Machine Learning Cheatsheet: *Model Regresi*

(Konsep, Penerapan, Contoh Studi Kasus dengan Python Code)

Disusun oleh Tim Datasans

instagram : @datasans

medium: https://datasans.medium.com/

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 atau @datasans.book jika temanmu berminat agar ilmu yang bermanfaat bisa tersebar semakin luas.

Daftar Isi

Bab 1: Pengantar Machine Learning dan Regresi

1.1 Definisi Machine Learning

Machine Learning, atau Pembelajaran Mesin dalam bahasa Indonesia, adalah cabang ilmu komputer dan kecerdasan buatan yang memungkinkan komputer untuk belajar dari data dan memperbaiki kinerjanya seiring waktu tanpa harus diprogram secara eksplisit. Istilah ini pertama kali diperkenalkan oleh Arthur Samuel pada tahun 1959. Machine Learning merupakan salah satu metode yang digunakan dalam bidang data science untuk membangun model prediktif, menggali pola dalam data, dan mengoptimalkan proses pengambilan keputusan.

Machine Learning melibatkan berbagai teknik dan algoritma yang dapat digunakan untuk mengidentifikasi pola dalam data, mengekstrak informasi yang berguna, dan membuat keputusan atau prediksi berdasarkan data tersebut. Algoritma ini meliputi metode statistika, matematika, dan teknik-teknik optimasi yang digunakan untuk mengembangkan model yang mampu belajar dari data. Dalam konteks ini, "belajar" berarti memperoleh pengetahuan atau pemahaman tentang pola yang mendasari data melalui pengalaman atau instruksi.

Machine Learning dapat dikelompokkan menjadi beberapa jenis berdasarkan cara mereka belajar dari data, yaitu Supervised Learning, Unsupervised Learning, Semi-Supervised Learning, dan Reinforcement Learning. Dalam buku ini, kita akan fokus pada Supervised Learning dan khususnya, metode regresi.

Regresi adalah salah satu teknik yang digunakan dalam Machine Learning untuk memprediksi nilai kontinu dari suatu variabel berdasarkan data yang ada. Teknik ini melibatkan analisis hubungan antara variabel dependen (variabel yang ingin diprediksi) dan satu atau lebih variabel independen (variabel yang digunakan untuk memprediksi). Tujuan dari regresi adalah untuk mengembangkan model yang mampu menjelaskan hubungan antara variabel-variabel ini dan membuat prediksi yang akurat.

1.2 Supervised dan Unsupervised Learning

Supervised Learning adalah jenis pembelajaran di mana algoritma dilatih menggunakan data yang sudah diberi label, yang berarti bahwa setiap contoh data memiliki nilai target yang diketahui. Tujuan dari Supervised Learning adalah untuk mempelajari hubungan antara fitur (variabel independen) dan target (variabel dependen) sehingga algoritma dapat membuat prediksi yang akurat untuk data yang belum pernah dilihat sebelumnya. Contoh

teknik Supervised Learning meliputi regresi linier, regresi logistik, dan klasifikasi berbasis pohon keputusan.

Unsupervised Learning adalah jenis pembelajaran di mana algoritma dilatih menggunakan data yang tidak diberi label, yang berarti bahwa tidak ada nilai target yang diketahui untuk setiap contoh data. Tujuan dari Unsupervised Learning adalah untuk menemukan pola atau struktur yang mendasari data tanpa memerlukan informasi tentang target. Contoh teknik Unsupervised Learning meliputi pengelompokan (clustering), analisis komponen utama (principal component analysis), dan deteksi pencilan (outlier detection).

1.3 Regresi dalam Machine Learning

Dalam konteks Machine Learning, regresi adalah metode yang digunakan untuk memprediksi nilai kontinu dari suatu variabel, seperti harga, pendapatan, atau suhu. Teknik ini melibatkan analisis hubungan antara variabel dependen (variabel yang ingin diprediksi) dan satu atau lebih variabel independen (variabel yang digunakan untuk memprediksi). Regresi dapat dianggap sebagai salah satu bentuk Supervised Learning, di mana algoritma dilatih menggunakan data yang sudah diberi label dengan nilai target yang diketahui.

Model regresi mencoba meminimalkan kesalahan antara prediksi yang dibuat oleh model dan nilai sebenarnya dari variabel dependen. Kesalahan ini diukur menggunakan fungsi biaya atau fungsi kerugian, seperti Mean Squared Error (MSE) atau Mean Absolute Error (MAE). Selama proses pelatihan, model regresi mencoba meminimalkan fungsi biaya ini dengan mengoptimalkan parameter atau koefisien yang digunakan dalam model.

1.4 Jenis-jenis Regresi

Berbagai jenis regresi telah dikembangkan untuk mengatasi berbagai jenis masalah prediksi. Beberapa jenis regresi yang umum digunakan dalam Machine Learning meliputi:

Regresi Linier Sederhana: Model ini melibatkan hubungan linier antara variabel dependen dan satu variabel independen. Dalam regresi linier sederhana, kita mencoba menemukan garis terbaik yang sesuai dengan data, yang disebut garis regresi.

Regresi Linier Berganda: Model ini melibatkan hubungan linier antara variabel dependen dan beberapa variabel independen. Regresi linier berganda memperluas konsep regresi linier sederhana dengan mempertimbangkan lebih dari satu variabel independen dalam model.

Regresi Logistik: Model ini digunakan untuk memprediksi probabilitas suatu peristiwa terjadi berdasarkan satu atau lebih variabel independen. Regresi logistik sering digunakan untuk klasifikasi biner, di mana kita mencoba memprediksi apakah suatu contoh data termasuk dalam satu dari dua kelas yang berbeda.

Regresi Polynomial: Model ini melibatkan hubungan antara variabel dependen dan variabel independen yang dinyatakan sebagai polinomial derajat tertentu. Regresi polinomial memperluas konsep regresi linier dengan memungkinkan hubungan nonlinier antara variabel dependen dan variabel independen.

Regresi Ridge dan Lasso: Model ini merupakan variasi dari regresi linier yang menggunakan teknik regularisasi untuk menghindari overfitting dan meningkatkan kinerja model pada data yang belum pernah dilihat sebelumnya. Regresi Ridge menggunakan regularisasi L2, sedangkan Regresi Lasso menggunakan regularisasi L1.

Regresi Nonparametrik: Model ini tidak membuat asumsi tentang bentuk fungsi yang mendasari hubungan antara variabel dependen dan variabel independen. Regresi nonparametrik sering digunakan ketika data tidak mengikuti pola linier atau polinomial, dan meliputi metode seperti metode seperti regresi spline, regresi kernel, dan k-nearest neighbors.

Setiap jenis regresi memiliki kelebihan dan kekurangan tersendiri, serta asumsi yang mendasarinya. Pemilihan jenis regresi yang tepat tergantung pada sifat data yang digunakan dan tujuan analisis. Sebagai contoh, jika hubungan antara variabel dependen dan variabel independen tampak linier, maka regresi linier sederhana atau berganda mungkin merupakan pilihan yang baik. Namun, jika hubungan antara variabel tampak nonlinier atau lebih kompleks, maka regresi polinomial atau nonparametrik mungkin lebih sesuai.

Seiring berjalannya waktu, peneliti dan praktisi Machine Learning terus mengembangkan teknik regresi baru dan lebih canggih yang mampu menangani berbagai jenis masalah prediksi dengan lebih baik. Beberapa contoh teknik regresi yang lebih baru meliputi regresi support vector, regresi basis radial, dan regresi dengan jaringan saraf tiruan (neural networks).

Dalam buku ini, kita akan membahas konsep, penjelasan matematis, studi kasus, dan contoh Python script untuk berbagai jenis regresi, mulai dari yang paling sederhana hingga yang lebih canggih. Tujuan dari buku ini adalah untuk memberikan pemahaman yang mendalam tentang teknik regresi dalam Machine Learning, serta untuk memandu pembaca dalam mengaplikasikan teknik ini untuk memecahkan masalah prediksi nyata.

Bab 2: Konsep Dasar Regresi

2.1 Regresi Linier Sederhana

Regresi Linier Sederhana adalah metode statistik yang digunakan untuk memprediksi nilai kontinu dari suatu variabel dependen (variabel yang ingin kamu prediksi) berdasarkan satu variabel independen (variabel yang digunakan untuk memprediksi). Konsep di balik Regresi Linier Sederhana cukup sederhana: kamu mencoba menemukan garis terbaik yang dapat menjelaskan hubungan antara variabel dependen dan variabel independen. Garis ini disebut sebagai garis regresi atau garis best fit.

Sebagai contoh, misalkan kamu ingin memprediksi harga rumah berdasarkan luas tanahnya. Dalam hal ini, harga rumah adalah variabel dependen (yang ingin kamu prediksi), dan luas tanah adalah variabel independen (yang digunakan untuk memprediksi). Dengan menggunakan Regresi Linier Sederhana, kamu akan mencoba menemukan garis yang paling mewakili hubungan antara harga rumah dan luas tanah. Kemudian, kamu dapat menggunakan garis ini untuk memprediksi harga rumah berdasarkan luas tanah yang diberikan.

Dalam Regresi Linier Sederhana, garis regresi dinyatakan dalam bentuk persamaan linier berikut:

$$y = b0 + b1 * x$$

Di mana:

- y adalah variabel dependen (nilai yang ingin kamu prediksi)
- x adalah variabel independen (nilai yang digunakan untuk memprediksi)
- b0 adalah konstanta atau titik potong sumbu Y (intercept)
- b1 adalah koefisien regresi atau gradien garis

Koefisien b0 dan b1 merupakan parameter model yang perlu kita estimasi dari data. Estimasi ini biasanya dilakukan dengan menggunakan metode Kuadrat Terkecil (Least Squares), yang mencoba meminimalkan jumlah kuadrat kesalahan antara nilai prediksi dan nilai sebenarnya dari variabel dependen. Kesalahan ini juga dikenal sebagai residu.

Berikut adalah langkah-langkah yang terlibat dalam proses estimasi parameter regresi linier sederhana:

1. Menghitung rata-rata variabel dependen (y) dan variabel independen (x). Menghitung deviasi variabel dependen dan variabel independen dari rata-rata mereka (y - rata-rata(y) dan x - rata-rata(x)).
2. Menghitung kovarians antara variabel dependen dan variabel independen, yang merupakan jumlah dari perkalian deviasi yang dihitung pada langkah 2.

3. Menghitung varians dari variabel independen, yang merupakan jumlah kuadrat deviasi yang dihitung pada langkah 2.
4. Menghitung koefisien regresi (b1) dengan membagi kovarians yang dihitung pada langkah 3 dengan varians yang dihitung pada langkah 4.
5. Menghitung titik potong sumbu Y (b0) dengan mengurangkan hasil perkalian koefisien regresi (b1) dengan rata-rata variabel independen (x) dari rata-rata variabel dependen (y).
6. Setelah parameter regresi (b0 dan b1) diestimasi, kamu sekarang memiliki persamaan garis regresi yang dapat digunakan untuk memprediksi nilai variabel dependen berdasarkan nilai variabel independen. Namun, sebelum menggunakan model ini untuk membuat prediksi, penting untuk mengukur seberapa baik garis regresi sesuai dengan data.

Ada beberapa metrik yang dapat digunakan untuk mengukur kinerja model regresi linier sederhana, seperti Mean Squared Error (MSE), Mean Absolute Error (MAE), dan koefisien determinasi (R-squared).

MSE adalah metrik yang menghitung rata-rata kesalahan kuadrat antara nilai prediksi dan nilai sebenarnya dari variabel dependen. Semakin kecil MSE, semakin baik model regresi sesuai dengan data. Namun, MSE memiliki satu kelemahan: nilai-nilai kesalahan dikuadratkan, sehingga model lebih peka terhadap outlier atau kesalahan besar. Oleh karena itu, jika data kamu memiliki outlier, kamu mungkin ingin menggunakan metrik yang lebih robust, seperti MAE.

MAE adalah metrik yang menghitung rata-rata kesalahan absolut antara nilai prediksi dan nilai sebenarnya dari variabel dependen. Berbeda dengan MSE, MAE tidak memperhatikan kuadrat kesalahan, sehingga lebih toleran terhadap outlier. Namun, MAE mungkin kurang informatif daripada MSE jika kamu tertarik pada kesalahan yang lebih besar.

R-squared, atau koefisien determinasi, adalah metrik yang mengukur seberapa baik model regresi menjelaskan variabilitas dalam data. R-squared berkisar antara 0 dan 1, di mana 1 menunjukkan bahwa model menjelaskan 100% variabilitas dalam data, dan 0 menunjukkan bahwa model tidak menjelaskan variabilitas sama sekali. Secara umum, semakin tinggi R-squared, semakin baik model regresi sesuai dengan data. Namun, penting untuk dicatat bahwa R-squared tidak selalu menggambarkan seberapa baik model memprediksi nilai variabel dependen yang belum pernah dilihat sebelumnya. Oleh karena itu, selalu baik untuk mengevaluasi kinerja model regresi dengan menggunakan metrik lain juga, seperti MSE atau MAE.

2.2 Regresi Linier Berganda

Regresi Linier Berganda adalah perluasan dari Regresi Linier Sederhana yang memungkinkan kamu untuk memprediksi nilai variabel dependen berdasarkan lebih dari satu variabel independen. Dalam banyak kasus nyata, hubungan antara variabel dependen

dan independen tidak dapat dijelaskan hanya dengan satu variabel independen saja. Oleh karena itu, Regresi Linier Berganda menjadi penting untuk memodelkan hubungan yang lebih kompleks antara variabel dependen dan beberapa variabel independen.

Sebagai contoh, misalkan kamu ingin memprediksi harga rumah berdasarkan beberapa faktor, seperti luas tanah, jumlah kamar tidur, dan usia rumah. Dalam hal ini, harga rumah adalah variabel dependen, dan luas tanah, jumlah kamar tidur, dan usia rumah adalah variabel independen. Dengan menggunakan Regresi Linier Berganda, kamu akan mencoba menemukan permukaan (dalam kasus tiga dimensi) atau hyperplane (dalam kasus lebih dari tiga dimensi) yang paling mewakili hubungan antara variabel dependen dan variabel independen.

Dalam Regresi Linier Berganda, persamaan regresi dinyatakan dalam bentuk berikut:

$$y = b0 + b1 * x1 + b2 * x2 + ... + bn * xn$$

Di mana:

- y adalah variabel dependen (nilai yang ingin kamu prediksi)
- x1, x2, ..., xn adalah variabel independen (nilai yang digunakan untuk memprediksi)
- b0 adalah konstanta atau titik potong (intercept)
- b1, b2, ..., bn adalah koefisien regresi yang sesuai dengan variabel independen

Seperti dalam Regresi Linier Sederhana, koefisien regresi (b0, b1, ..., bn) merupakan parameter model yang perlu kita estimasi dari data. Estimasi ini biasanya dilakukan dengan menggunakan metode Kuadrat Terkecil (Least Squares), yang mencoba meminimalkan jumlah kuadrat kesalahan antara nilai prediksi dan nilai sebenarnya dari variabel dependen.

Berikut adalah langkah-langkah yang terlibat dalam proses estimasi parameter regresi linier berganda:

1. Menghitung matriks korelasi antara variabel dependen dan variabel independen.
2. Menghitung matriks kovarians antara variabel dependen dan variabel independen.
3. Menghitung matriks invers dari matriks kovarians yang dihitung pada langkah 2.
4. Menghitung koefisien regresi (b1, b2, ..., bn) dengan mengalikan matriks invers yang dihitung pada langkah 3 dengan matriks korelasi yang dihitung pada langkah 1.
5. Menghitung titik potong sumbu Y (b0) dengan mengurangkan hasil perkalian koefisien regresi (b1, b2, ..., bn) dengan rata-rata variabel independen yang relevan dari rata-rata variabel dependen.
6. Setelah parameter regresi (b0, b1, ..., bn) diestimasi, kamu sekarang memiliki persamaan regresi yang dapat digunakan untuk memprediksi nilai variabel dependen berdasarkan nilai variabel independen.

Namun, sebelum menggunakan model ini untuk membuat prediksi, penting untuk mengukur seberapa baik permukaan atau hyperplane regresi sesuai dengan data. Ada beberapa metrik yang dapat digunakan untuk mengukur kinerja model regresi linier

berganda, seperti Mean Squared Error (MSE), Mean Absolute Error (MAE), dan koefisien determinasi (R-squared).

2.3 Regresi Logistik

Regresi Logistik adalah salah satu teknik regresi yang digunakan ketika variabel dependen adalah kategorikal. Dalam kasus ini, model regresi logistik memprediksi probabilitas suatu hasil, daripada hasil langsung. Regresi logistik umumnya digunakan untuk memodelkan hubungan antara variabel dependen biner (dengan hanya dua kategori) dan satu atau lebih variabel independen. Misalnya, kamu mungkin ingin memprediksi apakah seorang pelanggan akan melakukan pembelian atau tidak berdasarkan usia, pendapatan, dan riwayat pembelian sebelumnya.

Dalam regresi logistik, model memprediksi probabilitas suatu hasil (misalnya, pembelian) dengan menggunakan fungsi logistik (juga dikenal sebagai fungsi sigmoid). Fungsi logistik memiliki bentuk kurva S dan menghasilkan nilai antara 0 dan 1, yang bisa diinterpretasikan sebagai probabilitas. Persamaan regresi logistik memiliki bentuk berikut:

$$P(Y = 1) = 1 / (1 + e^(- (b0 + b1 * x1 + b2 * x2 + ... + bn * xn)))$$

Di mana:

- P(Y = 1) adalah probabilitas bahwa variabel dependen (Y) sama dengan 1 (misalnya, pelanggan melakukan pembelian)
- x1, x2, ..., xn adalah variabel independen (nilai yang digunakan untuk memprediksi)
- b0 adalah konstanta atau titik potong (intercept)
- b1, b2, ..., bn adalah koefisien regresi yang sesuai dengan variabel independen
- e adalah bilangan Euler (sekitar 2.71828)

Seperti dalam Regresi Linier, koefisien regresi (b0, b1, ..., bn) merupakan parameter model yang perlu kita estimasi dari data. Estimasi ini biasanya dilakukan dengan menggunakan metode Maximum Likelihood Estimation (MLE), yang mencoba memaksimalkan kemungkinan (likelihood) dari parameter yang diberikan data. Berbeda dengan metode Kuadrat Terkecil yang digunakan dalam Regresi Linier, MLE dianggap lebih robust terhadap outlier dan memiliki interpretasi statistik yang lebih kuat.

Berikut adalah langkah-langkah yang terlibat dalam proses estimasi parameter regresi logistik:

1. Menginisialisasi koefisien regresi (b0, b1, ..., bn) dengan nilai sembarang atau nol.
2. Menghitung probabilitas hasil (P(Y = 1)) menggunakan persamaan regresi logistik.
3. Menghitung nilai likelihood dari parameter yang diberikan data (produk probabilitas hasil yang sesuai dengan data).
4. Menggunakan teknik optimasi, seperti Gradient Descent atau Newton-Raphson, untuk memperbarui koefisien regresi sehingga nilai likelihood maksimum tercapai.

5. Ulangi langkah 2 hingga 4 sampai konvergensi tercapai (perubahan dalam koefisien regresi menjadi sangat kecil).
6. Setelah parameter regresi (b0, b1, ..., bn) diestimasi, kamu sekarang memiliki persamaan regresi logistik yang dapat digunakan untuk memprediksi probabilitas suatu hasil berdasarkan nilai variabel independen.

Namun, sebelum menggunakan model ini untuk membuat prediksi, penting untuk mengukur seberapa baik model regresi logistik sesuai dengan data. Ada beberapa metrik yang dapat digunakan untuk mengukur kinerja model regresi logistik, seperti akurasi, presisi, recall, F1 score, dan Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC).

Akurasi adalah metrik yang menghitung persentase prediksi yang benar dari total prediksi. Akurasi umum digunakan untuk mengukur kinerja model, tetapi mungkin tidak selalu informatif, terutama jika data tidak seimbang (yaitu, satu kelas jauh lebih sering daripada kelas lain).

Presisi adalah metrik yang menghitung persentase prediksi positif yang benar dari total prediksi positif. Recall, atau sensitivitas, adalah metrik yang menghitung persentase prediksi positif yang benar dari total kasus positif sebenarnya. F1 score adalah rata-rata harmonik dari presisi dan recall, yang memberikan nilai tunggal yang mencerminkan keseimbangan antara presisi dan recall.

AUC-ROC adalah metrik yang mengukur seberapa baik model membedakan antara dua kelas. Nilai AUC-ROC berkisar antara 0 dan 1, di mana 1 menunjukkan model yang sempurna, dan 0,5 menunjukkan model yang tidak lebih baik daripada tebakan acak. Secara umum, semakin tinggi AUC-ROC, semakin baik model regresi logistik dalam membedakan antara kelas positif dan negatif.

2.4 Regresi Polinomial

Regresi Polinomial adalah jenis regresi yang digunakan ketika hubungan antara variabel dependen dan variabel independen tidak bisa dijelaskan dengan baik oleh model linier. Dalam regresi polinomial, model mencoba menangkap hubungan yang lebih kompleks dengan menambahkan polinomial berbagai derajat dari variabel independen. Regresi polinomial memberikan fleksibilitas lebih dalam memodelkan hubungan yang mungkin tidak linear antara variabel.

Misalkan kita memiliki satu variabel independen x dan variabel dependen y. Dalam regresi linier sederhana, model berbentuk:

$$y = b0 + b1 * x$$

Sedangkan dalam regresi polinomial derajat 2 (kuadratik), model akan berbentuk:

$$y = b0 + b1 * x + b2 * x^2$$

Demikian pula, dalam regresi polinomial derajat n, model akan berbentuk:

$$y = b0 + b1 * x + b2 * x^2 + ... + bn * x^n$$

Di mana b0, b1, ..., bn adalah koefisien yang perlu diestimasi dari data, dan n adalah derajat polinomial.

Regresi polinomial bisa diperluas untuk mengakomodasi lebih dari satu variabel independen dengan menambahkan kombinasi polinomial dari variabel independen. Misalnya, untuk regresi polinomial derajat 2 dengan dua variabel independen x1 dan x2, model akan berbentuk:

$$y = b0 + b1 * x1 + b2 * x2 + b3 * x1^2 + b4 * x1 * x2 + b5 * x2^2$$

Estimasi koefisien dalam regresi polinomial melibatkan metode yang sama dengan regresi linier, yaitu metode Kuadrat Terkecil (Least Squares). Untuk melakukan ini, kita dapat menggunakan teknik transformasi yang disebut "fitur polinomial" untuk mengubah variabel independen menjadi bentuk yang sesuai, dan kemudian menerapkan regresi linier pada fitur yang diubah. Langkah-langkah untuk mengestimasi koefisien regresi polinomial adalah sebagai berikut:

Mengubah variabel independen x menjadi fitur polinomial menggunakan fungsi yang sesuai (misalnya, dalam kasus polinomial derajat 2, kita akan menghitung x^2, dan dalam kasus polinomial derajat 3, kita akan menghitung x^2 dan x^3, dan seterusnya).
Menambahkan fitur polinomial ke dalam model regresi linier dan mengestimasi koefisien menggunakan metode Kuadrat Terkecil.
Salah satu tantangan dalam menggunakan regresi polinomial adalah memilih derajat polinomial yang tepat. Jika derajat polinomial terlalu rendah, model mungkin tidak cukup fleksibel untuk menangkap hubungan yang kompleks antara variabel dependen dan independen. Sebaliknya, jika derajat polinomial terlalu tinggi, model mungkin terlalu fleksibel dan mulai menangkap pola yang sebenarnya tidak ada dalam data (overfitting).

Untuk menghindari overfitting dan memilih derajat polinomial yang tepat, kamu bisa menggunakan teknik validasi silang (cross-validation). Validasi silang melibatkan membagi data menjadi beberapa bagian (biasanya 5 atau 10), menggunakan sebagian data untuk melatih model, dan sisanya untuk menguji kinerja model. Proses ini diulang untuk setiap bagian data dan metrik kinerja dihitung sebagai rata-rata dari semua percobaan. Kamu bisa mencoba berbagai derajat polinomial dan memilih derajat yang memberikan kinerja terbaik dalam validasi silang.

2.5 Regresi Ridge dan Lasso

Regresi Ridge dan Lasso adalah teknik regresi yang digunakan untuk mengatasi masalah overfitting dan multicollinearity dalam model regresi. Keduanya merupakan jenis regresi regularisasi, yang menggabungkan teknik pemulusan atau penalti dalam proses pengoptimalan untuk menghasilkan model yang lebih robust dan general.

Overfitting terjadi ketika model terlalu kompleks dan mencoba menangkap pola dalam data yang sebenarnya bukan merupakan hubungan yang mendasar antara variabel dependen dan independen. Akibatnya, model menjadi terlalu spesifik untuk data pelatihan dan memiliki kinerja buruk pada data yang tidak dikenal. Multicollinearity adalah keadaan di mana beberapa variabel independen dalam model regresi saling berkorelasi. Hal ini bisa menyebabkan estimasi koefisien menjadi tidak stabil dan sulit untuk diinterpretasi.

Untuk mengatasi masalah ini, Regresi Ridge dan Lasso menggunakan teknik regularisasi yang menambahkan penalti pada koefisien regresi. Penalti ini berfungsi untuk mendorong koefisien menuju nol, mengurangi kompleksitas model dan menghasilkan model yang lebih general dan stabil. Secara khusus, Regresi Ridge menggunakan penalti L2, sedangkan Regresi Lasso menggunakan penalti L1.

Dalam Regresi Ridge, penalti L2 diterapkan pada koefisien regresi:

minimalkan ($RSS + \lambda\Sigma\beta i$^2)

Di mana RSS adalah jumlah kuadrat residual, λ adalah parameter regularisasi, dan βi adalah koefisien regresi.

Dalam Regresi Lasso, penalti L1 diterapkan pada koefisien regresi:

minimalkan ($RSS + \lambda\Sigma|\beta i|$)

Di mana |βi| adalah nilai absolut koefisien regresi.

Parameter regularisasi λ berfungsi untuk mengontrol sejauh mana penalti diterapkan pada koefisien. Jika λ = 0, maka model menjadi model regresi linier biasa. Jika λ sangat besar, maka koefisien akan ditekan ke nol, dan model akan menjadi sangat sederhana. Pemilihan nilai λ yang tepat penting untuk mencapai keseimbangan yang baik antara bias dan varians.

Salah satu cara untuk memilih nilai λ yang tepat adalah dengan menggunakan validasi silang. Validasi silang melibatkan membagi data menjadi beberapa bagian (biasanya 5 atau 10), menggunakan sebagian data untuk melatih model, dan sisanya untuk menguji kinerja model. Proses ini diulang untuk setiap bagian data dan metrik kinerja dihitung sebagai rata-rata dari semua percobaan. Kamu bisa mencoba berbagai nilai λ dan memilih nilai yang memberikan kinerja terbaik dalam validasi silang.

