Machine Learning Cheatsheet : *Unsupervised Model (Clustering)*

(Konsep, Penerapan, Python Code Lengkap)

Disusun oleh Datasans
https://www.instagram.com/datasans.book/

Peringatan

Materi ini telah divalidasi dan semua syntax telah diuji coba menggunakan default engine google colab, namun bagaimanapun juga, ebook ini tidak luput dari kesalahan baik definisi, konten secara umum, maupun syntax. Segala masukkan dari pengguna sangat terbuka. DM kami di instagram @datasans.book

Himbauan

1. Tidak menjadikan ebook ini satu-satunya sumber pegangan, *cross check* dan validasi segala informasi dari sumber lain.
2. Tidak membagikan atau mencetaknya untuk diperbanyak dan dikomersialkan (cetak untuk pribadi dipersilahkan).
3. Disarankan untuk merekomendasikan langsung ke instagram @datasans.book jika temanmu berminat agar ilmu yang bermanfaat bisa tersebar semakin luas.

Daftar Isi

Bab 1. Pendahuluan

1.1. Pengertian Machine Learning

Machine Learning adalah bidang ilmu yang berfokus pada pengembangan algoritma yang dapat belajar dari dan membuat prediksi atau keputusan berdasarkan data. Proses ini melibatkan penggunaan metode matematika dan statistika yang diterapkan pada data untuk membangun model yang dapat meniru atau memprediksi perilaku yang diinginkan. Dalam beberapa dekade terakhir, Machine Learning telah mengalami perkembangan pesat dan menjadi salah satu bidang yang paling banyak diterapkan dalam kehidupan sehari-hari.

Secara umum, ada tiga kategori utama dalam Machine Learning, yaitu Supervised Learning, Unsupervised Learning, dan Reinforcement Learning. Pada buku ini, kita akan fokus pada Unsupervised Learning. Sebelum membahas lebih lanjut mengenai Unsupervised Learning, ada baiknya kamu memahami dulu konsep Supervised Learning, yang merupakan dasar dari Machine Learning.

Supervised Learning adalah proses melatih model dengan menggunakan data berlabel, di mana setiap data memiliki label atau target yang diinginkan. Model akan belajar dari data tersebut untuk memprediksi label yang belum diketahui dari data baru. Contoh metode Supervised Learning adalah regresi dan klasifikasi.

1.2. Pengertian Unsupervised Learning

Unsupervised Learning adalah metode Machine Learning yang tidak menggunakan data berlabel. Model akan belajar dari data tanpa ada informasi mengenai target yang diinginkan. Dalam Unsupervised Learning, algoritma akan mencoba mengidentifikasi pola atau struktur yang tersembunyi dalam data. Unsupervised Learning memiliki banyak potensi untuk digunakan dalam berbagai aplikasi karena dapat mengatasi masalah di mana data berlabel tidak tersedia atau sulit diperoleh.

Unsupervised Learning umumnya mencakup dua jenis metode, yaitu Clustering dan Dimensionality Reduction. Clustering adalah proses mengelompokkan data menjadi beberapa kelompok atau cluster berdasarkan kesamaan atau perbedaan antara data tersebut. Dimensionality Reduction adalah proses mengurangi jumlah fitur atau dimensi dalam data tanpa kehilangan informasi yang penting. Teknik ini sangat berguna ketika kamu memiliki data berdimensi tinggi yang sulit dianalisis atau ditampilkan secara visual.

1.3. Aplikasi Unsupervised Learning dalam Industri

Unsupervised Learning memiliki berbagai aplikasi dalam industri dan penelitian. Beberapa contoh aplikasi yang umum dan bermanfaat dari Unsupervised Learning adalah:

a. Segmentasi Pelanggan: Unsupervised Learning dapat digunakan untuk mengelompokkan pelanggan berdasarkan karakteristik demografis, perilaku belanja, atau preferensi produk. Dengan mengetahui segmentasi ini, perusahaan dapat menargetkan strategi pemasaran yang lebih efektif dan meningkatkan penjualan serta kepuasan pelanggan.

b. Deteksi Anomali: Unsupervised Learning dapat digunakan untuk mengidentifikasi pola atau aktivitas yang tidak biasa dalam data. Contohnya, pada sistem keamanan jaringan, deteksi anomali dapat mengidentifikasi serangan atau kegiatan yang mencurigakan. Pada bidang keuangan, deteksi anomali dapat membantu mengidentifikasi transaksi mencurigakan atau penipuan kartu kredit.

c. Rekomendasi Produk: Sistem rekomendasi menggunakan Unsupervised Learning untuk mengidentifikasi pola dalam preferensi pengguna dan mengelompokkan item serupa. Dengan mengetahui hubungan antara produk, sistem dapat merekomendasikan produk yang paling relevan dengan preferensi pengguna, sehingga meningkatkan penjualan dan kepuasan pelanggan.

d. Analisis Teks: Unsupervised Learning sangat berguna dalam mengelola dan menggali informasi dari teks dalam jumlah besar. Dengan menggunakan metode seperti clustering dan topic modeling, algoritma dapat mengidentifikasi topik utama dalam teks dan mengelompokkan dokumen yang terkait. Hal ini sangat berguna untuk analisis sentimen, pengelompokan berita, atau pengelompokan dokumen.

e. Analisis Citra: Unsupervised Learning dapat digunakan untuk mengelompokkan atau mengurangi dimensi data citra. Hal ini sangat berguna dalam aplikasi seperti pengenalan wajah, kompresi citra, atau pemisahan objek dalam citra.

f. Bioinformatika: Unsupervised Learning memiliki banyak aplikasi dalam bioinformatika, seperti analisis ekspresi gen, pengelompokan protein, atau analisis struktur molekul. Dengan mengidentifikasi pola dalam data biologis, ilmuwan dapat memahami proses yang mendasari kehidupan dan mengembangkan terapi yang lebih efektif untuk berbagai penyakit.

Seiring dengan perkembangan teknologi dan penelitian, Unsupervised Learning akan terus menemukan aplikasi baru dalam berbagai bidang. Salah satu area yang menjanjikan adalah integrasi Unsupervised Learning dengan teknologi Deep Learning dan Artificial Intelligence (AI), yang akan membuka potensi baru untuk analisis data dan pemecahan masalah yang kompleks.

Dalam buku ini, kita akan membahas berbagai algoritma Unsupervised Learning secara detail, termasuk K-Means, Affinity Propagation, Mean Shift, Spectral Clustering, Ward Hierarchical Clustering, Agglomerative Clustering, DBSCAN, OPTICS, Birch, dan Gaussian

Mixture. Kamu akan belajar cara mengimplementasikan algoritma ini dalam Python, mengoptimalkannya, dan menerapkannya pada berbagai aplikasi industri.

Bab 2. Persiapan Data pada Unsupervised Learning

2.1. Data Collection

Data collection merupakan langkah awal dalam proses pembelajaran mesin yang sangat penting. Kualitas data yang kamu kumpulkan akan sangat mempengaruhi kualitas model yang dihasilkan. Dalam Unsupervised Learning, kamu akan bekerja dengan data yang tidak memiliki label target. Oleh karena itu, sangat penting untuk mengumpulkan data yang informatif dan mewakili pola atau struktur yang ingin kamu temukan.

Pada subbab ini, kita akan membahas beberapa langkah penting yang perlu kamu pertimbangkan dalam proses pengumpulan data untuk Unsupervised Learning. Berikut adalah langkah-langkah tersebut:

2.1.1. Identifikasi Tujuan dan Aplikasi Model

Sebelum mulai mengumpulkan data, kamu harus menentukan tujuan dari model Unsupervised Learning yang ingin kamu bangun. Apakah kamu ingin mengelompokkan data berdasarkan kesamaan fitur? Atau mungkin kamu ingin mengurangi dimensi data untuk visualisasi atau analisis lebih lanjut? Dengan mengetahui tujuan dari model, kamu akan lebih mudah dalam menentukan jenis data yang perlu kamu kumpulkan.

Selain itu, kamu harus mempertimbangkan aplikasi dari model. Model yang akan digunakan untuk analisis teks memerlukan jenis data yang berbeda dari model yang digunakan untuk analisis citra. Jadi, pastikan untuk mengumpulkan data yang sesuai dengan aplikasi yang ingin kamu gunakan.

2.1.2. Identifikasi Sumber Data

Setelah mengetahui tujuan dan aplikasi dari model, langkah selanjutnya adalah mencari sumber data yang relevan. Sumber data dapat berasal dari berbagai tempat, seperti database perusahaan, situs web, sensor, atau bahkan data yang dikumpulkan oleh pengguna.

Ada beberapa hal yang perlu kamu pertimbangkan saat memilih sumber data:

Relevansi: Pastikan data yang kamu kumpulkan relevan dengan tujuan dan aplikasi model. Jika data tidak relevan, model yang dihasilkan mungkin tidak akan berguna atau bahkan menyesatkan.

Kualitas: Kualitas data sangat penting dalam proses pembelajaran mesin. Data yang buruk atau tidak akurat dapat menghasilkan model yang buruk. Pastikan sumber data yang kamu pilih memiliki reputasi yang baik dan konsisten dalam menghasilkan data berkualitas tinggi.

Aksesibilitas: Pastikan kamu memiliki akses yang mudah dan legal ke sumber data yang kamu pilih. Beberapa sumber data mungkin memiliki batasan dalam hal penggunaan atau memerlukan izin khusus.

Kuantitas: Dalam pembelajaran mesin, biasanya lebih banyak data akan menghasilkan model yang lebih baik. Pastikan kamu memiliki cukup data untuk melatih model Unsupervised Learning.

2.1.3. Mengumpulkan Data

Setelah mengidentifikasi sumber data yang sesuai, langkah selanjutnya adalah mengumpulkan data itu sendiri. Berikut adalah beberapa metode yang dapat kamu gunakan untuk mengumpulkan data:

Ekstraksi data dari database: Jika data yang kamu butuhkan tersimpan dalam database, kamu dapat menggunakan query SQL atau alat ekstraksi data lainnya untuk mengambil data yang diperlukan. Pastikan untuk memilih fitur yang relevan dengan tujuan dan aplikasi model.

Web Scraping: Jika data yang kamu butuhkan berasal dari situs web, kamu dapat menggunakan teknik web scraping untuk mengambil data tersebut. Web scraping adalah proses ekstraksi data dari situs web dengan menggunakan program atau script. Ada banyak library Python yang bisa kamu gunakan untuk web scraping, seperti BeautifulSoup, Scrapy, dan Selenium.

Pengumpulan data melalui API: Beberapa layanan atau platform menyediakan API (Application Programming Interface) yang memungkinkan kamu untuk mengakses data mereka secara langsung. Contohnya adalah Twitter API, Facebook API, dan Google Analytics API. Kamu bisa menggunakan library Python seperti Requests atau Tweepy untuk mengakses data melalui API.

Pengumpulan data dari sensor atau perangkat IoT: Jika kamu bekerja dengan data yang dikumpulkan dari sensor atau perangkat Internet of Things (IoT), kamu mungkin perlu mengembangkan sistem untuk mengumpulkan dan mengirim data ke server atau platform analisis. Kamu bisa menggunakan protokol komunikasi seperti MQTT atau HTTP untuk mengirim data dari perangkat ke server.

Survei atau pengumpulan data oleh pengguna: Dalam beberapa kasus, data yang kamu butuhkan mungkin tidak tersedia secara online atau melalui sumber eksternal. Dalam

situasi seperti itu, kamu mungkin perlu mengumpulkan data secara manual melalui survei atau melibatkan pengguna untuk mengumpulkan data. Pastikan untuk merancang survei atau alat pengumpulan data dengan baik agar menghasilkan data yang akurat dan relevan.

2.1.4. Penyimpanan Data

Setelah data berhasil dikumpulkan, kamu perlu menyimpannya dalam format yang sesuai untuk analisis lebih lanjut. Kamu mungkin perlu menggabungkan data dari berbagai sumber atau format sebelum menyimpannya. Berikut adalah beberapa format penyimpanan data yang umum digunakan dalam Machine Learning:

CSV (Comma Separated Values): CSV adalah format file teks sederhana yang digunakan untuk menyimpan data dalam bentuk tabel. CSV mudah dibaca oleh manusia dan kompatibel dengan berbagai aplikasi dan library.

Excel: Microsoft Excel adalah aplikasi pengolah spreadsheet yang populer dan sering digunakan untuk menyimpan data dalam bentuk tabel. Kamu bisa menggunakan library Python seperti Pandas atau Openpyxl untuk membaca dan menulis data dalam format Excel.

SQL: SQL (Structured Query Language) adalah bahasa pemrograman yang digunakan untuk mengelola data dalam sistem manajemen basis data relasional (RDBMS). Kamu bisa menggunakan library Python seperti SQLite, MySQL, atau PostgreSQL untuk menyimpan dan mengakses data dalam format SQL.

JSON (JavaScript Object Notation): JSON adalah format data ringan yang mudah dibaca oleh manusia dan komputer. JSON cocok untuk menyimpan data dengan struktur yang lebih kompleks daripada tabel, seperti data hierarkis atau data yang memiliki banyak relasi. Kamu bisa menggunakan library Python seperti json atau simplejson untuk membaca dan menulis data dalam format JSON.

NoSQL: NoSQL adalah keluarga sistem manajemen basis data yang tidak menggunakan SQL sebagai bahasa query utama. NoSQL cocok untuk menyimpan data dalam skala besar atau data dengan struktur yang tidak teratur. Beberapa contoh sistem NoSQL yang populer adalah MongoDB, Cassandra, dan Redis.

Dalam proses pengumpulan data, sangat penting untuk menjaga keamanan dan privasi data. Pastikan kamu mematuhi peraturan dan regulasi yang berlaku, seperti GDPR (General Data Protection Regulation) atau undang-undang perlindungan data setempat. Selalu lindungi data pengguna dan jangan mengumpulkan informasi yang tidak perlu atau sensitif.

Sekarang setelah kamu telah mengumpulkan data yang diperlukan, langkah selanjutnya adalah membersihkan, mengubah, dan menormalisasi data tersebut. Proses ini sangat penting untuk memastikan bahwa model Unsupervised Learning yang dihasilkan akurat

dan efektif. Pada subbab berikutnya, kita akan membahas langkah-langkah ini secara lebih detail.

2.2. Data Cleaning

Data cleaning adalah proses mengidentifikasi dan mengoreksi kesalahan, inkonsistensi, atau ketidaklengkapan dalam data yang dikumpulkan. Tujuan dari data cleaning adalah untuk meningkatkan kualitas data dan memastikan bahwa model Unsupervised Learning yang dihasilkan memiliki kinerja yang optimal. Berikut adalah beberapa langkah yang perlu kamu pertimbangkan dalam proses data cleaning:

2.2.1. Deteksi dan penghapusan data duplikat

Data duplikat adalah entri data yang identik atau sangat mirip satu sama lain. Data duplikat bisa menyebabkan bias dalam model Unsupervised Learning dan mengurangi kinerjanya. Kamu bisa menggunakan library Python seperti Pandas untuk mengidentifikasi dan menghapus data duplikat. Pandas menyediakan fungsi drop_duplicates() yang bisa kamu gunakan untuk menghapus baris duplikat dalam dataset. Sebagai contoh:

```
import pandas as pd

# Data sintetis
data = pd.DataFrame({'A': [1, 2, 2, 3, 4, 4],
                     'B': [5, 6, 6, 7, 8, 8]})

# Menghapus baris duplikat
data = data.drop_duplicates()

data
```

Output:

	A	B
0	1	5
1	2	6
3	3	7
4	4	8

Dalam contoh ini, kita membuat DataFrame sintetis menggunakan Pandas dengan kolom 'A' dan 'B'. Kemudian, kita menggunakan fungsi drop_duplicates() untuk menghapus baris duplikat dan mencetak DataFrame yang sudah dibersihkan.

2.2.2. Mengatasi data yang hilang

Data yang hilang atau tidak lengkap bisa menyebabkan masalah dalam proses pembelajaran mesin. Ada beberapa metode yang bisa kamu gunakan untuk mengatasi data yang hilang, seperti:

a. Penghapusan baris atau kolom yang memiliki data yang hilang: Metode ini efektif jika jumlah data yang hilang relatif kecil dan penghapusan data tidak akan menyebabkan kehilangan informasi yang penting. Kamu bisa menggunakan fungsi dropna() dari Pandas untuk menghapus baris atau kolom yang memiliki data yang hilang. Sebagai contoh:

```
import pandas as pd

# Data sintetis dengan nilai yang hilang (NaN)
data = pd.DataFrame({'A': [1, 2, None, 3, 4, None],
                     'B': [5, None, 6, 7, 8, None]})

# Menghapus baris yang memiliki data yang hilang
data = data.dropna()

data
```

Output:

	A	B
0	1.0	5.0
3	3.0	7.0
4	4.0	8.0

Dalam contoh ini, kita membuat DataFrame sintetis dengan beberapa nilai yang hilang (None). Kemudian, kita menggunakan fungsi dropna() untuk menghapus baris yang memiliki data yang hilang dan mencetak DataFrame yang sudah dibersihkan.

b. Imputasi data dengan menggunakan rata-rata atau median: Imputasi adalah proses penggantian data yang hilang dengan nilai yang diestimasi. Kamu bisa menggunakan rata-rata atau median dari kolom yang relevan untuk mengisi data yang hilang. Fungsi fillna() dari Pandas dapat digunakan untuk mengisi data yang hilang dengan nilai yang diinginkan. Sebagai contoh:

```
import pandas as pd

# Data sintetis dengan nilai yang hilang (NaN)
data = pd.DataFrame({'A': [1, 2, None, 3, 4, None],
'B': [5, None, 6, 7, 8, None]})
```

```
# Mengisi data yang hilang dengan rata-rata kolom
data = data.fillna(data.mean())

data
```

Output:

	A	B
0	1.0	5.0
1	2.0	6.5
2	2.5	6.0
3	3.0	7.0
4	4.0	8.0
5	2.5	6.5

Dalam contoh ini, kita membuat DataFrame sintetis dengan beberapa nilai yang hilang (None). Kemudian, kita menggunakan fungsi `fillna()` untuk mengisi data yang hilang dengan rata-rata kolom dan mencetak DataFrame yang sudah dibersihkan.

c. Penggunaan metode imputasi yang lebih canggih seperti K-Nearest Neighbors atau Multiple Imputation: Metode imputasi yang lebih canggih bisa memberikan hasil yang lebih akurat dalam mengisi data yang hilang. Kamu bisa menggunakan library seperti `scikit-learn` atau `fancyimpute` untuk mengimplementasikan metode imputasi ini. Sebagai contoh, kamu bisa menggunakan KNNImputer dari scikit-learn untuk mengisi data yang hilang dengan menggunakan metode K-Nearest Neighbors:

```
import pandas as pd
from sklearn.impute import KNNImputer

# Data sintetis dengan nilai yang hilang (NaN)
data = pd.DataFrame({'A': [1, 2, None, 3, 4, None],
                     'B': [5, None, 6, 7, 8, None]})

# Menggunakan KNNImputer untuk mengisi data yang hilang
imputer = KNNImputer(n_neighbors=2)
data_filled = imputer.fit_transform(data)

# Menyimpan dataset yang telah dibersihkan
clean_data = pd.DataFrame(data_filled, columns=data.columns)
clean_data
```

Output:

	A	B
0	1.0	5.0
1	2.0	6.0
2	2.0	6.0
3	3.0	7.0
4	4.0	8.0
5	2.5	6.5

Dalam contoh ini, kita membuat DataFrame sintetis dengan beberapa nilai yang hilang (None). Kemudian, kita menggunakan KNNImputer dari scikit-learn untuk mengisi data yang hilang berdasarkan 2 tetangga terdekat (n_neighbors=2). Terakhir, kita mencetak DataFrame yang sudah dibersihkan.

2.2.3. Koreksi kesalahan entri

Kesalahan entri adalah nilai data yang salah atau tidak akurat karena kesalahan manusia atau sistem. Kamu bisa menggunakan teknik seperti validasi data, aturan bisnis, atau algoritma deteksi anomali untuk mengidentifikasi dan mengoreksi kesalahan entri. Contoh berikut menunjukkan bagaimana kamu bisa memperbaiki kesalahan entri dengan menggantikan nilai yang tidak valid dengan rata-rata kolom:

```
import pandas as pd

# Data sintetis dengan kesalahan entri
data = pd.DataFrame({'A': [1, 2, -999, 3, 4, -999],
                     'B': [5, -999, 6, 7, 8, -999]})

# Menggantikan kesalahan entri (-999) dengan rata-rata kolom
data = data.replace(-999, data[data != -999].mean())

data
```

Output:

	A	B
0	1.0	5.0
1	2.0	6.5
2	2.5	6.0
3	3.0	7.0
4	4.0	8.0
5	2.5	6.5

Dalam contoh ini, kita membuat DataFrame sintetis dengan beberapa nilai yang salah (-999). Kemudian, kita menggunakan fungsi replace() untuk menggantikan nilai yang salah dengan rata-rata kolom (mengabaikan nilai yang salah saat menghitung rata-rata) dan mencetak DataFrame yang sudah dibersihkan.

2.2.4. Konversi tipe data

Dalam beberapa kasus, kamu mungkin perlu mengkonversi tipe data kolom untuk memastikan bahwa model Unsupervised Learning dapat memproses data dengan benar. Misalnya, kamu mungkin perlu mengkonversi kolom teks menjadi kolom numerik atau sebaliknya. Kamu bisa menggunakan fungsi astype() dari Pandas untuk mengkonversi tipe data kolom. Berikut adalah contoh konversi tipe data kolom dari float menjadi integer:

```
import pandas as pd

# Data sintetis dengan tipe data float
data = pd.DataFrame({'A': [1.0, 2.0, 3.0, 4.0, 5.0],
                     'B': [6.0, 7.0, 8.0, 9.0, 10.0]})

# Mengkonversi tipe data kolom menjadi integer
data = data.astype(int)

data
```

Output:

	A	B
0	1	6
1	2	7
2	3	8
3	4	9
4	5	10

Dalam contoh ini, kita membuat DataFrame sintetis dengan tipe data float. Kemudian, kita menggunakan fungsi astype() untuk mengkonversi tipe data kolom menjadi integer dan mencetak DataFrame yang sudah diubah.

Penanganan data kategorikal: Data kategorikal adalah data yang memiliki nilai terbatas yang mewakili kategori atau label. Untuk memproses data kategorikal dalam model Unsupervised Learning, kamu perlu mengubah data kategorikal menjadi bentuk numerik. Ada beberapa teknik yang bisa kamu gunakan, seperti:

a. One-Hot Encoding: One-hot encoding adalah teknik yang digunakan untuk mengkonversi data kategorikal menjadi vektor biner dengan panjang yang sama dengan jumlah kategori yang unik. Kamu bisa menggunakan pd.get_dummies() dari Pandas untuk mengimplementasikan one-hot encoding. Berikut adalah contoh penggunaan one-hot encoding untuk mengkonversi data kategorikal:

```
import pandas as pd

# Data sintetis dengan kolom kategorikal
data = pd.DataFrame({'A': ['red', 'green', 'blue', 'red', 'green'],
                     'B': [1, 2, 3, 4, 5]})

# Menggunakan one-hot encoding untuk mengkonversi kolom kategorikal 'A'
encoded_data = pd.get_dummies(data, columns=['A'])

encoded_data
```

Output:

	B	A_blue	A_green	A_red
0	1	0	0	1
1	2	0	1	0
2	3	1	0	0
3	4	0	0	1
4	5	0	1	0

Dalam contoh ini, kita membuat DataFrame sintetis dengan kolom kategorikal 'A' yang berisi warna. Kemudian, kita menggunakan pd.get_dummies() untuk mengkonversi kolom kategorikal menjadi kolom biner yang mewakili setiap kategori unik dan mencetak DataFrame yang sudah diubah.

b. Label Encoding: Label encoding adalah teknik yang menggantikan setiap kategori dengan nilai numerik unik. Kamu bisa menggunakan LabelEncoder dari scikit-learn untuk mengimplementasikan label encoding. Berikut adalah contoh penggunaan label encoding untuk mengkonversi data kategorikal:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Data sintetis dengan kolom kategorikal
data = pd.DataFrame({'A': ['red', 'green', 'blue', 'red', 'green'],
                     'B': [1, 2, 3, 4, 5]})

# Menggunakan label encoding untuk mengkonversi kolom kategorikal 'A'
encoder = LabelEncoder()
data['A'] = encoder.fit_transform(data['A'])

data
```

Output:

	A	B
0	2	1
1	1	2
2	0	3
3	2	4
4	1	5

Dalam contoh ini, kita membuat DataFrame sintetis dengan kolom kategorikal 'A' yang berisi warna. Kemudian, kita menggunakan `LabelEncoder` untuk mengkonversi kolom kategorikal menjadi nilai numerik yang mewakili setiap kategori unik dan mencetak DataFrame yang sudah diubah.

Dalam rangkuman, data cleaning adalah proses penting dalam persiapan data untuk Unsupervised Learning. Langkah-langkah yang dijelaskan di atas, seperti penghapusan data duplikat, penanganan data yang hilang, koreksi kesalahan entri, konversi tipe data, dan penanganan data kategorikal, membantu menghasilkan dataset yang berkualitas tinggi yang akan meningkatkan kinerja model Unsupervised Learning. Dengan menggunakan contoh kode Python yang diberikan, kamu dapat menerapkan teknik-teknik ini pada dataset kamu sendiri untuk memastikan bahwa data yang digunakan dalam proses pembelajaran mesin telah dibersihkan dan siap untuk digunakan.

2.3. Data Transformation

Data transformation adalah proses mengubah bentuk atau struktur data agar sesuai dengan kebutuhan model Unsupervised Learning. Transformasi data sering diperlukan karena model mungkin bekerja lebih baik dengan beberapa fitur yang berbeda atau dengan representasi yang lebih sederhana dari data asli. Dalam subbab ini, kita akan membahas beberapa teknik transformasi data yang umum digunakan dalam Unsupervised Learning, seperti agregasi fitur, reduksi dimensi, dan encoding fitur.

2.3.1. Agregasi Fitur

Agregasi fitur adalah proses menggabungkan beberapa fitur menjadi satu fitur baru yang mewakili informasi yang ada dalam fitur-fitur tersebut. Agregasi fitur dapat membantu mengurangi dimensi dataset dan mengurangi kompleksitas model. Selain itu, agregasi fitur dapat meningkatkan interpretasi model dan mengurangi risiko overfitting. Berikut adalah contoh mengagregasi fitur dengan menggunakan Python dan Pandas:

```
import pandas as pd

# Data sintetis
data = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                     'B': [6, 7, 8, 9, 10],
                     'C': [11, 12, 13, 14, 15]})

# Mengagregasi fitur 'A' dan 'B' dengan mengambil rata-rata
data['D'] = (data['A'] + data['B']) / 2

# Menghapus fitur 'A' dan 'B' yang lama
data = data.drop(columns=['A', 'B'])
```

```
data
```

Output:

	C	D
0	11	3.5
1	12	4.5
2	13	5.5
3	14	6.5
4	15	7.5

Dalam contoh ini, kita membuat DataFrame sintetis dengan tiga fitur ('A', 'B', dan 'C'). Kemudian, kita mengagregasi fitur 'A' dan 'B' dengan mengambil rata-rata dari kedua fitur tersebut dan menyimpan hasilnya dalam fitur baru 'D'. Terakhir, kita menghapus fitur 'A' dan 'B' yang lama dan mencetak DataFrame yang sudah diubah.

2.3.2. Reduksi Dimensi

Reduksi dimensi adalah proses mengurangi jumlah fitur dalam dataset dengan mempertahankan informasi penting. Reduksi dimensi dapat membantu mengurangi kompleksitas model, meningkatkan interpretasi, dan mengurangi risiko overfitting. Ada beberapa teknik reduksi dimensi yang umum digunakan dalam Unsupervised Learning, seperti Principal Component Analysis (PCA), t-SNE, dan UMAP. Berikut adalah contoh mengurangi dimensi dataset dengan menggunakan PCA dari scikit-learn:

```
import pandas as pd
from sklearn.decomposition import PCA

# Data sintetis
data = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                     'B': [6, 7, 8, 9, 10],
                     'C': [11, 12, 13, 14, 15]})

# Menggunakan PCA untuk mengurangi dimensi dataset menjadi 2 fitur
pca = PCA(n_components=2)
transformed_data = pca.fit_transform(data)

# Menyimpan dataset yang telah direduksi
reduced_data = pd.DataFrame(transformed_data, columns=['PC1', 'PC2'])

reduced_data
```

Output:

	PC1	PC2
0	3.464102	3.439900e-16
1	1.732051	-1.146633e-16
2	-0.000000	-0.000000e+00
3	-1.732051	1.146633e-16
4	-3.464102	2.293267e-16

Dalam contoh ini, kita membuat DataFrame sintetis dengan tiga fitur ('A', 'B', dan 'C'). Kemudian, kita menggunakan PCA untuk mengurangi dimensi dataset menjadi dua fitur, yang disimpan dalam kolom 'PC1' dan 'PC2'. Terakhir, kita mencetak DataFrame yang sudah direduksi.

2.3.3. Encoding Fitur

Encoding fitur adalah proses mengubah representasi fitur sehingga lebih sesuai untuk model Unsupervised Learning. Salah satu contoh encoding fitur adalah konversi data kategorikal menjadi data numerik, seperti yang telah dijelaskan di subbab 2.2. Encoding fitur lain yang umum digunakan dalam Unsupervised Learning adalah konversi teks menjadi vektor numerik dengan menggunakan teknik seperti Bag of Words, TF-IDF, atau Word2Vec. Berikut adalah contoh mengkonversi teks menjadi vektor numerik dengan menggunakan TF-IDF dari scikit-learn:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Synthetic data as text
data = pd.DataFrame({'A': ['this is a sample sentence',
                           'another example sentence',
                           'a third text for illustration']})

# Using TF-IDF to convert text into numerical vectors
vectorizer = TfidfVectorizer()
encoded_data = vectorizer.fit_transform(data['A'])

# Storing the encoded dataset
encoded_df = pd.DataFrame(encoded_data.toarray(),
columns=vectorizer.get_feature_names_out())

encoded_df
```

Output:

	another	example	for	illustration	is	sample	sentence	text	third	this
0	0.000000	0.000000	0.0	0.0	0.528635	0.528635	0.40204	0.0	0.0	0.528635
1	0.622766	0.622766	0.0	0.0	0.000000	0.000000	0.47363	0.0	0.0	0.000000
2	0.000000	0.000000	0.5	0.5	0.000000	0.000000	0.00000	0.5	0.5	0.000000

Dalam contoh ini, kita membuat DataFrame sintetis dengan satu kolom teks 'A'. Kemudian, kita menggunakan TfidfVectorizer untuk mengkonversi teks dalam kolom 'A' menjadi vektor numerik. Hasilnya disimpan dalam DataFrame baru, yang mencakup kolom untuk setiap kata yang unik dalam dataset dan nilai TF-IDF yang sesuai. Terakhir, kita mencetak DataFrame yang sudah di-encode.

Dalam rangkuman, transformasi data adalah langkah penting dalam persiapan data untuk Unsupervised Learning. Teknik transformasi data yang dijelaskan di atas, seperti agregasi fitur, reduksi dimensi, dan encoding fitur, membantu menghasilkan dataset yang lebih sesuai dengan kebutuhan model Unsupervised Learning. Dengan menggunakan contoh kode Python yang diberikan, kamu dapat menerapkan teknik-teknik ini pada dataset kamu sendiri untuk memastikan bahwa data yang digunakan dalam proses pembelajaran mesin telah ditransformasi sesuai kebutuhan.

2.4. Data Scaling and Normalization

Data scaling dan normalization adalah proses penyesuaian rentang nilai fitur data agar sesuai dengan skala yang ditentukan. Dalam banyak algoritma pembelajaran mesin, terutama yang berbasis jarak seperti K-Means, Affinity Propagation, MeanShift, dan Spectral Clustering, penting untuk memiliki fitur pada skala yang sama sehingga satu fitur tidak mendominasi yang lain.

2.4.1. Data Scaling

Data scaling adalah teknik yang digunakan untuk mengubah rentang nilai fitur sehingga semua fitur memiliki skala yang sama. Salah satu metode scaling yang populer adalah StandardScaler dari library scikit-learn, yang mengubah data sedemikian rupa sehingga memiliki rata-rata nol dan standar deviasi satu. Berikut adalah contoh penggunaan StandardScaler pada data sintetis:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Data sintetis
data = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                     'B': [100, 200, 300, 400, 500]})

# Menggunakan StandardScaler untuk menskalakan data
scaler = StandardScaler()
```

```
scaled_data = scaler.fit_transform(data)

# Menyimpan data yang telah diskalakan
scaled_df = pd.DataFrame(scaled_data, columns=['A_scaled', 'B_scaled'])

scaled_df
```

Output:

	A_scaled	B_scaled
0	-1.414214	-1.414214
1	-0.707107	-0.707107
2	0.000000	0.000000
3	0.707107	0.707107
4	1.414214	1.414214

Dalam contoh ini, kita membuat DataFrame sintetis dengan dua fitur ('A' dan 'B') yang memiliki skala yang sangat berbeda. Kemudian, kita menggunakan StandardScaler untuk menskalakan data sehingga semua fitur memiliki rata-rata nol dan standar deviasi satu.

2.4.2. Data Normalization

Data normalization adalah teknik yang digunakan untuk mengubah rentang nilai fitur sehingga semua fitur memiliki rentang antara 0 dan 1. Metode normalization yang umum digunakan adalah MinMaxScaler dari library scikit-learn. Berikut adalah contoh penggunaan MinMaxScaler pada data sintetis:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Data sintetis
data = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                     'B': [100, 200, 300, 400, 500]})

# Menggunakan MinMaxScaler untuk menormalisasi data
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)

# Menyimpan data yang telah dinormalisasi
normalized_df = pd.DataFrame(normalized_data, columns=['A_normalized',
'B_normalized'])

normalized_df
```

Output:

	A_normalized	B_normalized
0	0.00	0.00
1	0.25	0.25
2	0.50	0.50
3	0.75	0.75
4	1.00	1.00

Dalam contoh ini, kita membuat DataFrame sintetis dengan dua fitur ('A' dan 'B') yang memiliki skala yang sangat berbeda. Kemudian, kita menggunakan MinMaxScaler untuk menormalisasi data sehingga semua fitur memiliki rentang antara 0 dan 1.

Selain StandardScaler dan MinMaxScaler, ada metode scaling dan normalization lainnya seperti RobustScaler (menggunakan median dan kuartil, kurang sensitif terhadap pencilan) dan QuantileTransformer (mengubah distribusi data ke distribusi normal atau uniform).

Secara keseluruhan, data scaling dan normalization merupakan langkah penting dalam persiapan data untuk Unsupervised Learning. Penting untuk mencatat bahwa scaler atau normalizer yang sama yang digunakan pada data pelatihan harus digunakan pada data tes atau data baru nantinya untuk memastikan konsistensi dalam skala data. Metode scaling atau normalization yang digunakan dapat bervariasi tergantung pada karakteristik data dan algoritma pembelajaran mesin yang digunakan.

Untuk contoh, jika data memiliki banyak pencilan dan kita menggunakan algoritma berbasis jarak, mungkin lebih baik menggunakan RobustScaler. Jika data tidak mengikuti distribusi normal dan kita ingin mengubah distribusinya, QuantileTransformer bisa menjadi pilihan yang baik.

```
import pandas as pd
from sklearn.preprocessing import RobustScaler, QuantileTransformer

# Data sintetis
data = pd.DataFrame({'A': [1, 2, 3, 4, 5, 100],
                     'B': [100, 200, 300, 400, 500, 1000]})

# Menggunakan RobustScaler
scaler = RobustScaler()
robust_scaled_data = scaler.fit_transform(data)
robust_scaled_df = pd.DataFrame(robust_scaled_data,
columns=['A_robust_scaled', 'B_robust_scaled'])

display(robust_scaled_df)
```

```
# Menggunakan QuantileTransformer
transformer = QuantileTransformer(output_distribution='normal')
quantile_transformed_data = transformer.fit_transform(data)
quantile_transformed_df = pd.DataFrame(quantile_transformed_data,
columns=['A_quantile_transformed', 'B_quantile_transformed'])

display(quantile_transformed_df)
```

Output:

	A_robust_scaled	B_robust_scaled
0	-1.0	-1.0
1	-0.6	-0.6
2	-0.2	-0.2
3	0.2	0.2
4	0.6	0.6
5	38.6	2.6

	A_quantile_transformed	B_quantile_transformed
0	-5.199338	-5.199338
1	-0.841621	-0.841621
2	-0.253347	-0.253347
3	0.253347	0.253347
4	0.841621	0.841621
5	5.199338	5.199338

Dalam contoh pertama, kita menggunakan RobustScaler untuk menskalakan data, yang kurang sensitif terhadap pencilan. Dalam contoh kedua, kita menggunakan QuantileTransformer untuk mengubah distribusi data menjadi distribusi normal.

Memahami dan menerapkan data scaling dan normalization dengan tepat dapat memainkan peran penting dalam meningkatkan kinerja model Unsupervised Learning. Menggunakan contoh kode Python di atas, kamu dapat menerapkan teknik-teknik ini pada dataset kamu sendiri untuk memastikan bahwa data yang digunakan dalam proses pembelajaran mesin telah diskalakan dan dinormalisasi dengan benar.

Bab 3. K-Means Clustering

3.1. Pengertian K-Means Clustering

Pada dunia machine learning, terdapat banyak metode yang digunakan untuk mengungkap pola atau struktur tersembunyi dari data, salah satunya adalah K-Means Clustering. Sebagai algoritma unsupervised learning, K-Means memiliki peran penting dalam mengklasifikasikan objek-objek dalam data yang tidak memiliki label, dengan mencoba memahami struktur data tersebut dan mengelompokkannya berdasarkan kemiripan fitur.

Sebagai salah satu algoritma clustering yang paling populer, K-Means memberikan pendekatan yang cukup intuitif dan mudah dimengerti dalam memecahkan permasalahan tersebut. Dalam K-Means, K adalah parameter yang mewakili jumlah cluster atau grup yang ingin kita buat dari data. Misalnya, jika K=3, maka K-Means akan mencoba untuk mengelompokkan data menjadi 3 cluster.

Algoritma K-Means berjalan dengan cara mengelompokkan data ke dalam K cluster sedemikian rupa sehingga setiap data paling dekat dengan pusat cluster (centroid) tempat ia dikelompokkan dibandingkan dengan centroid cluster lainnya. Dalam konteks ini, jarak antara data dan centroid menjadi kunci dalam menentukan pengelompokan data.

Secara lebih jelas, bayangkan kamu memiliki sekumpulan titik data pada sebuah bidang dua dimensi. Saat algoritma K-Means dijalankan, pertama-tama ia akan secara acak menetapkan K titik sebagai pusat cluster (centroid). Kemudian, algoritma ini akan menghitung jarak antara setiap titik data dengan semua centroid dan mengelompokkan setiap titik data ke cluster yang memiliki centroid terdekat. Setelah semua titik data dikelompokkan, centroid akan diupdate menjadi titik rata-rata dari semua titik data dalam cluster tersebut. Proses ini akan diulangi hingga posisi centroid tidak berubah lagi atau hingga mencapai jumlah iterasi maksimum yang ditentukan.

Keunggulan dari K-Means adalah algoritma ini cukup efisien dalam hal komputasi, sehingga bisa digunakan untuk data dengan skala besar. Selain itu, karena prinsip kerjanya yang cukup sederhana, K-Means juga mudah dimengerti dan diimplementasikan.

Namun, ada beberapa hal yang perlu kamu perhatikan saat menggunakan K-Means. Pertama, K-Means sensitif terhadap penentuan centroid awal dan bisa jadi memberikan hasil yang berbeda-beda setiap kali dijalankan. Kedua, K-Means beroperasi dengan asumsi bahwa semua cluster memiliki bentuk bulat dan variasi yang sama, yang bisa menjadi masalah jika data kamu memiliki cluster dengan bentuk atau variasi yang berbeda. Ketiga, kamu perlu menentukan jumlah cluster K sejak awal, yang bisa menjadi tantangan jika kamu tidak memiliki pengetahuan awal tentang data kamu.

Meski demikian, K-Means tetap menjadi algoritma clustering yang sangat populer dan banyak digunakan dalam berbagai bidang, seperti segmentasi pasar, pengelompokan dokumen, analisis citra, dan lainnya. Dengan memahami konsep dasar K-Means Clustering dan bagaimana algoritma ini bekerja, kamu sudah selangkah lebih dekat dalam menguasai dunia unsupervised learning.

Ada satu hal lagi yang perlu dipahami sebelum kita lanjut ke subbab selanjutnya: K-Means bukanlah solusi untuk semua permasalahan clustering. Setiap metode memiliki kelebihan dan kekurangannya masing-masing, dan pemilihan metode yang tepat sangat bergantung pada sifat dan karakteristik data kamu, serta apa yang ingin kamu capai dari proses clustering. Sebagai contoh, jika kamu memiliki data yang mengandung noise atau outlier, metode lain seperti DBSCAN atau OPTICS mungkin lebih cocok. Jika kamu tidak yakin berapa banyak cluster yang ada dalam data, kamu bisa mencoba metode seperti Hierarchical Clustering atau Gaussian Mixture Models yang tidak memerlukan penentuan jumlah cluster sejak awal.

Dengan demikian, penting untuk selalu mengeksplorasi berbagai metode dan pendekatan, dan tidak takut untuk mencoba sesuatu yang baru. Dalam dunia machine learning, tidak ada satu metode yang paling baik untuk semua situasi, dan seringkali keberhasilan suatu proyek sangat bergantung pada kreativitas dan kemampuan kita dalam memahami dan mengadaptasi berbagai metode dan teknik.

