Perbandingan Algoritma Machine Learning Umum Berbasis TF-IDF untuk Klasifikasi Artikel Bahasa Indonesia

http://dx.doi.org/10.28932/jste.vXiX.X

Received: 26 Agustus 2025 | *Revised*: 27 September 2025 | *Accepted*: 29 September 2025

Andi Gunawan ✉#1, Hendra Bunyamin*2

Program Studi Teknik Informatika, Fakultas Teknologi dan Rekayasa Cerdas, Universitas Kristen Maranatha
Jl. Prof. drg. Surya Sumantri No.65, Bandung, Jawa Barat 40164, Indonesia
[1]if1972032@student.it.maranatha.edu
[2]hendra.bunyamin@it.maranatha.edu
✉Corresponding author: if1972032@student.it.maranatha.edu

How to cite this article:

A. Gunawan, H. Bunyamin, "Perbandingan Algoritma Machine Learning Umum Berbasis TF-IDF untuk Klasifikasi Artikel Bahasa Indonesia," *Journal of Smart Technology and Engineering*, vol. 1, no. 1, 01–XX. pp. 11-23, https://doi.org/10.28932/jts.vXXiXX.XXXX

Abstrak — **Penelitian ini membandingkan kinerja algoritma *machine learning* umum dalam klasifikasi artikel berita Indonesia. Dataset yang terdiri dari 2160 artikel dari Detik.com telah diproses sebelumnya dan diubah menjadi vektor fitur dengan menggunakan teknik *Term Frequency-Inverse Document Frequency* (TF-IDF). Algoritma yang diuji meliputi *Multinomial Naïve Bayes*, *Bernoulli Naïve Bayes*, *K-Nearest Neighbor*, *Random Forest*, dan *AdaBoost*. *Hyperparameter tuning* dilakukan menggunakan *5-fold cross-validation*, dan metrik evaluasi yang digunakan mencakup *accuracy*, *precision*, *recall*, dan F1-*score*. Hasil penelitian menunjukkan bahwa algoritma *Multinomial Naïve Bayes* dengan nilai alpha sebesar 0.1 menghasilkan performa terbaik secara keseluruhan dengan *accuracy* sebesar 0.8781, *precision* sebesar 0.8138, *recall* sebesar 0.8143, dan *F1-score* sebesar 0.814.**

Kata Kunci— ***Bernoulli Naïve Bayes, K-Nearest Neighbor, Machine Learning, Multinomial Naïve Bayes, News text classification***

Comparison of Common TF-IDF-Based Machine Learning Algorithms for Indonesian Article Classification

Abstract — ***This study compares the performance of common machine learning algorithms in the classification of Indonesian news articles. A Dataset of 2160 articles from Detik.com was pre-processed and transformed into feature vectors using the Term Frequency-Inverse Document Frequency (TF-IDF) technique. The algorithms tested were Multinomial Naïve Bayes, Bernoulli Naïve Bayes, K-Nearest Neighbor, Random Forest and AdaBoost. Hyperparameter tuning was conducted using 5-fold cross-validation, and evaluation metrics included accuracy, precision, recall, and F1-score. The results indicate that Multinomial Naïve Bayes, with alpha set to 0.1, achieved the best overall performance with an accuracy of 0.8781, precision of 0.8138, recall of 0.8143, and F1-score of 0.814.***

Keywords— ***Bernoulli Naïve Bayes, K-Nearest Neighbor, Machine Learning, Multinomial Naïve Bayes, News text classification*.**

I. PENDAHULUAN

Seiring bertambahnya informasi berupa artikel berita, proses penyaringan dan pengelompokan berita berdasarkan tipe menjadi semakin sulit dan menantang. Jika hal ini dibiarkan secara terus menerus maka berita yang relevan ataupun menarik akan semakin susah untuk ditemukan [1], [2]. Kondisi yang dinamakan *information overload* ini menuntut adanya solusi yang efektif untuk mengatasi permasalahan tersebut. Salah satu pendekatan yang dapat digunakan adalah teknik klasifikasi berbasis *machine learning*. Teknik klasifikasi berbasis *machine learning* adalah upaya untuk mengklasifikasikan tipe berita sesuai dengan konten yang ada di dalam artikel berita. Proses ini melibatkan analisis elemen penting seperti judul berita, isi berita, dll., kemudian memanfaatkan algoritma *machine learning* untuk melakukan prediksi kategori tipe berita. Penggunaan algoritma machine learning memungkinkan analisis yang lebih mendalam terhadap pola dan karakteristik berita, sehingga algoritma tersebut dapat menghasilkan prediksi kategori yang lebih akurat. Dari studi literatur yang sudah dilakukan [3]-[10], algoritma-algoritma klasifikasi artikel yang sering digunakan adalah *Multinomial Naive bayes*, *Bernoulli Naive bayes*, *K-Nearest Neighbors*, *Adaboost*, dan *Random Forest*. Oleh karena itu, penelitian ini bertujuan untuk membandingkan algoritma-algoritma tersebut dan menemukan algoritma yang paling efektif dalam klasifikasi artikel berita bahasa Indonesia.

