TUGAS BESAR IF3070

Dasar Intelegensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Disusun oleh:

Ananda Farhan Raihandra	18222084
Alfaza Naufal Zakiy	18222126

Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

Daftar Isi

LATAR BELAKANG

Pembelajaran mesin adalah cabang kecerdasan buatan yang memungkinkan sistem mempelajari pola dari data dan menghasilkan prediksi atau keputusan tanpa membutuhkan pemrograman eksplisit.

Tugas ini dirancang untuk memberikan kesempatan kepada mahasiswa untuk secara langsung mengaplikasikan algoritma pembelajaran mesin pada permasalahan dunia nyata. Pada tugas ini mahasiswa diminta untuk mengimplementasikan algoritma pembelajaran mesin yang telah dipelajari selama perkuliahan, yaitu KNN dan Gaussian Naive-Bayes, menggunakan dataset PhiUSIIL Phishing URL Dataset. Implementasi untuk kedua algoritma yang dibuat menggunakan pendekatan from scratch. Kemudian hasil dari algoritma from scratch dibandingkan dengan algoritma scikit-learn.

Selain implementasi algoritma, mahasiswa diminta untuk model yang telah dibuat harus bisa disimpan dan dimuat ulang agar dapat digunakan kembali.

IMPLEMENTASI PROGRAM

1. Implementasi Algoritma KNN

K-Nearest Neighbors (KNN) adalah salah satu algoritma pembelajaran mesin yang digunakan untuk masalah klasifikasi dan regresi. KNN termasuk dalam kelompok algoritma pembelajaran berbasis instance (instance-based learning) atau lazy learning, di mana algoritma ini tidak melakukan proses pelatihan eksplisit tetapi membuat prediksi berdasarkan data yang sudah ada.

KNN bekerja berdasarkan prinsip kedekatan data. Algoritma ini menentukan kelas atau nilai dari sebuah data baru (instance) dengan mencari sejumlah *k* tetangga terdekat dari data tersebut dalam dataset yang ada.

Langkah Kerja Algoritma KNN:

1. Tentukan jumlah tetangga terdekat (*k*) yang akan digunakan.
2. Menghitung jarak antara data baru dan seluruh data dalam dataset. Metode perhitungan jarak yang umum digunakan adalah:
   - Euclidean Distance: $d(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$
   - Manhattan Distance: $d(x_1, x_2) = \sum_{i=1}^{n} |x_{1i} - x_{2i}|$
   - Minkowski Distance: $d(x_1, x_2) = \left(\sum_{i=1}^{n} |x_{1i} - x_{2i}|^p \right)^{\frac{1}{p}}$
3. Memilih *k* data dengan jarak terdekat berdasarkan metrik yang digunakan.
4. Menentukan kelas atau nilai
5. Prediksi kelas atau nilai untuk data baru berdasarkan hasil langkah sebelumnya.

Kode Implementasi Algoritma KNN kami:

```python
# Kelas KNN from scratch
class KNN:
    def __init__(self, k=3, metric='euclidean'):
        self.k = k
        self.metric = metric
        self.X_train = None
        self.y_train = None

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
```

```
        def _distance(self, x1, x2):
            if self.metric == 'euclidean':
                return np.sqrt(np.sum((x1 - x2) ** 2))
            elif self.metric == 'manhattan':
                return np.sum(np.abs(x1 - x2))
            elif self.metric == 'minkowski':
                p = 3
                return np.sum(np.abs(x1 - x2) ** p) ** (1 / p)

        def predict(self, X):
            predictions = []
            for x in tqdm(X[:10000], desc="KNN Prediction Progress"):
# Subset data
                distances = np.linalg.norm(self.X_train - x, axis=1)
# Optimasi jarak
                k_indices = np.argsort(distances)[:self.k]
                k_nearest_labels = [self.y_train[i] for i in
k_indices]
                predictions.append(max(set(k_nearest_labels),
key=k_nearest_labels.count))
            return np.array(predictions)

    # Data latih dan validasi
    # Pastikan `X_train_resampled` dan `y_train_resampled` sesuai
dengan hasil preprocessing sebelumnya
    # Misalnya, gunakan subset untuk percepatan
    X_train_subset = X_train_resampled.to_numpy()[:40000]
    y_train_subset = y_train_resampled.to_numpy()[:40000]
    X_val_subset = X_val.to_numpy()[:10000]
    y_val_subset = y_val.to_numpy()[:10000]

    # KNN from scratch
    knn_scratch = KNN(k=5, metric='euclidean')
    knn_scratch.fit(X_train_subset, y_train_subset)
    y_pred_scratch = knn_scratch.predict(X_val_subset)
```

```
        # KNN scikit-learn
        knn_sklearn = KNeighborsClassifier(n_neighbors=5,
metric='euclidean')
        knn_sklearn.fit(X_train_subset, y_train_subset)
        y_pred_sklearn = knn_sklearn.predict(X_val_subset)

        # Evaluasi akurasi
        accuracy_scratch = accuracy_score(y_val_subset, y_pred_scratch)
        accuracy_sklearn = accuracy_score(y_val_subset, y_pred_sklearn)

        # Cetak hasil
        print(f"Akurasi KNN from scratch (subset):
{accuracy_scratch:.5f}")
        print(f"Akurasi KNN scikit-learn (subset):
{accuracy_sklearn:.5f}")
```

