

Tugas Besar

Pembelajaran Mesin 1 : Clustering

Kaenova Mahendra Auditama
1301190324

CII3C3-IF-43-02
Fakultas Informatika, S1 Informatika
Universitas Telkom

1 Pendahuluan

Tugas Besar pada Mata Kuliah Pembelajaran Mesin (CII3C3-IF-43-02) merupakan tugas pertama dari dua projek tugas yang ada. Pada tugas ini, saya diminta untuk membuat suatu sistem atau model yang dapat mengklasifikasi dari dataset yang disediakan. Data yang diberikan merupakan data pelanggan terhadap ketertarikan untuk memiliki kendaraan baru. Ada beberapa atribut dalam dataset tersebut, seperti *id*, *Jenis_Kelamin*, *Umur*, *SIM*, *Kode_Daerah*, *Sudah_Asuransi*, *Umur_Kendaraan*, *Kendaraan_Rusak*, *Premi*, *Kanal_Penjualan*, *Lama_Berlangganan*, *Tertarik*.

Dengan data-data tersebut saya diminta untuk membuat kelompok-kelompok atau meng*cluster* sehingga dapat terlihat pengelompokan datanya. Disini, saya menggunakan pendekatan *K-Means*. Pendekatan ini merupakan salah satu pendekatan dalam *machine learning* dalam kelompok *unsupervised learning*.

2 Formulasi Masalah

Pada kasus ini kami diberikan suatu dataset terkait ketertarikan pelanggan dengan beberapa atribut-atribut yang sudah disebutkan pada bagian 1. Dari sana, saya diminta untuk mengelompokkan data-data tersebut dengan menggunakan salah satu teknik dalam pembelajaran mesin *unsupervised learning*. Data yang diberikan sebesar 285.831 *records*. Dari sana saya bisa mengira akan ada beberapa data yang tidak lengkap ataupun salah, sehingga akan sangat dibutuhkan pra-pemrosesan data sebelum saya lanjutkan untuk melakukan pengelompokan data.

Ada beberapa pendekatan dalam membuat model pembelajaran mesin yang bertipe *unsupervised learning* untuk melakukan pengelompokan data. Beberapa model diantaranya ialah *Affinity Propagation*, *BIRCH*, *DBSCAN*, *K-Means*, *Mean Shift* yang biasa dipakai [Brownlee, 2020]. Dilihat dengan data sebesar itu, akhirnya saya memilih untuk menggunakan ***K-Means*** sebagai model untuk melakukan pengelompokan data.

3 Eksplorasi Data dan Pra-Pemrosesan Data

Dari 285.831 *records* dan atribut-atribut *id*, *Jenis_Kelamin*, *Umur*, *SIM*, *Kode_Daerah*, *Sudah_Asuransi*, *Umur_Kendaraan*, *Kendaraan_Rusak*, *Premi*, *Kanal_Penjualan*, *Lama_Berlangganan*, *Tertarik* saya

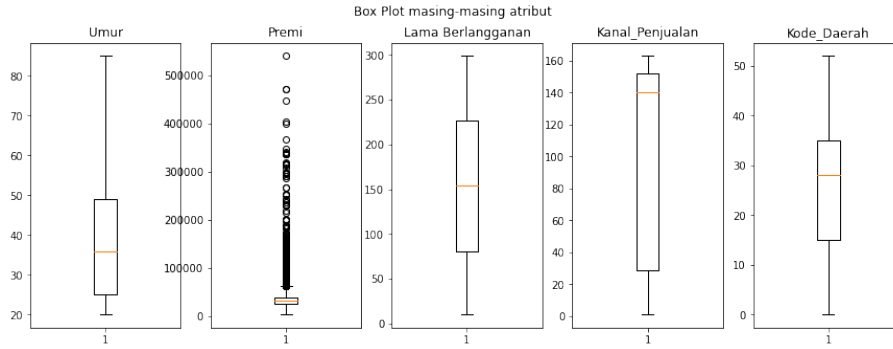


Figure 1: Boxplot setiap atribut sebelum dilakukan penghilangan outlier

melihat berbagai jenis tipe data. Ada beberapa data berjenis kategori, contohnya pada atribut *Jenis_Kelamin*, *SIM*, *Sudah_Asuransi*, *Kendaraan_Rusak*, *Tertarik*. *K-Means* merupakan salah satu model yang tidak baik dalam mengklusterisasi kelompok data berjenis kategorikal [Huá and Huong, 2012]. Sehingga atribut-atribut tersebut harus dihilangkan untuk sebelum selanjutnya diproses ke dalam model *K-Means*.

