# Tugas Besar 2

# II3170 Dasar Inteligensi Artifisial

# Implementasi Algoritma Pembelajaran Mesin


**Disusun Oleh:**

**Kelompok 4 K2**

| | |
|---|---|
| Rashid May | / 18222014 |
| Lutfi Khairul Amal | / 18222018 |
| Filbert Fuvian | / 18222024 |
| Qady Zakka Raymaula | / 18222038 |

**Program Studi Sistem dan Teknologi Informasi**

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

**Jl. Ganesha 10, Bandung 40132**

**2024**

# Daftar Isi

# KNN

## Algoritma

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

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

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return predictions

    def _predict(self, x):
        distances = [self._compute_distance(x, x_train) for x_train in
self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = np.bincount(k_nearest_labels).argmax()
        return most_common

    def _compute_distance(self, x1, x2):
        if self.distance_metric == 'euclidean':
            return np.sqrt(np.sum((x1 - x2) ** 2))
        elif self.distance_metric == 'manhattan':
            return np.sum(np.abs(x1 - x2))
        elif self.distance_metric == 'minkowski':
            return np.sum(np.abs(x1 - x2) ** self.p) ** (1 / self.p)
        else:
            raise ValueError(f"Distance metric salah:
{self.distance_metric}")
```

Cara penggunaan:

```
knn_model_manhattan = KNN(k=3, distance_metric='manhattan')
knn_model_manhattan.fit(X_train, y_train)
knn_predictions_manhattan = knn_model_manhattan.predict(X_test)
```

## Penjelasan

Algoritma KNN adalah metode pembelajaran mesin yang digunakan untuk klasifikasi dan regresi. Pada tugas ini, model ini digunakan untuk menentukan apakah sebuah URL memiliki label phising atau legitimate dengan menggunakan data karakteristik dari URL tersebut.

Inisialisasi kelas KNN akan dimulai dengan mengambil parameter k yang menentukan jumlah *nearest neighbor* yang dipertimbangkan dan parameter *distance* metric untuk menentukan metode perhitungan *distance* yang digunakan, parameter X_train dan y_train juga diambil sebagai dataset *training* dimana X_train berisi data yang di analisa dan y_train data label.

Kerja algoritma utama ada pada *method* predict. Pada metode ini, untuk setiap row dari data X (data yang di-evaluasi / di-tes), row tersebut akan dimasukan ke *method* _predict. Pada *method* _predict, sebuah sampel x / row x, akan di prediksi dengan pertama tama menghitung jarak *euclidean* antara x dan setiap sampel dalam X_train. Kemudian, indeks dari jarak-jarak (sesuai parameter k) yang terdekat akan diambil menggunakan fungsi np.argsort. Dari indeks, k *neighbors* tersebut, akan dicari *most common* label atau label yang paling banyak muncul dan sampel x akan diprediksi sama dengan label yang paling banyak muncul dari k *neighbors.*
n, dan label yang paling umum di antara tetangga tersebut dipilih sebagai prediksi.

Secara keseluruhan, algoritma KNN bekerja dengan mencari k tetangga terdekat dari sampel yang ingin diprediksi, kemudian menentukan label berdasarkan mayoritas label dari tetangga tersebut.

# Naive-Bayes

## Algoritma

```
class NaiveBayes:
    def __init__(self):
        self.class_priors = {}
        self.feature_likelihoods = {}
        self.num_features = 0

    def fit(self, X, y):
        self.num_features = X.shape[1]
        self.class_priors = self._calculate_class_priors(y)
        self.feature_likelihoods =
self._calculate_feature_likelihoods(X, y)

    def _calculate_class_priors(self, y):
        class_priors = {}
        total_samples = len(y)
        for label in set(y):
            class_priors[label] = sum(y == label) / total_samples
        return class_priors

    def _calculate_feature_likelihoods(self, X, y):
        likelihoods = {}
        for label in set(y):
            X_label = X[y == label]
            likelihoods[label] = {
                feature_index: (X_label[:, feature_index].mean(),
X_label[:, feature_index].std())
                for feature_index in range(self.num_features)
            }
        return likelihoods

    def predict(self, X):
        predictions = []
        for x in X:
            posteriors = {}
            for label in self.class_priors:
```