KNN memiliki keunggulan karena algoritmanya yang sederhana dan mudah diimplementasikan, serta tidak memerlukan proses pelatihan model sehingga cocok digunakan pada dataset kecil. Selain itu, KNN dapat diterapkan baik untuk masalah klasifikasi maupun regresi. Namun, algoritma ini memiliki kelemahan, seperti performa yang menurun pada dataset besar karena perhitungan jarak yang memakan waktu. KNN juga sensitif terhadap fitur dengan skala berbeda, sehingga normalisasi data diperlukan, dan pemilihan nilai k yang tidak tepat dapat memengaruhi akurasi model secara signifikan.

2. Implementasi Algoritma Gaussian Naive-Bayes

Gaussian Naive-Bayes adalah salah satu varian dari algoritma Naive-Bayes, yang merupakan algoritma pembelajaran mesin berbasis probabilitas untuk masalah klasifikasi. Algoritma ini bekerja berdasarkan Teorema Bayes, dengan asumsi bahwa setiap fitur bersifat independen (independent assumption) satu sama lain, meskipun dalam kenyataannya, asumsi ini seringkali tidak sepenuhnya terpenuhi.

Teorema Bayes adalah aturan dalam teori probabilitas yang digunakan untuk menghitung probabilitas suatu hipotesis berdasarkan informasi atau bukti yang tersedia. Teorema ini menyatakan hubungan antara probabilitas prior (sebelum bukti diketahui) dan probabilitas posterior (setelah bukti diketahui).

Teorema Bayes mendasari algoritma ini, yang dinyatakan dalam persamaan berikut:

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

- $P(C|X)$: Probabilitas posterior bahwa data X berasal dari kelas C

- $P(X|C)$: Likelihood, yaitu probabilitas data X diberikan kelas C
- $P(C)$: Probabilitas prior dari kelas C
- $P(X)$: Probabilitas data X (konstanta untuk semua kelas)

Algoritma Naive-Bayes memprediksi kelas C yang memiliki probabilitas $P(C|X)$ tertinggi.

Karakteristik Gaussian Naive-Bayes

- Gaussian Naive-Bayes digunakan ketika fitur-fitur input bersifat numerik dan diasumsikan mengikuti distribusi Gaussian (normal distribution).
- Probabilitas likelihood $P(X|C)$ dihitung menggunakan fungsi densitas Gaussian:

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

- μ: Mean (rata-rata) dari data untuk kelas tertentu.
- σ^2: Variansi dari data untuk kelas tertentu.

Langkah Kerja Algoritma Gaussian Naive-Bayes:

1. Menghitung parameter yang diperlukan, seperti menghitung rata-rata (μ) dan variansi (σ^2) untuk setiap fitur pada setiap kelas dan menghitung probabilitas prior $P(C)$ untuk setiap kelas.
2. Menghitung probabilitas likelihood $P(X|C)$ menggunakan fungsi densitas Gaussian untuk setiap kelas.
3. Menghitung probabilitas posterior $P(C|X)$ untuk semua kelas.
4. Prediksi kelas dengan $P(C|X)$ tertinggi.