Dalam subbab berikutnya, kita akan membahas lebih detail tentang bagaimana algoritma K-Means bekerja, bagaimana menentukan jumlah cluster yang tepat, dan bagaimana melakukan peningkatan pada algoritma K-Means. Setelah itu, kita akan membahas bagaimana mengimplementasikan K-Means dalam Python, dengan contoh kode dan penjelasan yang lengkap. Jadi, tetaplah bersama kami dan mari kita lanjutkan perjalanan kita dalam memahami dunia unsupervised learning dan K-Means Clustering.

3.2. Algoritma K-Means

Setelah memahami apa itu K-Means Clustering, sekarang saatnya kita mengeksplorasi lebih dalam tentang bagaimana algoritma ini bekerja. Algoritma K-Means adalah sebuah proses iteratif yang berusaha mengelompokkan data ke dalam K cluster berdasarkan jarak antara data dan centroid.

Langkah-langkah umum dari algoritma K-Means adalah sebagai berikut:

1. Inisialisasi Centroid: Pertama-tama, kamu harus menentukan jumlah cluster K yang diinginkan. Setelah itu, K-Means akan secara acak menetapkan K titik sebagai pusat cluster atau centroid. Biasanya, titik-titik ini dipilih secara acak dari data yang ada, meskipun ada juga metode lain seperti K-Means++ yang kita akan bahas di subbab selanjutnya.
2. Pengelompokan Data: Setelah centroid diinisialisasi, K-Means akan menghitung jarak antara setiap titik data dengan semua centroid. Jarak ini biasanya dihitung

menggunakan jarak Euclidean, meskipun ada juga metrik jarak lain yang bisa digunakan tergantung pada kasus yang dihadapi. Setelah itu, setiap titik data akan dikelompokkan ke cluster yang memiliki centroid terdekat.

3. Update Centroid: Setelah semua titik data dikelompokkan, posisi centroid akan diupdate menjadi titik rata-rata dari semua titik data dalam cluster tersebut. Dengan kata lain, koordinat centroid baru adalah rata-rata koordinat semua titik data dalam cluster tersebut.
4. Iterasi: Langkah-langkah di atas akan diulangi hingga posisi centroid tidak berubah lagi atau hingga mencapai jumlah iterasi maksimum yang ditentukan. Setiap kali iterasi berlangsung, centroid mungkin berpindah posisi dan beberapa titik data mungkin berpindah ke cluster lain. Namun, jika algoritma sudah konvergen, posisi centroid tidak akan berubah lagi dan setiap titik data sudah berada di cluster yang paling dekat dengan centroidnya.

Sebagai algoritma berbasis jarak, K-Means sangat bergantung pada pengukuran jarak antara titik data dan centroid. Dalam banyak kasus, jarak Euclidean digunakan sebagai metrik jarak karena mudah dihitung dan dipahami. Namun, dalam beberapa kasus, kamu mungkin ingin menggunakan metrik jarak lain seperti jarak Manhattan atau jarak Minkowski, tergantung pada sifat data dan permasalahan yang dihadapi.

Salah satu kelemahan dari algoritma K-Means adalah sensitif terhadap inisialisasi centroid awal. Jika centroid diinisialisasi secara acak, ada kemungkinan algoritma akan terjebak di local optimum dan tidak bisa mencapai solusi yang optimal. Untuk mengatasi masalah ini, kamu bisa menjalankan algoritma K-Means beberapa kali dengan inisialisasi centroid yang berbeda-beda dan memilih hasil yang terbaik. Alternatif lain adalah menggunakan metode seperti K-Means++ yang mencoba memperbaiki proses inisialisasi centroid.

Selain itu, K-Means juga mengasumsikan bahwa setiap cluster memiliki bentuk spherical dan varians yang sama di semua dimensi. Hal ini mungkin bukan asumsi yang tepat untuk semua jenis data. Misalnya, jika cluster dalam data kamu memiliki bentuk yang lebih panjang atau tidak simetris, atau jika varian data berbeda di setiap dimensi, K-Means mungkin tidak akan bekerja dengan baik. Dalam kasus-kasus seperti itu, kamu mungkin perlu mencoba algoritma clustering lain seperti DBSCAN atau Gaussian Mixture Models.

Satu lagi hal yang perlu dipertimbangkan adalah cara kita menentukan jumlah cluster K. Pada dasarnya, tidak ada cara yang tepat untuk menentukan jumlah cluster. Beberapa teknik yang bisa digunakan adalah metode elbow, silhoutte method, atau gap statistic, tetapi semuanya memiliki kelebihan dan kekurangan masing-masing. Pada akhirnya, pemilihan jumlah cluster seringkali bergantung pada pengetahuan domain dan tujuan analisis kamu. Kamu mungkin juga perlu mencoba beberapa nilai K yang berbeda dan melihat hasilnya.

Meskipun memiliki beberapa kelemahan, K-Means tetap menjadi salah satu algoritma clustering yang paling populer dan banyak digunakan. Alasan utamanya adalah karena K-Means mudah dipahami dan diimplementasikan, serta cukup efisien secara komputasi

dibandingkan dengan algoritma clustering lainnya. K-Means juga bisa diterapkan pada berbagai jenis data dan permasalahan, asalkan kamu memahami asumsi dan batasan yang ada.

Itulah penjelasan tentang algoritma K-Means. Semoga penjelasan ini bisa membantu kamu memahami bagaimana K-Means bekerja dan bagaimana cara menggunakannya. Dalam subbab berikutnya, kita akan membahas tentang bagaimana menentukan jumlah cluster yang tepat dan bagaimana melakukan peningkatan pada algoritma K-Means. Jadi, tetaplah bersama kami dan mari kita lanjutkan perjalanan kita dalam memahami dunia unsupervised learning dan K-Means Clustering.

3.3. Pemilihan Jumlah Cluster

Ketika kamu menggunakan K-Means Clustering, salah satu keputusan penting yang harus kamu buat adalah menentukan jumlah cluster, K. Bagaimana cara menentukan jumlah cluster yang optimal? Sayangnya, tidak ada jawaban yang pasti untuk pertanyaan ini. Pemilihan jumlah cluster seringkali bergantung pada pengetahuan domain dan tujuan analisis. Namun, ada beberapa metode heuristik yang bisa kamu gunakan untuk membantu menentukan jumlah cluster yang paling sesuai. Dalam subbab ini, kita akan membahas beberapa metode tersebut, yaitu metode Elbow, Silhouette, dan Gap Statistic.

3.3.1. Metode Elbow

Metode Elbow adalah salah satu metode yang paling sering digunakan untuk menentukan jumlah cluster. Ide dasarnya adalah menjalankan K-Means dengan berbagai jumlah cluster dan menghitung Sum of Squared Errors (SSE) untuk setiap jumlah cluster. SSE adalah jumlah kuadrat jarak antara setiap titik data dengan centroid cluster-nya, dan bisa dianggap sebagai ukuran seberapa baik model K-Means kita.

Setelah kamu menghitung SSE untuk berbagai jumlah cluster, kamu bisa membuat plot antara jumlah cluster dan SSE. Plot ini biasanya akan memiliki bentuk seperti siku atau 'elbow', dengan SSE yang cepat menurun ketika jumlah cluster ditingkatkan hingga suatu titik, dan kemudian mulai menurun dengan lebih lambat setelah titik tersebut. Titik 'elbow' ini bisa dianggap sebagai jumlah cluster yang optimal, karena menambahkan lebih banyak cluster setelah titik ini tidak akan memberikan peningkatan signifikan dalam penyesuaian model.

3.3.2. Metode Silhouette

Metode Silhouette adalah metode lain yang bisa digunakan untuk menentukan jumlah cluster. Metode ini menghitung skor Silhouette untuk setiap titik data, yang merupakan ukuran seberapa dekat titik data dengan titik-titik lain dalam cluster yang sama dibandingkan dengan titik-titik di cluster lain.

Skor Silhouette berkisar antara -1 dan 1. Skor yang mendekati 1 menunjukkan bahwa titik

data jauh lebih dekat dengan titik-titik dalam cluster yang sama dibandingkan dengan titik-titik di cluster lain, yang berarti titik data tersebut sudah dikelompokkan dengan baik. Sebaliknya, skor yang mendekati -1 menunjukkan bahwa titik data lebih dekat dengan titik-titik di cluster lain dibandingkan dengan titik-titik dalam cluster yang sama.

Untuk menentukan jumlah cluster, kamu bisa menjalankan K-Means dengan berbagai jumlah cluster dan menghitung rata-rata skor Silhouette untuk setiap jumlah cluster. Jumlah cluster dengan rata-rata skor Silhouette tertinggi bisa dianggap sebagai jumlah cluster yang optimal.

3.3.3. Metode Gap Statistic

Metode Gap Statistic adalah metode yang cukup baru dan lebih kompleks dibandingkan dengan metode Elbow dan Silhouette. Metode ini mencoba untuk menstandardisasi SSE dengan membandingkannya dengan SSE dari data yang dihasilkan secara acak.

Untuk setiap jumlah cluster, kamu menjalankan K-Means dan menghitung SSE seperti pada metode Elbow. Namun, kamu juga menghasilkan beberapa set data acak dengan distribusi uniform, menjalankan K-Means pada data acak tersebut, dan menghitung SSE untuk setiap set data acak. Gap statistic adalah logaritma dari rasio antara SSE data asli dan rata-rata SSE data acak.

Jumlah cluster dengan Gap statistic tertinggi bisa dianggap sebagai jumlah cluster yang optimal. Keuntungan dari metode ini adalah bahwa ia tidak hanya bergantung pada struktur data asli, tetapi juga mempertimbangkan apa yang diharapkan dari data yang dihasilkan secara acak.

Sebagai catatan, metode-metode yang kita bahas di atas bukanlah satu-satunya cara untuk menentukan jumlah cluster. Beberapa peneliti dan praktisi mungkin menggunakan metode lain, atau bahkan kombinasi beberapa metode. Namun, metode-metode ini memberikan titik awal yang baik dan seringkali cukup efektif.

Ingatlah bahwa K-Means dan algoritma clustering lainnya adalah alat yang bisa membantu kamu memahami struktur data kamu, dan tidak ada algoritma yang bisa menggantikan pengetahuan dan pemahaman kamu tentang data dan domain masalah kamu. Jangan ragu untuk bereksperimen dengan berbagai jumlah cluster dan metode, dan selalu cek hasilnya untuk memastikan bahwa mereka masuk akal dan sesuai dengan tujuan analisis kamu.

Di subbab berikutnya, kita akan membahas tentang K-Means++, sebuah penyempurnaan dari algoritma K-Means yang bisa membantu mengatasi beberapa kelemahan K-Means. Tetaplah bersama kami, dan mari kita lanjutkan belajar tentang dunia yang menarik dari unsupervised learning dan K-Means Clustering.

3.4. K-Means++: Penyempurnaan K-Means

Selama ini, kita telah membahas tentang K-Means dan bagaimana algoritma ini bekerja. Kita juga telah mempelajari bagaimana menentukan jumlah cluster yang optimal. Tetapi, seperti yang kamu mungkin sudah sadari, ada satu aspek penting dalam K-Means yang masih belum kita bahas: pemilihan titik awal atau centroid awal. Dalam subbab ini, kita akan membahas tentang bagaimana pemilihan titik awal dapat mempengaruhi hasil K-Means, dan bagaimana K-Means++ bisa membantu mengatasi masalah ini.

3.4.1. Pengaruh Titik Awal pada K-Means

Dalam K-Means, kita memulai algoritma dengan memilih secara acak K titik data sebagai centroid awal. Kemudian, kita mengulangi langkah-langkah K-Means sampai centroid tidak bergerak lagi atau sampai iterasi mencapai batas maksimal. Namun, apa yang terjadi jika kita memilih titik awal yang berbeda? Apakah kita akan mendapatkan hasil yang sama?

Jawabannya, sebenarnya tidak. Pemilihan titik awal bisa sangat mempengaruhi hasil K-Means. Jika kamu beruntung dan memilih titik awal yang 'baik', kamu bisa mendapatkan hasil yang sangat bagus. Namun, jika kamu tidak beruntung dan memilih titik awal yang 'buruk', kamu bisa mendapatkan hasil yang jauh dari optimal.

3.4.2. K-Means++: Mengatasi Masalah Pemilihan Titik Awal

K-Means++ adalah sebuah metode yang dirancang untuk mengatasi masalah pemilihan titik awal dalam K-Means. Ide dasarnya adalah bukan memilih titik awal secara acak, tetapi memilihnya dengan cara yang lebih cerdas yang berpotensi menghasilkan hasil yang lebih baik.

Berikut adalah langkah-langkah dasar dari K-Means++:

1. Pilih satu titik data secara acak sebagai centroid pertama.
2. Untuk setiap titik data yang belum dipilih sebagai centroid, hitung jarak kuadrat ke centroid terdekat yang sudah dipilih.
3. Pilih titik data berikutnya untuk menjadi centroid dengan probabilitas yang proporsional dengan jarak kuadratnya ke centroid terdekat. Artinya, titik data yang lebih jauh dari centroid yang sudah dipilih memiliki probabilitas lebih tinggi untuk dipilih sebagai centroid berikutnya.
4. Ulangi langkah 2 dan 3 sampai kita memiliki K centroid.
5. Setelah kita memiliki K centroid awal, kita bisa menjalankan K-Means seperti biasa.

3.4.3. Keuntungan K-Means++

K-Means++ memiliki beberapa keuntungan dibandingkan dengan K-Means biasa. Pertama, K-Means++ cenderung menghasilkan hasil yang lebih baik dibandingkan dengan K-Means dengan inisialisasi acak. Karena cara pemilihan centroid awalnya, K-Means++ cenderung menghasilkan cluster yang lebih seimbang dan mengurangi risiko terjebak di solusi lokal yang buruk.

Kedua, meski K-Means++ membutuhkan waktu komputasi yang sedikit lebih lama untuk inisialisasi centroid dibandingkan K-Means biasa, dalam banyak kasus, K-Means++ sebenarnya bisa mengurangi jumlah iterasi yang diperlukan untuk konvergensi. Jadi, secara keseluruhan, K-Means++ bisa lebih efisien dari segi waktu komputasi.

Ketiga, K-Means++ bisa lebih robust terhadap inisialisasi. Dalam K-Means biasa, jika kamu menjalankan algoritma beberapa kali dengan inisialisasi acak yang berbeda, kamu mungkin mendapatkan hasil yang sangat berbeda. Dengan K-Means++, hasil yang kamu dapatkan cenderung lebih konsisten, meski kamu menjalankan algoritma beberapa kali.

Implementasi K-Means++ dalam Python

Mari kita lihat bagaimana cara mengimplementasikan K-Means++ dalam Python. Untungnya, library sklearn sudah menyediakan implementasi K-Means++ yang bisa kita gunakan dengan mudah.

Berikut adalah contoh kode untuk melakukan clustering dengan K-Means++:

```
from sklearn.cluster import KMeans

# Generate synthetic data
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)

# Perform KMeans with KMeans++
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=0)
kmeans.fit(X)

# Print cluster centers and labels
print('Cluster Centers:', kmeans.cluster_centers_)
print('Labels:', kmeans.labels_)
```

Output:

```
Cluster Centers: [[ 1.98258281  0.86771314]
 [ 0.94973532  4.41906906]
 [-1.37324398  7.75368871]
 [-1.58438467  2.83081263]]
Labels: [0 2 1 2 0 0 3 1 2 2 3 2 1 2 0 1 1 0 3 3 0 0 1 3 3 1 0 1 3 1 2 2 1 2 2 2 2
 2 3 0 1 3 1 1 3 3 2 3 2 0 3 0 2 0 0 3 2 3 2 0 2 1 2 3 3 3 2 0 2 3 1 3 2 3
 3 2 3 1 0 2 0 1 0 0 2 1 0 1 2 2 1 0 2 3 3 1 0 0 1 3 2 0 2 0 1 0 0 1 2 1 3
 3 0 2 0 1 2 0 0 1 3 0 3 0 0 0 0 3 0 3 2 3 3 0 2 3 3 2 1 2 2 3 1 3 1 3 2 1
 2 2 2 1 2 1 0 3 2 3 0 1 2 1 1 0 1 3 3 1 0 1 1 2 0 1 3 2 0 0 1 3 0 1 3 3 1
 1 1 1 0 2 1 3 1 1 3 3 3 1 3 2 1 3 0 3 1 2 3 2 1 2 1 3 1 1 2 3 3 0 0 1 2 0
 0 3 0 3 1 2 2 1 1 2 1 0 3 1 0 3 2 3 0 1 0 2 2 2 2 3 3 2 1 3 0 1 3 3 3 0 0
 2 1 1 3 0 2 3 1 2 1 0 0 3 3 1 0 0 0 1 2 2 0 0 1 0 0 0 2 3 2 1 0 0 2 2 2 0
 0 1 2 3]
```

Dalam kode di atas, kita menggunakan parameter init='k-means++' untuk mengaktifkan K-Means++. Selain itu, semua langkahnya sama dengan K-Means biasa.

Kesimpulannya, meski K-Means adalah algoritma yang kuat dan populer, ada beberapa aspek dalam algoritma ini yang bisa ditingkatkan, salah satunya adalah pemilihan titik awal atau centroid awal. K-Means++ menawarkan solusi yang cerdas dan efisien untuk masalah ini, dengan memilih titik awal dengan cara yang berpotensi bisa menghasilkan hasil clustering yang lebih baik dan lebih konsisten.

Penting untuk diingat bahwa, meski K-Means++ bisa memberikan hasil yang lebih baik dibandingkan K-Means biasa, tidak ada jaminan bahwa K-Means++ akan selalu memberikan hasil terbaik untuk setiap dataset. Seperti halnya algoritma machine learning lainnya, performa K-Means++ juga tergantung pada karakteristik data kamu. Oleh karena itu, penting bagi kamu untuk selalu melakukan eksplorasi dan eksperimen dengan data kamu, dan tidak hanya bergantung pada satu algoritma saja.

3.5. Implementasi K-Means dalam Python

Dalam subbab ini, kita akan membahas bagaimana mengimplementasikan algoritma K-Means dalam Python. Kita akan menggunakan library scikit-learn yang populer, yang menyediakan alat yang mudah digunakan untuk melakukan clustering dengan K-Means dan K-Means++.

3.5.1. Membuat Data Sintetis

Langkah pertama dalam melakukan clustering adalah memiliki data yang akan di-cluster. Untuk tujuan pembelajaran ini, kita akan menggunakan fungsi make_blobs dari scikit-learn untuk membuat dataset sintetis.

```
from sklearn.datasets import make_blobs

# Membuat data sintetis dengan 5 cluster
X, y_true = make_blobs(n_samples=5000, centers=5, cluster_std=0.60,
random_state=0)
```

Fungsi make_blobs ini akan menghasilkan data acak dengan jumlah cluster yang telah ditentukan. Dalam hal ini, kita akan membuat 5000 sampel dengan 5 pusat cluster. Jika kamu melihat plot dari data ini, kamu akan melihat 5 cluster data yang jelas.

3.5.2. Melakukan Clustering dengan K-Means

Setelah kita memiliki data, kita dapat melakukan clustering. Di sini kita akan menggunakan algoritma K-Means++ yang telah kita bahas sebelumnya.

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)
kmeans.fit(X)
```

Kode di atas akan melakukan clustering pada data X dengan menggunakan 5 cluster. Parameter init='k-means++' menunjukkan bahwa kita menggunakan K-Means++ untuk inisialisasi centroid.

3.5.3. Menampilkan Hasil Clustering

Setelah melakukan clustering, kita bisa menampilkan hasilnya. Kita bisa mencetak pusat cluster dan label cluster untuk setiap sampel data.

```
print('Cluster Centers:', kmeans.cluster_centers_)
print('Labels:', kmeans.labels_)
```

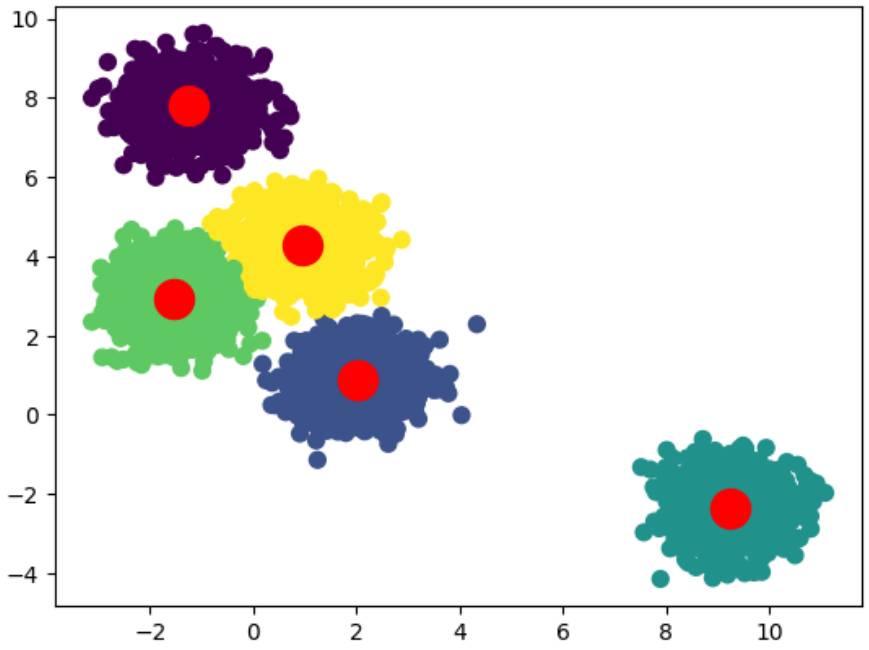
Output:

```
Cluster Centers: [[-1.23873703  7.81463031]
 [ 2.01821378  0.89619452]
 [ 9.24724341 -2.34848471]
 [-1.52429925  2.91978914]
 [ 0.96231466  4.2950961 ]]
Labels: [0 1 4 ... 1 2 2]
```

Selain itu, kita juga bisa membuat plot dari data dan pusat cluster untuk melihat hasil clustering secara visual.

```
import matplotlib.pyplot as plt

# Plot data points
plt.scatter(X[:, 0], X[:, 1], s=50, c=kmeans.labels_, cmap='viridis')

# Plot cluster centers
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], s=300, c='red')
plt.show()
```

Output:



Dalam plot tersebut, titik data ditampilkan dalam warna yang berbeda tergantung pada cluster mereka, dan pusat cluster ditampilkan sebagai titik merah.

3.5.4. Evaluasi Model

Dalam unsupervised learning, evaluasi model bisa menjadi tantangan karena tidak ada label sebenarnya yang bisa kita bandingkan dengan hasil clustering. Namun, kita bisa menggunakan metrik seperti Silhouette Score atau Davies-Bouldin Index untuk mengevaluasi seberapa baik hasil clustering.

```
from sklearn.metrics import silhouette_score

score = silhouette_score(X, kmeans.labels_)
print('Silhouette Score:', score)
```

Output:

```
Silhouette Score: 0.7134468304021891
```

Silhouette Score mengukur seberapa dekat setiap sampel dalam satu cluster dengan sampel dalam cluster lain. Nilai yang lebih tinggi berarti hasil clustering lebih baik.

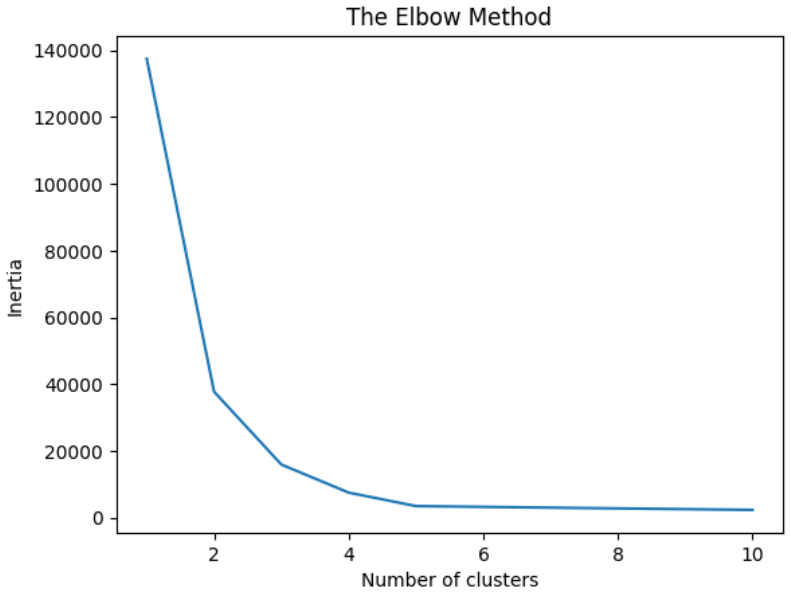
Sampai di sini, kita telah melihat bagaimana cara mengimplementasikan K-Means dalam Python dan cara mengevaluasi hasil clustering. Dengan pengetahuan ini, kamu sekarang memiliki pemahaman yang lebih baik tentang bagaimana algoritma K-Means bekerja dan bagaimana menggunakannya dalam praktik. Seperti yang kamu lihat, dengan beberapa baris kode saja, kita dapat melakukan clustering pada dataset dan memvisualisasikan hasilnya.

3.5.5. Optimisasi Jumlah Cluster

Ingatlah bahwa kita secara arbitrari memilih jumlah cluster sebagai 5 ketika kita menjalankan K-Means. Bagaimana jika kita tidak tahu berapa banyak cluster yang harus kita pilih? Salah satu metode yang populer untuk menentukan jumlah cluster optimal adalah metode Elbow. Metode ini melibatkan menjalankan K-Means untuk berbagai jumlah cluster dan merencanakan inersia (yaitu, jumlah kuadrat jarak dari setiap titik sampel ke centroid terdekatnya).

```
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 11), inertia)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

Output:



Pada plot tersebut, titik "elbow" (siku) - tempat kurva mulai melandai - merupakan indikasi yang baik dari jumlah cluster yang optimal. Namun, perlu diingat bahwa metode Elbow ini tidak selalu bekerja dengan sempurna untuk setiap jenis data.

3.5.6. Menyimpan dan Memuat Model

Setelah kamu menemukan model yang bekerja dengan baik, kamu mungkin ingin menyimpan model tersebut agar bisa digunakan di kemudian hari. Kamu bisa

melakukannya dengan menggunakan modul joblib dari scikit-learn.

```
from joblib import dump, load

# Save the model
dump(kmeans, 'kmeans.joblib')

# Load the model
kmeans_loaded = load('kmeans.joblib')
```

Dengan ini, kamu dapat menyimpan model K-Means yang telah kamu latih dan memuatnya kembali kapan saja kamu butuhkan.

Bab 4. Affinity Propagation

4.1. Pengertian Affinity Propagation

Affinity Propagation adalah teknik machine learning yang digunakan untuk mengidentifikasi cluster data, seperti K-Means, DBSCAN, atau Agglomerative Clustering. Namun, yang membedakan Affinity Propagation dari teknik lainnya adalah cara algoritma ini bekerja.

Affinity Propagation, seperti namanya, mempropagandakan afinitas atau kesamaan antara titik data untuk menentukan cluster. Ini adalah teknik yang cukup baru dalam dunia machine learning, diperkenalkan oleh Brendan J. Frey dan Delbert Dueck pada tahun 2007 dalam artikel mereka "Clustering by Passing Messages Between Data Points".

Berbeda dengan metode clustering lainnya yang seringkali memerlukan penentuan jumlah cluster sebelumnya (seperti K-Means), Affinity Propagation secara otomatis menentukan jumlah cluster berdasarkan data yang diberikan. Hal ini menjadikan Affinity Propagation sebagai pilihan yang menarik, terutama ketika kamu tidak yakin berapa banyak cluster yang seharusnya ada dalam data kamu.

Teknik ini berdasarkan pada konsep 'message passing' atau penyebaran pesan antar titik data. Dengan kata lain, setiap titik data mengirimkan pesan ke titik data lainnya tentang kemungkinan titik itu menjadi 'exemplar' atau perwakilan cluster. Proses ini dilakukan secara iteratif hingga titik-titik data mencapai konsensus tentang titik mana yang menjadi 'exemplar' untuk setiap cluster.

Affinity Propagation memiliki beberapa keuntungan dibandingkan teknik clustering lainnya. Pertama, seperti yang telah disebutkan, Affinity Propagation secara otomatis menentukan jumlah cluster, sehingga mengurangi beban penentuan jumlah cluster yang optimal. Kedua, teknik ini juga mampu menghasilkan cluster yang lebih baik dalam beberapa kasus, terutama ketika data yang kamu miliki memiliki struktur yang kompleks atau tidak biasa.

Namun, Affinity Propagation juga memiliki beberapa kekurangan. Algoritma ini memiliki kompleksitas komputasi yang relatif tinggi, sehingga bisa menjadi sangat lambat saat digunakan pada dataset yang sangat besar. Selain itu, Affinity Propagation juga memerlukan penyetelan parameter yang cermat untuk mendapatkan hasil yang optimal.

Dalam konteks praktis, Affinity Propagation bisa digunakan dalam berbagai aplikasi, seperti pengenalan wajah, pengelompokan dokumen, atau pengelompokan produk dalam sistem rekomendasi.

4.2. Algoritma Affinity Propagation

Setelah mengenal apa itu Affinity Propagation, sekarang waktunya kita mendalami cara kerja algoritma ini. Affinity Propagation mengandalkan pendekatan yang unik dan berbeda dari teknik clustering lainnya. Algoritma ini bekerja dengan memanfaatkan dua jenis "pesan" yang berpindah-pindah antara titik data: "responsibility" dan "availability".

Responsibility (r(i, k)) menggambarkan sejauh mana titik k cocok untuk menjadi exemplar bagi titik i, dibandingkan dengan semua titik lainnya. Responsibility pada dasarnya adalah pesan yang dikirimkan oleh titik data i ke titik k yang menyatakan seberapa layak titik k menjadi exemplar bagi i.

Availability (a(i, k)) menggambarkan sejauh mana titik i memilih titik k sebagai exemplar-nya, dengan mempertimbangkan semua titik lainnya yang juga mungkin memilih k sebagai exemplar mereka. Availability adalah pesan yang dikirimkan oleh titik data k ke titik i yang menyatakan seberapa layak titik i memilih k sebagai exemplar.

Konsep ini mungkin sedikit membingungkan, jadi mari kita coba ilustrasikan dengan contoh. Bayangkan kamu berada di sebuah pesta dan ingin memilih seseorang untuk menjadi DJ. Responsibility dalam konteks ini bisa diartikan sebagai seberapa cocok seseorang untuk menjadi DJ, berdasarkan preferensi musik kamu dan pengetahuan kamu tentang selera musik orang tersebut. Availability, di sisi lain, bisa diartikan sebagai seberapa cocok kamu memilih seseorang tersebut menjadi DJ, dengan mempertimbangkan semua orang lain di pesta yang mungkin juga ingin orang tersebut menjadi DJ.

Algoritma Affinity Propagation berjalan secara iteratif, dimana setiap titik data terus-menerus memperbarui dan mengirimkan pesan "responsibility" dan "availability" ke titik data lainnya. Proses ini diulangi hingga konvergensi, yaitu ketika pesan-pesan tersebut stabil dan tidak berubah lagi. Pada titik ini, titik-titik data telah mencapai konsensus tentang siapa yang harus menjadi exemplar untuk setiap cluster.

Secara matematis, perhitungan "responsibility" dan "availability" dapat dijelaskan sebagai berikut:

Perhitungan responsibility:

$$r(i,\ k) \leftarrow s(i,\ k)\ -\ max\{a(i,\ k')\ +\ s(i,\ k')\ for\ all\ k' \neq k\}$$

Dimana s(i, k) adalah nilai similarity antara titik i dan k (biasanya dihitung sebagai negatif jarak kuadrat antara dua titik), dan max{...} adalah nilai maksimum dari semua k' yang tidak sama dengan k.

Perhitungan availability:

$$a(i,\ k) \leftarrow min\{0,\ r(k,\ k)\ \ +\ \ sum\{max\{0,\ r(i',\ k)\}\ for\ all\ i' \neq i,\ k\}\}$$

untuk i ≠ k, dan

$$a(k,\ k) \leftarrow sum\{max\{0,\ r(i',\ k)\}\ for\ all\ i' \neq k\}$$

Pada akhirnya, titik data i akan memilih titik data k sebagai exemplar-nya jika dan hanya jika r(i, k) + a(i, k) > 0. Jadi, titik data akan memilih titik lain sebagai exemplar jika total responsibility dan availability yang diterimanya dari titik lain tersebut positif.

Sejauh ini kita telah membahas teori dasar di balik Affinity Propagation. Dalam prakteknya, algoritma ini memiliki beberapa keuntungan dan kerugian. Salah satu keuntungan utamanya adalah bahwa kita tidak perlu menentukan jumlah cluster sejak awal, yang biasanya menjadi tantangan utama dalam teknik clustering lain seperti K-means. Selain itu, Affinity Propagation juga bisa menghasilkan cluster yang lebih kompleks dan tidak terbatas pada bentuk yang sferis atau globular, seperti yang biasanya kita lihat di K-means atau Gaussian Mixture.

Namun, Affinity Propagation juga memiliki beberapa kelemahan. Salah satunya adalah efisiensi komputasi. Algoritma ini cenderung membutuhkan waktu dan memori yang lebih besar dibandingkan dengan algoritma clustering lainnya, terutama ketika dihadapkan dengan dataset yang sangat besar. Oleh karena itu, Affinity Propagation mungkin tidak ideal untuk aplikasi dengan jumlah data yang sangat besar. Meski demikian, Affinity Propagation masih menjadi alat yang sangat berguna dan efektif dalam berbagai aplikasi, terutama ketika jumlah cluster tidak diketahui sebelumnya dan bentuk cluster bisa beragam.

Untuk lebih memahami cara kerja algoritma Affinity Propagation, subbab berikutnya akan membahas tentang dua konsep penting dalam algoritma ini, yaitu preferensi dan similarity.

4.3. Preferensi dan Similarity

Ketika kita berbicara tentang algoritma Affinity Propagation, ada dua konsep yang sangat penting untuk dipahami, yaitu preferensi dan similarity. Keduanya berperan penting dalam menentukan bagaimana data dipartisi menjadi berbagai cluster.

Preferensi dalam konteks Affinity Propagation merujuk pada sejauh mana titik data lebih memilih untuk memilih dirinya sendiri sebagai exemplar atau pusat cluster. Dalam hal ini, preferensi biasanya ditentukan oleh nilai median dari similarity antara titik data dengan semua titik lainnya dalam dataset. Dengan kata lain, jika preferensi tinggi, titik data tersebut cenderung menjadi pusat cluster.

Pada awalnya, preferensi diatur ke nilai yang sama untuk semua titik data. Namun, nilai ini dapat disesuaikan untuk mendorong pembentukan jumlah cluster yang lebih besar atau lebih kecil. Misalnya, jika preferensi diatur lebih tinggi, lebih banyak titik data akan

menjadi pusat cluster, yang berarti akan ada lebih banyak cluster. Sebaliknya, jika preferensi diatur lebih rendah, maka akan ada lebih sedikit cluster yang terbentuk.

Similarity adalah ukuran lain yang sangat penting dalam Affinity Propagation. Similarity, atau kesamaan, adalah ukuran sejauh mana dua titik data mirip satu sama lain. Dalam konteks ini, similarity biasanya diukur dengan jarak negatif antara dua titik data. Oleh karena itu, jika dua titik sangat dekat satu sama lain, mereka memiliki similarity yang tinggi, dan jika mereka jauh apart, mereka memiliki similarity yang rendah.

Dalam algoritma Affinity Propagation, nilai similarity digunakan untuk mengupdate nilai responsibility dan availability, yang pada gilirannya digunakan untuk menentukan exemplar untuk setiap titik data. Oleh karena itu, similarity memainkan peran kunci dalam menentukan bagaimana data dipartisi menjadi cluster.

Pemahaman tentang preferensi dan similarity sangat penting untuk memahami bagaimana Affinity Propagation bekerja. Dengan memahami kedua konsep ini, kamu akan lebih mudah memahami bagaimana algoritma ini membuat keputusan tentang bagaimana mempartisi data menjadi berbagai cluster.

4.4 Implementasi Affinity Propagation dalam Python

Sekarang saatnya kita melihat bagaimana Affinity Propagation dapat diimplementasikan dalam Python. Python adalah bahasa pemrograman yang sering digunakan dalam bidang data science dan machine learning, dan memiliki banyak library yang memudahkan implementasi algoritma seperti Affinity Propagation.

Untuk mengimplementasikan Affinity Propagation, kita akan menggunakan library Scikit-Learn, yang adalah library machine learning populer dalam Python yang menyediakan berbagai algoritma, termasuk Affinity Propagation.

4.4.1. Persiapan Data

Pertama-tama, kita perlu data untuk bekerja. Untuk tujuan demonstrasi ini, kita akan menggunakan data sintetis yang dibuat dengan bantuan fungsi make_blobs dari Scikit-Learn. Fungsi ini akan menghasilkan data yang terbagi dalam beberapa cluster.

```
from sklearn.datasets import make_blobs

# membuat data sintetis
X, _ = make_blobs(n_samples=1000, centers=3, random_state=42)

print(X[:5])  # mencetak 5 data pertama
```

Output:

```
[[-6.59633932 -7.13901457]
 [-6.13753182 -6.58081701]
 [ 5.19820575  2.04917508]
 [-2.96855852  8.16444176]
 [-2.76878897  7.51114318]]
```

Pada kode di atas, make_blobs digunakan untuk membuat data sintetis dengan 1000 sampel yang dibagi menjadi tiga cluster. random_state digunakan untuk memastikan hasil yang konsisten setiap kali kode dijalankan.

4.4.2. Membuat Model Affinity Propagation

Setelah kita memiliki data, langkah berikutnya adalah membuat model Affinity Propagation. Kita bisa melakukannya dengan menggunakan class AffinityPropagation dari Scikit-Learn.

```
from sklearn.cluster import AffinityPropagation

# membuat model Affinity Propagation
model = AffinityPropagation(random_state=42)
```

Pada kode di atas, kita membuat instance dari class AffinityPropagation dan menyimpannya dalam variabel model. Seperti sebelumnya, kita menggunakan random_state untuk memastikan hasil yang konsisten.

4.4.3. Training Model

Setelah model dibuat, kita bisa melatihnya menggunakan data yang kita miliki. Kita bisa melakukannya dengan memanggil metode fit pada model.

```
# melatih model
model.fit(X)
```

Metode fit akan menjalankan algoritma Affinity Propagation pada data, belajar dari data dan menyesuaikan parameter model sehingga model dapat membuat prediksi yang akurat tentang cluster yang berbeda dalam data.

4.4.4. Evaluasi Model

Setelah model dilatih, kita bisa mengevaluasinya. Dalam kasus clustering, kita biasanya melihat seberapa baik model telah mempartisi data. Untuk tujuan ini, kita bisa melihat label cluster yang telah ditentukan oleh model.

```
# mendapatkan label cluster
labels = model.labels_

print(labels[:5])  # mencetak 5 label pertama
```

Output:

```
[54 54 16 84 84]
```

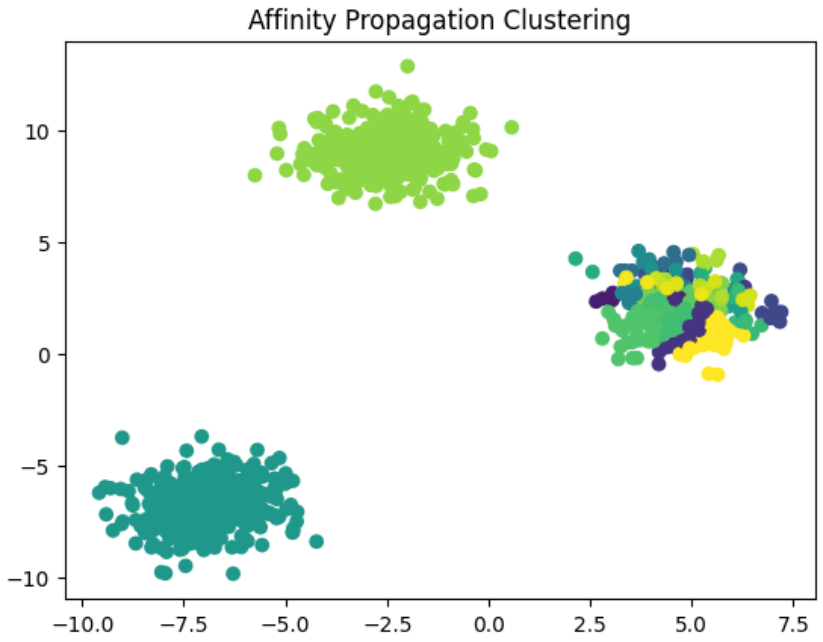
Pada kode di atas, kita menggunakan atribut labels_ pada model untuk mendapatkan label cluster untuk setiap titik data. Label ini menunjukkan cluster mana yang ditugaskan model untuk setiap titik data.

4.4.5. Visualisasi Hasil

Untuk memahami lebih baik bagaimana model telah mempartisi data, kita bisa memvisualisasikannya. Kita akan menggunakan library matplotlib, yang adalah library visualisasi populer dalam Python.

```
import matplotlib.pyplot as plt

# plot data
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.title('Affinity Propagation Clustering')
plt.show()
```

Output:



Pada kode di atas, kita menggunakan fungsi scatter untuk membuat scatter plot dari data. Kami menggunakan dua fitur pertama dari data sebagai koordinat x dan y, dan kami menggunakan label sebagai warna titik. Hasilnya adalah plot di mana titik data yang berada dalam cluster yang sama memiliki warna yang sama.

