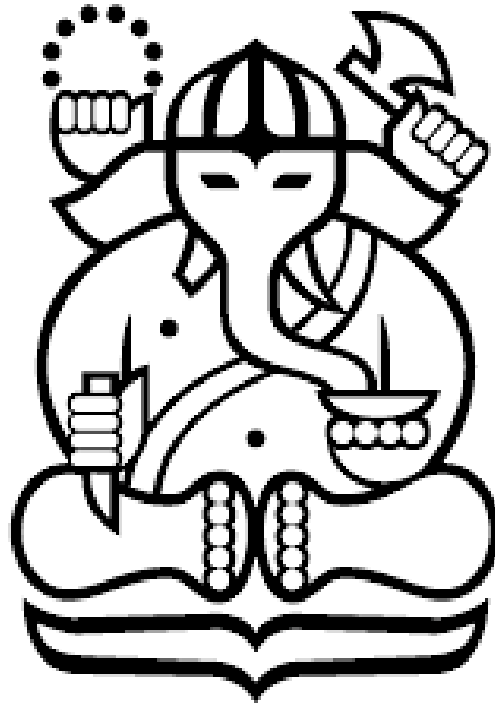


Implementasi KNN, Naive Bayes, DTL pada UNSW-NB15 Dataset

IF3170 - Inteligensi Artifisial



Nama Kelompok:

1. Ignatius Jhon Hezkiel Chan (13522029)
2. Matthew Vladimir Hutabarat (13522093)
3. Suthasoma Mahardhika Munthe (13522098)
4. Marvin Scifo Y. Hutahaeen (13522110)

**PRODI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA (STEI)
INSTITUT TEKNOLOGI BANDUNG
2024**

Daftar Isi

Daftar Isi.....	2
KNN (K Nearest Neighbor).....	3
Naive Bayes.....	4
DTL (Decision Tree Learning).....	4
Data Cleaning and Preprocessing.....	6
1. Data Cleaning.....	6
2. Data Preprocessing.....	8
Prediction Comparison.....	12
1. KNN.....	12
2. Naive Bayes.....	12
3. ID3.....	13
Lampiran dan Pembagian Tugas.....	16

KNN (K Nearest Neighbor)

Algoritma k-nearest neighbours (KNN) adalah pengklasifikasi pembelajaran terawasi non-parametrik, yang menggunakan kedekatan untuk membuat klasifikasi atau prediksi tentang pengelompokan titik data individual. Ini adalah salah satu pengklasifikasi dan pengklasifikasi regresi yang populer dan paling sederhana yang digunakan dalam pembelajaran mesin saat ini. Beginilah tahapan dari implementasi KNN dalam menyelesaikan dataset UNSW-NB15

1. Constructor KNN

KNN dibuat dengan 2 parameter yaitu k-nearest dan orde dalam Minkowski Distance. Data train sengaja tidak dimasukkan nilai apa-apa karena akan diisikan saat melakukan Model Fitting

2. Model Fitting

Model Fitting menyimpan data training untuk melakukan prediksi nanti

3. Prediksi

Prediksi dilakukan dengan cara konversi array menjadi numpy array. Untuk setiap data yang ingin diprediksi, cari Minkowski distance lalu cari fitur terdekat sebanyak k. Lalu, carilah nilai kelas yang paling dominan dari k neighbor tersebut untuk ditentukan sebagai nilai prediksi

Naive Bayes

Naive Bayes adalah metode pengklasifikasian dengan mengandalkan Teorema Bayes. Metode ini akan menghitung probabilitas suatu kejadian berdasarkan informasi sebelumnya. Teorema Bayes dapat dinyatakan dengan rumus :

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

$P(A|B)$: Probabilitas A terjadi jika diketahui B terjadi

$P(B|A)$: Probabilitas B terjadi jika diketahui A terjadi

$P(A)$: Probabilitas awal dari A

$P(B)$: Probabilitas awal dari B

Berikut langkah langkah dalam mengimplementasikan kelas NaiveBayes:

