AI VIET NAM – COURSE 2024

K-Nearest Neighbors and K-Mean

Quoc-Thai Nguyen, Tho-Anh-Khoa Nguyen, Dang-Nha Nguyen

Ngày 24 tháng 8 năm 2024

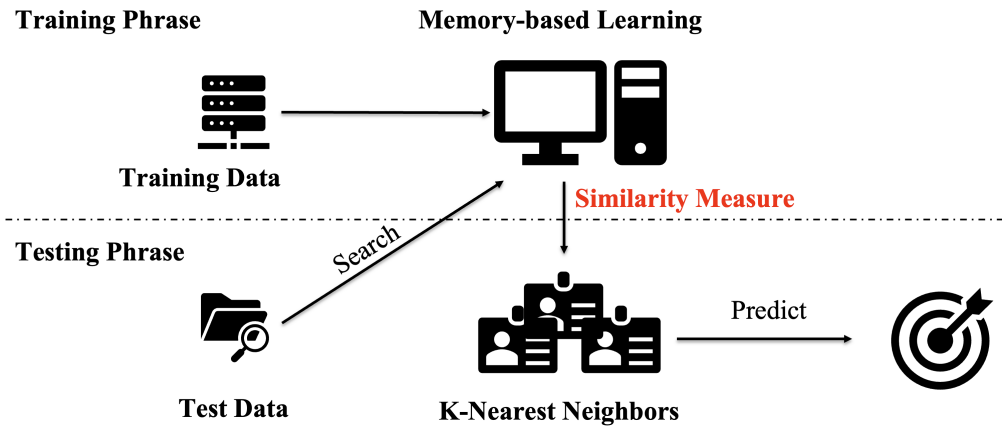
1. K-Nearest Neighbors

K-Nearest Neighbors (KNN) là một trong những thuật toán học máy có giám sát cơ bản. KNN còn được gọi là Lazy Learning, Memory-Based Learning,... Với bước huấn luyện mô hình, chỉ đơn giản là lưu trữ lại giá trị của dữ liệu huấn luyện, vì vậy KNN là phương pháp học máy không tham số (Non-Parametric). Ở bước dự đoán, mô hình sẽ dử dụng các độ đo khoảng cách để tìm các hàng xóm lân cận.

Một số độ đo thường được sử dụng trong KNN như:

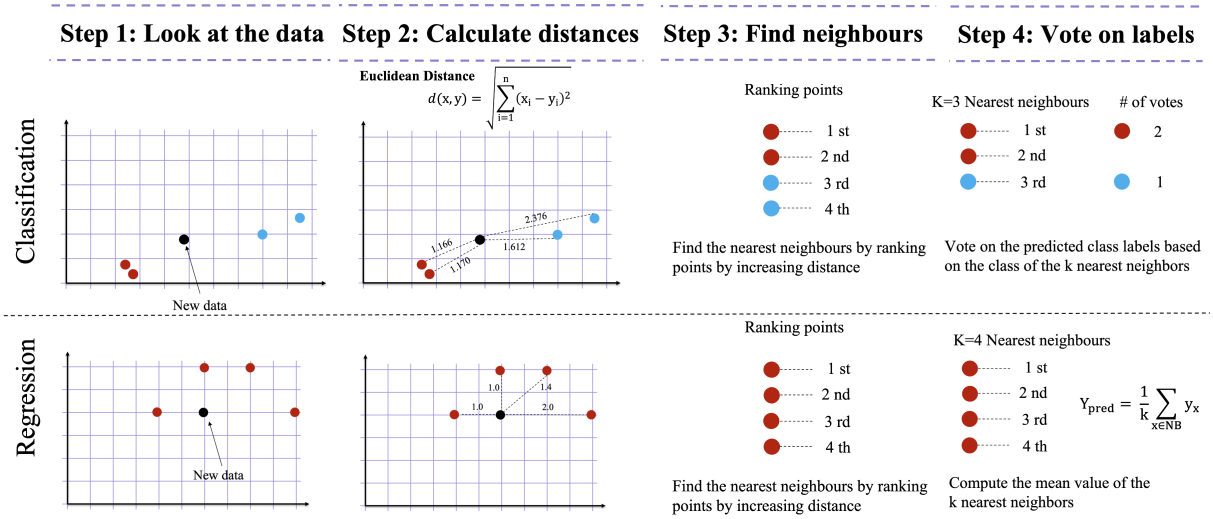
1. Euclidean
2. Chebyshev
3. Manhattan
4. Minkowski