II. KAJIAN TEORI

A. Studi Literatur

Penelitian mengenai klasifikasi artikel berita berbahasa Indonesia telah dilakukan sebelumnya dengan berbagai pendekatan. Devita, dkk. [3] membandingkan algoritma *Naive Bayes* dan *K-Nearest Neighbor* menggunakan *TF-IDF*, namun keterbatasan jumlah data menyebabkan risiko *overfitting*. Afrianto, dkk. [4] menggabungkan *FCM clustering* dengan *KNN* serta membandingkannya dengan *MLKNN*, dan memperoleh hasil terbaik pada *FCM-KNN*. Namun, hasil tersebut juga belum sepenuhnya stabil ketika nilai *parameter* diubah. Mahendra, dkk. [5] menggunakan *Random Forest* dengan *TF-IDF*, tetapi masih memerlukan data yang lebih beragam untuk hasil optimal. Arimbawa, dkk. [6] membandingkan *AdaBoost*, *C4.5*, dan *BPMLL*, dengan hasil terbaik pada *AdaBoost*, namun konsistensi perbandingan dan jumlah data masih menjadi kendala. Hidayat, dkk. [7], Irsan, dkk. [8], dan Patil, dkk. [9] juga mengeksplorasi performa *Naive Bayes* dengan berbagai jenis data dan metode ekstrasi fitur seperti *TF-IDF* dan *Word2Vec*.

Penelitian ini melanjutkan dan memperluas studi-studi terdahulu dengan membandingkan beberapa algoritma populer dalam klasifikasi teks, yaitu *Multinomial Naive Bayes*, *Bernoulli Naive Bayes*, *K-Nearest Neighbor*, *Random Forest*, dan *Adaboost*, menggunakan representasi fitur *TF-IDF* pada dataset artikel berita berbahasa Indonesia yang lebih besar dan beragam.

B. Algoritma Machine Learning

Algoritma *machine learning* merupakan kumpulan perintah yang disusun hingga dapat memungkinkan sistem mempelajari data dan mengindentifikasi pola tertentu secara otomatis, algoritma *machine learning* dapat digunakan untuk melakukan prediksi.

1) Multinomial Naive Bayes: Merupakan varian algoritma *Naive Bayes* yang digunakan untuk permasalahan klasifikasi teks berdasarkan frekuensi kemunculan kata pada dokumen, sehingga fitur yang berasal dari *TF-IDF* sangat cocok untuk algoritma ini [16].

2) Bernoulli Naive Bayes: Merupakan varian algoritma *Naive Bayes* yang digunakan untuk menyelesaikan permasalahan *binary/boolean* yaitu antara 0 atau 1, tidak atau iya, dan sebagainya [15].

3) K-Nearest Neighbor: Merupakan algoritma berbasis *instance* yang sederhana yang mencari data tetangga terdekat berdasarkan jarak dan biasanya digunakan untuk permasalahan klasifikasi dan *regresi* [4]. Algoritma ini dapat memprediksi label *output* berdasarkan mayoritas label dari data tetangga terdekat.

4) Random Forest: Merupakan algoritma yang menggabungkan beberapa *decision tree* sehingga tercipta *forest*. Algoritma ini dapat digunakan untuk menyelesaikan permasalahan klasifikasi.

5) AdaBoost: Merupakan algoritma yang berfokus pada proses *training* dengan cara memperbaiki nilai bobot prediksi dari iterasi sebelumnya. Cara kerja algoritma ini yaitu dengan memberikan bobot prediksi yang sama pada setiap sampel iterasi pertama lalu di lakukan *training* dan *prediksi*, untuk iterasi selanjutnya akan diberikan bobot yang lebih besar pada sampel yang salah pada prediksi sebelumnya, proses ini diulang hingga iterasi terakhir [6], [11].

C. Feature Extraction

Feature extraction merupakan bagian setelah dilakukan *preprocessing* data yang bertujuan untuk mengubah data menjadi representasi fitur yang dapat digunakan pada model *machine learning*. Metode *feature extraction* yang akan digunakan pada penelitian ini berupa *TF-IDF*

1) Term Frequency – Inverse Document Frequency: Merupakan metode yang menentukan fitur melalui pemberian bobot per kata berdasarkan kemunculan kata pada suatu dokumen dan frekuensi dokumen. Metode *TF-IDF* menggabungkan dua cara

penghitungan bobot kata, yaitu dengan menghitung frekuensi kemunculan kata di sebuah dokumen dan perhitungan *inverse* terhadap seluruh dokumen di dalam *database* yang mengandung kata tertentu [3].

D. Hyperparameter Tuning

Hyperparameter tuning merupakan proses untuk menentukan *hyperparameter* yang paling *optimal* pada algoritma *machine learning*. *Hyperparameter* adalah parameter dari sebuah model *machine learning* yang mempengaruhi proses *training*. Nilai dari sebuah *hyperparameter* ditentukan sebelumnya mulainya proses *training* model [17].

