Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Dibuat oleh

Kelompok 37

18222034	Christoper Daniel
18222042	Eldwin Pradipta
18222076	Theo Livius Natael
18222098	Muhammad Ridho Rabbani

Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

BAB I
IMPLEMENTASI ALGORITMA

1.1 Algoritma KNN

Algoritma K-Nearest Neighbors (KNN) adalah algoritma machine learning yang bekerja berdasarkan prinsip "data yang mirip akan berdekatan" dan biasanya algoritma ini digunakan untuk melakukan klasifikasi dan regresi. Algoritma dimulai dengan inisiasi parameter utama yaitu jumlah tetangga terdekat yang nantinya akan digunakan untuk melakukan prediksi dan metrik jarak untuk mengukur kedekatan (euclidean).

Implementasi algoritma KNN di tugas besar 2 ini dapat dijelaskan pada poin-poin berikut:

1. Inisiasi kelas (__init__)

   Kelas KNN memiliki 2 parameter, yaitu **`k`** dan **`distance_metric`**. **`k`** adalah blablabla, sedangkan **`distance_metric`** adalah metrik jarak antara data point.

2. Tahap preprocess data (preprocess)

   Preprocess data dilakukan pada setiap fungsi untuk memastikan bahwa setiap input adalah array 1 dimensi jika input tersebut *sparse* (ditentukan dengan fungsi *issparse()*).

3. Fungsi euclidean_distance()

   Fungsi ini memiliki 2 parameter yaitu x1 dan x2. Parameter tersebut menerima data yang sudah di-*preprocess*. Fungsi ini mengembalikan jarak euclidean antara 2 data point berdasarkan rumus seperti berikut,

   $$\sqrt{\left(x_1 - x_2\right)^2}$$

4. Fungsi manhattan_distance()

   Fungsi ini memiliki 2 parameter yaitu x1 dan x2. Parameter tersebut menerima data yang sudah di-*preprocess*. Fungsi ini mengembalikan jarak manhattan antara 2 data point berdasarkan rumus seperti berikut,

   $$\left|x_1 - x_2\right|$$

5. Fungsi minkowski_distance()

Fungsi ini memiliki 3 parameter yaitu x1, x2, dan p. Parameter tersebut menerima data yang sudah di-*preprocess*. Fungsi ini mengembalikan jarak manhattan antara 2 data point berdasarkan rumus seperti berikut,

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

6. Training model (fit function)

   Fungsi ini mengkonversi input menjadi *dense* jika input tersebut *sparse* (ditentukan dengan fungsi issparse())

7. Predict dan predict_single

   Fungsi predict() mengembalikan array predictions yang berisi data point yang di-predict dengan memanggil fungsi predict_single(). Fungsi predict_single() menghitung jarak antara data point, metrik jarak tergantung dengan input antara 'euclidean', 'manhattan', atau 'minkowski'. Setelah itu, ambil tetangga terdekat sejumlah *k.* Akhirnya mengembalikan hasil akhir yang ditentukan berdasarkan voting mayoritas

8. Fungsi score

   Fungsi ini mengembalikan skor akurasi yaitu proporsi prediksi yang benar terhadap seluruh data yang diuji

```
class KNN:
    def __init__(self, k=5, distance_metric='euclidean'):
        self.k = k
        self.distance_metric = distance_metric
        self.X_train = None
        self.y_train = None

    def euclidean_distance(self, x1, x2):
        # Ensure inputs are 1D arrays
        x1 = x1.toarray().flatten() if sp.issparse(x1) else x1
        x2 = x2.toarray().flatten() if sp.issparse(x2) else x2
        return np.sqrt(np.sum((x1 - x2)**2))

    def manhattan_distance(self, x1, x2):
        x1 = x1.toarray().flatten() if sp.issparse(x1) else x1
        x2 = x2.toarray().flatten() if sp.issparse(x2) else x2
        return np.sum(np.abs(x1 - x2))

    def minkowski_distance(self, x1, x2, p=3):
        x1 = x1.toarray().flatten() if sp.issparse(x1) else x1
        x2 = x2.toarray().flatten() if sp.issparse(x2) else x2
```

```
        return np.power(np.sum(np.power(np.abs(x1 - x2), p)), 1/p)

    def fit(self, X, y):
        # Convert to dense if sparse
        self.X_train = X.toarray() if sp.issparse(X) else X
        self.y_train = y.reset_index(drop=True)

    def predict(self, X):
        # Convert to dense if sparse
        X_dense = X.toarray() if sp.issparse(X) else X
        predictions = [self._predict_single(x) for x in X_dense]
        return predictions

    def _predict_single(self, x):
        # Calculate distances
        if self.distance_metric == 'euclidean':
            distances = [self.euclidean_distance(x, x_train) for x_train in
self.X_train]
        elif self.distance_metric == 'manhattan':
            distances = [self.manhattan_distance(x, x_train) for x_train in
self.X_train]
        elif self.distance_metric == 'minkowski':
            distances = [self.minkowski_distance(x, x_train) for x_train in
self.X_train]

        # Get k nearest neighbors
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # Majority vote
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)
```