Dengan melihat visualisasi ini, kamu dapat melihat bagaimana Affinity Propagation telah

membagi data menjadi cluster. Kamu mungkin melihat bahwa beberapa cluster lebih padat daripada yang lain, atau bahwa ada beberapa titik yang tampaknya berada di cluster yang salah. Ini adalah indikasi dari bagaimana Affinity Propagation bekerja, dan bagaimana ia mencoba menemukan struktur dalam data.

Bab 5. Mean Shift

5.1 Pengertian Mean Shift

Mean Shift adalah salah satu algoritma clustering dalam machine learning yang tidak memerlukan kita untuk menentukan jumlah cluster yang diinginkan. Berbeda dengan K-Means atau DBSCAN, jumlah cluster dalam Mean Shift akan ditentukan secara otomatis berdasarkan data. Algoritma ini biasanya digunakan pada aplikasi seperti pemrosesan gambar dan pengenalan objek, namun bisa juga digunakan untuk analisis data lainnya.

Untuk lebih memahami apa itu Mean Shift, bayangkan kamu sedang berada di padang pasir yang luas dan kamu buta. Kamu ingin mencapai puncak bukit yang ada di dekatmu, tapi karena kamu buta, kamu tidak bisa melihatnya. Jadi apa yang bisa kamu lakukan? Kamu bisa merasakan lereng bukit dan mulai berjalan ke arah yang kamu rasa lebih tinggi. Kamu akan terus berjalan ke arah yang lebih tinggi sampai kamu tidak merasakan kenaikan lagi. Di titik itu, kamu bisa berasumsi bahwa kamu berada di puncak bukit.

Ini adalah inti dari Mean Shift. Algoritma ini berusaha mencari "puncak" dalam distribusi data, tempat di mana kepadatan data paling tinggi. Dalam konteks algoritma ini, "puncak" ini disebut mode. Mean Shift akan mencoba menemukan mode ini dengan cara yang sama seperti kamu mencari puncak bukit: dengan bergerak ke arah yang memiliki kepadatan data yang lebih tinggi.

Tapi bagaimana Mean Shift bisa mengetahui di mana kepadatan data paling tinggi? Mean Shift melakukan ini dengan cara mempertimbangkan titik data di sekitarnya dalam jangkauan tertentu, yang disebut bandwidth. Untuk setiap titik data, Mean Shift akan menghitung rata-rata dari titik-titik yang berada dalam bandwidth ini, dan kemudian menggeser titik data tersebut ke rata-rata ini. Proses ini diulang terus-menerus hingga titik data tidak bergerak lagi atau bergerak kurang dari batas tertentu.

Dengan demikian, titik data yang awalnya tersebar akan bergerak ke arah yang memiliki kepadatan data yang lebih tinggi, dan akhirnya membentuk cluster di sekitar mode distribusi data. Jumlah cluster yang dihasilkan oleh Mean Shift adalah jumlah mode yang ditemukan dalam data.

Hal penting yang perlu diingat tentang Mean Shift adalah bahwa hasilnya sangat bergantung pada pilihan bandwidth. Bandwidth yang terlalu kecil mungkin akan menghasilkan terlalu banyak cluster (karena setiap titik data bisa menjadi mode sendiri), sementara bandwidth yang terlalu besar mungkin akan menghasilkan terlalu sedikit cluster (karena semua titik data bisa bergerak ke satu mode saja). Maka, pemilihan bandwidth yang tepat sangat penting dalam algoritma ini.

Sebagai kesimpulan, Mean Shift adalah algoritma clustering yang kuat yang tidak memerlukan penentuan jumlah cluster sebelumnya. Algoritma ini mencari mode dalam distribusi data dan membentuk cluster di sekitar mode tersebut. Meskipun sangat berguna dalam banyak aplikasi, Mean Shift sangat bergantung pada pemilihan bandwidth, yang bisa menjadi tantangan tersendiri.

5.2 Algoritma Mean Shift

Setelah memahami apa itu Mean Shift, mari kita membahas lebih dalam tentang bagaimana algoritma ini bekerja. Seperti yang sudah disebutkan sebelumnya, Mean Shift mencari mode dalam distribusi data dengan cara memindahkan setiap titik ke arah kepadatan data yang lebih tinggi. Namun, bagaimana cara algoritma ini menentukan arah tersebut? Mari kita jelaskan langkah-langkahnya.

Pertama-tama, Mean Shift memulai dengan memilih titik acak dalam data sebagai titik awal. Jika kamu menggunakan seluruh titik data sebagai titik awal, maka algoritma ini akan berakhir dengan menemukan semua mode yang ada dalam data. Namun, dalam praktiknya, kamu mungkin hanya memilih sebagian titik data sebagai titik awal untuk menghemat waktu komputasi.

Selanjutnya, untuk setiap titik awal, Mean Shift mempertimbangkan titik-titik data yang berada dalam jangkauan bandwidthnya. Jangkauan ini biasanya berbentuk lingkaran (untuk data dua dimensi) atau bola (untuk data tiga dimensi atau lebih), dengan titik awal di pusatnya. Titik-titik data di dalam jangkauan ini disebut titik lingkungan.

Kemudian, Mean Shift menghitung rata-rata dari titik-titik lingkungan ini. Rata-rata ini, atau mean, adalah titik baru yang akan menjadi titik awal di iterasi selanjutnya. Proses ini diulang, dengan titik baru menjadi pusat dari jangkauan bandwidth dan menghitung mean baru dari titik-titik lingkungan.

Algoritma ini berlanjut sampai titik awal tidak bergerak lagi atau bergerak kurang dari batas tertentu. Pada titik ini, kita bisa mengatakan bahwa titik awal telah berkonvergensi ke mode distribusi data. Setelah semua titik awal berkonvergensi, Mean Shift akan mengelompokkan titik-titik data berdasarkan mode yang mereka konvergensi. Setiap grup titik data yang konvergensi ke mode yang sama akan menjadi cluster.

Salah satu kelebihan dari algoritma Mean Shift adalah kemampuannya untuk menemukan bentuk cluster yang tidak bulat, yang bisa menjadi tantangan bagi algoritma clustering lain seperti K-Means. Namun, seperti yang sudah disebutkan sebelumnya, Mean Shift sangat bergantung pada pemilihan bandwidth. Bandwidth yang terlalu kecil mungkin akan menghasilkan terlalu banyak cluster, sementara bandwidth yang terlalu besar mungkin akan menghasilkan terlalu sedikit cluster.

Sekarang, kamu mungkin bertanya, bagaimana cara menentukan bandwidth yang tepat?

Sayangnya, tidak ada jawaban yang pasti untuk pertanyaan ini. Pemilihan bandwidth biasanya dilakukan berdasarkan pengetahuan sebelumnya tentang data atau dengan mencoba berbagai nilai dan melihat hasilnya. Meskipun ini bisa menjadi tantangan, ini juga memberikan Mean Shift fleksibilitas untuk menyesuaikan dengan berbagai jenis data.

Sekarang kamu sudah memahami bagaimana algoritma Mean Shift bekerja. Meskipun algoritma ini bisa tampak rumit pada awalnya, intinya cukup sederhana: mencari mode dalam distribusi data dengan memindahkan setiap titik ke arah yang memiliki kepadatan data yang lebih tinggi. Ini adalah prinsip dasar yang sama yang kamu gunakan ketika mencoba mencapai puncak bukit dalam gelap: merasakan arah yang lebih tinggi dan bergerak ke arah tersebut.

Pemahaman ini seharusnya cukup untuk memahami bagaimana Mean Shift bekerja. Namun, jika kamu ingin menggunakan algoritma ini dalam aplikasi nyata, kamu mungkin perlu memahami lebih banyak detail, seperti cara mengoptimalkan pemilihan titik awal atau cara menentukan bandwidth yang tepat. Untuk itu, ada banyak sumber belajar yang bisa kamu gunakan, mulai dari buku teks hingga tutorial online.

Terakhir, penting untuk diingat bahwa Mean Shift hanyalah salah satu dari banyak algoritma clustering yang ada. Meskipun algoritma ini memiliki kelebihan, seperti kemampuan untuk menemukan bentuk cluster yang tidak bulat dan tidak perlu menentukan jumlah cluster sebelumnya, itu juga memiliki kelemahan, seperti dependensi yang kuat pada pemilihan bandwidth. Oleh karena itu, sebelum memutuskan untuk menggunakan Mean Shift, pastikan untuk mempertimbangkan algoritma clustering lain dan memilih yang paling sesuai dengan kebutuhanmu.

5.3 Pemilihan Bandwidth dalam Mean Shift

Memahami bagaimana Mean Shift bekerja adalah satu hal, tapi menerapkannya pada kumpulan data nyata adalah tantangan yang sama sekali berbeda. Salah satu aspek yang paling sulit – dan paling penting – dari menggunakan Mean Shift adalah memilih bandwidth yang tepat. Dalam subbab ini, kita akan membahas mengapa bandwidth sangat penting dalam Mean Shift, bagaimana efeknya terhadap hasil clustering, dan beberapa metode yang bisa kamu gunakan untuk memilih bandwidth yang optimal.

5.3.1. Peran Bandwidth dalam Mean Shift

Sebelum kita membahas cara memilih bandwidth, penting untuk memahami apa itu bandwidth dan perannya dalam algoritma Mean Shift. Bandwidth adalah parameter yang menentukan sejauh mana pengaruh titik data terhadap pergeseran mean. Dengan kata lain, bandwidth menentukan "jendela" di sekitar titik data yang digunakan untuk menghitung mean berikutnya.

Dengan demikian, bandwidth mempengaruhi dua aspek utama dari Mean Shift: kecepatan

konvergensi dan jumlah cluster yang dihasilkan. Bandwidth yang besar akan mencakup lebih banyak titik data dalam setiap perhitungan mean, yang berarti algoritma akan cenderung konvergen lebih cepat. Namun, hal ini juga berarti bahwa algoritma akan cenderung menghasilkan jumlah cluster yang lebih sedikit, karena lebih banyak titik data akan "ditarik" ke dalam cluster yang sama.

Sebaliknya, bandwidth yang kecil akan mencakup lebih sedikit titik data dalam setiap perhitungan mean, yang berarti algoritma akan cenderung konvergen lebih lambat. Namun, hal ini juga berarti bahwa algoritma akan cenderung menghasilkan jumlah cluster yang lebih banyak, karena lebih sedikit titik data akan "ditarik" ke dalam cluster yang sama.

5.3.2. Efek Bandwidth terhadap Hasil Clustering

Efek bandwidth terhadap hasil clustering bisa diilustrasikan dengan mudah menggunakan kumpulan data sederhana. Misalnya, bayangkan kamu memiliki kumpulan data yang terdiri dari tiga grup titik data yang berjauhan. Jika kamu menggunakan bandwidth yang besar, Mean Shift mungkin akan menganggap ketiga grup tersebut sebagai satu cluster, karena jendela yang besar mencakup semua titik data.

Sebaliknya, jika kamu menggunakan bandwidth yang kecil, Mean Shift mungkin akan menghasilkan lebih dari tiga cluster, karena jendela yang kecil hanya mencakup sebagian kecil titik data dalam setiap perhitungan mean. Dalam kasus ekstrem, setiap titik data bisa menjadi cluster tersendiri!

5.3.3. Cara Memilih Bandwidth yang Tepat

Dengan begitu pentingnya peran bandwidth dalam Mean Shift, pertanyaannya kemudian adalah: bagaimana cara memilih bandwidth yang tepat? Sayangnya, tidak ada jawaban yang pasti untuk pertanyaan ini, karena bandwidth yang optimal sangat tergantung pada kumpulan data dan tujuan analisis kamu.

Namun, ada beberapa metode yang bisa kamu coba. Salah satu metode paling umum adalah menggunakan cross-validation. Dengan metode ini, kamu mencoba beberapa nilai bandwidth yang berbeda, menerapkan Mean Shift untuk setiap nilai, dan melihat mana yang menghasilkan hasil yang terbaik berdasarkan beberapa metrik tertentu. Misalnya, kamu bisa menggunakan Silhouette score atau Dunn index sebagai metrik. Namun, perlu diingat bahwa metode ini bisa sangat memakan waktu jika kumpulan data kamu sangat besar atau jika kamu mencoba banyak nilai bandwidth yang berbeda.

Metode lain yang bisa kamu coba adalah "rule of thumb". Dalam konteks Mean Shift, "rule of thumb" biasanya berarti mengatur bandwidth sama dengan standar deviasi dari kumpulan data. Walaupun metode ini lebih cepat dan mudah dibandingkan dengan cross-validation, hasilnya mungkin tidak sebaik jika kamu menggunakan cross-validation.

Selain itu, ada juga beberapa metode yang lebih canggih, seperti menggunakan estimasi

densitas kernel untuk memilih bandwidth secara otomatis. Metode ini melibatkan penggunaan algoritma lain untuk mengestimasi distribusi densitas dari kumpulan data, dan kemudian menggunakan informasi ini untuk memilih bandwidth. Namun, metode ini cenderung lebih kompleks dan memerlukan pemahaman yang lebih mendalam tentang statistik.

5.3.4. Pentingnya Eksperimen

Yang perlu diingat adalah bahwa tidak ada metode yang bisa menjamin kamu akan mendapatkan bandwidth yang "sempurna". Pada akhirnya, pemilihan bandwidth adalah proses yang melibatkan banyak percobaan dan kesalahan. Kamu mungkin perlu mencoba beberapa metode yang berbeda, atau bahkan kombinasi dari beberapa metode, sebelum kamu menemukan bandwidth yang paling cocok untuk kumpulan data dan tujuan analisis kamu.

Selain itu, penting juga untuk memahami bahwa Mean Shift, seperti algoritma clustering lainnya, bukanlah alat yang "sempurna". Hasil clustering selalu tergantung pada data dan parameter yang kamu gunakan, dan selalu ada kemungkinan bahwa hasil yang kamu dapatkan tidak sesuai dengan apa yang kamu harapkan. Oleh karena itu, penting untuk selalu menginterpretasikan hasil clustering dengan hati-hati dan kritis.

5.4. Implementasi Mean Shift dalam Python

Pada bagian ini, kita akan membahas cara mengimplementasikan algoritma Mean Shift menggunakan Python. Kita akan menggunakan library scikit-learn, yang merupakan library machine learning populer dalam Python, dan menyediakan implementasi algoritma Mean Shift yang siap pakai.

5.4.1. Importing Necessary Libraries

Langkah pertama dalam proses ini adalah mengimpor library yang diperlukan. Library yang kita perlukan adalah numpy, matplotlib, dan scikit-learn.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift
from sklearn.datasets import make_blobs
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import silhouette_score
```

5.4.2. Membuat Data Sintetis

Setelah mengimpor library yang diperlukan, langkah selanjutnya adalah membuat data sintetis. Kita akan menggunakan fungsi make_blobs dari scikit-learn untuk menghasilkan kumpulan data dengan tiga pusat cluster.

```
centers = [[1, 1], [-1, -1], [1, -1]]
X, _ = make_blobs(n_samples=5000, centers=centers, cluster_std=0.6)
```

5.4.3. Membuat dan Melatih Model Mean Shift

Langkah berikutnya adalah membuat dan melatih model Mean Shift. Untuk melakukan ini, kita menggunakan kelas MeanShift dari scikit-learn. Kita perlu memilih bandwidth yang sesuai, yang merupakan parameter penting dalam algoritma Mean Shift.

```
ms = MeanShift(bandwidth=0.8)
ms.fit(X)
```

5.4.4. Evaluasi Model

Setelah kita melatih model dan melakukan tuning hyperparameter, langkah selanjutnya adalah mengevaluasi model. Salah satu cara untuk melakukan ini adalah dengan menggunakan skor siluet, yang merupakan ukuran seberapa baik setiap titik dikelompokkan dengan titik lain dalam cluster yang sama dan seberapa jauh setiap titik dari titik dalam cluster lain.

```
labels = ms.labels_
score = silhouette_score(X, labels)

print("Silhouette score: ", score)
```

Output:

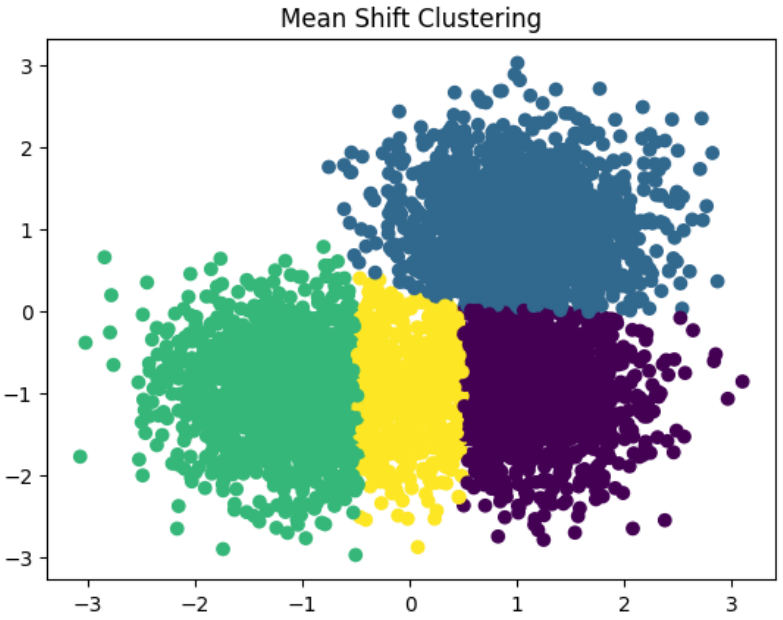
```
Silhouette score:  0.3828118629927535
```

5.4.5. Visualisasi Hasil

Terakhir, kita akan melakukan visualisasi hasil dengan matplotlib. Kita akan membuat scatter plot di mana setiap titik data direpresentasikan oleh titik dan warna titik menunjukkan cluster tempat titik tersebut berada.

```
plt.scatter(X[:,0], X[:,1], c=labels)
plt.title('Mean Shift Clustering')
plt.show()
```

Output:



Penting untuk diingat bahwa, kamu harus selalu melakukan eksplorasi dan eksperimen dengan data dan parameter yang berbeda-beda. Misalnya, kamu bisa mencoba menggunakan jumlah data yang berbeda, pusat cluster yang berbeda, atau standard deviasi yang berbeda saat membuat data sintetis. Kamu juga bisa mencoba menggunakan nilai bandwidth yang berbeda untuk melihat bagaimana hasil clustering berubah. Dalam setiap langkah dari proses ini, cobalah untuk memahami apa yang sedang terjadi dan bagaimana setiap perubahan yang kamu buat mempengaruhi hasil.

Berikutnya, setelah melihat visualisasi hasil, pikirkan tentang apa yang bisa kamu lakukan untuk meningkatkan hasil. Apakah terlalu banyak atau terlalu sedikit cluster? Apakah ada titik yang tampaknya berada di cluster yang salah? Jika ya, apa yang bisa kamu lakukan untuk memperbaikinya? Ingatlah bahwa Mean Shift adalah algoritma yang sangat kuat dan fleksibel, tetapi juga bisa menjadi sangat lambat jika kamu memiliki banyak data. Oleh karena itu, kamu mungkin perlu mencari cara untuk mengoptimalkan proses atau menggunakan algoritma lain jika kamu bekerja dengan set data yang sangat besar.

Bab 6. Spectral Clustering

6.1. Pengertian Spectral Clustering

Spectral Clustering adalah salah satu teknik dalam pembelajaran mesin yang menggunakan konsep matematika dari spektrum (eigenvalues) dari matriks untuk melakukan pengelompokan data. Teknik ini memiliki banyak keunggulan dibandingkan beberapa metode lainnya dan sering digunakan dalam berbagai bidang, termasuk pengenalan pola, pengolahan citra, jaringan sosial, bioinformatika, dan banyak lagi.

Untuk memahami Spectral Clustering, kamu perlu mengerti bahwa algoritma ini mengoperasikan data dalam ruang dimensi tinggi dan mencoba menemukan struktur dalam data tersebut. Untuk melakukan hal ini, Spectral Clustering menggunakan informasi dari grafik yang mewakili data. Dalam grafik ini, setiap titik data diwakili sebagai simpul dan kedekatan antara titik data diwakili sebagai tepi yang menghubungkan simpul.

Dalam konteks pengelompokan, tujuan utamanya adalah untuk memisahkan simpul-simpul dalam grafik ke dalam grup atau "komunitas" yang berbeda, di mana simpul dalam grup yang sama lebih mirip satu sama lain dibandingkan dengan simpul di grup lain. Untuk mencapai tujuan ini, Spectral Clustering menggunakan konsep dari aljabar linier dan teori grafik, seperti matriks adjacensi, Laplacian grafik, dan nilai dan vektor eigen.

Matriks adjacensi adalah matriks persegi yang digunakan untuk mewakili grafik berarah atau tak berarah. Setiap elemen dalam matriks menunjukkan apakah pasangan simpul yang sesuai dihubungkan oleh tepi dalam grafik.

Sementara itu, Laplacian grafik adalah matriks yang dihasilkan dari matriks adjacensi dan digunakan untuk mempelajari sifat-sifat grafik. Laplacian grafik memiliki banyak properti menarik yang menjadikannya alat yang berguna dalam analisis spektral grafik.

Nilai dan vektor eigen adalah konsep kunci dalam aljabar linier dan memiliki banyak aplikasi dalam berbagai bidang. Dalam konteks Spectral Clustering, vektor eigen dari Laplacian grafik digunakan untuk memetakan data ke dalam ruang dimensi lebih rendah, di mana algoritma pengelompokan seperti K-Means kemudian dapat diterapkan.

Salah satu keuntungan utama dari Spectral Clustering adalah kemampuannya untuk menemukan cluster yang tidak perlu berbentuk hiper bola dalam ruang asli. Ini berarti bahwa, berbeda dengan algoritma seperti K-Means, Spectral Clustering bisa menangani cluster dengan bentuk yang kompleks.

Namun, Spectral Clustering juga memiliki beberapa kelemahan. Misalnya, ia memiliki biaya komputasi yang tinggi, terutama untuk dataset berukuran besar, dan pemilihan parameter bisa menjadi tantangan.

Keseluruhan, Spectral Clustering adalah algoritma pengelompokan yang kuat dan fleksibel yang dapat digunakan untuk berbagai jenis data. Dengan pemahaman yang baik tentang bagaimana algoritma ini bekerja, kamu dapat menggunakan Spectral Clustering untuk menemukan struktur dalam data dan mendapatkan wawasan yang berharga.

6.2. Graph Laplacian dan Eigen Decomposition

Sekarang, kita akan membahas dua konsep penting dalam spectral clustering, yaitu Graph Laplacian dan Eigen Decomposition. Konsep-konsep ini merupakan batu penjuru dalam implementasi algoritma spectral clustering dan memahaminya akan membantu kamu memahami bagaimana algoritma ini bekerja.

6.2.1. Graph Laplacian

Grafik Laplacian adalah representasi matriks dari grafik yang digunakan dalam berbagai algoritma dalam teori grafik, termasuk spectral clustering. Secara intuitif, Grafik Laplacian memberikan gambaran tentang struktur dan karakteristik dari sebuah grafik.

Untuk membuat Graph Laplacian, kamu perlu melakukan beberapa langkah. Pertama, kamu harus membuat matriks adjacensi. Matriks adjacensi adalah matriks persegi dimana setiap elemen i, j adalah 1 jika ada tepi yang menghubungkan simpul i dan j, dan 0 jika tidak.

Selanjutnya, kamu perlu membuat matriks derajat. Matriks derajat adalah matriks diagonal dimana setiap elemen i, i adalah jumlah tepi yang terhubung dengan simpul i. Ini pada dasarnya menunjukkan berapa banyak "tetangga" yang dimiliki setiap simpul.

Setelah itu, Grafik Laplacian didefinisikan sebagai matriks derajat dikurangi matriks adjacensi. Secara matematis, L = D - A, dimana L adalah Laplacian, D adalah matriks derajat, dan A adalah matriks adjacensi. Hasilnya adalah matriks persegi yang menunjukkan struktur dan karakteristik dari grafik.

6.2.2. Eigen Decomposition

Eigen Decomposition adalah proses menguraikan matriks menjadi vektor dan nilai eigen-nya. Nilai eigen dari matriks adalah nilai yang tidak mengubah arah vektor saat matriks tersebut dikalikan dengan vektor. Vektor eigen adalah vektor yang sesuai dengan nilai eigen tersebut.

Dalam konteks spectral clustering, kita tertarik pada vektor eigen dari Graph Laplacian. Alasan di balik ini adalah bahwa vektor eigen dari Graph Laplacian dapat digunakan untuk

menemukan struktur dalam grafik, yang pada gilirannya dapat digunakan untuk pengelompokan.

Sebagai contoh, vektor eigen terkecil dari Graph Laplacian, yang juga dikenal sebagai vektor eigen terkecil atau Fiedler vector, dapat digunakan untuk membagi grafik menjadi dua komponen yang terhubung. Hal ini dapat dilakukan dengan mengatur titik potong pada vektor eigen dan kemudian membagi simpul menjadi dua grup berdasarkan sisi titik potong mereka berada.

Dengan menggunakan beberapa vektor eigen terkecil, kita bisa melakukan pengelompokan yang lebih kompleks. Secara umum, vektor eigen memberikan gambaran tentang 'frekuensi' atau 'getaran' dalam grafik, dan oleh karena itu dapat digunakan untuk menemukan struktur dalam data.

6.3. Algoritma Spectral Clustering

Dalam subbab ini, kita akan membahas algoritma spectral clustering secara lebih mendalam. Seperti yang telah kita lihat sebelumnya, spectral clustering adalah metode pengelompokan yang didasarkan pada analisis spektrum (nilai eigen) dari Graph Laplacian. Sekarang, kita akan melihat langkah-langkah yang terlibat dalam algoritma ini secara lebih rinci.

6.3.1. Langkah-Langkah dalam Algoritma Spectral Clustering

Algoritma spectral clustering terdiri dari beberapa langkah berikut:

1. Bentuk Matriks Similaritas: Langkah pertama dalam algoritma spectral clustering adalah pembentukan matriks similaritas. Dalam konteks ini, similaritas bisa diartikan sebagai seberapa dekat dua titik data satu sama lain. Matriks ini biasanya ditandai dengan S dan setiap elemennya S(i, j) menggambarkan tingkat kemiripan antara titik data i dan j.
2. Bentuk Graph Laplacian: Berdasarkan matriks similaritas, kita kemudian membentuk Graph Laplacian. Seperti yang sudah kita bahas sebelumnya, ini melibatkan pembuatan matriks derajat dan matriks adjacensi dan kemudian mengurangi matriks adjacensi dari matriks derajat.
3. Lakukan Eigen Decomposition pada Graph Laplacian: Setelah Graph Laplacian dibentuk, langkah selanjutnya adalah melakukan Eigen Decomposition pada matriks ini. Ini menghasilkan serangkaian nilai dan vektor eigen, yang dapat digunakan untuk menemukan struktur dalam data.
4. Pilih k vektor eigen terkecil: Dari vektor-vektor eigen yang dihasilkan, kita memilih k vektor eigen terkecil. Nilai k ini biasanya ditentukan sebelumnya dan sama dengan jumlah kluster yang kita cari.
5. Buat matriks Y dari vektor eigen yang dipilih: Vektor eigen yang dipilih kemudian digunakan untuk membentuk matriks Y. Setiap vektor eigen menjadi kolom dalam matriks ini.

6. Lakukan pengelompokan pada baris matriks Y: Terakhir, kita melakukan pengelompokan pada baris-baris matriks Y. Ini bisa dilakukan dengan menggunakan teknik pengelompokan seperti k-means.

6.3.2. Intuisi di Balik Algoritma

Intuisi di balik algoritma spectral clustering adalah bahwa, dengan menganalisis spektrum (nilai eigen) dari Graph Laplacian, kita dapat menemukan struktur dalam data yang tidak dapat ditemukan dengan teknik pengelompokan tradisional.

Pada dasarnya, spectral clustering mencoba untuk membagi grafik sedemikian rupa sehingga jumlah tepi antara grup adalah minimum. Dengan kata lain, ia mencoba untuk menemukan pemisahan dalam grafik di mana sebanyak mungkin tepi berada di dalam grup dan sebanyak mungkin tepi berada antara grup.

Melalui proses ini, spectral clustering dapat mengidentifikasi kluster yang tidak hanya terpisah secara spasial, tetapi juga memiliki struktur atau pola tertentu. Ini membuatnya menjadi alat yang kuat untuk analisis data yang kompleks dan struktur yang tidak biasa.

6.3.3. Kelebihan dan Kekurangan Spectral Clustering

Seperti algoritma clustering lainnya, spectral clustering memiliki kelebihan dan kekurangan tertentu yang perlu kamu pertimbangkan.

Kelebihan utama dari spectral clustering adalah kemampuannya untuk mengidentifikasi kluster yang memiliki struktur yang kompleks. Ini mencakup kluster yang memiliki bentuk yang tidak biasa atau yang tumpang tindih satu sama lain. Selain itu, spectral clustering tidak mengasumsikan bentuk tertentu dari kluster, seperti yang dilakukan oleh algoritma seperti k-means. Ini berarti bahwa ia dapat menemukan kluster yang tidak bisa ditemukan oleh algoritma lain.

Namun, spectral clustering juga memiliki beberapa kekurangan. Pertama, ia memiliki kompleksitas komputasi yang tinggi, terutama untuk data set besar. Hal ini disebabkan oleh kebutuhan untuk melakukan eigen decomposition, yang bisa menjadi proses yang mahal secara komputasi. Kedua, penentuan jumlah kluster yang tepat bisa menjadi tantangan, seperti halnya dengan banyak algoritma pengelompokan lainnya.

6.3.4. Aplikasi Spectral Clustering

Dikarenakan kemampuannya untuk menemukan struktur yang kompleks dalam data, spectral clustering banyak digunakan dalam berbagai bidang dan aplikasi. Beberapa contohnya termasuk pengenalan pola, pengolahan citra, analisis jaringan sosial, dan bioinformatika. Di sini, kemampuan spectral clustering untuk menemukan kluster berdasarkan 'kemiripan' antara titik data, daripada berdasarkan jarak spasial, menjadi sangat berharga.

Secara keseluruhan, algoritma spectral clustering adalah alat yang kuat dalam kit alat pengelompokan data. Meskipun ia mungkin tidak selalu menjadi pilihan terbaik untuk setiap tugas pengelompokan - terutama bagi mereka yang melibatkan data set besar atau di mana jumlah kluster sudah diketahui - ia dapat menawarkan wawasan yang berharga dalam situasi di mana struktur atau pola dalam data adalah hal yang penting.

6.4. Implementasi Spectral Clustering dalam Python

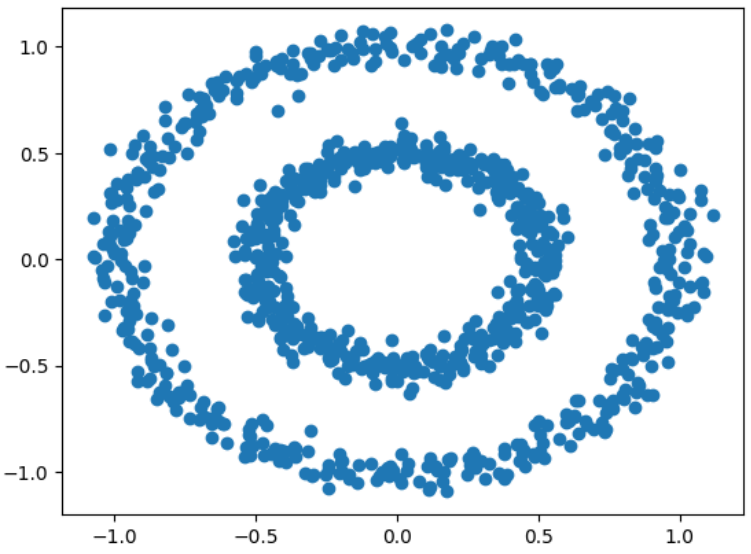
Setelah memahami konsep dasar dan algoritma spectral clustering, sekarang kita akan mencoba mengimplementasikan spectral clustering menggunakan Python dan library Scikit-learn, salah satu library yang paling sering digunakan untuk machine learning di Python. Dalam contoh ini, kita akan menggunakan data sintetis yang berjumlah banyak.

6.4.1. Membuat Data Sintetis

Pertama-tama, kita perlu membuat data sintetis yang akan digunakan. Kita akan menggunakan fungsi make_circles dari Scikit-learn untuk membuat data dengan dua kluster yang membentuk lingkaran. Data ini dihasilkan dalam bentuk array NumPy:

```
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
import numpy as np

# Membuat data sintetis
X, y = make_circles(n_samples=1000, noise=0.05, factor=0.5)
plt.scatter(X[:, 0], X[:, 1])
plt.show()
```

Output:

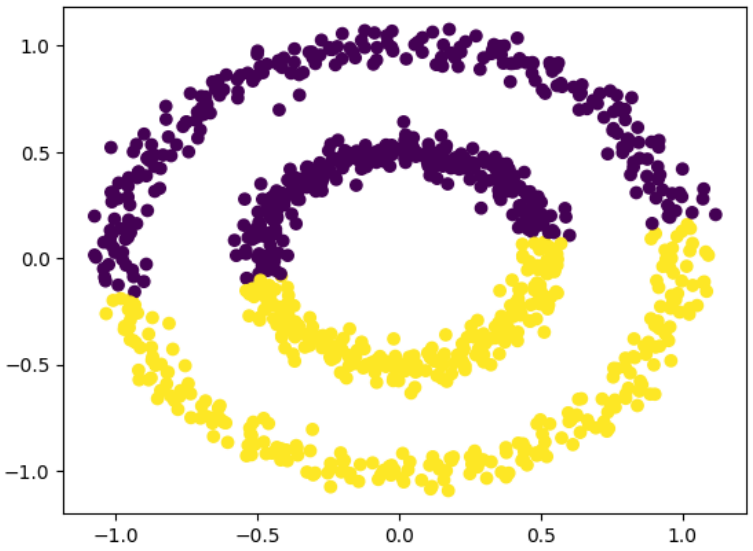


6.4.2. Implementasi Spectral Clustering

Setelah data dibuat, kita bisa mulai menerapkan spectral clustering. Scikit-learn menyediakan kelas SpectralClustering untuk ini. Sebelum menerapkan spectral clustering, kita perlu menentukan beberapa hyperparameter, seperti jumlah kluster (dalam hal ini 2) dan metode afinitas (dalam hal ini 'rbf' atau radial basis function):

```
from sklearn.cluster import SpectralClustering

# Menerapkan Spectral Clustering
sc = SpectralClustering(n_clusters=2, affinity='rbf')
y_pred = sc.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

Output:

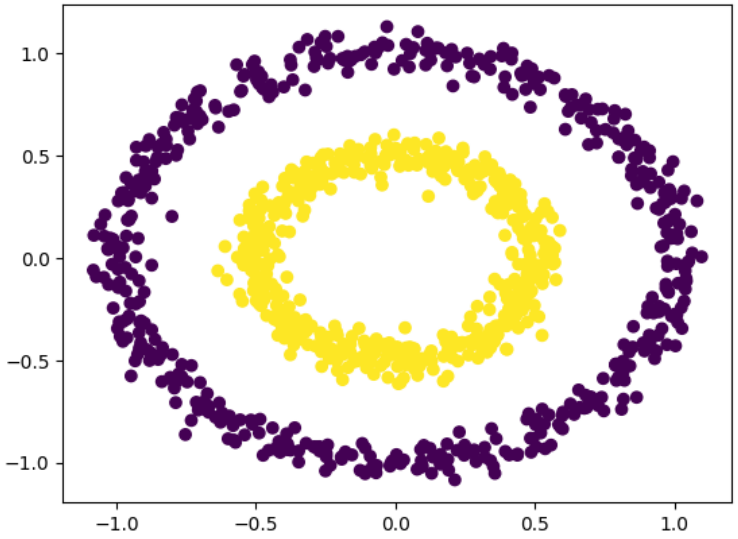


Bisa dilihat, hasil prediksi cluster dari algoritma spectral masih terlihat tidak sesuai dengan intuisi kita dimana seharusnya cluster terbagi menjadi cluster lingkaran dalam dan cluster lingkaran luar. Kira-kira algoritma apa yang cocok untuk bentuk data seperti diatas? atau parameter apa yang harus diubah? Mari kita coba ubah parameter affinity menjadi ‘nearest_neighbors’.

```
from sklearn.cluster import SpectralClustering

# Menerapkan Spectral Clustering
sc = SpectralClustering(n_clusters=2, affinity='nearest_neighbors')
y_pred = sc.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_pred)
```

```
plt.show()
```

Output:



Dengan mengubah parameter affinity menjadi nearest_neighbors, prediksi cluster menjadi lebih masuk akal. Lalu apa saja jenis parameter affinity dalam algoritma Spectral Clustering, dan kapan harus menggunakannya?

Pada algoritma Spectral Clustering dalam scikit-learn, terdapat beberapa parameter yang berkaitan dengan parameter affinity, yang menentukan metode pengukuran kesamaan yang digunakan untuk membangun matriks afinitas. Berikut ini adalah beberapa nilai umum untuk parameter affinity dalam SpectralClustering:

1. "nearest_neighbors": Metode ini menggunakan graf tetangga terdekat (k-nearest neighbors) sebagai matriks afinitas. Jumlah tetangga ditentukan oleh parameter n_neighbors. Metode ini cocok untuk kasus di mana titik-titik data memiliki konektivitas lokal.
2. "rbf": Metode ini menggunakan fungsi basis radial (RBF) untuk menghitung kesamaan antara pasangan titik data. Lebar kernel RBF dikendalikan oleh parameter gamma. Metode ini umumnya digunakan ketika tidak ada konsep tetangga yang jelas dan titik-titik data diharapkan memiliki kepadatan yang halus.
3. "precomputed": Metode ini memungkinkan Anda untuk langsung memberikan matriks afinitas yang telah dihitung sebelumnya sebagai masukan. Dalam hal ini, Anda perlu memasukkan matriks afinitas ke dalam metode fit() dari SpectralClustering.
4. "precomputed_nearest_neighbors": Metode ini menggunakan graf tetangga terdekat yang telah dihitung sebelumnya sebagai matriks afinitas. Anda perlu memasukkan graf tetangga terdekat ke dalam metode fit() dari SpectralClustering.

Pemilihan parameter affinity tergantung pada karakteristik data kamu dan masalah khusus yang ingin kamu selesaikan. Sebaiknya kamu mencoba berbagai metode afinitas dan mengevaluasi dampaknya terhadap hasil clustering.

6.4.3. Menyimpan dan Memuat Model

Setelah mendapatkan model yang diinginkan, kita bisa menyimpannya untuk digunakan di masa mendatang. Scikit-learn menyediakan fungsi dump dan load dari modul joblib untuk menyimpan dan memuat model:

```
from joblib import dump, load

# Menyimpan model
dump(sc, 'model.joblib')

# Memuat model
sc = load('model.joblib')
```

Dengan ini, kamu telah berhasil menerapkan spectral clustering pada data sintetis dan mengevaluasi hasilnya. Kamu juga telah belajar cara menyimpan dan memuat model. Perlu diingat bahwa spectral clustering adalah metode non-parametrik yang tidak membuat asumsi tentang bentuk kluster, sehingga bisa sangat berguna untuk data yang tidak bisa di-cluster dengan baik oleh metode lain seperti k-means. Namun, spectral clustering juga memiliki kekurangan, seperti sensitivitas terhadap pilihan hyperparameter dan kurang efisien untuk data berskala besar. Oleh karena itu, penting untuk memahami karakteristik data dan kebutuhan proyek sebelum memilih metode clustering.

Bab 7. Ward Hierarchical Clustering

7.1 Pengertian Ward Hierarchical Clustering

Hierarchical clustering adalah teknik clustering yang menciptakan hirarki atau struktur berjenjang dari cluster. Ada dua jenis utama dari hierarchical clustering: agglomerative (bottom-up) dan divisive (top-down). Agglomerative clustering dimulai dengan setiap titik data sebagai cluster sendiri dan kemudian menggabungkan cluster berdasarkan kedekatan mereka. Sebaliknya, divisive clustering dimulai dengan semua titik data dalam satu cluster dan kemudian membaginya menjadi cluster yang lebih kecil berdasarkan kedekatan.

Ward hierarchical clustering adalah jenis agglomerative clustering yang menggunakan metode minimum variance (varians minimum) yang dikenal sebagai metode Ward. Metode ini ditemukan oleh Joe H. Ward Jr., seorang psikolog matematika, dan oleh karena itu dikenal sebagai metode Ward atau Ward's method. Metode ini bertujuan untuk meminimalkan varians dalam setiap cluster. Dalam konteks ini, varians diukur sebagai jumlah kuadrat jarak antara titik data dan pusat cluster (rata-rata titik data dalam cluster).

Berbeda dengan metode agglomerative clustering lainnya seperti single linkage, complete linkage, atau average linkage, metode Ward mempertimbangkan keseluruhan struktur data dalam membuat keputusan tentang penggabungan cluster. Metode ini menghasilkan cluster yang lebih seimbang, dalam arti bahwa cluster memiliki jumlah titik data yang lebih merata dan bentuk yang lebih bulat dibandingkan dengan metode lain.

Satu aspek penting dari Ward hierarchical clustering adalah bahwa hasilnya dapat ditampilkan dalam bentuk dendrogram, yaitu diagram pohon yang menunjukkan bagaimana cluster dibentuk. Dalam dendrogram, setiap titik data direpresentasikan sebagai cabang di bagian bawah diagram, dan setiap penggabungan cluster direpresentasikan sebagai cabang yang bergabung. Tinggi cabang yang bergabung menunjukkan jarak antara cluster yang digabungkan, yang dalam hal ini adalah peningkatan total varians yang dihasilkan oleh penggabungan.

