

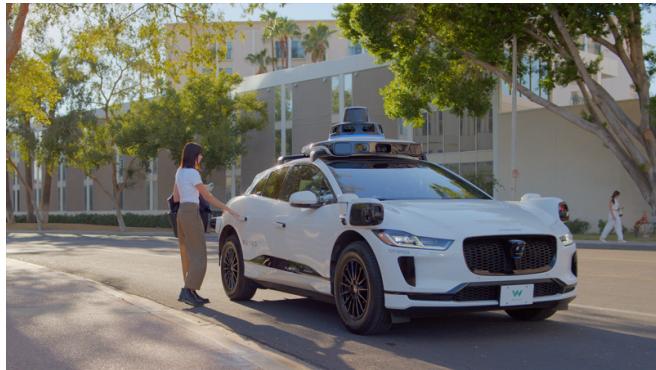
AI VIET NAM – AIO COURSE 2024

# XAI Introduction: LIME and ANCHOR

Truong-Binh Duong, Phuc-Thinh Nguyen, Quang-Vinh Dinh

## I Dẫn Nhập

Trong những năm gần đây, trí tuệ nhân tạo không ngừng phát triển và len lỏi vào mọi khía cạnh của cuộc sống. Từ các ứng dụng như ChatGPT – một trợ lý ảo được đông đảo người dùng ưa chuộng – cho đến những công nghệ tiên tiến như xe tự hành (hình 1), AI đang ngày càng khẳng định vai trò quan trọng trong xã hội hiện đại. Sự bùng nổ này mang lại nhiều tiện ích và hứa hẹn thay đổi sâu sắc cách con người làm việc cũng như sinh hoạt.



Hình 1: Một chiếc taxi không người lái tại Mỹ. Nguồn

Nhưng đằng sau những sự phát triển đáng kinh ngạc này là một thách thức lớn về tính minh bạch và khả năng giải thích. Các mô hình AI, đặc biệt là các mô hình học sâu, thường hoạt động như những “hộp đen”(black box) (hình 2). Thuật ngữ này ám chỉ việc người dùng đưa dữ liệu đầu vào cho mô hình và nhận kết quả đầu ra mà không thực sự hiểu rõ quy trình bên trong. Giống như một chiếc hộp kín, việc không thể nhìn thấu các thuật toán phức tạp đang diễn ra khiến chúng ta mù mờ về lý do mô hình đưa ra kết luận.



Hình 2: Minh họa cho các mô hình "hộp đen".

Điều này có thể không phải là vấn đề lớn khi AI chỉ được dùng cho mục đích giải trí hoặc hỗ trợ đơn giản, nhưng trong những lĩnh vực quan trọng như y tế, tài chính hay pháp luật, sự “mù mờ” đó có thể đe dọa nghiêm trọng. Một ví dụ điển hình là việc ứng dụng AI trong hệ thống đánh giá rủi ro tái phạm của tội phạm, những hệ thống này có xu hướng gán nhãn rủi ro cao hơn cho người da màu so với người da trắng, dù lịch sử phạm tội của người da trắng nguy hiểm hơn nhiều (ví dụ như hình 3). Điều này làm dậy lên mối lo ngại về tính công bằng và nhân quyền khi sử dụng AI trong lĩnh vực tư pháp.



Hình 3: Minh họa sự thiên vị của hệ thống COMPAS trong đánh giá nguy cơ tái phạm tội. Nguồn

Những thách thức này đã thúc đẩy sự ra đời của một lượng lớn các nghiên cứu mới trong lĩnh vực eXplainable AI (XAI). Các nghiên cứu này tập trung phát triển các thuật toán và phương pháp mới không chỉ có khả năng giải thích cơ chế nội bộ mà còn giúp người ta quyết định hiểu được lý do đằng sau các dự đoán của mô hình. Mục tiêu tổng quát của XAI là tạo ra lời giải thích dễ hiểu đối với con người về lý do (why) và cách thức (how) đưa ra những dự đoán cụ thể từ các hệ thống học máy hoặc học sâu. Trong tài liệu này, chúng ta sẽ tập trung vào việc giải thích các mô hình học máy có giám sát (Supervised Machine Learning).

