**BỘ THÔNG TIN VÀ TRUYỀN THÔNG**

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**- - - 🙞 🕮** **🙜 - - -**

Icon

Description automatically generated

**BÁO CÁO**

MÔN: **NHẬP MÔN KHOA HỌC DỮ LIỆU**

CÂU 4: **Trình bày và xử lý bài toán phân cụm KNN dựa vào học máy bằng ngôn ngữ Pyhon**

**Giảng viên hướng dẫn: TS. Tân Hạnh**

**Sinh viên thực hiện:**

Nguyễn Trần Đức Thuận – N19DCCN203

**TP. HCM, 20/06/2023**

**THUẬT TOÁN K-NEAREST NEIGHBORS**

## **Lý thuyết:**

Thuật toán k-nearest neighbors (k-NN) là một thuật toán học có giám sát sử dụng để phân loại các điểm dữ liệu dựa trên các điểm dữ liệu đã được gán nhãn trong tập huấn luyện. Ý tưởng của thuật toán là tìm ra k điểm gần nhất với điểm dữ liệu cần phân loại và dựa trên nhãn của những điểm này để quyết định nhãn của điểm đó.

Cụ thể, thuật toán k-NN làm việc như sau:

1. Tính khoảng cách giữa điểm dữ liệu cần phân loại và các điểm dữ liệu trong tập huấn luyện. Khoảng cách có thể được tính bằng nhiều phương pháp khác nhau, như Euclid, Manhattan, cosine similarity, v.v.

2. Chọn k điểm gần nhất với điểm dữ liệu cần phân loại dựa trên khoảng cách tính được.

3. Dựa trên nhãn của k điểm gần nhất để quyết định nhãn của điểm dữ liệu cần phân loại. Có thể sử dụng phương pháp đa số hoặc phương pháp trọng số để tính toán nhãn của điểm này.

4. Hoàn thành quá trình phân loại.

Tuy nhiên, điểm yếu của thuật toán k-NN là nó có thể bị ảnh hưởng bởi nhiễu và các điểm dữ liệu cách xa. Điều này có thể dẫn đến việc phân loại sai các điểm dữ liệu. Để cải thiện sự hiệu quả của thuật toán, ta có thể sử dụng các phương pháp tiền xử lý dữ liệu hoặc các phương pháp đánh giá hiệu suất để chọn giá trị tối ưu cho k.

## **Xử lý bài toán phân cụm KNN dựa vào học máy bằng ngôn ngữ Python:**

A picture containing text, screenshot, font

Description automatically generated

* Tính toán phiếu bầu:
* Đầu vào của hàm là một danh sách các nhãn **(labels)**.
* Hàm **Counter()** trong collections được sử dụng để đếm số lần xuất hiện của mỗi nhãn trong danh sách **labels**.
* Sau đó, hàm **most\_common()** được sử dụng để trả về danh sách các cặp (nhãn, số lần xuất hiện) được sắp xếp giảm dần theo số lượng.
* Vì chúng ta muốn lấy ra nhãn có số phiếu bầu nhiều nhất, do đó chúng ta chọn mục đầu tiên trong danh sách trả về bằng cách sử dụng [0].
* Cuối cùng, hàm trả về nhãn có số phiếu bầu lớn nhất.
* Kiểm tra hàm **raw\_majority\_vote()** với một danh sách nhãn và xác nhận kết quả trả về là nhãn có số phiếu bầu lớn nhất.

A picture containing text, screenshot, software, font

Description automatically generated

* Đây là một phiên bản cải tiến của hàm **raw\_majority\_vote(labels)** ở trên. Điểm khác biệt lớn nhất là hàm này giả định rằng các nhãn được sắp xếp từ gần đến xa, và do đó, nó sẽ xác định kết quả bằng cách loại bỏ nhãn xa nhất cho đến khi có một nhãn duy nhất được chọn.
* Đầu vào của hàm vẫn là một danh sách các nhãn **(labels)**.
* Hàm **Counter()** được sử dụng để đếm số lần xuất hiện của mỗi nhãn trong danh sách.
* Hàm **most\_common()** được sử dụng để trả về một danh sách các cặp **(nhãn, số lần xuất hiện)** được sắp xếp giảm dần theo số lượng.
* Hàm này sử dụng một biến **'winner\_count'** để lưu trữ số lần xuất hiện của nhãn có số phiếu bầu cao nhất.
* Sau đó, nó đếm số lượng nhãn có số phiếu bầu tương tự với nhãn chiến thắng, nếu chỉ có một nhãn như vậy, hàm sẽ trả về nhãn đó. Nếu có nhiều hơn một nhãn, hàm sẽ loại bỏ nhãn xa nhất và thực hiện lại quá trình với danh sách còn lại đến khi có một nhãn duy nhất được chọn.
* Trong ví dụ cuối cùng, khi có một sự cân bằng giữa số phiếu bầu của các nhãn, hàm loại bỏ nhãn xa nhất một cách tuần tự và trả về nhãn chiến thắng cuối cùng.

