THE MACHINE LEARNING JOURNEY (BRIEF)

July 11, 2024

Prepared by: Phan Kỳ Nhân - 2412432

Ho Chi Minh City University of Technology

Mục lục

1 Giới thiệu

2 Thuật toán K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) là một thuật toán *supervised learning* đơn giản, có thể dùng cho cả bài toán phân loại (classification) và hồi quy (regression). Cho một điểm dữ liệu mới x cần dự đoán, quy trình cơ bản của KNN gồm các bước sau:

1. **Chọn số láng giềng K.** Đây là số điểm gần nhất sẽ được xem xét để đưa ra dự đoán.

2. **Tính khoảng cách:** Tính khoảng cách giữa x và từng điểm trong tập huấn luyện. Có nhiều định nghĩa khoảng cách; phổ biến nhất là *Minkowski distance*:

$$d_p(x, y) = \Big(\sum_{j=1}^{d} |x_j - y_j|^p\Big)^{1/p},$$

   trong đó với $p = 2$ ta có **khoảng cách Euclid**:

$$d_{\text{Euclid}}(x, y) = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2},$$

   và với $p = 1$ là **khoảng cách Manhattan**.

3. **Chọn K láng giềng gần nhất:** Sắp xếp các điểm huấn luyện theo khoảng cách tăng dần và lấy K điểm nhỏ nhất.

4. **Dự đoán dựa trên láng giềng:**

   - Với bài toán *phân loại*, ta thực hiện bỏ phiếu đa số (majority vote): nhãn $\hat{y}$ của x là nhãn xuất hiện nhiều nhất trong K láng giềng gần nhất
   - Với bài toán *hồi quy*, ta lấy giá trị trung bình của các giá trị tương ứng của K láng giềng

$$\hat{y} \;=\; \frac{1}{K} \sum_{i=1}^{K} y_{(i)},$$

     trong đó $y_{(1)}, \ldots, y_{(K)}$ là giá trị đáp án của K điểm gần nhất.

Về mặt kỹ thuật, KNN là một thuật toán *lazy learning* (học cục bộ) và *non-parametric*. Khi áp dụng, một số điểm lưu ý gồm:

- **Chọn K phù hợp:** Không có công thức chặt chẽ cho K, thường ta thử nghiệm nhiều giá trị K (đặc biệt các giá trị lẻ) và dùng *cross-validation* để chọn K tối ưu. Nếu K quá nhỏ, mô hình dễ bị nhiễu (high variance); nếu K quá lớn, mô hình có thể quá tổng quát (high bias). Việc chọn K còn phụ thuộc vào số mẫu và tiếng ồn của dữ liệu.

- **Thước đo khoảng cách:** Thông dụng là khoảng cách Euclid như trên. Tuy nhiên, tuỳ bài toán có thể dùng khoảng cách Manhattan, Chebyshev hoặc Minkowski p-norm. Giá trị khoảng cách lớn hơn đồng nghĩa với sự khác biệt lớn hơn giữa các điểm.

- **Trọng số láng giềng:** Ta có thể gán trọng số cho các láng giềng khi biểu quyết. Với *uniform weight* mọi láng giềng có trọng số bằng nhau, còn với *distance weight* trọng số w_i của láng giềng thứ i tỷ lệ nghịch với khoảng cách d_i của nó tới x (thường là $w_i = 1/d_i$). Như vậy các điểm gần hơn sẽ có ảnh hưởng lớn hơn.

- **KNN trong hồi quy:** Tương tự phân loại nhưng sau khi chọn K láng giềng, ta lấy trung bình giá trị đầu ra để dự đoán (như công thức trên).

- **Tính toán và cấu trúc dữ liệu:** Cách cơ bản là duyệt toàn bộ tập huấn luyện (Brute Force), với độ phức tạp $O(n)$ cho mỗi điểm cần dự đoán. Khi dữ liệu lớn, người ta thường xây cấu trúc dữ liệu không gian nhằm tăng tốc. Ví dụ, *KD-Tree* chia dữ liệu theo từng trục đặc trưng tại điểm trung vị, giúp tìm kiếm láng giềng nhanh hơn. Hay *Ball Tree* phân chia dữ liệu thành các hình cầu siêu không gian chứa các tập con điểm. Các cấu trúc này cho phép loại trừ nhiều điểm không cần tính toán vì chúng nằm ngoài tầm cần tìm, làm giảm đáng kể thời gian truy vấn.