Dengan menghilangkan atribut-atribut tersebut, akhirnya tersisa atribut seperti *Umur*, *Premi*, *Lama_Berlangganan*, *Kanal_Penjualan*, *Kode_Daerah*. Data-data ini setelah saya periksa terdapat beberapa data yang tidak lengkap pada salah satu kolomnya, sehingga pada langkah selanjutnya saya melakukan penghilangan *records* yang memiliki, pada salah satu atributnya data yang kosong.

Setelah melakukan penghilangan data yang tidak lengkap, saya melakukan pemeriksaan terhadap data-data *outlier*. Pada *K-Means*, data-data outlier dapat menurunkan performa pengelompokan, sehingga outlier benar-benar harus dibersihkan [Patel and Mehta, 2011]. Setelah diperiksa pada ketiga atribut tersebut, dapat terlihat pada figure 1 bahwa atribut *Premi* memiliki *outlier* yang banyak. Saya menghilangkan nilai-nilai *outlier* tersebut menggunakan metode *interquartile range* yang dapat dirumuskan dengan rumus 1. Dengan penghilangan *outlier*, bisa dilihat bahwa data menjadi lebih dekat dengan boxplot yang terlihat pada figure 2 dan mengurangi besar penyebaran pada *outlier*.

$$\text{ValidData} \in (x \geq (Q_1 - 1.5 \cdot IQR)) \text{ and } (x \leq (Q_3 + 1.5 \cdot IQR)) \quad (1)$$

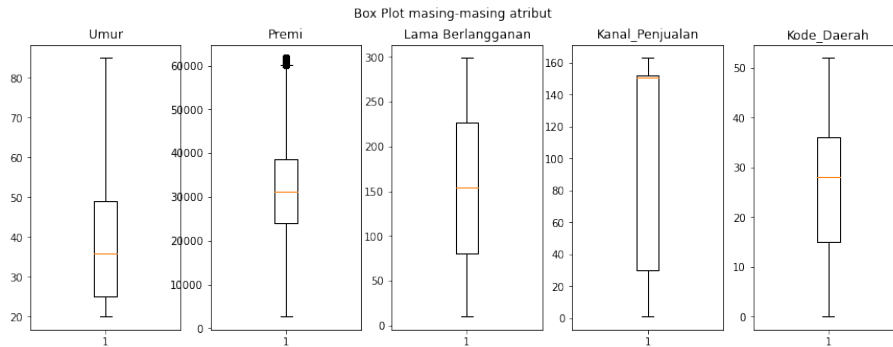


Figure 2: Boxplot setiap atribut setelah dilakukan penghilangan outlier

Setelah dilakukan semua pra pemrosesan dengan langkah-langkah di atas, data yang berjumlah 285.831 *records* diperkecil menjadi 166.396 *records*. Data-data tersebut saya simpan menjadi *file* baru. Sehingga secara lengkap, data yang akan dilakukan pengelompokan menjadi 166.396 *records* dengan atribut-atribut seperti *Umur*, *Premi*, dan *Lama_Berlangganan*.

Setelah dilakukan pemeriksaan outlier, saya melakukan eksplorasi korelasi untuk setiap atribut. Terlihat pada gambar 3 bahwa sebenarnya tidak ada korelasi yang kuat antar atribut. Pada akhirnya saya memilih korelasi terbesar yaitu pada atribut *Umur* dan *Kanal_Penjualan*.

	Id	Umur	Kode_Daerah	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
Id	1.000000	0.002691	0.000597	0.002643	-0.001621	0.001875	0.000203
Umur	0.002691	1.000000	0.044503	0.046519	-0.574807	-0.001055	0.108781
Kode_Daerah	0.000597	0.044503	1.000000	-0.004068	-0.044871	-0.003771	0.010484
Premi	0.002643	0.046519	-0.004068	1.000000	-0.105819	0.001831	0.019686
Kanal_Penjualan	-0.001621	-0.574807	-0.044871	-0.105819	1.000000	0.000017	-0.139186
Lama_Berlangganan	0.001875	-0.001055	-0.003771	0.001831	0.000017	1.000000	0.001819
Tertarik	0.000203	0.108781	0.010484	0.019686	-0.139186	0.001819	1.000000

Figure 3: Korelasi antar atribut

4 Pembangunan Model (Pemodelan)

K-Means merupakan salah satu algoritma pembelajaran mesin dalam ranah *unsupervised learning*. Algoritma ini berguna untuk membuat klusterisasi terhadap data-data yang ada. Data-data tersebut dapat berbentuk satu dimensi hingga banyak dimensi. Pada kasus ini saya membuat sebuah *library* agar algoritma *K-Means* ini dapat dijalankan pada banyak dimensi data.