## II Các khái niệm liên quan

### 1 Interpretability và Explainability

Trước khi đi vào các phương pháp giải thích, chúng ta sẽ tìm hiểu hai khái niệm thường được nhắc đến trong XAI, đó là interpretability (tạm dịch là khả năng diễn giải) và explainability (tạm dịch là khả năng giải thích). Mặc dù chưa có định nghĩa chung thống nhất và một số tài liệu sử dụng chúng thay thế cho nhau, nhưng ở đây chúng ta có thể phân biệt như sau:

- **Interpretability** đề cập đến khả năng hiểu trực tiếp quá trình hoạt động của một mô hình hoặc thuật toán. Một mô hình có tính khả năng diễn giải cao thường có cấu trúc đơn giản, trực quan và dễ hình dung, ví dụ như hồi quy tuyến tính hay cây quyết định. Chúng ta có thể nhanh chóng nắm bắt được cơ chế bên trong của mô hình qua các tham số và cấu trúc của nó.

Ví dụ: Mô hình hồi quy tuyến tính có thể được coi là khả năng diễn giải (interpretable model) vì chúng ta có thể hiểu được mối quan hệ giữa biến đầu vào và đầu ra thông qua các trọng số. Chẳng hạn, với một mô hình dự đoán giá nhà:

$$y = w_1x_1 + w_2x_2 + b$$

Trong đó:

- $y$  là giá nhà,  $x_1$  là diện tích,  $x_2$  là số phòng ngủ,  $b$  là hệ số bias.
- $w_1, w_2$  là các hệ số hồi quy (trọng số) thể hiện mức độ ảnh hưởng của  $x_1$  và  $x_2$ .

Nếu  $w_1 = 0.8$  và  $w_2 = 0.3$  sẽ cho thấy diện tích có ảnh hưởng lớn hơn số phòng ngủ trong việc định giá nhà. Mô hình này đơn giản, dễ hiểu, và không cần thêm công cụ giải thích.

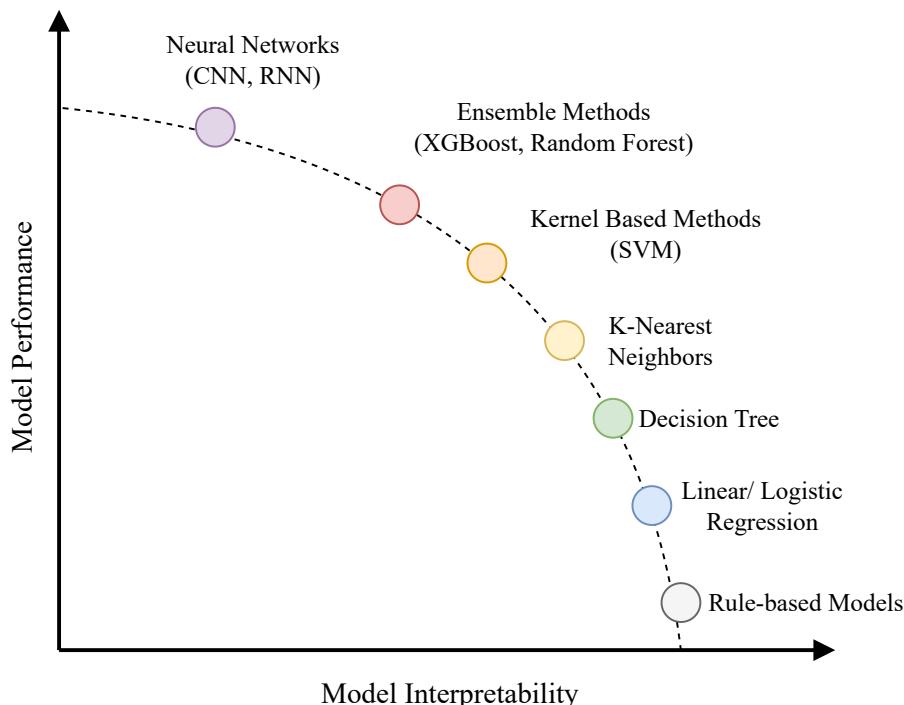
- **Explainability** tập trung làm rõ lí do và bằng chứng cho quyết định của các mô hình phức tạp theo cách mà đa số người dùng – kể cả những người không chuyên – có thể hiểu được mà không nhất thiết phải làm rõ toàn bộ cơ chế nội bộ.

Ví dụ: Một mạng nơ-ron sâu nhận diện khuôn mặt thường không có interpretability cao vì người dùng không thể giải thích được ý nghĩa của từng trọng số trong hàng triệu trọng số mà mạng học được. Thay vào đó, chúng ta cần sử dụng các phương pháp XAI để giúp mô hình trở nên “explainable”, giúp ta biết được phần nào của khuôn mặt ảnh hưởng đến kết quả của mô hình.

