IF3170 INTELEGENSI BUATAN

IMPLEMENTASI ALGORITMA KNN DAN NAIVE-BAYES

Kelompok exhaust

Anggota :

Brian Kheng	13521049
Farizki Kurniawan	13521082
Frankie Huang	13521092
Dewana Gustavus Haraka Otang	13521173

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

Daftar Isi

Daftar Isi 1
1. Algoritma KNN 2
2. Algoritma Naive-Bayes 4
3. Perbandingan Algoritma Dengan Pustaka Scikit-Learn 10
4. Kaggle 11
5. Pembagian Kerja 13

1. Algoritma KNN

K-Nearest Neighbors (KNN) adalah algoritma yang digunakan untuk klasifikasi dan regresi. Algoritma ini termasuk dalam kategori *supervised learning*, di mana model belajar dari data pelatihan yang berlabel untuk melakukan prediksi terhadap data baru yang tidak berlabel. KNN bekerja berdasarkan prinsip bahwa objek yang mirip cenderung berada dalam kelompok yang sama.

Algoritma KNN bekerja dengan cara mengukur jarak antara suatu data yang akan diprediksi dengan data pelatihan yang telah diberi label. Objek baru diklasifikasikan berdasarkan mayoritas label dari k-neighbors terdekat.

Langkah-langkah algoritma KNN adalah sebagai berikut:

1) Mengukur jarak antara data yang akan diprediksi dengan setiap titik data pelatihan.
   Pada implementasi yang kami lakukan, jarak dihitung menggunakan Euclidean Distance dengan fungsi sebagai berikut

```
# Calculate Euclidean distance

def euclidean_distance(row1, row2):

    distance = 0.0

    for i in range(len(row1)-1):

        distance += (row1[i] - row2[i])**2 * feature_weights[i]

    return math.sqrt(distance)
```

2) Menentukan k-tetangga terdekat dari data yang akan diprediksi.
   Pada implementasi yang kami lakukan, tetangga terdekat didapat lewat fungsi sebagai berikut.

```
# Get nearest neighbors

def get_neighbors(train, test_row, num_neighbors):

    distances = []
```

```
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = []
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

3) Menentukan kategori mayoritas dari tetangga-tetangga tersebut. Kategori ini akan digunakan sebagai prediksi untuk data yang akan diprediksi.
   Pada implementasi yang dilakukan, hal ini dilakukan lewat fungsi sebagai berikut.

```
# Make a prediction
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

4) Lakukan prediksi untuk semua baris data

2. Algoritma Naive-Bayes

Naive Bayes adalah salah satu algoritma klasifikasi yang berbasis pada teorema Bayes. "Naive" dalam Naive Bayes merujuk pada asumsi yang dilakukan, yaitu bahwa fitur-fitur yang digunakan untuk klasifikasi adalah independen satu sama lain. Adapun langkah-langkah algoritma Naive Bayes adalah sebagai berikut.

Untuk setiap kemungkinan nilai target akan dihitung sebuah peluang nilai tersebut terjadi dan akan diambil nilai dengan peluang terbesar, perhitungan peluang sebuah nilai dihitung dengan rumus sebagai berikut.

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

Dimana $P(x)$ menyatakan peluang sebuah nilai terjadi dan $P(y|x)$ menyatakan peluang nilai y terjadi apabila nilai x terpenuhi.

Terdapat 2 buah observasi yang dapat digunakan untuk menyederhanakan rumus tersebut, yaitu:

1. Peluang variabel saling lepas $P(x_i|y, x_1, \dots, x_n) = P(x_i|y)$
2. Nilai $P(x_1, \dots, x_n)$ konstan

Dengan menggunakan observasi tersebut, rumus dapat disederhanakan menjadi sebagai berikut.

$$P(y \mid x_1, \dots, x_n) = P(y) \prod_{i=1}^{n} P(x_i|y)$$

Pada tugas ini, akan diprediksi nilai target yaitu price_range berdasarkan nilai kolom lain sehingga pada rumus variabel y menyatakan nilai price_range dan variabel-variabel x merupakan nilai kolom selain price_range, peluang yang perlu dicari untuk sebuah nilai price_range adalah:

$$P(price\_range \mid battery\_power, \dots, wifi)$$

Untuk perhitungan nilai $P(y)$ dihitung dengan

$$P(y) = \frac{count(y)}{count(price\_range)}$$

Dimana $count(y)$ menyatakan berapa banyak kemunculan nilai y pada kolom price_range pada data, dan $count(price\_range)$ menyatakan berapa banyak nilai yang terdapat pada kolom price_range pada data.