Kode Implementasi Algoritma KNN kami:

```
class GaussianNaiveBayes:
    def __init__(self):
        self.classes = None
        self.mean = {}
        self.var = {}
        self.priors = {}

    def fit(self, X, y):
        """
        Melatih Gaussian Naive Bayes dengan menghitung rata-rata,
variansi,
        dan probabilitas prior untuk setiap kelas.
        """
        self.classes = np.unique(y)
```

```
        for c in self.classes:
            X_c = X[y == c]
            self.mean[c] = X_c.mean(axis=0)
            self.var[c] = X_c.var(axis=0)
            self.priors[c] = X_c.shape[0] / X.shape[0]

    def _gaussian_density(self, class_idx, x):
        """
        Menghitung densitas Gaussian untuk fitur tertentu dengan
regularisasi.
        """
        mean = self.mean[class_idx]
        var = self.var[class_idx] + 1e-9  # Regularisasi variansi
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return np.maximum(numerator / denominator, 1e-9)  # Hindari nol

    def predict(self, X):
        """
        Melakukan prediksi untuk data input.
        """
        y_pred = []
        for x in X:
            posteriors = []
            for c in self.classes:
                prior = np.log(self.priors[c])  # Log prior
                class_conditional =
np.sum(np.log(self._gaussian_density(c, x)))
                posterior = prior + class_conditional
                posteriors.append(posterior)
            y_pred.append(self.classes[np.argmax(posteriors)])
        return np.array(y_pred)

# Data latih dan validasi
X_train_resampled, y_train_resampled  # Data hasil SMOTE
X_val, y_val  # Data validasi
```

```
# Gaussian Naive-Bayes from scratch
gnb_scratch = GaussianNaiveBayes()
gnb_scratch.fit(X_train_resampled.to_numpy(),
y_train_resampled.to_numpy())
y_pred_scratch = gnb_scratch.predict(X_val.to_numpy())

# Evaluasi akurasi
accuracy_scratch = accuracy_score(y_val, y_pred_scratch)
print(f"Akurasi Gaussian Naive-Bayes from scratch:
{accuracy_scratch:.2f}")
```

Gaussian Naive-Bayes memiliki keunggulan berupa kecepatan dan efisiensi karena algoritmanya membutuhkan sedikit data untuk pelatihan, serta kinerjanya yang baik pada dataset besar, terutama jika asumsi Gaussian hampir terpenuhi. Algoritma ini juga fleksibel dan dapat digunakan untuk berbagai masalah klasifikasi dengan fitur numerik. Namun, kelemahannya adalah sensitivitas terhadap asumsi Gaussian, di mana jika data tidak mengikuti distribusi normal, akurasi model dapat menurun. Selain itu, asumsi independensi antar fitur jarang terpenuhi dalam data nyata, meskipun algoritma ini tetap dapat bekerja cukup baik dalam banyak kasus.

Cleaning and Preprocessing

1. Data Cleaning
    - Handling Missing Data

      Proses Handling Missing Data bertujuan untuk menangani nilai kosong dalam dataset sehingga data dapat digunakan secara optimal tanpa kehilangan informasi penting. Langkah pertama adalah membuat objek `SimpleImputer` dari pustaka `scikit-learn`, di mana strategi `mean` digunakan untuk menggantikan nilai kosong pada kolom numerik dengan rata-rata, dan strategi `most_frequent` digunakan untuk menggantikan nilai kosong pada kolom kategori dengan nilai yang paling sering muncul (modus). Selanjutnya, kolom numerik diidentifikasi berdasarkan tipe data `float64` dan `int64`, sedangkan kolom kategori diidentifikasi berdasarkan tipe data `object`. Imputasi dilakukan secara terpisah untuk data latih dan data validasi. Data latih diimputasi dengan nilai rata-rata atau modus yang dihitung langsung dari dataset latih, sedangkan data validasi menggunakan parameter yang sama untuk menjaga konsistensi. Setelah proses imputasi selesai, dilakukan pengecekan apakah masih ada nilai kosong menggunakan metode `isnull().sum().sum()`. Hasil pengecekan menunjukkan bahwa semua nilai kosong telah berhasil diatasi, memastikan data siap untuk langkah pemrosesan berikutnya. Teknik ini memastikan bahwa data latih dan validasi bebas dari masalah missing values, sehingga analisis lebih lanjut tidak terganggu oleh ketidaksempurnaan data.
    - Dealing with Outliers