1.2 Algoritma Naive Bayes

Algoritma Naive Bayes adalah algoritma klasifikasi probabilitas yang dibuat berdasarkan teorema Bayes,

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

dimana,

$P(C_k|X)$: probabilitas kelas C_k diberikan data X

$P(X|C_k)$: probabilitas data X terjadi jika data berasal dari kelas C_k

$P(C_k)$: probabilitas kelas C_k

$P(X)$: probabilitas data X muncul

Di algoritma yang kami buat, kami juga menggunakan teori Distribusi Gaussian yang merupakan distribusi probabilitas yang biasanya digunakan untuk menggambarkan fenomena alami, misalnya distribusi data berat badan dan banyak macam fenomena alami lainnya. Berikut rumus distribusi Gaussian standar,

$$P(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \times e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

dimana,

$P(X)$: distribusi probabilitas

σ^2 : *variance*

μ : *mean*

Implementasi algoritma Naive Bayes di tugas besar 2 ini dapat dijelaskan pada poin-poin berikut:

1. Inisiasi kelas (__init__)

   Bertujuan untuk melakukan inisiasi NaiveBayes dengan parameter var_smoothing sehingga di awal konstruktor akan menerima parameter berupa var_smoothing = 1e-9 untuk menghindari ketidakstabilan numerik.

2. Training model (fit)

   Bertujuan untuk menghitung parameter Gaussian dan probabilitas prior untuk setiap kelas. Untuk setiap kelas c akan dihitung rata-rata setiap fitur berdasarkan data dan menghitung variance dengan mengurangi rata-rata kuadrat data dengan kuadrat rata-rata data. Variance minimal harus sebesar var_smoothing.

3. Melakukan prediksi untuk banyak sampel (predict)

   Bertujuan untuk memprediksi kelas dari data baru. Jika data merupakan dataset sparse, maka akan dikonversi ke array dulu. Kemudian akan dihitung probabilitas log untuk tiap kelas dengan formula Gaussian yang sudah disesuaikan sebagai berikut,

$$log\ P(X|C) = \Sigma(-\frac{1}{2}log(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}).$$

Kelas dengan probabilitas log tertinggi akan dipilih sebagai hasil prediksi.

4. Mengevaluasi model (score)

   Bertujuan untuk menghitung akurasi model. Accuracy_score digunakan untuk membandingkan prediksi dengan label sebenarnya.

```
class NaiveBayes:
    def __init__(self, var_smoothing=1e-9):
        self.var_smoothing = var_smoothing

    def fit(self, X, y):
        self.classes = np.unique(y)
        n_classes = len(self.classes)
        n_features = X.shape[1]

        self.mean = np.zeros((n_classes, n_features))
        self.var = np.zeros((n_classes, n_features))
        self.priors = np.zeros(n_classes)

        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            self.mean[i] = np.array(X_c.mean(axis=0)).ravel()
            X_c_squared = X_c.copy()
            X_c_squared.data **= 2
            self.var[i] = np.array(X_c_squared.mean(axis=0)).ravel() -
self.mean[i] ** 2
            self.var[i] = np.maximum(self.var[i], self.var_smoothing)
            self.priors[i] = X_c.shape[0] / X.shape[0]

        return self

    def predict(self, X):
        X = X.toarray()
        y_pred = []
        for i in range(X.shape[0]):
            log_probs = []
            for j in range(len(self.classes)):
                log_prob = np.sum(-0.5 * np.log(2 * np.pi * self.var[j])
                                  -0.5 * ((X[i] - self.mean[j]) ** 2) /
self.var[j])
                log_prob += np.log(self.priors[j])
                log_probs.append(log_prob)
            y_pred.append(self.classes[np.argmax(log_probs)])

        return np.array(y_pred)

    def score(self, X, y):
        return accuracy_score(y, self.predict(X))
```

BAB II
CLEANING & PREPROCESSING

2.1 Data Cleaning

Ada beberapa cara dalam pelaksanaan data cleaning dengan tujuan untuk menjaga kekonsistenan data dan menjaga kebenaran data dalam model, yaitu:

2.1.1 Handling Missing Data

Mengatasi data yang hilang adalah salah satu proses yang penting utnuk menjaga akurasi dari model yang akan dibuat. Ada beberapa cara yang dapat digunakan untuk mengatasi masalah data - data yang hilang. Cara yang kelompok kami pilih untuk mengatasi data yang hilang adalah *data imputation*.