Untuk perhitungan nilai $P(x_i|y)$ akan terbagi menjadi 2 kasus yaitu kasus dimana x_i merupakan kolom kategorikal dan kasus dimana x_i merupakan kolom numerikal.

Untuk masing-masing kasus dapat dihitung peluang menggunakan rumus berikut.

1. Kolom kategorikal

$$P(x_i|y) = \frac{count(x_i, y)}{count(y)}$$

   Dimana $count(x_i, y)$ berarti berapa banyak data dengan nilai kolom adalah x_i dan nilai price_range adalah y, lalu $count(y)$ adalah berapa banyak kemunculan nilai y pada kolom price_range

2. Kolom numerikal

$$P(x_i|y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

   Disini rumus yang digunakan untuk mencari peluang pada kolom numerikal adalah Gaussian Naive Bayes yang memanfaatkan nilai distribusi normal dalam perhitungannya. Untuk menentukan nilai-nilai variabel pada rumus tersebut hanya digunakan data dengan nilai price_range = y dan hanya menggunakan memakai nilai-nilai pada kolom x_i

   Keterangan:

   σ : standar deviasi data
   μ : rata-rata data

π : konstanta dengan nilai 3.141592…

e : konstanta euler dengan nilai 2.718281828459…

Berikut adalah implementasi algoritma Naive Bayes dengan menggunakan bahasa python

```
# Naive Bayes processing

def preprocess(data: pd.DataFrame):

    # Occurence for all price range

    price_count = [len(data[data[target_column] == i]) for i in range(4)]


     # for each price range, what is the prob for (category, value, price
range) probability

    categorical_probs: dict[tuple[str, str, int], float] = dict()


    for category in categorical_columns:

        values: list[str] = list(set(data[category].values))

        for value in values:

            for price in price_range:

                        filtered_data = data[(data[category] == value) &
(data[target_column] == price)]

                count = len(filtered_data)

                prob = count / price_count[price]

                categorical_probs[(category, value, price)] = prob


    # statistic that will be used for calculating Gaussian Naive Bayes

    # dict value is dict[column, price range] : value

    mean: dict[tuple[str, int], float] = dict()

    std: dict[tuple[str, int], float] = dict()
```

```
    for column in numerical_columns:
        for price in price_range:
            filtered_data = data[data[target_column] == price][column]
            mean[(column, price)] = filtered_data.mean()
            std[(column, price)] = filtered_data.std()

    model = TrainingModel(price_count, categorical_probs, mean, std)
    return model
```

Pertama-tama akan dilakukan pre-processing data untuk mempersiapkan beberapa variabel yang digunakan pada rumus seperti jumlah kemunculan, rata-rata, dan standar deviasi

```
@dataclass
class TrainingModel:
    price_count: list[int]
    categorical_probs: dict[tuple[str, str, int], float]
    mean: dict[tuple[str, int], float]
    std: dict[tuple[str, int], float]
```

Pre-proccessing akan menghasilkan data dalam bentuk TrainingModel seperti diatas

```
def count_normal(value: float, mean: float, std: float):
    # function for calculating Gaussian Naive Bayes
    exponent = pow(value - mean, 2) / (2 * pow(std, 2))
    exp_value = exp(-exponent)
    denominator = std * sqrt(2 * pi)
    normal = exp_value / denominator
```

```
    return normal
```

