Bài tập 0 - Tìm hiểu Decision Tree

Trần Như Cẩm Nguyên - 22520004

September 2023

Link Google Colab: `https://colab.research.google.com/drive/18_Zu6ikAyDFsNE1wxvOpHlHEayF1MQBH?usp=sharing`

Mục lục

1 Giới thiệu **2**

2 Decision Trees trong sklearn **2**
2.1 Công dụng 2
2.2 Cách dùng 2

3 Các tiêu chuẩn split phổ biến **6**
3.1 Gini Impurity 6
3.2 Entropy 7

4 Thuật toán xây dựng Decision Tree **7**
4.1 ID3 (Iterative Dichotomiser 3) 7
4.2 C4.5 7
4.3 CART (Classification and Regression Trees) 7

5 Thuật toán ID3 **8**
5.1 Ý tưởng 8
5.2 Thuật toán 8

6 Điều kiện dừng phân chia **9**

7 Kết luận **9**

1 Giới thiệu

Decision Tree là một mô hình học máy phổ biến và mạnh mẽ, được sử dụng rộng rãi trong nhiều ứng dụng khác nhau. Decision Tree chia dữ liệu thành các phân nhánh dựa trên các quy tắc quyết định đơn giản dựa trên các đặc trưng. Báo cáo này sẽ trình bày về Decision Trees trong sklearn, bao gồm công dụng và cách sử dụng chúng. Cùng với đó là một số thuật toán xây dựng Decision Tree và các tiêu chuẩn split quan trọng: gini, entropy,...

2 Decision Trees trong sklearn

2.1 Công dụng

1. **Phân loại (Classification):** Decision Tree được sử dụng để phân loại các mẫu dữ liệu vào các lớp khác nhau. Ví dụ: Phân loại email là spam hoặc không phải spam
2. **Hồi quy (Regression):** Dùng để dự đoán giá trị số học. Ví dụ: Dự đoán giá nhà dựa trên các đặc trưng như diện tích, vị trí.

2.2 Cách dùng

1. **Dùng cho phân loại**

   Sử dụng DecisionTreeClassifier.

```
1  from sklearn.tree import
       DecisionTreeClassifier
2  model = DecisionTreeClassifier()
```

   DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)

   *** Parameters:**

   (a) **criterion**: {"entropy", "log_loss", default="gini"}
   Tiêu chí sử dụng để đánh giá chất lượng của sự phân chia tại mỗi nút.
      i. "entropy": Đo lường độ không chắc chắn trong phân phối lớp tại 1 nút.
      ii. "log_loss": Được dùng trong bài toán dự đoán xác suất. Đo lường sự khác biệt giữa xác suất dự đoán và xác suất thực tế.
      iii. "gini": Đo lường độ vẩn đục của 1 nút trong cây

   (b) **splitter**: {"random", default="best"}
   Đặc trưng để phân chia các nút

i. "random": Chọn ngẫu nhiên 1 đặc trưng để phân chia tại mỗi nút. Giảm độ phức tạp tính toán nhưng không đảm bảo phân chia tốt nhất.

ii. "best": Chọn đặc trưng tối ưu. Cây sẽ xem xét tất cả đặc trưng có sẵn, chọn đặc trưng tạo ra sự phân chia tốt nhất dựa trên tiêu chí được chọn (gini/entropy/log_loss). Phương pháp này đảm bảo phân chia tốt nhất, nhưng cần nhiều tính toán hơn.

