

# **Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial**

## ***Implementasi Algoritma Pembelajaran Mesin***

**Disusun oleh :**


### **Kelompok 18**

Arvyno Pranata Limahardja	18222007
David Dewanto	18222027
Bastian Natanael Sibarani	18222053
Dedy Hofmanindo Saragih	18222085

**Program Studi Sistem dan Teknologi Informasi**

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

**Jl. Ganesha 10, Bandung 40132**

	<b>Program Studi</b>	Jumlah Halaman
	<b>Sistem dan Teknologi Informasi  STEI – ITB</b>	<b>22</b>

---

## Daftar Isi

<b>BAB I - Implementasi KNN dan Naive Bayes.....</b>	<b>2</b>
A. Implementasi K-Nearest Neighbors (KNN).....	2
B. Implementasi Naive Bayes.....	5
<b>BAB II - Preprocessing dan Cleaning Data.....</b>	<b>10</b>
<b>BAB III - Perbandingan Scratch dan Library Referensi.....</b>	<b>18</b>
A. Perbandingan Implementasi Scratch dan Scikit K-Nearest Neighbors (KNN).....	18
B. Perbandingan Implementasi Scratch dan Scikit pada Naive Bayes.....	19
<b>BAB 4 - Pembagian Tugas tiap Anggota Kelompok.....</b>	<b>20</b>
<b>Referensi.....</b>	<b>21</b>

## Pendahuluan

Laporan ini dibuat untuk menjelaskan lebih detail mengenai model pembelajaran mesin yang sudah kami buat untuk memenuhi Tugas Besar II Dasar Inteligensi Artifisial - Implementasi Algoritma Pembelajaran Mesin. Berikut merupakan detail dari anggota kelompok kami.

Kelompok 18

1. Arvyno Pranata Limahardja / 18222007
2. David Dewanto / 18222027
3. Bastian Natanael Sibarani / 18222053
4. Dedy Hofmanindo Saragih / 18222085

Berikut juga kami lampirkan link Github untuk mengakses *notebook* yang berisi model pembelajaran yang sudah kami buat.

Link Github : <https://github.com/david-dewanto/Tugas-DAI-2/tree/main>

## BAB I

### Implementasi KNN dan Naive Bayes

#### A. Implementasi K-Nearest Neighbors (KNN)

Algoritma K-Nearest Neighbors (KNN) merupakan model pembelajaran mesin yang bekerja dengan cara mengklasifikasikan sampel berdasarkan jarak ke sejumlah  $k$  tetangga terdekat. Proses utamanya melibatkan perhitungan jarak antara sampel data uji ( $X_{\text{test}}$ ) dan seluruh data pelatihan ( $X_{\text{train}}$ ), kemudian menentukan kelas berdasarkan mayoritas label tetangga terdekat atau dengan pemberian bobot berbasis jarak. Jarak dihitung menggunakan 3 *distance metrics*, yaitu Euclidean, Manhattan, atau Minkowski yang masing-masing memiliki kegunaan tergantung pada karakteristik dataset.

Adapun rumus pada tiap *distance metrics* yang kami gunakan adalah sebagai berikut.

1. *Euclidean Distance*: Rumus ini digunakan untuk mengukur jarak lurus geometris antara dua titik dalam ruang multidimensi.

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Metrik ini sensitif terhadap fitur dengan skala besar, sehingga lebih cocok jika data telah dinormalisasi.

2. *Manhattan Distance*: Rumus ini digunakan untuk menghitung jarak sebagai jumlah perbedaan absolut antar dimensi.

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

Metrik ini lebih tahan terhadap *outlier* karena tidak melibatkan operasi kuadrat.

3. *Minkowski Distance*: Rumus ini merupakan generalisasi dari euclidean dan manhattan.

$$d(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

Dengan  $p = 2$ , Minkowski menjadi Euclidean, dengan  $p = 1$ , Minkowski menjadi Manhattan. Parameter  $p$  dapat disesuaikan untuk menentukan sensitivitas terhadap variasi antar dimensi.

4. *Hamming Distance*: Rumus ini digunakan untuk data biner atau kategorikal, dan menghitung jumlah nilai yang berbeda antara dua vektor.

$$d(x, y) = \frac{\text{Jumlah perbedaan antar fitur}}{\text{Jumlah total fitur}}$$

Sekarang, kami akan menjelaskan bagaimana implementasi algoritma dari K Nearest Neighbor yang kami implementasi dari *scratch*. Kami menggunakan referensi yang digunakan oleh *library* Scikit yang berasal dari Buku “Introduction to Information Retrieval”. Berikut merupakan *pseudocode* yang diambil langsung dari buku tersebut.

```

Train-KNN (C,D)
1 D' ← Preprocess(D)
2 k ← Select-K(C,D')
3 return D',k

Apply-KNN (C,D',k,d)
1 Sk ← ComputeNearestNeighbors(D',k,d)
2 for each cj ∈ C
3 do p ← |Sk ∩ cj|/k
4 return arg maxj pj

```

Kami menerapkan *pseudocode* tersebut menggunakan bahasa Python dengan bantuan *library* numpy untuk memudahkan perhitungan.

```

1 class KNeighborsClassifierScratch:
2     def __init__(self, n_neighbors=5, metric='euclidean', p=2, weights='uniform'):
3         self.n_neighbors = n_neighbors
4         self.metric = metric
5         self.p = p
6         self.weights = weights
7
8     def fit(self, X_num, X_cat, y):
9         self.X_num_train = X_num
10        self.X_cat_train = X_cat
11        self.y_train = y
12        return self
13
14    def _calculate_numerical_distances(self, x_test):
15        if self.metric == 'euclidean':
16            return np.sqrt(np.sum((self.X_num_train - x_test) ** 2, axis=1))
17        elif self.metric == 'manhattan':
18            return np.sum(np.abs(self.X_num_train - x_test), axis=1)
19        elif self.metric == 'minkowski':
20            return np.sum(np.abs(self.X_num_train - x_test) ** self.p, axis=1) ** (1 / self.p)