Berikut ini adalah beberapa poin penting lainnya tentang Ward hierarchical clustering:

Metode Ward efektif untuk data yang berdistribusi normal. Jika data tidak berdistribusi normal, metode lain mungkin lebih cocok.

Metode Ward menghasilkan cluster yang seimbang dan berbentuk bulat. Jika struktur asli data tidak seperti ini, metode lain mungkin lebih cocok.

Metode Ward memerlukan perhitungan jarak antara titik data, jadi mungkin tidak efisien untuk data berskala besar.

Sebagai metode non-parametrik, Ward hierarchical clustering tidak membuat asumsi tentang bentuk kluster.

Metode Ward dapat digunakan untuk analisis eksplorasi awal data, terutama dengan bantuan dendrogram.

Secara keseluruhan, Ward hierarchical clustering adalah alat yang sangat berharga dalam toolkit machine learning. Namun, seperti semua metode, penting untuk memahami kekuatan dan kelemahannya dan bagaimana mereka berhubungan dengan karakteristik data dan tujuan analisis. Dalam subbab berikutnya, kita akan mempelajari lebih lanjut tentang bagaimana algoritma Ward bekerja dan bagaimana mengimplementasikannya dalam Python.

7.2 Algoritma Ward

Algoritma Ward untuk hierarchical clustering merupakan algoritma yang cukup sederhana untuk diikuti, meski mungkin sedikit membingungkan pada awalnya. Ingatlah bahwa tujuan utamanya adalah meminimalkan peningkatan total dalam varians antar grup setelah penggabungan dua cluster.

Berikut ini adalah langkah-langkah dasar dalam algoritma Ward:

1. Inisialisasi: Setiap titik data dianggap sebagai cluster sendiri. Ini berarti jika kamu memiliki N titik data, kamu akan memulai dengan N cluster.
2. Hitung matriks jarak: Hitung jarak antara semua pasangan cluster. Di awal, ini akan menjadi jarak antara titik data individu. Jarak antara dua cluster biasanya diukur sebagai peningkatan total dalam varians yang akan terjadi jika dua cluster tersebut digabungkan. Varians dihitung sebagai jumlah kuadrat jarak antara titik data dan pusat cluster.
3. Penggabungan cluster: Temukan dua cluster yang penggabungannya akan menghasilkan peningkatan minimum dalam total varians. Gabungkan dua cluster tersebut menjadi satu. Sekarang, jumlah total cluster menjadi N-1.
4. Perbarui matriks jarak: Setelah penggabungan cluster, perbarui matriks jarak untuk mencerminkan jarak antara cluster baru dan semua cluster lainnya.
5. Ulangi langkah 3 dan 4: Ulangi langkah 3 dan 4 sampai semua titik data berada dalam satu cluster. Hasilnya adalah dendrogram yang menunjukkan urutan penggabungan dan peningkatan dalam total varians pada setiap penggabungan.
6. Langkah-langkah ini cukup jelas, tetapi ada beberapa detail yang mungkin perlu kamu perhatikan. Salah satunya adalah cara menghitung peningkatan dalam total varians. Ada beberapa cara yang berbeda untuk melakukan ini, tetapi metode Ward biasanya menggunakan formula berikut:

$$\Delta(T1 \cup T2) = sum((n\_k/n\_3) * (c\_k - c\_3)\text{^}2)$$

Dimana:

- T1 dan T2 adalah dua cluster yang sedang dipertimbangkan untuk digabungkan,
- n_k adalah jumlah titik data dalam cluster k,
- c_k adalah pusat cluster k (rata-rata titik data dalam cluster),
- c_3 adalah pusat cluster yang akan dihasilkan jika T1 dan T2 digabungkan.

Formula ini menghitung peningkatan dalam total varians sebagai jumlah kuadrat jarak antara pusat cluster dan pusat cluster gabungan, dikalikan dengan fraksi jumlah titik data dalam cluster. Ini menghasilkan ukuran peningkatan varians yang berimbang dan menghasilkan cluster yang seimbang dan berbentuk bulat.

Algoritma Ward cukup efisien dan dapat menghasilkan hasil yang baik dalam banyak kasus. Namun, seperti semua algoritma clustering, itu tidak sempurna dan memiliki beberapa batasan. Misalnya, itu mungkin tidak efisien untuk data berskala besar, dan itu mungkin tidak menghasilkan hasil yang baik untuk data yang tidak berdistribusi normal atau yang memiliki cluster dengan bentuk yang tidak biasa.

Jadi, itulah gambaran umum tentang algoritma Ward. Di subbab berikutnya, kita akan membahas tentang dendrogram dan bagaimana kita bisa menentukan jumlah optimal cluster dari dendrogram tersebut.

Namun sebelum kita beranjak, penting untuk kamu mengingat bahwa meski algoritma Ward cukup baik dalam menangani sejumlah besar data dan menghasilkan cluster yang cukup berimbang, metode ini tetap memiliki keterbatasannya. Seperti yang disebutkan sebelumnya, algoritma Ward mungkin tidak bekerja dengan baik untuk data yang tidak berdistribusi normal atau memiliki cluster dengan bentuk yang tidak biasa.

Selain itu, perlu diingat bahwa penggunaan metode Ward tidak selalu menghasilkan solusi optimal. Ada banyak faktor lain yang perlu dipertimbangkan saat melakukan clustering, termasuk jenis data, jumlah data, dan asumsi yang dibuat tentang struktur data. Oleh karena itu, penting untuk selalu melakukan eksplorasi data awal dan menguji berbagai metode clustering sebelum memilih metode yang paling tepat untuk dataset spesifik kamu.

Mungkin kamu bertanya, bagaimana jika data kamu memiliki jutaan titik data? Apakah algoritma Ward masih efektif? Jawabannya adalah, meskipun algoritma Ward bisa digunakan, kamu mungkin akan menemui masalah skalabilitas karena kompleksitas waktu algoritma ini adalah O(n^3), di mana n adalah jumlah titik data. Untuk data berskala besar, kamu mungkin ingin menggunakan algoritma lain atau teknik reduksi dimensi seperti PCA sebelum melakukan clustering.

Terakhir, penting untuk diingat bahwa hasil dari algoritma Ward - dan algoritma clustering lainnya - harus selalu divalidasi dengan pengetahuan domain dan interpretasi yang bijaksana. Clustering adalah tugas yang tidak terawasi, yang berarti tidak ada "jawaban" yang benar yang bisa kita bandingkan. Oleh karena itu, kita harus selalu siap untuk

mempertanyakan hasil dan melakukan analisis lebih lanjut jika diperlukan.

7.3. Dendrogram dan Penentuan Jumlah Cluster

Pada bagian ini, kita akan membahas tentang dendrogram. Dendrogram adalah alat visual yang sangat berguna saat melakukan clustering hierarkis, dan dapat memberikan banyak wawasan tentang bagaimana data kamu terstruktur.

Jadi, apa itu dendrogram? Dendrogram adalah diagram pohon yang menggambarkan pengelompokan yang dilakukan oleh metode clustering hierarkis, termasuk metode Ward. Setiap titik data diwakili oleh sebuah leaf (daun), dan cluster dibentuk dengan menggabungkan leaf atau cabang (branch) yang sudah ada berdasarkan tingkat kemiripan mereka. Hasilnya adalah struktur pohon yang menunjukkan bagaimana data kamu bisa dikelompokkan pada berbagai tingkat kemiripan.

Dalam dendrogram, jarak antara leaf atau cabang merefleksikan tingkat kemiripan antara titik data atau cluster. Dua titik data yang sangat mirip akan digabungkan lebih awal (di bagian bawah dendrogram), sedangkan titik data atau cluster yang kurang mirip akan digabungkan kemudian (di bagian atas dendrogram).

Dendrogram memberikan cara visual dan intuitif untuk memahami bagaimana data kamu terstruktur. Dengan melihat dendrogram, kamu bisa melihat berapa banyak cluster yang mungkin ada, bagaimana titik data dikelompokkan, dan sejauh mana mereka mirip satu sama lain.

Dendrogram juga bisa membantu kamu memilih jumlah cluster yang optimal. Secara umum, jumlah cluster optimal adalah jumlah cabang besar pada dendrogram. Kamu bisa menentukan ini dengan melihat di mana kamu bisa "memotong" dendrogram untuk mendapatkan jumlah cabang yang paling masuk akal. Namun, perlu diingat bahwa ini adalah panduan umum dan mungkin tidak selalu berlaku. Jumlah cluster optimal tergantung pada data dan konteks spesifik kamu, dan mungkin perlu ditentukan dengan eksperimen dan pengetahuan domain.

Penting juga untuk mengetahui bahwa dendrogram bisa menjadi rumit dan sulit diinterpretasikan untuk data berskala besar. Untuk data dengan banyak titik, dendrogram bisa menjadi sangat padat dan sulit dibaca. Oleh karena itu, dendrogram mungkin paling berguna untuk data berskala kecil hingga menengah.

Namun, meski demikian, dendrogram tetap menjadi alat yang sangat berharga dalam clustering hierarkis. Dengan memahami bagaimana dendrogram bekerja dan bagaimana menginterpretasikannya, kamu bisa mendapatkan wawasan yang lebih dalam tentang struktur data kamu dan membuat keputusan yang lebih baik tentang bagaimana melakukan clustering.

7.4. Implementasi Ward Hierarchical Clustering dalam Python

Setelah mempelajari konsep dasar Ward Hierarchical Clustering, sekarang saatnya untuk menerapkannya dalam kode Python. Kita akan menggunakan data sintetis untuk memahami bagaimana Ward Hierarchical Clustering bekerja dalam praktek.

7.4.1. Membuat Data Sintetis

Sebelum melakukan clustering, tentu saja kita perlu memiliki data. Untuk tujuan tutorial ini, kita akan menggunakan fungsi make_blobs dari library sklearn.datasets untuk membuat data sintetis.

```
from sklearn.datasets import make_blobs

# Membuat data sintetis
X, y = make_blobs(n_samples=500, centers=5, random_state=42)

# Menampilkan ukuran data
print(f'Ukuran data: {X.shape}')
```

Output:

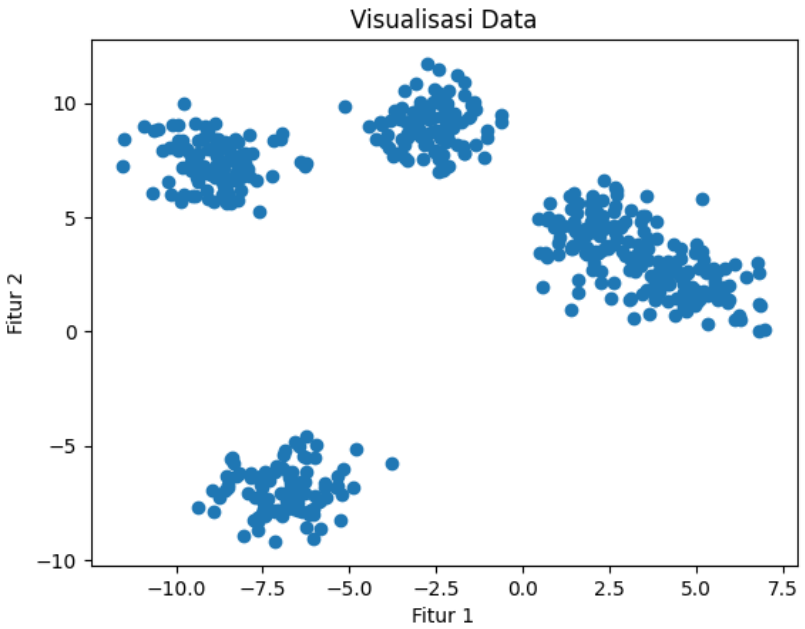
```
Ukuran data: (500, 2)
```

7.4.2. Visualisasi Data

Langkah ini opsional, tapi visualisasi data awal dapat membantu kamu memahami struktur data yang kamu miliki.

```
import matplotlib.pyplot as plt

# Visualisasi data
plt.scatter(X[:,0], X[:,1])
plt.title('Visualisasi Data')
plt.xlabel('Fitur 1')
plt.ylabel('Fitur 2')
plt.show()
```

Output:



7.4.3. Menerapkan Ward Hierarchical Clustering

Kita akan menggunakan fungsi AgglomerativeClustering dari sklearn.cluster untuk melakukan Ward Hierarchical Clustering. Parameter n_clusters digunakan untuk menentukan jumlah cluster yang diinginkan, dan linkage='ward' untuk menggunakan metode Ward.

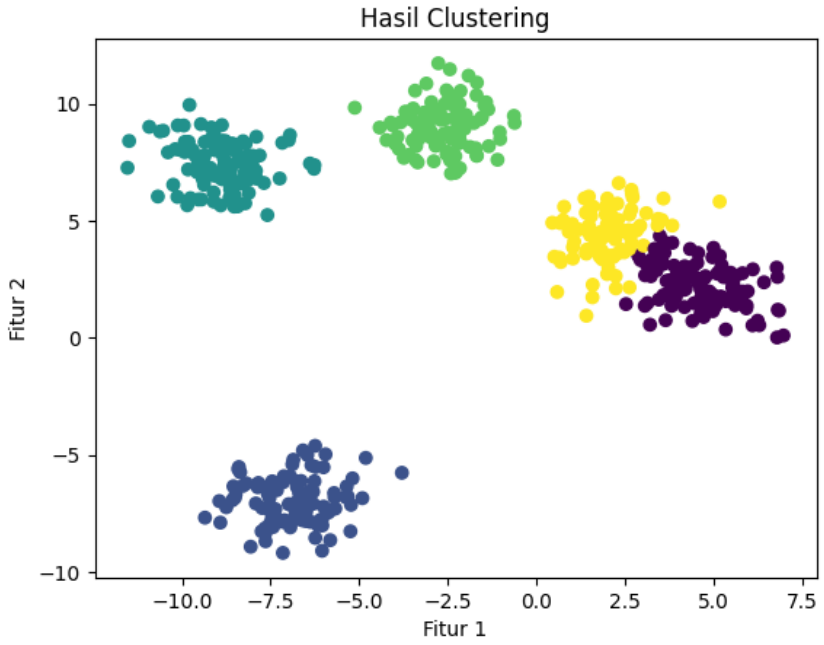
```
from sklearn.cluster import AgglomerativeClustering

# Menerapkan Ward hierarchical clustering
model = AgglomerativeClustering(n_clusters=5, linkage='ward')
model.fit(X)
```

7.4.4. Visualisasi Hasil Clustering

Setelah melakukan clustering, kita bisa visualisasi hasilnya.

```
# Visualisasi hasil clustering
plt.scatter(X[:,0], X[:,1], c=model.labels_)
plt.title('Hasil Clustering')
plt.xlabel('Fitur 1')
plt.ylabel('Fitur 2')
plt.show()
```

Output:

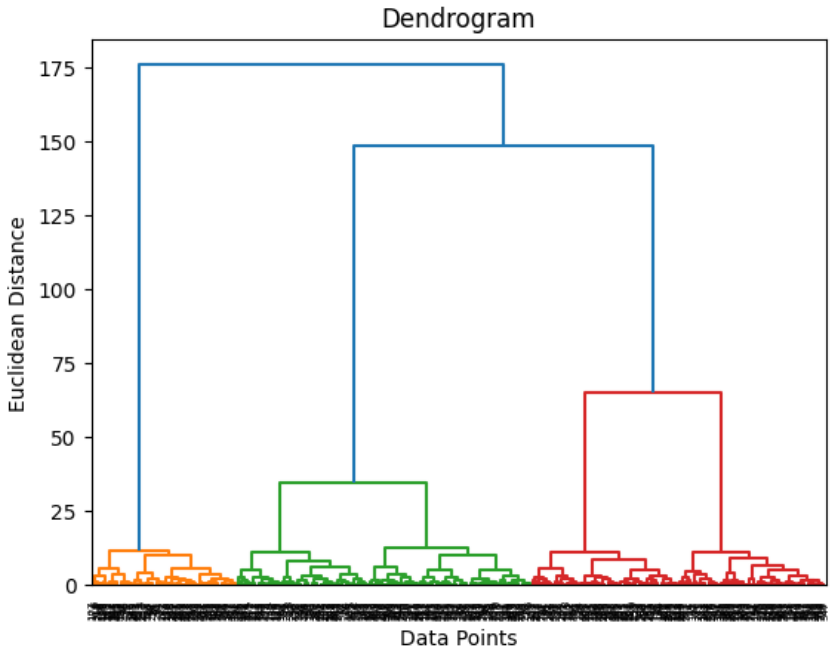


7.4.5. Membuat dan Menganalisis Dendrogram

Kita akan menggunakan fungsi dendrogram dan linkage dari scipy.cluster.hierarchy untuk membuat dan menampilkan dendrogram. Dendrogram ini bisa membantu menentukan jumlah cluster yang optimal.

```
from scipy.cluster.hierarchy import dendrogram, linkage

# Membuat matriks linkage
Z = linkage(X, method='ward')

# Membuat dendrogram
dendrogram(Z)
plt.title('Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()
```

Output:



7.4.6. Hyperparameter Tuning

Untuk tuning hyperparameter, kamu bisa mencoba mengubah nilai n_clusters dan melihat bagaimana hasilnya berubah. Kamu juga bisa mencoba metode linkage lain seperti 'complete' atau 'average', dan melihat bagaimana hasilnya dibandingkan dengan metode 'ward'.

Jangan lupa bahwa dalam clustering, tidak ada "jawaban" yang benar dan hasil terbaik sangat tergantung pada konteks dan tujuan kamu. Oleh karena itu, penting untuk selalu memeriksa hasil clustering dan memastikan bahwa mereka masuk akal untuk aplikasi kamu.

Bab 8. Agglomerative Clustering

8.1. Pengertian Agglomerative Clustering

Agglomerative Clustering adalah salah satu teknik dalam hierarchical clustering yang menggunakan pendekatan bottom-up. Di sinilah perjalananmu mengenal Agglomerative Clustering dimulai. Bukan hanya berhenti pada pengertian umum, kita juga akan membahas tentang bagaimana teknik ini bekerja, apa kelebihan dan kekurangannya, serta di mana dan kapan kamu sebaiknya menggunakan teknik ini.

Sebagai awal, mari kita pahami apa itu agglomerative clustering. Kata 'agglomerative' sendiri berasal dari kata 'agglomerate' yang berarti 'mengumpulkan' atau 'menggabungkan'. Sesuai dengan namanya, agglomerative clustering memulai prosesnya dengan menganggap setiap titik data sebagai sebuah cluster individu. Kemudian, secara bertahap dan iteratif, cluster-cluster ini digabungkan bersama berdasarkan ukuran kedekatan atau kemiripan antar titik data atau antar cluster.

Berbeda dengan metode clustering partitional seperti K-Means yang membutuhkan jumlah cluster yang sudah ditentukan sejak awal, agglomerative clustering tidak membutuhkan jumlah cluster yang spesifik di awal. Sebaliknya, kita dapat menentukan jumlah cluster yang optimal berdasarkan visualisasi dendrogram yang dihasilkan selama proses clustering.

Teknik ini dapat digunakan dalam berbagai kasus dalam dunia nyata. Misalnya, dalam analisis genetika untuk menemukan kemiripan antara spesies berbeda berdasarkan gen mereka, dalam sistem rekomendasi untuk mengelompokkan produk atau layanan yang serupa, atau dalam analisis media sosial untuk mengidentifikasi komunitas atau grup berdasarkan interaksi pengguna.

Walau memiliki banyak kegunaan, agglomerative clustering juga memiliki beberapa kelemahan. Salah satunya adalah bahwa teknik ini cukup sensitif terhadap outlier. Outlier yang ada dalam data dapat mengganggu proses penggabungan cluster dan menghasilkan cluster yang tidak optimal. Selain itu, agglomerative clustering juga memiliki kompleksitas komputasi yang tinggi, terutama jika jumlah titik data sangat banyak, hal ini membuat metode ini kurang cocok untuk dataset berukuran besar.

Namun demikian, agglomerative clustering tetap menjadi salah satu teknik clustering yang populer dan sering digunakan dalam berbagai bidang. Hal ini karena kelebihan-kelebihannya, seperti kemampuannya untuk menemukan struktur yang kompleks dan fleksibilitas dalam menentukan jumlah cluster, masih menjadikannya pilihan yang menarik.

8.2. Algoritma Agglomerative Clustering

Setelah memahami apa itu Agglomerative Clustering, sekarang saatnya kamu memahami bagaimana algoritma ini bekerja. Agglomerative Clustering memiliki serangkaian langkah yang cukup sistematis dan mudah diikuti, yang akan membantu kamu memahami bagaimana cara kerjanya dalam mengelompokkan data.

Berikut ini adalah langkah-langkah umum dalam algoritma Agglomerative Clustering:

1. Inisialisasi: Pada tahap awal ini, setiap titik data dianggap sebagai cluster tersendiri. Jadi, jika kita memiliki N titik data, maka kita memiliki N cluster.
2. Penghitungan matriks jarak: Tahap selanjutnya adalah menghitung jarak antara setiap pasangan cluster. Jarak ini bisa dihitung dengan berbagai cara, tergantung metode linkage yang kamu pilih (kita akan membahasnya nanti). Matriks jarak ini akan berisi jarak antara setiap pasangan cluster. Pada awalnya, karena setiap titik data dianggap sebagai cluster, matriks jarak ini akan berukuran N x N.
3. Penggabungan cluster: Di sini, dua cluster yang memiliki jarak terdekat akan digabungkan menjadi satu cluster. Setelah penggabungan, jumlah total cluster akan berkurang satu.
4. Perbaruan matriks jarak: Setelah penggabungan, matriks jarak perlu diperbarui. Jarak antara cluster baru dengan cluster lainnya perlu dihitung dan matriks jarak lama perlu diperbarui dengan jarak-jarak baru ini.
5. Iterasi: Langkah 3 dan 4 akan diulang hingga semua titik data digabungkan menjadi satu cluster.

Sebagai contoh, bayangkan kamu memiliki 5 titik data: A, B, C, D, dan E. Pada awalnya, setiap titik data ini akan dianggap sebagai cluster tersendiri. Kemudian, kamu akan menghitung jarak antara setiap pasangan cluster dan mencatatnya dalam matriks jarak. Misalkan, jarak terdekat adalah antara cluster A dan B, maka dua cluster ini akan digabungkan menjadi satu cluster, misalnya kita namakan cluster baru ini sebagai AB. Setelah itu, kamu perlu menghitung jarak antara cluster AB ini dengan cluster lainnya dan memperbarui matriks jarak. Proses ini diulang hingga semua titik data digabungkan menjadi satu cluster.

Hal yang perlu diperhatikan di sini adalah bagaimana kamu menghitung jarak antara dua cluster. Ada beberapa metode yang bisa kamu pilih, yang biasa disebut sebagai metode linkage. Metode ini akan kita bahas di subbab berikutnya.

Algoritma Agglomerative Clustering ini mungkin terdengar cukup kompleks pada awalnya, namun sebenarnya cukup sederhana jika kamu mengikutinya langkah demi langkah. Yang penting di sini adalah pemahamanmu tentang konsep dasarnya, yaitu bagaimana setiap titik data dianggap sebagai cluster dan bagaimana dua cluster dengan jarak terdekat digabungkan secara iteratif.

8.3. Linkage Methods

Saat kamu telah memahami bagaimana algoritma Agglomerative Clustering bekerja, kamu mungkin bertanya, "Bagaimana kita menghitung jarak antara dua cluster?" Nah, inilah saatnya kita membahas tentang metode linkage. Metode linkage adalah cara kita menghitung jarak antara dua cluster. Ada beberapa metode linkage yang umum digunakan dalam Agglomerative Clustering, yaitu: Single Linkage, Complete Linkage, Average Linkage, dan Ward's Method.

8.3.1. Single Linkage (Metode Nearest Neighbor)

Single linkage, juga dikenal sebagai metode nearest neighbor, menghitung jarak antara dua cluster berdasarkan jarak antara titik data terdekat dalam dua cluster tersebut. Dengan kata lain, jarak antara dua cluster adalah jarak terpendek antara setiap titik dalam satu cluster ke titik mana pun dalam cluster lainnya.

Single linkage bisa sangat sensitif terhadap outlier. Artinya, jika ada satu titik data yang sangat jauh dari titik data lainnya dalam cluster yang sama, maka titik data tersebut bisa sangat mempengaruhi jarak antara dua cluster.

8.3.2. Complete Linkage (Metode Farthest Neighbor)

Complete linkage, atau metode farthest neighbor, adalah kebalikan dari single linkage. Jarak antara dua cluster dihitung berdasarkan jarak antara titik data terjauh dalam dua cluster tersebut. Dengan kata lain, jarak antara dua cluster adalah jarak terpanjang antara setiap titik dalam satu cluster ke titik mana pun dalam cluster lainnya.

Complete linkage cenderung lebih tahan terhadap outlier dibandingkan single linkage. Namun, metode ini bisa menghasilkan cluster yang tidak berbentuk bulat atau elips.

8.3.3. Average Linkage

Average linkage menghitung jarak antara dua cluster berdasarkan rata-rata jarak antara setiap pasangan titik data di dua cluster tersebut. Dengan kata lain, kita menghitung jarak antara setiap titik dalam satu cluster ke setiap titik dalam cluster lainnya, lalu kita rata-rata jarak-jarak tersebut.

Average linkage cenderung memberikan hasil yang lebih seimbang dibandingkan single dan complete linkage. Metode ini tidak terlalu sensitif terhadap outlier dan bisa menghasilkan cluster yang berbentuk bulat atau elips.

8.3.4. Ward's Method

Ward's method berusaha meminimalkan variansi total dalam setiap cluster. Jarak antara dua cluster dihitung berdasarkan peningkatan jumlah kuadrat jarak dari setiap titik data ke pusat cluster jika dua cluster tersebut digabungkan.

Ward's method cenderung menghasilkan cluster yang berukuran serupa dan berbentuk bulat atau elips. Metode ini sangat berguna jika kamu menginginkan cluster yang seimbang dan kompak.

Dalam menentukan metode linkage mana yang akan digunakan, kamu perlu mempertimbangkan struktur data kamu dan apa yang ingin kamu capai dengan clustering. Setiap metode memiliki kelebihan dan kekurangannya masing-masing, sehingga tidak ada satu metode yang selalu terbaik untuk semua kasus. Cobalah beberapa metode dan lihat mana yang memberikan hasil terbaik untuk data kamu.

8.4. Implementasi Agglomerative Clustering dalam Python

Agglomerative Clustering adalah metode yang intuitif dan mudah diimplementasikan dalam python. Di subbab ini, kita akan mempraktikkan implementasi Agglomerative Clustering menggunakan library sklearn dan scipy. Kami akan membahas cara membuat data sintetis, melakukan clustering, dan visualisasi hasil. Selanjutnya, kita juga akan mencakup proses hyperparameter tuning untuk meningkatkan performa model.

Mari kita mulai dengan langkah pertama, yaitu membuat data sintetis.

8.4.1. Membuat Data Sintetis

Kita akan menggunakan fungsi make_blobs dari modul sklearn.datasets untuk membuat data sintetis. Fungsi ini akan menghasilkan kluster data Gaussian dengan jumlah kluster yang dapat ditentukan.

```
from sklearn.datasets import make_blobs

# Membuat data sintetis
X, y_true = make_blobs(n_samples=500, centers=4, cluster_std=0.8,
random_state=0)
```

Setelah kita memiliki data sintetis, mari kita lanjutkan ke langkah berikutnya, yaitu melakukan clustering.

8.4.2. Melakukan Clustering

Agglomerative Clustering dapat dengan mudah dilakukan menggunakan fungsi AgglomerativeClustering dari modul sklearn.cluster. Pada tahap ini, kita akan menggunakan semua metode linkage yang telah kita bahas sebelumnya, yaitu: ward, complete, average, dan single.

```
from sklearn.cluster import AgglomerativeClustering

# Membuat model agglomerative clustering
```

```
model = AgglomerativeClustering(n_clusters=4, affinity='euclidean',
linkage='ward')

# Melakukan fitting model ke data
y_pred = model.fit_predict(X)
```

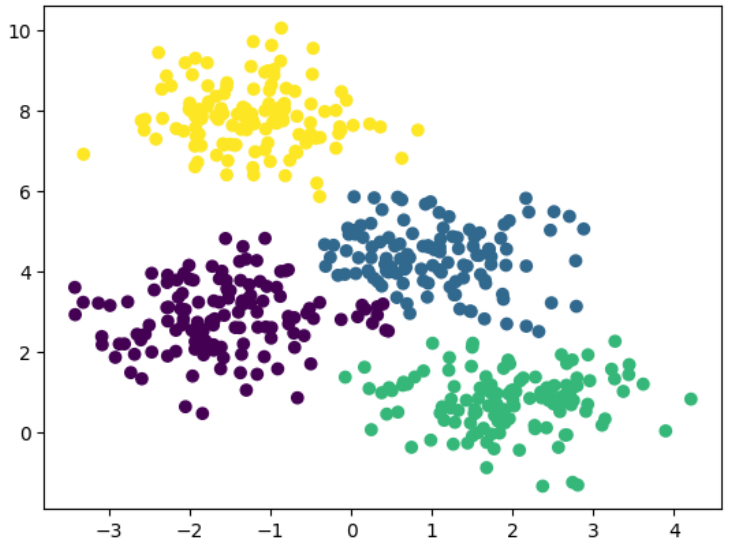
Setelah melakukan clustering, kita perlu memvisualisasikan hasilnya.

8.4.3. Visualisasi Hasil Clustering

Untuk memvisualisasikan hasil clustering, kita akan menggunakan matplotlib, sebuah library python yang populer untuk visualisasi data.

```
import matplotlib.pyplot as plt

# Membuat plot
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')

# Menampilkan plot
plt.show()
```

Output:



Dalam plot di atas, setiap warna mewakili satu kluster.

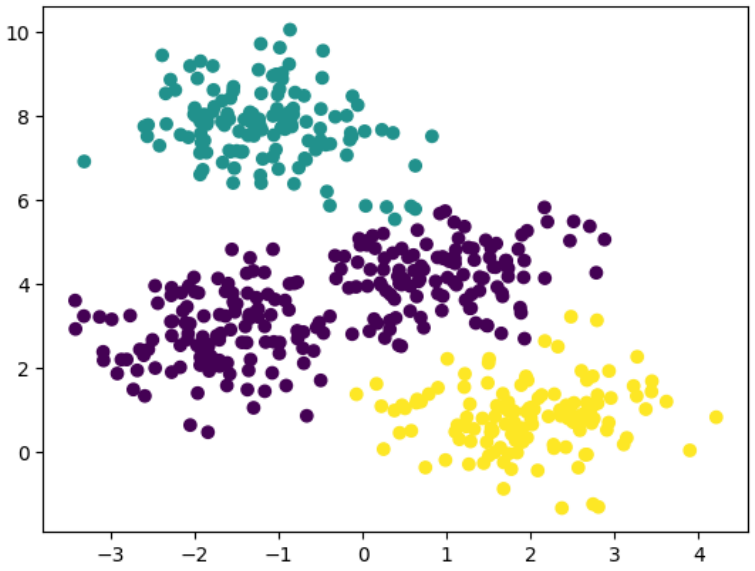
8.4.4. Hyperparameter Tuning

Hyperparameter tuning adalah proses menyesuaikan hyperparameter model untuk meningkatkan performa model. Dalam Agglomerative Clustering, ada beberapa hyperparameter yang bisa diubah-ubah, seperti jumlah kluster (n_clusters), metrik yang digunakan untuk menghitung jarak antara sampel (affinity), dan metode linkage (linkage).

Kamu bisa menggunakan Grid Search atau Random Search untuk mencari kombinasi hyperparameter yang optimal. Namun, dalam tutorial ini, kita akan menyesuaikan jumlah kluster dan metode linkage secara manual untuk melihat bagaimana perubahan tersebut mempengaruhi hasil clustering.

```
# Menyesuaikan jumlah kluster dan metode linkage
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean',
linkage='complete')
y_pred = model.fit_predict(X)

# Visualisasi hasil
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.show()
```

Output:



Dengan mengubah jumlah kluster dan metode linkage, kita bisa melihat perubahan pada hasil clustering. Namun, perlu diingat bahwa hasil yang optimal tidak selalu mudah ditemukan dan mungkin memerlukan beberapa percobaan.

8.4.5. Menggunakan Dendrogram untuk Menentukan Jumlah Kluster

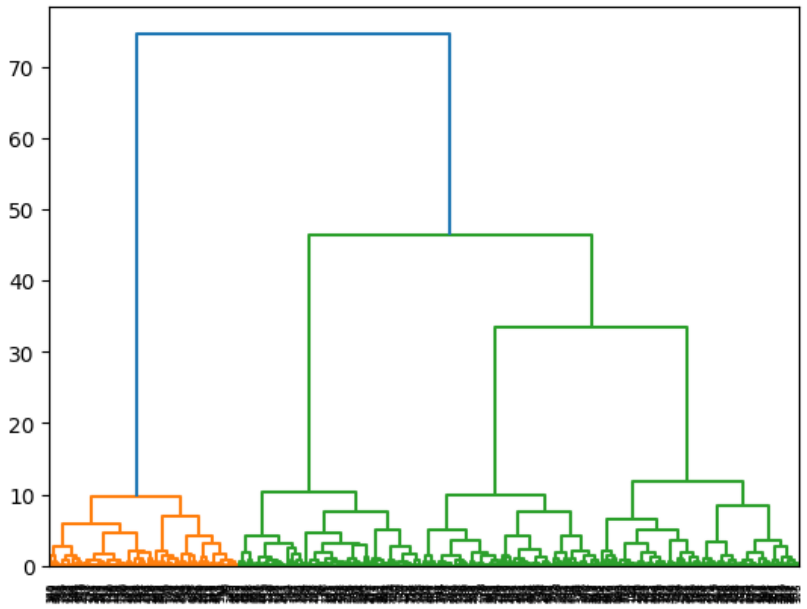
Dalam hierarchical clustering, dendrogram dapat digunakan untuk membantu menentukan jumlah kluster yang optimal. Dendrogram adalah diagram pohon yang menunjukkan bagaimana kluster dibentuk.

Kita dapat menggunakan library scipy untuk membuat dendrogram. Berikut adalah kode untuk membuat dendrogram dari model clustering kita:

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
# Membuat linkage matrix
linked = linkage(X, 'ward')

# Membuat dendrogram
dendrogram(linked)
plt.show()
```

Output:



Dalam dendrogram, setiap garis vertikal mewakili satu kluster, dan tinggi garis menunjukkan jarak antara kluster. Jadi, kamu bisa menggunakan dendrogram untuk menentukan jumlah kluster yang optimal dengan mencari "jumps" terbesar dalam jarak antara kluster berturut-turut.

8.4.6. Evaluasi Model Clustering

Bagaimana kita tahu bahwa model clustering kita bekerja dengan baik? Salah satu metode yang paling umum digunakan untuk mengevaluasi model clustering adalah Silhouette Coefficient.

Silhouette Coefficient mengukur sejauh mana setiap sampel berada dekat dengan sampel dalam kluster yang sama dibandingkan dengan sampel dalam kluster lain. Nilai Silhouette Coefficient berkisar antara -1 dan 1, di mana nilai yang lebih tinggi menunjukkan bahwa sampel lebih dekat dengan kluster mereka sendiri dan jauh dari kluster lain.

Kamu bisa menghitung Silhouette Coefficient dengan menggunakan fungsi silhouette_score dari modul sklearn.metrics.