(c) **max_depth**: {int, default=None}
Chiều sâu tối đa của cây

i. int: Xác định giới hạn chiều sâu của cây bằng 1 số nguyên.

ii. None: Cây được xây dựng mà không giới hạn về chiều sâu. Cây có thể phát triển đến khi nào nó chia tách hoàn toàn dữ liệu huấn luyện, hoặc đến khi 1 tiêu chí dừng được đáp ứng (min_samples_split hoặc min_samples_leaf)

(d) **min_samples_split**: {int, float, default=2}
Xác định số lượng mẫu tối thiểu mà 1 nút phải có trước khi nó được chia thành các nút con.

i. int: Nếu là số nguyên, nó là số lượng mẫu tối thiểu. Nếu số lượng mẫu tại nút hiện tại ít hơn 'min_samples_split' thì nút đó không được chia thêm và thành 1 nút lá.

ii. float: Nếu là số thập phân, số lượng mẫu tối thiểu để có thể chia thành các nút con là $ceil(min\_samples\_split * n\_samples)$

(e) **min_samples_leaf**: {int, float, default=1}
Xác định số lượng mẫu tối thiểu tại 1 nút lá.

i. int: Nếu là số nguyên, nó là số lượng mẫu tối thiểu. Tại bất kì độ sâu nào của cây, chỉ được xem xét nếu nó để lại ít nhất min_samples_leaf mẫu huấn luyện trong cả 2 nhánh trái và phải.

ii. float: Nếu là số thập phân, số lượng mẫu tối thiểu cho mỗi nút là $ceil(min\_samples\_leaf * n\_samples)$

(f) **min_weight_fraction_leaf**: {float, default=0.0}
Xác định tỉ lệ trọng số tối thiểu cần có tại 1 nút lá. Tham số này dùng để kiểm soát kích thước của các nút lá dựa trên trọng số của các mẫu. Các mẫu có cùng trọng số khi không cung cấp tham số sample_weight

(g) **max_features**: {int, float, "auto", "sqrt", "log2", default=None}
Số lượng đặc trưng mà mô hình xem xét khi tìm kiếm điểm chia tốt nhất

i. int: Nếu là số nguyên, xác định số lượng đặc trưng cần xem xét ở mỗi điểm chia

ii. float: Nếu là số thập phân, xem xét max(1, int(max_features * n_features_in_)) đặc trưng tại mỗi lần chia
iii. "auto": Tự động chọn
iv. "sqrt": max_features = $\sqrt{n\_features}$
v. "log2": max_features = $\log_2 n\_features$
vi. None: max_features = n_features

(h) **random_state**: {int, RandomState instance, default=None}
Kiểm soát tính ngẫu nhiên của mô hình. Các đặc trưng luôn được xáo trộn 1 cách ngẫu nhiên ở mỗi lần chia, ngay cả khi "splitter" được đặt thành "best". Khi max_features < n_features, thuật toán sẽ chọn max_features đặc trưng ngẫu nhiên ở mỗi lần chia trước khi tìm điểm chia tốt nhất trong số chúng.

(i) **max_leaf_nodes**: {int, default=None}
Xây dựng 1 cây với tối đa max_leaf_nodes lá theo cách tốt nhất. Các nút tốt nhất được định nghĩa dựa trên giảm thiểu tương đối về độ vẩn đục. Nếu là None thì không giới hạn số lá

(j) **min_impurity_decrease**: {float, default=0.0}
Một nút sẽ được chia nếu việc chia sẽ làm giảm độ vẩn đục 1 lượng lớn hơn hoặc bằng giá trị này. Công thức tính sự giảm độ vẩn đục có trọng số như sau:
$N_t/N*(impurity-N_{t_R}/N_t*right\_impurity-N_{t_L}/N_t*left\_impurity)$
Với N là tổng số mẫu, N_t là số lượng mẫu tại nút hiện tại, N_{t_L} là số lượng mẫu tại nút con bên trái, N_{t_R} là số lượng mẫu tại nút con bên phải.

(k) **class_weight**: {dict, list of dict or "balanced", default=None}
Trọng số liên quan đến các lớp dưới dạng class_label: weight. Nếu None, tất cả các lớp được giả định có trọng số là 1.

(l) **ccp_alpha**: {non-negative float, default=0.0}
Tham số phức tạp sử dụng cho Minimal Cost-Complexity Pruning. Cây con có độ phức tạp chi phí lớn nhất mà nhỏ hơn ccp_alpha sẽ được chọn.