E. Confusion Matrix

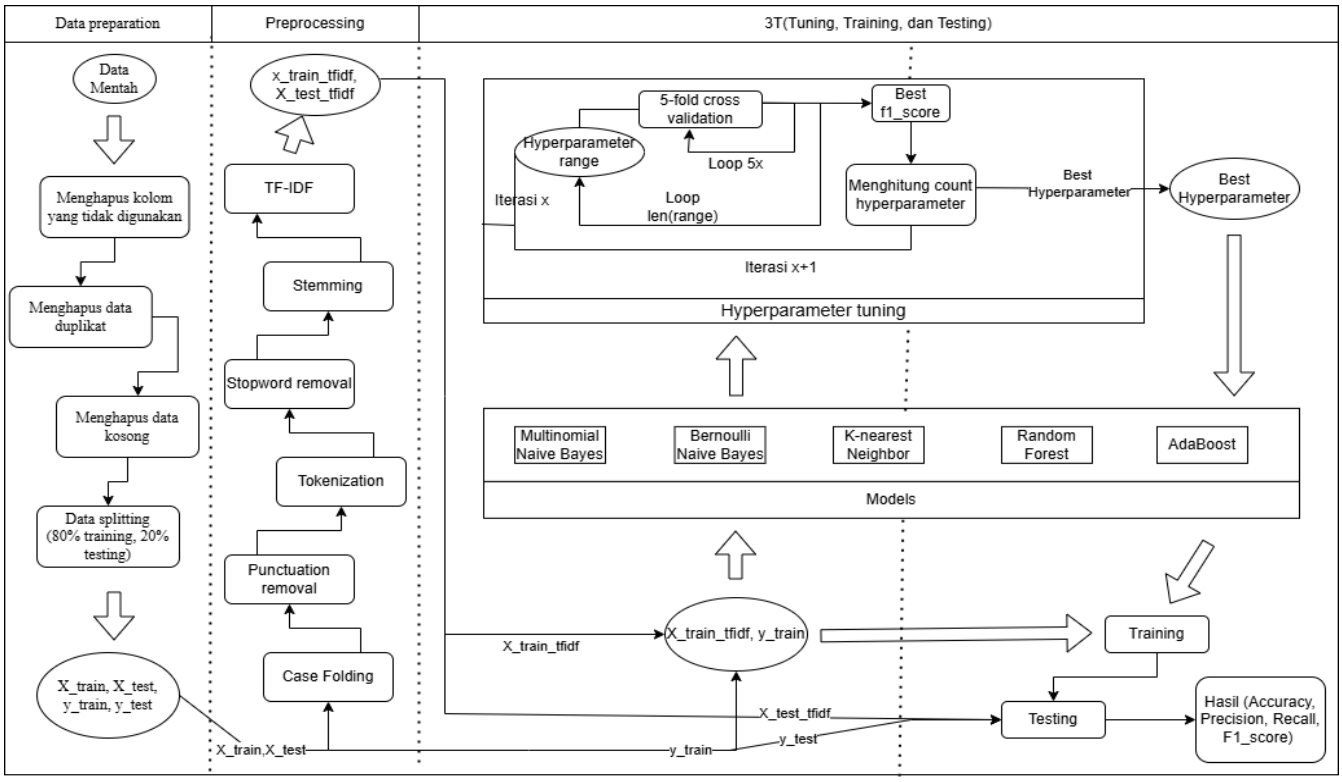
Confusion matrix merupakan metode evaluasi model yang dapat mengukur kinerja model dalam *machine learning*. Cara kerja dari metode ini yaitu dengan menghitung 4 elemen pengukuran utama yaitu *TP (True positive)* merupakan jumlah prediksi *positif* pada label *positif*, *TN (True negative)* merupakan jumlah prediksi *negatif* pada label *negatif*, *FP (False positive)* merupakan jumlah prediksi *positif* pada label *negatif*, dan *FN (False Negative)* merupakan jumlah prediksi *negatif* pada label *positif* [14].

F. K-fold Cross Validation

K-fold Cross Validation merupakan teknik evaluasi model yang membagi *dataset* menjadi beberapa *subset* yang disebut *fold*. Cara kerjanya yaitu dengan membagikan total data dengan jumlah *K-fold*, saat proses *training* iterasi pertama data dimulai, f*old* paling pertama akan digunakan sebagai data *testing*, sisanya akan digunakan sebagai data *training*, lalu pada iterasi kedua data *fold* kedua yang akan menjadi data *testing*, sedangkan data *fold* pertama akan berubah menjadi data *training*, dan dilanjutkan hingga iterasi ke K.

III. ANALISIS

Analisis pada penelitian ini mengikuti tahapan dalam proses *machine learning*, yaitu *data preparation*, *preprocessing data*, *feature extraction*, dan teknik evaluasi model seperti diagram alur pengerjaan pada Gambar 1.



Gambar 1. Diagram alur pengerjaan

A. Data Preparation

Data yang digunakan berupa 2160 data yang terdiri dari 5 kolom, yaitu *title*, *text*, *date*, *link*, dan *category*. Data kolom *category* berisikan 3 macam label yaitu olahraga sebagai label 0, politik sebagai label 1, dan teknologi sebagai label 2. Kolom data yang akan digunakan yaitu *text* dan *category*, lalu data akan diperiksa untuk memastikan tidak terdapat entri kosong maupun duplikat. Jika ditemukan, data tersebut akan dihapus. Selanjutnya, data dibagi menjadi dua bagian, yaitu 80% data *training* dan 20% data *testing*.

B. Preprocessing Data

Data *text* akan diproses melalui beberapa tahapan, yaitu *case folding*, *punctuation removal*, *stopwords removal*, *stemming*, dan *tokenization*. Proses ini bertujuan untuk membentuk struktur data yang terstruktur agar model *machine learning* dapat melakukan proses komputasi dengan mudah. Data *category* tidak perlu diproses karena akan digunakan sebagai label.

1) Case folding: Merupakan proses pengubahan semua huruf besar menjadi kecil

2) Punctuation Removal: Merupakan proses penghapusan semua simbol.

3) Stopword Removal: Merupakan proses penghapusan kata-kata dalam bahasa manusia yang tidak diolah oleh algoritma komputer [12].

4) Stemming: Merupakan proses perubahan kata-kata menjadi bentuk paling dasar.

5) Tokenization: Merupakan proses pemecahan kalimat menjadi bentuk kata per kata.

C. Feature Extraction

Feature extraction dilakukan dengan menggunakan metode *TF-IDF*. Setiap kata akan diberi bobot berdasarkan seberapa sering kata tersebut muncul dalam dokumen, dan seberapa jarang kata itu muncul di seluruh dokumen. Semakin spesifik sebuah kata terhadap dokumen tertentu, maka bobotnya akan semakin besar. Saat proses penghitungan nilai *IDF*, ada kemungkinan terjadinya pembagian dengan nol yang perlu dilakukan penyesuaian dengan menggunakan *smoothing* yaitu penambahan nilai 1 pada jumlah total dokumen dan jumlah kemunculan kata pada seluruh dokumen.