      Proses Dealing with Outliers dilakukan dengan metode clipping untuk membatasi nilai-nilai ekstrim pada data numerik berdasarkan persentil 1% (batas bawah) dan 99% (batas atas). Kolom numerik diidentifikasi, lalu nilai yang melebihi batas tersebut disesuaikan ke batas yang ditentukan menggunakan fungsi `clip`. Proses ini diterapkan pada data latih dan validasi untuk memastikan konsistensi, sehingga outlier tidak lagi memengaruhi model tanpa kehilangan data penting. Clipping menjaga data tetap dalam rentang wajar sambil mempertahankan informasi utama dataset.
    - Remove Duplicates

      Proses Remove Duplicates bertujuan untuk menghapus data duplikat pada dataset latih (`X_train`) dan validasi (`X_val`) guna memastikan bahwa model tidak terpengaruh oleh data yang berulang, yang dapat menyebabkan bias. Langkah pertama adalah menyimpan bentuk awal dataset sebelum penghapusan untuk perbandingan. Duplikat dihapus menggunakan fungsi `drop_duplicates(keep='first')`, yang mempertahankan kemunculan pertama dari data yang berulang. Setelah itu, label target (`y_train` dan `y_val`) disesuaikan dengan indeks data fitur yang telah dibersihkan. Proses ini mengurangi kemungkinan bias dalam model dan memastikan bahwa dataset lebih representatif tanpa kehilangan informasi unik. Hasil akhirnya ditampilkan untuk menunjukkan perubahan dimensi dataset.
    - Feature Engineering

      Proses Feature Engineering bertujuan untuk meningkatkan kualitas data dengan menambahkan fitur baru dan memodifikasi fitur yang ada. Pertama, nilai kosong pada kolom URL diisi dengan string kosong untuk memastikan

konsistensi data. Selanjutnya, fitur baru ditambahkan berdasarkan karakteristik URL, seperti panjang URL (`URL_Length`), jumlah digit (`Digit_Count`), dan jumlah karakter khusus (`Special_Char_Count`). Pada kolom numerik, dilakukan log transform untuk mengurangi efek distribusi tidak normal. Selain itu, kolom kategori dikodekan menjadi nilai numerik menggunakan metode encoding berbasis kategori (`cat.codes`). Proses ini memastikan data lebih representatif untuk analisis lebih lanjut dan mendukung performa model pembelajaran mesin. Setelah perubahan, dimensi data diverifikasi untuk memastikan keberhasilan proses feature engineering.

2. Data Preprocessing
    - Feature Scaling

      Proses Feature Scaling bertujuan untuk menormalisasi nilai kolom numerik agar berada dalam rentang yang sama, sehingga algoritma pembelajaran mesin dapat bekerja lebih optimal. Langkah pertama adalah mengidentifikasi kolom numerik berdasarkan tipe data `float64` dan `int64`. Kemudian, `MinMaxScaler` digunakan untuk mengubah nilai data menjadi rentang [0, 1]. Proses scaling dilakukan secara terpisah untuk data latih menggunakan `fit_transform`, dan data validasi menggunakan `transform` agar konsistensi data tetap terjaga. Setelah scaling selesai, hasilnya diverifikasi dengan menampilkan contoh data latih yang telah dinormalisasi. Pendekatan ini memastikan fitur numerik memiliki skala yang seragam, mengurangi pengaruh fitur dengan rentang nilai yang besar.
    - Feature Encoding