2. **Dùng cho hồi quy**

Sử dụng DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
```

DecisionTreeRegressor(*, criterion='squared_error', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0)

*** Parameters:**

Giống với DecisionTreeClassifier dùng cho phân loại, chỉ khác tham số "criterion"và không có tham số "class_weight"

(a) **criterion**: {“friedman_mse”, “absolute_error”, “poisson”, default=”squared_error”}
Tiêu chí sử dụng để đánh giá chất lượng của sự phân chia tại mỗi nút.

i. “friedman_mse”: Lối bình phương của Friedman. Lối bình phương trung bình với điểm cải thiện của Friedman để tìm các lần chia tiềm năng.
ii. “absolute_error”: Sử dụng cho độ đo sai số tuyệt đối trung bình, tối thiểu hóa lỗi L1 bằng cách sử dụng trung vị của mỗi nút lá.
iii. “poisson”: Sử dụng giảm thiểu độ sai lệch Poisson để tìm các lần chia.
iv. “squared_error”: Lối bình phương. Sử dụng cho độ sai số bình phương trung bình, tương đương với việc giảm phương sai làm tiêu chí lựa chọn đặc trưng và tối thiểu hóa lỗi L2 bằng cách sử dụng trung bình của mỗi lá.

3. **Huấn luyện**

Sau khi tạo mô hình với các tham số phù hợp, dùng bộ dữ liệu huấn luyện để huấn luyện mô hình, với X_train chứa các đặc trưng của dữ liệu huấn luyện, y_train chứa nhãn tương ứng với mỗi mẫu dữ liệu

```
1    model.fit(X_train, y_train)
```

4. **Dự đoán**

Dùng mô hình đã huấn luyện để dự đoán trên bộ dữ liệu kiểm tra, với X_test chứa các đặc trưng của dữ liệu kiểm tra

```
1    y_pred = model.predict( X_test )
```

5. **Đánh giá mô hình**

Có nhiều cách để đánh giá mô hình:

(a) **Accuracy**:
Đo lường tỉ lệ các dự đoán đúng trên tổng số dự đoán, với y_test chứa nhãn tương ứng với mỗi mẫu dữ liệu trong X_test. Tuy nhiên, với dữ liệu bị mất cân bằng (một lớp xuất hiện nhiều hơn lớp khác), thì cách đánh giá này không thích hợp.

```
1    from sklearn.metrics import
         accuracy_score
2    accuracy = accuracy_score(y_test,
         y_pred)
```

(b) **Precision, recal, và F1-score**:
Phù hợp để đánh giá khi dữ liệu mất cân bằng

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

# Precision
precision = precision_score(y_test,
    y_pred)
# Recall
recall = recall_score(y_test, y_pred)
# F1-score
f1 = f1_score(y_test, y_pred)
```

(c) **Cross-Validation (Kiểm định chéo)**:
Đánh giá hiệu suất của mô hình trên nhiều tập dữ liệu khác nhau

```
from sklearn.model_selection import
    cross_val_score
# 5-fold
scores = cross_val_score(model, X_train
    , y_train, cv=5)
```

3 Các tiêu chuẩn split phổ biến

Trong thuật toán xây dựng Decision Tree, các tiêu chuẩn split (criterion) định nghĩa cách chúng ta chọn thuộc tính và giá trị ngưỡng để chia tập dữ liệu thành các phân nhánh con, dùng để đánh giá chất lượng của 1 cách phân chia. Các tiêu chuẩn split phổ biến gồm Gini Impurity, Entropy, Information Gain,...

