi semua (dari object)
    for col in dataset_copy.columns:
        dataset[col] = dataset[col].astype(dataset_copy[col].dtype)
```

3.1.2 Dealing With Outliers

Kami menggunakan metode clipping. Pada kode ini, kami menangani outlier pada data numerik di dataset menggunakan metode IQR (Interquartile Range). Pertama, kami menghitung kuartil pertama (Q1) dan kuartil ketiga (Q3) untuk setiap kolom numerik, lalu menghitung IQR sebagai selisih antara Q3 dan Q1. Kemudian, kami menentukan batas bawah (lowerBound) dan batas atas (upperBound). Selanjutnya, nilai-nilai yang berada di luar batas tersebut dianggap outlier dan di-clip agar berada dalam rentang batas bawah dan atas. Untuk tipe data Int64, batas bawah dan atas dibulatkan menggunakan np.floor dan np.ceil agar konsisten dengan tipe datanya, sementara untuk tipe numerik lainnya tidak dilakukan pembulatan. Langkah ini bertujuan untuk menangani outlier secara efektif tanpa menghilangkan data, sehingga distribusi data tetap wajar untuk analisis atau pemodelan. Berikut adalah code-nya.

```python
def outlierHandler(dataset) :
  numerical_dataset = dataset.select_dtypes(include='number').columns

  for i in numerical_dataset:
    Q1 = dataset[i].quantile(0.25)
    Q3 = dataset[i].quantile(0.75)

    IQR = Q3-Q1

    threshold = 1

    lowerBound = Q1 - (threshold*IQR)
    upperBound = Q3 + (threshold*IQR)

    if (dataset[i].dtype == 'Int64') :
      dataset[i] = dataset[i].clip(lower=np.floor(lowerBound), upper=np.ceil(upperBound))
    else :
      dataset[i] = dataset[i].clip(lower=lowerBound, upper=upperBound)
```

3.1.3 Remove Duplicates

Kami menghapus duplikasi data berdasarkan kolom URL menggunakan metode drop_duplicates. Hal ini dilakukan karena setiap URL dalam dataset seharusnya unik. Jika terdapat URL yang sama dengan label yang sama, data tersebut tidak menambah informasi baru dan hanya meningkatkan redundansi. Sebaliknya, jika URL yang sama memiliki label yang berbeda (contoh: phishing dan non-phishing), hal ini dapat menyebabkan inkonsistensi data. Berikut adalah code-nya.

```python
def duplicateHandler(dataset):
    dataset.drop_duplicates(subset=["URL"], inplace=True)
```

3.1.4 Feature Engineering

Kami menggunakan metode binning. Fungsi featureBinning digunakan untuk menyederhanakan nilai numerik dalam dataset menjadi kategori dengan melakukan proses binning. Kolom numerik, kecuali id, diubah menjadi tiga kategori: low, medium, dan high, berdasarkan nilai batas yang dihitung dari pembagian rentang minimum hingga maksimum menjadi tiga bagian yang sama besar. Setelah proses ini, kolom numerik asli dihapus, dan kolom baru hasil binning ditambahkan ke dataset dengan tipe data category untuk efisiensi memori dan pemrosesan. Proses ini membantu menyederhanakan data, mengurangi pengaruh outlier, dan mempermudah model dalam mengenali pola pada fitur dengan rentang nilai yang besar. Fungsi featureEngineeringHandler mengelola proses feature engineering, dengan menambahkan hasil binning ke dataset asli dan menghapus semua kolom bertipe object. Berikut adalah code-nya.

```python
def featureBinning(dataset):
    numerical_dataset = dataset[[col for col in dataset.select_dtypes(include='number').columns if col != 'id']].copy()
    old_columns = numerical_dataset.columns

    for col in numerical_dataset:
        min_val = numerical_dataset[col].min()
        max_val = numerical_dataset[col].max()
        val1 = min_val + ((max_val - min_val) // 3)
        val2 = min_val + ((max_val - min_val) * 2 // 3)

        numerical_dataset.loc[numerical_dataset[col] <= val1, 'bin_' + col] = 'low'
        numerical_dataset.loc[(numerical_dataset[col] > val1) & (numerical_dataset[col] <= val2), 'bin_' + col] = 'medium'
        numerical_dataset.loc[numerical_dataset[col] > val2, 'bin_' + col] = 'high'

    numerical_dataset = numerical_dataset.drop(columns=old_columns)
    numerical_dataset = numerical_dataset.astype('category')
    return numerical_dataset