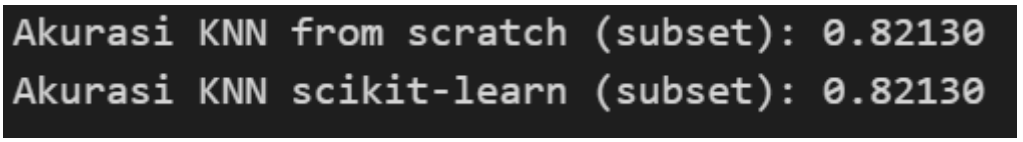
      Proses Feature Encoding bertujuan untuk mengubah data kategori menjadi representasi numerik agar dapat digunakan oleh algoritma pembelajaran mesin. Langkah pertama adalah mengidentifikasi kolom kategori berdasarkan tipe data `object`. Selanjutnya, `LabelEncoder` digunakan untuk mengonversi setiap nilai unik pada kolom kategori menjadi angka. Proses encoding dilakukan secara terpisah untuk data latih dengan metode `fit_transform` dan data validasi dengan metode `transform` untuk menjaga konsistensi. Setiap encoder yang digunakan disimpan dalam dictionary `label_encoders` untuk keperluan referensi di masa depan. Setelah proses encoding selesai, hasilnya diverifikasi dengan menampilkan contoh data latih yang telah dikodekan. Pendekatan ini memastikan bahwa data kategori dapat dimanfaatkan oleh model pembelajaran mesin secara efektif.
    - Handling Imbalanced Dataset

      Proses Handling Imbalanced Dataset menggunakan teknik SMOTE (Synthetic Minority Oversampling Technique) untuk menyeimbangkan distribusi kelas pada data latih. Langkah pertama adalah mencetak distribusi awal kelas menggunakan fungsi `Counter` untuk mengetahui ketidakseimbangan. Kemudian, objek SMOTE diinisialisasi dengan parameter `random_state=42` untuk menghasilkan hasil yang konsisten. SMOTE diterapkan ke data latih menggunakan metode `fit_resample`, yang secara sintetis membuat sampel baru untuk kelas minoritas berdasarkan data yang ada. Setelah proses resampling selesai, distribusi kelas diperiksa kembali untuk memastikan bahwa kelas-kelas sudah seimbang. Pendekatan ini membantu meningkatkan performa model pada data dengan ketidakseimbangan kelas yang signifikan.

HASIL DAN ANALISIS

Setelah data diproses dan model selesai dibuat, langkah berikutnya adalah mengevaluasi kinerja model. Evaluasi dilakukan dengan teknik pembagian data 80:20, di mana 80% data digunakan sebagai data pelatihan (training) dan 20% sebagai data validasi. Dalam proses ini, kami menggunakan dua model, yaitu KNN dan Gaussian Naive Bayes, dengan dua pendekatan, yakni implementasi manual (from scratch) dan menggunakan library sklearn sebagai pembanding.

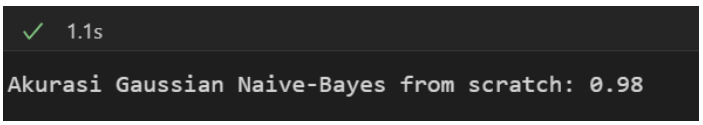
1. Hasil Algoritma KNN



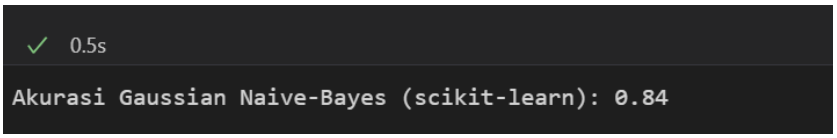
Gambar 1 Hasil KNN from scratch dan from scikit-learn

Implementasi KNN, baik secara manual (from scratch) maupun menggunakan Sklearn, menghasilkan evaluasi dengan akurasi yang sama, yaitu 0.82130. Kesamaan ini menunjukkan bahwa implementasi manual telah berhasil meniru cara kerja Sklearn dengan tepat. Algoritma KNN bekerja dengan menghitung jarak antar data menggunakan metrik seperti Euclidean, Manhattan, dan Minkowski untuk menentukan kelas berdasarkan mayoritas tetangga terdekat. Keberhasilan tersebut didukung oleh kesamaan parameter yang digunakan pada kedua metode. Tidak ditemukan perbedaan signifikan, yang menandakan bahwa prinsip dasar algoritma KNN telah diterapkan secara benar.