## 2 Mối Quan Hệ giữa Performance và Interpretability

Trong học máy, có một sự đánh đổi thường xuyên giữa performance (hiệu suất của mô hình) và interpretability. Sự đánh đổi này không phải lúc nào cũng tuyệt đối, nhưng là một xu hướng chung khi lựa chọn mô hình:

- Mô hình đơn giản, như hồi quy tuyến tính hoặc cây quyết định, thường dễ hiểu và minh bạch, nhưng hiệu suất của chúng có thể thấp hơn, đặc biệt trong các bài toán phức tạp.
- Mô hình phức tạp, như mạng nơ-ron sâu hoặc các mô hình ensemble, thường đạt hiệu suất cao hơn nhờ khả năng học các mối quan hệ phi tuyến tính hoặc tương tác phức tạp giữa các đặc trưng, nhưng đồng thời chúng mất đi tính khả thi giải.



Hình 4: Đồ thị minh họa tương đối mối quan hệ giữa performance và interpretability của các mô hình.

### 3 Phân Loại Các Thuật Toán Giải Thích

Các thuật toán giải thích trong XAI có thể được phân loại dựa trên hai tiêu chí chính:

- **Thời điểm giải thích:**

- Giải thích nội tại (Intrinsic): Các mô hình này có khả năng giải thích ngay từ đầu, ví dụ như hồi quy tuyến tính hoặc cây quyết định. Những phương pháp này thường chỉ áp dụng cho một số loại mô hình cụ thể (model-specific).
- Giải thích hậu kiểm định (Post-hoc): Các phương pháp này được áp dụng sau khi mô hình đã được huấn luyện. Post-hoc thường có tính tổng quát cao hơn và có thể áp dụng với hầu hết các mô hình (model-agnostic).

- **Phạm vi áp dụng:**

- Toàn cục (Global): Cung cấp cái nhìn tổng quan về toàn bộ hoạt động của mô hình, giúp hiểu cách mô hình ra quyết định trên toàn bộ dữ liệu.
- Cục bộ (Local): Tập trung vào giải thích một dự đoán cụ thể của mô hình, phù hợp với các trường hợp cần hiểu rõ từng quyết định riêng lẻ.

Trong phần tiếp theo, chúng ta sẽ tìm hiểu về thuật toán LIME và ANCHOR, đây là hai thuật toán thuộc loại Post-hoc, Model-agnostic và Local.