Perbedaan utama antara Regresi Ridge dan Lasso adalah dalam cara penalti diterapkan pada koefisien. Penalti L1 dalam Regresi Lasso dapat menyebabkan beberapa koefisien menjadi nol, yang berarti variabel independen yang bersangkutan akan dieliminasi dari model. Ini menjadikan Regresi Lasso sebagai metode yang berguna untuk seleksi fitur, di mana kamu tertarik untuk mengidentifikasi variabel independen yang paling penting dalam memprediksi variabel dependen.

Di sisi lain, Regresi Ridge menggunakan penalti L2 yang mendorong koefisien menuju nol tetapi tidak pernah membuatnya benar-benar nol. Oleh karena itu, Regresi Ridge lebih cocok untuk situasi di mana kamu ingin mempertahankan semua variabel independen dalam model, tetapi mengurangi dampak variabel yang kurang penting.

Dalam beberapa kasus, kamu mungkin ingin menggunakan kombinasi dari Regresi Ridge dan Lasso untuk mencapai keseimbangan yang baik antara penalti L1 dan L2. Metode ini disebut Regresi Elastic Net, yang mencakup penalti L1 dan L2 dalam fungsi optimasi:

minimalkan ($RSS + \lambda 1 \Sigma |\beta i| + \lambda 2 \Sigma \beta i \hat{} 2$)

Di mana λ1 dan λ2 adalah parameter regularisasi untuk penalti L1 dan L2, secara berurutan.

Untuk mengimplementasikan Regresi Elastic Net dalam Python, kamu bisa menggunakan kelas ElasticNet dari scikit-learn dan mengikuti langkah-langkah yang sama seperti yang dijelaskan sebelumnya untuk Regresi Ridge dan Lasso.

2.6 Regresi Nonparametrik

Regresi Nonparametrik adalah teknik regresi yang fleksibel dan bisa digunakan untuk memodelkan hubungan yang sangat kompleks dan non-linear antara variabel dependen dan independen. Berbeda dengan metode regresi parametrik, seperti Regresi Linier dan Regresi Logistik, yang mengasumsikan bentuk tertentu dari hubungan antara variabel, Regresi Nonparametrik tidak mengasumsikan bentuk tertentu dan karenanya, bisa menyesuaikan diri dengan struktur yang lebih kompleks dalam data.

Keuntungan utama dari Regresi Nonparametrik adalah kemampuannya untuk menggambarkan hubungan yang tidak dapat dijelaskan oleh model parametrik sederhana. Namun, kelemahannya adalah bahwa mereka seringkali memerlukan lebih banyak data untuk melatih dan mungkin lebih sulit untuk diinterpretasi dibandingkan dengan model parametrik.

Berikut adalah beberapa metode Regresi Nonparametrik yang populer:

Regresi Spline: Regresi Spline adalah metode yang menggunakan fungsi polinomial yang halus untuk memodelkan hubungan antara variabel dependen dan independen. Fungsi

polinomial disambungkan pada titik-titik kontrol, yang disebut knot, untuk membentuk spline. Dengan menambahkan lebih banyak knot, kamu bisa meningkatkan fleksibilitas model dan memungkinkannya untuk menangkap pola yang lebih kompleks dalam data. Namun, terlalu banyak knot bisa menyebabkan overfitting.

Regresi Kernel: Regresi Kernel menggunakan fungsi kernel untuk memperkirakan hubungan antara variabel dependen dan independen. Fungsi kernel menentukan bagaimana pengaruh dari titik data individual menyebar ke titik data lainnya. Dengan memilih kernel yang tepat dan mengatur parameter bandwidth-nya, kamu bisa mengontrol sejauh mana model menyesuaikan diri dengan data. Beberapa kernel yang umum digunakan adalah Gaussian, Epanechnikov, dan Laplace.

Regresi K-Nearest Neighbors (KNN): Regresi KNN adalah metode yang memprediksi nilai variabel dependen untuk suatu titik data berdasarkan nilai rata-rata dari K titik data terdekat. K adalah parameter yang dapat kamu atur untuk mengontrol sejauh mana model menyesuaikan diri dengan data. Regresi KNN sangat intuitif dan mudah diimplementasikan, tetapi mungkin tidak efisien dalam hal komputasi jika dataset sangat besar.

Regresi Local Regression (LOESS): Regresi LOESS adalah metode yang memprediksi nilai variabel dependen untuk suatu titik data dengan melakukan regresi linier lokal pada titik data yang terdekat. Regresi linier lokal dilakukan dengan memberikan bobot lebih tinggi kepada titik data yang lebih dekat dengan titik yang diprediksi. Parameter bandwidth mengontrol jumlah titik data yang digunakan dalam regresi linier lokal dan sejauh mana model menyesuaikan diri dengan data.
Salah satu tantangan dalam menggunakan Regresi Nonparametrik adalah memilih parameter yang tepat, seperti jumlah knot dalam Regresi Spline, bandwidth dalam Regresi Kernel, atau jumlah tetangga dalam Regresi KNN. Salah satu cara untuk memilih parameter ini adalah dengan menggunakan validasi silang, seperti yang dijelaskan sebelumnya dalam konteks Regresi Ridge dan Lasso.

Selain itu, penting untuk mencatat bahwa Regresi Nonparametrik mungkin lebih rentan terhadap overfitting dibandingkan dengan metode regresi parametrik. Oleh karena itu, penting untuk mengawasi kinerja model pada data pengujian dan menggunakan teknik seperti validasi silang dan penalti regularisasi untuk menghindari overfitting.

Dalam beberapa kasus, kamu mungkin ingin menggabungkan metode Regresi Nonparametrik dengan metode regresi parametrik, seperti Regresi Linier atau Regresi Logistik, untuk mencapai keseimbangan yang baik antara fleksibilitas dan interpretasi. Metode yang menggabungkan elemen regresi parametrik dan nonparametrik disebut regresi semiparametrik.

Sebagai contoh, kamu bisa menggunakan Regresi Spline atau Regresi Kernel sebagai komponen nonparametrik dan Regresi Linier atau Regresi Logistik sebagai komponen

parametrik. Dengan menggabungkan kedua jenis metode, kamu bisa menghasilkan model yang lebih baik dalam memprediksi variabel dependen dan lebih mudah untuk diinterpretasi.

Dalam kesimpulan, Regresi Nonparametrik merupakan metode regresi yang fleksibel dan berguna untuk memodelkan hubungan yang kompleks dan non-linear antara variabel dependen dan independen. Meskipun mereka mungkin lebih sulit untuk diinterpretasi dan memerlukan lebih banyak data untuk melatih, mereka bisa menjadi alat yang sangat kuat dalam analisis data dan pembuatan model prediktif. Jangan ragu untuk bereksperimen dengan berbagai metode Regresi Nonparametrik dan parameter mereka untuk menemukan model terbaik yang sesuai dengan data dan kebutuhan analisis kamu.

Bab 3: Penjelasan Matematis Regresi

3.1 Fungsi Hipotesis

Dalam konteks regresi, fungsi hipotesis adalah sebuah model matematika yang digunakan untuk menggambarkan hubungan antara variabel dependen (variabel output) dan satu atau lebih variabel independen (variabel input). Fungsi hipotesis merupakan landasan dari metode regresi, yang bertujuan untuk menemukan hubungan terbaik antara variabel-variabel tersebut.

Fungsi hipotesis biasanya ditulis sebagai h(x), di mana x adalah vektor variabel independen, dan h(x) adalah nilai prediksi dari variabel dependen y. Fungsi hipotesis harus memenuhi beberapa kriteria agar dapat digunakan untuk regresi, seperti linearitas, aditivitas, dan homoskedastisitas. Berikut adalah penjelasan lebih lanjut tentang masing-masing kriteria:

Linearitas: Fungsi hipotesis harus linear dalam parameter yang akan diestimasi. Dalam kasus regresi linier, ini berarti bahwa h(x) harus memiliki bentuk seperti $h(x) = \theta_0 + \theta_1x_1 + \theta_2x_2 + ... + \theta_nx_n$, di mana θ_0, θ_1, ..., θ_n adalah parameter yang akan diestimasi, dan x_1, x_2, ..., x_n adalah variabel independen. Linearitas memungkinkan kita untuk menggunakan teknik-teknik matematika yang lebih sederhana, seperti metode kuadrat terkecil (Least Squares), untuk mengestimasi parameter.

Aditivitas: Fungsi hipotesis harus aditif, yang berarti bahwa efek dari masing-masing variabel independen pada variabel dependen harus dapat ditambahkan bersama. Dalam bentuk matematis, ini berarti bahwa $h(x) = h_1(x_1) + h_2(x_2) + ... + h_n(x_n)$, di mana $h_i(x_i)$ adalah fungsi yang menggambarkan efek dari variabel independen x_i pada variabel dependen y. Aditivitas memungkinkan kita untuk memisahkan efek dari masing-masing variabel independen dan memahami bagaimana mereka berkontribusi pada variabel dependen secara keseluruhan.

Homoskedastisitas: Fungsi hipotesis harus homoskedastik, yang berarti bahwa varians dari kesalahan (selisih antara nilai prediksi dan nilai sebenarnya) harus konstan untuk semua nilai x. Homoskedastisitas adalah asumsi penting dalam regresi, karena memungkinkan kita untuk mengukur ketidakpastian dalam estimasi parameter dan membuat inferensi tentang hubungan antara variabel dependen dan independen.

Untuk mengilustrasikan konsep fungsi hipotesis, kita akan menggunakan contoh regresi linier sederhana, yang melibatkan satu variabel independen x dan satu variabel dependen y. Dalam kasus ini, fungsi hipotesis memiliki bentuk $h(x) = \theta_0 + \theta_1x$, di mana θ_0 dan θ_1 adalah parameter yang akan diestimasi.

Tujuan dari regresi linier sederhana adalah untuk menemukan nilai-nilai θ_0 dan θ_1 yang menghasilkan fungsi hipotesis yang paling baik menggambarkan hubungan antara variabel x dan y. Dengan kata lain, kita ingin menemukan nilai-nilai parameter yang meminimalkan selisih antara nilai prediksi h(x) dan nilai sebenarnya y untuk sejumlah titik data. Untuk mencapai ini, kita perlu mengestimasi parameter θ_0 dan θ_1 berdasarkan data yang kita miliki.

Salah satu metode umum untuk mengestimasi parameter dalam regresi linier sederhana adalah metode kuadrat terkecil (Least Squares), yang berupaya meminimalkan jumlah kuadrat kesalahan (selisih antara nilai prediksi dan nilai sebenarnya) untuk semua titik data. Dalam notasi matematika, tujuan dari metode kuadrat terkecil adalah:

$$min(\theta_0,\ \theta_1)\ \Sigma_i\ (h(x_i)\ -\ y_i)^2$$

di mana Σ_i adalah jumlah dari i=1 sampai n, dengan n adalah jumlah titik data, dan $(h(x_i) - y_i)^2$ adalah kuadrat kesalahan untuk titik data ke-i.

Untuk menemukan nilai-nilai θ_0 dan θ_1 yang meminimalkan jumlah kuadrat kesalahan, kita dapat menggunakan teknik-teknik optimasi matematika, seperti turunan parsial dan metode gradient descent. Setelah kita menemukan nilai optimal untuk parameter ini, kita dapat menggunakan fungsi hipotesis yang dihasilkan untuk membuat prediksi tentang variabel dependen berdasarkan nilai-nilai variabel independen.

Selain regresi linier sederhana, ada banyak jenis regresi lain yang menggunakan fungsi hipotesis yang berbeda, seperti regresi linier berganda, regresi logistik, regresi polinomial, dan regresi nonparametrik. Meskipun bentuk matematika dari fungsi hipotesis mungkin berbeda untuk setiap jenis regresi, konsep dasar yang sama berlaku: kita ingin menemukan model yang paling baik menggambarkan hubungan antara variabel dependen dan independen dengan mengestimasi parameter yang relevan.

Dalam konteks regresi yang lebih umum, kita mungkin memiliki beberapa variabel independen dan/atau variasi yang lebih kompleks dari fungsi hipotesis. Namun, prinsip-prinsip dasar yang telah dijelaskan di atas tetap sama. Dalam setiap kasus, kita ingin mengidentifikasi model yang paling baik menggambarkan hubungan antara variabel-variabel tersebut dengan meminimalkan kesalahan dalam prediksi kita.

Secara keseluruhan, fungsi hipotesis merupakan konsep kunci dalam regresi dan merupakan titik awal untuk memahami bagaimana model regresi bekerja. Dengan menggali lebih dalam ke dalam berbagai jenis fungsi hipotesis dan teknik yang digunakan untuk mengestimasi parameter, kamu akan menjadi lebih terampil dalam menganalisis data dan membuat model prediktif yang akurat dan efisien. Selanjutnya, kita akan membahas konsep-konsep penting lainnya dalam regresi, seperti fungsi biaya (cost function), fungsi kerugian (loss function), metode optimasi, evaluasi model regresi, dan validasi silang.

3.2 Fungsi Biaya (Cost Function)

Fungsi biaya (cost function) merupakan salah satu konsep kunci dalam pemodelan regresi yang digunakan untuk mengukur seberapa baik model regresi yang dibangun dapat merepresentasikan hubungan antara variabel dependen dan variabel independen. Secara intuisi, fungsi biaya menghitung "biaya" dari kesalahan yang dibuat oleh model dalam memprediksi nilai variabel dependen berdasarkan nilai variabel independen. Tujuan utama dalam pemodelan regresi adalah untuk meminimalkan biaya ini, sehingga model yang dihasilkan memiliki kesalahan prediksi yang paling kecil.

Dalam konteks regresi linier, fungsi biaya yang umum digunakan adalah Mean Squared Error (MSE), yang merupakan rata-rata dari kuadrat kesalahan (selisih antara nilai prediksi dan nilai sebenarnya) untuk semua titik data. Secara matematis, MSE didefinisikan sebagai:

$$MSE(\theta) = (1/n)\,\Sigma_i\,(h(x_i) - y_i)^2$$

di mana n adalah jumlah titik data, $h(x_i)$ adalah nilai prediksi dari model regresi untuk titik data ke-i, y_i adalah nilai sebenarnya dari variabel dependen untuk titik data ke-i, dan θ adalah vektor parameter yang akan diestimasi.

Fungsi biaya seperti MSE berguna karena memberikan metrik yang kuantitatif untuk mengevaluasi seberapa baik model regresi bekerja dan memungkinkan kita untuk membandingkan kinerja antara model yang berbeda. Dengan meminimalkan fungsi biaya, kita dapat menemukan nilai-nilai parameter yang menghasilkan model regresi terbaik yang sesuai dengan data yang kita miliki.

Berikut adalah langkah-langkah yang umum dilakukan dalam proses pemodelan regresi menggunakan fungsi biaya:

1. Tentukan model regresi: Pilih jenis regresi yang akan digunakan (misalnya, regresi linier sederhana, regresi linier berganda, regresi logistik, dll.) dan tentukan bentuk matematika dari fungsi hipotesis yang sesuai.
2. Hitung fungsi biaya: Gunakan data yang tersedia untuk menghitung nilai dari fungsi biaya (misalnya, MSE) dengan menggantikan nilai-nilai variabel dependen dan independen ke dalam rumus.
3. Minimalkan fungsi biaya: Temukan nilai-nilai parameter yang meminimalkan fungsi biaya menggunakan teknik optimasi matematika, seperti metode gradient descent atau metode kuadrat terkecil.
4. Evaluasi model: Gunakan nilai-nilai parameter yang ditemukan untuk menghitung nilai prediksi dari variabel dependen menggunakan fungsi hipotesis. Bandingkan nilai prediksi ini dengan nilai sebenarnya untuk mengukur kinerja model.
5. Lakukan validasi silang: Gunakan teknik validasi silang, seperti k-fold cross-validation, untuk memastikan bahwa model yang dihasilkan memiliki kinerja yang baik pada data yang belum pernah dilihat sebelumnya.

Beberapa catatan penting tentang fungsi biaya:

Fungsi biaya harus konveks dan memiliki gradien yang dapat dihitung, agar teknik-teknik optimasi seperti gradient descent dapat digunakan dengan sukses. Konveksitas memastikan bahwa ada solusi optimal yang unik untuk masalah minimasi, dan gradien yang dapat dihitung memungkinkan kita untuk mengikuti arah tercepat untuk mencapai titik optimal tersebut.

Terkadang, kita mungkin ingin menambahkan istilah regularisasi ke fungsi biaya untuk menghindari overfitting dan menjaga kompleksitas model. Regularisasi membantu mengurangi beban parameter dalam model dan mencegah model terlalu menyesuaikan diri dengan data latih, sehingga meningkatkan kinerja pada data baru. Contoh teknik regularisasi yang umum digunakan dalam regresi adalah regresi Ridge dan Lasso.

Pemilihan fungsi biaya yang tepat sangat penting dalam pemodelan regresi. Fungsi biaya harus mencerminkan karakteristik data dan masalah yang ingin diselesaikan. Misalnya, jika data memiliki banyak pencilan, mungkin lebih baik menggunakan fungsi biaya yang lebih toleran terhadap pencilan, seperti Mean Absolute Error (MAE) atau Huber loss, daripada MSE.

Terkadang, fungsi biaya mungkin tidak cukup untuk mengevaluasi kinerja model secara menyeluruh. Dalam kasus seperti itu, kita mungkin ingin menggunakan metrik evaluasi tambahan, seperti koefisien determinasi (R^2), Mean Absolute Percentage Error (MAPE), atau F1-score, tergantung pada konteks masalah yang dihadapi.

Secara keseluruhan, fungsi biaya merupakan konsep kunci dalam pemodelan regresi yang memungkinkan kita untuk mengevaluasi seberapa baik model regresi bekerja dan menemukan parameter yang optimal untuk menghasilkan model terbaik. Dengan memahami bagaimana fungsi biaya bekerja dan bagaimana menggunakannya dalam konteks regresi yang berbeda, kamu akan menjadi lebih terampil dalam menganalisis data dan membuat model prediktif yang akurat dan efisien.

Selanjutnya, kita akan membahas konsep-konsep penting lainnya dalam regresi, seperti fungsi kerugian (loss function), metode optimasi, evaluasi model regresi, dan validasi silang. Memahami topik-topik ini akan membantu kamu menggali lebih dalam ke dalam berbagai teknik regresi dan menjadi lebih mahir dalam pemodelan data dan analisis prediktif.

3.3 Fungsi Kerugian (Loss Function)

Fungsi kerugian (loss function) adalah konsep yang erat kaitannya dengan fungsi biaya dan merupakan komponen penting dalam pemodelan regresi. Sementara fungsi biaya mengukur kesalahan secara keseluruhan pada semua titik data dalam set data, fungsi

kerugian mengukur kesalahan individu pada setiap titik data. Oleh karena itu, fungsi biaya pada dasarnya adalah agregasi dari fungsi kerugian pada setiap titik data.

Fungsi kerugian digunakan untuk menghitung seberapa besar kesalahan antara nilai prediksi yang dihasilkan oleh model dan nilai sebenarnya dari variabel dependen. Dengan meminimalkan fungsi kerugian, kita dapat mengoptimalkan model regresi untuk menghasilkan prediksi yang lebih akurat.

Berikut ini beberapa jenis fungsi kerugian yang umum digunakan dalam regresi:

3.3.1 Mean Squared Error (MSE) Loss:

$$L(y_i,\ h(x_i)) \ = \ (h(x_i) \ - \ y_i)^2$$

MSE loss adalah fungsi kerugian yang paling umum digunakan dalam regresi linier, di mana kita menghitung kuadrat perbedaan antara nilai prediksi dan nilai sebenarnya. Fungsi kerugian ini sensitif terhadap pencilan dan mengutamakan akurasi prediksi.

3.3.2 Mean Absolute Error (MAE) Loss:

$$L(y_i,\ h(x_i)) \ = \ |h(x_i) \ - \ y_i|$$

MAE loss menghitung nilai absolut perbedaan antara nilai prediksi dan nilai sebenarnya. Fungsi kerugian ini lebih toleran terhadap pencilan dan memiliki gradien yang konstan, sehingga lebih mudah dioptimalkan.

3.3.3 Huber Loss:

$$L(y_i,\ h(x_i)) \ = \ 0.5(h(x_i) \ - \ y_i)^2,\ jika\ |h(x_i) \ - \ y_i| \ <= \ \delta$$
$$L(y_i,\ h(x_i)) \ = \ \delta|h(x_i) \ - \ y_i| \ - \ 0.5\delta^2,\ jika\ |h(x_i) \ - \ y_i| \ > \ \delta$$

Huber loss adalah kombinasi dari MSE loss dan MAE loss, di mana fungsi kerugian ini bersifat kuadratik untuk kesalahan kecil dan linier untuk kesalahan besar. Ini menciptakan fungsi kerugian yang lebih halus dan memiliki gradien yang lebih kecil untuk kesalahan besar, sehingga lebih toleran terhadap pencilan.

Untuk mengoptimalkan model regresi, kita perlu memilih fungsi kerugian yang sesuai dengan karakteristik data dan tujuan pemodelan. Setelah memilih fungsi kerugian yang sesuai, kita dapat menggabungkannya dengan metode optimasi, seperti gradient descent,

untuk menemukan nilai-nilai parameter yang meminimalkan fungsi biaya dan menghasilkan model regresi yang optimal.

Selain fungsi kerugian yang telah disebutkan di atas, ada banyak fungsi kerugian lain yang dapat digunakan dalam berbagai situasi, tergantung pada jenis regresi yang digunakan dan tujuan pemodelan. Beberapa fungsi kerugian tambahan yang mungkin dijumpai dalam regresi meliputi:

3.3.4 Log-Cosh Loss:

$$L(y_i, h(x_i)) = cosh(h(x_i) - y_i) - 1$$

Log-Cosh loss merupakan fungsi kerugian berbasis eksponensial yang memberikan tingkat kehalusan yang lebih besar daripada MSE dan MAE. Fungsi ini bekerja dengan menghitung kosinus hiperbolik dari perbedaan antara nilai prediksi dan nilai sebenarnya, kemudian mengurangkan satu. Seperti Huber loss, Log-Cosh loss lebih toleran terhadap pencilan dan dapat digunakan ketika data mengandung banyak noise atau ketidakpastian.

3.3.5 Quantile Loss:

$$L(y_i, h(x_i)) = \tau(y_i - h(x_i)), jika\, y_i >= h(x_i)$$
$$L(y_i, h(x_i)) = (1 - \tau)(h(x_i) - y_i), jika\, y_i < h(x_i)$$

Quantile loss digunakan dalam regresi kuantil, di mana tujuannya adalah untuk memprediksi kuantil tertentu dari distribusi variabel dependen. Fungsi kerugian ini memungkinkan kita untuk mengoptimalkan model secara kondisional pada kuantil yang diinginkan. Nilai τ menentukan kuantil yang diinginkan, misalnya, τ = 0,5 untuk median.

3.3.6 Hinge Loss:

$$L(y_i, h(x_i)) = max(0, 1 - y_i h(x_i))$$

Hinge loss digunakan dalam regresi SVM (Support Vector Machine) dan umumnya ditemukan dalam klasifikasi biner. Namun, dengan beberapa modifikasi, hinge loss juga dapat digunakan dalam konteks regresi. Fungsi kerugian ini mencoba memaksimalkan margin antara titik data dan hyperplane yang dipilih, yang menghasilkan model yang lebih tahan terhadap noise dan potensi overfitting.

Menggunakan fungsi kerugian yang sesuai dalam regresi sangat penting untuk mencapai hasil yang optimal. Menyesuaikan fungsi kerugian dengan karakteristik data dan masalah

yang ingin kamu selesaikan akan memungkinkan model untuk menghasilkan prediksi yang lebih akurat dan efisien.

3.4 Metode Optimasi

Setelah memahami konsep fungsi hipotesis, fungsi biaya, dan fungsi kerugian, langkah selanjutnya dalam proses regresi adalah menemukan parameter yang optimal untuk model dengan meminimalkan fungsi biaya. Untuk melakukan ini, kita perlu menggunakan teknik optimasi yang efisien dan efektif. Dalam subbab ini, kita akan membahas beberapa metode optimasi yang populer digunakan dalam regresi, seperti gradient descent, stochastic gradient descent, dan optimasi berbasis metode lainnya.

3.4. Gradient Descent

Gradient descent adalah algoritma optimasi yang paling umum digunakan dalam regresi. Algoritma ini bekerja dengan menghitung gradien (turunan parsial) dari fungsi biaya sehubungan dengan setiap parameter model dan menggabungkan informasi ini untuk menghasilkan update iteratif pada parameter.

Cara kerja gradient descent adalah sebagai berikut:

1. Inisialisasi parameter model dengan nilai sembarang (misalnya, nol atau nilai acak kecil).
2. Hitung gradien fungsi biaya sehubungan dengan setiap parameter.
3. Perbarui setiap parameter dengan mengurangkan gradien yang dikalikan dengan learning rate (α).
4. Ulangi langkah 2 dan 3 hingga konvergensi (misalnya, perubahan parameter di bawah ambang batas yang ditentukan).

Rumus umum untuk perbaruan parameter dalam gradient descent adalah:

$$\theta_{k} = \theta_{k} - \alpha * \nabla J(\theta_{k})$$

di mana θ_k adalah parameter ke-k, α adalah learning rate, dan $\nabla J(\theta_k)$ adalah gradien fungsi biaya sehubungan dengan parameter ke-k.

3.4.2 Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) adalah variasi dari gradient descent yang dirancang untuk mengatasi masalah kecepatan dan efisiensi pada data set yang sangat besar. Alih-alih menghitung gradien dari seluruh data set (seperti yang dilakukan oleh gradient descent), SGD menghitung gradien pada satu sampel data yang dipilih secara acak pada setiap iterasi. Ini membuat algoritma ini jauh lebih cepat daripada gradient descent, terutama pada data set yang sangat besar.

Cara kerja stochastic gradient descent adalah sebagai berikut:

1. Inisialisasi parameter model dengan nilai sembarang.
2. Pilih satu sampel data secara acak dari data set.
3. Hitung gradien fungsi biaya sehubungan dengan setiap parameter, menggunakan sampel data yang dipilih.
4. Perbarui setiap parameter dengan mengurangkan gradien yang dikalikan dengan learning rate (α).
5. Ulangi langkah 2 hingga 4 hingga konvergensi.

3.4.3 Mini-Batch Gradient Descent

Mini-batch gradient descent merupakan kompromi antara gradient descent dan stochastic gradient descent. Algoritma ini menghitung gradien menggunakan sejumlah kecil sampel data (mini-batch) yang dipilih secara acak pada setiap iterasi, daripada menggunakan satu sampel (seperti SGD) atau seluruh data set (seperti gradient descent). Ini menggabungkan kecepatan dan efisiensi dari SGD dengan kestabilan dari gradient descent.

Cara kerja mini-batch gradient descent adalah sebagai berikut:

1. Inisialisasi parameter model dengan nilai sembarang.
2. Pilih sejumlah sampel data (mini-batch) secara acak dari data set.
3. Hitung gradien fungsi biaya sehubungan dengan setiap parameter, menggunakan mini-batch yang dipilih.
4. Perbarui setiap parameter dengan mengurangkan gradien yang dikalikan dengan learning rate (α).
5. Ulangi langkah 2 hingga 4 hingga konvergensi.