1. Melakukan fit pada kolom target, pada langkah ini kelas akan menyimpan nilai nilai unik dari kolom target dan menghitung setiap rata rata dan varian setiap kolom yang ada.
2. Selanjutnya, setiap baris dari dataset keseluruhan (kecuali target kolom) akan diiterasi untuk dihitung probabilitas terhadap nilai kolom target menggunakan distribusi gaussian dengan rumus sebagai berikut :

$$P(X|C) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right)$$

μ : Rata rata dari class C

σ^2 : Varian dari class C

Pada dataset ini kami mengubah semua kolom kategori menjadi numerik dengan menggunakan *Onehot Encoding* sehingga tidak perlu menggunakan rumus probabilitas yang diajarkan di kelas

3. Setelah menghitung nilai distribusi gaussian setiap kelas, kita akan memasukkan nilai dengan probabilitas terbesar ke dalam array prediction

DTL (Decision Tree Learning)

ID3 (Iterative Dichotomiser 3) adalah algoritma pembentukan *decision tree* yang digunakan untuk melakukan klasifikasi. ID3 memilih atribut terbaik untuk memisahkan data menggunakan metrik Information Gain yang dihitung dari Entropy. Atribut dengan *information gain* tertinggi akan digunakan sebagai *node* atau *decision point* dalam pohon. Berikut ini adalah tahapan yang dilakukan untuk mengimplementasikan kelas ID3DecisionTree.

1. Entropy

Method ini adalah sebuah fungsi yang digunakan untuk menghitung entropi dari sebuah label y. Entropi digunakan untuk mengukur ketidakaturan dalam dalam

dataset dan dirumuskan sebagai:

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

Nilai entropi ini akan digunakan untuk menentukan fitur mana yang akan ditetapkan untuk membagi data.

2. **Information Gain**

Nilai entropi yang berkurang setelah membagi data menggunakan fitur tersebut dapat diperoleh menggunakan rumus berikut:

$$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t)$$

Fitur dengan *information gain* tertinggi dipilih untuk membagi data. Untuk menentukan sebuah fitur sebagai pembagi data, ada nilai gain minimum yang ditetapkan yaitu, 0.0001. Nilai ini digunakan untuk menghentikan pembagian data yang tidak signifikan. Nilai gain ini berfungsi untuk menghindari pembuatan model yang terlalu kompleks dan pemisahan data itu tidak membawa keuntungan yang berpengaruh. Selain itu, nilai minimum gain ini bertujuan untuk menghindari masalah *overfitting* pada model.

3. **Build Tree**

Tree secara rekursif dibangun dengan menghitung nilai *information gain* yang terbesar untuk setiap fitur yang masih tersisa. Jika tree sudah mencapai kedalaman tertentu maka prediksi untuk node ini adalah nilai label mayoritas untuk data pada node ini, yaitu *attack_cat*.

Jika data suatu node seluruhnya sama, maka nilai prediksi pada node tersebut adalah nilai target data. Jika tidak ada fitur yang dapat digunakan untuk membagi data, tetapkan nilai prediksi sebagai nilai mayoritas.

4. **Predict**

Prediksi dilakukan dengan melakukan iterasi *decision tree* yang telah dibentuk saat melakukan *fitting* menggunakan data latih. Beberapa kasus di mana nilai prediksi tidak ada saat sudah mencapai sebuah node, nilai prediksi yang diganti menjadi nilai mayoritas (*leaves*) pada tree dengan akar node tersebut.

Data Cleaning and Preprocessing

Untuk membuat prediksi pada dataset, ada baiknya data dibersihkan dan diproses terlebih dahulu. Berikut adalah metode yang kami lakukan untuk tiap tahapan Data Cleaning dan Preprocessing.