```

```

21     else:
22         raise ValueError("Unsupported metric for numerical features")
23
24     def _calculate_categorical_distances(self, x_test_cat):
25         # Check if categorical training or test data is None
26         if self.X_cat_train is None or x_test_cat is None or self.X_cat_train.shape[1] == 0:
27             return np.zeros(self.X_num_train.shape[0]) # Return zero distances if no
categorical data
28
29         # Calculate Hamming distance for categorical features
30         return np.sum(self.X_cat_train != x_test_cat, axis=1) / self.X_cat_train.shape[1]
31
32     def _predict_single(self, x_test_num, x_test_cat):
33         num_distances = self._calculate_numerical_distances(x_test_num)
34         cat_distances = self._calculate_categorical_distances(x_test_cat)
35         distances = num_distances + cat_distances
36
37         neighbors_indices = np.argsort(distances)[:self.n_neighbors]
38         neighbors_labels = self.y_train[neighbors_indices]
39
40         if self.weights == 'uniform':
41             return np.argmax(np.bincount(neighbors_labels))
42         elif self.weights == 'distance':
43             weights = 1 / (distances[neighbors_indices] + 1e-5) # Avoid division by zero
44             weighted_votes = np.bincount(neighbors_labels, weights=weights)
45             return np.argmax(weighted_votes)
46         else:
47             raise ValueError("Unsupported weights")
48
49     def predict(self, X_num, X_cat=None):
50         if X_cat is None: # Handle case with no categorical data
51             return np.array([self._predict_single(x_num, None) for x_num in X_num])
52         else:
53             return np.array([self._predict_single(x_num, x_cat) for x_num, x_cat in
zip(X_num, X_cat)])
54
55     def score(self, X_num, X_cat, y):
56         y_pred = self.predict(X_num, X_cat)
57         return np.mean(y_pred == y)

```

Dalam algoritma KNN, bobot menentukan seberapa besar pengaruh tetangga terhadap prediksi.

1. *Uniform Weight*, setiap tetangga memiliki bobot yang sama, terlepas dari jaraknya.

$$w_i = 1$$

2. *Distance Weight*, tetangga yang lebih dekat memiliki bobot lebih besar dibandingkan yang lebih jauh.

$$w_i = \frac{1}{d_i + \epsilon}$$

Ket:

$d_i$ : Jarak ke tetangga ke- $i$

$\epsilon$ : Nilai kecil untuk menghindari pembagian dengan nol

Demikian implementasi dari K Nearest Neighbor yang kami gunakan pada tugas besar ini.

## B. Implementasi Naive Bayes

Algoritma Klasifikasi (*classifier*) Naive Bayes merupakan model pembelajaran mesin yang menggunakan teorema statistik Bayes untuk melakukan klasifikasi terhadap suatu data tertentu. Algoritma ini masuk ke dalam kategori *Supervised Machine Learning*, di mana model ini membutuhkan data latih dengan *label* yang sudah diberikan. Dikatakan “Naive” karena model ini menganggap bahwa satu *feature* (kolom) independen atau tidak berkaitan dengan *feature* (kolom lainnya) pada sebuah data. Kemudian, seperti yang sudah dijelaskan sebelumnya algoritma ini menggunakan teorema statistik, lebih spesifiknya teori probabilitas.

Model Naive Bayes sendiri memiliki beberapa variasi, seperti Gaussian Naive Bayes, Multinomial Naive Bayes, Complement Naive Bayes, Bernoulli Naive Bayes, dan Categorical Naive Bayes. Setiap variasi ini memiliki kelebihan dan kekurangannya masing-masing, serta mempunyai syarat masukan yang berbeda-beda. Gaussian Naive Bayes hanya bisa menerima data numerik. Categorical Naive Bayes hanya bisa menerima data kategorikal. Multinomial Naive Bayes dan Complement Naive Bayes hanya bisa menerima data diskrit non-negatif. Terakhir, Bernoulli Naive Bayes hanya bisa menerima data biner.

Pada akhirnya, kami memilih untuk menggunakan model Bernoulli Naive Bayes untuk beberapa alasan di bawah ini yang sebagian besar kami ambil dari tahap EDA (*Exploratory Data Analysis*) yang sudah pernah dilakukan.

- Alasan Memilih Bernoulli Naive Bayes - Terdapat beberapa *categorical features* dengan korelasi Cramer's V yang tinggi dengan target variabel ( $>0.2$ ). Di mana, sebagian besar *categorical features* tersebut memiliki merupakan data biner (kecuali TLD). Selain itu,

kami juga melihat menggunakan grafik proporsi kelas target berdasarkan *categorical features* yang menunjukkan adanya korelasi antara fitur kategorik dengan target label. Maka dari itu, kami melihat bahwa model Bernoulli Naive Bayes cocok untuk digunakan pada permasalahan ini karena Bernoulli Naive Bayes mempunyai kelebihan dalam melakukan klasifikasi data biner.

- Alasan Tidak Memilih Gaussian Naive Bayes - Kami tidak menerapkan variasi Gaussian Naive Bayes menggunakan data numerik yang asli karena data numerik tersebut memiliki distribusi yang tidak normal (dari uji Anderson-Darling yang kami lakukan), di mana variasi Gaussian Naive Bayes mengambil asumsi bahwa data numerik yang menjadi masukan memiliki distribusi yang normal. Selain itu, korelasi antara data numerik dan target juga tidak optimal (uji korelasi Pearson dan Spearman tidak ada yang melebihi  $|R| > 0.5$ ). Maka dari itu, kami tidak memilih variasi Gaussian Naive Bayes karena tidak sesuai dengan data yang kami miliki.
- Alasan Tidak Memilih Categorical Naive Bayes - Hanya terdapat satu *features* kategorik dengan *unique values* lebih dari 2, yaitu TLD. Selebihnya, *features* kategorik bersifat biner sehingga lebih cocok menggunakan variasi Bernoulli Naive Bayes yang memang fokus pada data biner.
- Alasan Tidak Memilih Multinomial Naive Bayes - Terdapat variasi lain yang terbukti lebih baik, yaitu Complement Naive Bayes.
- Alasan Tidak Memilih Complement Naive Bayes - Complement Naive Bayes ditargetkan pada data diskrit non-negatif, di mana hal ini tidak tersedia pada data yang kita miliki (sebagian besar data kategorik yang ada adalah data biner).