Secara sederhana algoritma *K-Means* yang saya bangun adalah seperti ini:

Algorithm 1 Algoritma *K-Means*

Require: *k_value*, *max_step*, *convergence_threshold*, *data*

```

1: convergence  $\leftarrow$  False
2: step  $\leftarrow$  0
3: normalize_data  $\leftarrow$  min_max_normalization(data)
4: centroid  $\leftarrow$  initialize_centroids(data)
5: while (not convergence) and (step < max_step) do
6:   initial_point  $\leftarrow$  centroid
7:   distance  $\leftarrow$  calculate_euclidean(normalize_data, initial_point)
8:   cluster  $\leftarrow$  clustering(distance)
9:   new_point  $\leftarrow$  centroid_normalization(data, point, cluster)
10:  convergence  $\leftarrow$  convergence_check(initial_point, new_point, convergence_threshold)
11:  if convergence then
12:    point  $\leftarrow$  new_point
13:    break
14:  else
15:    point  $\leftarrow$  new_point
16:    step  $\leftarrow$  step + 1
17:  end if
18: end while
19: inertia  $\leftarrow$  calculate_inertia(data, cluster, point)
20: point  $\leftarrow$  min_max_denormalization(point, data)
21: return cluster, point, inertia

```

Berdasarkan algoritma 1 kita akan mendapatkan cluster pada setiap data yang diberikan, mendapatkan titik-titik *centroid*, dan mendapatkan nilai *inertia* yang akan digunakan untuk menentukan nilai *k* terbaik dari percobaan yang akan dijalankan.

Saya membangun algoritma ini menggunakan bahasa pemrograman python yang dapat dilihat melalui link berikut https://raw.githubusercontent.com/kaenova/Malin_Tubes1/main/

module/KMeans.py.

4.1 Inisialisasi Model

Sebelum menjalankan algoritma ini, haruslah diinisialisasikan dahulu parameter-parameter yang dibutuhkan. Pada algoritma 1 dibutuhkan beberapa parameter seperti *k_value*, *max_step*, *convergence_threshold*, *data* untuk menjalankan algoritma *K-Means* ini. *k_value* digunakan untuk memberikan berapa banyak cluster yang dibutuhkan. *max_step* digunakan untuk menentukan jumlah maksimal normalisasi centroid yang dilakukan. *convergence_threshold* digunakan sebagai batas besar perubahan yang dapat dinyatakan konvergen, semakin kecil nilai tersebut, maka akan dinyatakan konvergen ketika perubahan nilai tersebut sangatlah kecil. *data* adalah array yang berisi data-data yang akan dilakukan klusterisasi. Array tersebut berbentuk seperti `[[attr11, attr21, attr31], [attr12, attr22, attr32]]` yang dapat digambarkan. Setiap baris merupakan parent index dan setiap atribut merupakan child index¹.

Setelah dilakukan inisialisasi parameter, selanjutnya ada inisialisasi seperti menyiapkan *convergence* bernilai *False*, *step* bernilai 0, melakukan *min-max* normalisasi data seperti pada line 3 algoritma 1. Melakukan normalisasi data dapat mempercepat dalam perhitungan, serta menormalisasikan satuan yang tidak sama.

Dalam python, normalisasi dilakukan dengan cara seperti di bawah ini, saya menggunakan library scikit-learn module *MinMaxScaler* untuk melakukan normalisasi:

```
def __normalize_data__(self, data: np.array) -> np.array:
    """
    Fungsi ini digunakan untuk menormalisasikan data dengan menggunakan min-max scaling
    .
    Sehingga data berjenis dan bersatuan apapun data diproses dengan baik.
    """
    data = data.copy()
    for i in range(len(data[0])):
        col_arr = data[:, i]
        minmax = MinMaxScaler()
        normalize = minmax.fit_transform(col_arr.reshape(-1, 1)).reshape(1, -1)
        data[:, i] = normalize[0]

    return data
```

Pada tahap akhir algoritma, centroid akan dilakukan denormalisasi untuk mendapatkan nilai asli dari titik tersebut.