1. Data Cleaning

a. Handling Missing Data

Mengidentifikasi dan melakukan sesuatu terhadap nilai yang hilang di dataset. Strategi yang kami gunakan adalah untuk nilai kolom numeric, kami menggunakan strategi imputer dengan menggantikan nilai *missing* dengan median kolom tersebut pada dataset. Sedangkan untuk nilai kolom kategorikal, kami menggunakan strategi imputer dengan menggantikan nilai missing dengan *most frequent value* kolom tersebut pada dataset.

```
1 class FeatureImputer(BaseEstimator, TransformerMixin):
2     def __init__(self, strategy='mean', fill_value=None):
3         """
4         Initialize the imputer for handling missing values.
5
6         :param strategy: The strategy to use for imputation ('mean', 'median', 'most_frequent', 'constant').
7                         Default is 'mean'.
8         :param fill_value: The value to use for the 'constant' strategy. Default is None.
9         """
10        self.strategy = strategy
11        self.fill_value = fill_value
12        self.imputer = SimpleImputer(strategy=self.strategy, fill_value=self.fill_value)
13
14    def fit(self, X):
15        """
16        Fit the imputer to the data.
17
18        :param X: Features data with missing values
19        """
20        self.imputer.fit(X)
21
22    def transform(self, X):
23        """
24        Transform the data by imputing the missing values.
25
26        :param X: Features data with missing values
27        :return: Data with missing values imputed
28        """
29        return self.imputer.transform(X)
30
31    def fit_transform(self, X, y=None):
32        """
33        Fit the imputer and transform the data.
34
35        :param X: Features data with missing values
36        :return: Data with missing values imputed
37        """
38        return self.imputer.fit_transform(X)
39
40    def get_imputation_statistics(self):
41        """
42        Get the imputation statistics (e.g., mean or median values used for imputation).
43
44        :return: The statistics used for imputation (depending on the strategy)
45        """
46        return self.imputer.statistics_
```

b. Dealing with Outliers

Untuk menangani outliers, kami melakukan *Clipping* (pembatasan nilai) berdasarkan persentil yang kami tentukan. Kami membuat batas bawah persentil adalah persentil ke-1, dan batas atas adalah persentil ke-99. Nilai yang berada diluar range ini akan dipotong (*clipped*) menjadi nilai batas tersebut. Metode ini dilakukan untuk tiap kolom dengan nilai persentil tiap kolom terkait.

```
1 class OutlierClipper(BaseEstimator, TransformerMixin):
2     def __init__(self, lower_percentile=0.01, upper_percentile=0.99):
3         self.lower_percentile = lower_percentile
4         self.upper_percentile = upper_percentile
5
6     def fit(self, X, y=None):
7         if not isinstance(X, np.ndarray):
8             X = np.array(X)
9         self.lower_bounds = np.percentile(X, self.lower_percentile * 100, axis=0)
10        self.upper_bounds = np.percentile(X, self.upper_percentile * 100, axis=0)
11        return self
12
13    def transform(self, X):
14        if not isinstance(X, np.ndarray):
15            X = np.array(X)
16        return np.clip(X, self.lower_bounds, self.upper_bounds)
```

c. Removing Duplicates:

Kami melakukan penghapusan baris duplikat pada dataset. Penghapusan baris duplikat ini agar menjaga integritas data, sehingga mendapatkan *insight* dan analisis yang tepat dan sesuai.

```
1 class DuplicateRemover(BaseEstimator, TransformerMixin):
2     def fit(self, X, y=None):
3         return self
4
5     def transform(self, X):
6         return X
7
8     def fit_transform(self, X, y):
9         self.fit(X,y)
10        X_unique, indices = np.unique(X[0], axis=0, return_index=True)
11        y_unique = X[1][indices]
12        return X_unique, y_unique
```

d. Feature Engineering:

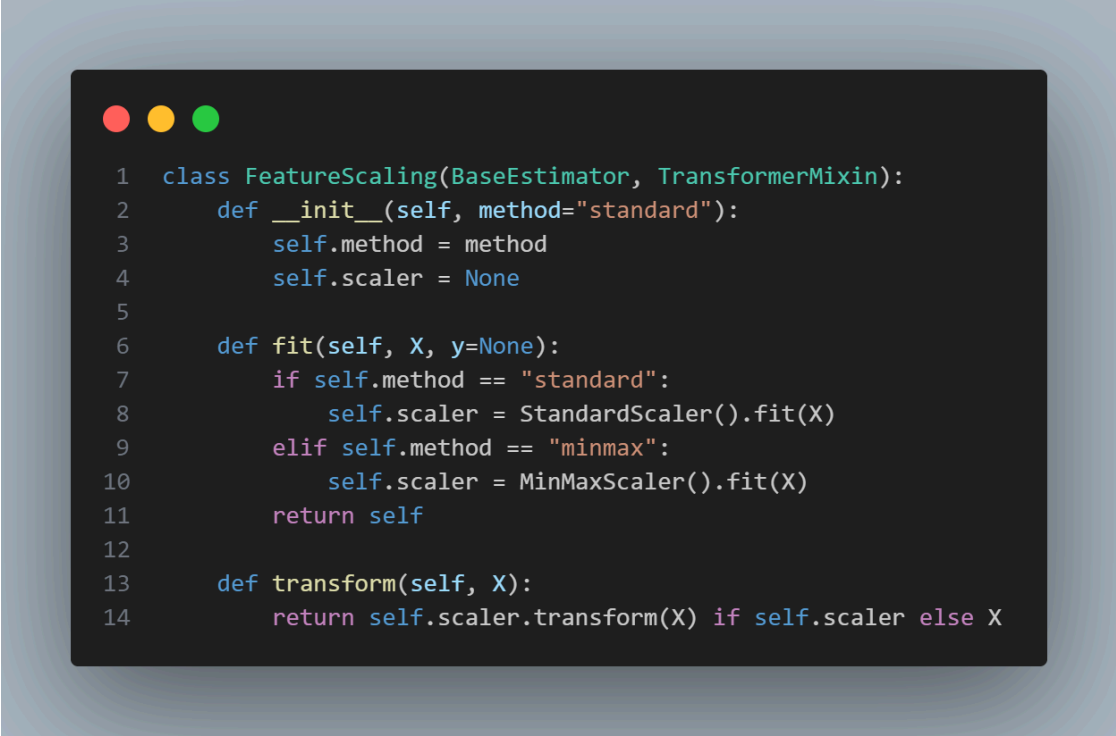
Kami melakukan proses *Discretization* untuk membagi nilai numeric menjadi beberapa kategori. Proses ini kami lakukan khususnya untuk model ID3, dimana ID3 pada dasarnya membagi jenis *keputusan* berdasarkan suatu kategori sehingga nilai numeric harus dapat dibagi menjadi beberapa kategori diskrit.

```
1 class FeatureDiscretizer(BaseEstimator, TransformerMixin):
2     def __init__(self, features, bins=10, strategy='uniform'):
3         self.features = features
4         self.bins = bins
5         self.strategy = strategy
6         self.discretizers = {}
7
8     def fit(self, X, y=None):
9         X_df = pd.DataFrame(X, columns=self.features)
10
11         for feature in self.features:
12             if self.strategy == 'uniform':
13                 discretizer = np.linspace(X_df[feature].min(), X_df[feature].max(), self.bins + 1)
14             elif self.strategy == 'quantile':
15                 discretizer = np.quantile(X_df[feature], np.linspace(0, 1, self.bins + 1))
16             else:
17                 raise ValueError(f"Invalid strategy: {self.strategy}")
18
19             self.discretizers[feature] = discretizer
20
21         return self
22
23     def transform(self, X):
24         X_df = pd.DataFrame(X, columns=self.features)
25
26         for feature in self.features:
27             discretizer = self.discretizers[feature]
28             X_df[feature] = pd.cut(X_df[feature], bins=discretizer, labels=False, include_lowest=True, duplicates='drop')
29
30         return X_df.values
```

2. Data Preprocessing

a. Feature Scaling

Feature scaling digunakan untuk memastikan fitur numerik memiliki skala yang serupa. Kami menggunakan Standard Scalling untuk menormalisasi data sehingga skala fitur dataset memiliki rata-rata (mean) 0 dan simpangan baku (*standard deviation*) 1. Kami memilih scaling ini untuk memastikan distribusi fitur lebih seragam dan membantu model bekerja lebih baik pada data skala berbeda. Penggunaan scaling ini aman akan outlier karena outlier telah kami tangani pada tahap sebelumnya.



```

1  class FeatureScaling(BaseEstimator, TransformerMixin):
2      def __init__(self, method="standard"):
3          self.method = method
4          self.scaler = None
5
6      def fit(self, X, y=None):
7          if self.method == "standard":
8              self.scaler = StandardScaler().fit(X)
9          elif self.method == "minmax":
10             self.scaler = MinMaxScaler().fit(X)
11         return self
12
13     def transform(self, X):
14         return self.scaler.transform(X) if self.scaler else X

