IF3170 Intelegensi Artifisial

Tugas Besar Intelegensi Artifisial Implementasi Algoritma Pembelajaran Mesin

Oleh:

Jimly Nur Arif	(13522123)
Yosef Rafael Joshua	(13522133)
Samy Muhammad Haikal	(13522151)
Muhammad Roihan	(13522152)

PROGRAM STUDI INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

IMPLEMENTASI ALGORITMA

A. KNN

K-Nearest Neighbors (KNN) adalah algoritma klasifikasi yang sederhana namun kuat. Algoritma ini bekerja dengan cara mengklasifikasikan data baru berdasarkan label mayoritas dari k tetangga terdekat yang paling mirip dengan data tersebut. Berikut adalah prinsip dasar KNN:

1. **Menentukan K**: Tentukan jumlah tetangga terdekat (k) yang akan digunakan untuk menentukan kelas data baru.
2. **Menghitung Jarak**: Hitung jarak antara data baru dengan semua data pelatihan. Jarak ini biasanya dihitung menggunakan metrik seperti **Euclidean**, **Manhattan**, atau **Minkowski**.
3. **Menemukan Tetangga Terdekat**: Pilih k data pelatihan yang jaraknya paling dekat dengan data baru.
4. **Voting Mayoritas**: Tentukan kelas data baru dengan memilih kelas yang paling sering muncul di antara k tetangga terdekat (voting mayoritas).

Berikut penjelasan mengenai kelas yang kami buat

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

- k: Jumlah tetangga terdekat yang akan dipertimbangkan untuk prediksi.
- metric: Metrik jarak yang digunakan (default: euclidean).
- X_train: Menyimpan fitur dari data pelatihan.
- y_train: Menyimpan label dari data pelatihan.

```
def _calculate_distances(self, point):
```

Fungsi ini menghitung jarak antara satu titik data (point) dan seluruh data pelatihan (X_train).
Metrik jarak yang didukung:

- **Euclidean Distance**:
  $$d = \sqrt{\sum(x_i - y_i)^2}$$
- **Manhattan Distance**:
  $$d = \sum |x_i - y_i|$$
- **Minkowski Distance**:
  $$d = (\sum |x_i - y_i|^p)^{\frac{1}{p}}$$

Jika data yang digunakan berupa matriks sparse (*sparse matrix*), data akan diubah menjadi format array terlebih dahulu.

```
def fit(self, X, y):
```

Fungsi ini menyimpan data pelatihan (X dan y) di dalam atribut self.X_train dan self.y_train.

```
def predict(self, X):
```

Fungsi ini membuat prediksi untuk data baru X berdasarkan data pelatihan.
Langkah-langkah utama:

1. Hitung jarak setiap titik data di X terhadap semua data di X_train menggunakan _calculate_distances.
2. Temukan indeks dari k tetangga terdekat menggunakan np.argsort(distances)[:self.k].
3. Ambil label dari tetangga terdekat (nearest_labels) dan gunakan metode mayoritas untuk menentukan prediksi:

B. Naive Bayes

Naive Bayes adalah algoritma unsupervised learning. Naive Bayes memiliki hipotesis berupa model probabilitas yang digunakan untuk menghitung probabilitas suatu kelas berdasarkan fitur tertentu.

Ada dua hal penting pada Naive Bayes.

Yang pertama adalah menghitung probabilitas suatu kelas C berdasarkan fitur X

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

Dengan:
P(C I X) : probabilitas suatu data X termasuk ke dalam kelas C
P(X I C) : probabilitas fitur X jika kelas C
P(C) : probabilitas kelas C
P(X) : probabilitas fitur X

Yang kedua adalah asumsi independensi, yaitu Naive Bayes mengasumsikan bahwa semua fitur dalam data X = {x1, x2, … , xn} independen. Oleh karena itu berlaku:

$$P(X|C) = P(x_1|C) \cdot P(x_2|C) \cdot \cdots \cdot P(x_n|C)$$

Berikut penjelasan kelas yang kami buat

```
def __init__(self):
    self.classes = None
    self.mean = None
    self.variance = None
    self.prior = None
```

- Classes : untuk menyimpan daftar kelas unik dari target 'y'
- Mean : menyimpan nilai rata-rata untuk setiap fitur di masing-masing kelas
- Variance : menyimpan nilai variansi untuk setiap fitur di masing-masing kelas
- Prior : menyimpan probabilitas prior untuk setiap kelas

```
def fit(self, X, y):
    if sparse.issparse(X):
        X = X.toarray()
    self.classes = np.unique(y)
    n_features = X.shape[1]
    self.mean = np.zeros((len(self.classes), n_features))
    self.variance = np.zeros((len(self.classes), n_features))
    self.prior = np.zeros(len(self.classes))
    epsilon = 1e-9
    for idx, c in enumerate(self.classes):
        X_c = X[y == c]
        self.mean[idx, :] = X_c.mean(axis=0)
        self.variance[idx, :] = X_c.var(axis=0) + epsilon
        self.prior[idx] = X_c.shape[0] / X.shape[0]
```

Ini adalah kelas untuk melatih model dengan data x (fitur) dan y (label target).

```
def _gaussian_pdf(self, class_idx, x):
    mean = self.mean[class_idx]
    variance = self.variance[class_idx]
    numerator = np.exp(-((x - mean) ** 2) / (2 * variance))
    denominator = np.sqrt(2 * np.pi * variance)
    return numerator / denominator
```

Ini adalah kelas untuk menghitung probabilitas densitas Gaussian untuk sebuah nilai fitur pada kelas.

```
def _class_posterior(self, x):
    posteriors = []
    for idx, c in enumerate(self.classes):
        prior = np.log(self.prior[idx])
        conditional = np.sum(np.log(self._gaussian_pdf(idx, x)))
        posterior = prior + conditional
        posteriors.append(posterior)
    return self.classes[np.argmax(posteriors)]
```

Ini adalah kelas untuk menghitung probabilitas posterior untuk setiap kelas dengan nilai posterior terbesar.

```
def predict(self, X):
    if sparse.issparse(X):
        X = X.toarray()
    return np.array([self._class_posterior(x) for x in X])
```

Kelas ini digunakan untuk memprediksi label untuk data input X

```
def score(self, X, y):
    predictions = self.predict(X)
    y = y.values if isinstance(y, pd.Series) else np.array(y)
    return f1_score(y, predictions, average='macro')
```

Kelas ini digunakan untuk mengukur akurasi model berdasarkan prediksi.