Sekarang, kami akan menjelaskan bagaimana implementasi algoritma dari Bernoulli Naive Bayes yang kami implementasi dari *scratch*. Kami menggunakan referensi yang digunakan oleh *library* Scikit yang berasal dari Buku “Introduction to Information Retrieval”. Berikut merupakan *pseudocode* yang diambil langsung dari buku tersebut.

```
TrainBernoulliNB(C, D)
1 V ← ExtractVocabulary(D)
2 N ← CountDocs(D)
3 for each  $c \in C$ 
4 do  $N_c \leftarrow \text{CountDocsInClass}(D, c)$ 
5    $\text{prior}[c] \leftarrow N_c / N$ 
6   for each  $t \in V$ 
7   do  $N_{ct} \leftarrow \text{CountDocsInClassContainingTerm}(D, c, t)$ 
```



```

8   condprob[t][c] ← (Nct + 1) / (Nc + 2)
9   return V, prior, condprob

APPLYBERNOULLINB(C, V, prior, condprob, d)
1  Vd ← ExtractTermsFromDoc(V, d)
2  for each c ∈ C
3  do score[c] ← log prior[c]
4    for each t ∈ V
5    do if t ∈ Vd
6      then score[c] += log condprob[t][c]
7      else score[c] += log(1 − condprob[t][c])
8  return arg maxc ∈ C score[c]

```

Berikut juga merupakan rumus formal dari penerapan Bernoulli Naive Bayes untuk menghitung probabilitas suatu kelas.

$$P(y | x) = \frac{P(y) \prod_{i=1}^n P(x_i | y)^{x_i} (1 - P(x_i | y))^{1-x_i}}{P(x)}$$

Keterangan :

1.  $P(y | x)$  adalah probabilitas awal kelas  $y$  apabila diberikan *feature* biner  $x = x_1, x_2, \dots$
2.  $P(y)$  adalah probabilitas awal kelas  $y$
3.  $P(x_i | y)$  adalah probabilitas *feature*  $x_i = 1$  apabila diberikan kelas  $y$
4.  $P(x)$  adalah probabilitas marginal dari *feature* biner  $x = x_1, x_2, \dots$

Kami menerapkan *pseudocode* tersebut menggunakan bahasa Python dengan bantuan *library* numpy untuk memudahkan perhitungan.

```

1 class BernoulliNB_Scratch(BaseEstimator, ClassifierMixin):
2     def __init__(self, alpha=1.0): # alpha for Laplace smoothing
3         self.alpha = alpha
4         self.classes_ = None
5         self.class_priors_ = None
6         self.feature_probs_ = None
7
8     def fit(self, X, y):
9         n_samples, n_features = X.shape
10        self.classes_ = np.unique(y)
11        n_classes = len(self.classes_)
12
13        # Calculate class priors (prior[c])
14        self.class_priors_ = np.zeros(n_classes)
15        for i, c in enumerate(self.classes_):

```

```

16     self.class_priors_[i] = np.sum(y == c) / n_samples
17
18     # Calculate conditional probabilities for each feature
19     # condprob[t][c] = (Nct + 1)/(Nc + 2)
20     self.feature_probs_ = np.zeros((n_classes, n_features))
21     for i, c in enumerate(self.classes_):
22         X_c = X[y == c]
23         Nc = X_c.shape[0]
24
25         # Add Laplace smoothing (alpha)
26         Nct = X_c.sum(axis=0) + self.alpha
27         self.feature_probs_[i] = Nct / (Nc + 2 * self.alpha)
28
29     return self
30
31 def predict_proba(self, X):
32     n_samples = X.shape[0]
33     nclasses = len(self.classes_)
34
35     # Calculate scores for each class
36     scores = np.zeros((n_samples, nclasses))
37
38     for i, c in enumerate(self.classes_):
39         # Start with log of class prior
40         scores[:, i] = np.log(self.class_priors_[i])
41
42         # Add log probabilities for each feature
43         # If feature present (1), add log(condprob)
44         # If feature absent (0), add log(1-condprob)
45         feature_probs = self.feature_probs_[i]
46         scores[:, i] += np.sum(
47             X * np.log(feature_probs) +
48             (1 - X) * np.log(1 - feature_probs),
49             axis=1
50         )
51
52     return scores
53
54 def predict(self, X):
55     return self.classes_[np.argmax(self.predict_proba(X), axis=1)]
56
57 def score(self, X, y):
58     y_pred = self.predict(X)
59     return accuracy_score(y, y_pred)

```

Hal menarik pertama yang ingin kami bahas lebih lanjut terkait implementasi di Python adalah penggunaan *Laplace Smoothing*. *Laplace Smoothing* ini digunakan pada

implementasi dunia nyata untuk menangani kasus probabilitas nol pada sebuah fitur yang dapat berdampak pada penilaian akhir nol (karena apapun dikali dengan nilai nol akan menghasilkan nol). Maka dari itu, diperkenalkan teknik *Laplace Smoothing* yang mencoba memodifikasi nilai probabilitas untuk mencegah nilai probabilitas nol. *Laplace Smoothing* memiliki parameter *alpha* yang menunjukkan seberapa besar *smoothing* yang ingin dilakukan dari data asli (semakin besar *alpha*, distribusi *feature* akan semakin mendekati distribusi *uniform*). Penerapan *Laplace Smoothing* dapat dilihat pada *pseudocode* bagian TrainBernoulli baris ke-8 (menggunakan asumsi *alpha* = 1). Kemudian, pada implementasi menggunakan Python dapat dilihat pada baris ke-27.