Các độ đo này tính toán dữ liệu dự đoán với các điểm dữ liệu khác được lưu trữ trong mô hình huấn luyện. Từ đó xếp hạng và tìm ra K điểm dữ liệu huấn luyện có kết quả gần với dữ liệu dự đoán nhất. Cuối cùng dựa vào phương pháp biểu quyết của các dữ liệu hàng xóm trong tập huấn luyện để đưa ra kết quả dự đoán.



Hình 1: Mô tả quá trình huấn luyện và dự đoán mô hình KNN.

KNN được sử dụng rộng rãi cho ứng dụng phân loại (Classification) và dự đoán (Regression) được mô tả như hình sau:



Hình 2: Mô tả quá trình huấn luyện và dự đoán mô hình KNN.

Câu hỏi 1 Hàm đánh giá nào trong các hàm sau đây không được sử dụng trong KNN?
a) Manhattan
b) Minkowski
c) Tanimoto
d) ROUGE

Câu hỏi 2 Cho hàm tính khoảng cách sau đây:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Hàm tính độ đo khoảng cách trên được gọi là:
a) Manhattan
b) Minkowski
c) Euclidean
d) Tanimoto

Câu hỏi 3 Hoàn thành đoạn code sau đây để được thứ tự đúng cho bài toán phân loại hoa Iris dử dụng mô hình KNN?

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# Load the diabetes dataset
iris_X, iris_y = datasets.load_iris(return_X_y=True)

# Split train:test = 8:2
```

```
X_train, X_test, y_train, y_test = train_test_split(
    iris_X,
    iris_y,
    test_size=0.2,
    random_state=42
)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build KNN Classifier
*** Your code here ***

# Predict and Evaluate test set
y_pred = knn_classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

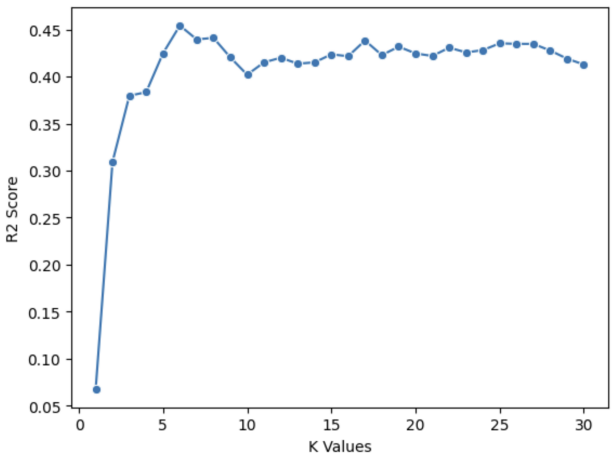
```
a)
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

b)
knn_classifier = KNeighborsRegessor(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

c)
knn_classifier = KNeighborsModel(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

d)
knn_classifier = KNeighbors(n_neighbors=5)
knn_classifier.fit(X_train, y_train)
```

Câu hỏi 4 Quan sát hình dưới về ảnh hưởng của giá trị K đến độ đo R_2Score.



Hình 3: Ảnh hưởng của giá trị K đến độ đo R_2Score.

Giá trị K nào trong các giá trị sau đây là tốt nhất cho mô hình KNN ở trong hình 5?

a) 10
b) 7
c) 17
d) 4

Câu hỏi 5 Sắp xếp các đoạn code sau đây để được thứ tự đúng cho bài toán dự đoán chỉ số bệnh trên bô dữ liệu *Diabetes* dử dụng mô hình KNN?