Langkah-langkah utama:

1. Latih model (identifikasi kelas unik, hitung probabilitas prior, hitung statistik fitur)
2. Prediksi kelas (hitung likelihood setiap kelas, hitung posterior, pilih kelas yang sesuai)
3. Evaluasi performa model dengan data uji yang telah disiapkan.

C. ID3

ID3 atau Iterative Dichotomiser 3 adalah algoritma unsupervised learning yang memberikan hasil berupa pohon keputusan untuk klasifikasi data. Algoritma ini menggunakan entropi dan juga information gain untuk menghasilkan pohon keputusan

Entropi

$$H(S) = -\sum_{i=1}^{n} P_i \log_2 P_i$$

Information Gain

$$IG(A,S) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

Langkah-langkah utama:

- Pilih root node (semua data dianggap sebagai root node, hitung entropi awal dataset)
- Hitung information gain setiap atribut dan pilih atribut dengan information gain tertinggi.

- Pecah dataset berdasarkan nilai dari atribut yang dipilih
- Bangun cabang untuk setiap nilai atribut.
- Ulangi hingga pohon selesai.

Cleaning dan Preprocessing

1. **Handling Missing Value**
    - Untuk menangani nilai yang hilang (missing values), kami menggunakan SimpleImputer dengan metode median.
    - Pemilihan median didasarkan pada analisis distribusi fitur. Karena fitur memiliki distribusi yang skewed (tidak simetris), median lebih tahan terhadap outliers dibandingkan mean, sehingga memberikan estimasi yang lebih andal untuk nilai yang hilang.
2. **Handling outlier**
    - Outliers adalah nilai-nilai ekstrim yang secara signifikan menyimpang dari data lainnya. Nilai ini dapat memengaruhi analisis dan performa model.
    - Sama seperti pada missing values, outliers diganti dengan median dari masing-masing fitur. Pendekatan ini menjaga agar data tetap konsisten tanpa dipengaruhi oleh nilai-nilai ekstrim.
3. **Scaler**

    Untuk menangani fitur numerik, kami menggunakan library StandardScaler dari scikit-learn. Proses ini disebut standardization, yang bertujuan untuk mengubah skala data sehingga memiliki properti statistik tertentu.StandardScaler bekerja dengan menstandarkan data sehingga setiap fitur memiliki mean 0 dan standar deviasi 1.
4. **Encoder**

    Untuk menangani fitur kategorikal, kami menggunakan library OneHotEncoder dari scikit-learn. Teknik ini mengubah data kategorik menjadi format numerik agar dapat digunakan dalam algoritma pembelajaran mesin.
5. **Feature selection**

    Kelas FeatureSelector adalah transformer berbasis scikit-learn yang dirancang untuk melakukan seleksi fitur berdasarkan metode statistik tertentu, seperti f_classif (ANOVA F-value). Kelas ini memungkinkan pengguna untuk memilih sejumlah fitur terbaik (k) yang paling relevan dengan target variabel. Proses seleksi dilakukan melalui metode fit, yang menyesuaikan selektor fitur berdasarkan data input dan target, dan metode transform, yang mengembalikan subset fitur terpilih.

Hasil Prediksi

1. KNN

```
Scratch score :
0.4079250636010635
Library score :
0.3885944145034547
```

- Scratch score (0.4079) dan Library score (0.3886) menunjukkan bahwa performa model yang diimplementasikan secara manual (Scratch) sedikit lebih baik daripada model yang menggunakan KNeighborsClassifier dari scikit-learn (Library).
- Nilai F1-score mengukur keseimbangan antara presisi dan recall, di mana nilai lebih tinggi menunjukkan performa model yang lebih baik dalam hal deteksi kelas positif. Perbedaan kecil ini menunjukkan bahwa meskipun model dari library scikit-learn sudah dioptimalkan dan dipakai secara luas, implementasi manual pada model Scratch memberikan sedikit keuntungan dalam hal kinerja.
- Pada kasus ini, model Scratch mungkin lebih cocok karena kinerja yang lebih baik sedikit meskipun implementasi lebih sederhana, atau bisa jadi pengaturan k dan metric yang lebih tepat membantu menghasilkan hasil yang lebih optimal.

2. Naive Bayes

```
cratch score :
ipython-input-231-c21c052e551
  conditional = np.sum(np.log(
.16009386805269046
ibrary score :
.16009386805269046
```

- Scratch score (0.1601) dan Library score (0.1601) menunjukkan bahwa implementasi manual dari Naive Bayes yang dibuat sendiri (Scratch) dan Naive Bayes yang diambil dari scikit-learn (GaussianNB) memberikan hasil yang identik pada dataset ini.
-

- F1-score mengukur keseimbangan antara presisi dan recall, di mana nilai yang lebih tinggi menunjukkan model yang lebih baik dalam mendeteksi kelas positif secara umum. Namun, pada kasus ini, kedua model memberikan hasil yang sama, yang menunjukkan bahwa baik implementasi manual maupun library menghasilkan performa yang serupa dalam hal klasifikasi.