4.2 Inisialisasi Centroid

Hal terpenting pertama setelah menyiapkan parameter-parameter yang ditentukan, menaruh titik awal centroid. Penginisialisasian centroid ini sangatlah penting, beberapa metode seperti *random placement* ataupun menginisialisasikan dengan mengambil sampel dari data yang ada tidaklah optimal dalam penggunaan algoritma *K-Means*. Penginisialisasian dengan menggunakan metode *K-Means++* merupakan salah satu cara untuk mendapatkan posisi centroid yang optimal. Dengan algoritma ini kita juga memudahkan algoritma dan perhitungan kedepannya sehingga mempercepat pemrosesan pengklusteran [Arthur and Vassilvitskii, 2007].

Algoritma ini bekerja secara general dengan mencari jarak terjauh antara titik cluster yang sudah terinisialisai, sehingga akan mengecilkan kemungkinan dalam adanya kluster yang saling berdekatan. Saya mengimplementasikan algoritma tersebut menggunakan python dengan kode di bawah ini:

¹Dapat digambarkan saat pemanggilan array seperti: `array[no_baris][atribut]`

```

def __initialize_centroids__(self, data:np.array, k:np.array) -> np.array:
'''
Fungsi ini digunakan untuk menginisialisasikan centroid. Menggunakan algoritma k-
means++
referensi membantu: https://www.youtube.com/watch?v=HatwtJSsj5Q
'''
    centroids = []
    centroids.append( data[random.randrange(0, len(data))] )

    for i in range(1, k):
        min_dist = []
        for data_point in data:
            distance_data_point = []
            for point in centroids:
                distance_data_point.append(np.linalg.norm(data_point - point))
            min_dist.append(min(distance_data_point))

        probcum = sum(min_dist)
        prob_point = [value / probcum for value in min_dist]

        centroids.append(data[np.argmax(prob_point)])

    return np.array(centroids)

```

4.3 Perhitungan Jarak ke Centroid

Dalam *K-Means* dibutuhkan perhitungan jarak dari titik data ke centroid, saya mengimplementasikan hal ini pada algoritma 1 line 10. Disini saya menggunakan *Euclidean distance* dengan rumus multidimensional, hal ini digunakan karena data yang diberikan belum tentu data 2 dimensi [Tabak, 2004]. Rumus tersebut dapat dituliskan pada rumus 2 dimana p dan q merupakan titik data dengan jumlah dimensi atau atribut sebesar n .

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_i - q_i)^2 + \dots + (p_n - q_n)^2} \quad (2)$$

Saya mengimplementasikan perhitungan jarak ini menggunakan library dari numpy, karena kecepatan pemrosesan array yang dibutuhkan. Sehingga kode python yang digunakan seperti di bawah ini:

```

def __calculate_distance__(self, data:np.array, point: np.array) -> np.array:
'''
Fungsi ini akan menghitung setiap titik dengan point dan mengembalikan jarak dari
titik ke point
'''
    distance = np.zeros((len(data), len(point)))

    for i in range(len(data)):
        current_record = data[i]
        for j in range(len(point)):
            current_point = point[j]
            # numpy euclidean distance
            distance[i][j] = np.linalg.norm(current_point - current_record)

    return distance

```

4.4 Klusterisasi Data

Setelah dilakukan perhitungan jarak, haruslah ditentukan, pada setiap titik data masuk ke dalam centroid terdekat. Hal ini dilakukan dengan cara mencari nilai terkecil berdasarkan perhitungan jarak yang sudah dilakukan. Bisa dilihat hal ini dilakukan pada algoritma 1 line 11.

Saya mengimplementasikan hal ini pada python dengan kode di bawah:

```
def __clustering__(self, distance: np.array) -> np.array:
    '''
    Fungsi ini akan mengembalikan hasil clustering berdasarkan distance
    '''
    cluster = np.zeros(len(distance))
    for i in range(len(cluster)):
        cluster[i] = np.argmin(distance[i])
    return cluster
```

4.5 Perhitungan Ulang Centroid

Hal terpenting dalam algoritma *K-Means* adalah melakukan perhitungan ulang terhadap posisi centroid dalam data. Dalam algoritma 1 saya mengimplementasikan ini pada line 12. Dengan melakukan perhitungan ulang terhadap centroid, hasil klusterisasi akan semakin pasti, dan semakin banyak kita melakukan normalisasi centroid, semakin pasti juga hasil klusterisasinya. Pada algoritma *K-Means* perhitungan ulang ini dilakukan dengan menghitung rata-rata dari setiap cluster hingga titik centroid ini konvergen.