Kemudian, kami juga menggunakan bantuan fungsi logaritma untuk menghindari *underflow* (probabilitas akhir sangat kecil karena perkalian antara probabilitas setiap fitur yang menyebabkan program error karena angka yang terlalu kecil). Hal ini dapat dilihat pada bagian ApplyBernoulli baris ke 3, 6, dan 7. Kemudian, pada implementasi Python dapat dilihat pada baris ke 40 dan 46-49.

Demikian implementasi dari Bernoulli Naive Bayes yang kami gunakan pada tugas besar ini.

---

## BAB II

### ***Preprocessing dan Cleaning Data***

Penanganan data yang hilang dalam dataset dilakukan melalui beberapa langkah dan juga memakai strategi-strategi tertentu. Penanganan data yang hilang ini dilakukan dengan tujuan untuk melengkapi data tanpa mengubah distribusi aslinya secara signifikan. Langkah pertama adalah menganalisis data untuk mengidentifikasi kolom-kolom yang memiliki nilai yang kosong/hilang. Analisis ini mencakup perhitungan jumlah dan persentase data yang hilang untuk setiap kolom, pengelompokan kolom dilakukan berdasarkan tipe datanya, seperti numerik, kategorial, dan juga boolean. Hasil analisis ini memberikan gambaran menyeluruh mengenai distribusi data yang hilang dalam dataset dan membantu menentukan pendekatan yang paling tepat untuk mengatasi permasalahan tersebut.

Setelah memahami distribusi mengenai data yang hilang, langkah berikutnya merupakan memproses kolom boolean secara khusus. Untuk kolom bertipe data boolean, strategi imputasi ditentukan berdasarkan distribusi nilai *True* dan *False*. Jika nilai *True* mendominasi (lebih dari 75%), maka nilai hilang diisi dengan *True*. Sebaliknya, jika nilai *False* mendominasi maka data yang hilang akan diisi dengan nilai *False*. Jika distribusi nilai lebih simbang, modus (nilai yang sering muncul) digunakan untuk mengisi nilai yang hilang. Pendekatan ini memastikan bahwa distribusi nilai boolean tetap terjaga meskipun terdapat proses imputasi.

Kolom numerik diproses dengan cara yang lebih mendalam. Analisis distribusi dilakukan untuk menentukan apakah kolom memiliki distribusi yang simetris atau *skewed* (miring). Jika distribusinya simetris, nilai hilang akan diisi dengan rata-rata (mean), sedangkan jika distribusinya *skewed* (miring), nilai yang hilang akan diisi dengan median karena lebih tahan terhadap outliers. Selain itu, untuk kolom numerik yang memiliki korelasi tinggi dengan kolom lain, metode prediktif seperti *Iterative Imputation* digunakan untuk memprediksi nilai hilang berdasarkan fitur yang berkorelasi. Metode ini memastikan bahwa nilai hilang tidak diisi secara sembarangan, melainkan berdasarkan pola yang terdeteksi dalam dataset.

Sementara itu, kolom kategorial diproses menggunakan pendekatan yang berbeda. Modus digunakan untuk mengisi nilai yang hilang dalam kolom kategorial. Pendekatan ini menjaga distribusi kategori dalam dataset tetap konsisten dan menghindari bias yang

mungkin muncul dari penggunaan nilai acak atau strategi lain yang tidak mempertimbangkan distribusi data.

Dalam kasus dataset dengan fitur yang berasal dari URL, terdapat langkah tambahan untuk menangani data yang hilang. Data dengan informasi URL lengkap diproses untuk menghasilkan fitur turunan, seperti panjang URL, jumlah subdomain, status HTTPS, dan pola karakteristik tertentu. Fitur probabilitas berbasis karakter dan TLD juga dihitung menggunakan akurasi prediksi data phishing atau legitimate, terutama jika sebagian fitur hilang atau tidak tersedia. Jika ditemukan pola obfuscasi pada URL seperti encoding khusus, substitusi karakter, atau penggunaan Unicode, fitur tambahan seperti tingkat obfuscasi dihitung untuk memperkaya analisis.

Setelah fitur tambahan ini dihasilkan, data yang hilang pada fitur-fitur tersebut diisi menggunakan nilai yang diperoleh dari analisis data yang tersedia. Proses ini dilakukan secara batch untuk menghindari masalah memori, terutama pada dataset berukuran besar. Hasil pengisian ini kemudian divalidasi untuk memastikan bahwa proses berjalan sesuai rencana tanpa mengubah distribusi data secara signifikan.

Pendekatan ini menggabungkan metode-metode sederhana seperti imputasi mean, median, dan mode dengan teknik yang lebih kompleks seperti Iterative Imputation dan analisis berbasis fitur URL. Dengan demikian, proses ini tidak hanya menangani missing data secara efektif, namun juga menjaga integritas dan distribusi data dalam dataset. Hal ini memungkinkan dataset yang telah diproses untuk digunakan dalam analisis atau pemodelan dengan tingkat keakuratan yang lebih tinggi.

Penanganan baris yang terduplikasi dalam dataset adalah langkah penting untuk memastikan data yang digunakan dalam analisis bersifat unik dan tidak redundant. Dalam konteks dataset yang berbasis URL, langkah pertama adalah mengidentifikasi baris-baris yang memiliki duplikasi pada kolom URL. Untuk setiap URL yang muncul lebih dari sekali, data terkait dari semua baris tersebut harus digabungkan menjadi satu baris tunggal yang merepresentasikan informasi terbaik.

Setelah identifikasi duplikat, proses penggabungan dilakukan dengan memprioritaskan data dan dataset asli. Hal ini bertujuan untuk menjaga integritas data mentah yang biasanya lebih dipercaya daripada data yang telah dimodifikasi atau diimputasi. Namun, jika terdapat nilai yang kosong atau hilang dalam dataset asli, data dari dataset hasil imputasi

digunakan untuk melengkapi kekosongan tersebut. Pendekatan ini memastikan bahwa data yang hilang tidak diabaikan, tetapi diisi menggunakan sumber informasi yang tersedia.