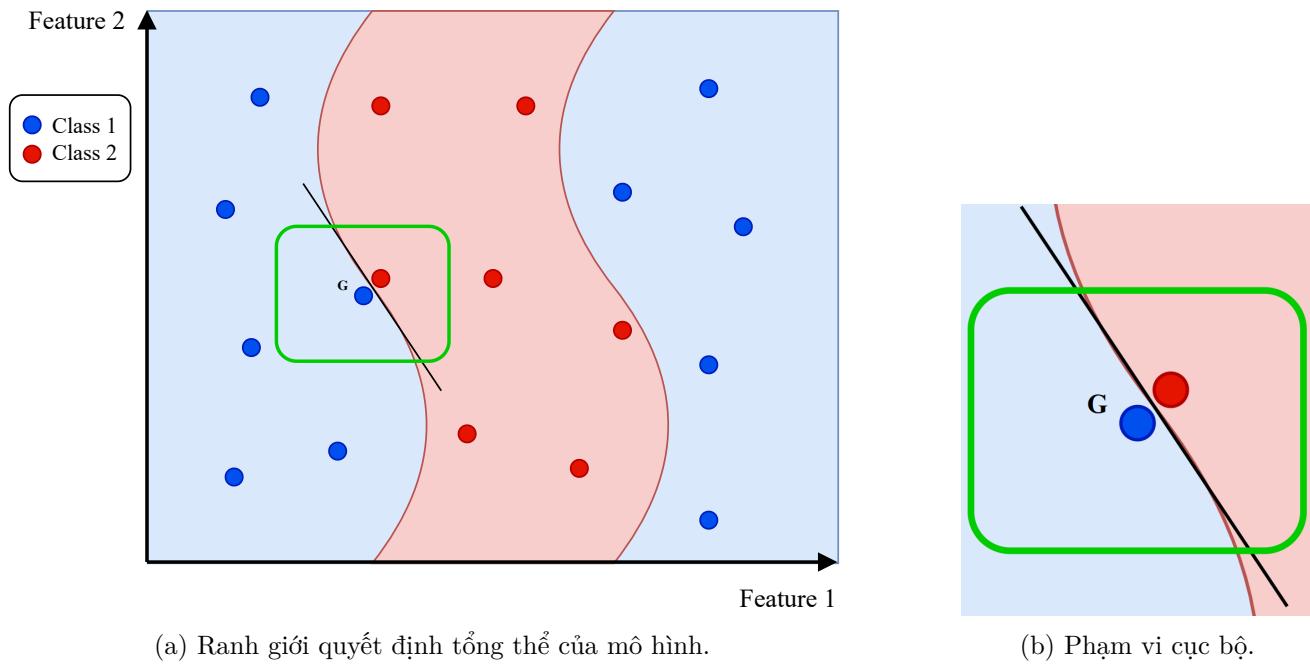
## III LIME

LIME (Local Interpretable Model-Agnostic Explanations) là một thuật toán được giới thiệu vào năm 2016, giúp giải thích cho dự đoán của các mô hình phân loại. Thay vì cố gắng phân tích toàn bộ cấu trúc phức tạp của mô hình, LIME tập trung vào việc diễn giải quyết định của mô hình tại một điểm dữ liệu cụ thể. Thuật toán này hoạt động bằng cách xây dựng một mô hình đơn giản để xấp xỉ hành vi của mô hình gốc trong phạm vi lân cận của điểm đó.

### 1 Ý tưởng

Giả sử chúng ta có một mô hình phân loại sau khi được huấn luyện có ranh giới quyết định được minh họa trong hình 5a. Các vùng màu xanh và đỏ thể hiện hai lớp dữ liệu, với các điểm dữ liệu thuộc lớp 1 (Class 1) được đánh dấu bằng màu xanh dương và lớp 2 (Class 2) được đánh dấu bằng màu đỏ. Xét điểm dữ liệu  $\mathbf{G}$  (màu xanh dương) nằm trong vùng được mô hình phân loại là lớp 1. Làm thế nào để giải thích quyết định của mô hình đối với điểm này?

Nếu xem xét tổng thể, ranh giới quyết định có thể rất phức tạp và khó giải thích. Tuy nhiên, nếu chỉ tập trung vào một vùng nhỏ xung quanh điểm  $G$  (hình 5b), ta có thể xấp xỉ ranh giới quyết định bằng một mô hình tuyến tính đơn giản. Đây chính là ý tưởng cốt lõi của LIME: sử dụng mô hình đơn giản và dễ hiểu để xấp xỉ mô hình gốc trong phạm vi cục bộ.

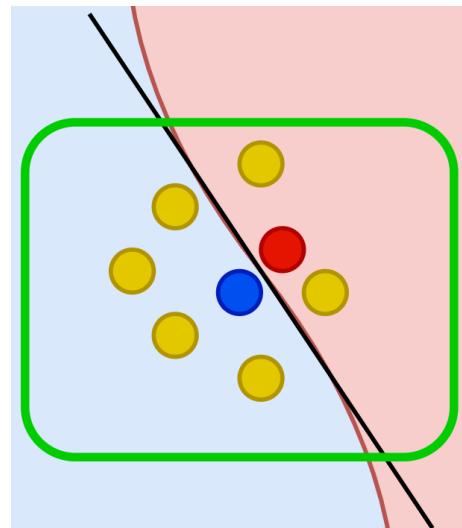


Hình 5: Minh họa nguyên lý hoạt động của LIME.

## 2 Cơ chế hoạt động

Để giải thích dự đoán của mô hình tại một điểm đang xét, LIME thực hiện các bước sau:

- Tạo mẫu dữ liệu lân cận** (Perturbation): Sinh ra các điểm dữ liệu mới xung quanh điểm dữ liệu cần giải thích bằng cách thay đổi (ví dụ như thêm nhiễu, hoán vị, ...) các giá trị đặc trưng của điểm dữ liệu gốc (hình 6).
- Dự đoán bằng mô hình gốc** (Make Prediction): Sử dụng mô hình phác tạp ban đầu để dự đoán nhãn cho các điểm dữ liệu giả đã tạo.
- Tính khoảng cách và trọng số** (Calculate Distance and Weights): Đo khoảng cách giữa mỗi điểm dữ liệu giả và điểm gốc để sử dụng làm trọng số trong bước tiếp theo. Điểm càng gần điểm gốc sẽ có trọng số càng cao.
- Chọn các đặc trưng quan trọng** (Select Features): Lựa chọn một tập hợp nhỏ các đặc trưng (thường ký hiệu là  $m$ ) có ảnh hưởng lớn nhất đến kết quả dự đoán của mô hình phác tạp. Việc giảm số lượng đặc trưng giúp đơn giản hóa mô hình giải thích và tập trung vào những yếu tố quan trọng nhất.
- Huấn luyện mô hình đơn giản** (Fit Simple Model): Sử dụng các mẫu dữ liệu giả đã tạo ra với