Imputasi data adalah metode yang dapat mengatasi data yang hilang dengan menggantikan data itu dengan nilai rata-rata, modus, atau median. Kami memilih metode ini karena metode ini efektif untuk menggantikan data hilang yang random sambil menjaga akurasi dari dataset. Kami menggunakan nilai rata - rata untuk menggantikan nilai yang hilang di data kami

2.1.2 Dealing With Outliers

Outliers adalah data yang nilainya sangat berbeda secara signifikan dari data lain hingga mengganggu akurasi dari model. Mengatasi data outlier juga merupakan salah satu proses yang penting dalam *data cleaning*. Ada beberapa cara yang dapat digunakan untuk mengatasi data dengan nilai outlier. Cara yang kelompok kami pilih untuk mengatasi *outlier* adalah dengan *clipping.*

Clipping adalah metode untuk mengatasi data *outlier* dengan menetapkan batasan maksimum dan minimum data sehingga perbedaan data tidak signifikan sembari menjaga akurasi data. Kami memilih metode ini karena metode ini simpel, efektif, dan tetap menjaga akurasi dari model.

2.1.3 Remove Duplicates

Duplicates adalah baris dalam tabel yang nilainya sama persis dengan baris lain. Hal ini dapat terjadi karena banyak alasan, seperti kesalahan dalam menginput data. Data duplikat dapat merusak akurasi model dan integritas data, sehingga penting untuk mengambil langkah

yang dapat mengatasi masalah duplikasi data. Karena itu data duplikat harus dihapus dari dataset.

2.1.4 Feature Engineering

Seringkali, fitur-fitur yang telah tersedia pada dataset masih belum cukup untuk membuat model yang akurat dan baik. Maka, perlu dilakukan beberapa modifikasi fitur untuk meningkatkan kemampuan model dalam membaca pola pada data. Berikut adalah perubahan yang kami lakukan terhadap beberapa fitur dalam data.

.

```
class featureCreator(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.numeric_features = None

    def fit(self, X, y=None):
        if isinstance(X, pd.DataFrame):
            self.numeric_features = X.select_dtypes(include=['int64',
'float64']).columns
        else:
            self.numeric_features = np.arange(X.shape[1])
        return self

    def transform(self, X):
        X_new = X.copy()

        if 'URLLength' in X_new.columns and 'DomainLength' in
X_new.columns:
            X_new['URL_Domain_Ratio'] = X_new['URLLength'] /
(X_new['DomainLength'] + 1)

        if 'NoOfLettersInURL' in X_new.columns and 'NoOfDegitsInURL' in
X_new.columns:
            X_new['Letter_Digit_Ratio'] = X_new['NoOfLettersInURL'] /
(X_new['NoOfDegitsInURL'] + 1)

        special_char_cols = ['NoOfEqualsInURL', 'NoOfQMarkInURL',
'NoOfAmpersandInURL', 'NoOfOtherSpecialCharsInURL']
        if all(col in X_new.columns for col in special_char_cols):
            X_new['Total_Special_Chars'] =
X_new[special_char_cols].sum(axis=1)

        if 'IsHTTPS' in X_new.columns and 'HasFavicon' in X_new.columns:
            X_new['Security_Score'] = X_new['IsHTTPS'] +
X_new['HasFavicon']

        form_cols = ['HasExternalFormSubmit', 'HasSubmitButton',
'HasHiddenFields', 'HasPasswordField']
```

```
        if all(col in X_new.columns for col in form_cols):
            X_new['Form_Complexity'] = X_new[form_cols].sum(axis=1)

        return X_new.values if isinstance(X, np.ndarray) else X_new
```

Kelompok kami melakukan modifikasi berupa menambahkan fitur-fitur baru pada data, yaitu “URL_Domain_Ratio”, “Letter_Digit_Ratio”, “Total_Special_Chars”, “Security_Score”, “Form_Complexity” . Dengan fitur baru tersebut, model yang dibuat akan memiliki akurasi yang lebih baik.