D. Hyperparameter Tuning

Hyperparameter tuning yang akan digunakan pada penelitian ini berupa *manual hyperparameter tuning* yaitu menggunakan struktur *looping* yang memuat proses *training* dan *evaluasi*, kemudian *variable* untuk menghitung *hyperparameter* terbaik dari setiap iterasi. Saat proses penghitungan dengan *variable* sudah selesai, *hyperparameter* dengan jumlah total terbanyak akan digunakan untuk proses selanjutnya yaitu *training* dan *testing*.

IV. IMPLEMENTASI

A. Data Preparation

Menggunakan *library pandas*, Kode Program 1 dapat membaca data dalam berkas *csv* dan menyimpan data kesebuah *variable* agar dapat diproses ketahap selanjutnya. Kode Program 2 akan melakukan penghapusan kolom data yang tidak digunakan pada penelitian ini.

```
import pandas as pd
df = pd.read_csv('/content/sample_data/detik.csv')
```

Kode Program 1 Membaca data dengan menggunakan *pandas*

```
df = df.drop(['Title', 'Date', 'Link'], axis=1)
```

Kode Program 2 Menghapus kolom yang tidak digunakan

Data selanjutnya akan dilakukan penghapusan data duplikat dan data kosong dengna menggunakan Kode Program 3. Tujuan dari penghapusan data duplikat maupun data kosong yaitu agar model tidak *bias* terhadap label tertentu dan tidak mengganggu akurasi model seakan data kosong atau duplikat itu penting. Jumlah data duplikat yang ditemukan dan dihapus yaitu 58 data. Jumlah total data berubah dari 2160 menjadi 2102.

```
df.drop_duplicates(subset="Text", keep='first', inplace=True)
df.dropna(subset=['Text'], inplace=True)
```

Kode Program 3 Menghapus data duplikat dan hilang

Sebelum dilakukan *preprocessing*, data perlu dipisah menjadi data *training (X_train, y_train)* dan data *testing (X_test, y_test)* dengan rasio 80% dan 20%. Kode program 4 akan memisah data dan mengubah nilai dari *parameter random_state* menjadi 99 agar hasil percobaan tetap konsisten (nilai awal adalah *None*).

```
from sklearn.model_selection import train_test_split
X = df['Text']
y = df['Category']
X_train, X_test, y_train, y_test = train_test_split(
   X, y,
   test_size=0.20,
   random_state=99
)
```

Kode Program 4 Proses *train test splitting*

B. Preprocessing Data

Data akan melalui 5 tahap *preprocessing* dimulai dengan melakukan *case folding* pada Kode Program 5 *str.lower()* akan mengubah semua huruf besar menjadi huruf kecil.

```
X_train = X_train.str.lower()
X_test = X_test.str.lower()
```

Kode Program 5 *Case folding*

Tahap kedua yaitu *punctuation removal*, Kode Program 6 baris pertama dan kedua akan menghapus semua simbol pada data sedangkan baris tiga dan empat akan menghapus angka.

```
X_train = X_train.str.replace(r'[^\w\s]+', '', regex=True)
X_test = X_test.str.replace(r'[^\w\s]+', '', regex=True)
X_train = X_train.str.replace(r'\d+', '', regex=True)
X_test = X_test.str.replace(r'\d+', '', regex=True)
```

Kode Program 6 *Punctuation removal*

Tahap ketiga yaitu *tokenization*, Kode Program 7 menggunakan *library nltk*, fungsi *word_tokenize*, dan model *punkt_tab* untuk memisahkan data teks menjadi bentuk kata perkata.

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt_tab')

X_train = X_train.apply(word_tokenize)
X_test = X_test.apply(word_tokenize)
```

Kode Program 7 *Tokenization*

Tahap keempat yaitu *stopword removal*, Kode Program 8 fungsi *stopwords.words("indonesian")* akan mengambil kumpulan *stopword* dalam bahasa Indonesia dan fungsi *stopword_removal(text)* akan mengembalikan data yang sudah bersih dari *stopwords*.

```
from nltk.corpus import stopwords
nltk.download('stopwords'
stop_words = set(stopwords.words('indonesian'))

def stopword_removal(text):
    filtered_words = [
        word for word in text if word not in
stop_words
    ]
    return filtered_words

X_train = X_train.apply(stopword_removal)
X_test = X_test.apply(stopword_removal)
```

Kode Program 8 *Stopword removal*

Tahap terakhir dari *preprocessing* data yaitu *stemming*, Kode Program 9 menggunakan *library sastrawi* untuk mengubah kata-kata ke bentuk paling sederhana. Proses ini akan memakan waktu kurang lebih 10 menit.

```
!pip install Sastrawi
from Sastrawi.Stemmer.StemmerFactory import
StemmerFactory
stemmer = StemmerFactory().create_stemmer()

def stemming(text):
    stemmed_words = [stemmer.stem(word) for word in text]
    return stemmed_words

X_train = X_train.apply(stemming)
X_test = X_test.apply(stemming)
```

Kode Program 9 *Stemming*

C. TF-IDF

Kode Program 10 menggunakan *library sklearn*, fungsi *TfidfVectorizer()* dapat digunakan untuk melakukan *feature extractio*n pada data yang sudah melewati tahap *preprocessing*. *Parameter analyzer=lambda x: x* berfungsi untuk memberi tanda bahwa data yang akan diproses sudah melewati tahap *tokenizer* dan *parameter smooth_idf=True* bertujuan untuk mencegah *zero division error* saat menghitung nilai *idf*. Seluruh kosakata yang sudah dipelajari oleh *TF-IDF* disimpan pada

variable vectorizer sedangkan nilai *TF-IDF* berada di *variable X_train_tfidf* dan *X_test_tfidf*.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(
                analyzer=lambda x: x, smooth_idf=True
            )
X_train_tfidf = vectorizer.fit_transform(X_train).toarray()
X_test_tfidf = vectorizer.transform(X_test).toarray()
```