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

Paragraph A:
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

Paragraph B:
# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

Paragraph C:
# Build KNN model
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train, y_train)

Paragraph D:
# Split train:test = 8:2
X_train, X_test, y_train, y_test = train_test_split(
    diabetes_X,
    diabetes_y,
    test_size=0.2,
    random_state=42
)
```

Thứ tự đúng của các đoạn trên là:

a) D - C - B - A
b) A - D - B - C
c) C - B - A - D
d) B - C - D - A

Bag-of-Words (BoW): Đây là phương pháp đơn giản để biểu diễn văn bản bằng cách đếm số lần xuất hiện của mỗi từ trong văn bản.

TF-IDF (Term Frequency-Inverse Document Frequency): Phương pháp này cải tiến BoW bằng cách tính toán tần suất xuất hiện của từ trong văn bản và điều chỉnh theo tần suất xuất hiện của từ đó trong toàn bộ tập dữ liệu.

Câu hỏi 6 Phương pháp nào không được sử dùng để biểu diễn văn bản thành các giá trị vector là:

a) Bag-of-Word
b) TF-IDF
c) One Hot Encoding
d) F-Score

One-Hot Encoding: Phương pháp này biểu diễn mỗi từ bằng một vector nhị phân, trong đó chỉ có một phần tử là 1 và các phần tử còn lại là 0.

F-Score là một thước đo hiệu suất của mô hình, thường được sử dụng trong các bài toán phân loại để đánh giá độ chính xác và độ bao phủ của mô hình, chứ không phải là phương pháp biểu diễn văn bản thành các giá trị vector.

Câu hỏi 7 Hoàn thành đoạn code sau đây cho bài toán phân loại văn bản sử dụng mô hình KNN trên bộ dữ liệu đánh giá phim:

```
# Import library
!pip install -q datasets
import numpy as np
from datasets import load_dataset
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
# Load IMDB dataset
imdb = load_dataset("imdb")
imdb_train, imdb_test = imdb['train'], imdb['test']
# Convert text to vector using BoW
*** Your code here ***
vectorizer =
X_train =
X_test =
y_train = np.array(imdb_train['label'])
y_test = np.array(imdb_test['label'])

# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build KNN Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=1, algorithm='ball_tree')
knn_classifier.fit(X_train, y_train)

# predict test set and evaluate
y_pred = knn_classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Chọn đáp án đúng trong các đáp án sau đây:

```
a)
vectorizer = CountVectorizer(max_features=1000)
X_train = vectorizer.fit_transform(imdb_train['text']).toarray()
X_test = vectorizer.transform(imdb_test['label']).toarray()

b)
vectorizer = CountVectorizer(max_features=1000)
X_train = vectorizer.fit_transform(imdb_train['label']).toarray()
X_test = vectorizer.transform(imdb_test['text']).toarray()

c)
vectorizer = CountVectorizer(max_features=1000)
X_train = vectorizer.fit_transform(imdb_train['text']).toarray()
X_test = vectorizer.transform(imdb_test['text']).toarray()

d)
vectorizer = CountVectorizer(max_features=1000)
X_train = vectorizer.fit_transform(imdb_train['text']).toarray()
X_test = vectorizer.transform(imdb_test['label']).toarray()
```

2. K-Mean

2.1 Lý thuyết

Thuật toán **K-Means** là một thuật toán phân cụm phổ biến và được sử dụng rộng rãi trong các bài toán học máy (machine learning) và khai phá dữ liệu (data mining). Mục tiêu của K-Means là phân chia n điểm dữ liệu thành k cụm sao cho các điểm trong cùng một cụm có độ tương đồng cao nhất.

Quy trình cơ bản của K-Means bao gồm các bước sau:

1. **Khởi tạo**: Lựa chọn ngẫu nhiên k điểm từ tập dữ liệu làm các tâm cụm ban đầu.
2. **Gán nhãn**: Với mỗi điểm dữ liệu, tính khoảng cách từ điểm đó tới mỗi tâm cụm và gán nó vào cụm có tâm gần nhất. Khoảng cách thường được đo bằng khoảng cách Euclid, được tính như sau:

$$d(\mathbf{x}_i, \mathbf{c}_j) = \sqrt{\sum_{l=1}^{m} (x_{il} - c_{jl})^2}$$

   trong đó, $\mathbf{x}_i$ là điểm dữ liệu, $\mathbf{c}_j$ là tâm cụm, và m là số chiều của dữ liệu.