3.4.4 Optimasi Berbasis Metode Lainnya

Selain metode gradient-based yang telah disebutkan, terdapat berbagai metode optimasi lain yang bisa digunakan dalam regresi, di antaranya:

a. Algoritma Newton-Raphson

Algoritma Newton-Raphson adalah metode optimasi yang menggunakan informasi tentang gradien dan hessian (matriks turunan kedua) dari fungsi biaya untuk mencari minimum lokal. Metode ini sering kali lebih cepat dalam mencapai konvergensi dibandingkan dengan gradient descent, karena menggunakan informasi tentang kelengkungan fungsi biaya. Namun, algoritma ini mungkin lebih sulit diimplementasikan dan memerlukan lebih banyak perhitungan.

b. Metode Quasi-Newton

Metode Quasi-Newton adalah variasi dari algoritma Newton-Raphson yang menggunakan perkiraan hessian daripada menghitung hessian secara eksplisit. Salah satu metode Quasi-Newton yang populer adalah Broyden-Fletcher-Goldfarb-Shanno (BFGS) dan

Limited-memory BFGS (L-BFGS). Metode-metode ini biasanya lebih cepat dan lebih mudah diimplementasikan daripada Newton-Raphson, tetapi mungkin masih lebih lambat daripada metode gradient-based.

c. Metode Optimasi Konveks

Beberapa masalah regresi dapat diformulasikan sebagai masalah optimasi konveks, di mana fungsi biaya memiliki bentuk konveks dan minimum global dijamin ada. Dalam kasus ini, metode optimasi konveks, seperti Simplex, Barier, atau Metode Titik Dalam, dapat digunakan untuk mencari minimum global.

3.5 Evaluasi Model Regresi

Setelah membangun model regresi dan mengoptimalkan parameter dengan metode optimasi yang sesuai, langkah selanjutnya adalah mengevaluasi kinerja model tersebut. Evaluasi model regresi melibatkan pengukuran seberapa baik model mampu memprediksi variabel target pada data yang tidak digunakan saat melatih model. Subbab ini akan membahas metrik evaluasi yang umum digunakan dalam regresi, serta teknik pemilihan model yang berguna untuk menentukan model yang paling sesuai dengan data.

3.5.1 Metrik Evaluasi

Berikut adalah beberapa metrik evaluasi yang umum digunakan untuk mengukur kinerja model regresi:

3.5.1.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) adalah metrik evaluasi yang menghitung rata-rata perbedaan absolut antara nilai prediksi dan nilai sebenarnya:

$$MAE = (1/n) * \Sigma |y\_i - \hat{y}\_i|$$

di mana n adalah jumlah sampel data, y_i adalah nilai sebenarnya, dan ŷ_i adalah nilai prediksi.

MAE memberikan ukuran kesalahan prediksi secara rata-rata, dan semakin kecil nilai MAE, semakin baik model dalam membuat prediksi.

3.5.1.2 Mean Squared Error (MSE)

Mean Squared Error (MSE) adalah metrik evaluasi yang menghitung rata-rata kuadrat perbedaan antara nilai prediksi dan nilai sebenarnya:

$$MSE = (1/n) * \Sigma (y\_i - \hat{y}\_i)^2$$

MSE memberikan ukuran kesalahan prediksi yang dikuadratkan, yang memberikan penekanan lebih pada kesalahan yang lebih besar. Semakin kecil nilai MSE, semakin baik model dalam membuat prediksi.

3.5.1.3 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) adalah metrik evaluasi yang menghitung akar kuadrat dari rata-rata kuadrat perbedaan antara nilai prediksi dan nilai sebenarnya:

$$RMSE = \sqrt{(1/n) * \Sigma (y\_i - \hat{y}\_i)^2}$$

RMSE memberikan ukuran kesalahan prediksi yang mirip dengan MSE tetapi dalam satuan yang sama dengan variabel target. Semakin kecil nilai RMSE, semakin baik model dalam membuat prediksi.

3.5.1.4 R-squared (Koefisien Determinasi)

R-squared, atau koefisien determinasi, adalah metrik evaluasi yang mengukur seberapa baik model regresi menjelaskan variasi dalam data. R-squared didefinisikan sebagai proporsi variasi total dalam variabel target yang dijelaskan oleh model:

$$R^2 = 1 - (\Sigma (y\_i - \hat{y}\_i)^2) / (\Sigma (y\_i - \bar{y})^2)$$

di mana $\bar{y}$ adalah rata-rata nilai target.

R-squared berkisar antara 0 dan 1, dengan 1 menunjukkan model menjelaskan 100% variasi dalam data, dan 0 menunjukkan model tidak menjelaskan variasi sama sekali. Semakin tinggi nilai R-squared, semakin baik model dalam menjelaskan variasi dalam data.

3.5.2 Pemilihan Model

Pemilihan model adalah proses memilih model yang paling sesuai dengan data dari sekumpulan model kandidat.
Dalam konteks regresi, pemilihan model melibatkan mengevaluasi berbagai model yang berbeda dalam hal kompleksitas dan kinerja prediksi, serta memilih model yang memberikan keseimbangan yang baik antara keduanya. Berikut adalah beberapa teknik umum yang digunakan dalam pemilihan model regresi:

3.5.2.1 Train-Test Split

Train-test split adalah metode sederhana untuk membagi data menjadi dua bagian: satu untuk melatih model (train set) dan yang lainnya untuk menguji kinerja model (test set). Biasanya, sekitar 70-80% data digunakan untuk pelatihan, dan sisanya digunakan untuk pengujian. Model yang menghasilkan kesalahan prediksi terendah pada data pengujian dianggap sebagai model yang paling sesuai.

3.5.2.2 K-Fold Cross-Validation

K-Fold Cross-Validation adalah metode yang lebih canggih untuk mengevaluasi kinerja model regresi. Dalam metode ini, data dibagi menjadi k lipatan yang sama besar. Model dilatih k kali, di mana pada setiap iterasi, satu lipatan digunakan sebagai data pengujian dan sisanya sebagai data pelatihan. Metrik evaluasi kemudian dihitung sebagai rata-rata dari kesalahan prediksi di semua iterasi. Model yang menghasilkan kesalahan prediksi terendah pada cross-validation dianggap sebagai model yang paling sesuai.

3.5.2.3 Regularisasi

Regularisasi adalah teknik yang digunakan untuk mencegah overfitting dalam model regresi. Overfitting terjadi ketika model terlalu kompleks dan menyesuaikan diri dengan kebisingan data, sehingga menghasilkan kinerja prediksi yang buruk pada data baru. Dalam regresi, regularisasi melibatkan menambahkan suatu hukuman pada fungsi biaya yang berkaitan dengan besar nilai koefisien model. Metode regularisasi yang umum digunakan dalam regresi meliputi regresi Ridge dan Lasso, yang telah dibahas sebelumnya di Subbab 2.5.

3.5.2.4 Information Criteria

Information criteria adalah metode lain untuk memilih model regresi yang paling sesuai. Metode ini melibatkan penghitungan skor yang mempertimbangkan kesalahan prediksi dan jumlah parameter dalam model. Dua kriteria informasi yang umum digunakan dalam pemilihan model regresi adalah Akaike Information Criterion (AIC) dan Bayesian Information Criterion (BIC):

$$AIC = 2k - 2\,ln(L)$$
$$BIC = k\,ln(n) - 2\,ln(L)$$

di mana k adalah jumlah parameter dalam model, n adalah jumlah sampel data, dan L adalah kemungkinan maksimum model. Model dengan nilai AIC atau BIC yang lebih rendah dianggap sebagai model yang lebih sesuai.

Dengan menggabungkan metrik evaluasi dan teknik pemilihan model yang telah dibahas, kamu dapat menilai kinerja model regresi yang telah dikembangkan dan memilih model yang paling sesuai dengan data. Setelah memilih model terbaik, langkah selanjutnya adalah menerapkan model tersebut untuk membuat prediksi pada data baru dan menggunakan hasilnya dalam pengambilan keputusan atau analisis lebih lanjut.

3.6 Validasi Silang

Validasi silang adalah teknik yang digunakan untuk menilai kinerja model regresi dengan lebih teliti dan mengurangi risiko overfitting. Dalam validasi silang, data dibagi menjadi beberapa lipatan (folds) yang sama besar, dan model dilatih dan diuji beberapa kali dengan setiap lipatan digunakan sebagai data pengujian tepat satu kali. Validasi silang sangat

penting dalam memastikan bahwa model yang dipilih memiliki kinerja yang baik pada data yang tidak digunakan dalam pelatihan.

3.6.1 Prinsip Dasar Validasi Silang

Validasi silang adalah proses di mana model regresi dilatih dan diuji beberapa kali dengan menggunakan bagian yang berbeda dari data. Tujuannya adalah untuk mengurangi risiko overfitting dan memperoleh estimasi yang lebih akurat tentang kinerja model pada data yang tidak terlihat sebelumnya. Validasi silang melibatkan langkah-langkah berikut:

Bagi data menjadi beberapa lipatan yang sama besar.
Untuk setiap lipatan:
a. Gunakan lipatan tersebut sebagai data pengujian dan sisanya sebagai data pelatihan.
b. Latih model pada data pelatihan dan buat prediksi pada data pengujian.
c. Hitung metrik evaluasi (misalnya, RMSE) pada data pengujian.
Hitung rata-rata metrik evaluasi di semua lipatan.

3.6.2 K-Fold Cross-Validation

K-Fold Cross-Validation adalah metode validasi silang yang paling umum digunakan. Dalam metode ini, data dibagi menjadi k lipatan yang sama besar (biasanya k = 5 atau 10). Proses validasi silang k-lipatan melibatkan langkah-langkah berikut:

Bagi data menjadi k lipatan yang sama besar.
Untuk setiap lipatan ke-i (i = 1, 2, ..., k):
a. Gunakan lipatan ke-i sebagai data pengujian dan sisanya sebagai data pelatihan.
b. Latih model pada data pelatihan dan buat prediksi pada data pengujian.
c. Hitung metrik evaluasi (misalnya, RMSE) pada data pengujian.

Hitung rata-rata metrik evaluasi di semua lipatan.
Rata-rata metrik evaluasi yang dihasilkan dari proses validasi silang k-lipatan memberikan estimasi yang lebih akurat tentang kinerja model pada data yang tidak digunakan dalam pelatihan. Ini membantu mengurangi risiko overfitting dan memastikan bahwa model yang dipilih memiliki kinerja yang baik pada data yang tidak terlihat sebelumnya.

3.6.3 Leave-One-Out Cross-Validation (LOOCV)

Leave-One-Out Cross-Validation (LOOCV) adalah bentuk validasi silang di mana k setara dengan jumlah sampel data. Dalam metode ini, setiap sampel digunakan sebagai data pengujian tepat satu kali, dan sisanya digunakan sebagai data pelatihan. LOOCV melibatkan langkah-langkah berikut:

Untuk setiap sampel ke-i (i = 1, 2, ..., n), di mana n adalah jumlah total sampel:
a. Gunakan sampel ke-i sebagai data pengujian dan sisanya sebagai data pelatihan.
b. Latih model pada data pelatihan dan buat prediksi pada data pengujian.

c. Hitung metrik evaluasi (misalnya, RMSE) pada data pengujian.
Hitung rata-rata metrik evaluasi di semua sampel.
LOOCV memiliki beberapa keuntungan, seperti estimasi kinerja model yang lebih akurat dan penggunaan data yang maksimal untuk pelatihan. Namun, LOOCV juga memiliki beberapa kelemahan, seperti biaya komputasi yang tinggi, terutama untuk dataset yang sangat besar.

3.6.4 Stratified K-Fold Cross-Validation

Stratified K-Fold Cross-Validation adalah variasi dari K-Fold Cross-Validation yang digunakan ketika data memiliki distribusi kelas yang tidak seimbang. Dalam metode ini, setiap lipatan dibuat dengan memastikan bahwa proporsi kelas target sama di setiap lipatan seperti yang ada di dataset asli. Hal ini membantu memastikan bahwa model memiliki representasi yang baik dari semua kelas selama proses pelatihan dan pengujian.

Langkah-langkah untuk Stratified K-Fold Cross-Validation sama seperti K-Fold Cross-Validation, dengan perbedaan utama dalam cara pembagian lipatan dilakukan.

3.6.5 Time Series Cross-Validation

Time Series Cross-Validation adalah metode validasi silang yang dirancang khusus untuk data time series. Dalam metode ini, data dipisahkan berdasarkan urutan waktu, dan model dilatih pada data historis dan diuji pada data masa depan. Time Series Cross-Validation melibatkan langkah-langkah berikut:

Tentukan jumlah lipatan yang ingin digunakan (k).
Bagi data menjadi k bagian yang berurutan.
Untuk setiap lipatan ke-i (i = 1, 2, ..., k):
a. Gunakan bagian 1 sampai (i-1) sebagai data pelatihan dan bagian ke-i sebagai data pengujian.
b. Latih model pada data pelatihan dan buat prediksi pada data pengujian.
c. Hitung metrik evaluasi (misalnya, RMSE) pada data pengujian.
Hitung rata-rata metrik evaluasi di semua lipatan.

3.6.6 Nested Cross-Validation

Nested Cross-Validation adalah metode validasi silang yang digunakan untuk pemilihan model dan evaluasi kinerja model secara bersamaan. Dalam metode ini, validasi silang dilakukan pada dua tingkatan: validasi silang luar dan validasi silang dalam. Validasi silang dalam digunakan untuk pemilihan model, sedangkan validasi silang luar digunakan untuk evaluasi kinerja model.

3.6.7 Penggunaan Validasi Silang dalam Pemilihan Model

Validasi silang dapat digunakan untuk pemilihan model dengan cara membandingkan kinerja model yang berbeda pada data pengujian. Model dengan kinerja terbaik (misalnya,

rata-rata metrik evaluasi terendah) dapat dipilih sebagai model terbaik. Validasi silang dapat digunakan untuk memilih model dalam berbagai situasi, seperti:

- Memilih antara model regresi yang berbeda (misalnya, regresi linier, regresi logistik, regresi polinomial, dll.)
- Memilih nilai terbaik untuk parameter model (misalnya, nilai alpha pada regresi Ridge atau Lasso)
- Memilih fitur yang paling relevan untuk model (fitur seleksi)
- Menyesuaikan parameter algoritma pembelajaran (misalnya, laju pembelajaran dalam metode optimasi seperti gradient descent)

Dengan menggunakan validasi silang dalam pemilihan model, kamu dapat mengurangi risiko overfitting dan memastikan bahwa model yang dipilih memiliki kinerja yang baik pada data yang tidak terlihat sebelumnya.

Ringkasan

Dalam subbab ini, kita telah membahas berbagai aspek dari validasi silang dan bagaimana teknik ini dapat digunakan untuk menilai kinerja model regresi dengan lebih teliti. Validasi silang memungkinkan kamu untuk mengurangi risiko overfitting dan memilih model yang paling sesuai dengan data.

Kita telah membahas beberapa metode validasi silang yang berbeda, termasuk K-Fold Cross-Validation, Leave-One-Out Cross-Validation (LOOCV), Stratified K-Fold Cross-Validation, Time Series Cross-Validation, dan Nested Cross-Validation. Setiap metode ini memiliki kelebihan dan kekurangannya, dan metode yang paling sesuai akan tergantung pada jenis data dan tujuan analisis.

Bab 4: Persiapan Data untuk Regresi

4.1 Pemilihan Fitur

Pemilihan fitur adalah proses pemilihan fitur yang paling relevan dan penting dari kumpulan data asli untuk digunakan dalam pembuatan model regresi. Pemilihan fitur yang efektif dapat menghasilkan model yang lebih sederhana, mudah diinterpretasi, dan memiliki performa yang lebih baik. Dalam subbab ini, kita akan membahas berbagai teknik pemilihan fitur yang dapat membantu kamu dalam meningkatkan kinerja model regresi. Selanjutnya, kita akan memberikan contoh script Python untuk mengaplikasikan beberapa teknik pemilihan fitur ini, dan menjelaskan secara matematis.

Pertama, kita akan membuat data sintetis sebagai contoh untuk mendemonstrasikan teknik pemilihan fitur. Kamu dapat menggantinya dengan data nyata saat menerapkan teknik ini pada kasus nyata.

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression

np.random.seed(42)

# Membuat data sintetis
X, y = make_regression(n_samples=100, n_features=10, noise=0.1)
data = pd.DataFrame(X, columns=[f'fitur_{i}' for i in range(1, 11)])
data['target'] = y

data.head()
```

Output:

	fitur_1	fitur_2	fitur_3	fitur_4	fitur_5	fitur_6	fitur_7	fitur_8	fitur_9	fitur_10	target
0	-0.926930	-1.430141	1.632411	-3.241267	-1.247783	-1.024388	0.130741	-0.059525	-0.252568	-0.440044	-501.122200
1	0.202923	0.334457	0.285865	1.547505	-0.387702	1.795878	2.010205	-1.515744	-0.612789	0.658544	416.118299
2	-0.241236	0.456753	0.342725	-1.251539	1.117296	1.443765	0.447709	0.352055	-0.082151	0.569767	231.940342
3	0.289775	-1.008086	-2.038125	0.871125	-0.408075	-0.326024	-0.351513	2.075401	1.201214	-1.870792	-203.757931
4	-0.007973	-0.190339	-1.037246	0.077368	0.538910	-0.861284	-1.382800	1.479944	1.523124	-0.875618	-183.602424

4.1.1. Pemilihan Fitur Berbasis Statistik

Teknik pemilihan fitur berbasis statistik melibatkan penggunaan metrik statistik untuk menentukan seberapa penting suatu fitur dalam menjelaskan variabilitas dalam variabel target. Beberapa metrik umum yang digunakan dalam pemilihan fitur berbasis statistik meliputi koefisien korelasi Pearson.

a. Koefisien Korelasi Pearson
Koefisien korelasi Pearson mengukur korelasi linier antara dua variabel. Nilai koefisien korelasi berkisar antara -1 dan 1. Nilai yang mendekati 1 menunjukkan korelasi positif yang kuat, sementara nilai yang mendekati -1 menunjukkan korelasi negatif yang kuat. Nilai yang mendekati 0 menunjukkan tidak adanya korelasi antara variabel.

Secara matematis, koefisien korelasi Pearson dihitung sebagai:

$r = \Sigma((x - mean(X)) * (y - mean(Y))) / sqrt(\Sigma(x - mean(X))^2 * \Sigma(y - mean(Y))^2)$

Dimana x dan y adalah dua variabel yang ingin dihitung korelasinya, mean(X) dan mean(Y) adalah rata-rata dari masing-masing variabel, dan Σ adalah simbol untuk menjumlahkan.

Untuk menggunakan koefisien korelasi Pearson dalam pemilihan fitur, kamu dapat menghitung korelasi antara setiap fitur dan variabel target. Fitur dengan korelasi yang tinggi (positif atau negatif) dengan variabel target akan dianggap lebih penting.

```
# Menghitung korelasi Pearson antara setiap fitur dan variabel target
correlations = data.corr()['target'].drop('target')

# Menampilkan fitur yang memiliki korelasi yang tinggi dengan variabel
target (misalnya, korelasi absolut 0.5)
selected_features = correlations[abs(correlations) > 0.5].index
print("Fitur yang terpilih berdasarkan korelasi Pearson:")
print(selected_features)
```

Output:

```
Fitur yang terpilih berdasarkan korelasi Pearson:
Index(['fitur_6', 'fitur_7'], dtype='object')
```

4.1.2. Pemilihan Fitur Berbasis Model

Pemilihan fitur berbasis model melibatkan penggunaan model machine learning untuk menilai pentingnya setiap fitur dalam menjelaskan variabilitas dalam variabel target. Beberapa model yang umum digunakan dalam pemilihan fitur berbasis model meliputi regresi Lasso dan algoritma berbasis pohon seperti Random Forest.

a. Regresi Lasso

Regresi Lasso adalah teknik regresi linier yang menggunakan regularisasi L1 untuk mengurangi kompleksitas model. Dalam regresi Lasso, beberapa koefisien fitur mungkin menjadi nol, yang berarti fitur tersebut tidak berkontribusi pada prediksi variabel target. Oleh karena itu, fitur dengan koefisien yang tidak nol dianggap penting dan dipilih untuk digunakan dalam model.

Secara matematis, regresi Lasso mencoba meminimalkan fungsi berikut:

$$L(w) = \Sigma(y - Xw)^2 + \alpha\Sigma|w|$$

Dimana y adalah variabel target, X adalah matriks fitur, w adalah vektor koefisien, dan α adalah parameter regularisasi yang mengontrol seberapa kuat efek regularisasi L1.

```
from sklearn.linear_model import Lasso

# Membuat dan melatih model Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X, y)

# Menampilkan fitur yang memiliki koefisien yang tidak nol
selected_features = [col for col, coef in zip(data.columns[:-1],
lasso.coef_) if coef != 0]
print("Fitur yang terpilih berdasarkan Regresi Lasso:")
print(selected_features)
```

Output:

```
Fitur yang terpilih berdasarkan Regresi Lasso:
['fitur_1', 'fitur_2', 'fitur_3', 'fitur_4', 'fitur_5', 'fitur_6', 'fitur_7', 'fitur_8', 'fitur_9', 'fitur_10']
```

b. Algoritma Berbasis Pohon (Random Forest)
Algoritma berbasis pohon, seperti Random Forest, dapat digunakan untuk pemilihan fitur dengan menghitung pentingnya fitur berdasarkan jumlah pemisahan yang melibatkan fitur tersebut dan penurunan kepentingan Gini atau penurunan impuritas yang dihasilkan oleh pemisahan tersebut. Fitur dengan nilai penting yang lebih tinggi dianggap lebih penting dan dipilih untuk digunakan dalam model.

Secara matematis, kepentingan fitur dihitung berdasarkan perbaikan Gini yang dihasilkan oleh pemisahan pada fitur tersebut. Perbaikan Gini dihitung sebagai:

$$\Delta Gini = Gini(parent) - \Sigma(weight(child) * Gini(child))$$

Dimana Gini(parent) adalah impuritas Gini dari simpul induk, weight(child) adalah bobot simpul anak (jumlah sampel di simpul anak dibagi jumlah sampel di simpul induk), dan Gini(child) adalah impuritas Gini dari simpul anak.

```
from sklearn.ensemble import RandomForestRegressor
# Membuat dan melatih model Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)
```

```
# Menampilkan fitur dengan nilai penting yang lebih tinggi dari ambang
batas (misalnya, 0.05)
selected_features = [col for col, importance in zip(data.columns[:-1],
rf.feature_importances_) if importance > 0.05]
print("Fitur yang terpilih berdasarkan Random Forest:")
print(selected_features)
```

Output:

```
Fitur yang terpilih berdasarkan Random Forest:
['fitur_2', 'fitur_5', 'fitur_6', 'fitur_7', 'fitur_10']
```

4.1.3. Pemilihan Fitur Secara Rekursif

Pemilihan fitur secara rekursif adalah teknik pemilihan fitur yang melibatkan pembuatan model berulang kali dan penghapusan fitur yang paling tidak penting pada setiap iterasi. Teknik ini biasanya digunakan dengan algoritma berbasis pohon, seperti Random Forest atau XGBoost, untuk mengukur pentingnya fitur.

Secara matematis, pemilihan fitur secara rekursif melibatkan langkah-langkah berikut:

1. Melatih model pada semua fitur dan menghitung pentingnya fitur (misalnya, menggunakan perbaikan Gini untuk algoritma berbasis pohon).
2. Menghapus fitur dengan pentingnya yang paling rendah dan melatih model lagi.
3. Mengulangi langkah 2 sampai jumlah fitur yang diinginkan terpilih.

```
from sklearn.feature_selection import RFE

# Membuat model Random Forest dan RFE
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rfe = RFE(estimator=rf, n_features_to_select=5)

# Melatih RFE dan menampilkan fitur yang terpilih
rfe.fit(X, y)
selected_features = [col for col, is_selected in zip(data.columns[:-1],
rfe.support_) if is_selected]
print("Fitur yang terpilih berdasarkan RFE dengan Random Forest:")
print(selected_features)
```

Output:

```
Fitur yang terpilih berdasarkan Random Forest:
['fitur_2', 'fitur_5', 'fitur_6', 'fitur_7', 'fitur_10']
```

Dalam contoh di atas, kita menggunakan RFE untuk memilih 5 fitur terpenting dengan menggunakan Random Forest sebagai model dasar. Kamu dapat menggantinya dengan model lain atau mengubah jumlah fitur yang ingin dipilih sesuai kebutuhan.

Dengan mengikuti contoh-contoh di atas, kamu dapat menerapkan berbagai teknik pemilihan fitur untuk meningkatkan kinerja model regresi. Setelah kamu memiliki fitur yang dipilih, langkah selanjutnya dalam persiapan data adalah normalisasi dan standarisasi data, imputasi data hilang, deteksi dan penanganan outlier, dan pembagian data untuk pelatihan dan pengujian.

4.2. Normalisasi dan Standarisasi Data

Dalam proses persiapan data, normalisasi dan standarisasi adalah teknik penting yang digunakan untuk mengubah data mentah menjadi format yang lebih mudah diolah oleh algoritma machine learning. Kedua teknik ini membantu mengurangi masalah skala dalam fitur yang berbeda, sehingga model dapat bekerja lebih efisien. Dalam subbab ini, kita akan menjelaskan secara matematis normalisasi dan standarisasi data serta memberikan contoh penggunaan Python script menggunakan data sintetis.

4.2.1 Normalisasi Data

Normalisasi adalah proses mengubah fitur numerik sehingga semua fitur memiliki rentang nilai yang sama, biasanya antara 0 dan 1. Ini membantu mengurangi bias yang disebabkan oleh skala fitur yang berbeda, terutama dalam algoritma yang bergantung pada jarak seperti k-Nearest Neighbors dan Support Vector Machines. Normalisasi data secara matematis dapat dilakukan dengan menggunakan formula berikut:

$$x\_normalized = (x - x\_min) / (x\_max - x\_min)$$

Di mana:

- x: nilai fitur yang ingin dinormalisasi
- x_min: nilai minimum dari fitur
- x_max: nilai maksimum dari fitur
- x_normalized: nilai fitur yang telah dinormalisasi

Berikut contoh penggunaan Python script untuk melakukan normalisasi data menggunakan data sintetis:

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Membuat data sintetis
data = np.random.randint(0, 100, (10, 3))
print("Data sebelum normalisasi:")
```

```
print(data)

# Melakukan normalisasi menggunakan MinMaxScaler
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)
print("\nData setelah normalisasi:")
print(normalized_data)
```

Output:

```
Data sebelum normalisasi:
[[69 45 42]
 [20 22 79]
 [46 14 24]
 [35 74 50]
 [14 67 20]
 [12 85 18]
 [33 77 67]
 [51 26 28]
 [27 34  7]
 [15 35 29]]

Data setelah normalisasi:
[[1.         0.43661972 0.48611111]
 [0.14035088 0.11267606 1.        ]
 [0.59649123 0.         0.23611111]
 [0.40350877 0.84507042 0.59722222]
 [0.03508772 0.74647887 0.18055556]
 [0.         1.         0.15277778]
 [0.36842105 0.88732394 0.83333333]
 [0.68421053 0.16901408 0.29166667]
 [0.26315789 0.28169014 0.        ]
 [0.05263158 0.29577465 0.30555556]]
```

4.2.2 Standarisasi Data

Standarisasi adalah proses mengubah fitur numerik sehingga mereka memiliki mean (rata-rata) 0 dan standard deviation (standar deviasi) 1. Teknik ini berguna untuk algoritma yang sangat sensitif terhadap variasi dalam skala fitur, seperti algoritma berbasis gradien (misalnya, Gradient Descent) dan Regularization. Standarisasi data secara matematis dapat dilakukan dengan menggunakan formula berikut:

$$x\_standardized = (x - x\_mean) / x\_std$$

Di mana:

- x: nilai fitur yang ingin distandarisasi
- x_mean: nilai mean dari fitur
- x_std: nilai standard deviation dari fitur

- x_standardized: nilai fitur yang telah distandarisasi

Berikut contoh penggunaan Python script untuk melakukan standarisasi data menggunakan data sintetis:

```
from sklearn.preprocessing import StandardScaler

# Membuat data sintetis
data = np.random.randint(0, 100, (10, 3))
print("Data sebelum standarisasi:")
print(data)

# Melakukan standarisasi menggunakan StandardScaler
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data)
print("\nData setelah standarisasi:")
print(standardized_data)
```

Output:

```
Data sebelum standarisasi:
[[17  1 48]
 [76  3 89]
 [88 93 74]
 [ 6 94 73]
 [38 12 30]
 [70 62 55]
 [41 60 41]
 [55 35 83]
 [67  6 53]
 [ 0 43 44]]

Data setelah standarisasi:
[[-0.99324518 -1.1783865  -0.59136366]
 [ 1.04152793 -1.11931951  1.61280999]
 [ 1.45538009  1.53869516  0.806405  ]
 [-1.37260966  1.56822866  0.75264466]
 [-0.2690039  -0.85351804 -1.55904966]
 [ 0.83460185  0.62315677 -0.21504133]
 [-0.16554086  0.56408978 -0.967686  ]
 [ 0.31728666 -0.17424763  1.29024799]
 [ 0.73113881 -1.03071902 -0.322562  ]
 [-1.57953574  0.06202034 -0.806405  ]]
```

Kesimpulan

Normalisasi dan standarisasi data merupakan langkah penting dalam persiapan data untuk regresi. Kedua teknik ini membantu mengurangi masalah yang dihadapi oleh algoritma machine learning saat mengolah data dengan skala yang berbeda. Dengan menggunakan

normalisasi dan standarisasi, kita dapat meningkatkan kinerja model regresi dan memastikan bahwa semua fitur memiliki pengaruh yang seimbang dalam proses pembelajaran.