A picture containing text, screenshot, font

Description automatically generated

* Phân loại điểm mới:
* Đầu vào của hàm gồm **k (số lượng điểm gần nhất cần tìm), labeled\_points (danh sách các LabeledPoint) và new\_point (điểm mới cần phân loại)**.
* **LabeledPoint** là một **namedtuple** bao gồm một vector point và một nhãn label.
* Hàm sắp xếp **labeled\_points** theo thứ tự từ gần đến xa với **new\_point** bằng cách sử dụng hàm **sorted()** và **key=lambda lp: distance(lp.point, new\_point)**, trong đó **distance(lp.point, new\_point)** là khoảng cách giữa điểm **lp.point** với **new\_point**.
* Sau đó, hàm lấy **k** nhãn của **k\_labeled\_points** gần nhất bằng cách sử dụng **list comprehension** và **slice operator**.
* Cuối cùng, hàm trả về nhãn chiến thắng bằng cách sử dụng hàm **majority\_vote()** đã được triển khai ở trên để xác định nhãn nào được bầu chọn nhiều nhất trong **k** nhãn gần nhất.

A picture containing text, screenshot, font

Description automatically generated

* Tạo các điểm ngẫu nhiên và tính khoảng cách ngẫu nhiên giữa các cặp điểm:
* Hàm **random\_point(dim: int) -> Vector** trả về một điểm ngẫu nhiên trong không gian dim chiều, nơi mỗi tọa độ được chọn ngẫu nhiên trong khoảng [0, 1).
* Hàm **random\_distances(dim: int, num\_pairs: int) -> List[float]** tạo ra một danh sách gồm **num\_pairs** khoảng cách ngẫu nhiên giữa các cặp điểm trong không gian dim chiều. Hàm sử dụng hàm **distance()** từ module **linear\_algebra** để tính toán khoảng cách giữa hai điểm ngẫu nhiên.

A picture containing text, screenshot

Description automatically generated

* Đọc dữ liệu từ tập tin **iris.data** và xử lý thành một danh sách **LabeledPoint**:
* Hàm **parse\_iris\_row(row: List[str]) -> LabeledPoint** được sử dụng để chuyển đổi mỗi dòng trong tệp thành một **LabeledPoint**. Hàm này lấy một danh sách các giá trị chuỗi (row) làm đầu vào và trả về một **LabeledPoint** với các giá trị đầu vào được chuyển đổi thành một vector (measurements) và nhãn được tách ra từ tên lớp (label).
* Dữ liệu được đọc từ tệp **iris.data** bằng cách sử dụng module **csv**. Mỗi dòng trong tệp được đọc bởi reader và được chuyển đổi thành một **LabeledPoint** bằng hàm **parse\_iris\_row()**. Khi một **LabeledPoint** được tạo ra, nó được thêm vào danh sách **iris\_data**.
* Cuối cùng, các điểm được nhóm lại theo loài/ nhãn bằng cách sử dụng **defaultdict**. Mỗi **LabeledPoint** được thêm vào danh sách tương ứng với nhãn của nó bằng cách sử dụng phương thức **append()**.

A picture containing text, screenshot

Description automatically generated

* Đoạn code này sử dụng **matplotlib** để tạo ra một biểu đồ phân tán của các điểm dữ liệu **iris\_data**. Mỗi cặp thuộc tính **(metrics)** là một trục của biểu đồ, và các điểm được đánh dấu bằng các ký hiệu khác nhau **(marks)** tương ứng với các loài **(species)**. Biểu đồ được lưu vào tệp **'iris\_scatter.png'**.
* **metrics** là một danh sách các thuộc tính của **iris**, bao gồm **'sepal length'**, **'sepal width'**, **'petal length'**, và **'petal width'**.
* **pairs** là một danh sách các cặp chỉ số của các thuộc tính, được chọn sao cho chỉ có một cặp thuộc tính được hiển thị trên mỗi trục của biểu đồ.
* **marks** là một danh sách các ký hiệu được sử dụng để đại diện cho các loài khác nhau, và được chọn sao cho chỉ có ba loài khác nhau trong bộ dữ liệu **iris**.
* Biểu đồ được tạo ra bằng cách lặp lại các cặp chỉ số trong **pairs** và sử dụng chúng để truy xuất các giá trị tương ứng của thuộc tính trong các điểm **iris\_data**. Các điểm được đánh dấu bằng các ký hiệu khác nhau tương ứng với các loài khác nhau, và được thêm vào biểu đồ bằng cách sử dụng phương thức **scatter()**.
* Cuối cùng, biểu đồ được xuất ra tệp **'iris\_scatter.png'** bằng cách sử dụng phương thức **savefig()**.

