Chapter 6: Decision Trees

Pendahuluan

Decision Trees atau Pohon Keputusan adalah salah satu algoritma Machine Learning yang paling serbaguna dan intuitif. Model ini mampu melakukan tugas klasifikasi, regresi, dan bahkan multioutput. Decision Trees merupakan komponen dasar dari algoritma *ensemble* yang sangat populer seperti Random Forests. Salah satu keunggulan utamanya adalah model ini bersifat "white box". Artinya, cara kerjanya mudah dipahami dan proses pengambilan keputusannya dapat diinterpretasikan dengan jelas, tidak seperti model "black box" seperti Jaringan Saraf Tiruan.

1. Pelatihan dan Proses Prediksi

Secara visual, Decision Tree adalah sebuah struktur yang mirip dengan diagram alir (*flowchart*).

- **Node**: Setiap *node* (simpul) dalam pohon mengajukan sebuah pertanyaan tentang satu fitur.
- **Branch**: Setiap *branch* (cabang) mewakili jawaban dari pertanyaan tersebut (misalnya, "ya" atau "tidak").
- **Leaf Node**: Simpul paling ujung yang tidak memiliki cabang lagi disebut *leaf node*. Simpul ini berisi hasil prediksi akhir.

Untuk membuat prediksi pada sebuah instance baru, kita mulai dari *root node* (simpul paling atas) dan menelusuri pohon ke bawah. Pada setiap simpul, kita menjawab pertanyaan tentang fitur instance tersebut dan mengikuti cabang yang sesuai hingga mencapai sebuah *leaf node*. Prediksi untuk instance tersebut adalah kelas atau nilai yang ada di *leaf node* tersebut.

Selain prediksi kelas, Decision Tree juga dapat mengestimasi **probabilitas** sebuah instance termasuk dalam kelas tertentu. Probabilitas ini dihitung berdasarkan rasio instance pelatihan dari kelas tersebut yang ada di dalam *leaf node* tempat instance baru itu berakhir.

2. Algoritma Pelatihan CART

Scikit-Learn menggunakan algoritma **Classification And Regression Tree (CART)** untuk melatih Decision Trees. Algoritma ini bersifat *greedy*, artinya ia mencari pilihan terbaik secara lokal pada setiap langkah dan tidak pernah meninjau kembali keputusannya.

Prosesnya adalah sebagai berikut:

1. Algoritma membagi dataset pelatihan menjadi dua subset menggunakan satu fitur k dan sebuah *threshold* t_k (misalnya, "apakah panjang kelopak ≤ 2.45 cm?").
2. Algoritma mencari pasangan (k, t_k) yang menghasilkan subset paling "murni" (*pure*), yang diukur menggunakan sebuah *cost function*.
3. Setelah dataset terbagi, algoritma melanjutkan proses ini secara rekursif pada setiap subset, dan terus berlanjut hingga mencapai kedalaman maksimum yang diizinkan atau

ketika ia tidak dapat menemukan pembagian yang akan mengurangi tingkat ketidakmurnian (*impurity*).

Cost Function yang Digunakan:

- **Untuk Klasifikasi**:
    - **Gini Impurity (Default)**: Sebuah simpul dikatakan murni (Gini=0) jika semua instance di dalamnya berasal dari kelas yang sama. Gini impurity mengukur seberapa sering sebuah elemen acak dari set akan salah diberi label jika diberi label secara acak sesuai dengan distribusi label di dalam subset.
    - **Entropy**: Konsep dari teori informasi yang mengukur ketidakteraturan atau ketidakpastian. Entropy mencapai nol jika sebuah simpul murni. Penggunaan entropy cenderung menghasilkan pohon yang sedikit lebih seimbang.
- **Untuk Regresi**:
    - Tujuannya bukan untuk memisahkan kelas, tetapi untuk membagi data agar setiap *leaf* berisi instance dengan nilai target yang semirip mungkin.
    - *Cost function* yang digunakan adalah **Mean Squared Error (MSE)**. Algoritma CART mencari pembagian yang paling dapat mengurangi MSE.

3. Regularisasi untuk Mencegah Overfitting

Decision Tree adalah model non-parametrik, artinya struktur model tidak ditentukan sebelum pelatihan. Jika tidak dibatasi, model ini akan terus tumbuh dan beradaptasi secara sempurna dengan data pelatihan, yang pada akhirnya menyebabkan **overfitting** yang parah. Untuk mencegahnya, kita perlu melakukan regularisasi dengan membatasi kebebasan model selama pelatihan.

Beberapa *hyperparameter* regularisasi yang umum digunakan adalah:

- max_depth: Mengatur kedalaman maksimum pohon.
- min_samples_split: Menentukan jumlah minimum sampel yang harus dimiliki sebuah simpul sebelum ia dapat dibagi lagi.
- min_samples_leaf: Menentukan jumlah minimum sampel yang harus ada di sebuah *leaf node*.
- max_leaf_nodes: Membatasi jumlah maksimum *leaf node* yang dapat dimiliki pohon.
- max_features: Membatasi jumlah fitur yang dievaluasi untuk setiap pembagian pada setiap simpul.

Meningkatkan hyperparameter min_* atau mengurangi max_* akan meningkatkan regularisasi model.

4. Decision Trees untuk Regresi

Cara kerja Decision Tree untuk regresi sangat mirip dengan klasifikasi. Perbedaan utamanya terletak pada hasil prediksi di setiap *leaf node*.

- Bukan lagi memprediksi sebuah kelas, setiap *leaf node* memprediksi sebuah **nilai kontinu**.
- Nilai prediksi untuk sebuah instance baru adalah **rata-rata dari semua nilai target** instance pelatihan yang ada di dalam *leaf node* tersebut.
- Proses pelatihannya bertujuan untuk meminimalkan **MSE**, bukan Gini impurity atau Entropy.

5. Keterbatasan dan Ketidakstabilan

Meskipun kuat dan intuitif, Decision Trees memiliki beberapa keterbatasan:

1. **Batas Keputusan Ortogonal**: Decision Tree cenderung menciptakan batas keputusan yang tegak lurus (aksial), membuatnya sensitif terhadap rotasi data.
2. **Sangat Sensitif terhadap Variasi Data**: Kelemahan utamanya adalah ketidakstabilannya. Perubahan kecil pada data pelatihan (seperti menghapus satu instance) dapat menghasilkan struktur pohon yang sangat berbeda. Ini karena sifat *greedy* dari algoritma CART.

Karena ketidakstabilan ini, performa satu Decision Tree seringkali tidak sebaik model lain. Namun, kelemahan ini dapat diatasi dengan menggunakan sekumpulan pohon (*ensemble*), seperti yang akan dibahas pada bab selanjutnya tentang Random Forests.