```
from sklearn.metrics import silhouette_score
```

```
# Menghitung Silhouette Coefficient
score = silhouette_score(X, y_pred)

print("Silhouette Coefficient: ", score)
```

Output:

```
Silhouette Coefficient:  0.5215289263304937
```

Itulah cara kamu melakukan Agglomerative Clustering dalam Python. Seperti yang kamu lihat, prosesnya cukup sederhana dan mudah dipahami. Namun, penting untuk diingat bahwa Agglomerative Clustering, seperti algoritma clustering lainnya, mungkin tidak selalu bekerja dengan baik untuk setiap jenis data. Oleh karena itu, selalu penting untuk mencoba beberapa algoritma berbeda dan melihat mana yang paling cocok untuk data kamu.

Bab 9. DBSCAN

9.1. Pengertian DBSCAN

DBSCAN, yang merupakan singkatan dari Density-Based Spatial Clustering of Applications with Noise, adalah algoritma pengelompokan yang berbasis pada densitas. Algoritma ini dikembangkan oleh Martin Ester, Hans-Peter Kriegel, Jörg Sander, dan Xiaowei Xu pada tahun 1996 dan telah menjadi salah satu algoritma pengelompokan paling populer sejak saat itu. Kenapa? Karena DBSCAN memiliki beberapa keunggulan dibandingkan algoritma pengelompokan lainnya. Tetapi sebelum kita membahas lebih jauh, mari kita memahami apa itu pengelompokan dan mengapa itu penting.

Apa itu DBSCAN?

Seperti yang disebutkan sebelumnya, DBSCAN adalah algoritma pengelompokan berbasis densitas. Ini berarti bahwa DBSCAN mencari wilayah di ruang fitur di mana banyak titik data berada dekat satu sama lain (densitas tinggi), dan mengelompokkan titik-titik ini bersama-sama. Titik-titik data dalam wilayah dengan densitas rendah (yang biasanya ditemukan di antara kluster) dianggap sebagai noise atau titik-titik data anomali.

DBSCAN memiliki tiga jenis titik data

1. Core points: Titik yang memiliki setidaknya sejumlah minimum titik data (MinPts) dalam jangkauan tertentu (eps) dianggap sebagai titik inti.
2. Border points: Titik yang memiliki lebih sedikit titik data dari MinPts dalam jangkauan eps, tetapi berada dalam jangkauan eps dari titik inti, dianggap sebagai titik batas.
3. Noise points: Titik yang bukan titik inti atau titik batas dianggap sebagai noise atau titik data anomali.

DBSCAN memiliki beberapa keunggulan dibandingkan algoritma pengelompokan lainnya:

1. Tidak perlu menentukan jumlah kluster: Berbeda dengan algoritma pengelompokan seperti K-means, kita tidak perlu menentukan jumlah kluster sebelum menjalankan DBSCAN. Algoritma ini secara otomatis menentukan jumlah kluster berdasarkan struktur data dan parameter yang kita tentukan (eps dan MinPts).
2. Dapat mendeteksi kluster dengan bentuk dan ukuran apa pun: Algoritma pengelompokan berbasis partisi seperti K-means memiliki kesulitan untuk menemukan kluster yang bukan berbentuk hiperboloid atau kluster dengan variasi ukuran yang signifikan. DBSCAN, di sisi lain, tidak memiliki asumsi semacam ini, dan karena itu dapat menemukan kluster dengan bentuk dan ukuran apa pun.
3. Dapat menangani noise dan titik data anomali: Seperti yang telah disebutkan sebelumnya, DBSCAN membedakan antara titik inti, titik batas, dan titik noise, yang

memungkinkannya untuk menangani noise dan titik data anomali. Titik data yang dianggap sebagai noise tidak ditugaskan ke kluster mana pun.
4. Hanya memiliki dua parameter: DBSCAN hanya memerlukan dua parameter, eps dan MinPts, yang menjadikannya relatif mudah untuk disesuaikan. Meskipun, pemilihan parameter yang tepat bisa menjadi tantangan, yang akan kita bahas nanti.

Namun, DBSCAN juga memiliki beberapa keterbatasan. Misalnya, algoritma ini dapat memiliki kesulitan ketika densitas kluster sangat bervariasi. Dalam hal ini, pemilihan satu set nilai eps dan MinPts yang dapat menemukan semua kluster mungkin tidak mungkin. Selain itu, DBSCAN tidak dapat menangani data berdimensi tinggi dengan baik.

Meskipun DBSCAN dapat digunakan dalam berbagai aplikasi, ada beberapa kasus di mana algoritma ini khususnya berguna:

1. Deteksi anomali: Karena DBSCAN mampu menangani noise dan titik data anomali, algoritma ini sering digunakan dalam deteksi anomali. Misalnya, dalam deteksi penipuan, titik data anomali mungkin menunjukkan transaksi yang mencurigakan.
2. Segmentasi citra: DBSCAN juga dapat digunakan dalam segmentasi citra, di mana tujuannya adalah untuk mengelompokkan piksel yang serupa. Dalam kasus ini, jarak antara piksel (berdasarkan intensitas, warna, dll.) dapat digunakan sebagai ukuran kesamaan.
3. Analisis spasial: DBSCAN, dengan sifat berbasis densitasnya, adalah pilihan yang baik untuk analisis spasial, seperti identifikasi area padat dalam data lokasi geografis.

Sekarang setelah kamu memiliki pemahaman yang baik tentang apa itu DBSCAN dan kapan harus menggunakannya, kita dapat melanjutkan ke subbab berikutnya, di mana kita akan membahas algoritma DBSCAN lebih detail.

9.2. Algoritma DBSCAN

Setelah memahami apa itu DBSCAN dan konteks penggunaannya, saatnya kita melihat lebih dalam bagaimana algoritma ini bekerja. DBSCAN adalah algoritma berbasis densitas yang mempartisi data menjadi subkelompok atau kluster berdasarkan kepadatan titik data.

Algoritma DBSCAN beroperasi berdasarkan dua konsep utama, yaitu eps dan MinPts. Eps mengacu pada jarak maksimum di antara dua sampel agar satu dapat dianggap berada di lingkungan yang sama dengan yang lainnya, sementara MinPts adalah jumlah minimum titik yang harus ada dalam lingkungan eps dari titik untuk membentuk kluster yang padat.

Berikut adalah langkah-langkah yang ditempuh oleh algoritma DBSCAN:

1. Langkah Pertama: Algoritma dimulai dengan titik acak yang belum dikunjungi dalam dataset. Kemudian, algoritma mencari semua titik data dalam jarak eps dari titik ini (disebut sebagai region query). Jika ada setidaknya MinPts titik dalam jarak eps dari titik awal, maka kluster baru dibentuk dan titik awal dianggap sebagai titik

inti. Jika tidak, titik tersebut ditandai sebagai noise (yang mungkin kemudian berubah menjadi titik batas).
2. Langkah Kedua: Jika kluster baru berhasil dibentuk, maka semua titik yang dapat dijangkau langsung dari titik inti awal ditambahkan ke kluster. Titik dapat dijangkau langsung dari titik lain jika jaraknya kurang dari atau sama dengan eps dan ada setidaknya MinPts titik dalam lingkungan eps.
3. Langkah Ketiga: Proses tersebut diulangi untuk semua titik yang baru ditambahkan ke kluster (yaitu, melakukan region query dan menambahkan titik-titik baru ke kluster jika kondisinya terpenuhi). Langkah ini berlanjut hingga tidak ada titik lain yang dapat ditambahkan ke kluster.
4. Langkah Keempat: Setelah tidak ada titik lain yang dapat ditambahkan ke kluster, algoritma bergerak ke titik berikutnya dalam dataset yang belum dikunjungi dan mengulangi proses dari langkah pertama.
5. Langkah Kelima: Proses ini berlanjut hingga semua titik dalam dataset telah dikunjungi. Pada akhir algoritma, setiap titik akan ditandai sebagai titik inti, titik batas, atau noise.

Titik-titik inti dan batas bersama-sama membentuk kluster, sementara titik-titik noise tidak ditugaskan ke kluster mana pun. Itulah mengapa DBSCAN dikatakan sebagai algoritma clustering yang berbasis densitas: kluster dibentuk berdasarkan area di mana densitas titik-titik (yaitu, jumlah titik dalam jarak tertentu) melebihi ambang batas yang ditentukan.

Satu aspek penting dari algoritma DBSCAN yang perlu ditekankan adalah bahwa outputnya mungkin berbeda tergantung pada urutan data. Namun, dalam banyak kasus, perbedaan ini kecil dan tidak signifikan.

DBSCAN juga memiliki keunggulan berupa efisiensi komputasional. Meskipun dalam kasus terburuk, algoritma ini berjalan dalam waktu O(n^2), jika digunakan struktur data indeks spasial yang efisien (seperti KD-tree atau R*-tree), kompleksitasnya dapat dikurangi menjadi O(n log n).

Dengan kata lain, DBSCAN dapat menangani dataset yang cukup besar dengan efisien, asalkan jumlah dimensi data (yaitu, jumlah fitur atau variabel) tidak terlalu besar. Faktanya, DBSCAN seringkali lebih efisien dibandingkan algoritma klustering berbasis partisi seperti K-Means dalam hal memori dan waktu komputasi.

Namun, kamu harus berhati-hati saat menentukan parameter eps dan MinPts. Pilihan yang tidak tepat untuk parameter ini dapat menghasilkan kluster yang kurang optimal atau bahkan tidak dapat menemukan kluster sama sekali.

Secara umum, algoritma DBSCAN memiliki beberapa kelebihan dan kekurangan. Kelebihannya adalah kemampuannya untuk menemukan kluster berbagai bentuk dan ukuran, serta kemampuannya untuk menangani noise dan outlier. Algoritma ini juga tidak memerlukan kita untuk menentukan jumlah kluster sebelumnya, yang merupakan

keuntungan besar dibandingkan algoritma klustering lainnya seperti K-Means.

Di sisi lain, kekurangan DBSCAN adalah sensitivitasnya terhadap pilihan parameter eps dan MinPts, serta keterbatasannya dalam menangani data berdimensi tinggi. Pada data berdimensi tinggi, konsep 'kepadatan' menjadi kurang intuitif dan sulit untuk didefinisikan, yang bisa mengakibatkan performa DBSCAN menurun.

Selain itu, DBSCAN juga mungkin tidak berfungsi dengan baik jika densitas kluster bervariasi secara signifikan, karena kita hanya dapat menentukan satu nilai eps dan MinPts untuk seluruh data.

Meskipun demikian, dengan pemahaman yang baik tentang data dan pengaturan yang tepat untuk parameter, DBSCAN bisa menjadi alat yang sangat kuat untuk analisis kluster. Dengan kata lain, pengetahuan yang baik tentang algoritma ini, dipadukan dengan pemahaman yang baik tentang data, akan menjadi kunci untuk berhasil menerapkan DBSCAN.

Oleh karena itu, berlatihlah untuk menerapkan algoritma ini pada berbagai jenis data, dan jangan ragu untuk bereksperimen dengan berbagai pengaturan parameter. Ingatlah bahwa tidak ada satu algoritma klustering yang 'terbaik' untuk semua jenis data dan situasi, dan pilihan algoritma yang tepat selalu tergantung pada sifat data dan tujuan analisis kita.

Sekarang setelah kamu mengerti bagaimana algoritma DBSCAN bekerja, kita akan melanjutkan dengan membahas tentang bagaimana menentukan nilai eps dan MinPts yang tepat dalam subbab berikutnya.

9.3. Pemilihan Eps dan MinPts

Ketika kamu menggunakan algoritma DBSCAN untuk pengelompokan data, ada dua parameter utama yang perlu diputuskan: eps (epsilon) dan MinPts. Parameter ini sangat mempengaruhi hasil klustering yang dihasilkan oleh DBSCAN, dan oleh karena itu, memilih nilai yang tepat untuk keduanya sangat penting. Dalam subbab ini, kita akan membahas bagaimana cara memilih nilai yang tepat untuk eps dan MinPts dalam algoritma DBSCAN.

Mari kita mulai dengan eps, atau epsilon. Ini adalah parameter yang menentukan radius lingkungan sekitar setiap titik data. Dengan kata lain, jika jarak antara dua titik kurang dari atau sama dengan eps, maka dua titik tersebut dianggap berada dalam 'lingkungan' satu sama lain.

Pemilihan nilai eps sangat penting karena mempengaruhi seberapa besar kluster yang dapat ditemukan oleh DBSCAN, serta berapa banyak titik data yang dianggap sebagai noise atau outlier. Jika kamu memilih nilai eps yang sangat kecil, maka banyak titik data mungkin tidak akan memiliki cukup 'tetangga' dalam radius eps dan oleh karena itu akan dianggap sebagai noise. Di sisi lain, jika kamu memilih nilai eps yang sangat besar, maka banyak titik data mungkin akan dikelompokkan menjadi satu kluster besar, dan struktur kluster yang

lebih halus mungkin akan hilang.

Jadi, bagaimana cara memilih nilai yang tepat untuk eps? Salah satu metode yang dapat digunakan adalah melalui plot k-dist. Plot k-dist menggambarkan jarak ke tetangga k-th terdekat untuk setiap titik data. Untuk membuat plot ini, kamu harus menghitung jarak ke tetangga k-th terdekat untuk setiap titik data, lalu mengurutkan nilai-nilai ini dalam urutan menurun. Nilai eps yang baik biasanya adalah nilai yang mewakili 'lutut' atau 'siku' dalam plot ini, yaitu, titik di mana peningkatan jarak mulai melambat secara signifikan.

Misalkan kita memiliki data dengan 100 titik dan kita memilih MinPts = 4. Untuk setiap titik, kita menghitung jarak ke 4 tetangga terdekatnya dan kita mendapatkan 100 nilai. Kemudian, kita urutkan nilai-nilai ini dan kita plot. Titik 'lutut' atau 'siku' dalam plot ini adalah nilai eps yang baik.

Namun, perlu diingat bahwa pemilihan nilai eps juga bergantung pada skala data dan juga pada MinPts. Jadi, metode ini mungkin tidak selalu memberikan hasil yang optimal, dan mungkin perlu disesuaikan berdasarkan pengetahuan domain dan eksplorasi data awal.

Sekarang mari kita beralih ke MinPts. MinPts adalah parameter yang menentukan jumlah minimum titik data yang harus ada dalam lingkungan eps sebuah titik data agar titik tersebut dianggap sebagai titik inti.

Pemilihan MinPts juga sangat penting karena mempengaruhi kepadatan kluster yang dihasilkan oleh DBSCAN. Jika kamu memilih nilai MinPts yang sangat kecil, maka banyak kluster kecil mungkin akan ditemukan, dan mungkin ada banyak titik noise. Di sisi lain, jika kamu memilih nilai MinPts yang sangat besar, maka hanya kluster-kluster dengan kepadatan yang sangat tinggi yang akan ditemukan, dan banyak titik data mungkin akan dianggap sebagai noise.

Sebuah aturan umum yang sering digunakan dalam pemilihan MinPts adalah menggunakan nilai 2d, di mana d adalah dimensi dataset. Jadi, misalnya, jika dataset kamu adalah dataset 2D, maka kamu bisa memulai dengan MinPts = 4. Alasan di balik aturan ini adalah bahwa dalam ruang dimensi d, volume bola berdimensi d meningkat secara eksponensial dengan d. Oleh karena itu, jika kamu memiliki MinPts yang tetap dan kamu menambahkan dimensi ke data, kamu akan perlu meningkatkan eps untuk menjaga jumlah titik rata-rata dalam bola eps. Dengan memilih MinPts = 2d, kamu menyesuaikan jumlah titik minimum yang diperlukan untuk membentuk kluster dengan dimensi data.

Namun, seperti eps, pemilihan MinPts juga tergantung pada skala dan sifat data, dan mungkin perlu disesuaikan berdasarkan pengetahuan domain dan eksplorasi data awal.

Jadi, secara umum, untuk memilih nilai yang baik untuk eps dan MinPts dalam DBSCAN, kamu perlu memahami skala dan dimensi data, serta struktur kluster yang diharapkan. Metode seperti plot k-dist dapat membantu dalam pemilihan eps, dan aturan seperti MinPts

= 2*d dapat digunakan sebagai titik awal dalam pemilihan MinPts. Namun, penting untuk selalu memeriksa hasil klustering secara visual (jika dimensi data memungkinkan) dan menyesuaikan parameter berdasarkan hasil tersebut.

Tambahan, penting juga untuk mencoba berbagai kombinasi nilai eps dan MinPts dan melihat bagaimana hasil klustering berubah. Kamu mungkin akan menemukan bahwa ada beberapa kombinasi nilai yang menghasilkan hasil klustering yang masuk akal, dan pilihan akhir mungkin bergantung pada tujuan analisis data kamu.

Ingatlah bahwa DBSCAN adalah algoritma yang sangat kuat dan fleksibel yang bisa bekerja dengan baik pada berbagai jenis data, tetapi keberhasilannya sangat bergantung pada pemilihan parameter yang tepat. Jadi, jangan ragu untuk bereksperimen dan menyesuaikan parameter sampai kamu mendapatkan hasil yang memuaskan.

Akhirnya, harus disebutkan bahwa meskipun DBSCAN bisa bekerja sangat baik pada banyak jenis data, itu bukan alat yang cocok untuk setiap situasi. DBSCAN bekerja dengan baik pada data di mana kluster berbentuk sembarang dan kepadatan titik dalam kluster sama, tetapi mungkin tidak bekerja dengan baik pada data di mana kepadatan kluster bervariasi. Selain itu, DBSCAN tidak bisa menangani dengan baik data berdimensi sangat tinggi karena "kutukan dimensi". Dalam kasus-kasus seperti itu, metode klustering lain mungkin lebih sesuai.

9.4. Implementasi DBSCAN dalam Python

Berikut ini adalah implementasi dari algoritma DBSCAN menggunakan Python. Kami akan membahas setiap langkah dengan detail untuk memastikan kamu memahami apa yang sedang terjadi.

9.4.1. Membuat Data Sintetis

Sebagai contoh, mari kita buat data sintetis dua dimensi yang terdiri dari empat kluster. Kita akan menggunakan fungsi make_blobs dari modul sklearn.datasets:

```
import numpy as np
from sklearn.datasets import make_blobs

# Membuat data sintetis
np.random.seed(0)
X, labels_true = make_blobs(n_samples=1000, centers=4, cluster_std=0.60)
```

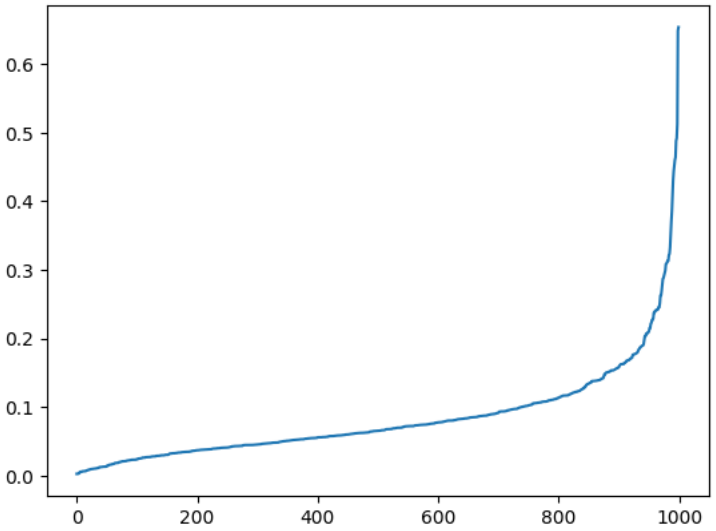
Di sini, X adalah matriks fitur, dan labels_true adalah vektor label yang menunjukkan kluster sebenarnya dari setiap titik (yang kita buat untuk tujuan demonstrasi ini, tetapi tentu saja dalam aplikasi dunia nyata kita biasanya tidak memiliki ini).

9.4.2. Memilih Eps dan MinPts

Seperti yang kita bahas di subbab sebelumnya, pemilihan nilai yang tepat untuk eps dan MinPts sangat penting. Dalam kasus ini, kita akan menggunakan plot k-dist untuk memilih eps, dan kita akan memulai dengan MinPts = 2*dim:

```
from sklearn.neighbors import NearestNeighbors

# Membuat objek NearestNeighbors dan melakukan fitting ke data
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)

# Mengurutkan jarak dan plot
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
plt.show()
```

Output:



‘eps’ dapat dipilih sebagai jarak di mana perubahan jarak yang signifikan terjadi, yaitu ketika grafik garis menunjukkan perubahan yang signifikan.
Kita tentukan MinPts (jumlah minimum titik dalam radius epsilon) sebagai 2 kali jumlah fitur dalam data X. Ini adalah aturan umum yang digunakan untuk memilih MinPts, tetapi dapat disesuaikan berdasarkan karakteristik data dan kebutuhan clustering.

```
# Menentukan eps dan MinPts
eps = 0.3
MinPts = 2 * X.shape[1]
```

9.4.3. Melakukan Clustering dengan DBSCAN

Sekarang kita siap untuk melakukan clustering dengan DBSCAN. Kita akan menggunakan implementasi DBSCAN dari modul sklearn.cluster:

```
from sklearn.cluster import DBSCAN

# Membuat objek DBSCAN dan melakukan fitting ke data
db = DBSCAN(eps=eps, min_samples=MinPts).fit(X)

# Mendapatkan label kluster
labels = db.labels_
```

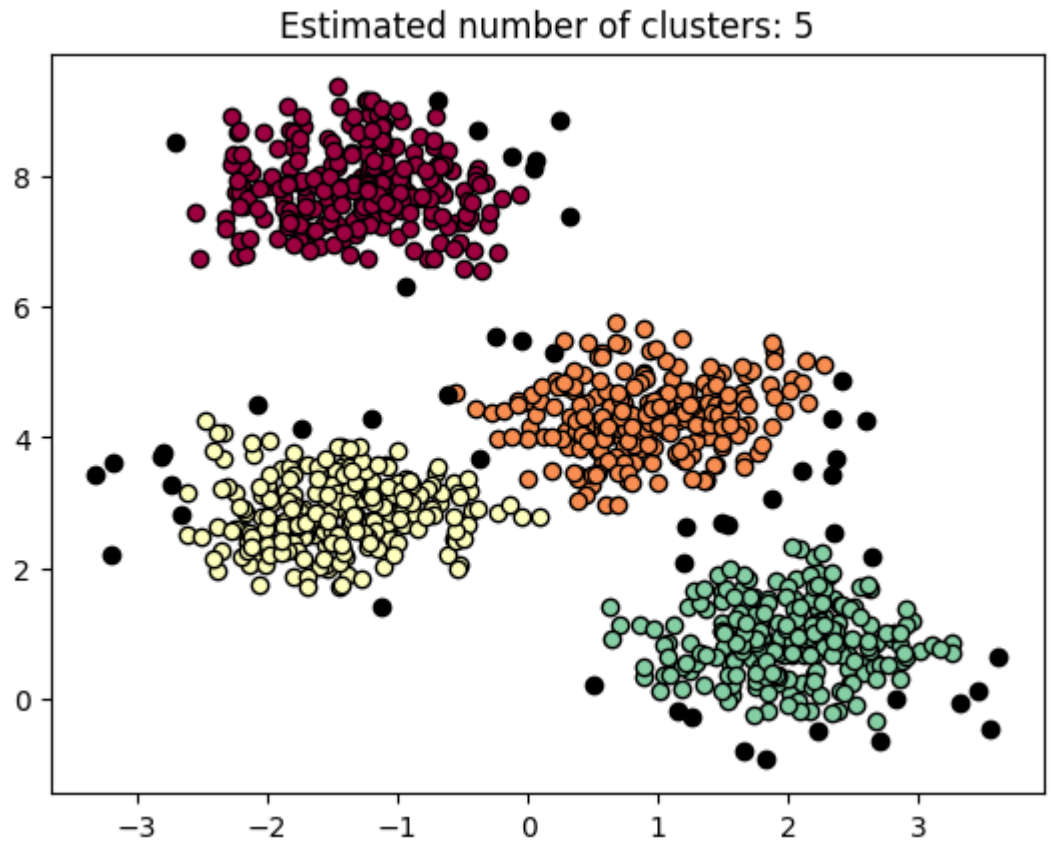
Setelah menjalankan kode ini, labels akan berisi label kluster untuk setiap titik data. Titik yang dianggap sebagai noise oleh DBSCAN akan memiliki label -1.

9.4.4. Visualisasi Hasil

Untuk memahami hasil kita, mari kita visualisasikan kluster:

```
import matplotlib.pyplot as plt

# Membuat plot
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,
len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % len(unique_labels))
plt.show()
```

Output:



Hasilnya adalah plot dari data dengan setiap titik diwarnai sesuai dengan kluster yang ditugaskan oleh DBSCAN. Titik yang dianggap sebagai noise ditampilkan dalam warna hitam.

9.4.5. Hyperparameter Tuning

Seperti yang telah kita sebutkan sebelumnya, pemilihan hyperparameter yang tepat untuk DBSCAN sangat penting untuk hasil yang baik. Dalam contoh kita, kita memilih eps berdasarkan plot k-dist, dan kita memulai dengan MinPts = 2*dim, tetapi ini mungkin tidak selalu memberikan hasil terbaik.

Ada beberapa pendekatan yang dapat kamu coba untuk melakukan tuning hyperparameter. Salah satunya adalah dengan mencoba berbagai kombinasi nilai eps dan MinPts, melakukan clustering dengan setiap kombinasi, dan kemudian membandingkan hasilnya. Sebagai contoh, kamu bisa menggunakan metrik seperti silhouette score atau Davies-Bouldin index untuk mengevaluasi kualitas kluster.

Berikut adalah contoh bagaimana kamu bisa melakukan ini:

```
from sklearn.metrics import silhouette_score

# Menentukan rentang nilai untuk eps dan MinPts
eps_values = np.arange(0.1, 1.0, 0.1)
```

```
MinPts_values = np.arange(2, 10, 1)

# Mencatat skor terbaik dan nilai eps dan MinPts yang sesuai
best_score = -1
best_eps = None
best_MinPts = None

# Mencoba setiap kombinasi eps dan MinPts
for eps in eps_values:
    for MinPts in MinPts_values:
        db = DBSCAN(eps=eps, min_samples=MinPts).fit(X)
        labels = db.labels_

        # Menghitung skor silhouette
        score = silhouette_score(X, labels)

        # Jika skor ini lebih baik dari skor terbaik sejauh ini, simpan
nilai ini dan nilai eps dan MinPts yang sesuai
        if score > best_score:
            best_score = score
            best_eps = eps
            best_MinPts = MinPts

print("Best silhouette score:", best_score)
print("Best eps:", best_eps)
print("Best MinPts:", best_MinPts)
```

Output:

```
Best silhouette score: 0.6509408279221554
Best eps: 0.4
Best MinPts: 7
```

Perlu diingat bahwa tidak ada algoritma clustering yang sempurna untuk semua kasus. Penting untuk memahami sifat data kamu dan apa yang kamu harapkan dari hasil clustering. Untuk DBSCAN, ini berarti memahami bagaimana algoritma ini menangani noise dan bagaimana pilihan eps dan MinPts dapat mempengaruhi hasil.

Dengan pengetahuan ini, kamu sekarang harus memiliki pemahaman yang baik tentang bagaimana menerapkan DBSCAN dalam Python, dan bagaimana memilih dan menyesuaikan parameter agar sesuai dengan data kamu. Selalu ingat bahwa praktik terbaik dalam data science adalah eksplorasi dan eksperimen: jangan ragu untuk mencoba berbagai pendekatan dan melihat apa yang bekerja terbaik untuk kasus kamu.

Bab 10. OPTICS

10.1 Pengertian OPTICS

Ordering Points To Identify the Clustering Structure (OPTICS) adalah algoritme pengelompokan data yang diperkenalkan oleh Ankerst, Breunig, Kriegel, dan Sander pada tahun 1999. OPTICS merupakan pengembangan dari algoritma DBSCAN yang populer dengan beberapa peningkatan yang signifikan. Tujuan utama dari OPTICS adalah untuk mengatasi beberapa keterbatasan dari DBSCAN, terutama sensitivitas terhadap pemilihan parameter dan ketidakmampuan untuk menemukan kluster dengan kepadatan yang berbeda.

Seperti DBSCAN, OPTICS adalah algoritma pengelompokan berbasis kepadatan. Ini berarti bahwa algoritma ini menganggap sebuah kluster sebagai wilayah dalam ruang data di mana banyak objek (atau titik data) berada dekat satu sama lain. Sebaliknya, wilayah di mana objek langka (yaitu, wilayah dengan kepadatan rendah) dianggap sebagai noise atau batas antara kluster.

Namun, berbeda dengan DBSCAN, OPTICS tidak menghasilkan satu set kluster eksplisit. Sebaliknya, algoritma ini menghasilkan apa yang disebut sebagai 'ordering' dari data, yang menyediakan representasi multi-resolusi dari struktur kluster dalam data. Dari 'ordering' ini, berbagai kluster dengan berbagai level kepadatan bisa diidentifikasi.

Bagaimana caranya? OPTICS memperkenalkan konsep 'reachability distance' dan 'core distance', dua ukuran yang memungkinkan kita untuk mengukur kepadatan lokal dari setiap titik dalam data. Dengan mengurutkan data berdasarkan 'reachability distance' ini, kita mendapatkan representasi linear dari struktur kluster data: titik-titik yang berdekatan dalam 'ordering' ini cenderung berada dalam kluster yang sama.

Sebaliknya, titik-titik yang 'reachability distance'-nya tinggi relatif terhadap tetangganya dalam 'ordering' ini dapat dianggap sebagai batas antara kluster. Dengan demikian, alih-alih memutuskan terlebih dahulu jumlah kluster atau kepadatan kluster, kamu bisa menggunakan 'ordering' ini untuk mengeksplorasi struktur data pada berbagai level kepadatan.

Ini menjadikan OPTICS alat yang sangat kuat untuk analisis eksplorasi data, di mana kita mungkin tidak memiliki pengetahuan awal yang jelas tentang struktur kluster data. Selain itu, seperti DBSCAN, OPTICS juga mampu menangani noise dan menemukan kluster dengan bentuk yang tidak biasa.

Namun, perlu diingat bahwa algoritma ini memiliki kompleksitas komputasi yang relatif tinggi: waktu eksekusinya sebanding dengan kuadrat dari jumlah titik data, yang berarti

bahwa untuk dataset yang sangat besar, algoritma ini bisa menjadi lambat. Meskipun demikian, OPTICS tetap menjadi alat yang sangat berharga dalam toolkit setiap data scientist.

10.2 Algoritma OPTICS

Setelah kita membahas pengertian dari OPTICS, mari kita beranjak ke bagian berikutnya, yaitu memahami algoritma dari OPTICS itu sendiri. Kita akan membahas secara mendalam tentang cara kerja algoritma ini dalam memproses data dan bagaimana ia menemukan struktur kluster dalam data.

OPTICS bekerja dengan cara yang mirip dengan DBSCAN dalam banyak hal. Sama seperti DBSCAN, OPTICS memulai dengan titik data acak dan kemudian mengeksplorasi lingkungan sekitarnya berdasarkan kepadatan. Namun, OPTICS memperkenalkan dua konsep penting baru: "reachability distance" dan "core distance". Mari kita bahas satu per satu.

Core Distance: Core distance adalah konsep yang sama dengan DBSCAN. Untuk titik data yang diberikan, core distance adalah jarak ke MinPts-th tetangga terdekat, di mana MinPts adalah parameter yang ditentukan oleh pengguna yang menentukan berapa banyak tetangga yang harus dianggap saat menghitung kepadatan. Jika titik data memiliki kurang dari MinPts tetangga dalam radius Eps (parameter lain yang ditentukan oleh pengguna), core distance dianggap tidak terdefinisi. Jadi, core distance adalah ukuran kepadatan lokal seputar titik data.

Reachability Distance: Untuk dua titik data p dan o, *reachability distance of p with respect to o* adalah jarak maksimum antara core distance of o dan jarak Euclidean antara p dan o. Jadi, reachability distance adalah ukuran jarak yang menyesuaikan dengan kepadatan lokal.

Algoritma OPTICS kemudian beroperasi dengan cara yang mirip dengan DBSCAN, tetapi dengan perbedaan penting: alih-alih hanya mengunjungi titik-titik dalam radius Eps dari titik data saat ini, OPTICS mengunjungi semua titik dalam dataset, dan mengurutkannya berdasarkan reachability distance mereka.

Berikut adalah langkah-langkah algoritma OPTICS:

1. Inisialisasi: Pilih titik data acak yang belum diproses dan hitung core distance-nya. Jika core distance tidak terdefinisi (yaitu, titik tidak memiliki cukup tetangga), pilih titik data lain. Jika core distance terdefinisi, lanjutkan ke langkah 2.
2. Pemrosesan titik: Hitung reachability distance dari titik data saat ini ke semua titik lain dalam dataset. Simpan titik-titik ini dan reachability distance mereka dalam struktur data yang disebut 'ordered file'.
3. Pengurutan: Urutkan 'ordered file' berdasarkan reachability distance. Pilih titik dengan reachability distance terkecil yang belum diproses, dan ulangi langkah 2.
4. Berhenti: Jika semua titik telah diproses, berhenti. Jika tidak, kembali ke langkah 3.

Dalam proses ini, OPTICS menghasilkan 'ordering' dari titik-titik dalam dataset berdasarkan reachability distance mereka. 'Ordering' ini merepresentasikan struktur kluster dalam data dan dapat digunakan untuk mengekstrak kluster pada berbagai tingkat kepadatan.

Namun, perlu diingat bahwa OPTICS sendiri tidak secara eksplisit mengembalikan kluster. Sebaliknya, menghasilkan urutan titik yang merepresentasikan struktur kepadatan data. Kluster kemudian dapat diekstraksi dari urutan ini menggunakan berbagai teknik, seperti memotong diagram reachability pada tingkat kepadatan tertentu.

Untuk memahami algoritma ini dengan lebih baik, mari kita gunakan analogi. Bayangkan kamu sedang melakukan perjalanan di sebuah kota baru. Kamu mulai dari satu tempat dan mulai berjalan, selalu memilih untuk pergi ke tempat yang paling menarik dan paling dekat. Selama perjalananmu, kamu mencatat semua tempat yang kamu lewati dan seberapa jauh mereka dari tempat kamu sebelumnya. Pada akhir perjalanan, kamu akan memiliki daftar semua tempat yang kamu kunjungi, diurutkan berdasarkan seberapa jauh mereka dari tempat sebelumnya. Inilah yang dilakukan OPTICS dengan titik data.

Seperti yang bisa kamu lihat, algoritma OPTICS cukup kompleks dan mengharuskan kita untuk memahami beberapa konsep baru. Namun, ini juga merupakan alat yang sangat kuat untuk analisis kluster, terutama pada data yang memiliki variasi kepadatan.

Sekarang setelah kita telah memahami bagaimana algoritma OPTICS bekerja, mari kita lanjutkan ke subbagian berikutnya di mana kita akan membahas lebih lanjut tentang konsep 'reachability distance' dan 'ordering'.

10.3. Reachability Distance dan Ordering

Ketika membahas algoritma OPTICS (Ordering Points To Identify the Clustering Structure), dua konsep yang sangat penting adalah 'reachability distance' dan 'ordering'. Mari kita jelajahi kedua konsep ini dengan lebih mendalam.

10.3.1. Reachability Distance

Reachability distance atau jarak terjangkau adalah konsep sentral dalam algoritma OPTICS. Ini adalah ukuran yang menentukan sejauh mana satu titik dapat dianggap berada dalam 'jangkauan' titik lain, berdasarkan kepadatan data di sekitarnya.

Reachability distance dari suatu objek p dengan respect terhadap objek o didefinisikan sebagai: max(core-distance(o), euclidean-distance(o, p)). Jika objek o belum diproses, reachability distance tidak didefinisikan.

Di sini, core-distance adalah jarak minimum antara objek dan jumlah minimum objek

(MinPts) dalam set data, dan euclidean-distance adalah jarak Euclidean standar antara dua titik.

Konsep 'reachability distance' ini memungkinkan OPTICS untuk menangani variasi kepadatan dalam data. Dalam kumpulan data di mana kepadatan bervariasi, jarak antara titik data tidak lagi menjadi penentu yang baik untuk apakah titik tersebut seharusnya berada dalam kluster yang sama. Sebagai gantinya, reachability distance, yang mempertimbangkan kepadatan data di sekitar titik, memberikan ukuran yang lebih baik.

10.3.2. Ordering

Konsep kedua yang penting dalam algoritma OPTICS adalah 'ordering'. Seperti yang disebutkan sebelumnya, OPTICS tidak mengembalikan kluster secara eksplisit. Sebaliknya, itu menghasilkan urutan titik yang merepresentasikan struktur kepadatan data. Urutan ini dikenal sebagai 'reachability-plot', di mana titik-titik diperintahkan sedemikian rupa sehingga titik-titik yang berdekatan dalam urutan juga cenderung berdekatan dalam ruang data.

Ordering ini memberikan representasi visual yang kuat dari struktur kluster data. Plot reachability dapat dengan mudah diinterpretasikan: titik data yang berdekatan dalam ruang data akan berdekatan dalam urutan, dan 'valley' dalam plot menunjukkan kluster.

Menginterpretasikan plot reachability memungkinkan kita untuk mengekstrak kluster dari data tanpa harus menentukan jumlah kluster terlebih dahulu. Selain itu, karena OPTICS dapat menangani variasi kepadatan, kluster yang diekstrak dari urutan dapat memiliki kepadatan yang sangat berbeda.

Namun, penting untuk diingat bahwa mengekstraksi kluster dari urutan ini bisa menjadi tantangan tersendiri, terutama jika data memiliki banyak noise atau jika kepadatan antara kluster sangat berbeda.

Kedua konsep ini, reachability distance dan ordering, adalah dasar dari bagaimana OPTICS bekerja. Mereka memungkinkan algoritma untuk menangani variasi kepadatan dalam data, dan untuk menghasilkan urutan titik yang merepresentasikan struktur kluster data.

Dalam subbagian berikutnya, kita akan melihat bagaimana kita dapat menerapkan algoritma OPTICS dalam Python, dan bagaimana kita dapat menggunakan konsep reachability distance dan ordering ini dalam praktek.

10.3.3. Memahami Reachability Distance Lebih Lanjut

Untuk memahami reachability distance secara lebih mendalam, mari kita pertimbangkan contoh berikut. Misalkan kita memiliki sekumpulan titik data dalam ruang dua dimensi, dan kita sedang mencoba untuk menentukan reachability distance dari titik p ke o.

Pertama-tama, kita perlu menentukan core-distance dari titik o. Misalkan kita tentukan MinPts (jumlah minimum titik) sebagai 5. Maka core-distance adalah jarak dari titik o ke titik terdekat kelima. Misalkan ini adalah d1.

Selanjutnya, kita perlu menentukan jarak Euclidean antara titik o dan p. Misalkan ini adalah d2.

Reachability distance dari titik p ke o kemudian didefinisikan sebagai maksimum dari d1 dan d2. Ini berarti bahwa jika titik p sangat dekat dengan o (misalnya, d2 < d1), maka reachability distance akan didominasi oleh core-distance. Sebaliknya, jika titik p cukup jauh dari o (misalnya, d2 > d1), maka reachability distance akan didominasi oleh jarak Euclidean antara o dan p.

10.3.4. Memahami Ordering Lebih Lanjut

Sekarang mari kita pertimbangkan konsep ordering. Misalkan kita memiliki urutan titik yang telah dihasilkan oleh algoritma OPTICS. Bagaimana kita bisa mengekstrak kluster dari urutan ini?

Satu cara adalah dengan melihat plot reachability. Dalam plot ini, titik-titik diwakili sebagai bar vertikal, dan tinggi bar mewakili reachability distance dari titik tersebut.

Kluster kemudian dapat diidentifikasi sebagai "valley" dalam plot ini, di mana titik-titik dalam kluster memiliki reachability distance yang rendah, dan titik-titik antara kluster memiliki reachability distance yang tinggi. Ini berarti bahwa titik-titik dalam kluster akan berdekatan dalam urutan, dan titik-titik antara kluster akan terpisah oleh gap dalam urutan.

Namun, mengekstraksi kluster dari urutan ini bukanlah tugas yang mudah. Salah satu tantangan adalah menentukan apa yang harus dianggap sebagai "valley" dan apa yang harus dianggap sebagai "gap". Selain itu, jika data memiliki banyak noise, atau jika ada variasi kepadatan yang besar antara kluster, maka mengekstraksi kluster bisa menjadi lebih sulit.

Meski demikian, dengan pemahaman yang baik tentang konsep reachability distance dan ordering, kamu akan memiliki alat yang kuat untuk mengidentifikasi dan memahami struktur kluster dalam data.

Pada subbab selanjutnya, kita akan membahas bagaimana implementasi OPTICS dalam Python, dan bagaimana cara kerja algoritmanya dalam praktek.

10.4. Implementasi OPTICS dalam Python

Untuk memahami konsep OPTICS secara mendalam, tidak ada cara yang lebih baik daripada mengimplementasikannya secara langsung. Dalam subbab ini, kita akan membahas implementasi OPTICS menggunakan Python, serta membahas beberapa aspek

penting yang perlu diperhatikan saat menggunakan algoritma ini.

Pertama, perlu diketahui bahwa Scikit-learn, library populer dalam Python untuk machine learning, sudah memiliki implementasi OPTICS yang siap pakai. Meski demikian, kita akan mencoba membuat implementasi sendiri untuk memahami lebih baik bagaimana algoritma ini bekerja.

10.4.1. Persiapan Data

Sebelum kita mulai, kita perlu menyiapkan dataset. Untuk membuat hal-hal sederhana dan fokus pada algoritma, kita akan menggunakan data sintetis yang dibuat menggunakan fungsi make_blobs dari Scikit-learn. Dalam contoh ini, kita akan membuat 500 titik data dengan 3 pusat kluster:

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=500, centers=3, random_state=42)
```

10.4.2. Pembuatan Fungsi Bantuan

Sebelum kita bisa melanjutkan ke implementasi OPTICS, kita perlu membuat beberapa fungsi bantuan. Pertama, kita perlu fungsi untuk menghitung jarak Euclidean antara dua titik. Kita juga perlu fungsi untuk menghitung reachability distance dan fungsi untuk mengurutkan data:

```
import numpy as np

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

def reachability_distance(point, neighbors, MinPts):
    distances = np.sort([euclidean_distance(point, n) for n in
neighbors])
    if len(distances) < MinPts:
        return float('inf')  # Or some other value indicating the issue
    else:
        return max(distances[:MinPts])

def order_seeds(X, point, processed, MinPts, max_value=float('inf')):
    distances = []
    for i, p in enumerate(X):
        if not processed[i]:
            d = euclidean_distance(point, p)
            distances.append((d, i))
    distances.sort()
```

return [i for d, i in distances if d <= max_value]

10.4.3. Implementasi OPTICS

Sekarang kita siap untuk implementasi OPTICS. Di sini adalah langkah-langkah yang akan kita lakukan:

```
def optics(X, MinPts, xi):
    n = X.shape[0]
    processed = [False]*n
    reachability_distances = [float('inf')]*n
    ordering = []

    for p in range(n):
        if not processed[p]:
            processed[p] = True
            ordering.append(p)
            seeds = order_seeds(X, X[p], processed, MinPts)

            while seeds:
                q = seeds.pop(0)  # equivalent to seeds[0] followed by
seeds = seeds[1:]
                processed[q] = True
                ordering.append(q)

                neighbors = order_seeds(X, X[q], processed, MinPts)
                rd = reachability_distance(X[q], neighbors, MinPts)
                reachability_distances[q] =
min(reachability_distances[q], rd)

                for n in neighbors:
                    if not processed[n] and reachability_distances[n] >
rd:
                        reachability_distances[n] = rd
                        if n not in seeds:
                            seeds.append(n)

    return ordering, reachability_distances
```

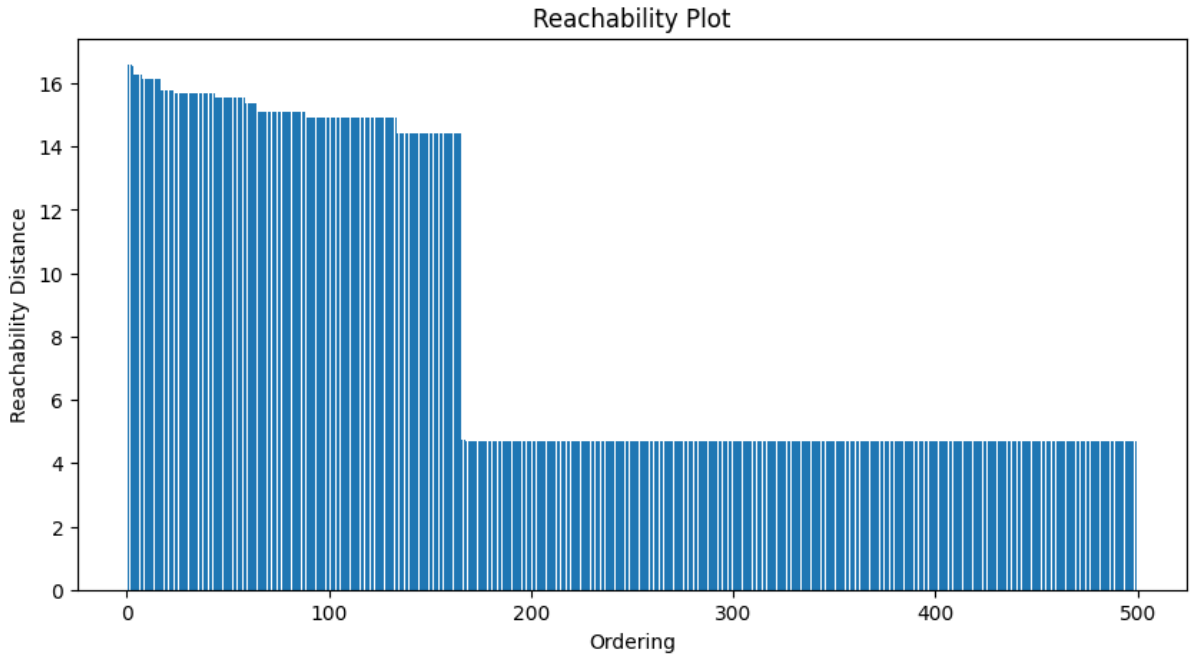
Pada fungsi optics di atas, kita pertama-tama menandai semua titik sebagai belum diproses dan mengatur jarak reachability mereka menjadi tak terhingga. Selanjutnya, kita memproses setiap titik satu per satu. Jika titik tersebut belum diproses, kita menandainya sebagai sudah diproses dan menambahkannya ke ordering. Kemudian, kita mendapatkan semua titik yang belum diproses dalam jarak MinPts dari titik tersebut dan mengurutkannya berdasarkan jarak. Kita mengulangi proses ini sampai semua titik telah diproses.

10.4.4. Visualisasi Hasil

Setelah kita mendapatkan ordering dan reachability distances, kita bisa visualisasikan hasilnya. Untuk melakukan ini, kita bisa membuat plot reachability:

```
import matplotlib.pyplot as plt

ordering, reachability_distances = optics(X, MinPts=5, xi=0.05)

plt.figure(figsize=(10, 5))
plt.bar(range(len(reachability_distances)), [reachability_distances[i]
for i in ordering])
plt.title('Reachability Plot')
plt.xlabel('Ordering')
plt.ylabel('Reachability Distance')
plt.show()
```

Output:



Dalam plot ini, sumbu x adalah ordering dan sumbu y adalah jarak reachability. Titik-titik yang berdekatan dalam plot ini kemungkinan besar berada dalam kluster yang sama.