```

b. Encoding Categorical Variables

Model pembelajaran mesin biasanya bekerja dengan data numerik, sehingga variabel kategori perlu dikodekan. Kami mengimplementasikan dua jenis Encoding, yakni One hot Encoding dan Ordinal Encoding. Ordinal Encoding kami gunakan pada Pipeline ID3 dan Naive Bayes. Sedangkan One Hot Encoding kami gunakan pada Pipeline KNN.

c. Handling Imbalanced Classes

Untuk mengatasi ketidakseimbangan Kelas Attack Category pada dataset, kami menggunakan teknik oversampling yakni SMOTE. SMOTE bekerja dengan menambah data dari kelas minoritas dengan membuat sampel-sampel sintetik berdasarkan statistik data.

```

1 class SMOTEHandler(BaseEstimator, TransformerMixin):
2     def __init__(self, random_state=None, sampling_strategy='auto'):
3         self.random_state = random_state
4         self.sampling_strategy = sampling_strategy
5         self.smote = SMOTE(random_state=self.random_state, sampling_strategy=self.sampling_strategy)
6
7     def fit(self, X, y):
8         self.smote.fit(X, y)
9         return self
10
11    def transform(self, X):
12        return X
13
14    def fit_transform(self, X, y):
15        return self.smote.fit_resample(X, y)

```

d. Dimensionality Reduction

Kami menggunakan Dimensionality Reduction, yakni bentuk pengurangan jumlah fitur menggunakan teknik Analisis Komponen Utama (PCA) untuk menyederhanakan model dan meningkatkan performa. PCA bekerja dengan menggabungkan beberapa fitur menjadi suatu “*principal component*” yang berpotensi mewakili beberapa kolom lebih baik.

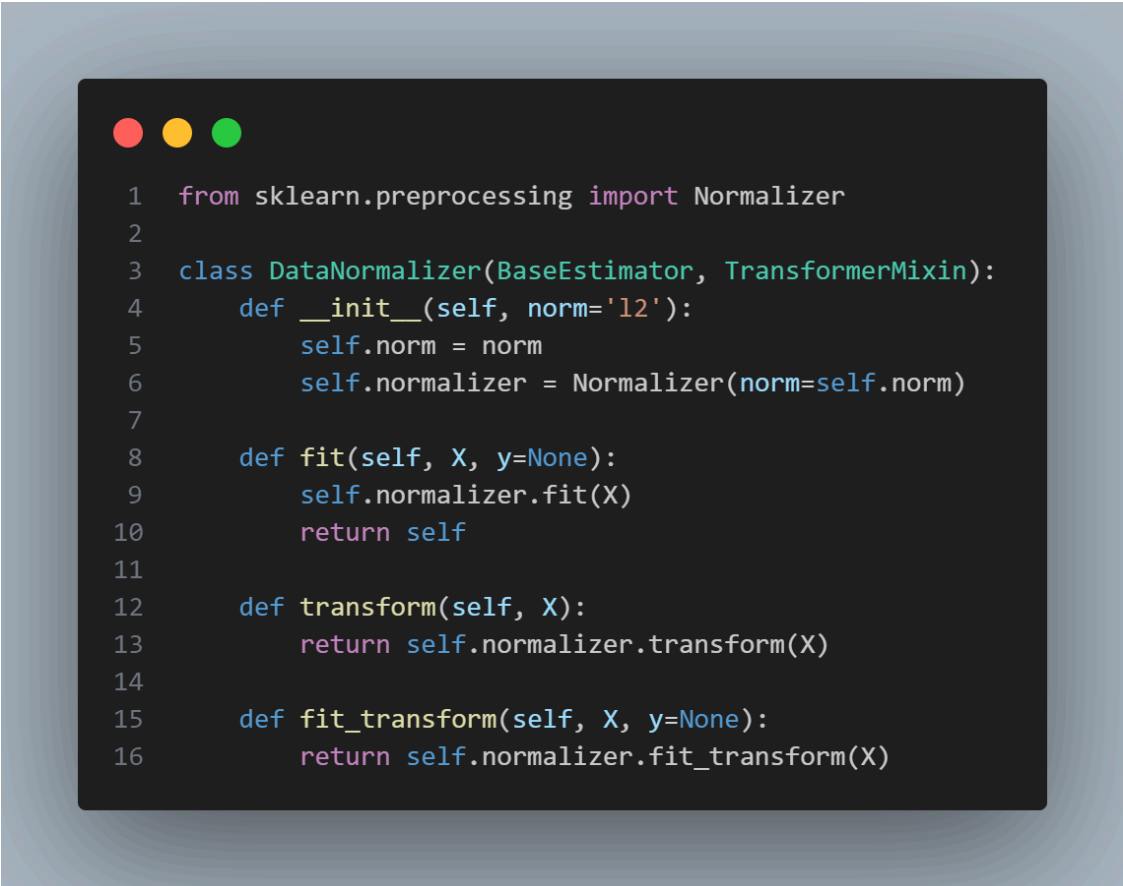
```

1 class DimensionalityReducer(BaseEstimator, TransformerMixin):
2     def __init__(self, n_components=None):
3         self.n_components = n_components
4         self.pca = PCA(n_components=self.n_components)
5
6     def fit(self, X, y=None):
7         self.pca.fit(X)
8         return self
9
10    def transform(self, X):
11        return self.pca.transform(X)
12
13    def fit_transform(self, X, y=None):
14        return self.pca.fit_transform(X[0])
15
16    def explained_variance_ratio(self):
17        return self.pca.explained_variance_ratio_
18
19    def components(self):
20        return self.pca.components_