Selanjutnya terdapat fungsi count_normal yang digunakan untuk menghitung nilai distribusi normal

```
def train_data(data: pd.DataFrame, model: TrainingModel):
    price_predictions: list[int] = []
    for i in range(len(data)):
        maximum_prob = 0
        max_prob_price = -1
        for price in price_range:
            # Initial probability
            price_prob = model.price_count[price] / len(data)
            prob = price_prob

            # Categorical probability
            for category in categorical_columns:
                    category_prob = model.categorical_probs[(category, data[category][i], price)]
                prob *= category_prob

            # Numerical probability
            for column in numerical_columns:
                tuple_data = (column, price)
                        numerical_prob = count_normal(data[column][i], model.mean[tuple_data], model.std[tuple_data])
                prob *= numerical_prob
```

```
            if prob > maximum_prob:
                maximum_prob = prob
                max_prob_price = price

        price_predictions.append(max_prob_price)

    training_result[target_column] = price_predictions
```

Selanjutnya terdapat sebuah fungsi train_data yang menerima masukan berupa data yang ingin di predict dan juga model yang sudah di training, nantinya fungsi akan menghitung prediksi nilai price_range menggunakan rumus yang sudah dijelaskan sebelumnya dan akan mengisi nilai tersebut pada data yang diberikan.

3. Perbandingan Algoritma Dengan Pustaka Scikit-Learn

Selain implementasi algoritma secara manual, dilakukan pula implementasi algoritma KNN dan Naive-Bayes menggunakan pustaka Scikit-Learn.

Berikut merupakan perbandingan akurasi dari algoritma yang digunakan

Algoritma	Akurasi
KNN – Manual	94.167%
KNN – Scikit-Learn	92.5%
Naive-Bayes – Manual	78.167%
Naive-Bayes – Scikit-Learn	78.167%

Dari perbandingan di atas, dapat dilihat bahwa akurasi untuk algoritma manual dan pustaka sama persis untuk algoritma Naive-Bayes dan berbeda sebesar 1.667% untuk algoritma KNN, dengan akurasi algoritma manual lebih tinggi dibandingkan dengan algoritma Scikit-Learn. Hal ini mungkin terjadi karena perbedaan dalam cara perhitungan jarak antara algoritma manual dan pustaka, dimana pada algoritma manual, cara perhitungan jarak telah diubah secara heuristik sehingga dapat menghasilkan akurasi yang lebih tinggi.