Cuối cùng, với bài toán phân loại nhị phân, thường cần sử dụng ma trận nhiễu và các chỉ số như **Precision** và **Recall** để đánh giá hiệu quả. Định nghĩa:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN},$$

trong đó TP (true positives) là số trường hợp thực tế dương được dự đoán đúng, FP (false positives) là số trường hợp âm bị dự đoán sai là dương, và FN (false negatives) là số trường hợp dương bị dự đoán sai. Precision cho biết tỉ lệ dự đoán dương chính xác, còn Recall cho biết tỉ lệ trường hợp dương thực sự được phát hiện. Ví dụ, trong bài toán chẩn đoán ung thư, ta quan tâm đặc biệt Recall cao (ít bỏ sót bệnh), trong khi cũng cần đảm bảo Precision không quá thấp.

3 Decision Trees (Cây quyết định)

Cây quyết định là một mô hình phân chia dữ liệu theo các đặc trưng một cách đệ quy, tạo nên cấu trúc cây nhị phân. Mỗi nút trong cây tương ứng với một quyết định dựa trên một đặc trưng (có thể có ngưỡng với biến liên tục). Quá trình xây dựng cây thường thực hiện các bước: tại nút cha, chọn đặc trưng và ngưỡng sao cho dữ liệu ở hai nhánh con trở nên "thuần nhất" nhất theo một tiêu chí; rồi tiếp tục đệ quy với mỗi nhánh. Độ thuần nhất của một tập dữ liệu được đo bằng các chỉ số như **Gini impurity** hoặc **Entropy**.

3.1 Gini Impurity

Với tập dữ liệu D gồm các mẫu thuộc K lớp, Gini được định nghĩa như sau:

$$Gini(D) = 1 - \sum_{i=1}^{K} p_i^2,$$

trong đó p_i là tỉ lệ mẫu lớp i trong D. Nếu tất cả mẫu đều cùng một lớp ($p_i = 1$ cho một lớp), thì $Gini(D) = 0$ (hoàn toàn thuần). Ngược lại, khi dữ liệu phân bố đồng đều giữa các lớp, giá trị Gini càng lớn. Ví dụ với 2 lớp, Gini đạt giá trị tối đa 0.5 khi $p_1 = p_2 = 0.5$.

Khi một nút cha D có N mẫu được chia thành hai nút con D_L, D_R có N_L và N_R mẫu, ta tính Gini sau chia bằng tổng trọng số của Gini từng nhánh con:

$$Gini_{\text{split}} = \frac{N_L}{N} Gini(D_L) + \frac{N_R}{N} Gini(D_R).$$

Chiến lược xây dựng cây là chọn phép chia (theo tính chất nào, ngưỡng nào) sao cho chỉ số Gini tổng sau chia này nhỏ nhất (tức độ tạp nhiễu giảm nhiều nhất). Ví dụ trong thực nghiệm, Gini của phép chia "Past Trend" cho kết quả nhỏ nhất nên được chọn làm nút gốc.

3.2 Entropy và Information Gain

Entropy là một thước đo hỗn loạn (uncertainty) của tập dữ liệu:

$$H(D) = -\sum_{i=1}^{K} p_i \log_2(p_i),$$

trong đó p_i giống định nghĩa trên. Entropy bằng 0 khi tập hoàn toàn thuần (không hỗn loạn). Khi chia dữ liệu tại một nút, ta tính *Information Gain* (IG) để đo sự giảm hỗn loạn:

$$IG = H(D) - \sum_{c} \frac{N_c}{N} H(D_c),$$

với D là tập mẹ, D_c các tập con, N_c cỡ mẫu con và N cỡ mẫu mẹ. Ta chọn phép chia có IG lớn nhất (tức H_{split} nhỏ nhất). Công thức IG này chính là hiệu entropy trước và sau chia (tổng entropy con được trung bình theo xác suất). Trong bài toán xây dựng cây, tiêu chí dùng Gini hay IG (entropy) đều nhằm tạo ra nhánh con càng thuần nhất càng tốt.

3.3 Xây dựng và cắt tỉa cây

Thuật toán xây cây đệ quy tiếp tục như sau: tại mỗi nút, thử tất cả đặc trưng (và nếu là biến liên tục thì thử nhiều ngưỡng có thể) để tính chỉ số (Gini hoặc IG) sau chia, rồi chọn phép chia tốt nhất. Quá trình dừng khi gặp điều kiện dừng: ví dụ nếu một nút thuần (Gini=0 hoặc Entropy=0) thì không cần chia tiếp; hoặc số mẫu trong nút nhỏ hơn ngưỡng (min_samples_leaf) hoặc đạt độ sâu tối đa.