10.4.5. Melakukan Prediksi Kluster

Menghasilkan label kluster dan memvisualisasikan hasil dari algoritma OPTICS lebih rumit dibandingkan dengan algoritma lain seperti k-means karena OPTICS tidak secara eksplisit menghasilkan kluster. Sebaliknya, itu menghasilkan daftar titik yang diurutkan dan jarak jangkauannya, yang dapat ditafsirkan untuk menemukan kluster.

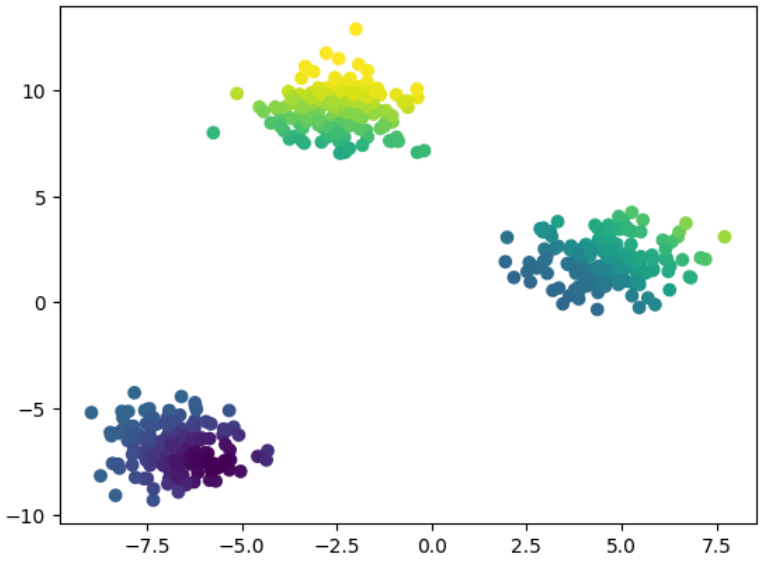
Berikut adalah contoh dasar bagaimana Anda mungkin menghasilkan label kluster dari jarak jangkauan:

```
def ekstrak_kluster(ordering, reachability_distances, xi):
    kluster = [0]*len(ordering)
    id_kluster = 0
    for i in range(1, len(ordering)):
        if reachability_distances[ordering[i]] > xi *
reachability_distances[ordering[i-1]]:
            id_kluster += 1
        kluster[ordering[i]] = id_kluster
    return kluster
```

Fungsi ini memberikan titik ke kluster yang sama jika jarak jangkauannya tidak meningkat sebesar faktor xi atau lebih.
Kamu bisa memvisualisasikan kluster dalam plot scatter. Berikut adalah bagaimana Anda mungkin melakukannya untuk data 2D:

```
import matplotlib.pyplot as plt

# Jalankan algoritma OPTICS
ordering, reachability_distances = optics(X, MinPts=5, xi=0.05)

# Ekstrak kluster
kluster = ekstrak_kluster(ordering, reachability_distances, xi=0.05)

# Plot data point, diberi warna berdasarkan kluster
plt.scatter(X[:, 0], X[:, 1], c=kluster, cmap='viridis')
plt.show()
```

Output:



Harap dicatat bahwa kamu mungkin perlu menyesuaikan parameter xi untuk mendapatkan

hasil yang baik.

Kamu juga bisa memodifikasi fungsi ekstrak_kluster jika kamu ingin menggunakan metode yang berbeda untuk mengekstrak kluster dari jarak jangkauan. Misalnya, kamu bisa mencoba metode berbasis ambang batas atau metode berbasis kepadatan.

Selain itu, jika data kamu lebih dari dua dimensi, kamu mungkin perlu menggunakan teknik reduksi dimensi seperti PCA (Principal Component Analysis) atau t-SNE untuk mereduksi data menjadi dua dimensi sebelum memvisualisasikannya.

Pada akhirnya, penting untuk diingat bahwa tujuan utama kita adalah mencari model yang paling baik menjelaskan data kita. Oleh karena itu, selalu baik untuk mencoba berbagai pendekatan, memahami apa yang dilakukan oleh masing-masing, dan memilih yang terbaik berdasarkan pengetahuan dan pemahaman kita.

Bab 11. Birch

11.1. Pengertian Birch

Dalam ranah machine learning dan data mining, BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) merupakan algoritma clustering yang efisien dan mengesankan. Algoritma ini dirancang khusus untuk menangani data berdimensi tinggi yang jumlahnya sangat besar, yang sering dijumpai dalam berbagai bidang, seperti bioinformatika, pengolahan citra, dan analisis teks. Sebelum membahas lebih jauh tentang BIRCH, mari kita pahami dulu apa itu clustering dan mengapa itu penting.

Clustering adalah proses pengelompokan objek atau data ke dalam grup atau cluster berdasarkan kesamaan antara objek tersebut. Objek dalam satu cluster memiliki kemiripan yang lebih tinggi dibandingkan dengan objek di cluster lain. Clustering digunakan dalam berbagai bidang, termasuk pengenalan pola, pemrosesan gambar, analisis data, dan bioinformatika.

Sebagai seorang praktisi data, kamu pasti pernah menemui tantangan dalam mengelola data berdimensi tinggi. Saat jumlah dimensi dalam data meningkat, banyak algoritma clustering tradisional mulai kehilangan efektivitasnya. Inilah yang dikenal sebagai "kutukan dimensionalitas". Salah satu cara untuk mengatasi tantangan ini adalah dengan menggunakan algoritma clustering yang dirancang khusus untuk menangani data berdimensi tinggi, seperti BIRCH.

BIRCH, yang diperkenalkan oleh Tian Zhang, Raghu Ramakrishnan, dan Miron Livny pada tahun 1996, adalah salah satu algoritma clustering tersebut. Algoritma ini dirancang untuk memanfaatkan sumber daya komputasi dan memori secara efisien saat mengelola data berdimensi tinggi.

Inti dari algoritma BIRCH adalah struktur pohon yang disebut Clustering Feature Tree (CF Tree). CF Tree adalah pohon B-balanced (atau pohon B+) dimana setiap node menyimpan informasi statistik tentang data dalam sub-cluster yang diwakilinya. Informasi ini digunakan untuk mempercepat proses clustering.

Dalam konteks BIRCH, "fitur clustering" adalah ringkasan dari data dalam sub-cluster. Ini adalah vektor yang terdiri dari jumlah data points dalam sub-cluster, jumlah linier, dan jumlah kuadrat. Dengan fitur clustering ini, BIRCH dapat menghitung jarak antara data points dan sub-cluster tanpa harus menghitung jarak antara setiap data point secara individual.

BIRCH melakukan clustering dalam dua tahap. Pada tahap pertama, algoritma ini membangun CF Tree dari data. Kemudian, pada tahap kedua, algoritma ini melakukan

clustering hierarkis pada sub-cluster yang dihasilkan dari CF Tree.

Salah satu keunggulan BIRCH adalah kemampuannya untuk menangani data yang sangat besar. Algoritma ini dirancang untuk memanfaatkan memori secara efisien, dan dapat menyesuaikan diri dengan keterbatasan memori. Selain itu, BIRCH juga dapat menangani data yang datang dalam aliran (streaming data), membuatnya sangat cocok untuk aplikasi real-time.

Namun, BIRCH juga memiliki beberapa keterbatasan. Misalnya, algoritma ini mungkin tidak berkinerja baik untuk data yang memiliki banyak noise atau outlier. Selain itu, BIRCH tidak dirancang untuk menangani data categorical atau binary dengan baik, dan lebih cocok untuk data numerik.

BIRCH juga mengandalkan pemilihan parameter yang tepat untuk menghasilkan hasil clustering yang baik. Ada beberapa parameter penting dalam algoritma BIRCH, termasuk ukuran maksimum dari CF Tree dan threshold untuk menyatukan sub-cluster. Pemilihan parameter ini membutuhkan pemahaman yang baik tentang data dan tujuan clustering, dan mungkin memerlukan beberapa percobaan dan penyesuaian.

Salah satu tantangan dalam menggunakan BIRCH adalah interpretasi hasil clustering. Meskipun BIRCH menyediakan ringkasan statistik tentang data dalam setiap sub-cluster, mungkin sulit untuk memahami makna dari sub-cluster ini tanpa pengetahuan tambahan tentang data.

Namun, meski ada keterbatasan dan tantangan tersebut, BIRCH tetap menjadi algoritma yang berharga dan penting dalam toolbox seorang praktisi data. BIRCH memiliki kemampuan untuk menangani data berdimensi tinggi yang sangat besar dengan efisiensi memori dan komputasi yang luar biasa. Ini menjadikan BIRCH sebagai algoritma yang sangat berguna dalam berbagai aplikasi, dari pengenalan pola hingga analisis teks.

Sebagai penutup, penting untuk diingat bahwa tidak ada algoritma clustering yang terbaik untuk semua kasus. Pilihan algoritma clustering yang tepat sangat bergantung pada sifat data dan tujuan analisis. Oleh karena itu, sangat penting untuk memahami prinsip-prinsip dasar dari berbagai algoritma clustering, termasuk BIRCH, dan bagaimana cara kerjanya.

Dengan pemahaman ini, kamu akan dapat memilih algoritma yang paling cocok untuk tugas clusteringmu dan mengoptimalkannya untuk mendapatkan hasil terbaik. Serta, penting untuk selalu melakukan validasi dan interpretasi hasil clustering dengan cermat, untuk memastikan bahwa hasilnya sesuai dengan pengetahuan domain dan tujuan analisis kamu.

11.2. Algoritma BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) adalah algoritma yang dirancang untuk melakukan clustering pada data yang sangat besar. Algoritma ini mengadopsi pendekatan hierarkis dan incremental, yang memungkinkannya mengelola data yang tidak bisa ditampung dalam memori sekaligus. Algoritma ini bekerja dalam dua langkah utama: pembentukan Clustering Feature (CF) Tree, dan penggunaan algoritma clustering lain (misalnya, agglomerative hierarchical clustering) pada daun CF Tree untuk memperoleh cluster akhir.

Langkah 1: Pembentukan CF Tree

Langkah pertama dalam algoritma BIRCH adalah pembentukan CF Tree. CF Tree adalah struktur data yang digunakan oleh BIRCH untuk menyimpan ringkasan data. Setiap node dalam CF Tree mewakili satu atau lebih titik data, dan menyimpan ringkasan statistik tentang titik-titik data tersebut. Ringkasan ini disimpan dalam bentuk Clustering Feature (CF), yang terdiri dari tiga komponen: jumlah titik data (N), jumlah vektor (LS), dan jumlah kuadrat vektor (SS).

CF Tree dibentuk secara incremental. Artinya, titik data diproses satu per satu, dan CF Tree diperbarui setiap kali titik data baru diproses. Untuk setiap titik data baru, algoritma mencari cabang dalam CF Tree yang paling mirip dengan titik data tersebut. Kemudian, titik data tersebut ditambahkan ke cabang tersebut, dan CF dari cabang tersebut diperbarui.

Selama proses pembentukan CF Tree, algoritma memastikan bahwa ukuran CF Tree tidak melebihi batas ukuran yang ditentukan. Ini dilakukan dengan cara menggabungkan cabang-cabang yang mirip satu sama lain jika ukuran CF Tree melebihi batas ukuran. Kriteria kesamaan antara cabang-cabang ini ditentukan oleh threshold yang diberikan.

Langkah 2: Clustering

Setelah CF Tree dibentuk, langkah kedua dalam algoritma BIRCH adalah melakukan clustering pada daun CF Tree. Pada tahap ini, algoritma clustering lain digunakan. Algoritma yang biasanya digunakan adalah agglomerative hierarchical clustering, tetapi algoritma clustering lain juga dapat digunakan.

Dalam proses ini, setiap daun CF Tree dianggap sebagai satu cluster. Kemudian, algoritma clustering digunakan untuk menggabungkan cluster-cluster ini berdasarkan kesamaan antara CF mereka. Proses ini dilanjutkan hingga jumlah cluster yang diinginkan tercapai.

Algoritma BIRCH memiliki beberapa keunggulan dibandingkan algoritma clustering lainnya. Pertama, algoritma ini mampu menangani data yang sangat besar dengan efisiensi memori dan komputasi yang luar biasa. Kedua, algoritma ini menghasilkan hasil yang

konsisten dan stabil, yang tidak bergantung pada urutan data masuk.

Namun, BIRCH juga memiliki beberapa keterbatasan. Misalnya, algoritma ini tidak cocok untuk data yang memiliki banyak noise atau outlier. Selain itu, algoritma ini juga sensitif terhadap pemilihan parameter, seperti threshold untuk penggabungan cabang dan ukuran maksimum CF Tree.

Sekarang, setelah kamu memahami algoritma BIRCH, mari kita beralih ke komponen penting lainnya dari BIRCH: Clustering Feature Tree. Clustering Feature Tree adalah struktur data yang digunakan oleh BIRCH untuk menyimpan ringkasan data. Setiap node dalam Clustering Feature Tree mewakili satu atau lebih titik data dan menyimpan ringkasan statistik tentang titik-titik data tersebut.

Membangun Clustering Feature Tree adalah proses yang kompleks dan membutuhkan pemahaman yang mendalam tentang struktur data dan algoritme. Namun, dengan pemahaman yang benar, proses ini bisa menjadi mudah dan intuitif.

Secara keseluruhan, BIRCH adalah algoritma clustering yang sangat efisien dan efektif. Algoritma ini mampu menangani data yang sangat besar dengan efisiensi memori dan komputasi yang sangat baik. Selain itu, algoritma ini juga menghasilkan hasil yang konsisten dan stabil, yang tidak bergantung pada urutan data masuk. Meskipun demikian, algoritma ini memiliki beberapa keterbatasan, seperti sensitivitas terhadap noise dan outlier, serta sensitivitas terhadap pemilihan parameter.

11.3. Clustering Feature Tree

Clustering Feature Tree (CF Tree) adalah struktur data pokok yang digunakan dalam algoritma BIRCH untuk menyimpan ringkasan data. Ini berfungsi sebagai inti dari algoritma BIRCH dan merupakan tempat di mana semua komputasi penting terjadi. Dalam sub-bab ini, kita akan membahas detail tentang Clustering Feature Tree, bagaimana ia bekerja, dan peranannya dalam algoritma BIRCH.

11.3.1. Struktur Clustering Feature Tree

Clustering Feature Tree adalah struktur pohon dengan setiap node yang mewakili satu atau lebih titik data. Setiap node menyimpan ringkasan statistik dari titik data yang ia wakili, yang disebut Clustering Feature (CF).

Sebuah Clustering Feature adalah vektor yang terdiri dari tiga elemen: jumlah titik data (N), jumlah vektor titik data (LS), dan jumlah kuadrat vektor titik data (SS). LS dan SS digunakan untuk menghitung pusat massa dan radius dari cluster.

Struktur pohon ini memungkinkan BIRCH untuk menyimpan ringkasan data secara efisien,

yang memungkinkannya untuk mengelola dataset yang sangat besar. Dengan menyimpan ringkasan data bukan data mentah, BIRCH dapat mengurangi kebutuhan memori dan meningkatkan efisiensi komputasi.

11.3.2. Pembangunan Clustering Feature Tree

Pembangunan Clustering Feature Tree adalah proses iteratif yang dimulai dengan pohon kosong. Untuk setiap titik data baru, BIRCH mencari node terdekat di CF Tree dan mencoba untuk memasukkan titik data ke dalam node tersebut. Jika penambahan titik data tidak mengakibatkan peningkatan radius node di atas threshold yang telah ditentukan, titik data akan ditambahkan ke node tersebut. Jika tidak, BIRCH akan mencoba membagi node menjadi dua.

Proses ini berlanjut hingga semua titik data telah dimasukkan ke dalam CF Tree. Selama proses ini, BIRCH dapat memutuskan untuk menggabungkan beberapa node bersama-sama atau membagi node menjadi dua, berdasarkan peningkatan jumlah radius dan jumlah titik data di setiap node.

11.3.3. Pemilihan Threshold

Threshold adalah parameter penting dalam algoritma BIRCH yang menentukan sejauh mana suatu node dapat tumbuh sebelum dibagi menjadi dua. Pemilihan threshold yang tepat adalah kunci untuk mendapatkan hasil yang baik dari BIRCH.

Jika threshold terlalu besar, CF Tree mungkin terlalu kasar dan tidak mampu mengidentifikasi cluster yang lebih kecil. Di sisi lain, jika threshold terlalu kecil, CF Tree mungkin menjadi terlalu rinci dan menghasilkan terlalu banyak cluster.

Oleh karena itu, pemilihan threshold yang tepat adalah kunci untuk menyeimbangkan antara detail dan kekasaran dalam hasil clustering. Biasanya, threshold dipilih berdasarkan pengetahuan sebelumnya tentang data atau melalui proses percobaan dan kesalahan.

11.3.4. Kegunaan Clustering Feature Tree

Clustering Feature Tree adalah alat yang sangat kuat dalam algoritma BIRCH. Dengan menyimpan ringkasan data bukan data mentah, CF Tree memungkinkan BIRCH untuk mengelola dataset yang sangat besar dengan efisiensi memori dan komputasi yang tinggi. Selain itu, dengan menyediakan ringkasan statistik dari titik data, CF Tree memungkinkan BIRCH untuk melakukan analisis yang lebih mendalam dan menghasilkan hasil yang lebih akurat.

Misalnya, dengan menggunakan CF Tree, BIRCH dapat mengidentifikasi dan mengekstrak cluster berdasarkan jarak antara node, yang mungkin tidak mungkin dilakukan jika hanya menggunakan data mentah. Ini memungkinkan BIRCH untuk menghasilkan hasil yang lebih granular dan mendetail, yang mungkin lebih sesuai dengan kebutuhan pengguna.

Selain itu, dengan membagi data menjadi node dan sub-node, CF Tree memungkinkan BIRCH untuk menganalisis data pada berbagai tingkat granularitas. Misalnya, BIRCH dapat melihat gambaran besar dataset dengan menganalisis node tingkat atas, atau dapat melihat detail lebih mendalam dengan menganalisis node tingkat bawah. Ini memberikan fleksibilitas yang besar bagi pengguna dalam menganalisis data mereka.

Akhirnya, Clustering Feature Tree juga memungkinkan BIRCH untuk melakukan pemrosesan incremental. Dengan kata lain, BIRCH dapat memproses titik data satu per satu, daripada harus memproses seluruh dataset sekaligus. Ini sangat berguna untuk dataset yang sangat besar, di mana memproses seluruh dataset sekaligus mungkin tidak praktis atau bahkan tidak mungkin.

Clustering Feature Tree adalah bagian inti dari algoritma BIRCH yang memungkinkan untuk efisiensi memori dan komputasi yang tinggi serta fleksibilitas dalam analisis data. Dengan memahami bagaimana Clustering Feature Tree bekerja, kamu dapat lebih memahami bagaimana algoritma BIRCH bekerja dan bagaimana memanfaatkannya untuk kebutuhan clustering data kamu.

Namun, meskipun Clustering Feature Tree adalah alat yang sangat kuat, penting untuk diingat bahwa ia tidak sempurna. Seperti semua algoritma clustering, BIRCH dan Clustering Feature Tree memiliki batasannya sendiri, dan hasilnya dapat bervariasi tergantung pada karakteristik data dan parameter yang dipilih. Oleh karena itu, penting untuk selalu melakukan validasi dan penyetelan model untuk memastikan hasil yang optimal.

Secara keseluruhan, Clustering Feature Tree adalah alat yang sangat kuat dalam toolbox clustering data. Dengan pemahaman yang kuat tentang bagaimana itu bekerja, kamu dapat memanfaatkan kekuatannya untuk membantu memecahkan masalah clustering data yang kompleks.

11.4. Implementasi BIRCH dalam Python

Setelah memahami prinsip dasar dan algoritma BIRCH, kita akan beralih ke bagian implementasi. Python, dengan pustaka machine learning-nya yang luas dan kuat, seperti Scikit-Learn, menyediakan dukungan yang luar biasa untuk BIRCH dan banyak algoritma clustering lainnya.

11.4.1. Persiapan Data

Pertama-tama, kita perlu membuat dataset sintetis untuk demonstrasi ini. Scikit-Learn memiliki modul yang dapat digunakan untuk menghasilkan dataset sintetis. Namun, sebelum kita melangkah lebih jauh, mari impor semua pustaka yang diperlukan.

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.cluster import Birch
import matplotlib.pyplot as plt
```

Berikut adalah cara membuat dataset sintetis menggunakan Scikit-Learn:

```
# Mendefinisikan jumlah sampel, pusat cluster, dan standar deviasi
setiap cluster
n_samples = 10000
centers = [[1, 1], [-1, -1], [1, -1]]
stds = [0.6, 0.5, 0.7]

# Membuat dataset sintetis
X, labels_true = make_blobs(n_samples=n_samples, centers=centers,
cluster_std=stds, random_state=0)

# Melakukan standardisasi fitur
X = StandardScaler().fit_transform(X)
```

11.4.2. Menentukan dan Melatih Model

Setelah kita memiliki dataset, kita bisa membuat model BIRCH dan melatihnya dengan data tersebut. Kita juga perlu menentukan parameter untuk model. Dalam hal ini, kita akan menggunakan parameter default, namun kamu dapat menyesuaikannya sesuai kebutuhan.

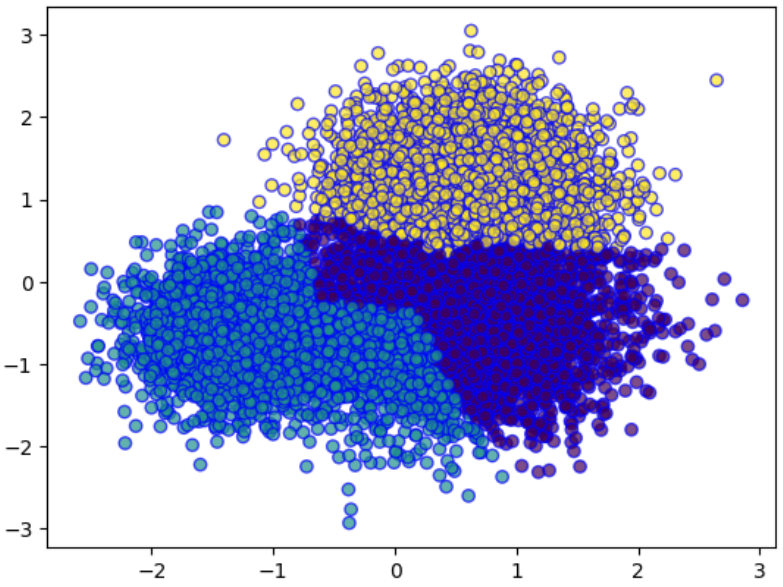
```
# Mendefinisikan model BIRCH
brc = Birch(n_clusters=3)

# Melatih model
brc.fit(X)
```

11.4.3. Visualisasi Hasil

Setelah model dilatih, kita dapat menggunakan model tersebut untuk melakukan prediksi pada data, dan kita dapat memvisualisasikan hasilnya.

```
# Memprediksi label cluster
labels = brc.predict(X)

# Plot data
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis', alpha=0.7,
edgecolors='b')
plt.show()
```

Output:



11.4.4. Hyperparameter Tuning

Hyperparameter tuning adalah proses menyesuaikan parameter model untuk meningkatkan kinerjanya. Dalam kasus BIRCH, beberapa parameter yang dapat disesuaikan antara lain:

threshold: Ambang batas untuk memutuskan apakah suatu titik data harus dimasukkan ke dalam suatu node CF atau tidak. Nilai yang lebih besar akan menghasilkan CF Tree yang lebih padat dan lebih sedikit cluster, sedangkan nilai yang lebih rendah akan menghasilkan CF Tree yang lebih jarang dan lebih banyak cluster.

branching_factor: Faktor percabangan untuk CF Tree. Nilai yang lebih besar akan menghasilkan CF Tree yang lebih lebar dan lebih dangkal, sedangkan nilai yang lebih rendah akan menghasilkan CF Tree yang lebih sempit dan lebih dalam.
n_clusters: Jumlah cluster yang diinginkan. Nilai ini dapat digunakan untuk mempengaruhi jumlah cluster akhir dengan menjalankan tahap post-processing pada CF Tree menggunakan algoritma lain, seperti k-means.

Untuk melakukan hyperparameter tuning, kita bisa mencoba kombinasi berbagai nilai parameter tersebut dan melihat mana yang memberikan hasil terbaik. Biasanya, ini dilakukan dengan teknik seperti grid search atau random search. Namun, di sini kita akan mencobanya secara manual untuk memahami dampak dari masing-masing parameter.

```
# Mendefinisikan daftar nilai untuk setiap parameter
thresholds = [0.1, 0.5, 1]
branching_factors = [20, 50, 100, 200]
n_clusters = [2, 3, 4, 5]

# Melakukan tuning parameter
```

```
for t in thresholds:
    for b in branching_factors:
        for n in n_clusters:
            brc = Birch(threshold=t, branching_factor=b, n_clusters=n)
            brc.fit(X)
            labels = brc.predict(X)
            print(f"threshold={t}, branching_factor={b}, n_clusters={n},
silhouette_score={silhouette_score(X, labels)}")
```

Output:

```
threshold=0.1, branching_factor=20, n_clusters=2, silhouette_score=0.4343871641114348
threshold=0.1, branching_factor=20, n_clusters=3, silhouette_score=0.46582936233428
threshold=0.1, branching_factor=20, n_clusters=4, silhouette_score=0.3681693265610835
threshold=0.1, branching_factor=20, n_clusters=5, silhouette_score=0.3489062111672461
threshold=0.1, branching_factor=50, n_clusters=2, silhouette_score=0.4076674667932986
threshold=0.1, branching_factor=50, n_clusters=3, silhouette_score=0.33651221565915
threshold=0.1, branching_factor=50, n_clusters=4, silhouette_score=0.26982322588597196
threshold=0.1, branching_factor=50, n_clusters=5, silhouette_score=0.2987404061249491
threshold=0.1, branching_factor=100, n_clusters=2, silhouette_score=0.4103124336825796
threshold=0.1, branching_factor=100, n_clusters=3, silhouette_score=0.47351160896672334
threshold=0.1, branching_factor=100, n_clusters=4, silhouette_score=0.4175522306441507
threshold=0.1, branching_factor=100, n_clusters=5, silhouette_score=0.4047004442096465
threshold=0.1, branching_factor=200, n_clusters=2, silhouette_score=0.3873452081141163
```

Perhatikan bahwa kita menggunakan silhouette_score sebagai metrik untuk mengevaluasi kinerja model. Silhouette score adalah metrik yang digunakan untuk mengukur sejauh mana objek dalam satu cluster mirip satu sama lain dibandingkan dengan objek di cluster lain.

11.4.5. Menyimpan dan Memuat Model

Setelah kita menemukan kombinasi parameter yang memberikan hasil terbaik, kita mungkin ingin menyimpan model tersebut sehingga kita dapat menggunakannya kembali di masa mendatang tanpa perlu melatihnya lagi.

Python menyediakan modul pickle yang dapat digunakan untuk menyimpan dan memuat model. Berikut adalah cara menyimpan model BIRCH ke file:

```
import pickle

# Menyimpan model
with open('model.pkl', 'wb') as f:
    pickle.dump(brc, f)
```

Dan berikut ini adalah cara memuat model dari file:

```
# Memuat model
with open('model.pkl', 'rb') as f:
```

```
    brc = pickle.load(f)
```

Dengan ini, kita telah melihat bagaimana menerapkan algoritma BIRCH menggunakan Python dan Scikit-Learn, dari persiapan data, pelatihan model, visualisasi hasil, tuning hyperparameter, hingga penyimpanan dan pemuatan model. Pada titik ini, kamu harus memiliki pemahaman yang baik tentang cara kerja algoritma BIRCH dan bagaimana menggunakannya dalam praktik.

Bab 12. Gaussian Mixture Model

12.1. Pengertian Gaussian Mixture Model

Gaussian Mixture Model (GMM) adalah model statistik yang mewakili probabilitas keberadaan sub-populasi di dalam keseluruhan populasi. GMM merupakan model probabilitas yang menafsirkan data sebagai kumpulan objek Gaussian. Gaussian sendiri adalah distribusi probabilitas yang bentuknya seperti lonceng, yang juga dikenal sebagai distribusi normal.

Model ini banyak digunakan dalam berbagai bidang, mulai dari analisis gambar dan suara, hingga algoritma pembelajaran mesin. GMM memiliki kemampuan untuk memodelkan data yang kompleks, yang mungkin tidak dapat dihandle oleh algoritma clustering lain seperti k-means atau hierarchical clustering.

12.1.1. Mengapa Gaussian?

Sebelum memahami GMM, pertama-tama kita harus memahami apa itu distribusi Gaussian. Distribusi Gaussian atau distribusi normal adalah distribusi probabilitas yang paling umum digunakan dalam statistika. Distribusi ini memiliki bentuk seperti lonceng dan didefinisikan oleh dua parameter, yaitu rata-rata (mean) dan standar deviasi (standard deviation). Rata-rata menunjukkan pusat distribusi, sementara standar deviasi mengukur sebaran data di sekitar rata-rata.

Dalam konteks GMM, kita menganggap bahwa data dihasilkan dari beberapa distribusi Gaussian. Masing-masing distribusi Gaussian ini mewakili satu cluster. Kenapa Gaussian? Alasannya adalah karena distribusi Gaussian memiliki beberapa properti yang membuatnya sangat berguna dalam statistika dan pembelajaran mesin:

Simpel: Distribusi Gaussian hanya membutuhkan dua parameter (mean dan variance) untuk sepenuhnya ditentukan. Ini membuatnya lebih sederhana dibandingkan dengan distribusi lain yang mungkin membutuhkan lebih banyak parameter.

Mudah dihitung: Rumus matematika distribusi Gaussian relatif sederhana, yang memungkinkan perhitungan yang cepat dan efisien.

Universality: Menurut teorema limit pusat (central limit theorem), jumlah dari variabel acak independen cenderung mendekati distribusi normal, tidak peduli distribusi awal dari variabel tersebut. Ini berarti bahwa distribusi Gaussian dapat dengan baik memodelkan berbagai jenis data dalam praktiknya.

12.1.2. Konsep Mixture

Dalam GMM, kita menganggap bahwa data dihasilkan dari beberapa distribusi Gaussian. Setiap Gaussian ini disebut sebagai komponen, dan GMM adalah gabungan (mixture) dari komponen-komponen ini. Setiap komponen mewakili satu cluster. Oleh karena itu, GMM memungkinkan kita untuk memodelkan data yang berasal dari beberapa grup atau cluster.

Misalnya, bayangkan bahwa kita memiliki data tentang tinggi orang dewasa. Jika kita mencoba memodelkan data ini dengan satu distribusi Gaussian, kita mungkin akan mendapatkan hasil yang kurang baik karena ada dua grup yang berbeda dalam data, yaitu pria dan wanita. Dalam kasus ini, akan lebih baik jika kita menggunakan GMM dengan dua komponen, di mana setiap komponen mewakili satu grup.

12.1.3. Soft Clustering vs Hard Clustering

Salah satu aspek unik dari GMM adalah pendekatannya terhadap konsep clustering. Dalam banyak algoritma clustering, seperti k-means, setiap titik data diberi label dengan cluster tertentu. Ini disebut 'hard clustering' - setiap titik data hanya dapat menjadi bagian dari satu cluster.

Namun, GMM mengambil pendekatan yang berbeda. Dalam GMM, setiap titik data memiliki probabilitas tertentu untuk setiap cluster. Misalnya, titik data A bisa memiliki probabilitas 70% berada di cluster 1, dan 30% berada di cluster 2. Ini disebut 'soft clustering' atau clustering probabilistik. Pendekatan ini memberikan fleksibilitas tambahan dan dapat lebih baik memodelkan realitas yang sering kali tidak hitam dan putih.

12.1.4. Latent Variables

Konsep lain yang penting dalam GMM adalah variabel laten (latent variables). Dalam konteks GMM, variabel laten adalah variabel yang menentukan cluster mana yang menghasilkan setiap titik data. Misalnya, dalam contoh tinggi manusia tadi, variabel laten bisa adalah jenis kelamin.

Variabel laten ini tidak dapat diobservasi secara langsung (itulah sebabnya disebut 'laten'), tapi kita bisa mengestimasi mereka dari data yang kita punya. Dalam GMM, kita mencoba mempelajari parameter dari distribusi Gaussian (yaitu, mean dan standar deviasi), serta variabel laten, yang dalam hal ini adalah probabilitas setiap titik data berasal dari setiap cluster.

12.1.5. GMM dan Maximum Likelihood

GMM memanfaatkan prinsip 'maximum likelihood' untuk mempelajari parameter-parameter model. Intinya, kita mencoba mencari parameter yang paling mungkin menghasilkan data yang kita punya. Dalam konteks GMM, kita mencoba mencari parameter distribusi Gaussian (mean dan standar deviasi) serta probabilitas setiap cluster yang paling mungkin menghasilkan data kita.

Sayangnya, karena GMM adalah model yang cukup kompleks, kita tidak bisa menemukan solusi maksimum likelihood secara langsung. Sebaliknya, kita harus menggunakan algoritma khusus, seperti Expectation-Maximization, yang akan kita bahas nanti.

Secara umum, meskipun GMM adalah model yang cukup kompleks, itu sangat fleksibel dan kuat, dan dapat digunakan untuk memodelkan berbagai jenis data. GMM mampu memodelkan data yang memiliki struktur yang lebih kompleks dibandingkan dengan model clustering lainnya, seperti k-means, dan ini menjadikannya pilihan yang sangat baik untuk banyak aplikasi pembelajaran mesin dan statistik.

12.2. Expectation-Maximization (EM) Algorithm

Expectation-Maximization (EM) adalah algoritma iteratif yang digunakan dalam Gaussian Mixture Model (GMM) untuk estimasi maksimum likelihood ketika kita memiliki variabel laten. Algoritma ini digunakan secara luas di berbagai bidang, termasuk statistik, pembelajaran mesin, dan pengolahan sinyal.

Algoritma EM beroperasi melalui dua langkah utama yang berulang-ulang: langkah Expectation (E) dan langkah Maximization (M). Secara umum, langkah E bertanggung jawab untuk mengestimasi variabel laten, sementara langkah M bertanggung jawab untuk memperbarui parameter model.

12.2.1. Langkah Expectation (E)

Langkah Expectation, juga dikenal sebagai langkah E, melibatkan pembuatan fungsi yang dikenal sebagai fungsi harapan log-likelihood. Di sini, 'harapan' merujuk pada nilai rata-rata dari suatu variabel acak, yang dalam hal ini adalah log-likelihood.

Pada langkah ini, kita menghitung harapan log-likelihood dengan menganggap parameter model kita tetap dan memperbarui estimasi kita tentang variabel laten. Untuk Gaussian Mixture Model, ini berarti kita menghitung probabilitas bahwa setiap titik data berasal dari setiap cluster, dengan asumsi nilai parameter model kita saat ini.

12.2.2. Langkah Maximization (M)

Setelah langkah Expectation, kita bergerak ke langkah Maximization, atau langkah M. Di sini, kita memperbarui parameter model kita untuk memaksimalkan fungsi harapan log-likelihood yang kita hitung pada langkah E.

Untuk GMM, ini berarti kita memperbarui parameter distribusi Gaussian (mean dan standar deviasi) dan probabilitas setiap cluster, berdasarkan probabilitas titik data yang berasal dari setiap cluster yang kita hitung pada langkah E.

12.2.3. Iterasi EM

Setelah langkah M, kita kembali ke langkah E dan mengulangi proses ini sampai konvergensi, yaitu sampai parameter model kita berhenti berubah secara signifikan antara iterasi. Ini memastikan bahwa kita telah menemukan solusi maksimum likelihood yang paling baik untuk data kita.

12.2.4. EM dan Inisialisasi

Salah satu aspek penting dari algoritma EM adalah bagaimana kita menginisialisasi parameter model kita. Jika kita memulai dengan inisialisasi yang buruk, EM bisa terjebak dalam apa yang dikenal sebagai maksimum lokal, yang berarti kita mungkin tidak menemukan solusi terbaik.

Ada beberapa strategi yang bisa digunakan untuk mengatasi ini. Salah satunya adalah menggunakan metode seperti k-means untuk inisialisasi awal. Metode lain adalah menjalankan EM beberapa kali dengan inisialisasi yang berbeda dan memilih model dengan likelihood tertinggi.

12.2.5. Keuntungan dan Kekurangan EM

Salah satu keuntungan utama EM adalah fleksibilitas dan kemampuannya untuk memodelkan situasi yang kompleks dan tidak pasti dengan variabel laten. Namun, ini juga berarti EM bisa menjadi cukup rumit dan memerlukan lebih banyak waktu komputasi dibandingkan dengan beberapa metode lainnya.

Selain itu, algoritma EM juga memiliki beberapa kekurangan. Seperti disebutkan sebelumnya, salah satu tantangan dengan EM adalah masalah inisialisasi dan potensi untuk terjebak di maksimum lokal. Selain itu, algoritma EM juga cenderung lebih sensitif terhadap outlier atau data noise dibandingkan dengan beberapa metode lainnya.

Terlepas dari kekurangannya, algoritma EM tetap menjadi alat yang sangat berguna dalam toolbox setiap data scientist. Kemampuannya untuk menangani situasi yang kompleks dan tidak pasti membuatnya sangat berharga di banyak aplikasi nyata.

12.2.6. EM di Luar GMM

Meskipun kita telah membahas algoritma EM dalam konteks Gaussian Mixture Models, penting untuk diingat bahwa EM adalah algoritma umum yang dapat digunakan dalam berbagai situasi. Di mana pun kita memiliki model dengan variabel laten, dan kita ingin memaksimalkan likelihood, EM dapat digunakan.

Contoh lain penggunaan EM termasuk pemodelan topik dalam Natural Language Processing (NLP), di mana kita ingin memodelkan distribusi topik dalam dokumen dan distribusi kata dalam topik, tetapi topik itu sendiri adalah variabel laten. EM juga digunakan dalam sistem rekomendasi, di mana kita ingin memodelkan preferensi pengguna dan

atribut item, tetapi preferensi tersebut tidak diamati secara langsung.

12.3. Model Selection dan Akaike Information Criterion (AIC)

Setelah memahami konsep dasar Gaussian Mixture Models (GMM) dan bagaimana algoritma Expectation-Maximization (EM) digunakan untuk mengestimasi parameter model, kamu mungkin bertanya, berapa banyak komponen Gaussian yang harus digunakan? Bagaimana kita menentukan model mana yang paling baik mewakili data kita? Nah, itulah yang akan kita bahas di bagian ini.

12.3.1. Memilih Jumlah Komponen Gaussian

Salah satu pertanyaan penting dalam penerapan GMM adalah berapa banyak komponen Gaussian yang harus digunakan. Jumlah ini tidak diketahui sebelumnya dan harus ditentukan berdasarkan data. Menggunakan terlalu sedikit komponen bisa berarti kita tidak menangkap semua pola dalam data, sedangkan menggunakan terlalu banyak komponen bisa berarti kita overfit data, yaitu model kita terlalu kompleks dan mungkin tidak akan berfungsi dengan baik pada data baru.

12.3.2. Overfitting dan Underfitting

Sebelum kita melanjutkan, mari kita bahas konsep overfitting dan underfitting. Overfitting terjadi ketika model kita terlalu kompleks dan menangkap noise atau fluktuasi acak dalam data pelatihan alih-alih pola umum. Model seperti ini mungkin memiliki performa yang sangat baik pada data pelatihan, tetapi performanya akan buruk pada data baru. Sebaliknya, underfitting terjadi ketika model kita terlalu sederhana dan tidak menangkap pola penting dalam data. Model ini mungkin tidak berfungsi dengan baik baik pada data pelatihan maupun data baru.

12.3.3. Model Selection

Salah satu cara untuk memilih model yang tepat adalah dengan menggunakan teknik yang dikenal sebagai model selection. Model selection adalah proses memilih model yang paling baik mewakili data kita dari kumpulan model kandidat. Ada berbagai kriteria yang dapat digunakan dalam model selection, dan salah satunya adalah Akaike Information Criterion (AIC).

12.3.4. Akaike Information Criterion (AIC)

Akaike Information Criterion (AIC) adalah metode yang digunakan untuk membandingkan model statistik yang berbeda pada dataset yang sama. AIC mengukur sejauh mana model mewakili data, dengan mempertimbangkan kompleksitas model. Secara umum, model yang lebih kompleks akan menyesuaikan data dengan lebih baik, tetapi juga akan memiliki risiko overfitting yang lebih besar. AIC membantu kita menemukan titik tengah antara underfitting dan overfitting.

AIC didefinisikan sebagai:

AIC = 2k - 2ln(L)

di mana k adalah jumlah parameter dalam model dan L adalah maksimum nilai likelihood yang dihasilkan oleh model. Dengan kata lain, AIC memberi penalti kepada model dengan banyak parameter (untuk mencegah overfitting) dan memberi poin kepada model yang memaksimalkan likelihood data.

Model dengan nilai AIC terendah dianggap yang terbaik. Jadi, dalam konteks GMM, kita bisa mencoba berbagai jumlah komponen Gaussian, menghitung AIC untuk masing-masing, dan memilih model dengan AIC terendah.

12.3.5. Menghitung AIC

Mari kita bahas bagaimana cara menghitung AIC untuk model GMM. Pertama, kita perlu menentukan jumlah parameter dalam model. Dalam konteks GMM, parameter adalah mean, varians, dan bobot dari masing-masing komponen Gaussian. Jadi, jika kita memiliki c komponen Gaussian dalam model dan data d-dimensi, jumlah parameter adalah k = c * (1 + d + d * (d + 1) / 2), di mana (1 + d + d * (d + 1) / 2) adalah jumlah parameter per komponen (1 untuk bobot, d untuk mean, dan d * (d + 1) / 2 untuk matriks kovariansi).

Selanjutnya, kita perlu menghitung nilai maksimum likelihood. Ini bisa didapatkan dari proses EM yang kita gunakan untuk mengestimasi parameter model. Kita ingat bahwa tujuan EM adalah untuk memaksimalkan likelihood data.

Setelah kita mendapatkan kedua nilai tersebut, kita bisa menggantikannya ke dalam rumus AIC dan mendapatkan nilai AIC untuk model kita.

12.3.6. Menggunakan AIC untuk Memilih Jumlah Komponen Gaussian

Setelah kita tahu cara menghitung AIC, kita bisa menggunakannya untuk memilih jumlah komponen Gaussian dalam model GMM. Prosesnya adalah sebagai berikut:

Tentukan berbagai jumlah komponen Gaussian yang mungkin, misalnya dari 1 hingga 10. Untuk setiap jumlah komponen, buat model GMM dan latih dengan data. Hitung AIC untuk model tersebut.
Pilih model dengan AIC terendah.
Perlu dicatat bahwa sementara AIC adalah alat yang bermanfaat, itu bukanlah aturan yang keras dan cepat. Kadang-kadang, model dengan AIC terendah mungkin masih overfitting data, atau mungkin masih terlalu sederhana. Selalu baik untuk memeriksa model secara visual (jika dimungkinkan), dan mempertimbangkan pengetahuan domain dan tujuan analisis ketika memilih model.

12.3.7. Metode Seleksi Model Lainnya

Sementara AIC adalah metode yang populer dan sering digunakan, ada juga metode lain yang bisa digunakan untuk seleksi model. Salah satunya adalah Bayesian Information Criterion (BIC), yang mirip dengan AIC tetapi memberikan penalti yang lebih besar terhadap kompleksitas model. Metode lainnya termasuk cross-validation, di mana model diuji pada subset data yang tidak digunakan dalam pelatihan, dan metode berbasis likelihood seperti likelihood ratio test.

Semua metode ini memiliki kelebihan dan kekurangan sendiri, dan pilihan metode tergantung pada konteks dan tujuan analisis. Penting untuk diingat bahwa tidak ada metode yang sempurna untuk semua situasi, dan seringkali perlu dilakukan beberapa percobaan dan penyesuaian untuk menemukan model yang paling baik mewakili data.

12.4. Implementasi Gaussian Mixture Model dalam Python

Setelah memahami konsep Gaussian Mixture Model (GMM), Expectation-Maximization (EM) Algorithm, dan Akaike Information Criterion (AIC), sekarang kita beralih ke penerapannya dalam Python.

12.4.1. Mengimpor Library yang Dibutuhkan

Sebelum memulai, kita perlu mengimpor library yang dibutuhkan. Untuk tutorial ini, kita akan menggunakan numpy untuk manipulasi array, matplotlib untuk visualisasi data, dan sklearn.mixture untuk model GaussianMixture.

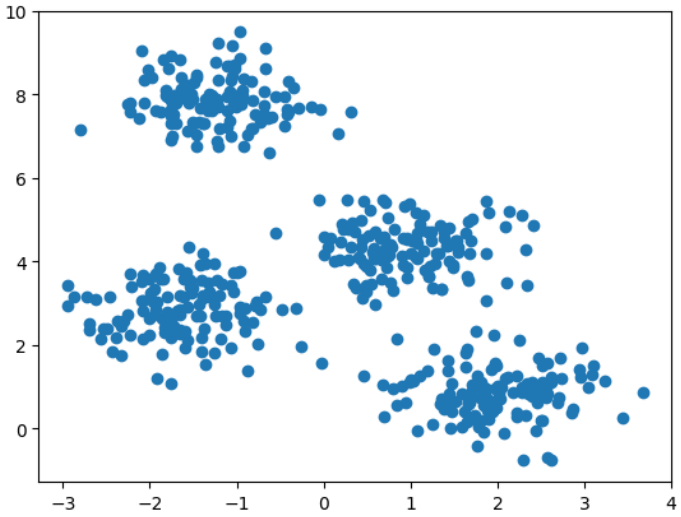
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
```

12.4.2. Membuat Data Sintetis

Kemudian, kita akan membuat data sintetis untuk proses klastering. Kita akan menggunakan fungsi make_blobs dari sklearn.datasets.

```
X, y = make_blobs(n_samples=500, centers=4, cluster_std=0.60,
random_state=0)
plt.scatter(X[:,0], X[:,1])
```

Output:



12.4.3. Membuat dan Melatih Model GMM

Selanjutnya, kita akan membuat model GMM menggunakan GaussianMixture dari sklearn.mixture. Kita akan membuat model dengan 4 komponen (sesuai dengan jumlah pusat yang kita tentukan saat membuat data sintetis), dan kita akan menggunakan full covariance type yang berarti setiap komponen memiliki matriks kovariansi sendiri.