```
                prior = self.class_priors[label]
                likelihood = self._calculate_likelihood(x, label)
                posteriors[label] = prior * likelihood
            predictions.append(int(max(posteriors, key=posteriors.get)))
        return predictions

    def _calculate_likelihood(self, x, label):
        likelihood = 1
        for feature_index in range(self.num_features):
            mean, std = self.feature_likelihoods[label][feature_index]
            likelihood *= self._gaussian_probability(x[feature_index],
mean, std)
        return likelihood

    def _gaussian_probability(self, x, mean, std):
        if std == 0:
            return 1 if x == mean else 0
        exponent = np.exp(-((x - mean) ** 2 / (2 * std ** 2)))
        return (1 / (np.sqrt(2 * np.pi) * std)) * exponent
```

## Penjelasan

Algoritma Naive Bayes adalah metode pembelajaran mesin yang digunakan untuk klasifikasi. Dalam tugas ini, model ini digunakan untuk menentukan probabilitas apakah sebuah URL memiliki label phising atau legitimate dengan menggunakan data karakteristik dari URL tersebut.

Inisialisasi kelas Naive-Bayes dimulai dengan membuat dua variabel: class_priors untuk menyimpan probabilitas prior dari setiap kelas, dan feature_likelihoods untuk menyimpan likelihood (rata-rata dan deviasi standar) dari setiap fitur untuk setiap kelas. Variabel num_features menyimpan jumlah fitur dalam dataset.

Kerja algoritma utama ada pada method fit. Pada metode ini, dataset pelatihan X dan label y diberikan ke model. Metode fit menghitung prior kelas menggunakan _calculate_class_priors dan likelihood fitur menggunakan _calculate_feature_likelihoods.

Metode _calculate_class_priors menghitung probabilitas prior dari setiap kelas dengan membagi jumlah sampel dari setiap kelas dengan total sampel. Metode _calculate_feature_likelihoods menghitung rata-rata dan deviasi standar dari setiap fitur untuk

setiap kelas. Informasi ini digunakan untuk menghitung likelihood dari nilai fitur yang diberikan kelas tertentu.

Method predict digunakan untuk memprediksi label dari data baru. Pada metode ini, untuk setiap data dalam X, dihitung probabilitas posterior untuk setiap kelas. Ini dilakukan dengan mengalikan prior kelas dengan likelihood dari data tersebut yang diberikan kelas tertentu, yang dihitung menggunakan metode _calculate_likelihood. Kelas dengan probabilitas posterior tertinggi dipilih sebagai label prediksi.

Metode _calculate_likelihood menghitung likelihood dari data yang diberikan kelas tertentu dengan mengalikan probabilitas Gaussian dari setiap fitur. Metode _gaussian_probability menghitung probabilitas Gaussian dari nilai fitur yang diberikan rata-rata dan deviasi standar fitur tersebut untuk kelas tertentu. Probabilitas Gaussian diasumsikan mengikuti distribusi normal.

Secara keseluruhan, algoritma Naive Bayes bekerja dengan menghitung prior dan likelihood fitur dari data pelatihan, kemudian menggunakan informasi tersebut untuk menghitung probabilitas posterior dari data baru dan memprediksi kelas berdasarkan probabilitas tertinggi.

# Data Cleaning & Preprocessing

## Data Cleaning

1. Handling Missing Data
   Identify and address missing values in the dataset. This can include imputing missing values, removing rows or columns with excessive missing data, or using more advanced techniques like interpolation.

```
binary_imputer = SimpleImputer(strategy='constant', fill_value=0)
train_set[binary_features] =
binary_imputer.fit_transform(train_set[binary_features])
val_set[binary_features] =
binary_imputer.transform(val_set[binary_features])

numerical_imputer = SimpleImputer(strategy='median')
train_set[numerical_features] =
numerical_imputer.fit_transform(train_set[numerical_features])
val_set[numerical_features] =
numerical_imputer.transform(val_set[numerical_features])

train_set[categorical_features] =
train_set[categorical_features].fillna('Unknown')
val_set[categorical_features] =
```

```
val_set[categorical_features].fillna('Unknown')
```

*Missing value handling* dilakukan dengan tiga cara tergantung tipe data pada feature tersebut. Jika feature merupakan data bertipe binary, yaitu True or False / 0 atau 1, maka *missing data* akan diisi dengan constant value yaitu '0' atau False. Jika feature bertipe numerical, maka *missing value* akan diisi dengan median dari feature tersebut. Terakhir jika feature merupakan categorical, maka akan diisi dengan 'Unknown'.

2. Dealing with Outliers
   Identify and handle outliers, which are data points significantly different from the rest of the dataset. Outliers can be removed or transformed to improve model performance.