Setelah baris-baris duplikat digabung, dataset terakhir dibentuk dengan menggabungkan baris hasil penggabungan dengan baris-baris yang tidak memiliki duplikasi. Baris non-duplikat dipertahankan tanpa perubahan untuk menjaga keutuhan data. Hasil akhirnya adalah dataset yang hanya memiliki satu baris untuk setiap URL, di mana data asli diprioritaskan, sementara nilai yang hilang dilengkapi dengan data hasil imputasi. Pendekatan ini memastikan bahwa dataset akhir bersifat konsisten, lengkap, dan siap digunakan untuk analisis atau pemodelan lebih lanjut.

Feature selection adalah proses penting dalam pengolahan data yang bertujuan untuk memilih fitur-fitur yang paling relevan dengan kolom target yang kita inginkan dalam sebuah dataset. Proses ini membantu meningkatkan performa model, mengurangi risiko overfitting, dan menyederhanakan kompleksitas perhitungan. Dalam pendekatan yang diterapkan, fitur numerik dan kategorikal dianalisis secara terpisah menggunakan berbagai metrik statistik, termasuk variansi, *mutual information* (MI), p-value, dan korelasi. Proses ini tidak hanya memastikan dataset lebih efisien namun juga mempertahankan informasi penting yang mendukung prediksi target.

Pada fitur numerik, langkah pertama adalah mengevaluasi variansi. Variansi mengukur seberapa tersebar nilai-nilai dalam suatu fitur dari rata-ratanya, yang dirumuskan sebagai :

$$Variance = 1/n \sum (x_i - \mu)^2$$

Fitur dengan variansi rendah dihapus karena cenderung tidak memberikan informasi yang signifikan untuk prediksi target. Selanjutnya, uji analisis ANOVA yang digunakan untuk mengevaluasi hubungan antara fitur dan target dengan membandingkan variasi antar kelompok (MS *Between*) dan dalam kelompok (MS *Within*). Statistik F dirumuskan sebagai :

$$F = \frac{MS \text{ Between}}{MS \text{ Within}}$$

Fitur dianggap signifikan jika nilai p-value dari uji ini lebih kecil dari ambang batas tertentu. Selain itu, *mutual information* (MI) digunakan untuk mengukur ketergantungan antara fitur dan target. Rumus MI adalah :

$$I(X; Y) = \sum \sum P(x, y) \log \frac{P(x, y)}{P(x)P(y)}$$

$P(x, y)$  adalah probabilitas bersama fitur X dan target Y. Fitur hanya dipertahankan jika memiliki nilai p-value yang signifikan dan MI yang cukup tinggi.

Fitur kategorikal dianalisis dengan pendekatan yang sedikit berbeda. Fitur dengan jumlah kategori yang terlalu banyak dihapus karena dapat meningkatkan kompleksitas model. Hubungan antara fitur kategorikal dan target diuji menggunakan *Chi-Square Test*. Statistik *Chi-Square* dihitung dengan rumus :

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

O adalah nilai yang diamati dan E adalah nilai yang diharapkan. MI juga digunakan untuk fitur kategorik dengan prinsip yang sama seperti pada fitur numerik. Hanya fitur kategorikan dengan prinsip yang sama seperti pada fitur numerik. Hanya fitur dengan p-value rendah dan MI yang memadai yang dipertahankan.

Setelah fitur numerik dan kategorikal terpilih, langkah terakhir adalah memeriksa korelasi antar fitur untuk menghindari redundansi. Untuk fitur numerik, *Spearman Rank Correlation* digunakan untuk mengukur hubungan monotonik antara dua variabel :

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

$d$  merupakan selisih peringkat antara dua fitur. Sementara itu, fitur kategorikal dievaluasi menggunakan *Cramer's*, yang dihitung sebagai :

$$V = \sqrt{\frac{\chi^2}{n * \min(k-1, r-1)}}$$

$n$  merupakan jumlah total sampel,  $k$  adalah jumlah kategori fitur pertama, dan  $r$  merupakan jumlah kategori fitur kedua. Jika ditentukan dua fitur dengan korelasi tinggi ( $\rho$  atau  $V > 0.8$ ), fitur dengan nilai MI yang lebih rendah dihapus untuk menjaga efisiensi model.

Hasil akhir dari proses ini mencakup daftar fitur terpilih yang paling relevan untuk prediksi, daftar fitur yang dihapus beserta alasan penghapusannya, serta skor statistik untuk setiap fitur. Dengan pendekatan ini, dataset akhir menjadi lebih efisien, tidak redundant, dan siap digunakan untuk analisis atau pelatihan model. Rumus-rumus yang digunakan memastikan proses seleksi berbasis pada dasar matematis yang kuat, sehingga meningkatkan keandalan hasil analisis dan model pembelajaran mesin.

Proses pengembangan fitur dilakukan untuk meningkatkan kemampuan dataset dalam merepresentasikan pola yang relevan dengan tujuan analisis. Dalam tahap ini, dibuat beberapa fitur baru yang dirancang untuk memperkaya informasi pada dataset. Fitur polinomial dikembangkan untuk menangkap pola non-linear, seperti menambahkan kuadrat skor kecocokan domain dan judul (*DomainTitleMatchScore*). Selain itu, akar kuadrat dari jumlah gambar juga dihitung untuk mereduksi pengaruh nilai ekstrim. Fitur interaksi, seperti interaksi antara panjang URL dan skor kecocokan domain-judul atau antara panjang baris terbesar dan jumlah referensi eksternal, ditambahkan untuk mengidentifikasi hubungan antar variabel yang tidak terlihat pada data asli. Kombinasi logis dari fitur boolean, seperti keberadaan jejaring sosial dan informasi pembayaran, juga dirancang untuk mengungkap pola unik yang dapat meningkatkan analisis lebih lanjut.