2.2 Data Preprocessing

Pemrosesan data pada tugas ini memanfaatkan pipeline serta menggunakan BaseEstimator dan TransformerMixin dengan tahapan sebagai berikut:

2.2.1 Feature Scaling

Feature scaling dilakukan untuk memastikan semua fitur numerik memiliki skala yang sama, sehingga semua fitur dapat berkontribusi secara setara dalam melatih model. Ada beberapa metode yang dapat digunakan untuk feature scaling, salah satu metode yang kelompok kami gunakan adalah *log transformation.* Berikut adalah implementasi dari kelompok kami :

```
class featureScaler(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.numerical_cols = [col for col in X.columns
                               if X[col].dtype != 'object' and
X[col].nunique() > 2 and col != "label"]
        return self

    def transform(self, X):
        X_transformed = X.copy()
        for col in self.numerical_cols:
            if (X[col] <= 0).any():
                X_transformed[col] = X[col] + 1 - X[col].min()
            X_transformed[col] = np.log(X_transformed[col])
        return X_transformed.values
```

Teknik *log transformation* membantu dalam menstabilisasi variasi nilai dalam dataset.

2.2.2 Feature Encoding

Feature encoding adalah proses mengubah nilai data kategorikal pada fitur tertentu menjadi suatu nilai numerik agar dapat dijadikan input untuk melatih model. Ada beberapa metode yang dapat digunakan untuk melakukan *feature encoding*, seperti *label encoding, one-hot encoding,* dan *target encoding*. Kelompok kami memilih menggunakan metode *one-hot encoding,* karena data kategorikal yang ada pada dataset berupa data ordinal. Berikut implementasi one-hot encoding kami :

```
class featureEncoder(BaseEstimator, TransformerMixin):
  def __init__(self):
    self.encoder = OneHotEncoder(handle_unknown='ignore', sparse_output =
False).set_output(transform = 'pandas')

  def fit(self, X, y=None):
    data = X.copy()
    categorical_cols = data.select_dtypes(include=['object',
'category']).columns
    self.encoder.fit(data[categorical_cols])
    return self

  def transform(self, X):
    data = X.copy()
    categorical_cols = data.select_dtypes(include=['object',
'category']).columns

    encoded_data = self.encoder.transform(data[categorical_cols])

    initial_data = X.drop(columns=categorical_cols)
    encoded_df = pd.concat([initial_data, encoded_data], axis=1)
    return encoded_df
```

BAB III
EVALUASI

3.1 Evaluasi KNN

Hasil akurasi dari model KNN yang dibuat memiliki hasil akurasi yang cukup baik. Nilai ini didapati karena KNN merupakan model yang sangat baik dalam mengenal pola pada data dan mendeteksi *intrusion.* Tidak hanya itu dataset yang dipakai juga non-linear, dan *noisy* sehingga lebih cocok untuk KNN

3.2 Evaluasi Naive Bayes

Hasil akurasi dari model Naive Bayes yang kami buat adalah 11% yang berarti akurasi model tidak akurat. Hal ini dapat terjadi karena mungkin hipotesis fungsi dari model naive Bayes yang menganggap bahwa semua fitur independent tidak cocok dengan dataset. Sehingga hasilnya tidak memuaskan. Selain itu, mungkin juga model yang dibuat tidak akurat karena metode yang digunakan untuk membersihkan dan menyiapkan data masih belum sempurna.

Yang sklearn ngecrash google collan jajaja

BAB IV
KESIMPULAN

4.1 Kesimpulan

Secara keseluruhan, algoritma KNN memiliki hasil yang lebih baik jika dibandingkan dengan algoritma Naive Bayes. Algoritma Naive Bayes tidak terlalu cocok digunakan dalam dataset dengan dimensi jumlah data yang sangat besar. Namun, algoritma KNN memiliki runtime yang jauh lebih lama jika dibandingkan dengan Naive Bayes sehingga bisa dikatakan bahwa algoritma KNN tidak terlalu efektif dari segi runtime.

4.2 Saran

- Melakukan proses pembersihan dan prapemrosesan data secara lebih baik dan cermat.
- Memproses data secara kelompok/*batch* untuk mengoptimalkan penggunaan memori
- Menggunakan lingkungan pengembangan selain Google Collab (gratis) jika proses membutuhkan sumber daya besar, karena rentan *crash* (berdasarkan percobaan), sumber daya kurang memadai, dan waktu runtime terbatas.

BAB V
PEMBAGIAN TUGAS

Nama (NIM)	Kontribusi
Christoper Daniel (18222034)	Membuat data cleaning dan Naive Bayes
Eldwin Pradipta (18222042)	Membuat preprocess pipeline, Knn, dan Naive Bayes
Theo Livius Natael (18222076)	Mengisi dokumen BAB II dan BAB III
Muhammad Ridho R. (18222098)	Mengimplementasi KNN dan mendokumentasi pada dokumen