Kode Program 10 *TF-IDF*

D. Confusion Matrix

Library sklearn, fungsi *confusion_matrix()* dapat menghasilkan prediksi model berupa *matrix* berukuran jumlah label x jumlah label. *Matrix* hasil prediksi model dapat digunakan untuk menghitung *TP (True Positive), FP (False Positive), FN (False Negative), TN (True Negative), Accuracy, Precision, Recall*, dan *F1_score*. Kode program 11 merupakan fungsi *ConfusionMatrix()* yang sudah dimodifikasi sesuai dengan kebutuhan uji coba.

```
from sklearn.metrics import confusion_matrix

def ConfusionMatrix(X, y, model):
  conf_matrix = confusion_matrix(y, model.predict(X))
  label_count = len(conf_matrix)
  accuracy = 0
  precision = 0
  recall = 0
  f1_score = 0

  for label in range(label_count):
    TP = conf_matrix[label, label]
    FP = conf_matrix[:, label].sum() - TP
    FN = conf_matrix[label, :].sum() - TP
    TN = conf_matrix.sum() - (TP + FP + FN)

    acc = 0 if (TP + TN + FP + FN) == 0 else (TP + TN) / (TP + TN + FP + FN)
    prec = 0 if (TP + FP) == 0 else TP / (TP + FP)
    rec = 0 if (TP + FN) == 0 else TP / (TP + FN)
    f1 = 0 if (prec + rec) == 0 else 2 * (prec * rec) / (prec + rec)

    accuracy += acc
    precision += prec
    recall += rec
    f1_score += f1

  accuracy = accuracy/label_count
  precision = precision/label_count
  recall = recall/label_count
  f1_score = f1_score/label_count

  return accuracy, precision, recall, f1_score
```

Kode Program 11 Fungsi *Confusion Matrix*

E. K-fold Cross Validation (5-fold)

Library sklearn, fungsi *Kfold* dapat melakukan proses *cross validation* secara otomatis dan mengembalikan metrik evaluasi yang hanya berupa *accuracy*. Dikarenakan kebutuhan metrik evaluasi berupa *accuracy, prevision, recall*, dan *f1_score*, fungsi *Kfold* perlu sedikit modifikasi. Fungsi *Kfold()* pada Kode Program 12 dapat mengembalikan keempat metrik evaluasi yang sudah dirata-ratakan setelah selesai proses *cross validation*.

```
from sklearn.model_selection import KFold

k = 5
kf = KFold(n_splits=k, shuffle=True)

def Kfold(X, y, model):
  accuracy = 0
  precision = 0
  recall = 0
  f1_score = 0
```

```
  for train_index, validation_index in kf.split(X, y):
    X_trains, X_validation = X[train_index], X[validation_index]
    y_trains, y_validation = y.iloc[train_index], y.iloc[validation_index]

    model.fit(X_trains, y_trains)
    a, p, r, f = ConfusionMatrix(X_validation, y_validation, model)
    accuracy, precision, recall, f1_score = accuracy + a, precision + p, recall + r,
f1_score + f

  accuracy = accuracy/k
  precision = precision/k
  recall = recall/k
  f1_score = f1_score/k

  return accuracy, precision, recall, f1_score
```

Kode Program 12 Fungsi *K-Fold Cross Validation*

V. UJI COBA DAN HASIL

A. Hyperparameter Tuning

Jumlah iterasi yang akan digunakan pada seluruh model *machine learning* yaitu 75, 100, dan 125 (15, 20, dan 25 untuk *AdaBoost*). Tabel I merupakan *hyperparameter* dan *hyperparameter range* yang akan digunakan.

TABEL I
HYPERPARAMETER DAN *HYPERPARAMETER RANGE*

Model	***Hyperparameter***	***Hyperparameter range***
Multinomial Naive Bayes	*alpha*	[0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
Bernoulli Naive Bayes	*alpha*	[0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
K-Nearest Neighbor	*n_neighbours*	[1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
Random Forest	*n_estimators*	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
AdaBoost	*n_estimators*, *learning_rates*	[50, 60, 70, 80 ,90], [0.01, 0.1, 1.0, 10.0]

Tabel II merupakan jumlah iterasi yang diuji, *hyperparameter* terbaik dari *hyperparameter tuning* pada setiap modelnya beserta waktu yang digunakan selama proses *hyperparameter tuning* berlangsung.

TABEL II
HYPERPARAMETER TERBAIK DAN WAKTU *HYPERPARAMETER TUNING*

Model	**Jumlah *iterasi***	***Best Hyperparameter***	**Waktu *Hyperparameter tuning***
Multinomial Naive Bayes	75	*alpha* = 0.1	± *3* menit
	100	*alpha* = 0.1	± 5 menit
	125	*alpha* = 0.1	± *6* menit
Bernoulli Naive Bayes	75	*alpha* = 1.0	± *4* menit
	100	*alpha* = 1.0	± *6* menit
	125	*alpha* = 0.1	± *8* menit
K-Nearest Neighbor	75	*n_neighbors* = 5	± 13 menit
	100	*n_neighbors* = 5	± 18 menit
	125	*n_neighbors* = 5	± *22* menit
Random Forest	75	*n_estimators = 80*	± *1* jam
	100	*n_estimators = 80*	± 2 jam 20 menit
	125	*n_estimators = 100*	± 3 jam
AdaBoost	15	*n_estimators* = 90, *learning_rate* = 1.0	± 5 jam

	20	*n_estimators* = 90, *learning_rate* = 1.0	± 7 jam
	25	*n_estimators* = 90, *learning_rate* = 1.0	± 9 jam