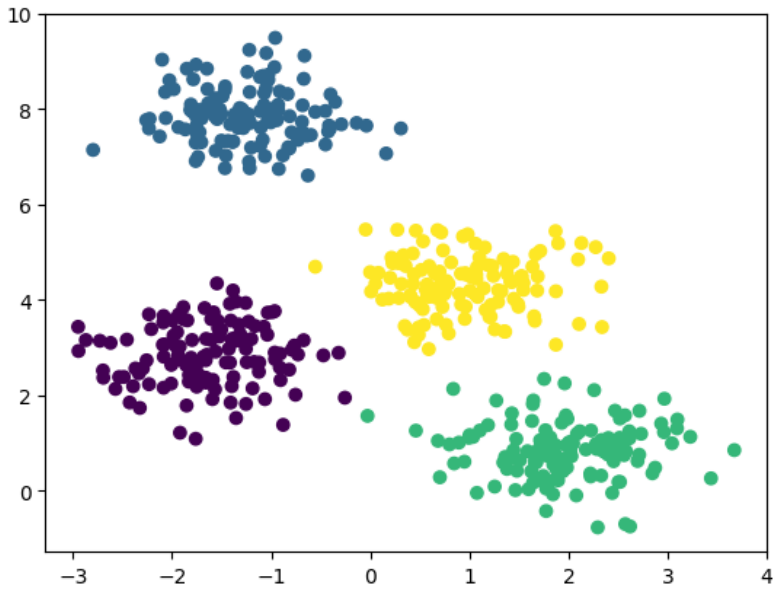
```
gmm = GaussianMixture(n_components=4, covariance_type='full')
gmm.fit(X)
```

12.4.4. Melakukan Prediksi

Setelah model dilatih, kita bisa menggunakan fungsi predict untuk mendapatkan label klaster untuk setiap sampel data.

```
labels = gmm.predict(X)
plt.scatter(X[:,0], X[:,1], c=labels)
```

Output:



12.4.5. Evaluasi Model dengan AIC

Kita dapat menghitung AIC model kita dengan menggunakan method aic yang tersedia pada objek model GaussianMixture.

```
aic = gmm.aic(X)
print(f'AIC: {aic}')
```

Output:

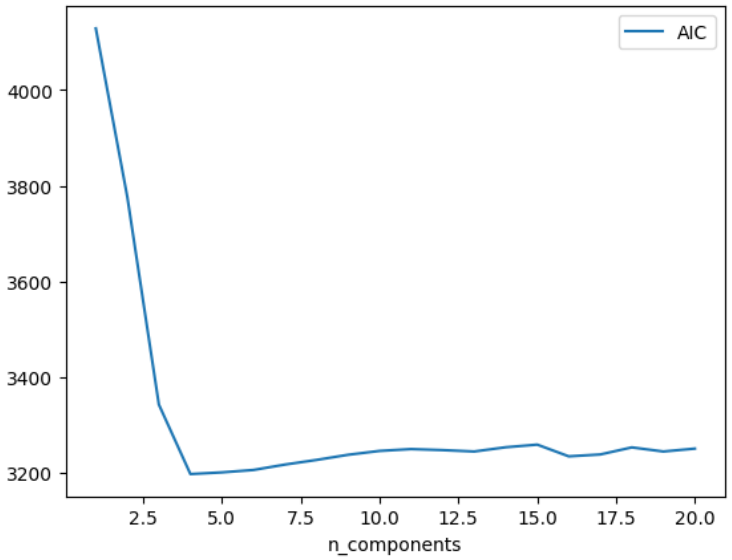
```
AIC: 3197.6596337289093
```

12.4.6. Tuning Hyperparameter

Untuk mendapatkan model terbaik, kita mungkin perlu melakukan beberapa tuning hyperparameter. Salah satu hyperparameter yang perlu kita tuning adalah jumlah komponen Gaussian. Kita bisa melakukan ini dengan melatih beberapa model dengan jumlah komponen yang berbeda, menghitung AIC untuk masing-masing, dan memilih model dengan AIC terendah.

```
n_components = np.arange(1, 21)
models = [GaussianMixture(n, covariance_type='full',
random_state=0).fit(X)
          for n in n_components]

plt.plot(n_components, [m.aic(X) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_components')
```

Output:



Perhatikan bahwa kita juga bisa mencoba tuning hyperparameter lain, seperti jenis covariance type (misalnya 'full', 'tied', 'diag', 'spherical'), dan parameter regulasi yang mengontrol derajat regularisasi yang diterapkan pada matriks kovariansi.

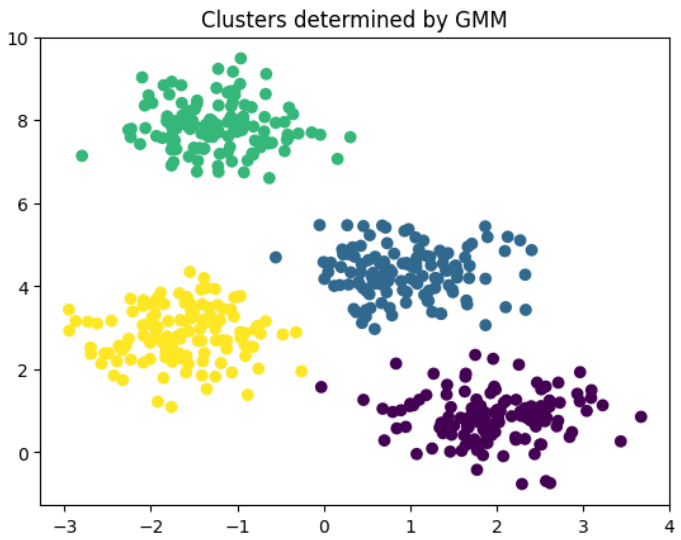
Untuk setiap perubahan yang kamu lakukan pada model, selalu pastikan untuk memeriksa bagaimana ini mempengaruhi performa model melalui AIC atau metode evaluasi lainnya. Tujuannya adalah untuk mencapai model yang seimbang antara kompleksitas (jumlah komponen dan jenis covariance type) dan kecocokan data.

12.4.7. Visualisasi Hasil

Setelah menemukan model optimal, kamu bisa memvisualisasikan hasil klastering. Salah satu cara adalah dengan menampilkan scatter plot dari data, di mana setiap titik diwarnai berdasarkan label klaster yang diberikan oleh model GMM.

```
gmm_optimal = models[np.argmin([m.aic(X) for m in models])]
labels = gmm_optimal.predict(X)

plt.scatter(X[:,0], X[:,1], c=labels)
plt.title('Clusters determined by GMM')
```

Output:



12.4.8. Predict Probabilitas

Dalam tutorial ini, kamu telah mempelajari cara mengimplementasikan Gaussian Mixture Model dalam Python. Ini termasuk membuat dan melatih model, melakukan prediksi, mengevaluasi model menggunakan AIC, dan melakukan tuning hyperparameter. Selain itu, kamu juga telah melihat bagaimana visualisasi hasil klastering.

Sebagai langkah selanjutnya, kamu bisa mencoba GMM pada dataset yang lebih kompleks dan nyata. Selain itu, kamu juga bisa mencoba metode klastering lain dan membandingkan hasilnya dengan GMM. Ingatlah bahwa tidak ada satu metode klastering yang paling baik untuk semua kasus, jadi penting untuk mengetahui berbagai metode dan kapan harus menggunakannya.

Perlu diingat juga bahwa GMM adalah model probabilistik, yang berarti selain memberikan label klaster, juga bisa memberikan probabilitas bahwa sampel tertentu milik klaster tertentu. Ini bisa menjadi informasi yang sangat berguna dalam beberapa kasus, seperti ketika kamu tidak yakin ke mana menempatkan sampel tertentu atau ketika ada sampel yang berada di antara dua klaster.

Untuk mendapatkan probabilitas ini, kamu bisa menggunakan fungsi predict_proba pada objek model GaussianMixture.

```
probs = gmm_optimal.predict_proba(X)
print(probs[:5].round(3))
```

Output:

```
[[1.    0.    0.    0.   ]
 [1.    0.    0.    0.   ]
 [0.    1.    0.    0.   ]
 [0.    1.    0.    0.   ]
 [0.998 0.002 0.    0.   ]]
```

Ini akan memberikan matriks di mana baris i adalah probabilitas bahwa sampel i milik setiap klaster. Perhatikan bahwa jumlah setiap baris adalah 1, karena ini adalah probabilitas dan harus berjumlah 1. Probabilitas ini bisa sangat membantu dalam memberikan gambaran yang lebih baik tentang bagaimana data kamu diorganisir dan bagaimana model GMM kamu memahaminya.

Bab 13. Evaluasi Model Unsupervised

13.1 Pengertian Evaluasi dan Validasi Model

Ketika kamu telah membangun suatu model machine learning, apakah itu supervised atau unsupervised, penting bagi kamu untuk mengetahui sejauh mana model tersebut bekerja dengan baik. Dalam bahasa machine learning, kita menyebut proses ini sebagai evaluasi dan validasi model. Proses ini penting agar kita bisa memahami kualitas dan performa model kita dan apakah model tersebut sudah cukup baik atau masih membutuhkan perbaikan. Dalam subbab ini, kita akan membahas secara mendalam tentang apa itu evaluasi dan validasi model, terutama dalam konteks unsupervised learning.

13.1.1 Evaluasi Model

Evaluasi model adalah proses di mana kita memeriksa seberapa baik model kita memprediksi hasil yang kita inginkan. Dalam konteks unsupervised learning, evaluasi model bisa menjadi tantangan, sebab tidak ada "label" yang benar sebagai acuan. Dalam supervised learning, kita bisa membandingkan prediksi model dengan label yang sebenarnya, dan mengukur sejauh mana prediksi tersebut cocok dengan label tersebut. Namun, dalam unsupervised learning, seperti clustering, kita tidak memiliki label ini, dan oleh karena itu, kita perlu menemukan cara lain untuk mengevaluasi performa model kita.

Beberapa metode evaluasi model unsupervised learning melibatkan penggunaan metrik internal yang berdasarkan struktur data dan model itu sendiri, seperti silhouette score, Calinski-Harabasz index, dan Davies-Bouldin index. Metode lainnya melibatkan penggunaan pengetahuan eksternal, seperti informasi tentang struktur yang diharapkan dari data, atau metrik evaluasi eksternal seperti Adjusted Rand Index, Mutual Information, dan Fowlkes-Mallows Index.

13.1.2 Validasi Model

Validasi model adalah proses di mana kita memeriksa seberapa baik model kita bekerja pada data baru yang belum pernah dilihat sebelumnya. Dalam konteks supervised learning, proses ini biasanya melibatkan pemisahan data menjadi set pelatihan dan set pengujian, di mana model dilatih pada set pelatihan dan kemudian diuji pada set pengujian. Hasilnya kemudian digunakan untuk memperkirakan seberapa baik model akan bekerja pada data baru.

Dalam konteks unsupervised learning, validasi model menjadi sedikit lebih rumit. Karena tidak ada label yang benar, sulit untuk mengetahui seberapa baik model bekerja pada data baru. Salah satu pendekatan adalah dengan menggunakan metrik internal untuk mengevaluasi sejauh mana struktur klaster dalam data baru mirip dengan struktur klaster dalam data pelatihan. Selain itu, jika ada informasi tambahan tentang data yang bisa digunakan sebagai "ground truth", ini juga bisa digunakan dalam proses validasi.

13.1.3 Pentingnya Evaluasi dan Validasi Model

Evaluasi dan validasi model sangat penting dalam machine learning. Tanpa evaluasi, kita tidak akan tahu seberapa baik model kita. Tanpa validasi, kita tidak akan tahu seberapa baik model kita akan bekerja pada data baru. Kedua proses ini membantu kita untuk memahami kekuatan dan kelemahan model kita, dan memberi kita informasi yang kita butuhkan untuk membuat perbaikan dan peningkatan. Tanpa evaluasi dan validasi yang tepat, risiko membuat keputusan berdasarkan model yang tidak akurat atau tidak efektif menjadi sangat tinggi.

13.1.4 Tantangan dalam Evaluasi dan Validasi Model Unsupervised Learning

Seperti yang sudah disebutkan sebelumnya, evaluasi dan validasi model unsupervised learning bisa jadi lebih menantang dibandingkan dengan supervised learning. Beberapa tantangan utama meliputi:

Tidak adanya label benar: Dalam supervised learning, kita bisa langsung membandingkan prediksi model dengan label yang sebenarnya untuk mengevaluasi performa model. Namun, dalam unsupervised learning, kita tidak memiliki label ini. Oleh karena itu, kita harus mengandalkan metrik lain untuk mengevaluasi performa model.

Kompleksitas struktur data: Dalam banyak kasus, data yang digunakan dalam unsupervised learning memiliki struktur yang kompleks dan tidak jelas. Ini membuat evaluasi model menjadi lebih sulit, karena tidak ada "jawaban yang benar" yang jelas.

Variabilitas dalam hasil: Unsupervised learning seringkali menghasilkan model yang berbeda tergantung pada inisialisasi dan urutan data. Ini berarti bahwa performa model dapat bervariasi secara signifikan, membuatnya lebih sulit untuk mengevaluasi.

Ketergantungan pada pengetahuan domain: Dalam banyak kasus, evaluasi model unsupervised learning sangat bergantung pada pengetahuan domain. Misalnya, jika kita melakukan clustering pada data genetik, pengetahuan tentang genetika mungkin diperlukan untuk benar-benar memahami dan mengevaluasi hasil.

Meskipun ada tantangan ini, evaluasi dan validasi model tetap sangat penting dalam unsupervised learning. Dengan memahami metrik dan teknik yang tersedia, serta kekuatan dan keterbatasan mereka, kita dapat membuat penilaian yang lebih baik tentang performa dan kegunaan model kita.

Pada bagian selanjutnya, kita akan membahas beberapa metrik evaluasi internal dan eksternal yang biasa digunakan dalam unsupervised learning, serta cara mereka digunakan dalam praktek. Selanjutnya, kita akan membahas beberapa strategi untuk memilih metrik evaluasi yang tepat untuk situasi tertentu. Terakhir, kita akan membahas bagaimana melakukan evaluasi dan validasi model dalam Python, dengan menggunakan beberapa contoh kode.

13.2. Metrik Evaluasi Internal

Dalam konteks pembelajaran tanpa pengawasan, metrik evaluasi internal adalah metrik yang tidak memerlukan pengetahuan tentang label kelas atau ground truth. Metrik ini mencoba untuk mengukur kualitas hasil clustering hanya berdasarkan data input. Dalam subbab ini, kita akan membahas tiga metrik evaluasi internal yang paling umum: Silhouette Score, Calinski-Harabasz Index, dan Davies-Bouldin Index.

13.2.1. Silhouette Score

Salah satu metrik evaluasi internal yang paling umum digunakan adalah Silhouette Score. Metrik ini mengukur sejauh mana setiap sampel mirip dengan cluster sendiri dibandingkan dengan cluster lain. Skor ini berkisar antara -1 dan +1, di mana nilai tinggi menunjukkan bahwa sampel tersebut cocok dengan baik dalam cluster sendiri dan buruk dalam cluster lain. Jika sebagian besar objek memiliki nilai tinggi, maka konfigurasi clustering dianggap tepat.

Untuk menghitung Silhouette Score untuk satu sampel, kita perlu menghitung dua nilai: a dan b. a adalah rata-rata jarak antara satu sampel dengan semua sampel lain dalam cluster yang sama, dan b adalah rata-rata jarak antara satu sampel dengan semua sampel dalam cluster terdekat lainnya. Silhouette Score dihitung sebagai (b - a) / max(a, b).

Berikut adalah contoh kode Python untuk menghitung Silhouette Score dengan menggunakan sklearn:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, _ = make_blobs(n_samples=500, n_features=2, centers=3, cluster_std=1,
random_state=42)

# Apply clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Compute Silhouette Score
labels = kmeans.labels_
silhouette_score = metrics.silhouette_score(X, labels,
metric='euclidean')

print('Silhouette Score: ', silhouette_score)
```

Output:

```
Silhouette Score:  0.8437565906781406
```

13.2.2. Calinski-Harabasz Index

Calinski-Harabasz Index, juga dikenal sebagai Variance Ratio Criterion, adalah metrik evaluasi lain yang digunakan untuk mengevaluasi model clustering. Metrik ini didefinisikan sebagai rasio antara dispersi antar-cluster dan dispersi intra-cluster.

Berbeda dengan Silhouette Score, nilai tinggi Calinski-Harabasz Index menunjukkan model clustering yang baik, sehingga kita perlu mencari model dengan nilai Calinski-Harabasz Index tertinggi, bukan terendah.

Berikut adalah contoh kode Python untuk menghitung Calinski-Harabasz Index dengan menggunakan sklearn:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, _ = make_blobs(n_samples=500, n_features=2, centers=3, cluster_std=1,
random_state=42)

# Apply clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Compute Calinski-Harabasz Index
labels = kmeans.labels_
ch_score = metrics.calinski_harabasz_score(X, labels)

print('Calinski-Harabasz Index: ', ch_score)
```

Output:

```
Calinski-Harabasz Index:  8309.607642024313
```

13.2.3. Davies-Bouldin Index

Davies-Bouldin Index adalah metrik evaluasi lainnya yang mengukur sejauh mana setiap cluster berbeda dengan cluster lain. Metrik ini didefinisikan sebagai rasio jarak antara dua cluster dan jumlah scatter dalam masing-masing cluster. Scatter dihitung sebagai rata-rata jarak antara setiap titik dalam cluster ke pusat cluster.

Berbeda dengan dua metrik sebelumnya, nilai Davies-Bouldin Index yang lebih rendah menunjukkan model clustering yang lebih baik. Dengan kata lain, kita ingin mencari model dengan Davies-Bouldin Index terendah.

Berikut adalah contoh kode Python untuk menghitung Davies-Bouldin Index dengan menggunakan sklearn:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, _ = make_blobs(n_samples=500, n_features=2, centers=3, cluster_std=1,
random_state=42)

# Apply clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Compute Davies-Bouldin Index
labels = kmeans.labels_
db_score = metrics.davies_bouldin_score(X, labels)

print('Davies-Bouldin Index: ', db_score)
```

Output:

```
Davies-Bouldin Index:  0.22152502252185613
```

Dalam subbab ini, kita telah membahas tiga metrik evaluasi internal yang berbeda: Silhouette Score, Calinski-Harabasz Index, dan Davies-Bouldin Index. Setiap metrik memiliki kelebihan dan kekurangan sendiri, dan pilihan metrik evaluasi yang tepat tergantung pada jenis data dan tujuan penelitian kamu. Namun, penting untuk diingat bahwa metrik evaluasi internal hanya memberikan panduan dan tidak memberikan kebenaran mutlak tentang kualitas model clustering.

Dalam prakteknya, penting untuk mempertimbangkan pengetahuan domain dan interpretasi model dalam konteks penelitian kamu. Metrik evaluasi internal bisa menjadi alat yang berguna untuk membandingkan model yang berbeda dan membantu kamu dalam menemukan model yang paling baik sesuai dengan data dan tujuan penelitian kamu.

13.3. Metrik Evaluasi Eksternal

Setelah membahas metrik evaluasi internal, mari kita pindah ke metrik evaluasi eksternal. Seperti namanya, metrik evaluasi eksternal digunakan ketika kita memiliki label

sebenarnya untuk data kita dan ingin membandingkan label tersebut dengan hasil clustering. Ada beberapa metrik evaluasi eksternal yang populer, termasuk Adjusted Rand Index, Mutual Information, dan Fowlkes-Mallows Index.

13.3.1. Adjusted Rand Index

Adjusted Rand Index (ARI) adalah metrik yang mengukur kesamaan antara dua pengelompokan, yang bisa digunakan untuk membandingkan hasil clustering dengan label sebenarnya. Nilai ARI berkisar antara -1 dan 1, di mana nilai 1 menunjukkan hasil clustering yang sempurna.

Berikut adalah contoh kode Python untuk menghitung ARI:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, y_true = make_blobs(n_samples=500, n_features=2, centers=3,
cluster_std=1, random_state=42)

# Apply clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
y_pred = kmeans.labels_

# Compute Adjusted Rand Index
ari_score = metrics.adjusted_rand_score(y_true, y_pred)

print('Adjusted Rand Index: ', ari_score)
```

Output:

```
Adjusted Rand Index:  1.0
```

13.3.2. Mutual Information

Mutual Information (MI) adalah metrik lain yang mengukur kesamaan antara dua pengelompokan. MI mengukur sejauh mana pengetahuan tentang pengelompokan satu membantu kita dalam memprediksi pengelompokan lainnya. Nilai MI berkisar antara 0 dan 1, di mana nilai 1 menunjukkan hasil clustering yang sempurna.

Berikut adalah contoh kode Python untuk menghitung MI:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

```
# Generate synthetic data
X, y_true = make_blobs(n_samples=500, n_features=2, centers=3,
cluster_std=1, random_state=42)

# Apply clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
y_pred = kmeans.labels_

# Compute Mutual Information
mi_score = metrics.adjusted_mutual_info_score(y_true, y_pred)

print('Mutual Information: ', mi_score)
```

Output:

```
Mutual Information:  1.0
```

13.3.3. Fowlkes-Mallows Index

Fowlkes-Mallows Index (FMI) adalah metrik yang mengukur kesamaan antara dua pengelompokan dengan menghitung rasio pasangan objek yang dikelompokkan bersama dalam kedua pengelompokan. Nilai FMI berkisar antara 0 dan 1, di mana nilai 1 menunjukkan hasil clustering yang sempurna.

Berikut adalah contoh kode Python untuk menghitung FMI:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, y_true = make_blobs(n_samples=500, n_features=2, centers=3,
cluster_std=1, random_state=42)

# Apply clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
y_pred = kmeans.labels_

# Compute Fowlkes-Mallows Index
fmi_score = metrics.fowlkes_mallows_score(y_true, y_pred)

print('Fowlkes-Mallows Index: ', fmi_score)
```

Output:

```
Fowlkes-Mallows Index:  1.0
```

Seperti yang kamu lihat, semua metrik ini membutuhkan label sebenarnya untuk melakukan evaluasi. Namun, dalam prakteknya, seringkali label sebenarnya tidak tersedia, terutama dalam kasus unsupervised learning. Dalam kasus seperti itu, metrik evaluasi internal menjadi sangat penting.

Selain itu, penting juga untuk dipahami bahwa tidak ada metrik evaluasi yang universal dan selalu memberikan hasil terbaik untuk semua kasus. Metrik yang paling tepat untuk digunakan tergantung pada data dan tujuan yang kamu miliki.

Misalnya, jika kamu memiliki data yang memiliki cluster dengan bentuk dan ukuran yang berbeda, Silhouette Score mungkin tidak akan bekerja dengan baik karena metrik ini berasumsi bahwa cluster ideal adalah yang compact dan terpisah dengan baik. Dalam kasus seperti ini, metrik lain seperti Davies-Bouldin Index mungkin lebih tepat.

Demikian pula, jika kamu memiliki label sebenarnya dan ingin membandingkan hasil clustering dengan label ini, kamu bisa menggunakan metrik evaluasi eksternal seperti Adjusted Rand Index atau Mutual Information. Namun, jika label sebenarnya tidak tersedia, kamu harus beralih ke metrik evaluasi internal.

Di sisi lain, jika kamu ingin mengevaluasi seberapa baik algoritma clustering kamu menangkap struktur data yang kompleks dan tidak teratur, kamu mungkin ingin menggunakan metrik yang lebih sophisticated seperti Calinski-Harabasz Index atau Fowlkes-Mallows Index.

Secara keseluruhan, penting untuk memahami kelebihan dan kekurangan setiap metrik dan memilih yang paling tepat berdasarkan konteks dan tujuan kamu.
Seperti yang kamu lihat, semua metrik ini membutuhkan label sebenarnya untuk melakukan evaluasi. Namun, dalam prakteknya, seringkali label sebenarnya tidak tersedia, terutama dalam kasus unsupervised learning. Dalam kasus seperti itu, metrik evaluasi internal menjadi sangat penting.

Selain itu, penting juga untuk dipahami bahwa tidak ada metrik evaluasi yang universal dan selalu memberikan hasil terbaik untuk semua kasus. Metrik yang paling tepat untuk digunakan tergantung pada data dan tujuan yang kamu miliki.

Misalnya, jika kamu memiliki data yang memiliki cluster dengan bentuk dan ukuran yang berbeda, Silhouette Score mungkin tidak akan bekerja dengan baik karena metrik ini berasumsi bahwa cluster ideal adalah yang compact dan terpisah dengan baik. Dalam kasus seperti ini, metrik lain seperti Davies-Bouldin Index mungkin lebih tepat.

Demikian pula, jika kamu memiliki label sebenarnya dan ingin membandingkan hasil clustering dengan label ini, kamu bisa menggunakan metrik evaluasi eksternal seperti Adjusted Rand Index atau Mutual Information. Namun, jika label sebenarnya tidak tersedia, kamu harus beralih ke metrik evaluasi internal.

Di sisi lain, jika kamu ingin mengevaluasi seberapa baik algoritma clustering kamu menangkap struktur data yang kompleks dan tidak teratur, kamu mungkin ingin menggunakan metrik yang lebih sophisticated seperti Calinski-Harabasz Index atau Fowlkes-Mallows Index.

Secara keseluruhan, penting untuk memahami kelebihan dan kekurangan setiap metrik dan memilih yang paling tepat berdasarkan konteks dan tujuan kamu.

13.4. Memilih Metrik yang Tepat

Sebelum kita melangkah lebih jauh ke dalam subbab ini, penting untuk kamu pahami bahwa tidak ada metrik evaluasi yang sempurna untuk setiap situasi. Memilih metrik yang tepat sangat bergantung pada konteks dan tujuan kamu. Karena itu, sangat penting untuk memahami kelebihan dan kekurangan setiap metrik.

Pertama, kamu harus mempertimbangkan apakah kamu memiliki akses ke label sebenarnya atau tidak. Jika kamu memilikinya, kamu bisa menggunakan metrik evaluasi eksternal seperti Adjusted Rand Index, Mutual Information, dan Fowlkes-Mallows Index. Metrik ini membandingkan hasil clustering dengan label sebenarnya dan memberikan gambaran tentang seberapa baik algoritma kamu menangkap struktur asli data.

Namun, dalam banyak kasus unsupervised learning, label sebenarnya tidak tersedia. Dalam situasi ini, kamu perlu mengandalkan metrik evaluasi internal seperti Silhouette Score, Calinski-Harabasz Index, dan Davies-Bouldin Index. Metrik ini tidak memerlukan label sebenarnya dan sebaliknya mengukur kualitas cluster berdasarkan properti data itu sendiri, seperti compactness dan separability cluster.

Setelah itu, kamu perlu mempertimbangkan sifat data dan cluster yang kamu miliki. Misalnya, jika kamu memiliki data dengan cluster yang berbeda bentuk dan ukuran, Silhouette Score mungkin tidak akan bekerja dengan baik karena metrik ini berasumsi bahwa cluster ideal adalah yang compact dan terpisah dengan baik. Dalam kasus seperti ini, metrik lain seperti Davies-Bouldin Index mungkin lebih tepat.

Demikian pula, jika kamu ingin mengevaluasi seberapa baik algoritma clustering kamu menangkap struktur data yang kompleks dan tidak teratur, kamu mungkin ingin menggunakan metrik yang lebih sophisticated seperti Calinski-Harabasz Index atau Fowlkes-Mallows Index.

Jadi, bagaimana cara memilih metrik yang tepat? Salah satu pendekatan adalah untuk mencoba beberapa metrik dan melihat mana yang paling konsisten dengan intuisi kamu tentang data. Misalnya, kamu bisa memvisualisasikan data dan melihat apakah hasil clustering yang dinilai baik oleh metrik tertentu juga tampak baik secara visual.

Berikut adalah contoh script python yang menunjukkan cara menggunakan beberapa metrik evaluasi dalam sklearn:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, y_true = make_blobs(n_samples=500, centers=4, random_state=0)

# Perform clustering
kmeans = KMeans(n_clusters=4, random_state=0).fit(X)

# Get cluster labels
y_pred = kmeans.labels_

# Compute Silhouette Score
sil_score = metrics.silhouette_score(X, y_pred)

# Compute Calinski-Harabasz Index
ch_score = metrics.calinski_harabasz_score(X, y_pred)

# Compute Davies-Bouldin Index
db_score = metrics.davies_bouldin_score(X, y_pred)

# Compute Adjusted Rand Index
ari_score = metrics.adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information
mi_score = metrics.adjusted_mutual_info_score(y_true, y_pred)

# Compute Fowlkes-Mallows Index
fmi_score = metrics.fowlkes_mallows_score(y_true, y_pred)

print('Silhouette Score: ', sil_score)
print('Calinski-Harabasz Index: ', ch_score)
print('Davies-Bouldin Index: ', db_score)
print('Adjusted Rand Index: ', ari_score)
print('Adjusted Mutual Information: ', mi_score)
print('Fowlkes-Mallows Index: ', fmi_score)
```

Output:

```
Silhouette Score:  0.5010253066728394
Calinski-Harabasz Index:  834.8936609880188
Davies-Bouldin Index:  0.6977467496491032
Adjusted Rand Index:  0.8475643367509819
Adjusted Mutual Information:  0.8094335016546887
Fowlkes-Mallows Index:  0.8854484227563895
```

Dalam kode di atas, kita pertama-tama menghasilkan data sintetis dengan empat pusat menggunakan fungsi `make_blobs` dari sklearn. Kemudian, kita melakukan clustering menggunakan algoritma KMeans dan mendapatkan label cluster dari model. Selanjutnya, kita menghitung dan mencetak nilai dari berbagai metrik evaluasi.

Namun, perlu diingat bahwa meski kita dapat menghitung semua metrik ini, bukan berarti semuanya relevan untuk masalah kita. Sebagai contoh, jika kita tidak memiliki akses ke label sebenarnya, maka tidak ada gunanya menghitung metrik evaluasi eksternal seperti Adjusted Rand Index, Mutual Information, atau Fowlkes-Mallows Index.

Oleh karena itu, salah satu kunci dalam memilih metrik yang tepat adalah memahami konteks dan tujuan kita. Untuk kasus clustering, kita mungkin tertarik untuk mengetahui apakah cluster yang dihasilkan kompak dan terpisah dengan baik (dalam hal ini, kita mungkin memilih Silhouette Score atau Davies-Bouldin Index), atau apakah struktur data asli dipertahankan dengan baik (dalam hal ini, kita mungkin memilih Calinski-Harabasz Index atau metrik evaluasi eksternal jika label sebenarnya tersedia).

Selain itu, penting juga untuk mempertimbangkan biaya komputasi. Beberapa metrik memerlukan perhitungan yang intensif dan mungkin tidak praktis untuk dataset yang sangat besar. Dalam hal ini, kamu mungkin perlu memilih metrik yang lebih sederhana atau mencoba pendekatan lain seperti sampling data.

Akhirnya, perlu diingat bahwa metrik evaluasi hanya alat bantu. Mereka bisa memberikan petunjuk tentang seberapa baik model kita, tetapi mereka bukanlah kebenaran mutlak. Selalu ada ruang untuk penilaian subjektif dan interpretasi dalam analisis data.

Bab 14. Optimasi dan Penyempurnaan Model

14.1 Hyperparameter Tuning

14.1.1 Grid Search

Grid Search adalah teknik yang digunakan untuk memperbaiki model dengan mencari optimal hyperparameters yang memberikan kinerja terbaik. Teknik ini secara sistematis membangun dan mengevaluasi model untuk setiap kombinasi hyperparameters yang ditentukan dalam grid parameter.

Misalkan kamu memiliki dua hyperparameter yang harus ditentukan untuk suatu model, dan kamu memiliki 3 pilihan untuk hyperparameter pertama dan 4 pilihan untuk yang kedua. Maka grid search akan membangun model untuk setiap kombinasi dari dua hyperparameters ini (3 * 4 = 12 model).

Berikut contoh penggunaan grid search untuk tuning hyperparameter pada algoritma K-Means:

```
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans

# Tentukan parameter grid
param_grid = {'n_clusters': range(2, 11), 'init': ['k-means++', 'random']}

# Inisialisasi model
kmeans = KMeans(random_state=0)

# Inisialisasi GridSearchCV
grid_search = GridSearchCV(kmeans, param_grid, cv=5,
scoring='adjusted_rand_score')

# Fit model dan cari hyperparameter optimal
grid_search.fit(X)

# Print hyperparameter terbaik
print('Best parameters: ', grid_search.best_params_)
```

Output:

```
Best parameters:  {'init': 'k-means++', 'n_clusters': 2}
```

Dalam contoh di atas, kita mencoba berbagai jumlah cluster dari 2 hingga 10 dan dua metode inisialisasi berbeda. GridSearchCV kemudian melatih model K-Means untuk setiap kombinasi ini dan mencari kombinasi yang memberikan Adjusted Rand Score terbaik.

14.1.2 Random Search

Berbeda dengan Grid Search yang mencoba semua kombinasi yang mungkin, Random Search memilih kombinasi hyperparameters secara acak. Walaupun terdengar tidak sistematis, tetapi Random Search bisa lebih efisien dibanding Grid Search, terutama jika jumlah hyperparameters yang perlu dituning sangat banyak.