A picture containing text, screenshot, font

Description automatically generated

* Đoạn code này sử dụng hàm **split\_data()** từ module **machine\_learning** để chia tập dữ liệu **iris\_data** thành hai phần riêng biệt: **iris\_train** và **iris\_test**.
* Hàm **split\_data()** nhận đầu vào là một danh sách các **LabeledPoint (data)** và một tỷ lệ phần trăm (prob). Nó trả về hai danh sách các phần tử, một danh sách là một phần của data với tỷ lệ prob, và một danh sách là phần còn lại của data.
* Trong đoạn code này, tập dữ liệu **iris\_data** được chia thành **iris\_train** và **iris\_test** với tỷ lệ 70% và 30%, tương ứng.

A picture containing text, screenshot, font, software

Description automatically generated

* Đoạn code này sử dụng thuật toán phân loại k-NN để phân loại các điểm trong tập dữ liệu kiểm tra iris\_test và tính toán độ chính xác của phân loại. Nó cũng tính toán ma trận nhầm lẫn **(confusion matrix)** để phân tích các phân loại sai.
* **confusion\_matrix** là một **defaultdict** được sử dụng để lưu trữ số lần mà một cặp **(predicted, actual)** xuất hiện trong quá trình phân loại. **predicted** là nhãn được dự đoán bởi thuật toán, **actual** là nhãn thực tế của điểm.
* Mỗi điểm trong tập kiểm tra **iris\_test** được phân loại bằng cách sử dụng hàm **knn\_classify()** với k=5 và tập huấn luyện **iris\_train**. Nhãn được dự đoán của điểm được lưu trữ trong **predicted**, và nhãn thực tế của điểm được lưu trữ trong **actual**.
* Nếu nhãn được dự đoán bởi thuật toán trùng khớp với nhãn thực tế, thì biến **num\_correct** được tăng lên 1.
* **confusion\_matrix** được cập nhật bằng cách tăng giá trị tương ứng của cặp **(predicted, actual)** lên 1.
* Cuối cùng, độ chính xác của thuật toán được tính toán bằng tỷ lệ số lượng các điểm được phân loại chính xác trên tổng số lượng điểm trong tập kiểm tra **iris\_test**. Ma trận nhầm lẫn được in ra để phân tích các phân loại sai.

A picture containing text, screenshot, software, font

Description automatically generated

* Đoạn code này tính toán khoảng cách trung bình và khoảng cách nhỏ nhất giữa các cặp điểm ngẫu nhiên trong một không gian n chiều, với n chạy từ 1 đến 100. Nó cũng tính toán tỷ lệ giữa khoảng cách nhỏ nhất và khoảng cách trung bình, được gọi là tỷ lệ tối thiểu-trung bình **(min\_avg\_ratio)**, để phân tích hiện tượng **"curse of dimensionality"**.
* **dimensions** là một danh sách các số nguyên từ 1 đến 100, đại diện cho số chiều của không gian.
* **avg\_distances** là một danh sách các giá trị khoảng cách trung bình, được tính toán bằng cách lấy tổng khoảng cách của 10,000 cặp điểm ngẫu nhiên và chia cho 10,000.
* **min\_distances** là một danh sách các giá trị khoảng cách nhỏ nhất, được tính toán bằng cách lấy khoảng cách nhỏ nhất của 10,000 cặp điểm ngẫu nhiên.
* **min\_avg\_ratio** là một danh sách các giá trị tỷ lệ giữa khoảng cách nhỏ nhất và khoảng cách trung bình, được tính toán bằng cách lấy giá trị của **min\_distances** chia cho giá trị của **avg\_distances** tương ứng.
* **tqdm** được sử dụng để hiển thị tiến trình tính toán trên thanh tiến trình.