Hình 6: Các điểm dữ liệu giả (màu vàng) được tạo quanh G.

các đặc trưng đã chọn và nhãn từ mô hình gốc dự đoán để huấn luyện một mô hình đơn giản. Mô hình này sẽ xấp xỉ hành vi của mô hình phức tạp trong vùng lân cận của điểm dữ liệu gốc.

6. **Giải thích kết quả** (Interpretation): Phân tích các hệ số của mô hình đơn giản để hiểu ảnh hưởng của từng đặc trưng đến dự đoán của mô hình phức tạp tại điểm dữ liệu đang xét.

Quá trình này có thể được tùy chỉnh dựa trên các yếu tố như số lượng mẫu giả được tạo ra, phương pháp đo lường độ tương đồng, số lượng đặc trưng được chọn, và loại mô hình đơn giản được sử dụng. Điều này cho phép LIME cung cấp các giải thích linh hoạt và phù hợp với từng trường hợp cụ thể, giúp ta hiểu được yếu tố nào ảnh hưởng đến quyết định của mô hình mà không cần hiểu toàn bộ cấu trúc của mô hình gốc.

### 3 Từng bước triển khai LIME cho dữ liệu ảnh

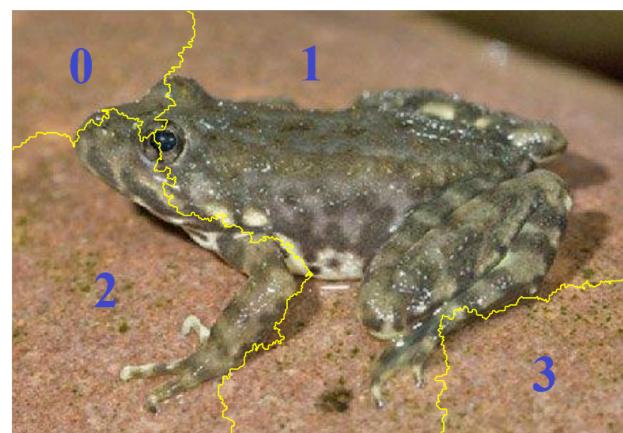
Để hiểu rõ hơn về thuật toán LIME, hãy cùng xem xét một ví dụ minh họa từng bước, trong đó LIME được sử dụng để giải thích cách mô hình ResNet50 đã được huấn luyện trước (pre-trained) dự đoán nhãn của một bức ảnh con ếch.

#### 1. Tạo mẫu dữ liệu lân cận

Đầu tiên ta sẽ chia bức ảnh gốc thành các thành phần nhỏ đồng nhất về màu sắc, texture, ... gọi là superpixel hay segment như hình 7a. Mỗi superpixel có thể xem là một đơn vị đặc trưng trong mô hình giải thích. Trong ví dụ này, để đơn giản, ta sẽ chia ảnh thành 4 superpixel như hình 7b.