```
def winsorize(data, lower_percentile=1, upper_percentile=99):
    lower_bound = np.percentile(data, lower_percentile)
    upper_bound = np.percentile(data, upper_percentile)
    data = np.clip(data, lower_bound, upper_bound)
    return data

for feature in numerical_features:
    train_set[feature] = winsorize(train_set[feature].values)
    val_set[feature] = winsorize(val_set[feature].values)
    test_df1[feature] = winsorize(test_df1[feature].values)
```

Fungsi winsorize digunakan untuk membatasi nilai-nilai dalam data agar berada dalam batas tertentu. Fungsi ini menerima data dan dua parameter persentil (lower_percentile dan upper_percentile). Nilai-nilai dalam data yang berada di bawah persentil bawah atau di atas persentil atas akan diubah menjadi nilai batas bawah atau batas atas tersebut. Ini membantu mengurangi pengaruh outliers ekstrim.

3. Removing Duplicates
   Identify and remove duplicate rows, as they can skew the model's training process and evaluation metrics.

```
train_set.drop_duplicates(inplace=True)
val_set.drop_duplicates(inplace=True)
```

Menggunakan drop_duplicates untuk menghapus *row* yang duplikat.

4. Feature Engineering
   Create new features or modify existing ones to extract relevant information. This step can involve scaling, normalizing, or encoding features for better model interpretability.
   1) Drop features
      Ada beberapa *features* yang menurut saya tidak diperlukan karena redundan ataupun memiliki data yang kurang berkualitas.
      beberapa *features* tersebut merupakan:
      - Data Redundan (Ada *features* lain yan dapat menjelaskan feature ini)
        [ 'FILENAME','TLD', 'URL', 'Domain', 'Title']
      - Data Jelek (Tidak make sense / tidak relevan)
        ['IsDomainIP', 'HasObfuscation', 'NoOfObfuscatedChar', 'ObfuscationRatio', 'NoOfDegitsInURL', 'DegitRatioInURL', 'NoOfEqualsInURL', 'NoOfQMarkInURL', 'NoOfAmpersandInURL', 'NoOfURLRedirect', 'NoOfSelfRedirect', 'NoOfPopup', 'NoOfiFrame', 'NoOfEmptyRef', 'DomainTitleMatchScore', 'URLTitleMatchScore']

```
dropped_features = ['FILENAME','TLD', 'URL', 'Domain',
'Title', 'IsDomainIP', 'HasObfuscation',
'NoOfObfuscatedChar', 'ObfuscationRatio', 'NoOfDegitsInURL',
'DegitRatioInURL', 'NoOfEqualsInURL', 'NoOfQMarkInURL',
'NoOfAmpersandInURL', 'NoOfURLRedirect',
'NoOfSelfRedirect', 'NoOfPopup', 'NoOfiFrame',
'NoOfEmptyRef', 'DomainTitleMatchScore','URLTitleMatchScore']
```

```
class DropFeatures(BaseEstimator, TransformerMixin):
    def __init__(self, features_to_drop):
        self.features_to_drop = features_to_drop

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.drop(columns=self.features_to_drop, axis=1)
```

      class digunakan agar DropFeatures dapat digunakan dengan *pipeline*

   2) Discretizing
      Dari beberapa Numerical features, ada beberapa *features* yang cocok untuk di *discretize* yaitu dikategorikan menjadi {'High' , 'Medium, 'Low'}. Lebih tepatnya, *features* yang akan di *discretize* adalah *features* yang bukan sebuah probabilitas.

```
features_to_discretize = [
'URLLength', 'DomainLength', 'LineOfCode',
'LargestLineLength', 'NoOfLettersInURL',
'NoOfOtherSpecialCharsInURL', 'NoOfImage', 'NoOfCSS',
'NoOfJS', 'NoOfSelfRef', 'NoOfExternalRef'
]
```

```
class Discretizer(BaseEstimator, TransformerMixin):
    def __init__(self, features_to_discretize, n_bins=3,
labels=['Low', 'Medium', 'High']):
        self.features_to_discretize = features_to_discretize
        self.n_bins = n_bins
        self.labels = labels

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_copy = X.copy()
        for feature in self.features_to_discretize:
            X_copy[feature + '_Category'] =
pd.cut(X_copy[feature], bins=self.n_bins, labels=self.labels)
        X_copy =
X_copy.drop(columns=self.features_to_discretize)
        return X_copy
```

class digunakan agar *Discretizer* dapat digunakan di *pipeline.*

## Preprocessing

1. Feature Scaling
   Ensure that numerical features have similar scales. Common techniques include Min-Max scaling (scaling to a specific range) or standardization (mean-centered, unit variance).