Cây quyết định (đặc biệt CART) cũng áp dụng cho hồi quy. Thay vì Gini/Entropy, người ta dùng *giảm phương sai* (variance reduction) hoặc *sum of squared residuals* (SSR). Cụ thể, mỗi nút con dự đoán giá trị trung bình của các y_i. Mỗi ngưỡng chia được chọn sao cho tổng bình phương sai số của hai nhánh con nhỏ nhất:

$$SSR = \sum_{i \in D_L} (y_i - \bar{y}_L)^2 + \sum_{i \in D_R} (y_i - \bar{y}_R)^2,$$

với $\bar{y}_L, \bar{y}_R$ là trung bình nhãn ở hai nhánh. Tương đương, chọn chia để giảm phương sai trong nút con.

Khi cây quá lớn, dễ xảy ra overfitting. Để khắc phục, ta có thể *cắt tỉa (pruning)* cây bằng cách đánh đổi giữa độ lỗi và độ phức tạp. Một phương pháp phổ biến là *cost-complexity pruning*: định nghĩa *điểm số* của cây con T là

$$\text{Score}(T) = SSR(T) + \alpha \cdot |leaves(T)|,$$

trong đó $|leaves(T)|$ là số nút lá và $\alpha \geq 0$ là tham số phạt phức tạp. Tăng α sẽ ưu tiên cây nhỏ gọn; $\alpha = 0$ cho cây lớn nhất. Ta chọn α tối ưu bằng cross-validation, sao cho lỗi trên tập kiểm thử (OOB hay K-fold) đạt nhỏ nhất. Sau khi pruning, cây thu được sẽ đạt độ tổng quát tốt hơn, tránh bắt trùng với tiếng ồn dữ liệu.

4 Random Forest (Rừng ngẫu nhiên)

Random Forest là phương pháp *ensemble* kết hợp nhiều cây quyết định độc lập để cải thiện độ chính xác và giảm phương sai. Một rừng ngẫu nhiên được xây dựng như sau:

- Sử dụng kỹ thuật **bagging** (bootstrap aggregating): từ tập huấn luyện gốc, lặp B lần chọn ngẫu nhiên (với phép lấy mẫu có hoàn lại) tập con kích thước n để huấn luyện B cây quyết định khác nhau. Mỗi cây do đó nhìn thấy một mẫu dữ liệu hơi khác so với cây khác.

- Tại mỗi bước chia nút trong một cây, chỉ chọn ngẫu nhiên một tập con đặc trưng (feature subset) thay vì xét tất cả. Ví dụ điển hình là chọn $\sqrt{p}$ đặc trưng (với p là số đặc trưng) cho bài toán phân loại. Điều này gọi là *feature bagging* (hoặc *random subspace*). Mục đích là làm giảm tương quan giữa các cây (nếu có vài đặc trưng mạnh, mỗi cây sẽ thường dùng chúng gây giảm hiệu quả khi kết hợp). Kết quả: các cây trong rừng thường *không tương quan* với nhau nhưng đều không bị lệ thuộc vào một vài đặc trưng nhất định.

- Sau khi huấn luyện B cây, cách kết hợp kết quả:

  - Với *phân loại*: mỗi cây đưa ra dự đoán lớp, sau đó lấy *đa số phiếu* (majority vote) của B cây
  - Với *hồi quy*: mỗi cây đưa ra giá trị ước lượng, kết quả cuối cùng là *trung bình* các giá trị này.

- Ngoài ra, mỗi cây dùng mẫu bootstrap nên có khoảng 1/3 mẫu gốc không được chọn (gọi là *out-of-bag* (OOB)). Ta có thể sử dụng tập OOB của từng cây để đánh giá ngẫu nhiên mô hình mà không cần tập kiểm thử riêng.

Nhờ ensemble, Random Forest giảm đáng kể phương sai so với cây đơn lẻ. Mặc dù mỗi cây có thể phức tạp và dễ overfit, kết quả trung bình của nhiều cây lại rất ổn định. Ngoài ra, do xây trên mẫu và đặc trưng ngẫu nhiên, rừng cũng chịu được phần dữ liệu bị thiếu; thí dụ ta có thể ước lượng giá trị thiếu dựa trên ma trận *proximity* giữa các điểm (hai điểm thường cùng xuất hiện ở cùng lá được coi là gần nhau).

Tóm lại, Random Forest là thuật toán mạnh cho cả phân loại và hồi quy, tích hợp các ưu điểm của nhiều cây quyết định: dễ triển khai, tự động đánh trọng số đặc trưng quan trọng (qua tần suất phân chia), và cho kết quả chính xác cao mà thường không cần tinh chỉnh quá nhiều. Kết quả là lỗi trung bình của rừng (có thể ước lượng qua OOB hoặc cross-validation) thường thấp và độ sai số tương đối ổn định