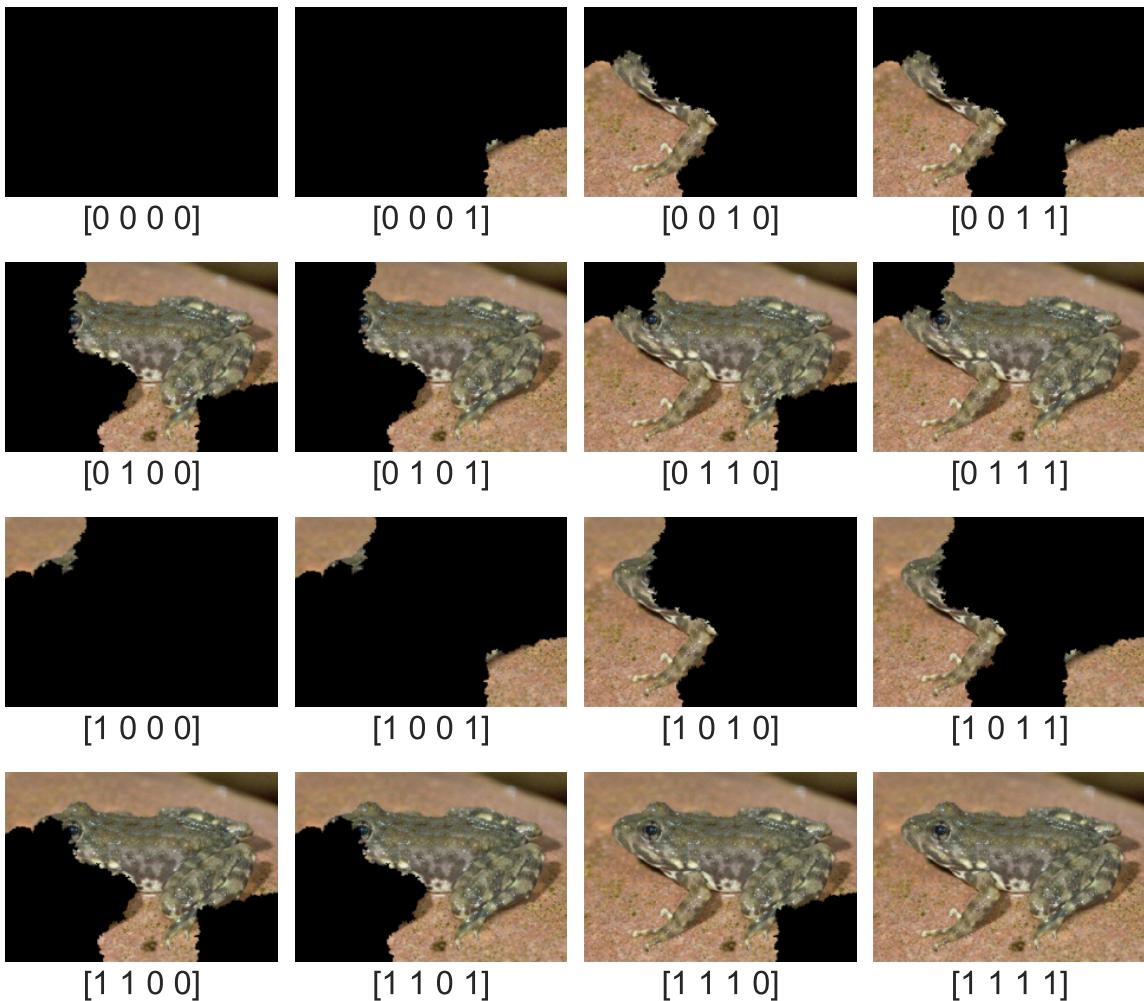
(a) Bức ảnh được phân đoạn thành các superpixel.



(b) Bốn superpixel dùng trong ví dụ.

Hình 7: Minh họa phân đoạn ảnh thành các superpixel.

Mỗi ảnh được biểu diễn bằng một vector nhị phân có chiều bằng số lượng superpixel, trong đó mỗi phần tử có giá trị 1 hoặc 0 tương ứng với việc giữ lại hay loại bỏ superpixel tương ứng. Theo đó, vector của ảnh gốc sẽ là  $X_{\text{original}} = [1, 1, 1, 1]$  vì nó gồm đầy đủ 4 superpixel. Dựa trên biểu diễn này, ta có thể sinh ra các mẫu dữ liệu bằng cách giữ lại hoặc loại bỏ từng superpixel, từ đó tạo ra tổng cộng  $2^4 = 16$  ảnh như hình 8.



Hình 8: Tất cả ảnh được tạo ra và vector tương ứng của chúng.

## 2. Dự đoán nhãn bằng mô hình gốc

Sử dụng mô hình ResNet50, ta thực hiện dự đoán lớp cho 16 ảnh và thu được xác suất thuộc lớp “tailed frog” (lớp cần giải thích quyết định của mô hình) đối với mỗi ảnh. Kết quả dự đoán được thể hiện trong hình 9.