3. **Cập nhật tâm cụm**: Sau khi đã gán nhãn cho tất cả các điểm, cập nhật lại vị trí của các tâm cụm bằng cách tính trung bình các điểm dữ liệu trong mỗi cụm:

$$\mathbf{c}_j = \frac{1}{|\mathcal{C}_j|} \sum_{\mathbf{x}_i \in \mathcal{C}_j} \mathbf{x}_i$$

   trong đó, $\mathcal{C}_j$ là tập hợp các điểm dữ liệu thuộc cụm j.

4. **Lặp lại**: Lặp lại quá trình gán nhãn và cập nhật tâm cụm cho đến khi các tâm cụm không còn thay đổi đáng kể hoặc đạt đến số lần lặp tối đa.

Thuật toán K-Means có độ phức tạp tính toán là $O(n \times k \times t \times m)$, trong đó t là số lần lặp và m là số chiều của dữ liệu. K-Means thường được sử dụng vì tính đơn giản và hiệu quả của nó trong việc phân cụm dữ liệu lớn, mặc dù nó có thể không tìm được nghiệm tối ưu toàn cục (global optimization) do phụ thuộc vào việc khởi tạo các tâm cụm ban đầu.

2.2 Ứng dụng

Về Dataset: Trong phần này, chúng ta sẽ Sử dụng Iris dataset, Iris là một tập dữ liệu cổ điển cho các nhiệm vụ phân cụm (clustering) và phân loại (classification). Nó chứa 150 mẫu hoa diên vĩ (Iris flower), mỗi mẫu được mô tả bằng bốn đặc điểm (sepal length, sepal width, petal length, and petal width)

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Hình 4: Dữ liệu hoa diên vĩ (Iris).

Để làm rõ cách K-mean hoạt động, chúng ta sẽ chỉ lấy 2 feature đầu (sepal leangth và sepa width) để áp dụng thuật toán và trực quan hóa chúng trong quá trình tính toán.

Thực hiện import các thư viện cần thiết:

```
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
```

Gọi tới hàm load_iris để lấy bộ iris dataset từ thư viện sklearn. Sau đó lấy 2 features đầu và xem như bài toán của chúng ta đang thực hiện trên tập dữ liệu 2 chiều.

```
iris_dataset = load_iris()
data = iris_dataset.data
data = iris_dataset.data[:, :2]

# Plot data
plt.scatter(data[:, 0], data[:, 1], c='gray')
plt.title("Initial Dataset")
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

Khai báo class KMeans bao gồm các hàm thực hiện đúng như các bước cơ bản được trình bày ở phần 3.1.

```
class KMeans:
def __init__(self, k=3, max_iters=100):
    self.k = k
    self.max_iters = max_iters
    self.centroids = None
    self.clusters = None

def initialize_centroids(self, data):
```

```
        np.random.seed(42)
        self.centroids = data[np.random.choice(data.shape[0], self.k, replace=False)]

    def euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum(np.power(x1 - x2, 2)))

    def assign_clusters(self, data):
        distances = np.array([[self.euclidean_distance(x, centroid) for centroid in self.
    centroids] for x in data])

        return np.argmin(distances, axis=1)

    def update_centroids(self, data):
        return np.array([data[self.clusters == i].mean(axis=0) for i in range(self.k)])

    def fit(self, data):
        self.initialize_centroids(data)

        for i in range(self.max_iters):
            self.clusters = self.assign_clusters(data)

            self.plot_clusters(data, i)

            new_centroids = self.update_centroids(data)

            if np.all(self.centroids == new_centroids):
                break

            self.centroids = new_centroids

        self.plot_final_clusters(data)


    def plot_clusters(self, data, iteration):
        plt.scatter(data[:, 0], data[:, 1], c=self.clusters, cmap='viridis', marker='o',
    alpha=0.6)
        plt.scatter(self.centroids[:, 0], self.centroids[:, 1], s=300, c='red', marker='x')
        plt.title(f"Iteration {iteration + 1}")
        plt.xlabel('Sepal length')
        plt.ylabel('Sepal width')
        plt.show()

    def plot_final_clusters(self, data):
        plt.scatter(data[:, 0], data[:, 1], c=self.clusters, cmap='viridis', marker='o',
    alpha=0.6)
        plt.scatter(self.centroids[:, 0], self.centroids[:, 1], s=300, c='red', marker='x')
        plt.title("Final Clusters and Centroids")
        plt.xlabel('Sepal length')
        plt.ylabel('Sepal width')
        plt.show()
```

Cuối cúng, chúng ta khởi tạo đối tượng của lớp K-Mean và xem cách mà thuật toán này thực hiện phân cụm.