```

e. Normalization

Bentuk penormalan data untuk mencapai distribusi standar. Hal ini sangat penting untuk algoritma yang mengasumsikan data terdistribusi normal.



```
1  from sklearn.preprocessing import Normalizer
2
3  class DataNormalizer(BaseEstimator, TransformerMixin):
4      def __init__(self, norm='l2'):
5          self.norm = norm
6          self.normalizer = Normalizer(norm=self.norm)
7
8      def fit(self, X, y=None):
9          self.normalizer.fit(X)
10         return self
11
12     def transform(self, X):
13         return self.normalizer.transform(X)
14
15     def fit_transform(self, X, y=None):
16         return self.normalizer.fit_transform(X)
```

Prediction Comparison

1. KNN

Berdasarkan hasil percobaan, didapatkan hasil sebagai berikut :

1. Model KNN dari Library Scikit Learn

```
Accuracy: 0.5379
Confusion Matrix:
[[ 164 128  30  19   6   1   4  11   1  20]
 [  91 123  39   9  16   0   0  16  13  60]
 [ 756 795 219  71  51   1   4 112  78 372]
 [2173 1299 783 469 136   2  21 217 248 1288]
 [ 396 551 159  53 1477  22  84 190 240 465]
 [  46  25  13   9  69 7741   9  14  12  37]
 [ 459 448 486  80 902  14 8380 148 126 211]
 [ 139 219  55  43  74   1   9 208 390 932]
 [   6  17   6   3  19   0   2  27  64 115]
 [   1   1   1   2   1   0   0   3   2 17]]
Classification Report:
              precision    recall  f1-score   support

     0       0.04       0.43       0.07       384
     1       0.03       0.34       0.06       367
     2       0.12       0.09       0.10      2459
     3       0.62       0.07       0.13      6636
     4       0.54       0.41       0.46      3637
     5       0.99       0.97       0.98       7975
     6       0.98       0.74       0.85     11254
     7       0.22       0.10       0.14      2070
     8       0.05       0.25       0.09       259
     9       0.00       0.61       0.01        28

 accuracy          0.54      35069
 macro avg         0.36       0.29      35069
 weighted avg      0.74       0.58      35069
```