3.1 Gini Impurity

Chọn thuộc tính và giá trị ngưỡng sao cho khi chia dữ liệu, Gini Impurity giảm nhiều nhất. Điểm Gini Impurity càng thấp, dữ liệu càng tinh khiết và dễ phân loại.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Trong đó, C là số lớp cần phân loại, $p_i = \frac{n_i}{N}$ với n_i là số lượng phần tử ở lớp thứ i, N là tổng số lượng phần tử ở node đó

3.2 Entropy

Chọn thuộc tính và giá trị ngưỡng sao cho khi chia dữ liệu, Entropy giảm nhiều nhất. Entropy đo độ không chắc chắn (uncertainty) của dữ liệu. Nó đo lường độ mất mát thông tin khi phân loại một mẫu ngẫu nhiên vào các lớp. Entropy càng thấp, thì dữ liệu càng tinh khiết và dễ phân loại.

Cho một phân phối xác suất của một biến rời rạc x có thể nhận n giá trị khác nhau $x_1, x_2, ..., x_n$. Giả sử xác suất để x nhận các giá trị này là $p_i = p(x = x_i)$ với $0 \leq p_i \leq 1, \sum_{i=1}^{n} p_i = 1$. Kí hiệu phân phối này là $P = (p_1, p_2, ..., p_n)$. Entropy của phân phối này là

$$H(P) = -\sum_{i=1}^{n} p_i * \log p_i$$

4 Thuật toán xây dựng Decision Tree

4.1 ID3 (Iterative Dichotomiser 3)

1. **Cách hoạt động:** Sử dụng entropy để chọn đặc trưng tốt nhất cho việc chia dữ liệu. Lặp lại quá trình chia cho đến khi đạt điều kiện dừng (độ sâu tối đa/số mẫu tối thiểu tại mỗi nút,...)
2. **Tiêu chuẩn split:** Entropy
3. **Ưu điểm:** Dễ hiểu, tạo ra cây nhỏ và dễ giải thích.
4. **Nhược điểm:** Không xử lí được giá trị bị thiếu. Dễ bị overfitting nếu cây quá sâu. Chỉ hỗ trợ bài toán phân loại, không hỗ trợ bài toán hồi quy.

4.2 C4.5

1. **Cách hoạt động:** Sử dụng entropy để chọn đặc trưng tốt nhất cho việc chia dữ liệu. Cải tiến từ ID3 bằng cách xử lý dữ liệu bị thiếu và hỗ trợ nhiều loại đầu ra. Lặp lại quá trình chia cho đến khi đạt điều kiện dừng.
2. **Tiêu chuẩn split:** Entropy
3. **Ưu điểm:** Xử lý tốt dữ liệu thiếu, hỗ trợ nhiều loại đầu ra.
4. **Nhược điểm:** Dễ bị overfitting nếu cây quá sâu, có thể tạo ra cây lớn. Chỉ hỗ trợ bài toán phân loại, không hỗ trợ bài toán hồi quy.

4.3 CART (Classification and Regression Trees)

1. **Cách hoạt động:** Chọn một đặc trưng và một ngưỡng để chia dữ liệu thành hai phân nhánh con. Lặp lại quá trình chia cho đến khi đạt được điều kiện dừng.

2. **Tiêu chuẩn split:** Gini Impurity cho phân loại, Mean Squared Error (MSE) cho hồi quy.
3. **Ưu điểm:** Đơn giản, hiệu quả, phù hợp cho phân loại và hồi quy.
4. **Nhược điểm:** Cần xác định hàm mất mát phù hợp cho bài toán cụ thể. Cây có thể phức tạp và khó giải thích.

5 Thuật toán ID3

Thuật toán ID3 (Iterative Dichotomiser 3) được tạo ra bởi Ross Quinlan nhằm xây dựng Decision Tree phù hợp từ một bộ dữ liệu, đặt nền tảng cho các thuật toán sau này như C4.5 và CART.

5.1 Ý tưởng

Trong ID3, ta cần xác định thứ tự của thuộc tính cần được xem xét tại mỗi bước, theo phương pháp tham lam. Ta chọn thuộc tính tốt nhất dựa trên Entropy, chia dữ liệu vào các nút con tương ứng với giá trị của thuộc tính đó rồi lặp lại quá trình này với các nút con.