```
class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self, numerical_features):
        self.numerical_features = numerical_features
        self.scaler = StandardScaler()
```

```
    def fit(self, X, y=None):
        self.scaler.fit(X[self.numerical_features])
        return self

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

*Feature scaling* digunakan menggunakan *StandardScaler* dari *sklearn library. Features* yang di *scale* adalah numerical features yang tidak di *discretize.*

2. Encoding Categorical Variables
   Machine learning models typically work with numerical data, so categorical variables need to be encoded. This can be done using one-hot encoding, label encoding, or more advanced methods like target encoding.

```
class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, categorical_features):
        self.categorical_features = categorical_features
        self.encoder = OneHotEncoder(sparse_output=False,
handle_unknown='ignore')

    def fit(self, X, y=None):
        self.encoder.fit(X[self.categorical_features])
        self.encoded_feature_names =
self.encoder.get_feature_names_out(self.categorical_features)
        return self

    def transform(self, X):
        X_encoded = X.copy()
        encoded_data =
self.encoder.transform(X_encoded[self.categorical_features])
        encoded_df = pd.DataFrame(encoded_data,
columns=self.encoded_feature_names, index=X_encoded.index)
        X_encoded =
X_encoded.drop(columns=self.categorical_features)
        X_encoded = pd.concat([X_encoded, encoded_df], axis=1)
        return X_encoded
```

Metode *encoding* yang digunakan adalah *OneHotEncoder* dari sklearn. Metode ini akan mengambil *categorical features* dan membuat *features* baru sesuai dan sebanyak dengan unique values dari *categorical features* tersebut. Setiap *features* tersebut akan bernilai biner (0, 1) sehingga metode ini akan mengubah *categorical features* menjadi *binary features*.

3. Handling Imbalanced Classes
   If dealing with imbalanced classes in a binary classification task, apply techniques such as oversampling, undersampling, or using different evaluation metrics to address class imbalance.

```
class ImbalanceHandler(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.smote = SMOTE(random_state=42)

    def fit(self, X, y):
        self.smote.fit_resample(X, y)
        return self

    def transform(self, X):
        return X
```

   *Imbalance Handling* dilakukan dengan metode *oversampling* menggunakan SMOTE dari library *imblearn*. SMOTE (Synthetic Minority Over-sampling Technique) bekerja dengan membuat sampel sintetik dari kelas yang minoritas. Dengan SMOTE, sampel dari kelas minoritas akan terus ditingkatkan sampai data seimbang.

# Hasil Prediksi

Menggunakan X_val kecil (0.001 dari dataset)

## KNN

- Hasil prediksi X_val menggunakan model KNN scratch
  - Akurasi KNN: 0.9716312056737588
  - Waktu yang dibutuhkan: 2 menit

- Hasil prediksi X_val menggunakan model KNN SCKit
  - Akurasi KNN SCKit: 0.96453900709219
  - Waktu yang dibutuhkan: < 1 detik

### Naive Bayes

- Hasil prediksi X_val menggunakan model nb scratch
    - Akurasi Naive Bayes: 0.8794326241134752
    - Waktu yang dibutuhkan: < 1 detik

- Hasil prediksi X_val menggunakan model nb SCKit
    - Akurasi KNN SCKit: 0.9645390070921985
    - Waktu yang dibutuhkan: < 1 detik

Kesimpulan : Algoritma KNN menghasilkan akurasi yang lebih baik, namun memakan waktu yang jauh lebih lama dibanding algoritma Gaussian Naive Bayes

## Kontribusi

| NIM | Nama | Tugas |
|---|---|---|
| 18222014 | Rashid May | • Membuat cover<br>• Merapikan dokumen |
| 18222018 | Lutfi Khairul Amal | • Mengimplementasikan model Naive Bayes serta menjelaskannya di laporan |
| 18222024 | Filbert Fuvian | • Mengimplementasikan data cleaning & preprocessing serta menjelaskannya di laporan<br>• Mengimplementasikan model KNN serta menjelaskannya di laporan |
| 18222038 | Qady Zakka Raymaula | • Menjelaskan hasil prediksi<br>• Membuat daftar isi |

## Referensi


- 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