## 3. Tính khoảng cách và trọng số

Khoảng cách giữa mỗi mẫu  $X_i$  và mẫu gốc  $X_{\text{original}}$  được tính dựa trên cosine similarity theo công thức:

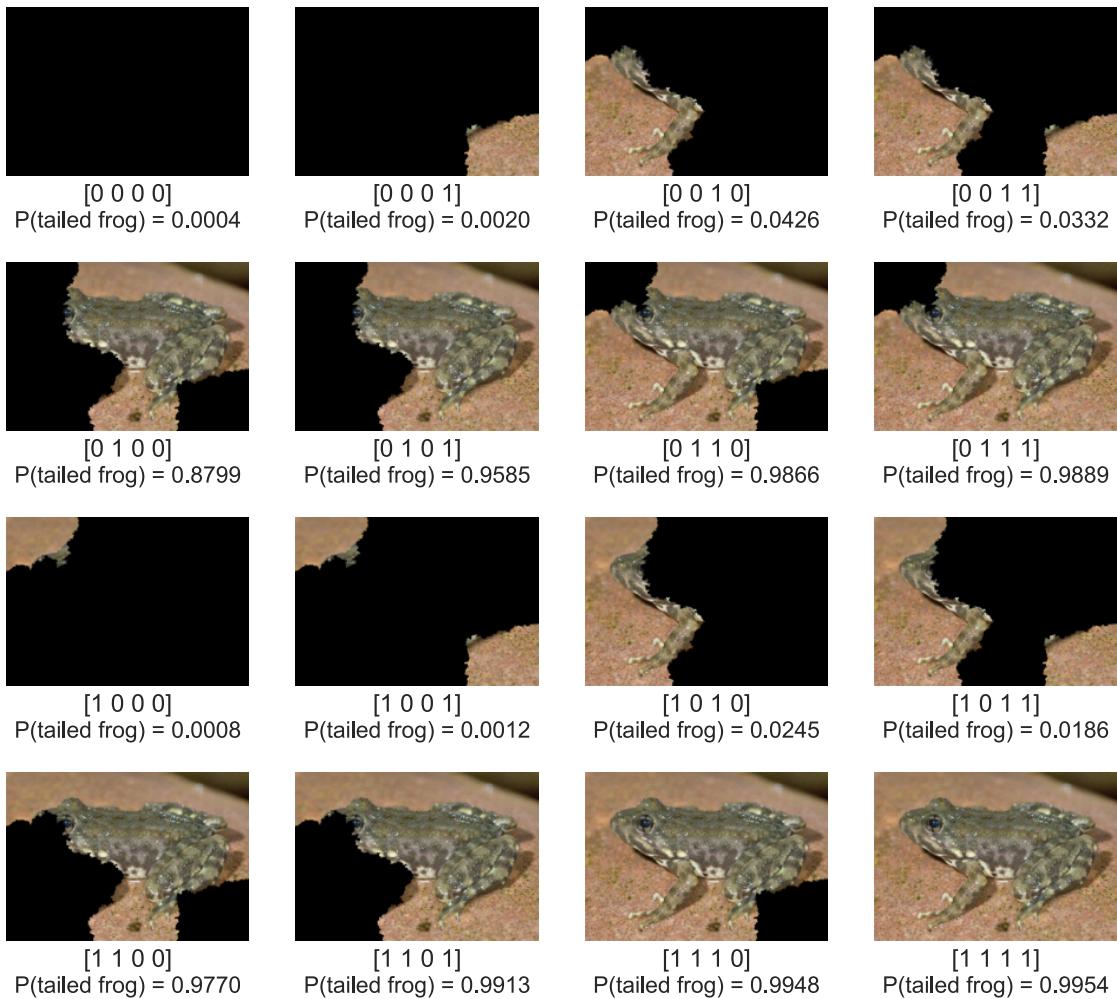
$$d_i = 1 - \text{cosine\_similarity}(X_{\text{original}}, X_i)$$

Ví dụ, với  $X_2 = [0, 1, 0, 0]$ , khoảng cách được tính như sau:

$$d_2 = 1 - \frac{1 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 0}{\sqrt{1^2 + 1^2 + 1^2 + 1^2} \times \sqrt{0^2 + 1^2 + 0^2 + 0^2}} = 0.5$$

Để chuyển khoảng cách thành trọng số, ta dùng hàm kernel mũ:

$$w_i = \exp\left(-\frac{d_i^2}{2\sigma^2}\right)$$

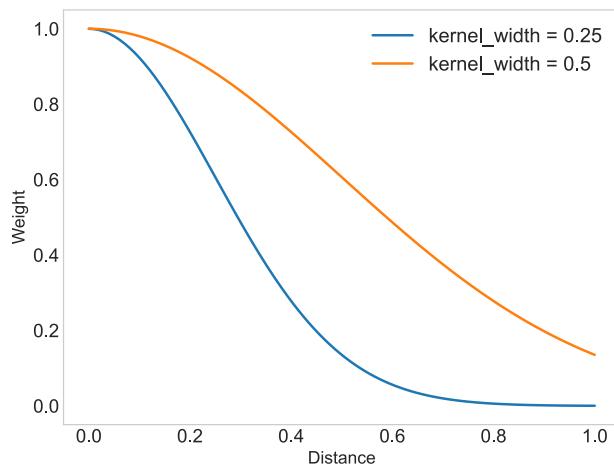


Hình 9: Xác suất dự đoán của mô hình đối với các ảnh được tạo ra.