Berikut contoh penggunaan random search untuk tuning hyperparameter pada algoritma K-Means:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.cluster import KMeans

# Tentukan parameter grid
param_dist = {'n_clusters': range(2, 11), 'init': ['k-means++',
'random']}

# Inisialisasi model
kmeans = KMeans(random_state=0)

# Inisialisasi RandomizedSearchCV
random_search = RandomizedSearchCV(kmeans, param_dist, cv=5,
scoring='adjusted_rand_score', n_iter=10, random_state=0)

# Fit model dan cari hyperparameter optimal
random_search.fit(X)

# Print hyperparameter terbaik
print('Best parameters: ', random_search.best_params_)
```

Output:

```
Best parameters:  {'n_clusters': 3, 'init': 'k-means++'}
```

Dalam contoh di atas, kita mengatur n_iter ke 10, yang berarti RandomizedSearchCV akan melatih model K-Means untuk 10 kombinasi hyperparameters yang dipilih secara acak.

Jadi, sebagai kesimpulan, Grid Search dan Random Search adalah teknik yang berbeda untuk mencari kombinasi hyperparameters yang optimal. Grid Search sistematis tetapi bisa sangat mahal secara komputasi. Random Search adalah pendekatan yang lebih efisien dan bisa lebih efektif jika ruang pencarian hyperparameters sangat besar.

Tetapi, perlu diingat bahwa tidak ada teknik tuning hyperparameters yang 'terbaik'. Pilihan antara Grid Search atau Random Search akan bergantung pada konteks spesifik, termasuk jumlah data, jumlah fitur, jumlah hyperparameters, dan sumber daya komputasi yang tersedia. Selain itu, penting juga untuk melakukan validasi silang untuk mendapatkan perkiraan yang tidak bias tentang kinerja model.

Hyperparameter tuning adalah bagian penting dari pembuatan model pembelajaran mesin dan dapat memiliki dampak besar pada kinerja model. Oleh karena itu, selalu disarankan untuk mencoba beberapa kombinasi hyperparameters untuk menemukan yang terbaik untuk dataset dan masalah spesifik kamu.

14.2. Feature Selection dan Extraction

Feature Selection dan Extraction adalah dua konsep yang sangat penting dalam pembelajaran mesin, terutama saat berurusan dengan data berdimensi tinggi. Banyak algoritma pembelajaran mesin dapat merasa kewalahan saat berhadapan dengan data berdimensi tinggi, yang bisa mengakibatkan model yang overfit atau underfit. Oleh karena itu, Feature Selection dan Extraction menjadi penting.

14.2.1. Feature Selection

Feature Selection adalah proses di mana kamu secara selektif memilih fitur yang paling relevan dan signifikan dari dataset kamu. Tujuannya adalah untuk mengurangi dimensi data, memperbaiki akurasi model, dan mengurangi kompleksitas komputasi.

Ada beberapa teknik yang umum digunakan dalam Feature Selection, diantaranya adalah metode filter, metode wrapper, dan metode embedded.

Metode Filter: Metode ini melibatkan pemilihan fitur berdasarkan metrik statistik tertentu seperti chi-square, correlation coefficient, dan lainnya. Fitur yang memiliki skor tertinggi dalam metrik ini biasanya dipilih.

Metode Wrapper: Metode ini menggunakan model pembelajaran mesin dan kinerjanya sebagai kriteria evaluasi. Algoritma wrapper seperti backward elimination, forward selection, dan recursive feature elimination digunakan untuk menentukan kombinasi fitur mana yang memberikan model terbaik.

Metode Embedded: Metode ini memadukan manfaat dari metode filter dan wrapper. Algoritma pembelajaran mesin yang mempunyai proses seleksi fitur sebagai bagian dari proses pembelajarannya termasuk dalam metode embedded. Contohnya adalah LASSO dan Ridge regression yang mempunyai mekanisme penalitas untuk mengurangi bobot fitur yang kurang penting.

14.2.2. Feature Extraction

Sementara itu, Feature Extraction adalah proses transformasi atau pemetaan data dari ruang dimensi tinggi ke ruang dimensi yang lebih rendah. Tujuan dari proses ini adalah untuk mengekstraksi informasi penting dari fitur dan mengurangi redundansi dalam data.

Principal Component Analysis (PCA) dan Linear Discriminant Analysis (LDA) adalah dua teknik Feature Extraction yang populer.

Principal Component Analysis (PCA): PCA adalah teknik statistik yang digunakan untuk mengurangi dimensi data dengan mempertahankan sebagian besar variasi yang penting. PCA mencari vektor-vektor eigen dari matriks kovariansi data yang bisa digunakan untuk memproyeksikan data ke ruang dimensi yang lebih rendah.

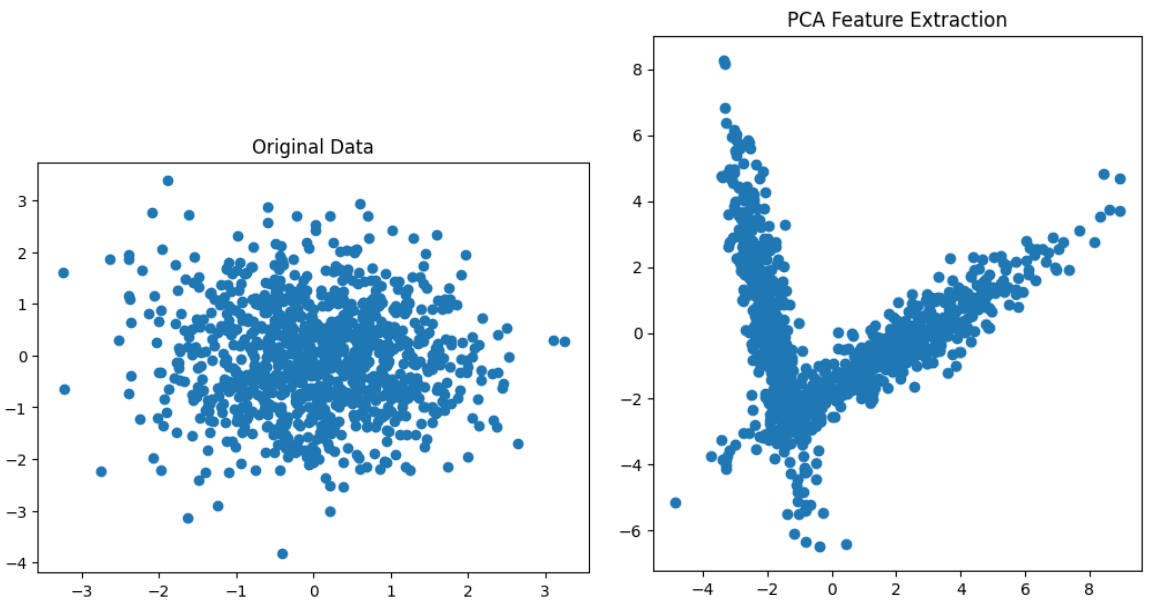
Linear Discriminant Analysis (LDA): LDA adalah teknik yang serupa dengan PCA, tetapi berfokus pada maksimisasi pemisahan antar kelas. LDA mencoba mencari kombinasi fitur yang memaksimalkan rasio jarak antar kelas terhadap jarak dalam kelas.

Berikut adalah contoh penggunaan PCA dan LDA untuk Feature Extraction pada dataset sintetis:

```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import make_classification

# Membuat dataset sintetis
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=2, n_redundant=10,
                           n_clusters_per_class=1, random_state=42)

# Visualisasi Original Data
plt.scatter(X[:, 0], X[:, 1])
plt.title('Original Data')

# Feature Extraction dengan PCA
pca = PCA(n_components=2)
X_pca = pca.fit(X).transform(X)

# Feature Extraction dengan LDA
lda = LinearDiscriminantAnalysis(n_components=1)
X_lda = lda.fit(X, y).transform(X)

# Visualisasi hasil Feature Extraction
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
```

```
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.title('PCA Feature Extraction')
```

Output:



Dalam contoh di atas, kita membuat dataset sintetis dengan 20 fitur, di mana hanya 2 fitur yang informatif dan 10 fitur yang redundan. Kemudian, kita menggunakan PCA dan LDA untuk mengekstraksi fitur dan mereduksi dimensi data menjadi 2 dimensi. Hasil Feature Extraction ditampilkan dalam plot scatter, di mana setiap titik merepresentasikan satu sampel dan warnanya merepresentasikan kelasnya.

Secara keseluruhan, proses Feature Selection dan Extraction sangat penting dalam pemrosesan data sebelum pembelajaran mesin. Dengan mengurangi dimensi data dan hanya mempertahankan fitur-fitur yang penting, kita bisa membuat model yang lebih efisien dan efektif.

14.3. Ensemble Methods untuk Unsupervised Learning

Dalam ranah pembelajaran mesin, istilah "ensemble" merujuk pada pendekatan di mana beberapa model atau algoritma digabungkan untuk memperbaiki kinerja prediksi. Ide dasarnya adalah dengan menggabungkan beberapa model, kita dapat mencapai model yang lebih baik dan lebih kuat.

Biasanya, metode ensemble sangat populer dalam konteks pembelajaran terawasi, seperti klasifikasi atau regresi. Metode-metode seperti bagging (seperti Random Forests), boosting

(seperti Gradient Boosting), dan stacking, semua merupakan contoh dari metode ensemble untuk pembelajaran terawasi.

Namun, bagaimana dengan unsupervised learning? Apakah metode ensemble juga dapat digunakan dalam konteks ini?

Ternyata, jawabannya adalah ya. Meskipun mungkin tidak sepopuler metode ensemble untuk pembelajaran terawasi, ada beberapa teknik yang dapat digunakan untuk menggabungkan beberapa model unsupervised learning. Contohnya termasuk ensemble clustering dan ensemble dimensionality reduction.

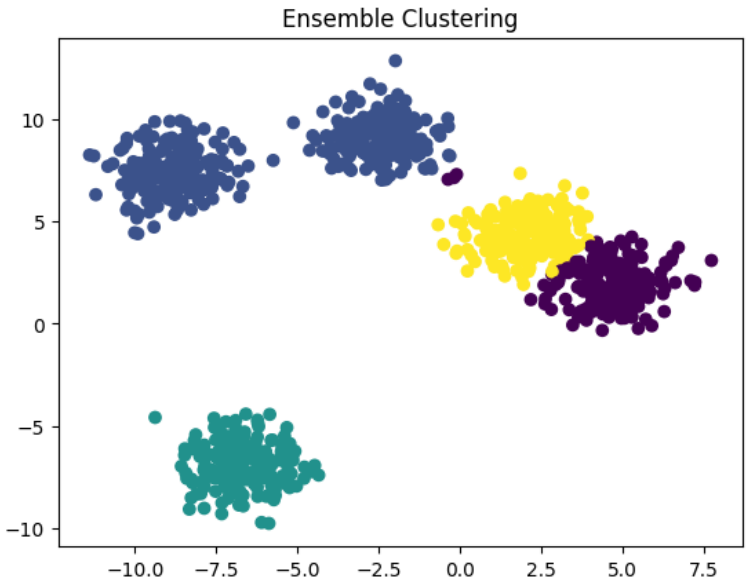
14.3.1. Ensemble Clustering

Ensemble clustering, juga dikenal sebagai consensus clustering, menggabungkan hasil dari beberapa algoritma clustering. Tujuannya adalah untuk mencapai hasil clustering yang lebih stabil dan akurat.

Berikut adalah contoh bagaimana ensemble clustering dapat dilakukan dengan Python:

```
import numpy as np
from scipy import stats

# Buat beberapa model KMeans dengan parameter yang berbeda
kmeans1 = KMeans(n_clusters=3, random_state=42).fit(X)
kmeans2 = KMeans(n_clusters=5, random_state=42).fit(X)
kmeans3 = KMeans(n_clusters=7, random_state=42).fit(X)

# Lakukan prediksi dengan setiap model
labels1 = kmeans1.predict(X)
labels2 = kmeans2.predict(X)
labels3 = kmeans3.predict(X)

# Gabungkan prediksi dari setiap model
labels_combined = np.vstack([labels1, labels2, labels3])

# Tentukan kluster yang paling sering muncul untuk setiap titik data
labels, _ = stats.mode(labels_combined, axis=0)

# Plot hasil clustering
plt.scatter(X[:, 0], X[:, 1], c=labels[0])
plt.title('Ensemble Clustering')
plt.show()
```

Output:



Perhatikan bahwa pendekatan ini hanya cocok jika kamu yakin bahwa semua model KMeans memiliki potensi untuk menghasilkan hasil yang baik. Jika tidak, mungkin lebih baik untuk memilih model terbaik berdasarkan beberapa metrik kualitas kluster, seperti Silhouette score atau Davies-Bouldin index.

Bab 15. Studi Kasus dan Contoh Aplikasi

15.1. Segmentasi Pelanggan

Segmentasi pelanggan adalah teknik membagi pelanggan ke dalam kelompok atau segmen berdasarkan karakteristik yang sama. Tujuan dari segmentasi pelanggan adalah untuk mengidentifikasi kelompok pelanggan dengan perilaku pembelian yang serupa sehingga dapat merancang strategi pemasaran yang lebih efektif dan personal untuk setiap segmen.

Tujuan

Dalam konteks ini, tujuan kita adalah untuk melakukan segmentasi pelanggan berdasarkan perilaku belanja mereka. Dengan memahami karakteristik unik dari setiap segmen, kita dapat merancang strategi pemasaran dan penjualan yang lebih tepat sasaran.

Pemilihan Variabel

Untuk melakukan segmentasi pelanggan, kita perlu memilih variabel yang paling relevan dengan perilaku belanja. Contoh variabel yang sering digunakan dalam segmentasi pelanggan adalah Recency (waktu terakhir seorang pelanggan melakukan pembelian), Frequency (berapa sering pelanggan melakukan pembelian), dan Monetary Value (jumlah total yang dihabiskan pelanggan). Variabel-variabel ini sering disebut sebagai RFM.

Mari kita buat data sintetis untuk contoh ini. Pertama, kita perlu mengimpor beberapa library:

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

Kemudian, kita akan membuat data sintetis menggunakan NumPy:

```
np.random.seed(0)
n_customers = 500
recency = np.random.randint(0, 20, n_customers)  # Waktu terakhir
seorang pelanggan melakukan pembelian (dalam hari)
frequency = np.random.randint(1, 10, n_customers)  # Berapa sering
pelanggan melakukan pembelian (dalam sebulan)
monetary_value = np.random.randint(5, 100, n_customers)  # Jumlah total
```

```
yang dihabiskan pelanggan (dalam dolar)

df = pd.DataFrame({'Recency': recency, 'Frequency': frequency,
'MonetaryValue': monetary_value})
```

Preprocessing Data

Pada langkah ini, kita perlu menyiapkan data sebelum melakukan clustering. Salah satu langkah penting dalam preprocessing adalah melakukan penskalaan fitur. Penskalaan fitur sangat penting dalam algoritma berbasis jarak seperti K-Means. Kita bisa menggunakan StandardScaler dari Scikit-learn untuk melakukan penskalaan fitur:

```
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

Pemilihan Metode Clustering

Dalam contoh ini, kita akan menggunakan algoritma K-Means untuk melakukan clustering. K-Means adalah algoritma clustering yang populer karena mudah dipahami dan implementasikan, tetapi juga efektif.

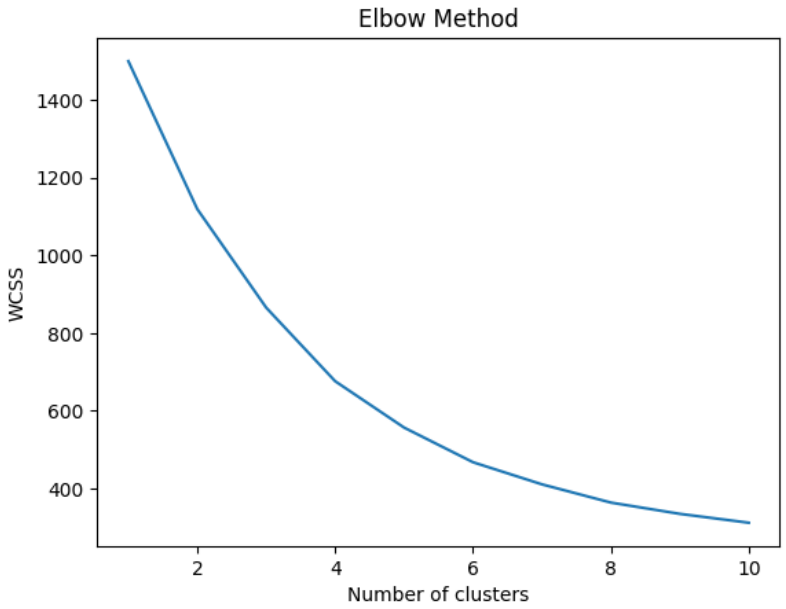
Penentuan Jumlah Cluster

Salah satu parameter penting dalam algoritma K-Means adalah jumlah cluster yang ingin kita buat. Ada beberapa teknik untuk menentukan jumlah cluster yang optimal, salah satunya adalah metode Elbow. Dalam metode ini, kita menjalankan algoritma K-Means dengan berbagai jumlah cluster dan menghitung Within-Cluster-Sum-of-Squares (WCSS) untuk setiap jumlah cluster. Titik 'elbow' atau 'siku' dalam plot WCSS adalah indikasi jumlah cluster yang optimal.

Berikut adalah script Python untuk menentukan jumlah cluster optimal dengan metode Elbow:

```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10, random_state=0)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Output:



Tidak ada jumlah cluster yang benar-benar valid berdasarkan metode Elbow. Pada kasus seperti ini, dan banyak kasus serupa lainnya, intusi dan keputusan dari sisi bisnis biasanya mempengaruhi keputusan penentuan jumlah cluster.

Pembentukan Model Clustering

Setelah menentukan jumlah cluster, langkah selanjutnya adalah melakukan pembentukan model clustering. Kita dapat menggunakan fungsi fit_predict untuk melakukan clustering dan mendapatkan label cluster untuk setiap pelanggan:

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10,
random_state=0)
cluster_labels = kmeans.fit_predict(df_scaled)
```

Validasi Model

Untuk mengukur seberapa baik model clustering kita, kita dapat menggunakan skor Silhouette. Skor Silhouette adalah metrik yang digunakan untuk menghitung sejauh mana objek dalam satu cluster mirip dengan objek dalam cluster yang sama dibandingkan dengan objek dalam cluster lain. Nilainya berkisar dari -1 hingga 1, di mana nilai yang lebih tinggi menunjukkan bahwa objek lebih mirip dengan objek dalam cluster yang sama dibandingkan dengan objek dalam cluster lain.

```
silhouette_avg = silhouette_score(df_scaled, cluster_labels)
print("The average silhouette score is :", silhouette_avg)
```

Output:

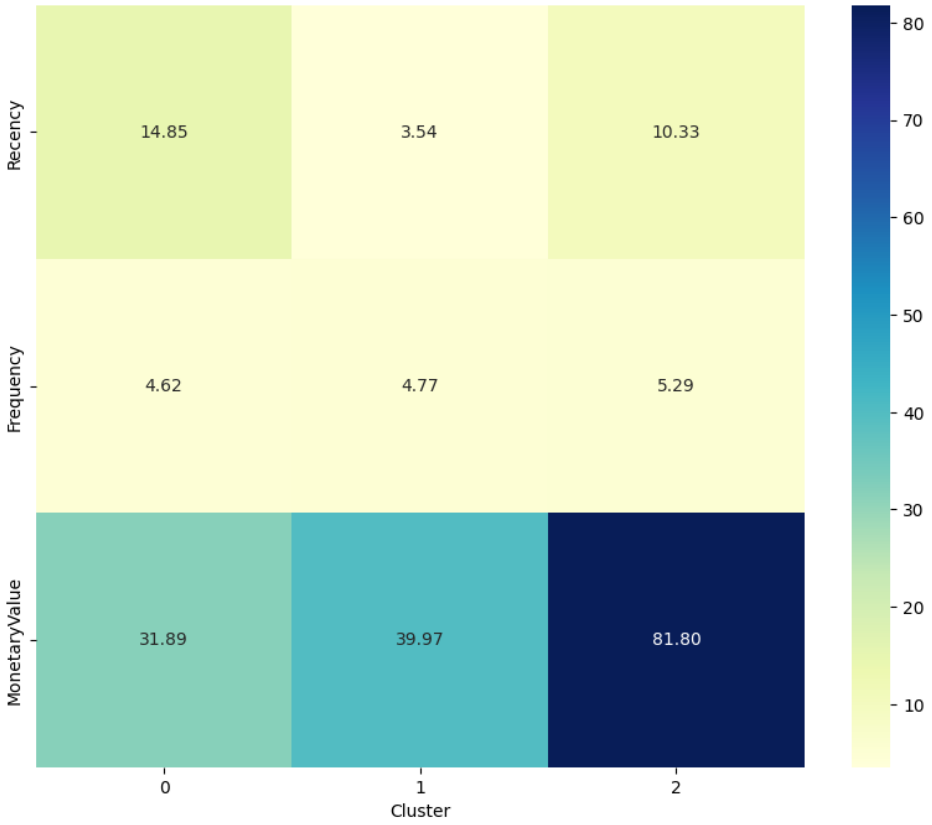
```
The average silhouette score is : 0.2580236482467085
```

Interpretasi Profile Cluster

Setelah melakukan clustering, kita perlu menginterpretasikan hasilnya. Salah satu cara untuk melakukannya adalah dengan melihat karakteristik rata-rata dari setiap cluster. Kita bisa menambahkan label cluster ke DataFrame asli dan menghitung nilai rata-rata untuk setiap variabel:

```
import seaborn as sns
df['Cluster'] = cluster_labels
clustered_data = df.groupby("Cluster").mean()
plt.figure(figsize=(10, 8))
sns.heatmap(clustered_data.T, cmap='YlGnBu', annot=True, fmt=".2f")
plt.show()
```

Output:



Dengan melihat profil rata-rata setiap cluster, kamu dapat menentukan strategi pemasaran yang paling efektif untuk setiap segmen pelanggan. Misalnya, jika salah satu segmen memiliki nilai Recency yang tinggi (artinya mereka belum berbelanja dalam waktu yang lama), kamu mungkin ingin menargetkan segmen ini dengan promosi khusus untuk mendorong mereka kembali berbelanja.

Demikianlah cara melakukan segmentasi pelanggan menggunakan teknik clustering. Harap diingat bahwa ini hanya contoh sederhana dan dalam prakteknya, kamu mungkin perlu

melakukan langkah-langkah tambahan seperti penghapusan outlier atau menggabungkan beberapa metode clustering untuk mendapatkan hasil yang lebih baik.

15.2 Deteksi Anomali

Deteksi anomali adalah proses identifikasi item atau peristiwa yang tidak sesuai dengan pola umum dalam dataset. Dalam konteks belajar mesin tanpa pengawasan, deteksi anomali sering digunakan untuk mengidentifikasi perilaku yang tidak biasa atau pengecualian dari pola umum. Contohnya termasuk deteksi penipuan kartu kredit, deteksi penyimpangan dalam log sistem, dan banyak lagi.

Tujuan

Tujuan dari studi kasus ini adalah untuk mendeteksi anomali dalam data transaksi kartu kredit. Dalam hal ini, anomali dapat diartikan sebagai transaksi yang tidak biasa yang mungkin menandakan adanya penipuan.

Pemilihan Variabel

Kita akan menggunakan data sintetis yang menggambarkan transaksi kartu kredit. Data tersebut akan mencakup fitur seperti jumlah transaksi, waktu transaksi, dan beberapa fitur lain yang dienkripsi untuk alasan keamanan. Variabel target adalah apakah transaksi tersebut adalah penipuan atau bukan.

Misalnya, kita bisa membuat DataFrame sintetis dengan pandas:

```
import pandas as pd
import numpy as np

np.random.seed(0)
n_samples = 5000
n_features = 5
n_outliers = 50

# Membuat data normal dengan distribusi Gaussian
data = np.random.randn(n_samples, n_features)

# Menambahkan outliers
outliers = np.random.uniform(low=-9, high=9, size=(n_outliers,
n_features))
data = np.concatenate([data, outliers], axis=0)

# Membuat DataFrame
df = pd.DataFrame(data, columns=['feature'+str(i+1) for i in
```

```
range(n_features)])
```

Preprocessing Data

Data preprocessing akan mencakup normalisasi fitur dan penghapusan outliers jika perlu. Karena kita sedang mencoba mendeteksi anomali, kita mungkin tidak ingin menghapus outliers sejak awal, tetapi kita mungkin ingin melakukan beberapa preprocessing lain seperti mengisi nilai yang hilang atau normalisasi fitur.

Pemilihan Metode Clustering

Untuk deteksi anomali, kita bisa menggunakan metode seperti Isolation Forest, DBSCAN, atau Local Outlier Factor (LOF). Dalam contoh ini, kita akan menggunakan Isolation Forest.

Penentuan Jumlah Cluster

Tidak seperti metode clustering lainnya, Isolation Forest tidak memerlukan penentuan jumlah cluster. Metode ini bekerja dengan cara 'memisahkan' setiap titik data dan mengukur seberapa mudah titik tersebut dipisahkan dari sisanya. Titik data yang mudah 'dipisahkan' dianggap sebagai anomali.

Pembentukan Model Clustering

Pada tahap ini, kita akan melatih model Isolation Forest dan menggunakannya untuk mendeteksi anomali.

```
from sklearn.ensemble import IsolationForest

# Melatih model
clf = IsolationForest(contamination=0.01)
clf.fit(df)

# Memprediksi outliers
outliers_prediction = clf.predict(df)
```

Validasi Model

Validasi model dalam konteks deteksi anomali bisa menjadi tantangan karena kita sering tidak memiliki label sebenarnya. Namun, jika kita memiliki label, kita dapat menggunakan metrik seperti F1-score, precision dan recall. Jika kita tidak memiliki label, kita bisa mencoba memvisualisasikan data dan melihat apakah anomali yang diidentifikasi oleh model masuk akal atau tidak.

Interpretasi Profile Cluster

Dalam konteks deteksi anomali, interpretasi biasanya melibatkan memahami karakteristik dari anomali yang telah dideteksi. Misalnya, kita dapat mencoba untuk mengetahui apa yang membuat suatu transaksi dianggap anomali oleh model. Apakah itu karena jumlah transaksi yang sangat besar? Atau karena transaksi dilakukan pada waktu yang tidak biasa?

```
# Menambahkan kolom 'anomaly' ke DataFrame
df['anomaly'] = outliers_prediction

# Menampilkan anomali
print(df[df['anomaly'] == -1])
```

Output:

	feature1	feature2	feature3	feature4	feature5	anomaly
137	-2.834555	2.116791	-1.610878	-0.035768	2.380745	-1
1316	2.720085	-1.613784	1.523005	-2.385016	0.916711	-1
5000	-6.194295	8.149244	3.822806	7.870434	4.178709	-1
5001	1.642643	0.142490	8.998537	-4.290621	-2.306720	-1
5002	6.378794	-0.412490	-6.180091	-0.142869	1.485558	-1
5003	3.590062	-7.237327	5.100940	-4.183644	-3.876341	-1
5004	3.845059	-5.729791	7.097605	2.777926	-3.922708	-1
5005	7.043198	-2.849784	-8.229296	8.256698	-4.588204	-1
5006	6.831731	1.523797	8.112228	6.004274	2.102384	-1

Dalam output di atas, semua baris dengan 'anomaly' == -1 dianggap anomali oleh model. Kamu bisa melihat nilai fitur untuk baris-baris ini dan mencoba memahami apa yang membuatnya dianggap anomali.

Secara keseluruhan, deteksi anomali adalah proses yang penting dalam banyak aplikasi, dari deteksi penipuan hingga pemantauan kesehatan sistem. Meskipun ini bisa menjadi tugas yang menantang, terutama dalam kasus di mana kita tidak memiliki label, ada banyak teknik dan metode yang dapat digunakan untuk mengatasi tantangan ini. Seperti yang ditunjukkan dalam studi kasus ini, belajar mesin tanpa pengawasan, seperti Isolation Forest, bisa menjadi alat yang sangat berguna dalam mendeteksi anomali.

15.3. Rekomendasi Produk

Rekomendasi produk adalah salah satu aplikasi paling populer dari pembelajaran mesin yang tidak diawasi, khususnya teknik pengelompokan. Contoh paling umum dari ini adalah sistem rekomendasi yang digunakan oleh platform e-commerce, seperti Amazon dan Netflix, untuk menyarankan produk atau film kepada pengguna berdasarkan preferensi mereka.

Tujuan

Tujuan utama dari kasus ini adalah untuk merancang sistem rekomendasi produk yang efektif dengan menggunakan teknik pembelajaran mesin yang tidak diawasi. Kita akan mencoba untuk memahami preferensi pelanggan dan menyarankan produk yang paling cocok untuk mereka berdasarkan preferensi tersebut.

Pemilihan Variabel

Untuk kasus ini, kita akan membutuhkan setidaknya dua jenis data: data tentang pelanggan (misalnya, usia, jenis kelamin, lokasi, dll.) dan data tentang interaksi mereka dengan produk (misalnya, produk yang dibeli, rating yang diberikan, dll.). Untuk tujuan contoh ini, kita akan membuat dataset sintetis dengan asumsi bahwa kita memiliki kedua jenis data ini.

Preprocessing Data

Preprocessing data adalah langkah penting dalam setiap pipeline pembelajaran mesin. Dalam kasus ini, kita mungkin perlu melakukan beberapa langkah preprocessing berikut:

```
# Mengimpor pustaka yang diperlukan
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Membuat data sintetis
np.random.seed(0)
n_customers = 500
n_products = 50

# Membuat data pelanggan
ages = np.random.normal(loc=35, scale=10, size=n_customers).astype(int)
genders = np.random.choice(['Male', 'Female'], size=n_customers)
locations = np.random.choice(['City', 'Suburb', 'Rural'],
size=n_customers)

customer_data = pd.DataFrame({
    'Age': ages,
    'Gender': genders,
    'Location': locations
})

# Membuat data produk
product_data = np.random.choice(n_products, size=(n_customers, 10))

# Normalisasi data numerik
```

```
scaler = StandardScaler()
customer_data['Age'] =
scaler.fit_transform(customer_data['Age'].values.reshape(-1, 1))

# One-hot encoding data kategorikal
customer_data = pd.get_dummies(customer_data, columns=['Gender',
'Location'])

customer_data.head()
```

Output:

	Age	Gender_Female	Gender_Male	Location_City	Location_Rural	Location_Suburb
0	1.779653	0	1	1	0	0
1	0.476832	1	0	0	1	0
2	0.977917	1	0	0	0	1
3	2.280738	1	0	0	1	0
4	1.879870	1	0	1	0	0

Pemilihan Metode Clustering

Untuk kasus ini, kita bisa menggunakan algoritma K-means atau algoritma pengelompokan lainnya. K-means adalah pilihan yang baik karena sifatnya yang sederhana dan efisien, tetapi juga bisa beradaptasi dengan baik dengan data besar.

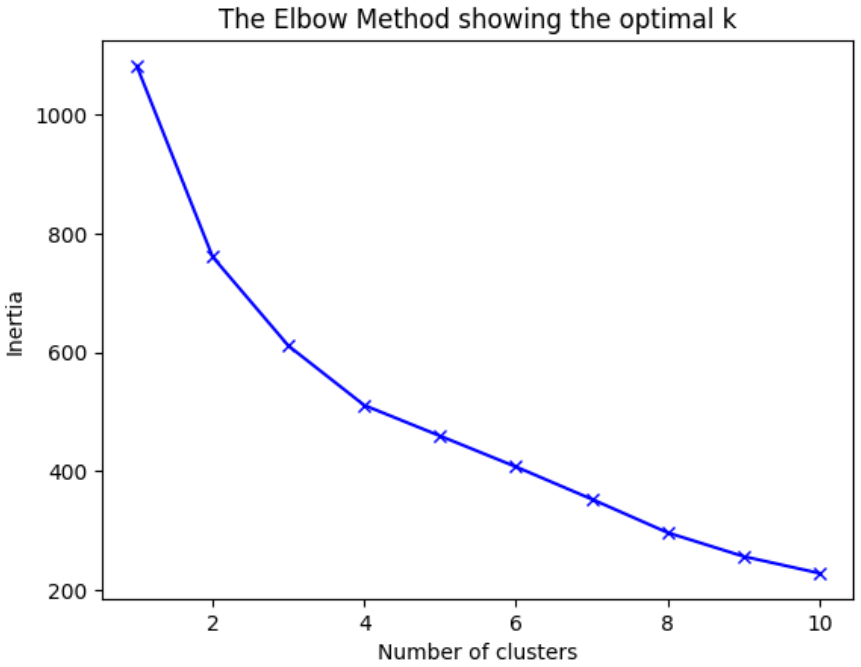
Penentuan Jumlah Cluster

Jumlah cluster dalam algoritma K-means perlu ditentukan sebelumnya. Salah satu metode untuk menentukan jumlah cluster yang optimal adalah metode Elbow. Ide dasarnya adalah menjalankan algoritma pengelompokan untuk berbagai jumlah cluster, dan kemudian memplot inersia (jumlah kuadrat jarak dari setiap titik ke pusat klusternya) sebagai fungsi dari jumlah cluster. Titik 'siku' dalam plot ini, di mana penurunan inersia mulai melambat, dapat digunakan sebagai indikasi jumlah cluster yang baik.

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Menghitung inersia untuk berbagai jumlah cluster
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(customer_data)
    inertia.append(kmeans.inertia_)

# Membuat plot Elbow
```

```
plt.plot(range(1, 11), inertia, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

Output:



Pembentukan Model Clustering

Setelah kita menentukan jumlah cluster yang optimal, kita bisa melatih model K-means dengan jumlah cluster tersebut.

```
# Melatih model K-means dengan jumlah cluster optimal
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters,
random_state=0).fit(customer_data)
```

Validasi Model

Untuk memvalidasi model, kita bisa menggunakan metrik internal seperti Silhouette Score, atau metrik eksternal jika kita memiliki label sebenarnya. Dalam kasus ini, karena kita tidak memiliki label sebenarnya, kita akan menggunakan Silhouette Score.
from sklearn.metrics import silhouette_score

```
# Menghitung Silhouette Score
labels = kmeans.labels_
silhouette_score = silhouette_score(customer_data, labels)

print(f'Silhouette Score: {silhouette_score}')
```

Output:

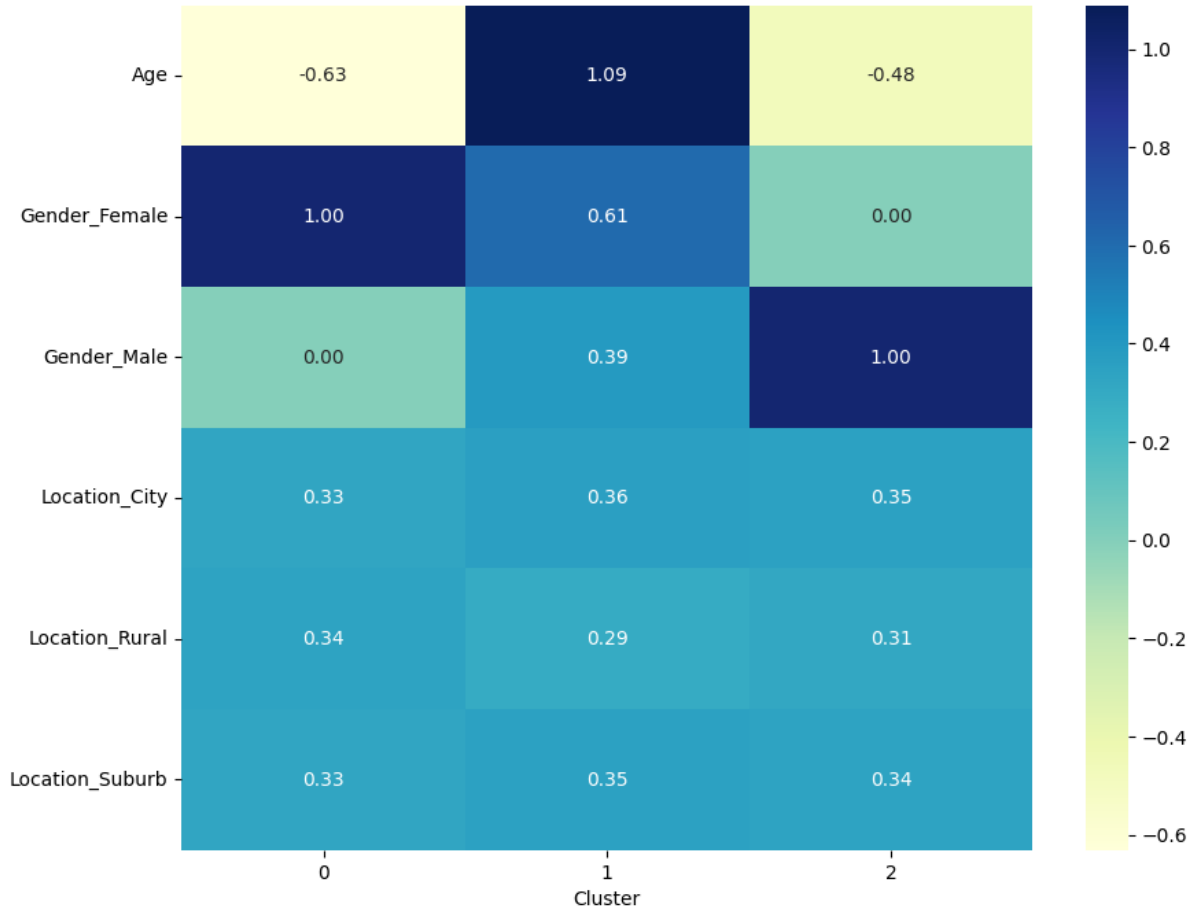
```
Silhouette Score: 0.26531059569827176
```

Interpretasi Profile Cluster

Akhirnya, setelah kita memiliki model pengelompokan yang dilatih, kita bisa melihat profil dari setiap cluster untuk mendapatkan wawasan tentang karakteristik pelanggan dalam setiap cluster. Ini bisa membantu kita dalam membuat rekomendasi yang lebih baik.

```
import seaborn as sns

# Menambahkan label cluster ke data
customer_data['Cluster'] = labels
clustered_data = customer_data.groupby('Cluster').mean()
plt.figure(figsize=(10, 8))
sns.heatmap(clustered_data.T, cmap='YlGnBu', annot=True, fmt=".2f")
plt.show()
```

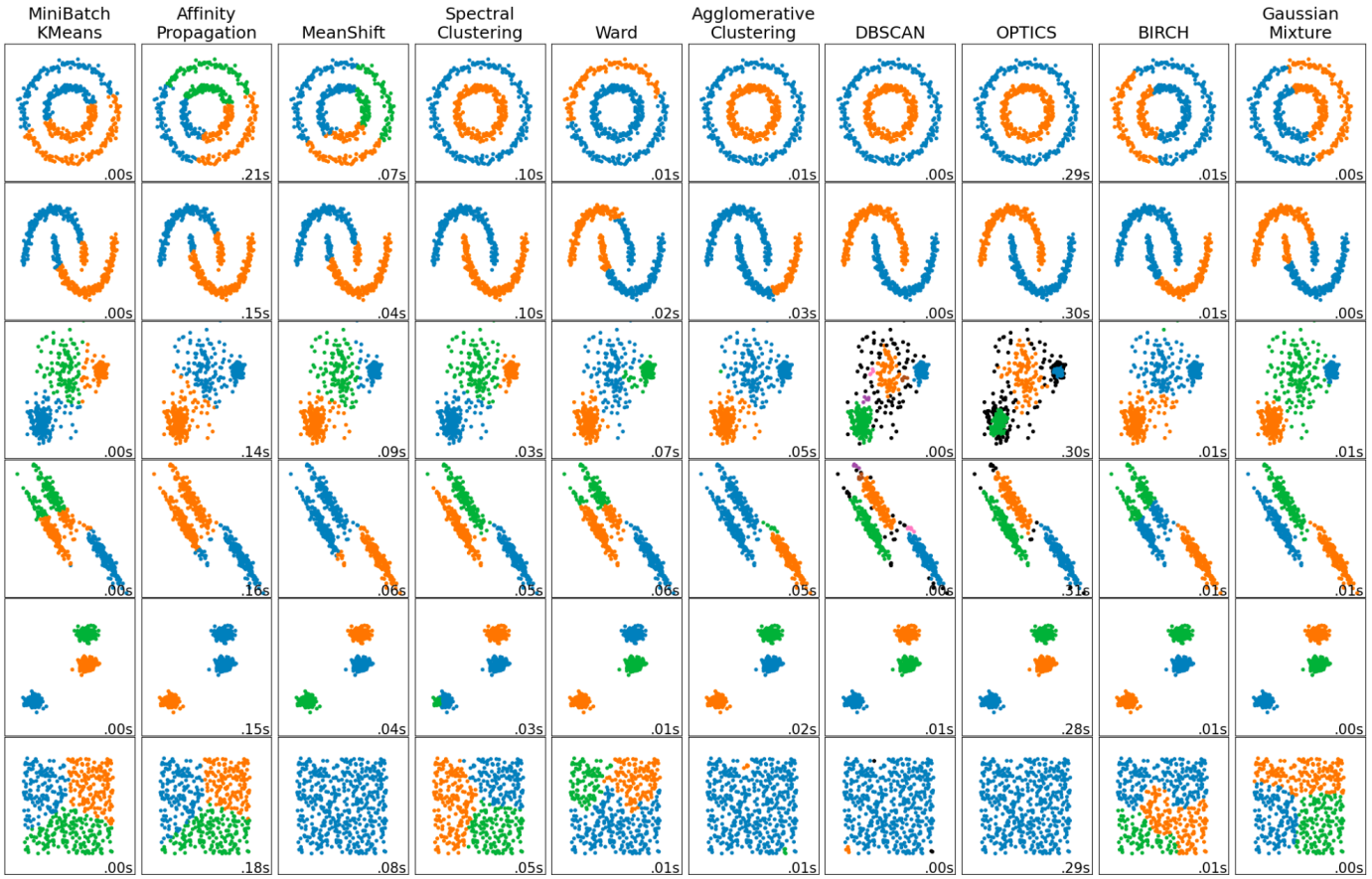
Output:



Pada akhirnya, dengan mengetahui profil setiap cluster, kita dapat menyusun strategi pemasaran yang lebih tepat untuk setiap segmen pelanggan, dan memberikan rekomendasi produk yang lebih personal dan efektif kepada pelanggan.

Overview Metode Clustering

source : https://scikit-learn.org/stable/modules/clustering.html



Nama Metode	Parameter	Skalabilitas	Kasus Penggunaan	Geometri (metrik yang digunakan)
K-Means	jumlah kluster	Sangat skalabel dengan n_samples yang sangat besar, n_clusters yang medium dengan kode MiniBatch	Umum, ukuran kluster yang merata, geometri datar, tidak terlalu banyak kluster, induktif	Jarak antara titik-titik
Affinity propagation	damping, preferensi sampel	Tidak skalabel dengan n_samples yang besar	Banyak kluster, ukuran kluster yang tidak merata, geometri non-datar, induktif	Jarak grafik (misalnya grafik tetangga terdekat)
Mean-shift	bandwidth	Tidak skalabel dengan n_samples yang besar	Banyak kluster, ukuran kluster yang tidak merata, geometri non-datar, induktif	Jarak antara titik-titik
Spectral clustering	jumlah kluster	n_samples sedang, n_clusters kecil	Sedikit kluster, ukuran kluster yang merata, geometri non-datar, transduktif	Jarak grafik (misalnya grafik tetangga terdekat)

Ward hierarchical clustering	jumlah kluster atau ambang jarak	n_samples dan n_clusters yang besar	Banyak kluster, mungkin ada batasan konektivitas, transduktif	Jarak antara titik-titik
Agglomerative clustering	jumlah kluster atau ambang jarak, jenis penghubung, jarak	n_samples dan n_clusters yang besar	Banyak kluster, mungkin ada batasan konektivitas, jarak non-Euclidean, transduktif	Jarak berpasangan apa pun
DBSCAN	ukuran tetangga	n_samples yang sangat besar, n_clusters yang medium	Geometri non-datar, ukuran kluster yang tidak merata, penghilangan outlier, transduktif	Jarak antara titik-titik terdekat
OPTICS	keanggotaan kluster minimum	n_samples yang sangat besar, n_clusters yang besar	Geometri non-datar, ukuran kluster yang tidak merata, kepadatan kluster yang beragam, penghilangan outlier, transduktif	Jarak antara titik-titik
Gaussian mixtures	banyak	Tidak skalabel	Geometri datar, baik untuk estimasi kepadatan, induktif	Jarak Mahalanobis ke pusat-pusat
BIRCH	faktor cabang, ambang, clusterer global opsional	n_clusters dan n_samples yang besar	Dataset besar, penghilangan outlier, reduksi data, induktif	Jarak Euclidean antara titik-titik
Bisecting K-Means	jumlah kluster	n_samples yang sangat besar, n_clusters yang medium	Umum, ukuran kluster yang merata, geometri datar, tidak ada kluster kosong, induktif, hirarkis	Jarak antara titik-titik

Terima Kasih