Saya mengimplementasikan dengan python seperti kode di bawah:

```
def __point_normalization__(self, data: np.array, point: np.array, cluster: np.array)
    -> (np.array, np.array):
    '''
    Fungsi ini digunakan untuk clustering dan normalisasi point
    '''
    new_point = np.zeros((len(point), len(point[0])))
    counter_array = np.zeros(len(point))
    for i in range(len(cluster)):
        new_point[int(cluster[i])] = new_point[int(cluster[i])] + data[i]
        counter_array[int(cluster[i])] += 1

    unique_on_cluster = np.unique(cluster)
    for i in range(len(point)):
        # nan handling
        if i not in unique_on_cluster:
            new_point[i] = point[i]
        else:
            new_point[i] = np.true_divide(new_point[i], counter_array[i])

    return new_point
```

4.6 Memeriksa Kekonvergenan

Untuk mengoptimalkan algoritma *K-Means* saya membuat pemeriksaan konvergen. Saya memberikan parameter batas konvergen. Dengan ini, saya tidak perlu menunggu hingga centroid tersebut tidak berubah. Dengan memasukkan nilai seperti $1e - 3$ mengartikan bahwa saya bisa mengatakan titik tersebut sudah konvergen ketika perubahan dari titik lama ke titik yang baru hanya kurang dari 0.001.

Saya mengimplementasikan dengan python seperti kode di bawah:

```

def __convergence_check__(self, points1: np.array, points2: np.array, threshold:
                                float) -> bool:
    '''
    Fungsi ini untuk mengecek convergence berdasarkan threshold yang dibuat.
    titik cluster pertama akan dibandingkan dengan titik cluster kedua.
    note: maybe i should use euclidean distance insted of menghitung satu-satu
    '''

    local_convergence = False
    normalize_threshold_positive, normalize_threshold_negative = 1 + threshold, 1
                                                - threshold

    points_counter = 0
    center = np.zeros(len(points1[0]))
    for i in range(len(points1)):
        current_first_point, current_second_point = points1[i], points2[i]
        distance_first_point, distance_second_point = np.linalg.norm(
                                                    current_first_point - center), np.
                                                    linalg.norm(current_second_point -
                                                    center)

        distance_threshold_positive = distance_first_point *
                                    normalize_threshold_positive
        distance_threshold_negative = distance_first_point *
                                    normalize_threshold_negative

        if distance_threshold_positive > distance_second_point and
            distance_threshold_negative <
            distance_second_point:

            points_counter += 1

    if points_counter == len(points1):
        local_convergence = True

    return local_convergence

```

4.7 Inertia

Inertia merupakan salah satu ukuran yang dapat digunakan untuk mengidentifikasi nilai k yang optimal dalam melakukan klusterisasi terutama dalam klusterisasi menggunakan *K-Means*. Perhitungan *inertia* dinyatakan dengan kuadrat dari total jarak titik data ke centroid klusternya [Brus, 2021]. Hal ini dapat dinyatakan rumus dengan x merupakan titik data ke i dan C_k merupakan titik centroid pada cluster yang sama dengan x_i .

$$\text{Inertia} = \sum_{i=1}^N (x_i - C_k)^2 \quad (3)$$

Saya mengimplementasikan dengan python seperti kode di bawah:

```

def __calculate_inertia__(self, data: np.array, cluster: np.array, points: np.array) -
    > np.array:
    '''
    K-Means: Inertia
    Inertia measures how well a dataset was clustered by K-Means. It is calculated by
    measuring the distance between each data point and its centroid, squaring this
    distance, and summing these squares across one cluster.
    ref: https://www.codecademy.com/learn/machine-learning/modules/dspath-clustering/
    cheatsheet
    '''

    inertia = 0

```

```
for i in range(len(data)):
    inertia += (np.linalg.norm(data[i] - points[int(cluster[i])]))**2

return inertia
```

5 Eksperimen

6 Kesimpulan

References

- [Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA. Society for Industrial and Applied Mathematics.
- [Brownlee, 2020] Brownlee, J. (2020). 10 Clustering Algorithms With Python.
- [Brus, 2021] Brus, P. (2021). Clustering: How to find hyperparameters using inertia.
- [Hu´n and Huong, 2012] Hu´n, H. X. and Huong, N. T. X. (2012). An extension of the k-means algorithm for mixed data. *Journal of Computer Science and Cybernetics*, 22(3):267–274.
- [Patel and Mehta, 2011] Patel, V. R. and Mehta, R. G. (2011). Impact of outlier removal and normalization approach in modified k-means clustering algorithm. *International Journal of Computer Science Issues (IJCSI)*, 8(5):331.
- [Tabak, 2004] Tabak, J. (2004). *Geometry : the language of space and form*. Facts On File, New York.