Meskipun kedua teknik ini sering digunakan secara luas, penting untuk dicatat bahwa tidak semua kasus memerlukan normalisasi atau standarisasi. Dalam beberapa situasi, algoritma tertentu mungkin tidak memerlukan pra-pemrosesan data dalam bentuk normalisasi atau standarisasi. Misalnya, algoritma berbasis pohon seperti Decision Trees dan Random Forests umumnya tidak terpengaruh oleh skala fitur dan tidak memerlukan normalisasi atau standarisasi.

Namun, jika kamu menggunakan algoritma yang sensitif terhadap skala fitur atau memerlukan pengukuran jarak antara titik data, normalisasi atau standarisasi data menjadi langkah yang penting. Oleh karena itu, penting untuk mengevaluasi kebutuhan dari algoritma yang kamu gunakan dan memilih metode pra-pemrosesan yang sesuai.

4.3 Imputasi Data Hilang

Mengatasi data hilang merupakan tantangan yang sering dihadapi dalam analisis data dan pembelajaran mesin. Data hilang dapat menyebabkan distorsi pada hasil analisis dan menyebabkan model regresi yang tidak efisien. Pada subbab ini, kita akan membahas beberapa metode umum untuk mengatasi data yang hilang, serta memberikan contoh Python script menggunakan data sintetis.

Ada beberapa alasan mengapa data hilang dalam dataset, seperti kesalahan pengumpulan data, ketidaklengkapan sumber data, atau kerusakan data. Ada beberapa pendekatan umum yang dapat digunakan untuk mengatasi data yang hilang, seperti:

- Penghapusan data
- Imputasi rata-rata
- Imputasi median
- Imputasi modus
- Imputasi berbasis model

Mari kita bahas masing-masing metode ini secara lebih rinci dan memberikan contoh Python script menggunakan data sintetis.

4.3.1. Penghapusan Data

Metode ini melibatkan penghapusan baris atau kolom yang memiliki data hilang. Meskipun ini adalah cara yang mudah dan cepat untuk mengatasi data hilang, metode ini tidak selalu ideal karena bisa mengakibatkan hilangnya informasi yang penting. Penghapusan data hanya disarankan jika jumlah data yang hilang relatif kecil dan tidak akan mempengaruhi kualitas analisis secara signifikan.

```python
import numpy as np
import pandas as pd

# Membuat data sintetis
data = {'Fitur_1': [1, 2, 3, np.nan, 5],
        'Fitur_2': [2, np.nan, 6, 7, 3],
        'Fitur_3': [9, 1, 2, np.nan, 8]}

df = pd.DataFrame(data)

# Menghapus baris dengan data hilang
df_no_missing = df.dropna()

print("Data asli:\n", df)
print("\nData setelah penghapusan data hilang:\n", df_no_missing)
```

Output:

```
Data asli:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      NaN      1.0
2       3.0      6.0      2.0
3       NaN      7.0      NaN
4       5.0      3.0      8.0

Data setelah penghapusan data hilang:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
2       3.0      6.0      2.0
4       5.0      3.0      8.0
```

4.1.2. Imputasi Rata-rata

Imputasi rata-rata menggantikan data yang hilang dengan rata-rata nilai pada kolom yang sama. Ini adalah metode yang populer karena cukup mudah diimplementasikan dan dapat mengurangi dampak data yang hilang pada analisis.

```python
# Menggantikan data hilang dengan rata-rata
df_mean_imputed = df.fillna(df.mean())

print("Data asli:\n", df)
print("\nData setelah imputasi rata-rata:\n", df_mean_imputed)
```

Output:

```
Data asli:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      NaN      1.0
2       3.0      6.0      2.0
3       NaN      7.0      NaN
4       5.0      3.0      8.0

Data setelah imputasi rata-rata:
    Fitur_1  Fitur_2  Fitur_3
0      1.00      2.0      9.0
1      2.00      4.5      1.0
2      3.00      6.0      2.0
3      2.75      7.0      5.0
4      5.00      3.0      8.0
```

4.1.3. Imputasi Median

Imputasi median menggantikan data yang hilang dengan median nilai pada kolom yang sama. Metode ini lebih tahan terhadap outlier daripada imputasi rata-rata.

```
# Menggantikan data hilang dengan median
df_median_imputed = df.fillna(df.median())

print("Data asli:\n", df)
print("\nData setelah imputasi median:\n", df_median_imputed)
```

Output:

```
Data asli:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      NaN      1.0
2       3.0      6.0      2.0
3       NaN      7.0      NaN
4       5.0      3.0      8.0

Data setelah imputasi median:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      4.5      1.0
2       3.0      6.0      2.0
3       2.5      7.0      5.0
4       5.0      3.0      8.0
```

4.1.4. Imputasi Modus

Imputasi modus menggantikan data yang hilang dengan modus (nilai yang paling sering muncul) pada kolom yang sama. Metode ini sering digunakan untuk data kategorikal atau diskrit, di mana rata-rata atau median mungkin tidak memiliki arti yang jelas.

```
# Menggantikan data hilang dengan modus
df_mode_imputed = df.fillna(df.mode().iloc[0])

print("Data asli:\n", df)
```

```
print("\nData setelah imputasi modus:\n", df_mode_imputed)
```

Output:

```
Data asli:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      NaN      1.0
2       3.0      6.0      2.0
3       NaN      7.0      NaN
4       5.0      3.0      8.0

Data setelah imputasi modus:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      2.0      1.0
2       3.0      6.0      2.0
3       1.0      7.0      1.0
4       5.0      3.0      8.0
```

4.1.5. Imputasi Berbasis Model

Imputasi berbasis model menggunakan model statistik atau pembelajaran mesin untuk memprediksi nilai yang hilang berdasarkan nilai-nilai yang diamati lainnya dalam dataset. Beberapa metode umum termasuk regresi linier, regresi logistik, dan algoritma k-Nearest Neighbors (k-NN). Imputasi berbasis model biasanya lebih akurat daripada metode imputasi sederhana lainnya, tetapi memerlukan lebih banyak sumber daya komputasi dan waktu.

Contoh di bawah ini menggunakan algoritma k-NN untuk imputasi data hilang:

```
from sklearn.impute import KNNImputer

# Membuat objek KNNImputer
imputer = KNNImputer(n_neighbors=2)

# Melakukan imputasi dengan KNN
df_knn_imputed = pd.DataFrame(imputer.fit_transform(df),
columns=df.columns)

print("Data asli:\n", df)
print("\nData setelah imputasi KNN:\n", df_knn_imputed)
```

Output:

```
Data asli:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      NaN      1.0
2       3.0      6.0      2.0
3       NaN      7.0      NaN
4       5.0      3.0      8.0

Data setelah imputasi KNN:
    Fitur_1  Fitur_2  Fitur_3
0       1.0      2.0      9.0
1       2.0      4.5      1.0
2       3.0      6.0      2.0
3       4.0      7.0      5.0
4       5.0      3.0      8.0
```

Sebagai kesimpulan, penting untuk memilih metode imputasi yang paling sesuai dengan data yang kamu miliki dan tujuan analisis. Imputasi data hilang merupakan langkah penting dalam persiapan data untuk regresi, karena model regresi sangat sensitif terhadap data yang hilang dan dapat menghasilkan hasil yang tidak akurat jika data yang hilang tidak ditangani dengan benar. Selalu pertimbangkan karakteristik data dan tujuan analisis saat memilih metode imputasi yang paling sesuai.

4.4. Deteksi dan Penanganan Outlier

Outlier adalah titik data yang secara signifikan berbeda dari titik data lain dalam dataset. Mereka dapat menyebabkan masalah dalam analisis regresi karena mereka memiliki pengaruh yang tidak proporsional pada model yang dihasilkan. Dalam subbab ini, kita akan membahas beberapa teknik untuk mendeteksi dan menangani outlier dalam data.

4.4.1. Deteksi Outlier

Ada beberapa metode yang digunakan untuk mendeteksi outlier dalam data. Beberapa metode yang umum digunakan meliputi:

a. Box plot
b. Z-score
c. IQR (interquartile range)

a. Box plot

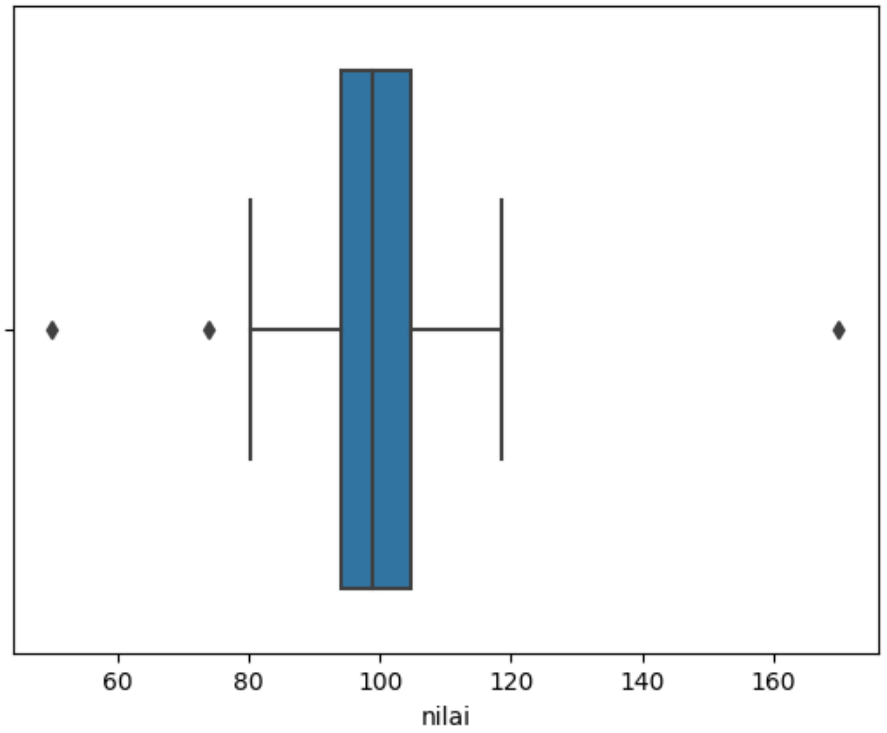
Box plot adalah representasi grafis dari distribusi data yang menunjukkan median, kuartil bawah (Q1), kuartil atas (Q3), dan nilai ekstrem dalam satu plot. Outlier diidentifikasi sebagai titik data yang berada di luar batas bawah dan atas, yang didefinisikan sebagai:

Batas bawah = Q1 - 1.5 * IQR
Batas atas = Q3 + 1.5 * IQR

IQR adalah jarak antara kuartil bawah dan atas (IQR = Q3 - Q1).

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Membuat data sintetis
np.random.seed(42)
data = np.random.normal(100, 10, 100)
data = np.append(data, [50, 170])  # Menambahkan outlier
df = pd.DataFrame(data, columns=["nilai"])

# Membuat box plot
sns.boxplot(x=df["nilai"])
plt.show()
```

Output:



b. Z-score

Z-score adalah ukuran seberapa jauh suatu titik data dari rata-rata populasi, diukur dalam satuan deviasi standar. Titik data yang memiliki z-score di luar batas (misalnya, lebih dari 2 atau 3) dianggap sebagai outlier.

```
from scipy import stats

# Menghitung z-score
```

```
z_scores = np.abs(stats.zscore(df["nilai"]))

# Menentukan batas (misalnya, z-score > 3)
outliers = np.where(z_scores > 3)

print("Index outlier:", outliers)
```

Output:

```
Index outlier: (array([100, 101]),)
```

c. IQR (interquartile range)

Seperti yang telah disebutkan sebelumnya, IQR adalah jarak antara kuartil bawah dan atas. Titik data yang berada di luar rentang IQR dianggap sebagai outlier.

```
# Menghitung Q1, Q3, dan IQR
Q1 = df["nilai"].quantile(0.25)
Q3 = df["nilai"].quantile(0.75)
IQR = Q3 - Q1

# Menentukan batas bawah dan atas
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Mendeteksi outlier
outliers = df[(df["nilai"] < lower_bound) | (df["nilai"] > upper_bound)]

print("Outlier:\n", outliers)
```

Output:

```
Outlier:
           nilai
74     73.802549
100    50.000000
101   170.000000
```

4.1.2. Penanganan Outlier

Setelah outlier terdeteksi, langkah selanjutnya adalah menangani mereka. Ada beberapa metode yang dapat digunakan untuk menangani outlier, seperti penghapusan, transformasi, dan imputasi.

a. Penghapusan

Metode yang paling sederhana untuk menangani outlier adalah dengan menghapusnya dari dataset. Namun, metode ini harus digunakan dengan hati-hati karena penghapusan outlier yang salah dapat mengakibatkan hilangnya informasi penting.

```
# Menghapus outlier menggunakan IQR
df_cleaned = df[(df["nilai"] >= lower_bound) & (df["nilai"] <=
upper_bound)]
print("Dataset setelah menghapus outlier:\n", df_cleaned)
```

Output:

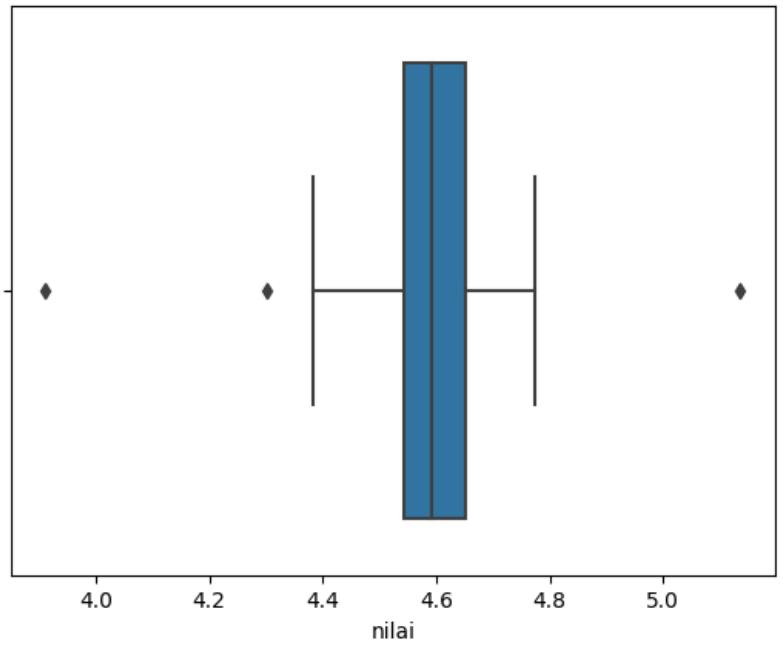
```
Dataset setelah menghapus outlier:
          nilai
0   104.967142
1    98.617357
2   106.476885
3   115.230299
4    97.658466
..         ...
95   85.364851
96  102.961203
97  102.610553
98  100.051135
99   97.654129

[99 rows x 1 columns]
```

b. Transformasi

Transformasi data dapat membantu mengurangi dampak outlier. Beberapa transformasi yang umum digunakan meliputi log, kuadrat, dan akar kuadrat. Transformasi ini membantu mengurangi efek outlier dengan mengurangi rentang variasi data.

```
# Melakukan transformasi log pada data
df_log_transformed = np.log(df["nilai"])

# Membuat box plot setelah transformasi
sns.boxplot(x=df_log_transformed)
plt.show()
```

Output:



c. Imputasi

Imputasi adalah teknik yang digunakan untuk menggantikan nilai outlier dengan nilai yang lebih sesuai, seperti median atau mean. Metode ini dapat membantu mengurangi efek outlier pada model regresi.

```
# Menggantikan outlier dengan median
median = df["nilai"].median()
df_imputed = df.copy()
df_imputed.loc[(df["nilai"] < lower_bound) | (df["nilai"] >
upper_bound), "nilai"] = median

print("Dataset setelah imputasi outlier:\n", df_imputed)
```

Output:

```
Dataset setelah imputasi outlier:
           nilai
0     104.967142
1      98.617357
2     106.476885
3     115.230299
4      97.658466
..           ...
97    102.610553
98    100.051135
99     97.654129
100    98.730437
101    98.730437

[102 rows x 1 columns]
```

4.5. Pembagian Data untuk Pelatihan dan Pengujian

Pembagian data menjadi set pelatihan dan pengujian merupakan langkah penting dalam proses regresi. Set pelatihan digunakan untuk melatih model, sedangkan set pengujian digunakan untuk mengevaluasi kinerja model. Pembagian yang baik antara data pelatihan dan pengujian akan memastikan bahwa model yang dihasilkan memiliki kinerja yang baik dan mampu menggeneralisasi pada data baru.

1. Mengapa perlu membagi data menjadi set pelatihan dan pengujian

Membagi data menjadi set pelatihan dan pengujian sangat penting untuk menghindari overfitting. Overfitting terjadi ketika model terlalu sempurna pada data pelatihan dan tidak mampu menggeneralisasi pada data baru. Dengan membagi data, kita dapat melatih model pada sebagian data dan mengukur kinerjanya pada data yang belum pernah dilihat sebelumnya. Hal ini akan memberikan gambaran yang lebih akurat tentang kinerja model dalam menghadapi data baru.

2. Bagaimana memilih proporsi data pelatihan dan pengujian yang tepat

Pemilihan proporsi data pelatihan dan pengujian tergantung pada ukuran dataset dan tujuan analisis. Beberapa aturan umum meliputi:

80/20: 80% data untuk pelatihan dan 20% untuk pengujian
70/30: 70% data untuk pelatihan dan 30% untuk pengujian
60/40: 60% data untuk pelatihan dan 40% untuk pengujian
Biasanya, semakin besar dataset, semakin kecil proporsi data pengujian yang dibutuhkan. Namun, sebaiknya pertimbangkan kebutuhan dan konteks analisis untuk menentukan proporsi yang paling sesuai.

3. Teknik untuk membagi data secara acak dan stratifikasi

Ada beberapa metode untuk membagi data menjadi set pelatihan dan pengujian, antara lain:

- Random sampling: Metode ini membagi data secara acak tanpa memperhatikan distribusi kelas target atau fitur. Ini adalah metode yang paling sederhana, namun mungkin tidak efektif jika distribusi kelas tidak seimbang.
- Stratified sampling: Metode ini membagi data sedemikian rupa sehingga distribusi kelas target atau fitur dalam set pelatihan dan pengujian sama. Ini memastikan bahwa setiap kelas memiliki representasi yang seimbang dalam kedua set.

4. Contoh script Python untuk membagi data sintetis

Berikut ini adalah contoh script Python untuk membagi data sintetis menggunakan library sklearn.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Membuat data sintetis
np.random.seed(0)
X = np.random.randn(100, 5)
y = 2 * X[:, 0] + 3 * X[:, 1] + 4 * X[:, 2] + 5 * X[:, 3] + 6 * X[:, 4]
+ np.random.normal(0, 1, 100)

# Menggabungkan fitur dan target menjadi satu DataFrame
data = pd.DataFrame(X, columns=['fitur1', 'fitur2', 'fitur3', 'fitur4',
'fitur5'])
data['target'] = y

# Membagi data menjadi set pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Menampilkan ukuran set pelatihan dan pengujian
print("Ukuran set pelatihan:", X_train.shape, y_train.shape)
print("Ukuran set pengujian:", X_test.shape, y_test.shape)
```

Output:

```
Ukuran set pelatihan: (80, 5) (80,)
Ukuran set pengujian: (20, 5) (20,)
```

Dalam contoh ini, kita membuat data sintetis dengan 100 observasi dan 5 fitur. Kemudian, kita menggabungkan fitur dan target ke dalam satu DataFrame. Selanjutnya, kita menggunakan fungsi `train_test_split` dari `sklearn.model_selection` untuk membagi data menjadi set pelatihan (80%) dan pengujian (20%). Terakhir, kita menampilkan ukuran set pelatihan dan pengujian untuk memastikan pembagian berhasil dilakukan.

Dalam rangkuman, membagi data menjadi set pelatihan dan pengujian merupakan langkah penting dalam proses regresi. Dengan memahami mengapa kita membagi data, bagaimana memilih proporsi yang tepat, dan teknik-teknik yang tersedia, kamu dapat mempersiapkan data dengan lebih baik untuk menghasilkan model regresi yang akurat dan dapat menggeneralisasi pada data baru.

Bab 5: Implementasi Regresi dengan Python

5.1. Memuat dan Menyelidiki Data

Sebelum kita mulai menerapkan teknik regresi, penting untuk memahami dan menyelidiki data yang akan kita gunakan. Dalam subbab ini, kita akan membahas cara memuat dan menyelidiki data menggunakan Python dan pustaka yang telah kita impor sebelumnya.

5.1.1 Membuat Data Sintetis

Untuk tujuan pembelajaran, kita akan membuat dataset sintetis dengan menggabungkan beberapa fitur acak dan target yang dihasilkan secara sintetis. Data sintetis ini akan membantu kita memahami konsep dasar dalam regresi tanpa harus khawatir tentang karakteristik data dunia nyata.

Berikut adalah contoh script Python untuk membuat data sintetis:

```
import numpy as np
import pandas as pd

np.random.seed(42)

# Membuat fitur acak
X1 = np.random.rand(100, 1) * 10
X2 = np.random.rand(100, 1) * 100
X3 = np.random.rand(100, 1) * 1000

# Membuat target sintetis
noise = np.random.randn(100, 1)
y = 3 * X1 + 5 * X2 + 2 * X3 + noise

# Menggabungkan fitur dan target ke dalam DataFrame
data = pd.DataFrame(np.hstack([X1, X2, X3, y]), columns=['X1', 'X2',
'X3', 'y'])
```

Di sini, kita membuat tiga fitur acak (X1, X2, X3) dengan rentang nilai yang berbeda dan menggabungkannya dengan target y yang dihasilkan secara sintetis. Kemudian, kita menggabungkan fitur dan target ini ke dalam DataFrame pandas untuk memudahkan manipulasi dan analisis data.

5.1.2 Menyelidiki Data

Setelah memuat data, langkah selanjutnya adalah menyelidikinya untuk memahami karakteristik dan struktur data. Kita akan menggunakan berbagai fungsi dari pustaka pandas untuk melakukan tugas ini.

```
# Menampilkan 5 baris pertama data
print('Sample Data')
display(data.head())

# Menampilkan informasi tentang kolom data
print('\nInformasi Kolom')
display(data.info())

# Menampilkan statistik deskriptif data
print('\nStatistika Deskriptif')
display(data.describe())

# Mengecek apakah ada nilai yang hilang
print('\nMengecek Missing Values')
display(data.isnull().sum())
```

Output:

```
Sample Data
```

	X1	X2	X3	y
0	3.745401	3.142919	642.031646	1311.059661
1	9.507143	63.641041	84.139965	514.354964
2	7.319939	31.435598	161.628714	504.539181
3	5.986585	50.857069	898.554189	2069.987396
4	1.560186	90.756647	606.429060	1669.296773

```
Informasi Kolom
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      100 non-null    float64
 1   X2      100 non-null    float64
 2   X3      100 non-null    float64
 3   y       100 non-null    float64
dtypes: float64(4)
memory usage: 3.2 KB
None
```

```
Statistika Deskriptif
```

	X1	X2	X3	y
count	100.000000	100.000000	100.000000	100.000000
mean	4.701807	49.783172	517.601331	1298.293773
std	2.974894	29.311125	293.426247	583.445738
min	0.055221	0.695213	5.061584	110.321490
25%	1.932008	24.200453	276.879864	856.946139
50%	4.641425	50.562486	562.554933	1312.807999
75%	7.302031	76.618360	752.366942	1795.511917
max	9.868869	98.565045	990.053850	2366.965127

```
Mengecek Missing Values
X1    0
X2    0
X3    0
y     0
dtype: int64
```

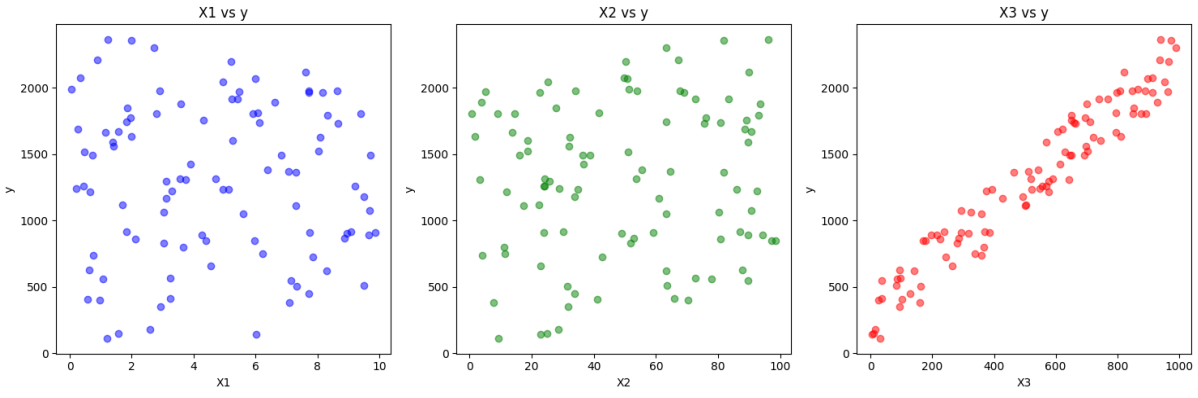
Dengan menjalankan kode di atas, kita dapat memahami karakteristik data, seperti jumlah baris dan kolom, tipe data, nilai minimum dan maksimum, rata-rata, standar deviasi, dan lainnya. Selain itu, kita juga dapat memeriksa apakah ada nilai yang hilang dalam data.