Setelah fitur baru ditambahkan, dilakukan evaluasi untuk menentukan fitur mana yang signifikan terhadap target yang dipelajari. Fitur numerik dievaluasi menggunakan analisis varians (ANOVA) untuk mengetahui pengaruhnya terhadap target. Sementara itu, fitur kategorikal diuji dengan uji *Chi-Square* untuk mengidentifikasi hubungan antara kategori dan target, *Mutual Information* digunakan untuk menangkap non-linear. Fitur yang menunjukkan hubungan signifikan dengan target dipertahankan dalam dataset untuk digunakan lebih lanjut.

Langkah selanjutnya yang digunakan adalah menghapus redundansi sebagai bentuk penghindaran pengaruh yang berlebihan dari fitur yang berkorelasi tinggi. Pasangan fitur yang memiliki korelasi kuat diidentifikasi, dan salah satu dari pasangan tersebut dihapus berdasarkan tingkat signifikansinya terhadap target. Proses ini memastikan bahwa dataset akhir hanya berisi fitur-fitur unik, relevan, dan memiliki pengaruh signifikan terhadap analisis yang akan dilakukan.

Dataset ini mencakup fitur-fitur yang telah diseleksi, termasuk fitur-fitur asli dan fitur baru yang memenuhi kriteria signifikan dan bebas dari redundansi. Dengan langkah ini,



dataset yang dihasilkan lebih efektif dalam merepresentasikan pola, sehingga mampu mendukung proses analisis dan prediksi secara optimal.

Proses binning dilakukan untuk mentransformasi data numerik menjadi kategori biner, yaitu 0 dan 1 untuk menyederhanakan analisis dan meningkatkan interpretasi. Dalam konteks ini binning digunakan untuk mendeteksi *outlier* dan mengklasifikasikan data sesuai dengan skema pelabelan dataset, di mana 0 menunjukkan phishing dan 1 menunjukkan tidak phishing. Proses deteksi outlier mengintegrasikan tiga metode utama, yaitu IQR, Z-Score, dan Modified Z-Score. Metode IQR digunakan untuk mengidentifikasi nilai yang berada di luar rentang normal kuartil pertama dan ketiga, sedangkan Z-Score yang menggunakan Median Absolute Deviation digunakan untuk menangani distribusi data yang tidak normal.

Untuk memastikan interpretasi yang konsisten, metadata digunakan untuk menjelaskan makna dari setiap kategori pada masing-masing kolom. Misalnya, untuk kolom 'URLLength', nilai 0 diartikan sebagai "Panjang URL mencurigakan", yang sering kali terkait dengan pola phishing, sedangkan nilai 1 diartikan sebagai "Panjang URL normal". Pada kolom 'NoOfJS', nilai 0 menunjukkan "Jumlah elemen JavaScript sedikit (normal)". Dengan cara ini, hasil binning tidak hanya mencerminkan deteksi outlier tetapi juga memberikan makna yang relevan dengan analisis phishing.

Pendekatan ini memberikan beberapa keuntungan utama. Kombinasi tiga metode deteksi outlier memastikan akurasi tinggi dengan menangkap pola distribusi data yang beragam. Selain itu, metadata yang disesuaikan dengan skema pelabelan membuat hasil binning lebih mudah diinterpretasikan dan relevan dengan tujuan analisis. Dengan data yang telah di-*binned* menjadi kategori biner, model prediktif dapat menggunakan fitur-fitur ini untuk meningkatkan akurasi dalam mendeteksi phishing. Proses ini memastikan bahwa nilai-nilai mencurigakan dapat diidentifikasi dengan tepat dan digunakan secara efektif dalam analisis tujuan.

Penanganan outlier merupakan langkah penting dalam memastikan kualitas dataset yang digunakan untuk analisis deteksi phishing URL. Pada dataset ini, outlier diidentifikasi dengan menganalisis distribusi data di setiap fitur menggunakan statistik deskriptif rata-rata, median, standar deviasi, dan skewness. Berdasarkan karakteristik distribusinya, digunakan tiga metode utama untuk mendeteksi *outlier*, yaitu Z-Score, IQR, dan Modified Z-Score. Z-Score digunakan untuk data yang mendekati distribusi normal dengan ambang batas plus

minus tiga standar deviasi. IQR diterapkan pada data dengan skewness moderat, dimana outlier diidentifikasi sebagai nilai di luar rentang kuartil pertama dan ketiga. Sementara itu, Modified Z-Score digunakan untuk distribusi yang sangat skewed atau tidak normal, menggunakan Median Absolute Deviation (MAD) untuk mendeteksi nilai yang jauh dari median.

Setelah outlier teridentifikasi, proses penanganan dilakukan dengan metode clipping untuk mempertahankan nilai-nilai dalam rentang wajar. Nilai ekstrim dibatasi sesuai dengan metode deteksi yang digunakan, misalnya pada distribusi normal dibatasi dalam rentang (mean plus minus  $3 \times \text{std}$ ), sedangkan pada distribusi IQR dan Modified Z-Score nilai dibatasi pada rentang Q1-Q3 atau median dengan toleransi tertentu. Selain itu, fitur-fitur khusus seperti Top-Level Domain dan string dianalisis lebih mendalam. Pola distribusi TLD dianalisis untuk mendeteksi tingkat phishing pada domain tertentu, dan TLD yang jarang digunakan namun memiliki tingkat phishing tinggi diberi flag mencurigakan (TLD\_suspicious). Untuk fitur string seperti URL atau domain, pola-pola mencurigakan seperti panjang yang ekstrem, jumlah karakter khusus yang tinggi, atau pola tertentu seperti karakter berulang atau encoded dianalisis dan ditandai dengan flag tambahan seperti suspicious\_length atau suspicious\_pattern.