5.2 Thuật toán

Xét một bài toán với C class khác nhau. Giả sử ta đang làm việc với một non-leaf-node với các điểm dữ liệu tạo thành một tập S có N phần tử. Giả sử thêm trong số N điểm dữ liệu, $N_c, c = 1, 2, ...C$ điểm thuộc vào class c. Xác suất để mỗi điểm dữ liệu rơi vào một class xấp xỉ $\frac{N_c}{N}$. Entropy tại node này được tính bởi

$$H(S) = -\sum_{c=1}^{C} \frac{N_c}{N} log(\frac{N_c}{N})$$

Tiếp theo, giả sử thuộc tính được chọn là x. Theo x, các điểm dữ liệu trong S được phân ra thành K child node $S_1, S_2, ..., S_K$ với số điểm trong mỗi child node lần lượt là $m_1, m_2, ..., m_K$. Tổng có trọng số entropy của mỗi child node bằng:

$$H(x, S) = \sum_{k=1}^{K} \frac{m_k}{N} H(S_k)$$

Tiếp theo, ta định nghĩa information gain dựa trên thuộc tính x:

$$G(x, S) = H(S) - H(x, S)$$

Trong ID3, tại mỗi node, thuộc tính khiến information gain đạt giá trị lớn nhất sẽ được chọn, xác định dựa trên:

$$x^* = argmax_x G(x, S) = argmin_x H(x, S)$$

Các bước chính của ID3:

1. Xác định nút gốc: Tìm thuộc tính tốt nhất theo công thức trên.
2. Tạo các nhánh con: Dựa trên thuộc tính đã chọn, tạo các nhánh con của nút gốc. Mỗi nhánh con tương ứng với một giá trị riêng của thuộc tính đã chọn.
3. Phân chia dữ liệu: Phân chia tập dữ liệu thành các tập con dựa trên giá trị của thuộc tính đã chọn. Mỗi tập con tương ứng với một nhánh con của nút gốc.
4. Lặp lại quá trình: Đối với mỗi nhánh con, lặp lại quá trình trên cho đến khi đạt điều kiện dừng.

6 Điều kiện dừng phân chia

Nếu ta tiếp tục phân chia các nút chưa tinh khiết, ta sẽ được một cây rất phức tạp với nhiều nút lá chỉ có một vài điểm dữ liệu. Khả năng cao bị overfitting. Một số điều kiện có thể dùng để dừng phân chia, tránh overfitting:

1. Nếu node đó có entropy = 0, tức mọi điểm trong node đều thuộc một class.
2. Nếu node đó có số phần tử nhỏ hơn một ngưỡng nào đó. Trong trường hợp này, ta chấp nhận có một số điểm bị phân lớp sai để tránh overfitting. Class cho nút lá này có thể được xác định dựa trên class chiếm đa số trong nút.
3. Nếu khoảng cách từ nút đó đến nút gốc đạt tới một giá trị nào đó. Việc hạn chế chiều sâu của cây này làm giảm độ phức tạp của cây và phần nào giúp tránh overfitting.
4. Nếu tổng số nút lá vượt quá một ngưỡng nào đó.
5. Nếu việc phân chia nút đó không làm giảm entropy quá nhiều (information gain nhỏ hơn một ngưỡng nào đó).
6. Dùng phương pháp pruning

7 Kết luận

Decision Tree là một công cụ mạnh mẽ và phổ biến trong machine learning. Các thuật toán xây dựng Decision Tree là một phần quan trọng của học máy và khai phá dữ liệu. Mỗi thuật toán có cách tiếp cận riêng và phù hợp cho các loại bài toán cụ thể. Sự hiểu biết về cách hoạt động của các thuật toán này có thể giúp bạn lựa chọn và tối ưu hóa mô hình Decision Tree cho nhiều tình huống khác nhau.