2. Model KNN Scratch

```
Accuracy: 0.5630
Confusion Matrix:
[[ 252  99   6   2   2   0   0   4   0  19]
 [ 101 175  10   4   0   0   0   3   7  67]
 [ 854 717 322  30  41   0   0  58  45 392]
 [1722 1825 729 567 129   2  17 161 166 1318]
 [ 443 525 136  50 1646   6  57 185 208 381]
 [  38  36  10  10  79 7742   4  12   8  36]
 [ 508 369 486  73 824  12 8563 116 104 199]
 [ 147 168  50  13  36   0   1 331 389 935]
 [   5  14   1   0   0   0   0   7 118 114]
 [   0   0   0   0   0   0   0   0   0  28]]
Classification Report:
              precision    recall  f1-score   support

     0       0.06       0.66       0.11       384
     1       0.04       0.48       0.08       367
     2       0.18       0.13       0.15      2459
     3       0.76       0.09       0.15      6636
     4       0.60       0.45       0.51      3637
     5       1.00       0.97       0.98       7975
     6       0.99       0.76       0.86     11254
     7       0.38       0.16       0.22      2070
     8       0.11       0.46       0.18       259
     9       0.01       1.00       0.02        28

 accuracy          0.56      35069
 macro avg         0.41       0.51       0.33      35069
 weighted avg      0.79       0.56       0.61      35069
```

2. Naive Bayes

Berdasarkan hasil percobaan, didapatkan hasil sebagai berikut :

1. Model Naive Bayes dari library Scikit Learn

```
Accuracy: 0.3049
Precision Score: 0.7103
Confusion Matrix:
[[ 305  0  0  0  0  1  0  0  78  0]
 [ 216  0  0  0  0  1  0  0 150  0]
 [1500  0  0  3  0 13 24  7 901 11]
 [4192  0  0 22  4 13 41  4 2358  2]
 [1006  0  0  3 151 154 21 133 2118 51]
 [  78  0  0  0 15 7750 53  2  75  2]
 [6972  1  1  1 151 223 2204 65 1614 22]
 [ 247  0  0  0  1  1  4  2 1790 25]
 [  0  0  0  0  2  0  0  0  257  0]
 [  1  0  0  0  0  0  0  0  27  0]]
Classification Report:
              precision    recall  f1-score   support

     0       0.02       0.79       0.04       384
     1       0.00       0.00       0.00       367
     2       0.00       0.00       0.00      2459
     3       0.76       0.00       0.01      6636
     4       0.47       0.04       0.08      3637
     5       0.95       0.97       0.96      7975
     6       0.94       0.20       0.32     11254
     7       0.01       0.00       0.00      2070
     8       0.03       0.99       0.05       259
     9       0.00       0.00       0.00        28

 accuracy          0.30     0.30     0.30     35069
 macro avg         0.32     0.30     0.15     35069
 weighted avg      0.71     0.30     0.33     35069
```

2. Model Naive Bayes scratch

```
Accuracy: 0.3067
Precision Score: 0.7083
Confusion Matrix:
[[ 303  0  0  0  0  2  0  0  78  1]
 [ 216  0  0  0  0  1  0  0 150  0]
 [1490  0  0  3  0 18 24  9 900 15]
 [4152  0  0 29  5 18 41  4 2357 30]
 [ 965  0  0  4 153 156 21 141 2116 81]
 [  78  0  0  0 15 7750 53  2  75  2]
 [6895  0  1  2 172 224 2260 66 1596 38]
 [ 231  0  0  0  2  4  4  2 1786 41]
 [  0  0  0  0  2  0  0  0  257  0]
 [  0  0  0  0  0  0  0  0  27  1]]
Classification Report:
              precision    recall  f1-score   support

     0       0.02       0.79       0.04       384
     1       0.00       0.00       0.00       367
     2       0.00       0.00       0.00      2459
     3       0.76       0.00       0.01      6636
     4       0.44       0.04       0.08      3637
     5       0.95       0.97       0.96      7975
     6       0.94       0.20       0.33     11254
     7       0.01       0.00       0.00      2070
     8       0.03       0.99       0.05       259
     9       0.00       0.04       0.01        28

 accuracy          0.31     0.30     0.31     35069
 macro avg         0.32     0.30     0.15     35069
 weighted avg      0.71     0.31     0.34     35069
```

Dapat dilihat dari akurasi diatas, hasil implementasi naive bayes buatan sendiri dan built in naive bayes memiliki performa yang hampir sama. Performa yang hampir sama ini menunjukkan kalau model yang ada pada scikit learn memiliki bentuk yang sama dengan model yang bangun secara *scratch*.