Proses ini menghasilkan dataset yang lebih bersih dan representatif, dengan nilai ekstrim yang telah dikelola dan kolom flag tambahan yang memberikan wawasan mendalam terkait pola mencurigakan. Penanganan ini tidak hanya meningkatkan kualitas dataset, tetapi juga mendukung analisis lanjutan dan pengembangan model prediktif yang lebih andal dalam mendeteksi phishing URL. Dataset akhir yang dihasilkan memiliki konsistensi yang lebih baik dan mampu merepresentasikan karakteristik data secara optimal.

Dataset untuk mendeteksi phishing URL seringkali memiliki skala fitur yang berbeda, tipe data yang kompleks, dan distribusi kelas yang tidak seimbang. Oleh karena itu, dilakukan tiga proses utama yaitu feature scaling, feature encoding, dan handling imbalanced data untuk memastikan dataset lebih seragam, representatif, dan siap digunakan dalam pemodelan prediktif. Pada tahap feature scaling, fitur numerik diidentifikasi dan dikategorikan menjadi beberapa jenis, yaitu fitur numerik umum, fitur rasio (dengan nilai antara 0 dan 1), fitur dengan skewness tinggi, dan fitur boolean. Fitur boolean dipertahankan tanpa transformasi, sedangkan fitur lainnya diskalakan dengan metode yang relevan. Untuk fitur dengan distribusi normal digunakan StandardScaler, fitur rasio menggunakan MinMax Scaler, dan

fitur dengan skewness tinggi menggunakan Power Transformer. Proses ini menghasilkan dataset dengan nilai fitur yang lebih seragam, meningkatkan efisiensi model pembelajaran mesin.

Pada tahap encoding, fitur kategorikal dan boolean dikonversi menjadi bentuk numerik agar dapat dipahami oleh model. Fitur boolean seperti IsDomainIP dan HasSocialNet diubah menjadi nilai integer ( 0 atau 1). Transformasi ini memastikan bahwa semua fitur memiliki representasi yang konsisten dan numerik.

Tahap terakhir adalah handling imbalanced data, yang bertujuan untuk mengatasi distribusi kelas yang tidak seimbang. Beberapa metode digunakan, termasuk SMOTE untuk menambahkan data sintetis ke kelas minoritas, Random Undersampling untuk mengurangi jumlah data pada kelas mayoritas, dan SMOTEENN yang menggabungkan SMOTE dengan penghapusan outlier menggunakan teknik Edited Nearest Neighbors (ENN). Selain itu, bobot kelas dihitung berdasarkan distribusi asli untuk digunakan dalam pelatihan model. Hasilnya adalah dataset dengan distribusi asli untuk digunakan dalam pelatihan model. Hasilnya adalah dataset dengan distribusi kelas yang seimbang, yang disimpan dalam direktori `imbalance_handled_results`, bersama dengan statistik distribusi kelas untuk setiap metode.

Secara keseluruhan, proses ini memastikan bahwa dataset menjadi lebih seragam, numerik, dan representatif untuk mendukung pengembangan model prediktif yang akurat. Dataset yang telah diolah dapat digunakan untuk analisis lanjutan atau pelatihan model tanpa bias terhadap kelas mayoritas.

### BAB III

## Perbandingan *Scratch* dan *Library* Referensi

### A. Perbandingan Implementasi *Scratch* dan Scikit K-Nearest Neighbors (KNN)

Hasil perbandingan implementasi K-Nearest Neighbors (KNN) secara manual (*scratch*) dengan Scikit-learn menunjukkan performa yang hampir identik, baik dari segi akurasi maupun prediksi. Implementasi *scratch* menghasilkan akurasi sebesar 99.60%, sementara Scikit-learn sedikit lebih rendah dengan akurasi 99.59%. Pada prediksi sampel yang diuji, kedua metode memberikan hasil yang sama pada setiap sampel, menandakan bahwa logika dasar algoritma telah direplikasi dengan baik dalam implementasi *scratch*. Perhitungan jarak pada kedua metode menggunakan metrik Euclidean, dengan rumus:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Strategi pemilihan tetangga terdekat juga konsisten, yaitu memilih kelas dengan jumlah mayoritas dari kkk tetangga terdekat (*uniform weights*). Hal ini menjadi alasan mengapa prediksi antara implementasi *scratch* dan Scikit-learn sangat mirip.

Meski hasil prediksi hampir identik, ada perbedaan dalam efisiensi dan fitur. Implementasi Scikit-learn menggunakan struktur data yang dioptimalkan seperti KD-Trees atau Ball Trees untuk mempercepat pencarian tetangga terdekat, sedangkan implementasi *scratch* mengandalkan metode *brute-force* yang membutuhkan perhitungan jarak langsung untuk semua data latih. Perbedaan kecil dalam akurasi kemungkinan besar disebabkan oleh presisi internal dalam perhitungan jarak atau metode pengurutan.

Dari sisi kode, Scikit-learn menawarkan fleksibilitas lebih, termasuk validasi parameter, integrasi pipeline, dan fitur tambahan lainnya. Implementasi *scratch*, di sisi lain, lebih sederhana dan fokus pada logika dasar algoritma, sehingga cocok untuk pembelajaran dan eksperimen konsep KNN.

Secara keseluruhan, perbedaan akurasi yang sangat kecil (0.01%) menandakan bahwa kedua metode hampir setara dalam menangani dataset ini. Namun, implementasi Scikit-learn

lebih direkomendasikan untuk penggunaan pada dataset besar karena efisiensi dan fitur tambahannya, sementara implementasi *scratch* memberikan pemahaman yang lebih mendalam tentang algoritma KNN.

## **B. Perbandingan Implementasi *Scratch* dan Scikit pada Naive Bayes**

Perbandingan implementasi secara *scratch* dengan implementasi Bernoulli Naive Bayes dengan Scikit dapat dikatakan sama. Hal ini dikarenakan kami mengimplementasikan Bernoulli Naive Bayes menggunakan referensi yang digunakan oleh Scikit, seperti yang sudah kami jelaskan dalam implementasi algoritma Bernoulli Naive Bayes. Secara hasil, algoritma yang dibangun dengan implementasi *scratch* dan Scikit menghasilkan hasil yang sama (perbedaan 0.00% dari hasil kode *scratch* yang kami buat).