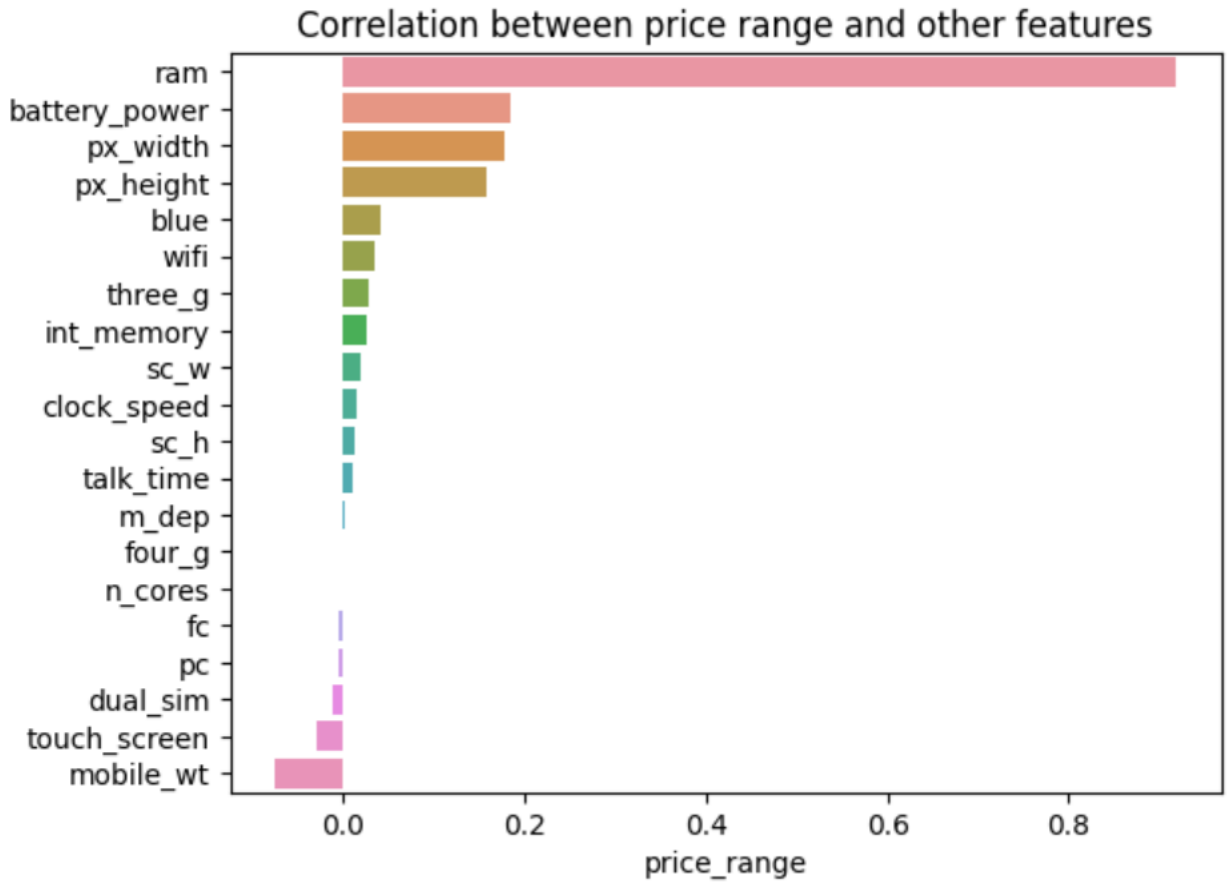
Dapat dilihat juga bahwa pada dataset ini, algoritma KNN menghasilkan akurasi yang lebih tinggi dibandingkan dengan algoritma Naive-Bayes. Hal ini mungkin disebabkan oleh asumsi pada Naive-Bayes, seperti asumsi bahwa atribut yang ada independen satu sama lain kurang akurat. Selain itu, banyaknya data yang lebih bersifat kategorikal juga dapat mempengaruhi kinerja algoritma Naive-Bayes.

4. Kaggle

Algoritma yang dikumpulkan pada Kaggle adalah KNN yang kami implementasikan dari scratch. KNN kami pilih sebagai opsi dikarenakan memiliki akurasi yang lebih tinggi dibandingkan Naive-Bayes pada kasus ini.

Langkah-Langkah pengerjaan:

1. Menentukan feature mana saja yang penting untuk diperhitungkan berdasarkan hasil analisis korelasi data.




```
# Define feature selection
#                    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19
feature_selection = [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Dapat dilihat, hanya 4 feature yang memiliki peran besar terhadap goal.

2. KNN kemudian dilakukan seperti biasanya, namun saat menentukan jarak (euclidean distance) masing-masing feature akan dikalikan dengan bobot yang dimilikinya. Sehingga, feature yang lebih berperan besar seperti ram akan memiliki bobot yang lebih besar.

```
# Define feature weights
#                 0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19
feature_weights = [1.5, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 1.5, 1.5, 7.5, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
```

3. Melakukan for loop untuk menentukan jumlah neighbor yang memiliki akurasi paling tinggi, dalam hal ini ditemukan angka 15 neighbor dengan akurasi 94.176%.
4. Menggabungkan data_train dengan data_validation sebelum melakukan prediksi terhadap data_test sehingga, data set menjadi lebih banyak.
5. Sisanya tinggal melakukan tuning (gacha) terhadap bobot dari masing-masing feature dan berharap akurasi menjadi semakin tinggi.

5. Pembagian Kerja

NIM	Nama	Bagian
13521049	Brian Kheng	Implementasi KNN
13521082	Farizki Kurniawan	Implementasi NB_scikit-learn
13521092	Frankie Huang	Implementasi KNN_scikit-learn
13521173	Dewana Gustavus Haraka Otang	Implementasi NB