5.1.3 Visualisasi Data

Visualisasi adalah cara yang efektif untuk memahami data dan mengidentifikasi tren atau pola yang mungkin ada. Kita akan menggunakan pustaka Matplotlib untuk membuat plot yang membantu kita memahami hubungan antara fitur dan target dalam dataset sintetis.

```
import matplotlib.pyplot as plt

# Membuat scatter plot antara fitur dan target
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].scatter(data['X1'], data['y'], color='blue', alpha=0.5)
axes[0].set_xlabel('X1')
axes[0].set_ylabel('y')
axes[0].set_title('X1 vs y')

axes[1].scatter(data['X2'], data['y'], color='green', alpha=0.5)
axes[1].set_xlabel('X2')
axes[1].set_ylabel('y')
axes[1].set_title('X2 vs y')

axes[2].scatter(data['X3'], data['y'], color='red', alpha=0.5)
axes[2].set_xlabel('X3')
axes[2].set_ylabel('y')
axes[2].set_title('X3 vs y')

plt.tight_layout()
plt.show()
```

Output:



Dengan menggunakan scatter plot, kita dapat melihat hubungan antara setiap fitur (X1, X2, X3) dan target (y). Melalui visualisasi ini, kita dapat mengidentifikasi pola atau tren yang mungkin ada dalam data.

5.1.4 Korelasi antara Fitur dan Target

Salah satu cara untuk mengukur hubungan antara fitur dan target adalah dengan menghitung korelasi. Korelasi menggambarkan seberapa kuat hubungan antara dua variabel. Dalam konteks regresi, kita tertarik pada korelasi antara setiap fitur dan target.

```
# Menghitung korelasi antara fitur dan target
correlations = data.corr()
print(correlations['y'])
```

Output:

```
X1   -0.030924
X2    0.103534
X3    0.968643
y     1.000000
Name: y, dtype: float64
```

5.1.5 Eksplorasi Data Lanjutan

Selain teknik yang telah kita bahas sejauh ini, kamu juga bisa menggunakan teknik eksplorasi data lanjutan untuk mendapatkan pemahaman yang lebih dalam tentang data. Beberapa teknik yang dapat kamu gunakan meliputi analisis komponen utama (PCA), analisis klaster, dan teknik reduksi dimensi lainnya. Eksplorasi data lanjutan dapat membantu kamu mengidentifikasi fitur yang paling relevan dan mengurangi kompleksitas data sebelum menerapkan regresi.

5.2 Pemrosesan Data dan Persiapan

Sebelum menerapkan teknik regresi pada data, penting untuk memproses dan mempersiapkan data terlebih dahulu. Dalam subbab ini, kita akan membahas beberapa langkah pemrosesan data yang umum, seperti:

- Pemilihan fitur
- Normalisasi dan standarisasi data
- Imputasi data hilang
- Deteksi dan penanganan outlier
- Pembagian data untuk pelatihan dan pengujian

5.2.1 Pemilihan Fitur

Pemilihan fitur adalah proses di mana kamu memilih fitur yang paling relevan dan informatif untuk digunakan dalam model regresi. Pemilihan fitur dapat membantu mengurangi kompleksitas model, mengurangi overfitting, dan meningkatkan kecepatan pelatihan. Berikut adalah contoh bagaimana melakukan pemilihan fitur menggunakan data sintetis:

```
import pandas as pd
```

```
import numpy as np
from sklearn.feature_selection import SelectKBest, f_regression

# Membuat data sintetis
np.random.seed(42)
X = np.random.randn(100, 5)
y = X[:, 0] + 2 * X[:, 1] + np.random.randn(100) * 0.5
data = pd.DataFrame(np.column_stack((X, y)), columns=['X1', 'X2', 'X3',
'X4', 'X5', 'y'])

# Pemilihan fitur dengan SelectKBest
selector = SelectKBest(score_func=f_regression, k=2)
selector.fit(data.drop('y', axis=1), data['y'])
selected_features = data.drop('y',
axis=1).columns[selector.get_support()]
print("Fitur yang terpilih:", selected_features)
```

Output:

```
Fitur yang terpilih: Index(['X1', 'X2'], dtype='object')
```

5.2.2 Normalisasi dan Standarisasi Data

Normalisasi dan standarisasi data adalah teknik yang digunakan untuk mengubah skala data sehingga fitur yang berbeda memiliki rentang yang serupa. Normalisasi mengubah data sehingga memiliki rentang antara 0 dan 1, sementara standarisasi mengubah data sehingga memiliki rata-rata 0 dan standar deviasi 1. Berikut adalah contoh bagaimana melakukan normalisasi dan standarisasi menggunakan data sintetis:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Normalisasi
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)
normalized_data = pd.DataFrame(normalized_data, columns=data.columns)
print("Data yang dinormalisasi:\n", normalized_data.head())

# Standarisasi
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data)
standardized_data = pd.DataFrame(standardized_data,
columns=data.columns)
print("\nData yang distandarisasi:\n", standardized_data.head())
```

Output:

```
Data yang dinormalisasi:
          X1        X2        X3        X4        X5         y
0   0.573927  0.516619  0.725088  0.688341  0.368575  0.553596
1   0.402637  0.926704  0.747415  0.289215  0.488577  0.840111
2   0.348901  0.438430  0.649441  0.000000  0.138251  0.304583
3   0.325728  0.307798  0.662919  0.201367  0.186550  0.285548
4   0.801017  0.495724  0.616919  0.097860  0.320644  0.554089

Data yang distandarisasi:
          X1        X2        X3        X4        X5         y
0   0.604418 -0.219795  0.757460  1.461092 -0.189194  0.296654
1  -0.211410  1.534205  0.879773 -0.625855  0.538455  1.797247
2  -0.467350 -0.554224  0.343036 -2.138097 -1.585788 -1.007527
3  -0.577716 -1.112960  0.416870 -1.085192 -1.292922 -1.107217
4   1.686012 -0.309168  0.164862 -1.626409 -0.479827  0.299237
```

5.2.3 Imputasi Data Hilang

Imputasi data hilang adalah proses menggantikan nilai yang hilang dalam data dengan estimasi yang masuk akal. Ada beberapa metode yang bisa digunakan, seperti menggantikan data yang hilang dengan rata-rata, median, atau modus dari kolom yang bersangkutan. Berikut adalah contoh bagaimana melakukan imputasi data hilang menggunakan data sintetis:

```python
from sklearn.impute import SimpleImputer

# Membuat data sintetis dengan nilai yang hilang
data_with_missing_values = data.copy()
data_with_missing_values.iloc[0, 0] = np.nan
data_with_missing_values.iloc[1, 2] = np.nan
data_with_missing_values.iloc[5, 4] = np.nan
print("Data dengan nilai yang hilang:\n",
data_with_missing_values.head(10))

 # Imputasi data hilang dengan rata-rata
imputer = SimpleImputer(strategy='mean')
imputed_data = imputer.fit_transform(data_with_missing_values)
imputed_data = pd.DataFrame(imputed_data, columns=data.columns)
print("\nData setelah imputasi:\n", imputed_data.head(10))
```

Output:

```
Data dengan nilai yang hilang:
          X1        X2        X3        X4        X5         y
0        NaN -0.138264  0.647689  1.523030 -0.234153  0.683274
1 -0.234137  1.579213       NaN -0.469474  0.542560  3.878997
2 -0.463418 -0.465730  0.241962 -1.913280 -1.724918 -2.094161
3 -0.562288 -1.012831  0.314247 -0.908024 -1.412304 -2.306465
4  1.465649 -0.225776  0.067528 -1.424748 -0.544383  0.688775
5  0.110923 -1.150994  0.375698 -0.600639       NaN -2.434627
6 -0.601707  1.852278 -0.013497 -1.057711  0.822545  2.806653
7 -1.220844  0.208864 -1.959670 -1.328186  0.196861 -1.235112
8  0.738467  0.171368 -0.115648 -0.301104 -1.478522  1.105464
9 -0.719844 -0.460639  1.057122  0.343618 -1.763040 -2.056597

Data setelah imputasi:
          X1        X2        X3        X4        X5         y
0 -0.050216 -0.138264  0.647689  1.523030 -0.234153  0.683274
1 -0.234137  1.579213 -0.102574 -0.469474  0.542560  3.878997
2 -0.463418 -0.465730  0.241962 -1.913280 -1.724918 -2.094161
3 -0.562288 -1.012831  0.314247 -0.908024 -1.412304 -2.306465
4  1.465649 -0.225776  0.067528 -1.424748 -0.544383  0.688775
5  0.110923 -1.150994  0.375698 -0.600639 -0.029581 -2.434627
6 -0.601707  1.852278 -0.013497 -1.057711  0.822545  2.806653
7 -1.220844  0.208864 -1.959670 -1.328186  0.196861 -1.235112
8  0.738467  0.171368 -0.115648 -0.301104 -1.478522  1.105464
9 -0.719844 -0.460639  1.057122  0.343618 -1.763040 -2.056597
```

5.2.4 Deteksi dan Penanganan Outlier

Outlier adalah titik data yang jauh berbeda dari sebagian besar titik data lainnya. Outlier dapat menyebabkan masalah dalam regresi karena mereka dapat memiliki pengaruh yang tidak proporsional pada model. Ada beberapa teknik untuk mendeteksi dan menangani outlier, seperti metode IQR (interquartile range) dan Z-score. Berikut adalah contoh bagaimana mendeteksi dan menangani outlier menggunakan metode IQR:

```python
def remove_outliers_iqr(data, factor=1.5):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - factor * IQR
    upper_bound = Q3 + factor * IQR
    return data[(data > lower_bound) & (data < upper_bound)]

# Membuat data sintetis dengan outlier
data_with_outliers = data.copy()
data_with_outliers.loc[2, 'X1'] = 10

# Deteksi dan penanganan outlier
data_no_outliers = remove_outliers_iqr(data_with_outliers)
print("Data sebelum penanganan outlier:\n", data_with_outliers.head())
```

```
print("\nData setelah penanganan outlier:\n", data_no_outliers.head())
```

Output:

```
Data sebelum penanganan outlier:
           X1        X2        X3        X4        X5         y
0    0.496714 -0.138264  0.647689  1.523030 -0.234153  0.683274
1   -0.234137  1.579213  0.767435 -0.469474  0.542560  3.878997
2   10.000000 -0.465730  0.241962 -1.913280 -1.724918 -2.094161
3   -0.562288 -1.012831  0.314247 -0.908024 -1.412304 -2.306465
4    1.465649 -0.225776  0.067528 -1.424748 -0.544383  0.688775

Data setelah penanganan outlier:
          X1        X2        X3        X4        X5         y
0   0.496714 -0.138264  0.647689  1.523030 -0.234153  0.683274
1  -0.234137  1.579213  0.767435 -0.469474  0.542560  3.878997
2        NaN -0.465730  0.241962 -1.913280 -1.724918 -2.094161
3  -0.562288 -1.012831  0.314247 -0.908024 -1.412304 -2.306465
4   1.465649 -0.225776  0.067528 -1.424748 -0.544383  0.688775
```

5.2.5 Pembagian Data untuk Pelatihan dan Pengujian

Pembagian data untuk pelatihan dan pengujian adalah langkah penting dalam mempersiapkan data untuk regresi. Biasanya, data dibagi menjadi dua set: satu set untuk melatih model (data pelatihan) dan satu set untuk menguji kinerja model (data pengujian). Berikut adalah contoh bagaimana membagi data menjadi data pelatihan dan pengujian:

```
from sklearn.model_selection import train_test_split

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(data.drop('y',
axis=1), data['y'], test_size=0.3, random_state=42)

print("Ukuran data pelatihan:", X_train.shape)
print("Ukuran data pengujian:", X_test.shape)
```

Output:

```
Ukuran data pelatihan: (70, 5)
Ukuran data pengujian: (30, 5)
```

5.3 Penerapan Regresi Linier Sederhana

5.3.1 Konsep Regresi Linier Sederhana

Regresi linier sederhana adalah teknik statistik yang digunakan untuk memodelkan hubungan antara dua variabel. Dalam kasus ini, satu variabel adalah variabel dependen (y) yang ingin kita prediksi, dan variabel lainnya adalah variabel independen (x) yang

digunakan untuk memprediksi y. Tujuan dari regresi linier sederhana adalah menemukan garis terbaik yang menggambarkan hubungan antara x dan y.

Secara matematis, model regresi linier sederhana didefinisikan sebagai:

$$y = b0 + b1 * x + e$$

Di mana:

- y adalah variabel dependen
- x adalah variabel independen
- b0 adalah intersep (nilai y ketika x = 0)
- b1 adalah koefisien regresi (kemiringan garis)
- e adalah kesalahan (selisih antara nilai sebenarnya dan nilai yang diprediksi)

Tujuan dari regresi linier sederhana adalah menemukan nilai b0 dan b1 yang menghasilkan garis yang paling sesuai dengan data.

5.3.2 Menghitung Koefisien Regresi Linier Sederhana

Untuk menghitung koefisien b0 dan b1, kita dapat menggunakan metode kuadrat terkecil (least squares method), yang mencari garis yang meminimalkan jumlah kuadrat kesalahan (e) antara nilai yang diprediksi dan nilai sebenarnya. Rumusnya adalah:

$$b1 = \Sigma((xi - x\_mean) * (yi - y\_mean)) / \Sigma((xi - x\_mean)\text{^}2)$$
$$b0 = y\_mean - b1 * x\_mean$$

Di mana x_mean dan y_mean adalah rata-rata dari variabel x dan y.

5.3.3 Implementasi Regresi Linier Sederhana dengan Python

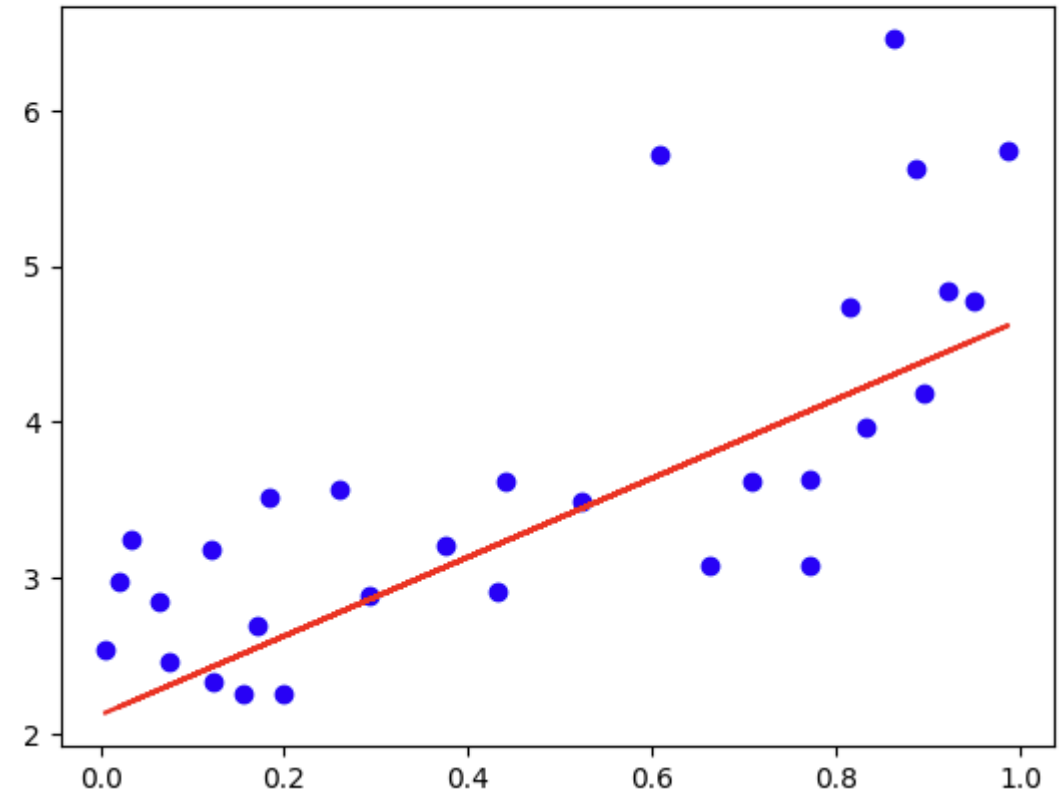
Berikut adalah contoh bagaimana mengimplementasikan regresi linier sederhana menggunakan Python dan data sintetis:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Membuat data sintetis
np.random.seed(42)
x = np.random.rand(100, 1)
y = 2 + 3 * x + np.random.randn(100, 1)
```

```
# Membagi data menjadi data pelatihan dan pengujian
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=42)

# Menerapkan regresi linier sederhana
regressor = LinearRegression()
regressor.fit(x_train, y_train)

# Memprediksi nilai y
y_pred = regressor.predict(x_test)

# Evaluasi model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R2 Score:", r2)

# Visualisasi hasil
plt.scatter(x_test, y_test, color='blue', label='Actual')
plt.plot(x_test, y_pred, color='red', label='Predicted')
```

Output:

```
Mean Squared Error: 0.6309495617838646
R2 Score: 0.4937514923042331
[<matplotlib.lines.Line2D at 0x7f0bc6406e00>]
```



5.3.4 Interpretasi Koefisien Regresi Linier Sederhana

Setelah kita memiliki koefisien b0 dan b1, kita dapat menginterpretasikan hubungan antara variabel independen dan dependen. Koefisien b1 (kemiringan garis) menggambarkan seberapa besar perubahan variabel dependen (y) ketika variabel independen (x) meningkat sebesar satu unit. Jika b1 positif, maka ada hubungan positif antara x dan y, yang berarti bahwa ketika x meningkat, y juga meningkat. Sebaliknya, jika b1 negatif, ada hubungan negatif antara x dan y, yang berarti bahwa ketika x meningkat, y menurun.

Koefisien b0 (intersep) adalah nilai variabel dependen (y) ketika variabel independen (x) sama dengan nol. Ini mungkin tidak selalu memiliki interpretasi praktis, tetapi penting untuk memprediksi nilai y pada x yang diberikan.

5.3.5 Penggunaan Model Regresi Linier Sederhana untuk Prediksi

Setelah kita memiliki model regresi linier sederhana yang telah dilatih, kita dapat menggunakannya untuk membuat prediksi pada data baru. Untuk melakukan ini, kita hanya perlu menggantikan nilai x dalam persamaan regresi dengan nilai x yang ingin kita prediksi y-nya.

```
# Contoh prediksi dengan model regresi linier sederhana
x_new = np.array([[0.45]])
y_new_pred = regressor.predict(x_new)
print("Nilai x baru:", x_new)
print("Prediksi y untuk x baru:", y_new_pred)
```

Output:

```
Nilai x baru: [[0.45]]
Prediksi y untuk x baru: [[3.2579211]]
```

Kesimpulan

Regresi linier sederhana adalah teknik yang berguna untuk memodelkan hubungan antara dua variabel dan membuat prediksi. Dalam contoh ini, kita menggunakan Python untuk mengimplementasikan regresi linier sederhana pada data sintetis. Kita juga membahas bagaimana menginterpretasikan koefisien regresi dan menggunakan model untuk membuat prediksi.
Meskipun regresi linier sederhana mudah dipahami dan diimplementasikan, terbatas pada situasi di mana hanya ada satu variabel independen. Dalam banyak kasus praktis, kita mungkin memiliki beberapa variabel independen yang mempengaruhi variabel dependen. Dalam situasi tersebut, kita dapat menggunakan regresi linier berganda, yang akan dibahas pada Subbab 5.4.

5.4 Penerapan Regresi Linier Berganda

5.4.1 Pengantar

Regresi linier berganda adalah perluasan dari regresi linier sederhana yang memungkinkan kita untuk memodelkan hubungan antara variabel dependen dan lebih dari satu variabel independen. Dalam bab ini, kita akan membahas cara mengimplementasikan regresi linier berganda menggunakan Python, serta bagaimana menginterpretasikan hasilnya dan membuat prediksi.

5.4.2 Persamaan Regresi Linier Berganda

Persamaan regresi linier berganda memiliki bentuk berikut:

$$y = b0 + b1x1 + b2x2 + ... + bn * xn + e$$

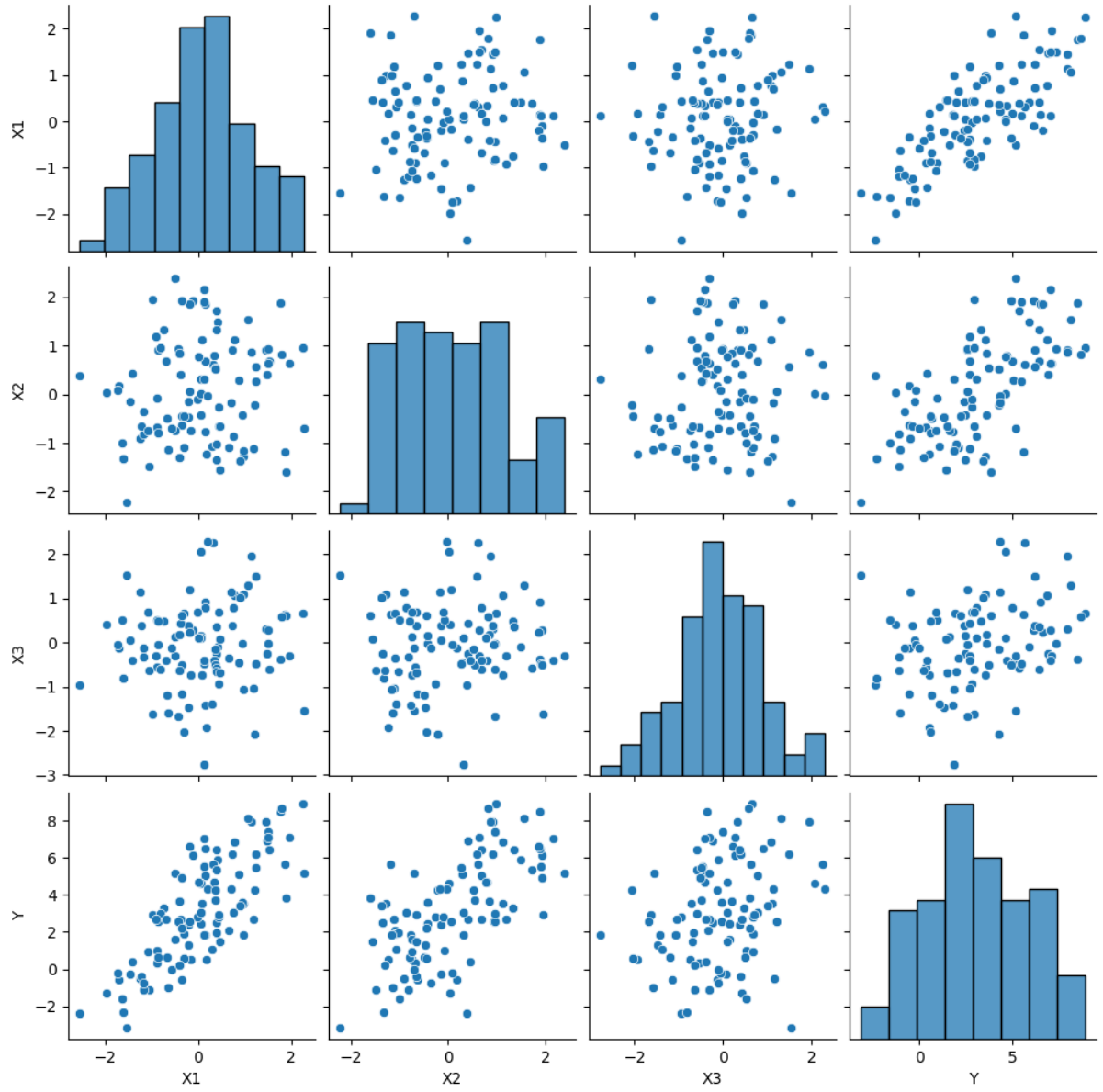
di mana y adalah variabel dependen, x1, x2, ..., xn adalah variabel independen, b0 adalah intersep, b1, b2, ..., bn adalah koefisien regresi, dan e adalah kesalahan acak.

5.4.3 Membuat Data Sintetis

Sebelum kita mulai menerapkan regresi linier berganda, mari buat data sintetis dengan beberapa variabel independen dan variabel dependen yang memiliki hubungan linier dengan variabel independen tersebut.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

np.random.seed(0)

# Membuat data sintetis
x1 = np.random.normal(0, 1, 100)
x2 = np.random.normal(0, 1, 100)
x3 = np.random.normal(0, 1, 100)

y = 3 + 2*x1 + 1.5*x2 + 0.5*x3 + np.random.normal(0, 0.5, 100)

data = pd.DataFrame({'X1': x1, 'X2': x2, 'X3': x3, 'Y': y})

sns.pairplot(data)
plt.show()
```

Output:



5.4.4 Membagi Data Menjadi Train dan Test

Mirip dengan regresi linier sederhana, kita perlu membagi data menjadi data train dan data test. Kita akan menggunakan 80% dari data untuk pelatihan dan 20% untuk pengujian.

```
X = data[['X1', 'X2', 'X3']]
y = data['Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

5.4.5 Menerapkan Regresi Linier Berganda

Untuk menerapkan regresi linier berganda, kita akan menggunakan kelas LinearRegression dari pustaka scikit-learn, sama seperti yang kita lakukan untuk regresi linier sederhana.

```
multiple_regression = LinearRegression()
multiple_regression.fit(X_train, y_train)
```

5.4.6 Interpretasi Koefisien Regresi Linier Berganda

Setelah kita melatih model regresi linier berganda, kita dapat melihat koefisien regresi untuk memahami hubungan antara variabel dependen dan independen.

```
coef_df = pd.DataFrame(multiple_regression.coef_, X.columns,
columns=['Coefficient'])
print("Intercept:", multiple_regression.intercept_)
print(coef_df)
```

Output:

```
Intercept: 2.9333177292900157
    Coefficient
X1     1.965746
X2     1.510721
X3     0.575220
```

Output ini akan menampilkan intersep dan koefisien regresi untuk setiap variabel independen dalam model. Dari hasil ini, kita dapat menginterpretasikan hubungan antara variabel dependen dan independen.