B. Multinomial Naive Bayes

Hyperparameter tuning akan dilakukan pada model ini dengan memilih *parameter alpha* dengan *range* 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, dan 1000.0. Kode Program 13 akan dijalankan pada semua algoritma yang akan diuji dengan mengganti *hyperparameter, hyperparameter range*, dan *library* algoritma.

```
from sklearn.naive_bayes import MultinomialNB
alphas = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
alpha_counts = {'0.0001': 0,'0.001': 0,'0.01': 0,'0.1': 0,
'1.0': 0,'10.0': 0,'100.0': 0,'1000.0': 0}

# Hyperparameter tuning
for i in range(100):
  best_parameter = ""
  best_f1_Score = 0
  for alpha in alphas:
    model_MultinomialNB = MultinomialNB(alpha=alpha)
    _, _, _, f1_score = Kfold(
                          X_train_tfidf,
                          y_train,
                          model_MultinomialNB
                        )
    if best_f1_Score < f1_score:
      best_f1_Score = f1_score
      best_parameter =model_MultinomialNB.get_params()['alpha']
  alpha_counts[str(best_parameter)] += 1
```

Kode Program 13 *Multinomial Naive Bayes hyperparameter tuning*

Tabel III menunjukkan bahwa model *Multinomial Naive Bayes* dengan *hyperparameter alpha* 0.1 memiliki performa yang paling baik di tiga jumlah iterasi yang berbeda dibandingkan dengan *parameter* lainnya. Dari segi hasil tidak ada perubahan dari ketiga instance *hyperparameter tuning* 75 iterasi, 100 iterasi, dan 125 iterasi. Kode Program 14 dilakukan setelah *Hyperparameter* terbaik telah ditemukan, dan akan dipakai juga ke semua algoritma yang diuji dengan mengganti *hyperparameter*, dan *library* algoritma.

TABEL III
HASIL *HYPERPARAMETER TUNING MULTINOMIAL NAIVE BAYES*

Alpha	75 iterasi	100 iterasi	125 iterasi
0.0001	0	0	0
0.001	0	0	1
0.01	6	6	5
0.1	45	61	84
1.0	25	33	35
10.0	0	0	0
100.0	0	0	0
1000.0	0	0	0

```
# Best hyperparameter
alpha = max(alpha_counts, key=alpha_counts.get)
print(f"Best hyperparameter value = {alpha}")

# Training model
model_MultinomialNB = MultinomialNB(alpha=float(alpha))
model_MultinomialNB.fit(X_train_tfidf, y_train)

# Testing model
accuracy_final, precision_final, recall_final, f1_score_final =
ConfusionMatrix(X_test_tfidf, y_test, model_MultinomialNB)
print(f"Acc={accuracy_final:.4f} Prec={precision_final:.4f}
Rec={recall_final:.4f} f1_s={f1_score_final:.4f}")
```

Kode Program 14 Model *Multinomial Naive Bayes idengan Menggunakan* Data *Testing* dan *Hyperparameter* terbaik

Tabel IV menunjukkan hasil dari Kode Program 14, dengan *accuracy* 0.8781, *precision* 0.8138, *recall* 0.8143, dan *f1_score* 0.814, model ini bekerja dengan sangat baik dan menghasilkan skor yang seimbang.

TABEL IV
HASIL *MULTINOMIAL NAIVE BAYES*

Alpha = 0.1	***Accuracy***	***Precision***	***Recall***	***F1_Score***
Label 0	0.8812	0.8092	0.8092	0.8092
Label 1	0.8931	0.8671	0.8509	0.8589
Label 2	0.8599	0.7652	0.7829	0.7739
Rata-rata	0.8781	0.8138	0.8143	0.8140

C. Bernoulli Naive Bayes

Hyperparameter tuning akan dilakukan pada model ini dengan memilih *parameter alpha* dengan *range* 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, dan 1000.0. Tabel V menunjukkan bahwa model *Bernoulli Naive Bayes* dengan *hyperparameter alpha* 1.0 memiliki performa yang paling baik pada hyperparameter tuning 75 iterasi dan 100 iterasi, sedangkan *hyperparameter alpha* 0.1 memiliki performa yang paling baik pada proses *hyperparameter tuning* 125 iterasi.

TABEL V
HASIL *HYPERPARAMETER TUNING BERNOULLI NAIVE BAYES*

Alpha	**75 iterasi**	**100 iterasi**	**125 iterasi**
0.0001	0	0	0
0.001	0	0	1
0.01	6	5	7
0.1	34	47	59
1.0	35	48	58
10.0	0	0	0
100.0	0	0	0
1000.0	0	0	0

TABEL VI
MODEL *BERNOULLI NAIVE BAYES HYPERPARAMETER* 1.0

Alpha = 1.0	***Accuracy***	***Precision***	***Recall***	***F1_Score***
Label 0	0.8670	0.7863	0.7863	0.7863
Label 1	0.8812	0.8675	0.8137	0.8397
Label 2	0.8290	0.7050	0.7597	0.7313
Rata-rata	0.8591	0.7863	0.7863	0.7858

TABEL VII
MODEL *BERNOULLI NAIVE BAYES HYPERPARAMETER* 0.1

Alpha = 0.1	***Accuracy***	***Precision***	***Recall***	***F1_Score***

Label 0	0.8860	0.8120	0.8244	0.8182
Label 1	0.8931	0.8867	0.8261	0.8553
Label 2	0.8504	0.7391	0.7907	0.7640
Rata-rata	0.8765	0.8126	0.8137	0.8125

Berdasarkan Tabel VI dan Tabel VII, *hyperparameter alpha* 0.1 bekerja lebih baik daripada *hyperparameter alpha* 1.0, perbedaan yang paling terlihat ada pada model 1.0 kesusahan dalam memahimi label 0 sehingga nilai rata-rata menjadi turun.