```
kmeans = KMeans(k=2)
kmeans.fit(data)
```

```
    kmeans = KMeans(k=3)
    kmeans.fit(data)
    kmeans = KMeans(k=4)
    kmeans.fit(data)
```

Tham khảo thêm ở link này.

2.3 Câu hỏi trắc nghiệm

Chú ý: đáp án của câu hỏi trước sẽ giúp trả lời cho các câu hỏi sau
Dưới đây là một bộ dữ liệu nhỏ với 3 thuộc tính và 8 mẫu:

Feature 1	Feature 2	Feature 3
2.0	3.0	1.5
3.0	3.5	2.0
3.5	3.0	2.5
8.0	8.0	7.5
8.5	8.5	8.0
9.0	8.0	8.5
1.0	2.0	1.0
1.5	2.5	1.5

Bảng 1: Bộ dữ liệu nhỏ với 3 thuộc tính và 8 mẫu.

Câu hỏi 8 Giả sử bạn khởi tạo 2 centroids cho bộ dữ liệu trên. Chọn hai điểm dữ liệu làm centroids ban đầu từ bảng trên.

A Cụm 1: (2.0, 3.0, 1.5) và Cụm 2: (8.0, 6.0, 7.5)

B Cụm 1: (3.5, 3.0, 2.5) và Cụm 2: (1.5, 4.5, 1.5)

C Cụm 1: (6.5, 8.5, 8.0) và Cụm 2: (1.0, 3.0, 1.0)

D Cụm 1: (2.0, 3.0, 1.5) và Cụm 2: (1.0, 2.0, 1.0)

Câu hỏi 9 Tính khoảng cách Euclid giữa điểm dữ liệu (2.0, 3.0, 1.5) và centroid (8.0, 8.0, 7.5). Kết quả gần nhất với giá trị nào?

A 8.6

B 9.0

C 7.7

D 10.0

Câu hỏi 10 Với các centroids đã được khởi tạo, xác định cụm của điểm dữ liệu (3.0, 3.5, 2.0) dựa trên khoảng cách Euclid đến các centroids. Điểm này thuộc cụm nào?

A Cụm 1

B Cụm 2

Câu hỏi 11 Sau khi gán các điểm dữ liệu vào các cụm, điểm centroid mới cho cụm 1 (bao gồm các điểm dữ liệu thuộc cụm 1) sẽ là gì?

A Trung bình của các điểm dữ liệu (2.0, 3.0, 1.5), (3.0, 3.5, 2.0).

B Trung bình của các điểm dữ liệu (8.0, 8.0, 7.5), (8.5, 8.5, 8.0), (9.0, 8.0, 8.5).

C Trung bình của tất cả các điểm dữ liệu thuộc centroid 1.

D Trung bình của các điểm dữ liệu (1.0, 2.0, 1.0), (1.5, 2.5, 1.5).

Câu hỏi 12 Sau khi cập nhật các centroids, nếu chúng không thay đổi sau một lần lặp, thuật toán K-Means có dừng lại không?

A Có, thuật toán dừng lại khi centroids không thay đổi.

B Không, thuật toán sẽ tiếp tục chạy thêm một số lần lặp.

C Có, nhưng chỉ dừng lại nếu đạt số lần lặp tối đa.

D Không, thuật toán sẽ không bao giờ dừng lại.

Câu hỏi 13 Hãy sửa lại code K-Mean bằng numpy ở phần ứng dụng với bộ dữ liệu mới, nếu k = 3, xác định các điểm tâm cụm (centroids) (chọn đán án gần đúng nhất)?

A (5.22 5.0 3.33), (7.7 7.5 6.7), (2.3 3.2 2.2)

B (3.25, 3.25, 2.25), (8.5, 8.17, 8.0), (1.5, 2.5, 1.33)

C (6.2 6.0 4.33), (6.7 6.5 5.72), (3.33 4.2 3.25)

D (5.2 2.0 3.33), (7.7 5.5 6.72), (4.33 3.2 4.25)

- Hết -