Misalnya, jika kita melihat koefisien regresi untuk X1, kita dapat mengatakan bahwa untuk setiap peningkatan satu unit dalam X1, nilai Y akan meningkat sebesar koefisien tersebut, dengan asumsi variabel independen lainnya tetap konstan. Interpretasi serupa juga berlaku untuk koefisien regresi lainnya.

5.4.7 Mengevaluasi Kinerja Model Regresi Linier Berganda

Untuk mengevaluasi kinerja model regresi linier berganda, kita akan menggunakan metrik yang sama seperti yang kita gunakan untuk regresi linier sederhana, yaitu Mean Squared Error (MSE) dan koefisien determinasi (R^2).

```
y_pred = multiple_regression.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R2 Score:", r2)
```

Output:

```
Mean Squared Error: 0.12155418700491058
R2 Score: 0.977862756086038
```

Nilai R^2 yang tinggi menunjukkan bahwa model kita dapat menjelaskan sebagian besar variasi dalam data, sementara nilai MSE yang rendah menunjukkan bahwa kesalahan prediksi model kita relatif kecil. Selalu penting untuk membandingkan kinerja model dengan kinerja model lain atau dengan model acuan untuk memastikan bahwa model kita benar-benar baik.

Kesimpulan

Dalam subbab ini, kita telah membahas cara mengimplementasikan regresi linier berganda menggunakan Python. Kita mulai dengan membuat data sintetis yang memiliki beberapa variabel independen dan variabel dependen. Selanjutnya, kita membagi data menjadi data train dan data test, melatih model regresi linier berganda, dan mengevaluasi kinerjanya.

Regresi linier berganda adalah teknik yang berguna untuk memodelkan hubungan antara beberapa variabel independen dan satu variabel dependen. Meskipun regresi linier berganda memiliki banyak kegunaan, penting untuk diingat bahwa teknik ini memiliki beberapa keterbatasan, seperti asumsi linearitas dan adanya multikolinearitas. Oleh karena itu, selalu penting untuk mengeksplorasi teknik lain, seperti regresi nonparametrik dan regresi regularisasi, yang akan kita bahas lebih lanjut dalam subbab berikutnya.

5.5 Penerapan Regresi Logistik

5.5.1 Pengantar Regresi Logistik

Regresi Logistik adalah teknik regresi yang digunakan ketika variabel dependen bersifat kategorikal. Dalam banyak kasus, variabel dependen adalah biner, yang berarti hanya ada dua kategori yang mungkin. Regresi logistik memungkinkan kita untuk memprediksi probabilitas kejadian suatu kategori tertentu berdasarkan variabel independen. Dalam subbab ini, kita akan membahas cara mengimplementasikan regresi logistik menggunakan Python.

5.5.2 Mengapa Regresi Logistik?

Regresi logistik adalah pilihan yang baik ketika ingin memodelkan hubungan antara variabel dependen biner dengan satu atau beberapa variabel independen. Beberapa contoh kasus penggunaan regresi logistik meliputi:

Memprediksi apakah seseorang akan membeli produk atau tidak berdasarkan usia, pendapatan, dan jenis kelamin.

Memprediksi apakah seorang pasien akan mengalami serangan jantung berdasarkan tekanan darah, kadar kolesterol, dan usia.

5.5.3 Persiapan Data Sintetis

Mari kita buat dataset sintetis untuk mengilustrasikan regresi logistik. Dalam contoh ini, kita akan memiliki satu variabel independen (X) dan satu variabel dependen biner (Y). Kita akan menggunakan pustaka NumPy dan scikit-learn untuk membuat dataset ini.

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification

np.random.seed(42)

X, y = make_classification(n_samples=1000, n_features=2,
n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=42)

data = pd.DataFrame(data=X, columns=["X1", "X2"])
data["Y"] = y

print(data.head())
```

Output:

```
          X1         X2  Y
0  0.601034   1.535353  1
1  0.755945 -1.172352  0
2  1.354479 -0.948528  0
3  3.103090   0.233485  0
4  0.753178   0.787514  1
```

5.5.4 Membagi Data menjadi Train dan Test Set

Seperti yang telah kita lakukan sebelumnya, kita perlu membagi data menjadi train set dan test set. Kita akan menggunakan 80% data untuk pelatihan dan 20% sisanya untuk pengujian.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5.5.5 Melatih Model Regresi Logistik

Sekarang kita akan melatih model regresi logistik menggunakan data latihan. Kita akan menggunakan pustaka scikit-learn untuk melakukannya.

```
from sklearn.linear_model import LogisticRegression

logistic_regression = LogisticRegression(random_state=42)
logistic_regression.fit(X_train, y_train)
```

5.5.6 Memprediksi Probabilitas dengan Model Regresi Logistik

Setelah melatih model, kita dapat menggunakan model ini untuk memprediksi probabilitas kategori tertentu dari variabel dependen berdasarkan variabel independen. Dalam contoh kita, kita akan memprediksi probabilitas bahwa Y = 1.

```
y_pred_proba = logistic_regression.predict_proba(X_test)

print("Predicted probabilities:")
print(y_pred_proba[:,1])
```

Output:

```
Predicted probabilities:
[0.9157771  0.00710017 0.25149346 0.01135919 0.00975021 0.99999116
 0.03827741 0.74262443 0.05553446 0.20794422 0.99246085 0.99810468
 0.08821741 0.07949559 0.99860211 0.1873917  0.29654065 0.96494183
 0.99999832 0.99999612 0.11099064 0.04294031 0.61532769 0.34136348
 0.122366   0.12547523 0.33512294 0.99988865 0.97674001 0.05291188
 0.42812903 0.70717289 0.0183893  0.98396177 0.04471833 0.03206637
 0.97935182 0.08991735 0.0291792  0.27488624 0.02878134 0.99998384
 0.92808969 0.07054694 0.0065145  0.17256529 0.31470501 0.06164494
 0.99506698 0.19951256 0.98849718 0.99130124 0.82236594 0.93142675
 0.02792557 0.99985518 0.03768262 0.24392825 0.60768166 0.35265827
 0.28266655 0.26180028 0.99364077 0.97811759 0.99859955 0.9872382
 0.72110113 0.09676562 0.94764407 0.04503025 0.42704799 0.10153939
 0.03105475 0.06685544 0.62059582 0.99866342 0.03802522 0.00778735
 0.22044923 0.01295974 0.19368535 0.00557477 0.99999148 0.99999581
 0.99966297 0.97727894 0.9987658  0.9476327  0.96412557 0.01853451
 0.16509959 0.68853844 0.98877639 0.46818002 0.31619062 0.01081594
 0.02340971 0.51807997 0.97056212 0.09493025 0.0037954  0.98180448
 0.9886232  0.00277887 0.12137699 0.83577259 0.15326532 0.98162927
 0.10535901 0.06238605 0.34640617 0.96414123 0.99819566 0.37959875
 0.98790902 0.05093044 0.00550851 0.98861809 0.02226598 0.35382061
 0.97478445 0.26662581 0.07943463 0.98930548 0.86010891 0.99972328
 0.28177467 0.00690495 0.69389976 0.0352842  0.51299149 0.06220392
 0.14709916 0.86428991 0.01756219 0.62159309 0.03502141 0.99998309
 0.99652009 0.01716258 0.04616217 0.31817666 0.4605422  0.99728739
 0.94948852 0.13365671 0.99977805 0.11386344 0.97565098 0.06348601
 0.99025726 0.98711858 0.07089171 0.20531562 0.04796338 0.91256703
 0.99809986 0.54111236 0.93042968 0.97587301 0.43052907 0.06679747
 0.75446291 0.16275953 0.00352874 0.01518928 0.00729611 0.15704219
 0.65881556 0.95611381 0.01907207 0.81706674 0.12933949 0.99997197
 0.9894458  0.99865223 0.22257977 0.72549313 0.04911039 0.01076191
 0.87312651 0.34885046 0.04433427 0.40754693 0.69476727 0.9753402
 0.95577346 0.07056172 0.01107307 0.0141977  0.99907869 0.03908921
 0.65560837 0.72734759 0.43058235 0.99371894 0.11546888 0.04813466
 0.98995285 0.99931681]
```

5.5.7 Data Preprocessing

Sebelum melanjutkan ke penyetelan hyperparameter dan evaluasi model, kita perlu memastikan data telah melalui langkah preprocessing yang benar. Beberapa langkah preprocessing umum yang perlu kita lakukan adalah:

- Penanganan nilai yang hilang
- Penskalaan fitur
- Pengkodean variabel kategorikal

Untuk contoh ini, dataset sintetis yang kita gunakan tidak memiliki nilai yang hilang, dan semua fitur sudah dalam format numerik. Namun, kita perlu penskalaan fitur untuk memastikan semua fitur berada dalam rentang yang sama. Kita akan menggunakan StandardScaler dari scikit-learn untuk melakukan penskalaan.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5.5.8 Penyetelan Hyperparameter

Regresi logistik memiliki beberapa hyperparameter yang dapat disetel untuk meningkatkan kinerja model. Salah satunya adalah parameter regularisasi, yang ditentukan oleh parameter 'C'. Kita akan menggunakan pencarian GridSearchCV dari scikit-learn untuk menemukan nilai terbaik untuk 'C'.

```
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

grid_search = GridSearchCV(LogisticRegression(random_state=42),
param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

best_C = grid_search.best_params_['C']
print("Best C:", best_C)

logistic_regression_best = LogisticRegression(C=best_C, random_state=42)
logistic_regression_best.fit(X_train_scaled, y_train)
```

5.5.9 Evaluasi Model

Sekarang kita akan mengevaluasi kinerja model regresi logistik menggunakan beberapa metrik kinerja. Kita akan membandingkan akurasi, presisi, recall, F1-score, dan ROC AUC score.

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score

y_pred = logistic_regression_best.predict(X_test_scaled)
y_pred_proba_best =
logistic_regression_best.predict_proba(X_test_scaled)

metrics = pd.DataFrame(columns=["Accuracy", "Precision", "Recall",
"F1-score", "ROC AUC score"], index=["Logistic Regression"])

metrics.loc["Logistic Regression"] = [
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred),
    recall_score(y_test, y_pred),
    f1_score(y_test, y_pred),
    roc_auc_score(y_test, y_pred_proba_best[:, 1])
]

display(metrics)
```

Output:

	Accuracy	Precision	Recall	F1-score	ROC AUC score
Logistic Regression	0.9	0.922222	0.864583	0.892473	0.945112

Dengan menggunakan contoh ini, kamu telah berhasil mengimplementasikan regresi logistik pada dataset sintetis. Kamu juga telah melakukan data preprocessing, penyetelan hyperparameter, dan evaluasi model menggunakan beberapa metrik kinerja. Dalam kasus aplikasi nyata, langkah-langkah ini akan membantu kamu membangun model yang lebih kuat dan menghasilkan prediksi yang lebih akurat.

5.6 Penerapan Regresi Polynomial

Regresi Polynomial adalah bentuk regresi linier yang digunakan ketika hubungan antara variabel independen dan dependen tidak linear. Regresi ini memodelkan hubungan dengan menambahkan variabel independen pangkat yang lebih tinggi ke dalam persamaan. Dalam subbab ini, kita akan membahas penerapan regresi polynomial pada dataset sintetis, termasuk preprocessing data, penyetelan hyperparameter, dan evaluasi model.

5.6.1 Pendahuluan

Regresi polynomial mencoba memodelkan hubungan antara variabel dependen dan independen dengan polinomial berdasarkan variabel independen. Persamaan regresi polynomial orde ke-n adalah:

$$y = b0 + b1 * x + b2 * x^2 + ... + bn * x^n$$

5.6.2 Membangun Dataset Sintetis

Kita akan membuat dataset sintetis yang memiliki hubungan non-linear antara variabel dependen dan independen menggunakan fungsi make_regression dari scikit-learn. Dataset ini akan digunakan untuk mempraktikkan penerapan regresi polynomial.

```
from sklearn.datasets import make_regression
import numpy as np
import pandas as pd

np.random.seed(42)

# Membuat dataset sintetis
X, y = make_regression(n_samples=100, n_features=1, noise=15)

# Menambahkan non-linearitas ke data
y = y ** 2

# Menampilkan data
data = pd.DataFrame(X, columns=['Feature'])
data['Target'] = y
print(data.head())
```

Output:

```
    Feature       Target
0  0.931280  3218.697479
1  0.087047   286.104554
2 -1.057711   918.246286
3  0.314247    85.222434
4 -0.479174   275.790320
```

5.6.3 Membagi Data

Membagi dataset menjadi set pelatihan dan pengujian untuk memvalidasi kinerja model.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5.6.4 Data Preprocessing

Untuk regresi polynomial, kita perlu mengkonversi fitur asli menjadi fitur polynomial. Kita akan menggunakan PolynomialFeatures dari scikit-learn untuk menghasilkan fitur polynomial dari derajat yang ditentukan.

```
from sklearn.preprocessing import PolynomialFeatures

# Menerapkan PolynomialFeatures
degree = 2
poly = PolynomialFeatures(degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

5.6.5 Melatih Model Regresi Polynomial

Kita akan melatih model regresi linier pada fitur polynomial yang telah dihasilkan.

```
from sklearn.linear_model import LinearRegression

poly_regression = LinearRegression()
poly_regression.fit(X_train_poly, y_train)
```

5.6.6 Penyetelan Hyperparameter

Untuk regresi polynomial, salah satu hyperparameter utama yang perlu kita pertimbangkan adalah derajat polinomial. Derajat yang lebih tinggi dapat meningkatkan kompleksitas model, tetapi juga dapat menyebabkan overfitting. Kita akan menggunakan validasi silang untuk menentukan derajat terbaik.

```
from sklearn.model_selection import cross_val_score

# Mencari derajat terbaik menggunakan validasi silang
best_degree = 0
best_score = -np.inf

for d in range(1, 6):
  poly = PolynomialFeatures(d)
  X_train_poly = poly.fit_transform(X_train)
  poly_regression = LinearRegression()
  scores = cross_val_score(poly_regression, X_train_poly, y_train, cv=5,
scoring='neg_mean_squared_error')
  mean_score = np.mean(scores)
  if mean_score > best_score:
```

```
    best_score = mean_score
    best_degree = d

print("Best_score: ",best_score,"\nBest Degree: ",best_degree)
```

Output:

```
Best_score:  -688061.1532495867
Best Degree:  3
```

5.6.7 Evaluasi Model

Setelah menemukan derajat terbaik, kita akan melatih model regresi polynomial dengan derajat terbaik dan mengevaluasi kinerjanya menggunakan beberapa metrik kinerja.

```
# Melatih model dengan derajat terbaik
poly = PolynomialFeatures(best_degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

best_poly_regression = LinearRegression()
best_poly_regression.fit(X_train_poly, y_train)

# Memprediksi target menggunakan model terlatih
y_train_pred = best_poly_regression.predict(X_train_poly)
y_test_pred = best_poly_regression.predict(X_test_poly)

# Menghitung metrik kinerja
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Menampilkan metrik kinerja dalam bentuk DataFrame
performance_df = pd.DataFrame({"MSE": [mse_train, mse_test],
                               "MAE": [mae_train, mae_test],
                               "R2": [r2_train, r2_test]},
                              index=["Train", "Test"])
display(performance_df)
```

Output:

	MSE	MAE	R2
Train	6.359006e+05	563.073896	0.907513
Test	2.851635e+06	1073.933102	0.489990

5.7 Penerapan Regresi Ridge dan Lasso

5.7.1 Pengantar

Pada subbab ini, kita akan membahas tentang penerapan regresi ridge dan lasso dalam regresi linier. Regresi ridge dan lasso merupakan teknik regularisasi yang bertujuan untuk mengurangi overfitting dan meningkatkan kinerja model pada data yang belum pernah dilihat sebelumnya. Regresi ridge menggunakan regularisasi L2, sedangkan regresi lasso menggunakan regularisasi L1. Dalam pembahasan ini, kita akan menggunakan contoh dataset sintetis dan menjelaskan secara detail langkah-langkah dalam penerapan regresi ridge dan lasso, termasuk preprocessing data, penyetelan hyperparameter, dan evaluasi model.

5.7.2 Membuat Dataset Sintetis

Sebelum memulai, mari kita buat dataset sintetis yang akan digunakan dalam contoh ini. Dataset ini akan memiliki 100 sampel dan 10 fitur. Target akan dihasilkan dari kombinasi linear fitur dengan beberapa noise yang ditambahkan.

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression

np.random.seed(42)

X, y = make_regression(n_samples=100, n_features=10, noise=0.1)
```

5.7.3 Membagi Data menjadi Set Pelatihan dan Pengujian

Sebelum melanjutkan ke tahap preprocessing, kita perlu membagi dataset menjadi set pelatihan dan pengujian. Kita akan menggunakan 80% data untuk pelatihan dan 20% data untuk pengujian.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5.7.4 Preprocessing Data

Karena regresi ridge dan lasso sangat sensitif terhadap skala fitur, kita perlu melakukan standarisasi fitur sebelum melatih model. Standarisasi mengubah setiap fitur sehingga memiliki rata-rata 0 dan standar deviasi 1.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5.7.5 Penerapan Regresi Ridge

Sekarang kita siap untuk menerapkan regresi ridge. Pertama, kita akan melakukan pencarian grid untuk menemukan nilai alpha terbaik. Alpha adalah parameter regularisasi yang mengontrol sejauh mana koefisien akan dikurangi. Alpha yang lebih besar akan menghasilkan koefisien yang lebih kecil, dan sebaliknya.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge = Ridge()
params = {'alpha': np.logspace(-5, 5, 100)}
grid_search = GridSearchCV(ridge, params, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

best_alpha_ridge = grid_search.best_params_['alpha']
print(f"Alpha terbaik untuk regresi ridge: {best_alpha_ridge}")
```

Output:

```
Alpha terbaik untuk regresi ridge: 1e-05
```

Setelah menemukan alpha terbaik, kita akan melatih model regresi ridge dengan alpha tersebut dan mengevaluasi kinerjanya.

```
ridge_best = Ridge(alpha=best_alpha_ridge)
ridge_best.fit(X_train, y_train)
y_pred_ridge = ridge_best.predict(X_test)

from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)
```

```
print("MSE Ridge: ",mse_ridge,"\nMAE Ridge: ",mae_ridge,"\nR2 Ridge:
",r2_ridge)
```

Output:

```
MSE Ridge:  0.01026719226416628
MAE Ridge:  0.07557670231516873
R2 Ridge:  0.9999998282077732
```

5.7.6 Penerapan Regresi Lasso

Setelah menerapkan regresi ridge, kita akan menerapkan regresi lasso. Prosesnya mirip dengan regresi ridge: kita akan mencari nilai alpha terbaik dengan pencarian grid, lalu melatih model regresi lasso dengan alpha terbaik, dan mengevaluasi kinerjanya.

```
from sklearn.linear_model import Lasso

lasso = Lasso()
params = {'alpha': np.logspace(-5, 5, 100)}
grid_search = GridSearchCV(lasso, params, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

best_alpha_lasso = grid_search.best_params_['alpha']
print(f"Alpha terbaik untuk regresi lasso: {best_alpha_lasso}")


lasso_best = Lasso(alpha=best_alpha_lasso)
lasso_best.fit(X_train, y_train)
y_pred_lasso = lasso_best.predict(X_test)

mse_lasso = mean_squared_error(y_test, y_pred_lasso)
mae_lasso = mean_absolute_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print("MSE Lasso: ",mse_lasso,"\nMAE Lasso: ",mae_lasso,"\nR2 Lasso:
",r2_lasso)
```

Output:

```
Alpha terbaik untuk regresi lasso: 1e-05
MSE Lasso:  0.01046868465396554
MAE Lasso:  0.07659917573184734
R2 Lasso:  0.9999998248363718
```

5.7.7 Evaluasi dan Perbandingan Model

Setelah melatih model regresi ridge dan lasso, kita akan mengevaluasi dan membandingkan kinerja keduanya menggunakan beberapa metrik, seperti mean squared error (MSE), mean absolute error (MAE), dan R-squared (R2).

```
performance = pd.DataFrame({"Model": ["Ridge", "Lasso"],
                            "MSE": [mse_ridge, mse_lasso],
                            "MAE": [mae_ridge, mae_lasso],
                            "R2": [r2_ridge, r2_lasso]})

performance
```

Output:

	Model	MSE	MAE	R2
0	Ridge	0.010267	0.075577	1.0
1	Lasso	0.010469	0.076599	1.0

Dengan membandingkan beberapa metrik kinerja, kita dapat memutuskan model mana yang lebih baik untuk dataset kita. Jika kinerja keduanya serupa, kita dapat memilih model yang lebih sederhana atau yang memiliki lebih sedikit fitur yang relevan. Dalam kasus ini, regresi lasso mungkin lebih disukai karena dapat menghasilkan model yang lebih jarang dengan beberapa fitur yang relevan.

5.8 Penerapan Regresi Nonparametrik

Regresi nonparametrik adalah teknik regresi yang tidak mengasumsikan bentuk fungsi yang mendasari hubungan antara variabel dependen dan independen. Metode nonparametrik umumnya lebih fleksibel daripada metode parametrik karena mereka dapat menangkap pola yang lebih kompleks dalam data. Salah satu metode regresi nonparametrik yang populer adalah Regresi Spline.

5.8.1 Regresi Spline

Regresi spline adalah metode yang membagi data menjadi beberapa segmen dan menyesuaikan polinomial terpisah untuk setiap segmen. Kelebihan metode ini adalah bahwa setiap polinomial dapat menangkap pola lokal dalam data, sehingga memungkinkan model untuk menangkap hubungan yang lebih kompleks. Untuk menghindari diskontinuitas antara segmen, batasan yang diberlakukan adalah bahwa polinomial harus bergabung pada titik-titik yang disebut knot.

5.8.2 Pemilihan Knot

Salah satu langkah kunci dalam regresi spline adalah pemilihan knot. Knot adalah titik di mana polinomial yang berbeda bergabung. Jumlah knot dan lokasi mereka dapat

mempengaruhi seberapa baik model menangkap pola dalam data. Terlalu sedikit knot dapat mengakibatkan model yang terlalu sederhana, sedangkan terlalu banyak knot dapat mengakibatkan model yang terlalu kompleks dan rentan terhadap overfitting. Salah satu pendekatan untuk pemilihan knot adalah menggunakan validasi silang.

5.8.3 Regresi Spline dengan Python

Kita akan menerapkan regresi spline pada dataset sintetis. Langkah-langkah meliputi memuat data, preprocessing, menyesuaikan model regresi spline, dan mengevaluasi kinerja model.

5.8.3.1 Memuat dan Preprocessing Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

np.random.seed(0)

# Membuat data sintetis
X = np.random.uniform(-5, 5, size=(300, 1))
y = 3 * np.sin(X[:, 0]) + np.random.normal(0, 1, size=300)

# Membagi data menjadi train dan test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Melakukan standardisasi data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5.8.3.2 Fitting Regresi Spline

Kita akan menggunakan library pygam untuk regresi spline. Pertama, kita perlu menginstal library ini:

```
pip install pygam
```

Kemudian kita dapat menyesuaikan model regresi spline dengan jumlah knot yang diberikan:

```
from pygam import LinearGAM, s
```

```
# Menentukan jumlah knot
knots = 5

# Membangun model regresi spline
gam = LinearGAM(s(0, n_splines=knots))
gam.gridsearch(X_train, y_train)
```

5.8.3.3 Evaluasi Kinerja Model

Untuk mengevaluasi kinerja model regresi spline, kita akan menggunakan beberapa metrik, seperti Mean Squared Error (MSE), Mean Absolute Error (MAE), dan R-squared. Selain itu, kita akan membandingkan kinerja model regresi spline dengan regresi linier sederhana untuk melihat apakah model nonparametrik memberikan hasil yang lebih baik.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.linear_model import LinearRegression

# Memprediksi target dengan model regresi spline
y_pred_spline = gam.predict(X_test)

# Menghitung metrik evaluasi untuk regresi spline
mse_spline = mean_squared_error(y_test, y_pred_spline)
mae_spline = mean_absolute_error(y_test, y_pred_spline)
r2_spline = r2_score(y_
test, y_pred_spline)

# Menyesuaikan model regresi linier sederhana
lr = LinearRegression()
lr.fit(X_train, y_train)

# Memprediksi target dengan model regresi linier sederhana
y_pred_lr = lr.predict(X_test)

# Menghitung metrik evaluasi untuk regresi linier sederhana
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

# Menampilkan hasil dalam bentuk dataframe
results = pd.DataFrame({"Model": ["Linear Regression", "Spline
Regression"],
```

	Model	MSE	MAE	R-squared
0	Linear Regression	5.482160	2.000227	-0.050325
1	Spline Regression	2.336848	1.296933	0.552284

```
                          "MSE": [mse_lr, mse_spline],
                          "MAE": [mae_lr, mae_spline],
                          "R-squared": [r2_lr, r2_spline]})
results
```

Output:

	Model	MSE	MAE	R-squared
0	Linear Regression	5.482160	2.000227	-0.050325
1	Spline Regression	2.336848	1.296933	0.552284

Dari hasil di atas, kita dapat melihat bahwa regresi spline menghasilkan kinerja yang lebih baik daripada regresi linier sederhana dalam hal semua metrik evaluasi yang digunakan. Hal ini menunjukkan bahwa model nonparametrik lebih cocok untuk menangkap pola yang lebih kompleks dalam data sintetis ini.

Bab 6: Evaluasi dan Penyempurnaan Model Regresi

6.1 Menggunakan Metrik Evaluasi yang Tepat

Dalam proses pembuatan model regresi, salah satu langkah penting adalah mengevaluasi kinerja model yang telah dibuat. Kinerja model regresi dapat diukur dengan beberapa metrik evaluasi. Menggunakan metrik evaluasi yang tepat sangat penting untuk memahami sejauh mana model dapat memprediksi target dengan akurat. Dalam subbab ini, kita akan membahas beberapa metrik evaluasi yang umum digunakan dalam regresi dan cara menghitungnya menggunakan Python.

Ada beberapa metrik evaluasi yang populer digunakan dalam regresi, di antaranya:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Percentage Error (MAPE)
- R-squared (R^2)

Mari kita bahas masing-masing metrik evaluasi ini secara lebih detail.

6.1.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) adalah metrik evaluasi yang menghitung rata-rata dari selisih absolut antara nilai aktual dan nilai prediksi. MAE memberikan gambaran umum tentang besarnya kesalahan tanpa memperhatikan arahnya. Nilai MAE yang lebih rendah menunjukkan kinerja model yang lebih baik.

Formula matematis untuk menghitung MAE adalah:

$$MAE = (1/n) * \Sigma|y\_i - \hat{y}\_i|$$

di mana:

- n adalah jumlah observasi
- y_i adalah nilai aktual
- ŷ_i adalah nilai prediksi

Contoh Python script untuk menghitung MAE:

```
import numpy as np

def mean_absolute_error(y_true, y_pred):
```

```
    return np.mean(np.abs(y_true - y_pred))

# Contoh data sintetis
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

mae = mean_absolute_error(y_true, y_pred)
print("MAE:", mae)
```

Output:

```
MAE: 0.5
```

6.1.2 Mean Squared Error (MSE)

Mean Squared Error (MSE) adalah metrik evaluasi yang menghitung rata-rata dari kuadrat selisih antara nilai aktual dan nilai prediksi. MSE lebih peka terhadap outlier dibandingkan dengan MAE karena kesalahan dikuadratkan. Nilai MSE yang lebih rendah menunjukkan kinerja model yang lebih baik.