def featureEngineeringHandler(dataset) :
    binned_data = featureBinning(dataset)
    dataset = pd.concat([dataset, binned_data], axis=1)
    dataset.drop(columns=dataset.select_dtypes(include='object').columns, inplace=True)
    return dataset
```

3.2 Data Preprocessing

Data preprocessing adalah langkah yang lebih luas yang mencakup data cleaning serta transformasi tambahan untuk membuat data siap digunakan oleh algoritma machine learning. Tujuan utamanya meliputi: memastikan konsistensi dan kualitas data, menangani nilai yang hilang atau tidak valid, mengubah data ke dalam format yang sesuai (seperti encoding data kategorikal atau normalisasi), dan mengurangi noise dalam dataset. Proses ini penting karena data mentah sering kali tidak langsung dapat digunakan, sehingga preprocessing menjadi langkah krusial untuk meningkatkan performa dan akurasi model. Adapun tahapan data preprocessing yang kami lakukan adalah feature scaling, feature encoding, dan handling imbalance dataset.

3.2.1 Feature Scaling

Kami menggunakan metode Min-Max Scaling. Proses feature scaling ini bertujuan untuk mengubah rentang nilai fitur menjadi skala yang seragam (biasanya antara 0 dan 1), sehingga model pembelajaran mesin tidak memberikan bobot berlebihan pada fitur dengan nilai yang lebih besar. Pada metode fit, kolom numerik dalam dataset diidentifikasi menggunakan tipe data Int64 dan float64, kemudian skala ditentukan berdasarkan nilai minimum dan maksimum dari kolom-kolom tersebut. Pada metode transform, dataset diubah dengan menskalakan nilai-nilai kolom numerik menggunakan skala yang telah ditentukan. Berikut adalah code-nya.

```python
class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.scaler = MinMaxScaler()
        self.numerical_columns = []

    def fit(self, X, y=None):
        self.numerical_columns = X.select_dtypes(include=['Int64', 'float64']).columns
        self.scaler.fit(X[self.numerical_columns])
        return self

    def transform(self, X):
        X_transformed = X.copy()
        X_transformed[self.numerical_columns] = self.scaler.transform(X_transformed[self.numerical_columns])
        return X_transformed
```

3.2.2 Feature Encoding

Kami menggunakan metode One-Hot Encoding. Proses ini bertujuan mengubah data kategori menjadi representasi numerik agar dapat digunakan oleh algoritma pembelajaran mesin, yang umumnya bekerja lebih baik dengan data numerik. Pada metode fit, kolom dengan tipe data category diidentifikasi, kemudian encoder dilatih untuk memahami kategori unik pada kolom tersebut. Pada metode transform, data kategori dikonversi menjadi bentuk one-hot encoded dalam format DataFrame pandas. Setiap kategori diubah menjadi kolom baru, dengan nama

kolom berupa gabungan nama kolom asli dan nilai kategori. Selanjutnya, kolom kategori asli dihapus, dan kolom hasil encoding ditambahkan ke dataset. Berikut adalah code-nya.

```python
class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.encoder = OneHotEncoder(sparse_output=False) #mengeluarkan dalam array dan bukan sparse matrix
        self.categorical_columns = []

    def fit(self, X, y=None):
        self.categorical_columns = X.select_dtypes(include=['category']).columns
        self.encoder.fit(X[self.categorical_columns])
        return self

    def transform(self, X):
        encoded_data = pd.DataFrame( #mengeluarkan pandas dataframe dan bukan numpy array
            self.encoder.transform(X[self.categorical_columns]),
            columns=self.encoder.get_feature_names_out(self.categorical_columns), #menggabungkan nama column lama dengan isi valuenya
            index=X.index
        )
        return pd.concat([X.drop(columns=self.categorical_columns), encoded_data], axis=1) #drop column original lalu membuat column baru dengan nama gabungan
```

3.2.2 Handling Imbalanced Dataset

Kami menggunakan metode undersampling dengan teknik Tomek Links untuk mengurangi ketidakseimbangan data. Dilakukan undersampling karena dataset yang digunakan untuk training sangat imbalanced, melakukan oversampling beresiko memperburuk kualitas data jika data minoritas memiliki kualitas buruk. Selain itu, Tomek Links bekerja dengan cara menghapus instance dari kelas mayoritas yang berada dekat dengan instance dari kelas minoritas sehingga berkontribusi pada proses data cleaning.