Namun, memang secara implementasi kode, terdapat beberapa perbedaan, terutama dalam eksekusi kode. Hal pertama yang paling signifikan adalah pada implementasi *scratch* tidak mengimplementasikan metode `partial_fit` yang dapat melakukan pembelajaran secara bertahap (jadi tidak melakukan proses *training* dalam satu kali eksekusi), tetapi data dapat dipisah-pisah menjadi beberapa bagian dan proses *training* dapat dilakukan secara bertahap (dengan tetap mengingat hal-hal yang sudah dipelajari sebelumnya). Hal ini tidak kami terapkan karena jumlah data yang kami gunakan tidak terlalu besar. Kemudian, struktur kode secara keseluruhan juga berbeda bila kita bandingkan dengan kode dokumentasi yang disediakan Scikit pada *repository* Github yang mereka punya. Di mana *library* Scikit menggunakan beberapa fungsi tambahan untuk mengecek apakah parameter yang dimasukkan sudah sesuai dengan yang seharusnya atau belum, sedangkan pada implementasi *scratch* tidak dilakukan karena diasumsikan bahwa data yang digunakan sudah sesuai dengan syarat yang diharuskan untuk bisa menjalankan Bernoulli Naive Bayes. Kemudian, terdapat sedikit perbedaan dengan cara perhitungan dengan *library* Scikit pada perhitungan probabilitas. *Library* Scikit menggunakan fungsi logaritma sampai pada akhir perhitungan, sebelum diubah kembali menggunakan fungsi eksponen. Pada implementasi *scratch*, dilakukan simplifikasi sehingga hanya menggunakan fungsi logaritma saja.

## **BAB 4**

### **Pembagian Tugas tiap Anggota Kelompok**

#### **Arvyno Pranata Limahardja - 18222007**

1. Mengerjakan Handle Outliers
2. Splitting Data
3. Membuat Pipeline
4. Membuat Feature Scaling

#### **David Dewanto - 18222027**

1. Implementasi, Pemilihan, Optimasi, dan Analisis Error Algoritma Naive Bayes
2. Membuat Laporan Naive Bayes (Implementasi dan Perbandingan Scikit Learn)
3. Submission Kaggle menggunakan Naive Bayes

#### **Bastian Natanael Sibarani - 18222053**

1. Membuat Feature Engineering
2. Membuat Laporan Mengenai Data Preprocessing
3. Membuat Fungsi Handle Duplicates
4. Membuat Fungsi untuk Handle Missing Values

#### **Dedy Hofmanindo Saragih - 18222085**

1. Membuat Algoritma KNN
2. Membuat Laporan Bagian KNN
3. Submission Kaggle menggunakan KNN

---

## Referensi

Cahya Alkahfi. (2024, April 25). *Algoritma K-Nearest Neighbors (KNN) dengan Python*

*Scikit-Learn* - SAINSDATA.ID. SAINSDATA.ID.

<https://sainsdata.id/machine-learning/273/machine-learning-k-nearest-neighbors-knn-dengan-python-scikit-learn/>

*Feature Selection In Machine Learning | Feature Selection Techniques With Examples |*

*Simplilearn*. (n.d.). [www.youtube.com](https://www.youtube.com). Retrieved September 24, 2022, from

[https://www.youtube.com/watch?v=5bHpPO6\\_OU4](https://www.youtube.com/watch?v=5bHpPO6_OU4)

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*.

Cambridge University Press.

manvendra7. (2024). *Skewness-and-kurtosis/SKEWNESS\_AND\_KURTOSIS\_(1).ipynb* at

*master · manvendra7/Skewness-and-kurtosis*. GitHub.

[https://github.com/manvendra7/Skewness-and-kurtosis/blob/master/SKEWNESS\\_AND\\_KURTOSIS\\_\(1\).ipynb](https://github.com/manvendra7/Skewness-and-kurtosis/blob/master/SKEWNESS_AND_KURTOSIS_(1).ipynb)

Nasib, S. K., Fadilah Istiqomah Pammus, None Nurwan, & La Ode Nashar. (2023).

COMPARISON OF FEATURE SELECTION BASED ON COMPUTATION TIME  
AND CLASSIFICATION ACCURACY USING SUPPORT VECTOR MACHINE.

*Indonesian Journal of Applied Research (IJAR)*, 4(1), 63–74.

<https://doi.org/10.30997/ijar.v4i1.252>

*Removing Outlier's from Data Using Modified Z-Score | Kaggle*. (2024). Kaggle.com.

<https://www.kaggle.com/discussions/general/382825>

Scikit-learn. (2019a). *1.9. Naive Bayes — scikit-learn 0.21.3 documentation*.

Scikit-Learn.org. [https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)

Scikit-learn. (2019b). *sklearn.naive\_bayes.BernoulliNB*. Scikit-Learn.

[https://scikit-learn.org/stable/modules/generated/sklearn.naive\\_bayes.BernoulliNB.html#sklearn.naive\\_bayes.BernoulliNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB)

scikit-learn. (2024). *scikit-learn/sklearn/utils/\_metadata\_requests.py* at

*d666202a9349893c1bd106cc9ee0ff0a807c7cf3* · *scikit-learn/scikit-learn*. GitHub.

[https://github.com/scikit-learn/scikit-learn/blob/d666202a9/sklearn/utils/\\_metadata\\_requests.py#L1251](https://github.com/scikit-learn/scikit-learn/blob/d666202a9/sklearn/utils/_metadata_requests.py#L1251)

Zach. (2021, April 5). *What is a Modified Z-Score? (Definition & Example)*. Statology.

<https://www.statology.org/modified-z-score/>