Formula matematis untuk menghitung MSE adalah:

$$MSE = (1/n) * \Sigma(y\_i - \hat{y}\_i)^2$$

di mana:

- n adalah jumlah observasi
- y_i adalah nilai aktual
- ŷ_i adalah nilai prediksi

Contoh Python script untuk menghitung MSE:

```
import numpy as np

def mean_squared_error(y_true, y_pred):
    return np.mean(np.square(y_true - y_pred))

# Contoh data sintetis
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

mse = mean_squared_error(y_true, y_pred)
print("MSE:", mse)
```

Output:

```
MSE: 0.375
```

6.1.3 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) adalah metrik evaluasi yang menghitung akar kuadrat dari rata-rata kuadrat selisih antara nilai aktual dan nilai prediksi. RMSE sering digunakan dalam evaluasi model regresi karena memiliki unit yang sama dengan variabel target, sehingga lebih mudah untuk diinterpretasikan. Nilai RMSE yang lebih rendah menunjukkan kinerja model yang lebih baik.

Formula matematis untuk menghitung RMSE adalah:

$$RMSE = \sqrt{(1/n)} * \Sigma(y\_i - \hat{y}\_i)\text{^}2$$

di mana:

- n adalah jumlah observasi
- y_i adalah nilai aktual
- ŷ_i adalah nilai prediksi

Contoh Python script untuk menghitung RMSE:

```
import numpy as np

def root_mean_squared_error(y_true, y_pred):
    return np.sqrt(np.mean(np.square(y_true - y_pred)))

# Contoh data sintetis
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

rmse = root_mean_squared_error(y_true, y_pred)
print("RMSE:", rmse)
```

Output:

```
RMSE: 0.6123724356957945
```

6.1.4 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) adalah metrik evaluasi yang menghitung rata-rata dari selisih absolut antara nilai aktual dan nilai prediksi dalam persentase. MAPE berguna untuk mengukur akurasi model dalam skala yang sama dengan target. Nilai MAPE yang lebih rendah menunjukkan kinerja model yang lebih baik.

Formula matematis untuk menghitung MAPE adalah:

$$MAPE = (100/n) * \Sigma(|y\_i - \hat{y}\_i| / y\_i)$$

di mana:

- n adalah jumlah observasi
- y_i adalah nilai aktual
- ŷ_i adalah nilai prediksi

Contoh Python script untuk menghitung MAPE:

```
import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Contoh data sintetis
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

mape = mean_absolute_percentage_error(y_true, y_pred)
print("MAPE:", mape)
```

Output:

```
MAPE: 32.73809523809524
```

6.1.5 R-squared (R^2)

R-squared (R^2) adalah metrik evaluasi yang mengukur sejauh mana variasi dalam variabel target dapat dijelaskan oleh model regresi. R^2 berkisar antara 0 hingga 1, dengan nilai yang lebih tinggi menunjukkan kinerja model yang lebih baik.

Formula matematis untuk menghitung R^2 adalah:

$$R^2 = 1 - (\Sigma(y\_i - \hat{y}\_i)^2) / (\Sigma(y\_i - \bar{y})^2)$$

di mana:

- n adalah jumlah observasi
- y_i adalah nilai aktual
- ŷ_i adalah nilai prediksi
- ȳ adalah rata-rata dari nilai aktual

Contoh Python script untuk menghitung R^2:

```
import numpy as np

def r_squared(y_true, y_pred):
    ss_res = np.sum(np.square(y_true - y_pred))
    ss_tot = np.sum(np.square(y_true - np.mean(y_true)))
```

```
    return 1 - (ss_res / ss_tot)

 # Contoh data sintetis
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

r2 = r_squared(y_true, y_pred)
print("R^2:", r2)
```

Output:

```
R^2: 0.9486081370449679
```

Setelah memahami metrik evaluasi yang berbeda, penting juga untuk mengetahui cara menggabungkannya dalam skenario evaluasi model yang lebih realistis. Berikut adalah contoh cara menggabungkan beberapa metrik evaluasi menggunakan Python.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Membuat data sintetis
np.random.seed(42)
X = np.random.rand(100, 1)
y = 2 + 3 * X + np.random.randn(100, 1)

# Membagi data menjadi training dan testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Membuat dan melatih model regresi linier
lr = LinearRegression()
lr.fit(X_train, y_train)

# Memprediksi target dengan model
y_pred = lr.predict(X_test)

# Menghitung metrik evaluasi
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("MAE:", mae)
```

```
print("MSE:", mse)
print("RMSE:", rmse)
print("R^2:", r2)
```

Output:

```
MAE: 0.5973395521654093
MSE: 0.6309495617838646
RMSE: 0.7943233357920845
R^2: 0.4937514923042331
```

Berikut adalah rangkuman yang menunjukkan kapan harus menggunakan setiap metrik evaluasi dalam regresi:

Metrik	Kapan Harus Digunakan
Mean Absolute Error (MAE)	Ketika kamu ingin mengukur kesalahan rata-rata tanpa memperhatikan arah (positif atau negatif) kesalahan.
Mean Squared Error (MSE)	Ketika kamu ingin mengukur kesalahan rata-rata dengan memberi penekanan lebih pada kesalahan yang lebih besar karena nilai dikuadratkan.
Root Mean Squared Error (RMSE)	Ketika kamu ingin mengukur kesalahan rata-rata dengan skala yang sama dengan target. Biasa digunakan ketika kesalahan besar lebih penting.
R-squared (R^2)	Ketika kamu ingin mengukur seberapa baik model regresi menjelaskan variabilitas data, atau seberapa "fit" model tersebut terhadap data.
Adjusted R-squared (Adj. R^2)	Ketika kamu ingin mengukur seberapa baik model regresi menjelaskan variabilitas data, sambil mempertimbangkan jumlah fitur dalam model.

Dalam praktiknya, kamu mungkin akan menggunakan beberapa metrik evaluasi untuk memberikan gambaran yang lebih lengkap tentang kinerja model regresi. Biasanya, RMSE dan R^2 adalah metrik yang paling umum digunakan karena mereka memberikan informasi yang berguna tentang kesalahan prediksi dan kecocokan model secara keseluruhan. Namun, sebaiknya selalu pertimbangkan konteks masalah yang sedang kamu kerjakan dan pilih metrik yang paling sesuai untuk tujuan spesifik tersebut.

6.2 Validasi Silang untuk Estimasi Performa

Dalam membangun model regresi, penting untuk mengestimasi seberapa baik model akan bekerja pada data yang belum pernah dilihat sebelumnya. Salah satu teknik yang populer dan efektif untuk mengestimasi performa model pada data baru adalah validasi silang (cross-validation). Dalam subbab ini, kita akan membahas validasi silang secara detail, termasuk konsep dasar, cara kerja, dan bagaimana menggunakannya dalam Python.

6.2.1 Konsep Dasar Validasi Silang

Validasi silang adalah teknik yang digunakan untuk mengestimasi kinerja suatu model dengan membagi data menjadi beberapa bagian dan menggunakan sebagian data untuk melatih model dan sebagian lainnya untuk menguji model. Proses ini dilakukan beberapa kali, dan hasilnya kemudian dirata-rata untuk menghasilkan estimasi yang lebih akurat tentang kinerja model. Validasi silang membantu mengatasi masalah overfitting dan underfitting serta memastikan model yang kita bangun dapat diterapkan pada data yang belum pernah dilihat sebelumnya.

6.2.2 Jenis-jenis Validasi Silang

Ada beberapa jenis validasi silang yang dapat digunakan, di antaranya:

- k-Fold Cross Validation: Data dibagi menjadi k bagian (fold) yang sama besar. Proses pelatihan dan pengujian dilakukan sebanyak k kali, dengan setiap kali menggunakan fold yang berbeda sebagai data pengujian dan sisanya sebagai data pelatihan. Hasil dari setiap iterasi kemudian dirata-rata untuk menghasilkan estimasi akhir kinerja model.
- Stratified k-Fold Cross Validation: Mirip dengan k-Fold Cross Validation, tetapi setiap fold dibuat dengan mempertahankan proporsi kelas target yang sama seperti dalam data asli. Ini sangat berguna ketika bekerja dengan data yang tidak seimbang.
- Leave-One-Out Cross Validation (LOOCV): Ini adalah kasus khusus dari k-Fold Cross Validation di mana k sama dengan jumlah sampel dalam dataset. Setiap kali, satu sampel digunakan sebagai data pengujian dan sisanya sebagai data pelatihan. Ini menghasilkan estimasi yang sangat akurat tetapi membutuhkan waktu yang lama untuk dihitung.
- Leave-p-Out Cross Validation (LpOCV): Mirip dengan LOOCV, tetapi menggunakan p sampel sebagai data pengujian setiap kali.

6.2.3 Validasi Silang dalam Python

Mari kita lihat contoh penggunaan validasi silang dalam Python menggunakan data sintetis:

```
import numpy as np
import pandas as pd

np.random.seed(42)

# Buat data sintetis
X = np.random.rand(100, 5)
y = 3*X[:, 0] - 4*X[:, 1] + 5*X[:, 2] + np.random.normal(0, 0.5, 100)

# Ubah ke dataframe
data = pd.DataFrame(X, columns=['X1', 'X2', 'X3', 'X4', 'X5'])
data['y'] = y
```

Siapkan model regresi linier:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

Terapkan k-Fold Cross Validation:

```
from sklearn.model_selection import cross_val_score

# Tentukan jumlah fold
k = 5

# Gunakan cross_val_score untuk menghitung skor validasi silang (R^2)
untuk setiap fold
scores = cross_val_score(model, X, y, cv=k)

# Cetak skor validasi silang
print(f'Skor validasi silang untuk setiap fold: {scores}')
print(f'Rata-rata skor validasi silang: {scores.mean()}')
```

Output:

```
Skor validasi silang untuk setiap fold: [0.91503299 0.89956102 0.915076   0.94404356 0.93054142]
Rata-rata skor validasi silang: 0.9208509968136556
```

Kita juga bisa menggunakan validasi silang dengan mean squared error (MSE) sebagai alternatif dari R^2. Pertama, kita akan mengimpor pustaka yang diperlukan dan membuat data sintetis seperti sebelumnya.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, make_scorer

# Membuat data sintetis
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 + 3 * X.squeeze() + np.random.randn(100)

# Membuat model regresi linier
model = LinearRegression()
```

Selanjutnya, kita akan menggunakan fungsi cross_val_score untuk menghitung skor validasi silang dengan metrik MSE. Untuk melakukannya, kita perlu membuat custom scoring function dengan bantuan make_scorer dari modul sklearn.metrics.

```
# Tentukan jumlah fold
k = 5

# Buat custom scoring function dengan mean squared error
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Gunakan cross_val_score untuk menghitung skor validasi silang (MSE)
untuk setiap fold
scores = cross_val_score(model, X, y, cv=k, scoring=mse_scorer)

# Karena MSE menggunakan 'greater_is_better=False', skor MSE akan
negatif.
# Kita perlu mengubah tanda skor menjadi positif.
mse_scores = -scores

# Cetak skor validasi silang (MSE)
print(f'Skor validasi silang (MSE) untuk setiap fold: {mse_scores}')
print(f'Rata-rata skor validasi silang (MSE): {mse_scores.mean()}')
```

Output:

```
Skor validasi silang (MSE) untuk setiap fold: [0.95236313 1.55991735 0.62052584 1.19069233 0.90008206]
Rata-rata skor validasi silang (MSE): 1.0447161410936832
```

Dalam contoh ini, kita menggunakan validasi silang k-Fold dengan k = 5 pada model regresi linier yang dibangun menggunakan data sintetis. Fungsi cross_val_score dari sklearn.model_selection digunakan untuk menghitung skor validasi silang (dalam hal ini, mean squared error) untuk setiap fold. Selanjutnya, kita mencetak skor validasi silang (MSE) untuk setiap fold dan rata-rata skor validasi silang (MSE).

6.3 Pemilihan Model dengan Hyperparameter Tuning

Pemilihan model merupakan tahap penting dalam proses membangun model regresi yang baik. Salah satu cara untuk meningkatkan kinerja model adalah dengan melakukan tuning hyperparameter. Hyperparameter adalah parameter yang tidak diestimasi dari data pelatihan, tetapi ditentukan sebelum proses pelatihan dimulai. Dalam bab ini, kita akan membahas cara melakukan hyperparameter tuning untuk model regresi.

Pengenalan Hyperparameter Tuning

Hyperparameter tuning bertujuan untuk mencari kombinasi hyperparameter yang optimal agar model dapat memberikan performa terbaik. Ada beberapa metode yang umum digunakan dalam hyperparameter tuning, seperti:
a. Grid Search
b. Random Search
c. Bayesian Optimization

6.3.1 Grid Search

Grid search merupakan metode tuning hyperparameter yang paling sederhana. Grid search mencoba setiap kombinasi hyperparameter yang mungkin dan memilih kombinasi yang memberikan performa terbaik. Grid search dapat digunakan untuk model regresi dengan jumlah hyperparameter yang relatif kecil, karena proses ini dapat menjadi sangat komputasi intensif jika ada banyak hyperparameter untuk dioptimalkan.

Berikut adalah contoh penggunaan Grid Search untuk hyperparameter tuning pada model Ridge Regression:

```
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, make_scorer

# Membuat data sintetis
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 + 3 * X.squeeze() + np.random.randn(100)

# Membuat model Ridge Regression
ridge_model = Ridge()

# Tentukan rentang nilai alpha untuk dicoba
alphas = np.logspace(-3, 3, 7)

# Buat custom scoring function dengan mean squared error
```

```
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Membuat objek GridSearchCV untuk mencari alpha terbaik
grid = GridSearchCV(estimator=ridge_model, param_grid={'alpha': alphas},
scoring=mse_scorer, cv=5)

# Melakukan Grid Search
grid.fit(X, y)

# Menampilkan alpha terbaik dan skor terbaik
best_alpha = grid.best_params_['alpha']
best_score = -grid.best_score_
print(f'Alpha terbaik: {best_alpha}')
print(f'Skor MSE terbaik: {best_score}')
```

Output:

```
Alpha terbaik: 0.001
Skor MSE terbaik: 1.0447223121225393
```

6.3.2 Random Search

Random search merupakan alternatif lain untuk grid search. Alih-alih mencoba setiap kombinasi hyperparameter yang mungkin, random search mencoba sejumlah kombinasi acak dan memilih kombinasi yang memberikan performa terbaik. Random search lebih efisien secara komputasi dibandingkan grid search, terutama jika ada banyak hyperparameter yang perlu dioptimalkan.

Berikut adalah contoh penggunaan Random Search untuk hyperparameter tuning pada model Ridge Regression:

```
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, make_scorer

# Membuat data sintetis
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 + 3 * X.squeeze() + np.random.randn(100)

# Membuat model Ridge Regression
ridge_model = Ridge()

# Tentukan rentang nilai alpha untuk dicoba
```

```
alphas = np.logspace(-3, 3, 1000)

# Buat custom scoring function dengan mean squared error
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Membuat objek RandomizedSearchCV untuk mencari alpha terbaik
random_search = RandomizedSearchCV(estimator=ridge_model,
param_distributions={'alpha': alphas}, n_iter=100, scoring=mse_scorer,
cv=5, random_state=42)

# Melakukan Random Search
random_search.fit(X, y)

# Menampilkan alpha terbaik dan skor terbaik
best_alpha = random_search.best_params_['alpha']
best_score = -random_search.best_score_
print(f'Alpha terbaik: {best_alpha}')
print(f'Skor MSE terbaik: {best_score}')
```

Output:

```
Alpha terbaik: 0.0011483124145435111
Skor MSE terbaik: 1.0447232301750877
```

6.3.3 Bayesian Optimization

Bayesian optimization merupakan metode yang lebih canggih untuk hyperparameter tuning. Metode ini menggunakan proses Gaussian untuk memperkirakan fungsi yang menggambarkan hubungan antara hyperparameter dan metrik evaluasi. Selanjutnya, proses ini mencari hyperparameter yang memberikan performa terbaik berdasarkan perkiraan tersebut.

Berikut adalah contoh penggunaan Bayesian Optimization untuk hyperparameter tuning pada model Ridge Regression dengan library `scikit-optimize`:

```
!pip3 install scikit-optimize

import numpy as np
from sklearn.linear_model import Ridge
from skopt import BayesSearchCV
from sklearn.metrics import mean_squared_error, make_scorer

# Membuat data sintetis
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 + 3 * X.squeeze() + np.random.randn(100)
```

```
# Membuat model Ridge Regression
ridge_model = Ridge()

# Tentukan rentang nilai alpha untuk dicoba
alphas = (1e-3, 1e3, 'log-uniform')

# Buat custom scoring function dengan mean squared error
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Membuat objek BayesSearchCV untuk mencari alpha terbaik
bayes_search = BayesSearchCV(estimator=ridge_model,
search_spaces={'alpha': alphas}, n_iter=100, scoring=mse_scorer, cv=5,
n_jobs=-1, random_state=42)

# Melakukan Bayesian Optimization
bayes_search.fit(X, y)

# Menampilkan alpha terbaik dan skor terbaik
best_alpha = bayes_search.best_params_['alpha']
best_score = -bayes_search.best_score_
print(f'Alpha terbaik: {best_alpha}')
print(f'Skor MSE terbaik: {best_score}')
```

Output:

```
Alpha terbaik: 0.001
Skor MSE terbaik: 1.0447223121225393
```

Berikut adalah kelebihan dan kekurangan dari metode hyperparameter tuning yang telah dijelaskan sebelumnya:

Metode	Kelebihan	Kekurangan
Grid Search	1. Mudah untuk diimplementasikan	1. Membutuhkan waktu komputasi yang lama, terutama jika rentang pencarian luas

	2. Hasilnya deterministik dan dapat direplikasi	2. Tidak efisien, karena mencoba semua kombinasi hyperparameter yang mungkin
		3. Rentan terhadap "curse of dimensionality" jika terdapat banyak hyperparameter untuk di-tune
Random Search	1. Lebih cepat daripada Grid Search	1. Hasilnya non-deterministik dan mungkin bervariasi setiap kali dijalankan
	2. Dapat menemukan solusi yang lebih baik daripada Grid Search dalam waktu yang lebih singkat	2. Tidak menjamin solusi optimal
	3. Lebih efisien daripada Grid Search dalam kasus banyak hyperparameter	
Bayesian Optimization	1. Lebih efisien daripada Grid Search dan Random Search	1. Implementasinya lebih kompleks

	2. Dapat menemukan solusi yang lebih baik dengan jumlah iterasi yang lebih sedikit	2. Memerlukan lebih banyak pengetahuan tentang metode-metode optimasi probabilistik
	3. Menggunakan informasi dari iterasi sebelumnya untuk mempercepat proses pencarian	3. Mungkin memerlukan waktu komputasi yang lebih lama per iterasi daripada metode lain

6.4 Analisis Residu dan Diagnostik Model

Setelah kamu menemukan model regresi terbaik dengan hyperparameter yang telah dioptimalkan, langkah selanjutnya adalah menganalisis kinerja model secara lebih mendalam. Salah satu cara untuk melakukannya adalah dengan menganalisis residu, yakni perbedaan antara nilai yang dihasilkan oleh model dan nilai sebenarnya. Analisis residu dapat membantu kamu mengidentifikasi pola atau masalah dalam model yang mungkin perlu diperbaiki.

6.4.1 Konsep Residu dan Menghitungnya

Residu adalah perbedaan antara nilai yang dihasilkan oleh model regresi (nilai prediksi) dan nilai sebenarnya (nilai observasi). Residu dapat dihitung sebagai:

$$Residu = y\_true - y\_pred$$

Di mana y_true adalah nilai observasi dan y_pred adalah nilai prediksi yang dihasilkan oleh model regresi.

Untuk menghitung residu dalam Python, kamu bisa menggunakan kode berikut:

```
import numpy as np

y_true = np.array([4, 6, 8, 10, 12])
y_pred = np.array([3, 6.5, 7.5, 9.5, 11.5])

residu = y_true - y_pred
print(residu)
```

Output:

```
[ 1.  -0.5  0.5  0.5  0.5]
```

6.4.2 Teknik-teknik Visualisasi Residu

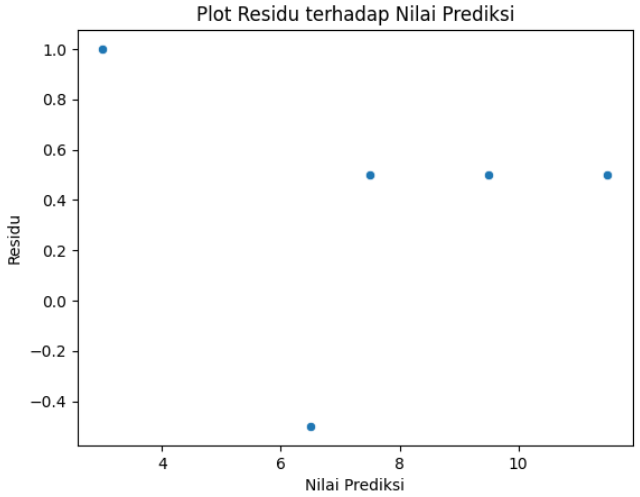
Visualisasi residu dapat membantu kamu mengidentifikasi pola atau masalah dalam model. Beberapa teknik visualisasi residu yang umum digunakan adalah:

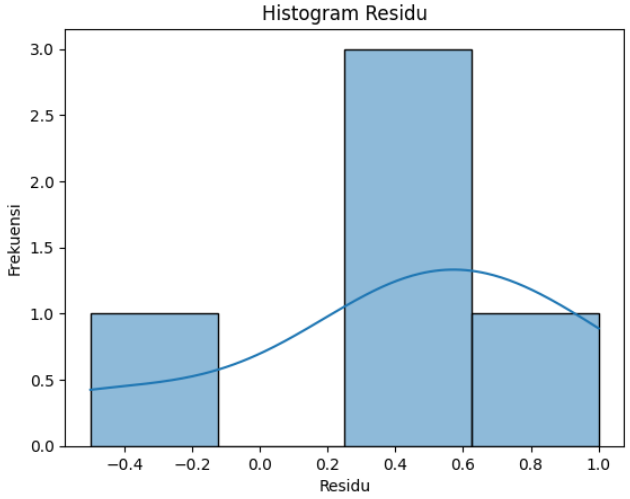
- Plot residu terhadap nilai prediksi (Residual Plot)
- Plot residu terhadap variabel independen (Residual vs. Predictor Plot)
- Histogram residu (Residual Histogram)
- QQ Plot (Quantile-Quantile Plot)

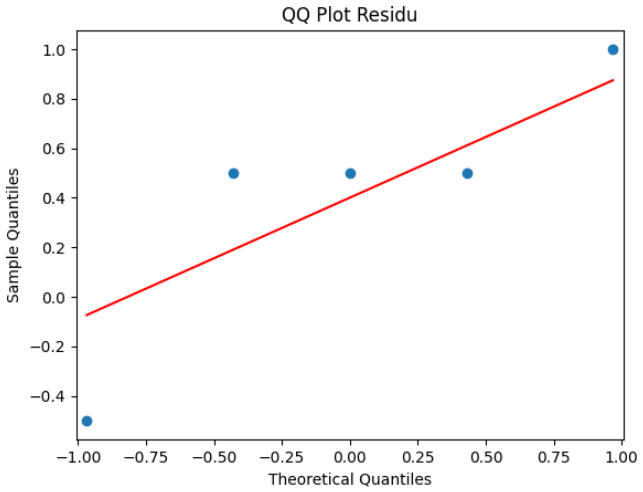
Berikut adalah contoh cara membuat plot residu menggunakan matplotlib dan seaborn:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Residual Plot
plt.figure()
sns.scatterplot(x=y_pred, y=residu)
plt.xlabel('Nilai Prediksi')
plt.ylabel('Residu')
plt.title('Plot Residu terhadap Nilai Prediksi')
plt.show()

# Residual Histogram
plt.figure()
sns.histplot(residu, kde=True)
plt.xlabel('Residu')
plt.ylabel('Frekuensi')
plt.title('Histogram Residu')
plt.show()

# QQ Plot
import statsmodels.api as sm

plt.figure()
sm.qqplot(residu, line='s')
plt.title('QQ Plot Residu')
plt.show()
```

Output:







6.4.3 Analisis Pola Residu

Setelah membuat plot residu, periksa pola yang muncul dalam plot tersebut. Jika residu menyebar secara acak, hal ini mengindikasikan bahwa model telah berhasil menangkap pola dalam data. Namun, jika residu menunjukkan pola tertentu (misalnya, pola berbentuk parabola), ini mengindikasikan bahwa model belum sepenuhnya menangkap pola dalam

data, dan kamu mungkin perlu mencoba model yang lebih kompleks atau menambahkan variabel baru ke dalam model.

Beberapa pola umum yang dapat kamu temukan dalam plot residu adalah:

- **Pola berbentuk parabola:** Ini mengindikasikan bahwa model mungkin kurang kompleks atau tidak mampu menangkap efek non-linear dalam data. Solusinya adalah mencoba model yang lebih kompleks, seperti regresi polinomial atau regresi nonparametrik.
- **Pola heteroskedastisitas:** Jika residu menunjukkan pola heteroskedastisitas (varians residu tidak konstan sepanjang rentang nilai prediksi), maka asumsi homoskedastisitas tidak terpenuhi. Kamu bisa mencoba menggabungkan teknik transformasi data atau menggunakan model yang lebih fleksibel untuk mengatasi heteroskedastisitas.
- **Pola otokorelasi:** Jika residu menunjukkan pola otokorelasi (ketergantungan antara residu pada titik waktu yang berdekatan), maka asumsi independensi residu tidak terpenuhi. Hal ini sering terjadi pada data time series. Kamu mungkin perlu menggabungkan teknik analisis time series atau menggabungkan informasi waktu ke dalam model regresi.

6.4.4 Uji Statistik untuk Asumsi Model Regresi

Selain analisis residu, kamu juga dapat melakukan uji statistik untuk menilai apakah asumsi yang mendasari model regresi terpenuhi. Beberapa uji yang umum digunakan meliputi:

Uji Durbin-Watson: Uji ini digunakan untuk mendeteksi adanya otokorelasi dalam residu. Nilai Durbin-Watson yang dekat dengan 2 menunjukkan tidak ada otokorelasi, sementara nilai yang jauh dari 2 mengindikasikan adanya otokorelasi.

Uji Shapiro-Wilk: Uji ini digunakan untuk menguji normalitas residu. Jika p-value dari uji Shapiro-Wilk kurang dari tingkat signifikansi yang ditentukan (misalnya, 0,05), maka hipotesis nol bahwa residu berdistribusi normal ditolak.

Uji Breusch-Pagan: Uji ini digunakan untuk menguji heteroskedastisitas dalam residu. Jika p-value dari uji Breusch-Pagan kurang dari tingkat signifikansi yang ditentukan, maka hipotesis nol bahwa residu homoskedastik ditolak.

Contoh penggunaan uji statistik tersebut dalam Python:

```
import statsmodels.stats.api as sms

# Uji Durbin-Watson
durbin_watson_stat = sms.durbin_watson(residu)
print('Durbin-Watson:', durbin_watson_stat)

# Uji Shapiro-Wilk
```

```
from scipy.stats import shapiro

shapiro_stat, shapiro_p_value = shapiro(residu)
print('Shapiro-Wilk:', shapiro_stat, 'p-value:', shapiro_p_value)

# Uji Breusch-Pagan
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

names = ['Lagrange multiplier statistic', 'p-value', 'f-value', 'f
p-value']
breusch_pagan_test = sms.het_breuschpagan(residu, y_pred.reshape(-1, 1))
lzip(names, breusch_pagan_test)
print('Breusch-Pagan:', lzip(names, breusch_pagan_test))
```

Output:

```
Durbin-Watson: 1.625
Shapiro-Wilk: 0.8282726407051086 p-value: 0.135022833943367
Breusch-Pagan: [('Lagrange multiplier statistic', 1.6734848484848486), ('p-value', nan), ('f-value', 2.0122978820314277), ('f p-value', 0.22902110880742585)]
```

Dengan menggunakan analisis residu dan uji statistik, kamu dapat lebih memahami kinerja model regresi dan mengidentifikasi masalah yang mungkin mempengaruhi hasil prediksi. Melakukan analisis residu dan uji statistik ini akan membantu kamu meningkatkan model regresi sehingga lebih akurat dan efisien dalam memprediksi data baru.

Ringkasan

Dalam Subbab 6.4 ini, kita telah membahas cara melakukan analisis residu dan diagnostik model untuk mengevaluasi kinerja model regresi. Kita telah mempelajari konsep residu, teknik-teknik visualisasi residu, analisis pola residu, serta uji statistik yang digunakan untuk menguji asumsi model regresi. Dengan menerapkan teknik-teknik ini, kamu dapat lebih memahami bagaimana model regresi bekerja dengan data dan mengidentifikasi masalah yang mungkin mempengaruhi hasil prediksi. Selanjutnya, kamu dapat melakukan penyempurnaan model untuk meningkatkan akurasi dan efisiensi prediksi pada data baru.

Bab 7: Studi Kasus Regresi

7.1 Prediksi Harga Rumah

7.1.1 Latar Belakang dan Tujuan Kasus

Salah satu aplikasi regresi yang paling umum dan menarik adalah prediksi harga rumah. Dalam kasus ini, kita ingin mengembangkan model regresi yang dapat memprediksi harga rumah berdasarkan fitur-fitur yang relevan, seperti luas tanah, jumlah kamar tidur, jumlah kamar mandi, dan sebagainya. Tujuan dari studi kasus ini adalah untuk memahami bagaimana kita dapat menerapkan teknik regresi yang telah kita pelajari sejauh ini untuk memecahkan masalah dunia nyata dan menghasilkan prediksi yang akurat.

7.1.2 Contoh Script Python dengan Data Sintetis

Pertama, mari kita buat data sintetis untuk studi kasus ini. Kita akan menggunakan pustaka Python numpy dan pandas untuk menghasilkan data yang mirip dengan data riil. Selanjutnya, kita akan menggunakan pustaka scikit-learn untuk melatih model regresi, mengoptimalkan hyperparameter, dan mengevaluasi kinerja model.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Membuat data sintetis
np.random.seed(0)
n_samples = 500
luas_tanah = np.random.randint(500, 2000, n_samples)
jumlah_kamar_tidur = np.random.randint(1, 6, n_samples)
jumlah_kamar_mandi = np.random.randint(1, 4, n_samples)
harga_rumah = 100000 + luas_tanah * 100 + jumlah_kamar_tidur * 5000 +
jumlah_kamar_mandi * 10000

# Membuat DataFrame dengan pandas
data = pd.DataFrame({'luas_tanah': luas_tanah,
                     'jumlah_kamar_tidur': jumlah_kamar_tidur,
                     'jumlah_kamar_mandi': jumlah_kamar_mandi,
                     'harga_rumah': harga_rumah})

# Membagi data menjadi fitur (X) dan target (y)
X = data.drop('harga_rumah', axis=1)
y = data['harga_rumah']
```

```
# Membagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Melatih model regresi linier
reg = LinearRegression()
reg.fit(X_train, y_train)

# Memprediksi harga rumah pada data uji
y_pred = reg.predict(X_test)

# Menampilkan hasil evaluasi model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
```

Output:

```
Mean Squared Error: 4.153136625703385e-16
Mean Absolute Error: 1.775420969352126e-08
R2 Score: 1.0
```

7.1.3 Preprocessing Data

Dalam contoh di atas, kita tidak perlu melakukan preprocessing data karena kita telah menggunakan data sintetis yang bersih. Namun, dalam praktiknya, kamu mungkin perlu melakukan langkah-langkah preprocessing seperti mengisi nilai yang hilang dan menskalakan fitur. Berikut ini adalah contoh bagaimana kamu dapat melakukan preprocessing data menggunakan pustaka scikit-learn:

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Contoh mengisi nilai yang hilang pada fitur numerik
numerical_imputer = SimpleImputer(strategy='mean')

# Contoh menskalakan fitur numerik
scaler = StandardScaler()

# Menggabungkan semua langkah preprocessing dalam satu transformer
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_imputer, ['luas_tanah', 'jumlah_kamar_tidur',
'jumlah_kamar_mandi']),
        ('scale', scaler, ['luas_tanah'])
    ])

# Membuat pipeline yang menggabungkan preprocessing dan model regresi
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', LinearRegression())])

# Melatih model dengan pipeline
pipeline.fit(X_train, y_train)

# Memprediksi harga rumah pada data uji
y_pred = pipeline.predict(X_test)
```

7.1.4 Model Training, Hyperparameter Tuning, dan Evaluasi

Dalam contoh awal, kita telah melatih model regresi linier dan mengevaluasi kinerja model menggunakan beberapa metrik. Namun, kita dapat mencoba model regresi lain atau mengoptimalkan hyperparameter untuk meningkatkan kinerja model. Berikut ini adalah contoh bagaimana kamu dapat menggunakan GridSearchCV untuk mencari hyperparameter terbaik dari model regresi ridge:

```
from sklearn.linear_model import Ridge

# Menyiapkan parameter grid untuk pencarian hyperparameter
param_grid = {'alpha': [0.1, 1, 10, 100]}

# Menggunakan GridSearchCV untuk mencari hyperparameter terbaik
grid_search = GridSearchCV(Ridge(), param_grid,
scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)

# Menampilkan hyperparameter terbaik
print("Best hyperparameters:", grid_search.best_params_)

# Melatih model regresi ridge dengan hyperparameter terbaik
best_ridge = grid_search.best_estimator_
best_ridge.fit(X_train, y_train)

# Memprediksi harga rumah pada data uji
y_pred = best_ridge.predict(X_test)
```

```
# Menampilkan hasil evaluasi model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
```

Output:

```
Best hyperparameters: {'alpha': 0.1}
Mean Squared Error: 9.918945319175242
Mean Absolute Error: 2.7347682216670366
R2 Score: 0.9999999947923738
```

7.2 Prediksi Konsumsi Energi

7.2.1 Latar Belakang dan Objektif Kasus

Konsumsi energi merupakan faktor penting dalam kehidupan sehari-hari, terutama dalam konteks industri dan perumahan. Prediksi konsumsi energi yang akurat sangat berguna bagi perusahaan listrik, pemerintah, dan pengguna akhir untuk mengelola alokasi sumber daya dan menjaga keberlanjutan lingkungan. Dalam subbab ini, kita akan membahas studi kasus yang bertujuan untuk memprediksi konsumsi energi menggunakan regresi berbasis pohon, yaitu Decision Tree Regressor.

Untuk kasus ini, kita akan menggunakan data sintetis yang mencakup fitur-fitur seperti suhu, kelembaban, dan kecepatan angin. Berikut adalah contoh script Python untuk membuat data sintetis:

```
import numpy as np
import pandas as pd

# Membuat data sintetis
np.random.seed(42)
n_samples = 1000
temperature = np.random.normal(25, 5, n_samples)
humidity = np.random.normal(60, 10, n_samples)
wind_speed = np.random.normal(10, 2, n_samples)
energy_consumption = 1000 + 10 * temperature - 5 * humidity + 2 *
wind_speed + np.random.normal(0, 50, n_samples)

# Membuat DataFrame
data = pd.DataFrame({'Temperature': temperature,
                     'Humidity': humidity,
```

```
                    'Wind_Speed': wind_speed,
                    'Energy_Consumption': energy_consumption})

data.head()
```

Output:

	Temperature	Humidity	Wind_Speed	Energy_Consumption
0	27.483571	73.993554	8.649643	826.776845
1	24.308678	69.246337	9.710963	873.257776
2	28.238443	60.596304	8.415160	975.552952
3	32.615149	53.530632	9.384077	1171.650868
4	23.829233	66.982233	6.212771	943.634363

7.2.2 Preprocessing Data

Sebelum melatih model, kita perlu membagi data menjadi data latih dan data uji, serta melakukan penskalaan fitur.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Membagi data menjadi data latih dan data uji
X = data.drop('Energy_Consumption', axis=1)
y = data['Energy_Consumption']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Penskalaan fitur
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

7.2.3 Melatih Model Decision Tree Regressor

Kita akan melatih model Decision Tree Regressor dengan data latih.

```
from sklearn.tree import DecisionTreeRegressor

# Membuat dan melatih model Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train, y_train)
```

7.2.4 Hyperparameter Tuning

Untuk meningkatkan kinerja model, kita akan mencari hyperparameter terbaik menggunakan GridSearchCV.

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

# Mencari hyperparameter terbaik
param_grid = {'max_depth': range(1, 11),
              'min_samples_split': range(2, 21)}

grid_search = GridSearchCV(DecisionTreeRegressor(random_state=42),
param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Menampilkan hyperparameter terbaik
print("Best Hyperparameters:")
print(grid_search.best_params_)

# Melatih model dengan hyperparameter terbaik
best_dt_regressor = grid_search.best_estimator_
```

Output:

```
Best Hyperparameters:
{'max_depth': 5, 'min_samples_split': 14}
```

7.2.5 Evaluasi Model dengan Metrics Regresi dan Validasi Silang

Sekarang, kita akan mengevaluasi kinerja model menggunakan beberapa metrics regresi dan menerapkan validasi silang.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.model_selection import cross_val_score

# Memprediksi konsumsi energi
y_pred_train = best_dt_regressor.predict(X_train)
y_pred_test = best_dt_regressor.predict(X_test)

# Menghitung metrics regresi
mse_train = mean_squared_error(y_train, y_pred_train)
mse_test = mean_squared_error(y_test, y_pred_test)
mae_train = mean_absolute_error(y_train, y_pred_train)
mae_test = mean_absolute_error(y_test, y_pred_test)
r2_train = r2_score(y_train, y_pred_train)
r2_test = r2_score(y_test, y_pred_test)

# Menampilkan metrics regresi
print("Mean Squared Error (Train):", mse_train)
print("Mean Squared Error (Test):", mse_test)
print("Mean Absolute Error (Train):", mae_train)
print("Mean Absolute Error (Test):", mae_test)
print("R^2 Score (Train):", r2_train)
print("R^2 Score (Test):", r2_test)

# Melakukan validasi silang
cv_scores = cross_val_score(best_dt_regressor, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')

# Menampilkan hasil validasi silang
print("Cross-Validation Mean Squared Error:", -np.mean(cv_scores))
```

Output:

```
Mean Squared Error (Train): 2208.3981047952348
Mean Squared Error (Test): 3538.50764529715
Mean Absolute Error (Train): 37.38052915693194
Mean Absolute Error (Test): 48.42437563304565
R^2 Score (Train): 0.7299735479344736
R^2 Score (Test): 0.5252039116708405
Cross-Validation Mean Squared Error: 3609.5747736546277
```

7.3 Prediksi Penjualan

7.3.1 Latar Belakang dan Objektif

Salah satu permasalahan yang sering dihadapi oleh perusahaan adalah meramalkan jumlah penjualan produk atau jasa mereka. Prediksi penjualan yang akurat sangat penting karena perusahaan perlu mengetahui berapa banyak produk yang harus diproduksi, serta bagaimana mengelola persediaan dan sumber daya manusia. Dalam studi kasus ini, kita akan menggunakan model CatBoost Regressor untuk membuat prediksi penjualan berdasarkan data sintetis.

Objektif dari studi kasus ini adalah untuk memahami bagaimana melakukan prediksi penjualan menggunakan model CatBoost Regressor. Kita akan melihat cara melakukan preprocessing, training model, hyperparameter tuning, evaluasi model, serta menerapkan cross validation.

7.3.2 Menyiapkan Data Sintetis

Untuk studi kasus ini, kita akan menggunakan data sintetis yang terdiri dari beberapa fitur seperti jumlah pengunjung, cuaca, hari libur, dan promosi. Berikut adalah contoh data sintetis yang akan kita gunakan:

```
import pandas as pd
import numpy as np

np.random.seed(42)

n_samples = 1000

date_rng = pd.date_range(start='1/1/2020', end='12/31/2021', freq='D')
date = pd.DataFrame(date_rng, columns=['date'])

visitor = np.random.randint(100, 5000, size=(n_samples, 1))
weather = np.random.choice(['sunny', 'rainy', 'cloudy'], n_samples,
p=[0.7, 0.2, 0.1])
holiday = np.random.choice([0, 1], n_samples, p=[0.95, 0.05])
promotion = np.random.choice([0, 1], n_samples, p=[0.8, 0.2])

sales = np.random.randint(1000, 10000, size=(n_samples, 1))

data = pd.DataFrame(np.hstack([visitor, weather.reshape(-1, 1),
holiday.reshape(-1, 1), promotion.reshape(-1, 1), sales]),
columns=['visitor', 'weather', 'holiday', 'promotion', 'sales'])
```

7.3.3 Preprocessing

Sebelum kita mulai melatih model CatBoost Regressor, kita perlu melakukan preprocessing terlebih dahulu. Langkah preprocessing yang perlu dilakukan adalah:

- Melakukan encoding pada fitur kategori (cuaca) menggunakan One-Hot Encoding.
- Memisahkan data menjadi fitur dan target.
- Memisahkan data menjadi data latih dan data uji.

```
# One-Hot Encoding
data = pd.get_dummies(data, columns=['weather'], drop_first=True)

# Memisahkan fitur dan target
X = data.drop('sales', axis=1)
y = data['sales']

# Memisahkan data latih dan data uji
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

7.3.4 Training Model

Kemudian, kita akan melatih model CatBoost Regressor menggunakan data latih.

```
from catboost import CatBoostRegressor

model = CatBoostRegressor(random_state=42, verbose=0)
model.fit(X_train, y_train)
```

7.3.5 Hyperparameter Tuning

Untuk meningkatkan performa model, kita akan melakukan hyperparameter tuning menggunakan GridSearchCV. Kita akan mencari parameter terbaik untuk model CatBoost Regressor.

```
!pip3 install catboost
from catboost import CatBoostRegressor

model = CatBoostRegressor()

param_grid = {
    'iterations': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'depth': [4, 6, 8],
    'l2_leaf_reg': [1, 3, 5],
```

```
}

grid_search = GridSearchCV(model, param_grid,
scoring='neg_mean_squared_error', cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
```

7.3.6 Evaluasi Model

Setelah menemukan parameter terbaik, kita akan mengevaluasi model menggunakan berbagai metrik evaluasi regresi.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

y_pred = best_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
```

Output:

```
Mean Squared Error: 6493791.55845953
Mean Absolute Error: 2194.565316670949
R2 Score: -0.00051406313677127
```

7.3.7 Cross Validation

Terakhir, kita akan menerapkan cross validation untuk memastikan bahwa model kita dapat memberikan hasil yang konsisten pada data yang berbeda. Kita akan menggunakan metode K-Fold Cross Validation.

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(best_model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')
cv_scores = -cv_scores
print("Cross Validation MSE:", cv_scores)
print("Mean MSE:", cv_scores.mean())
print("Standard Deviation:", cv_scores.std())
```

Output:

```
Cross Validation MSE: [6159205.95933055 7170029.47209087 6584419.05964981 6998800.23583225
 6573775.36080765]
Mean MSE: 6697246.017542227
Standard Deviation: 355505.60367049015
```

7.4 Prediksi Retensi Pelanggan

7.4.1 Latar Belakang dan Objektif

Mempertahankan pelanggan adalah salah satu aspek penting dalam menjalankan bisnis yang sukses. Dalam konteks ini, retensi pelanggan mengacu pada kemampuan perusahaan untuk menjaga pelanggan agar terus menggunakan produk atau layanan yang ditawarkan. Memahami faktor-faktor yang mempengaruhi retensi pelanggan dan memprediksi apakah pelanggan akan tetap atau pergi sangat penting untuk perusahaan, karena ini akan mempengaruhi strategi pemasaran dan pendekatan layanan pelanggan.

Tujuan dari studi kasus ini adalah untuk membangun model regresi yang mampu memprediksi retensi pelanggan berdasarkan faktor-faktor seperti usia akun, jumlah transaksi, jumlah keluhan, dan kepuasan pelanggan. Kita akan menggunakan data sintetis untuk tujuan pembelajaran ini.

7.4.2 Menyiapkan Data Sintetis

Pertama, kita akan membuat data sintetis yang mencerminkan hubungan antara variabel-variabel yang relevan dan retensi pelanggan. Untuk melakukan ini, kita akan menggunakan pustaka numpy dan pandas.

```
import numpy as np
import pandas as pd

np.random.seed(0)

n_samples = 1000

ages = np.random.randint(1, 10, n_samples)
transactions = np.random.randint(1, 100, n_samples)
complaints = np.random.randint(0, 10, n_samples)
satisfaction = np.random.randint(1, 100, n_samples)
retention = ages * 2 + transactions * 0.5 - complaints * 4 +
satisfaction * 0.8 + np.random.normal(0, 10, n_samples)

data = pd.DataFrame({'age': ages, 'transactions': transactions,
'complaints': complaints, 'satisfaction': satisfaction, 'retention':
retention})
```

7.4.3 Menyiapkan Data Train dan Test

Setelah membuat data sintetis, kita akan membaginya menjadi data pelatihan dan pengujian:

```
from sklearn.model_selection import train_test_split

X = data.drop(columns='retention')
y = data['retention']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

7.4.4 Data Preprocessing

Kemudian, kita akan melakukan preprocessing data, seperti penskalaan fitur:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

7.4.5 Model Training dan Evaluasi dengan Cross Validation

Setelah itu, kita akan melatih model regresi yang paling sesuai. Dalam kasus ini, kita akan menggunakan regresi linier berganda karena kita memiliki beberapa fitur yang mempengaruhi retensi pelanggan. Selain itu, kita akan menggunakan validasi silang untuk meningkatkan akurasi model.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

model = LinearRegression()
scores = cross_val_score(model, X_train_scaled, y_train, cv=5)

print("Skor validasi silang: ", scores)
```

Output:

```
Skor validasi silang:  [0.88847933 0.89497259 0.8858324  0.88890148 0.90487683]
```

Selanjutnya, kita akan melatih model dengan data pelatihan dan mengevaluasi kinerjanya menggunakan metrik evaluasi regresi yang relevan, seperti Mean Squared Error (MSE), Mean Absolute Error (MAE), dan R-squared.

```
model.fit(X_train_scaled, y_train)
```

```
y_pred = model.predict(X_test_scaled)

from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error: ", mse)
print("Mean Absolute Error: ", mae)
print("R-squared: ", r2)
```

Output:

```
Mean Squared Error:  107.95928271330253
Mean Absolute Error:  8.241718177454576
R-squared:  0.895560840316748
```

7.4.6 Menigkatkan Kinerja Model dengan Hyperparameter Tuning

Sekarang kita memiliki model yang telah dilatih dan dievaluasi menggunakan metrik yang relevan. Namun, kita bisa mencoba meningkatkan kinerja model lebih lanjut dengan melakukan penalaan hyperparameter. Dalam kasus regresi linier, kita dapat mencoba menggunakan regresi Ridge atau Lasso sebagai alternatif, yang menambahkan penalti terhadap koefisien untuk mengurangi overfitting.

```
from sklearn.linear_model import RidgeCV, LassoCV

ridge = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)
ridge.fit(X_train_scaled, y_train)
y_pred_ridge = ridge.predict(X_test_scaled)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("Ridge Regression")
print("Mean Squared Error: ", mse_ridge)
print("Mean Absolute Error: ", mae_ridge)
print("R-squared: ", r2_ridge)

lasso = LassoCV(alphas=[0.1, 1.0, 10.0], cv=5)
lasso.fit(X_train_scaled, y_train)
y_pred_lasso = lasso.predict(X_test_scaled)

mse_lasso = mean_squared_error(y_test, y_pred_lasso)
```

```
mae_lasso = mean_absolute_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print("\nLasso Regression")
print("Mean Squared Error: ", mse_lasso)
print("Mean Absolute Error: ", mae_lasso)
print("R-squared: ", r2_lasso)
```

Output:

```
Ridge Regression
Mean Squared Error:  107.92746881536364
Mean Absolute Error:  8.239994917647493
R-squared:  0.8955916168899474

Lasso Regression
Mean Squared Error:  107.86600803796966
Mean Absolute Error:  8.237455840914206
R-squared:  0.8956510736757206
```

7.5 Prediksi Hasil Panen

7.5.1 Latar Belakang dan Tujuan

Hasil panen merupakan aspek penting dalam pertanian dan pengelolaan sumber daya pangan. Dalam konteks ini, prediksi hasil panen menjadi sangat penting untuk mengevaluasi kebutuhan pangan, mengelola distribusi, dan mengendalikan harga. Dengan model prediksi yang akurat, petani dan pengambil kebijakan dapat membuat keputusan yang lebih baik dalam mengelola sumber daya mereka.

Tujuan dari studi kasus ini adalah untuk menjelaskan bagaimana teknik regresi dapat digunakan untuk memprediksi hasil panen berdasarkan faktor-faktor yang mempengaruhinya, seperti cuaca, luas lahan, dan penggunaan pupuk. Kita akan menggunakan data sintetis untuk tujuan ini dan membandingkan beberapa model regresi yang berbeda.

7.5.2 Menyiapkan Data Sintetis

Untuk memulai, kita akan membuat data sintetis yang mencakup berbagai faktor yang mempengaruhi hasil panen, seperti suhu rata-rata, curah hujan, luas lahan, dan penggunaan pupuk.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV
```

```
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.model_selection import cross_val_score

# Membuat data sintetis
np.random.seed(42)
n_samples = 1000

avg_temp = np.random.normal(20, 5, n_samples)
rainfall = np.random.normal(100, 20, n_samples)
land_area = np.random.normal(50, 10, n_samples)
fertilizer = np.random.normal(30, 5, n_samples)

harvest = 2 * avg_temp + 0.5 * rainfall + 3 * land_area + 1.5 *
fertilizer + np.random.normal(0, 10, n_samples)

data = pd.DataFrame({'avg_temp': avg_temp, 'rainfall': rainfall,
'land_area': land_area, 'fertilizer': fertilizer, 'harvest': harvest})

# Membagi data menjadi data pelatihan dan pengujian
X = data.drop('harvest', axis=1)
y = data['harvest']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Melakukan penskalaan fitur
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

7.5.3 Model Training dengan Beberapa Model Sekaligus

Kemudian, kita akan melatih beberapa model regresi berbeda menggunakan data yang telah disiapkan.

```
# Mendefinisikan model regresi
models = [
    ('Linear Regression', LinearRegression()),
    ('Ridge Regression', RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)),
    ('Lasso Regression', LassoCV(alphas=[0.1, 1.0, 10.0], cv=5)),
    ('Random Forest', RandomForestRegressor(n_estimators=100,
```

```
random_state=42)),
    ('Gradient Boosting', GradientBoostingRegressor(n_estimators=100,
random_state=42)),
('Support Vector Machine', SVR(kernel='linear', C=1))
]

# Melatih model dan menghitung skor evaluasi
evaluation_metrics = {'Model': [], 'Mean Absolute Error': [], 'Mean
Squared Error': [], 'R-squared': [], 'Cross Validation Score (Mean)':
[], 'Cross Validation Score (Standard Deviation)': []}

for name, model in models:
  model.fit(X_train_scaled, y_train)
  y_pred = model.predict(X_test_scaled)
  mae = mean_absolute_error(y_test, y_pred)
  mse = mean_squared_error(y_test, y_pred)
  r2 = r2_score(y_test, y_pred)

  # Melakukan validasi silang
  cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='neg_mean_squared_error')
  cv_mean = np.mean(cv_scores)
  cv_std = np.std(cv_scores)

  evaluation_metrics['Model'].append(name)
  evaluation_metrics['Mean Absolute Error'].append(mae)
  evaluation_metrics['Mean Squared Error'].append(mse)
  evaluation_metrics['R-squared'].append(r2)
  evaluation_metrics['Cross Validation Score (Mean)'].append(cv_mean)
  evaluation_metrics['Cross Validation Score (Standard
Deviation)'].append(cv_std)
evaluation_metrics_df = pd.DataFrame(evaluation_metrics)
print(evaluation_metrics_df)
```

Output:

	Model	Mean Absolute Error	Mean Squared Error	R-squared	Cross Validation Score (Mean)	Cross Validation Score (Standard Deviation)
0	Linear Regression	7.561333	91.877925	0.918252	-101.054296	9.905213
1	Ridge Regression	7.560889	91.872023	0.918258	-101.054978	9.905003
2	Lasso Regression	7.542397	91.684016	0.918425	-101.111567	9.701320
3	Random Forest	8.972122	132.846609	0.881801	-157.119323	4.397661
4	Gradient Boosting	8.199062	111.157427	0.901099	-130.887836	8.795969
5	Support Vector Machine	7.494852	92.134954	0.918024	-102.889367	9.087255

Setelah melatih model dan menghitung metrik evaluasi, kita akan menampilkan hasil dalam bentuk DataFrame yang membandingkan kinerja berbagai model regresi.

Kesimpulan

Dalam studi kasus ini, kita telah menjelaskan bagaimana regresi dapat digunakan untuk memprediksi hasil panen. Kita telah menggunakan data sintetis dan membandingkan kinerja berbagai model regresi, termasuk regresi linear, Ridge, Lasso, Random Forest, Gradient Boosting, dan Support Vector Machine. Selanjutnya, kita melakukan preprocessing data, pelatihan model, penyetelan hyperparameter, dan evaluasi lengkap dengan semua metrik evaluasi pada regresi termasuk menerapkan validasi silang.

Dari hasil studi kasus ini, kamu dapat melihat bagaimana regresi dapat digunakan dalam berbagai situasi nyata untuk memecahkan masalah yang kompleks dan membantu dalam pengambilan keputusan. Dengan menggabungkan berbagai teknik dalam machine learning, kita dapat mengembangkan model yang lebih akurat dan efisien untuk memprediksi dan mengoptimalkan hasil panen.

Terima Kasih