D. K-Nearest Neighbor

Hyperparameter yang akan digunakan pada model ini yaitu *n_neighbors* dengan *range* 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, dan 50. Tabel VIII menunjukan bahwa model *K-Nearest Neighbor* dengan *hyperparameter n-neighbor* 5 memiliki performa yang paling baik dibandingkan dengan parameter lainnya pada semua instance *hyperparameter tuning* 75 iterasi, 100 iterasi, dan 125 iterasi.

TABEL VIII
HASIL *HYPERPARAMETER TUNING K-NEAREST NEIGHBOR*

n_neighbor	75 iterasi	100 iterasi	125 iterasi
1	0	0	0
5	42	64	66
10	13	9	17
15	9	13	25
20	4	7	9
25	5	3	5
30	2	3	3
35	0	0	0
40	0	1	0
45	0	0	0
50	0	0	0

Tabel IX merupakan model *K-Nearest Neighbor* yang menggunakan *hyperparameter n_neighbor* 5 yang mencapai *accuracy* 0.8559, *precision* 0.7817, *recall* 0.7781, dan *f1_score* 0.7778, model ini bekerja dengan cukup baik, namun label 2 memiliki *recall* paling rendah sehingga mempengaruhi nilai secara keseluruhan.

TABEL IX
HASIL *K-NEIGHBOR HYPERPARAMETER N_NEIGHBOR 5*

n_neighbor = 5	*Accuracy*	*Precision*	*Recall*	*F1_Score*
Label 0	0.8456	0.7260	0.8092	0.7653
Label 1	0.8789	0.8354	0.8509	0.8431
Label 2	0.8432	0.7838	0.6744	0.7250
Rata-rata	0.8559	0.7817	0.7782	0.7778

E. Random Forest

Hyperparameter range yang akan digunakan yaitu *parameter n_estimators* dengan *range* 10, 20, 30, 40, 50, 60, 70, 80, 90, dan 100. Tabel X menunjukkan bahwa model *Random Forest* dengan *hyperparameter n_estimator* 80 unggul pada instance *hyperparameter tuning* 75 iterasi dan 100 iterasi sedangkan *instance hyperparameter tuning* 125 menghasilkan *hyperparameter n_estimator* 100.

TABEL X
HASIL *HYPERPARAMETER TUNING RANDOM FOREST*

n_estimator	75 iterasi	100 iterasi	125 iterasi
10	0	0	0

20	2	0	3
30	2	5	4
40	8	7	8
50	8	11	14
60	6	14	11
70	8	11	18
80	19	20	14
90	13	14	24
100	9	18	29

Tabel XI menghasilkan skor yang lebih bagus daripada Tabel XII secara keseluruhan. Akan tetapi, Model ini kesulitan dalam memahami label 0 sehingga nilai *recall* berkurang, terlalu berhati-hati dalam menebak label 1 menyebabkan ketidakseimbangan antara nilai *precision/recall*, dan banyak salah menebak label 2 sehingga nilai *precision* berkurang.

TABEL XI
MODEL *RANDOM FOREST HYPERPARAMETER N_ESTIMATOR 80*

n_estimator = 80	***Accuracy***	***Precision***	***Recall***	***F1_Score***
Label 0	0.8504	0.7881	0.7099	0.7470
Label 1	0.8836	0.9000	0.7826	0.8371
Label 2	0.7815	0.6135	0.7752	0.6849
Rata-rata	0.8385	0.7672	0.7559	0.7564

TABEL XII
MODEL *RANDOM FOREST HYPERPARAMETER N_ESTIMATOR 100*

n_estimator = 100	***Accuracy***	***Precision***	***Recall***	***F1_Score***
Label 0	0.8432	0.8218	0.6336	0.7155
Label 1	0.8694	0.8732	0.7702	0.8185
Label 2	0.7648	0.5843	0.8062	0.6775
Rata-rata	0.8258	0.7598	0.7367	0.7372

F. AdaBoost

Hyperparameter range yang akan digunakan *yaitu parameter n_estimator* dengan *range* 50, 60, 70, 80, 90, dan *parameter learning_rate* dengan *range* 0.01, 0.1, 1.0, 10.0.

TABEL XIII
HASIL *HYPERPARAMETER TUNING ADABOOST*

n_estimator	**Learning_rate**	**15 iterasi**	**20 iterasi**	**25 iterasi**
50	0.01	0	0	0
60	0.01	0	0	0
70	0.01	0	0	0
80	0.01	0	0	0
90	0.01	0	0	0
50	0.1	0	0	0
60	0.1	0	0	0
70	0.1	0	0	0
80	0.1	0	0	0
90	0.1	0	0	0
50	1.0	0	0	0

60	1.0	0	0	0
70	1.0	0	0	2
80	1.0	2	5	5
90	1.0	13	15	18
50	10.0	0	0	0
60	10.0	0	0	0
70	10.0	0	0	0
80	10.0	0	0	0
90	10.0	0	0	0

Tabel XIII Menunjukkan bahwa model *AdaBoost* dengan *hyperparameter n_estimator* 90 dan *learning_rate* 1.0 memiliki performa yang paling baik dibandingkan dengan *parameter* lainnya di seluruh *instance hyperparameter tuning*. Tabel XIV menghasilkan kinerja yang tidak baik, selain proses *hyperparameter tuning* pada model *AdaBoost* yang membutuhkan waktu sangat lama, hasil yang dicapai seperti pada label 0 dengan *recall* yang sangat rendah yaitu 0.1985 menunjukkan bahwa model kurang mengerti dengan karakteristik label 0, dan label 2 yang memiliki *precision* yang rendah menyebabkan prediksi yang salah sehingga *accuracy* juga ikut menurun.