3. ID3

Kedalaman yang ditetapkan untuk kedua percobaan pada model yang diimplementasikan dengan model yang menggunakan pustaka adalah 5. Hasil akurasi yang diperoleh menggunakan algoritma implementasi sendiri adalah 0.7346 dan yang

menggunakan pustaka adalah 0.7354. Nilai yang didapat menggunakan implementasi lebih buruk.

Kedua model menggunakan data hasil *pipeline* (*cleaning* dan *preprocess*) yang sama. Fitur numerik di transform menggunakan metode discretization. Sementara fitur kategori di-encode menggunakan *label encoder*.

Berikut ini adalah hasil uji coba pada kedua model menggunakan data yang sama.

```
Accuracy: 0.7354
Confusion Matrix:
[[ 49  18  99 165  12   3  27  10   0   1]
 [ 19  21 100 176  23   2   8  14   4   0]
 [112  98 775 1210  87  29  50  80  16   2]
 [150 143 1107 4330 298  77 196 303  18  14]
 [ 19  40  144  431 1744  20  824 361  53   1]
 [   3   1   43   82   26 7803   7   10   0   0]
 [ 31   5   57  196  923  17 9840 158  27   0]
 [   5  20  127  372  239   7   98 1190  10   2]
 [   0   3   13   23   68   1  34   81  36   0]
 [   0   0   1   13   2   0   1   8   1   2]]
Classification Report:
              precision    recall  f1-score   support

     0       0.13         0.13         0.13         384
     1       0.06         0.06         0.06         367
     2       0.31         0.32         0.31        2459
     3       0.62         0.65         0.64        6636
     4       0.51         0.48         0.49        3637
     5       0.98         0.98         0.98        7975
     6       0.89         0.87         0.88       11254
     7       0.54         0.57         0.56        2070
     8       0.22         0.14         0.17         259
     9       0.09         0.07         0.08          28
    ...
 accuracy                   0.74        35069
 macro avg                   0.43         0.43        0.43        35069
 weighted avg                 0.74         0.74         0.74        35069
```

Gambar 3.1 Hasil model menggunakan pustaka

```

Accuracy: 0.7346
Confusion Matrix:
[[ 27   7  72 257   6   1  10   4   0   0]
 [  6   5  65 226  28   1  23  13   0   0]
 [ 21  35 431 1644 118  17 113  76   4   0]
 [ 45  41 575 5101 340  23 360 146   5   0]
 [ 10   8  57 384 1932  22 638 572  14   0]
 [  1   0   8 102  40 7764  17  43   0   0]
 [  3   0   0 269 1175  11 9381 412   3   0]
 [  4   6  62 527 296   2  75 1098   0   0]
 [  0   0   0   0  78   0  70  90  21   0]
 [  0   0   0  24   2   0   1   1   0   0]]
Classification Report:
              precision    recall  f1-score   support

     0           0.23       0.07       0.11         384
     1           0.05       0.01       0.02         367
     2           0.34       0.18       0.23       2459
     3           0.60       0.77       0.67       6636
     4           0.48       0.53       0.50       3637
     5           0.99       0.97       0.98       7975
     6           0.88       0.83       0.86      11254
     7           0.45       0.53       0.49       2070
     8           0.45       0.08       0.14         259
     9           0.00       0.00       0.00          28
...
 accuracy                   0.73       35069
  macro avg                 0.45       0.40       0.40       35069
 weighted avg               0.73       0.73       0.72       35069

```

Gambar 3.2 Hasil model implementasi

Lampiran dan Pembagian Tugas

Link GitHub: [sotul04/gimana-sih-ben](https://github.com/sotul04/gimana-sih-ben)

Pembagian Tugas:

Nama	NIM	Tugas
Ignatius Jhon Hezekiel Chan	13522029	Data Cleaning and Preprocessing, Model Library
Matthew Vladimir Hutabarat	13522093	Naive Bayes
Suthasoma Mahardhika Munthe	13522098	Decision Tree Learning
Marvin Scifo Y. Hutahaeen	13522110	K-Nearest Neighbor