```python
pipe = ImbPipeline([
    ('preprocessor', preprocessor),
    ('undersampler', TomekLinks(sampling_strategy='auto')),
])
```

BAB IV

Perbandingan Hasil Prediksi

4.1 Hasil Prediksi Dengan Implementasi KNN

Berikut ini adalah perbandingan hasil prediksi dari implementasi KNN dengan Scikit-Learn dan KNN from scratch.

- Hasil prediksi KNN dengan Scikit Learn :

```
Accuracy: 0.9519
Precision Score: 0.9472
Confusion Matrix:
[[  959   866]
 [  294 21973]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.77      0.53      0.62      1825
         1.0       0.96      0.99      0.97     22267

    accuracy                           0.95     24092
   macro avg       0.86      0.76      0.80     24092
weighted avg       0.95      0.95      0.95     24092
```

- Hasil prediksi KNN from scratch :

```
Accuracy: 0.9535
Precision Score: 0.9531
Confusion Matrix:
[[  51   53]
 [   3 1097]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.49      0.65       104
         1.0       0.95      1.00      0.98      1100

    accuracy                           0.95      1204
   macro avg       0.95      0.74      0.81      1204
weighted avg       0.95      0.95      0.95      1204
```

Berdasarkan hasil prediksi dari implementasi KNN dengan Scikit-Learn dan KNN from scratch, terlihat hasil accuracy f1-score dari KNN dengan Scikit-Learn dan KNN from scratch memiliki nilai serupa, yaitu 0.95. Hal ini berarti algoritma KNN from scratch sudah cukup optimal. Untuk mendapatkan hasil yang lebih baik, diperlukan data cleaning dan data preprocessing yang lebih baik.

Implementasi KNN dengan Scikit-Learn jauh lebih mudah daripada from scratch karena hanya memasukan kedalam fungsi yang sudah ada. KNN dengan Scikit-Learn juga jauh lebih dioptimalkan sehingga lebih efisien dan cepat dibanding from scratch. Sehingga KNN dengan

Scikit-Learn lebih cocok untuk dataset yang lebih besar dan KNN from scratch lebih sulit dioptimalkan jika datasetnya besar.

4.2 Hasil Prediksi Dengan Implementasi Naive-Bayes

Berikut ini adalah perbandingan hasil prediksi dari implementasi Naive-Bayes dengan Scikit-Learn dan Naive-Bayes from scratch.

- Hasil prediksi Naive-Bayes dengan Scikit Learn :

```
Confusion Matrix:
[922 903]
[  200 22067]

Accuracy: 0.9542
Classification Report:
              precision    recall  f1-score   support

         0.0       0.82      0.51      0.63      1825
         1.0       0.96      0.99      0.98     22267

    accuracy                           0.95     24092
   macro avg       0.89      0.75      0.80     24092
weighted avg       0.95      0.95      0.95     24092
```

- Hasi prediksi Naive-Bayes from scratch :

```
Confusion Matrix:
[1517  308]
[ 1550 20717]

Accuracy: 0.9229
Classification Report:
              precision    recall  f1-score   support

         0.0       0.49      0.83      0.62      1825
         1.0       0.99      0.93      0.96     22267

    accuracy                           0.92     24092
   macro avg       0.74      0.88      0.79     24092
weighted avg       0.95      0.92      0.93     24092
```

-

Berdasarkan hasil prediksi dari implementasi Naive-Bayes dengan Scikit-Learn dan Naive-Bayes from scratch, terlihat hasil accuracy f1-score dari Naive-Bayes dengan Scikit-Learn bernilai 0.95, dan dari Naive-Bayes from scratch bernilai 0.92. Berdasarkan perbandingan tersebut, dapat disimpulkan bahwa implementasi yang kami buat pada saat ini dapat ditingkatkan lebih lanjut, terutama dalam implementasi formula/rumus yang digunakan, serta angka eps terbaik yang diperlukan untuk menghindari pembagian dengan 0.

BAB V

Pembagian Tugas

NIM	Tugas
18222004	- Mengerjakan notebook
18222006	- Mengerjakan notebook
18222016	- Mengerjakan notebook
18222054	- Mengerjakan laporan

BAB VI

Referensi

https://scikit-learn.org/1.5/glossary.html#term-random_state
https://scikit-learn.org/1.5/modules/neighbors.html
https://scikit-learn.org/1.5/modules/naive_bayes.html
https://scikit-learn.org/stable/modules/neighbors.html
https://scikit-learn.org/stable/modules/naive_bayes.html
https://scikit-learn.org/stable/glossary.html
https://www.geeksforgeeks.org/feature-encoding-techniques-machine-learning/