TABEL XIV
HASIL AKHIR DARI MODEL *ADABOOST*

n_estimator = 90, learning_rate = 1.0	***Accuracy***	***Precision***	***Recall***	***F1_Score***
Label 0	0.7363	0.8125	0.1985	0.3190
Label 1	0.8124	0.9659	0.5280	0.6827
Label 2	0.5629	0.4086	0.9535	0.5721
Rata-rata	0.7039	0.7290	0.5600	0.5721

VI. KESIMPULAN

Penelitian ini menggunakan *dataset* yang sudah melalui tahapan pembersihan data dengan jumlah total 2102 artikel dan label yang kemudian dibagi menjadi 80% *training* dan 20% *testing*. Hasilnya, ditemukan bahwa algoritma *Multinomial Naive Bayes* dengan *hyperparameter alpha* 0.1 merupakan algoritma yang paling efektif dalam menangani permasalahan terkait klasifikasi berita bahasa Indonesia, karena mampu memberikan performa tertinggi dan paling konsisten di antara semua algoritma yang diuji. Algoritma ini berhasil mencapai skor *accuracy* 0.8781, *precision* 0.8138, *recall* 0.8143, dan *f1_score* 0.814. Adapun algoritma lain yang diuji pada penelitian ini yaitu Algoritma *Bernoulli Naive Bayes* dengan *hyperparameter alpha* 0.1 berhasil mencapai *accuracy* 0.8765, *precision* 0.8126, *recall* 0.8137, dan *f1_score* 0.8125. Algoritma *K-Nearest Neighbor* dengan *hyperparameter n_neighbor* 5 berhasil mencapai *accuracy* 0.8559, *precision* 0.7817, *recall* 0.7782, dan *f1_score* 0.7778. Algoritma *Random Forest* dengan *hyperparameter n_estimator* 80 berhasil mencapai *accuracy* 0.8321, *precision* 0.7635, *recall* 0.7463, dan *f1_score* 0.7471. Algoritma *AdaBoost* dengan *hyperparameter n_estimator* 90 dan *learning_rate* 1.0 berhasil mencapai *accuracy* 0.7039, *precision* 0.729, *recall* 0.56, dan *f1_score* 0.5246.

DAFTAR PUSTAKA

[1] R. Wongso, F. A. Luwinda, B. C. Trisnajaya, O. Rusli and Rudy, "News Article Text Classification in Indonesian Language," *Procedia Computer Science,* vol. 116, pp. 137-143, 2017.

[2] C. C. Yang, H. Chen and K. Hong, "Visualization of large category map for Internet browsing," *Decis. Support Syst.*, vol. 35, pp. 89-102, 2003.

[3] R. N. Devita, H. W. Herwanto and A. P. Wibawa, "Perbandingan Kinerja Metode Naive Bayes dan K-Nearest Neighbor untuk Klasifikasi Artikel Berbahasa Indonesia," *Jurnal Teknologi Informasi dan Ilmu Komputer,* 2018.

[4] R. B. Afrianto and L. Y. Kurniawati, "Kategorisasi Dokumen Teks Secara Multi Label Menggunakan Fuzzy C-Means Dan k-nearest Neighbors Pada Artikel Berbahasa Indonesia," 2013.

[5] B. H. Mahendra, "Kategorisasi Berita Multi-Label Berbahasa Indonesia Menggunakan Algoritma Random Forest," vol. 6, 2019.

[6] I. M. G. Arimbawa and N. A. S. Er, "Penerapan Metode Adaboost Untuk Multi-label Classification Pada Dokumen Teks," vol. 9, pp. 127-140, 2020.

[7] E. Y. Hidayat and M. A. Rizqi, "Klasifikasi Dokumen Berita Menggunakan Algoritma Enhanced Confix Stripping Stemmer dan Naïve Bayes Classifier," *Journal Nasional Teknologi Dan Sistem Informasi*, vol. 6, pp. 90-99, 2020.

[8] I. C. Irsan and M. L. Khodra, "Hierarchical multi-label news article classification with distributed semantic model based features," *International Journal of Advances in Intelligent Informatics*, vol. 5, no. 1, pp. 40-47, 2019.

[9] S. Patil, V. Lokesha and A. S. G., "Multi-Label News Category Text Classification," *Journal of Algebraic Statistic*, vol. 13, no. 3, pp. 5485-5498, 2022.

[10] T. J. Watson, "An empirical study of the naive Bayes classifier," pp. 41-46, 2001.

[11] A. Geron, in *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, Canada, O'Reilly Media, Inc, 2019, pp. 263-267.

[12] N. Rahmalia, "Berpengaruh pada Ranking Google dan SEO, Apa itu Stop Word?," 18 June 2021. [Online]. Available: https://glints.com/id/lowongan/stop-word-adalah/. [Accessed 18 10 2024].

[13] "Scikit-Learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html. [Accessed 21 10 2024].

[14] "Scikit-Learn," [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.confusion_matrix.html# sklearn.metrics.confusion_matrix. [Accessed 21 10 2024].

[15] "Scikit-Learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html. [Accessed 13 4 2025].

[16] Scikit-Learn,"Scikit-Learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html. [Accessed 13 4 2025].

[17] Domino, "domino.ai," domino, [Online]. Available: https://domino.ai/data-science-dictionary/hyperparameter-tuning. [Accessed 5 5 2025].

[18] "Medium.com," [Online]. Available: https://medium.com/@ppraveen2150/different-types-to-data-imputation-techniques-e1d 0c3702610.

[19] "Medium.com," [Online]. Available: https://medium.com/almabetter/what-is-cross-validation-hyperparameter-tuning-426bd17869ea.