2. Hasil Algoritma Gaussian Naive-Bayes



Gambar 2 Hasil Gaussian Naive-Bayes from scratch



Gambar 3 Hasil Gaussian Naive-Bayes from scikit-learn

Perbedaan hasil akurasi antara Gaussian Naive-Bayes (GNB) from scratch dan GNB dari scikit-learn dapat disebabkan oleh beberapa faktor teknis. Pada implementasi from scratch, regularisasi variansi ditambahkan secara manual $(1e-9)$ untuk menghindari pembagian dengan nol atau nilai densitas yang ekstrem. Hal ini dapat meningkatkan akurasi pada dataset tertentu, tetapi mungkin kurang fleksibel untuk dataset lain. Sementara itu, scikit-learn menggunakan optimisasi internal dan algoritma numerik yang lebih presisi untuk menghitung probabilitas, sehingga menghasilkan hasil yang lebih stabil tetapi terkadang tidak seakurat implementasi manual pada dataset yang sangat spesifik.

Selain itu, scikit-learn memiliki fitur tambahan untuk menangani berbagai kondisi dataset, seperti normalisasi yang lebih terintegrasi, optimisasi model, dan mekanisme default untuk kasus edge (misalnya, variansi nol atau distribusi yang tidak normal). Hal ini membuat scikit-learn lebih robust tetapi terkadang menghasilkan akurasi lebih rendah pada dataset tertentu dibandingkan implementasi manual yang lebih disesuaikan. Perbedaan juga dapat timbul jika langkah preprocessing, seperti scaling dan encoding, tidak diterapkan secara identik pada kedua pendekatan. Perbedaan kecil dalam presisi numerik atau penanganan outlier juga dapat memengaruhi hasil akhir akurasi. Implementasi from scratch menunjukkan akurasi lebih tinggi karena model mungkin lebih sesuai dengan dataset spesifik ini, tetapi hasil ini tidak selalu menjamin kemampuan generalisasi lebih baik.

ERROR ANALYSIS

Gaussian Naive-Bayes (GNB) from scratch memiliki akurasi lebih tinggi dibandingkan dengan versi scikit-learn, yang dapat disebabkan oleh regularisasi manual pada variansi $(1e-9)$, membuat model lebih cocok untuk dataset tertentu. Sebaliknya, versi scikit-learn lebih robust dan dirancang untuk generalisasi pada dataset yang lebih beragam. Pada algoritma KNN, akurasi antara implementasi from scratch dan scikit-learn relatif serupa, menunjukkan bahwa algoritma ini tidak terlalu bergantung pada optimisasi internal seperti pada GNB.

Feature scaling menggunakan `MinMaxScaler` terbukti penting, terutama untuk algoritma berbasis jarak seperti KNN, karena skala fitur yang seragam meningkatkan akurasi model. Selain itu, langkah preprocessing seperti imputasi missing values dengan mean dan modus, serta balancing data menggunakan SMOTE, memberikan kontribusi signifikan dalam meningkatkan performa model. SMOTE secara khusus membantu menyeimbangkan distribusi kelas, sehingga model lebih baik dalam memprediksi kelas minoritas.

Rekomendasi perbaikan mencakup analisis confusion matrix untuk memahami prediksi antar kelas secara lebih mendetail dan penggunaan cross-validation untuk mengukur generalisasi model. Selain itu, eksperimen lebih lanjut dengan langkah preprocessing seperti imputasi atau scaling dapat membantu menentukan kombinasi terbaik untuk meningkatkan akurasi secara keseluruhan. Pendekatan ini akan membantu model menjadi lebih robust dan generalizable.

KONTRIBUSI

NIM	Nama	Tugas
18222084	Ananda Farhan Raihandra	● Import Libraries dan dataset ● Split Training Set and Validation Set ● Handling Missing Data dan Dealing with Outliers
18222126	Alfaza Naufal Zakiy	● Remove Duplicates dan Feature Engineering ● Feature Scaling, Feature Encoding, dan Handling Imbalanced Dataset ● KNN ● Gaussian Naive-Bayes ● Save dan Load Model ● Error Analysis ● Dokumen

REFERENSI

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27.

scikit-learn documentation: K-Nearest Neighbors

https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset

https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub

https://scikit-learn.org/1.5/modules/neighbors.html

https://scikit-learn.org/1.5/modules/naive_bayes.html

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

scikit-learn documentation: Naive Bayes

Rish, I. (2001). *An empirical study of the naive Bayes classifier*. IJCAI Workshop on Empirical Methods in AI.