với  $\sigma$  là tham số kernel width (mặc định  $\sigma = 0.25$ ). Ví dụ, với  $d_2 = 0.5$  và  $\sigma = 0.25$ :

$$w_2 = \exp\left(-\frac{0.5^2}{2 \times 0.25^2}\right) \approx 0.135335$$

Tham số  $\sigma = 0.25$  được lựa chọn sao cho các ảnh càng khác biệt với ảnh gốc sẽ có trọng số càng thấp (tiệm cận về 0), như được minh họa trong hình 10.



Hình 10: Mối tương quan giữa kernel width, khoảng cách và trọng số.

Áp dụng tương tự, ta tính khoảng cách và trọng số cho các ảnh còn lại để có kết quả như bảng 1.

	Superpixel 0	Superpixel 1	Superpixel 2	Superpixel 3	$d$	$w$	$P(\text{tailed frog})$
$X_0$	0	0	0	0	1.000000	0.000335	0.000444
$X_1$	0	0	0	1	0.500000	0.135335	0.001989
$X_2$	0	0	1	0	0.500000	0.135335	0.042633
$X_3$	0	0	1	1	0.292893	0.503440	0.033226
$X_4$	0	1	0	0	0.500000	0.135335	0.879853
$X_5$	0	1	0	1	0.292893	0.503440	0.958524
$X_6$	0	1	1	0	0.292893	0.503440	0.986611
$X_7$	0	1	1	1	0.133975	0.866240	0.988863
$X_8$	1	0	0	0	0.500000	0.135335	0.000794
$X_9$	1	0	0	1	0.292893	0.503440	0.001239
$X_{10}$	1	0	1	0	0.292893	0.503440	0.024486
$X_{11}$	1	0	1	1	0.133975	0.866240	0.018585
$X_{12}$	1	1	0	0	0.292893	0.503440	0.977022
$X_{13}$	1	1	0	1	0.133975	0.866240	0.991281
$X_{14}$	1	1	1	0	0.133975	0.866240	0.994823
$X_{15}$	1	1	1	1	0.000000	1.000000	0.995450

Bảng 1: Khoảng cách, trọng số và xác suất dự đoán lớp “tailed frog” của các mẫu ảnh sinh từ ảnh gốc.

#### 4. Chọn các đặc trưng quan trọng

Trong bước ban đầu, ảnh được biểu diễn dưới dạng 4 đặc trưng đại diện cho 4 superpixel. Ở ví dụ này, chúng ta sẽ chọn ra 1 đặc trưng quan trọng nhất, ứng với 1 superpixel có ảnh hưởng lớn nhất đến quyết định của mô hình cho bước giải thích phía sau.

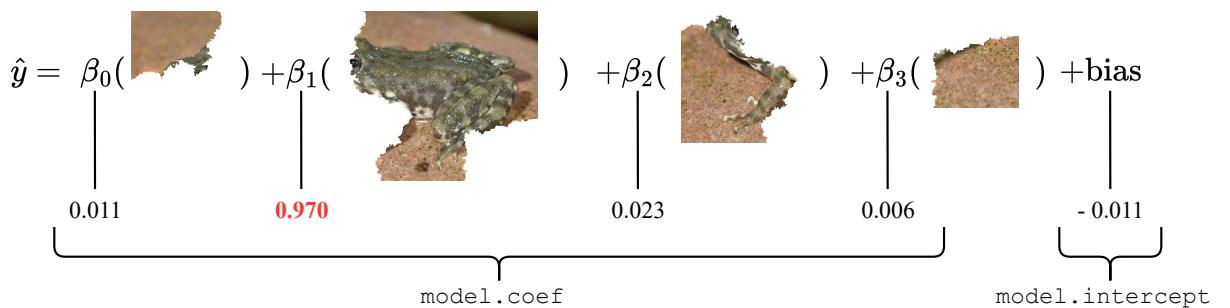
#### 5. Huấn luyện mô hình đơn giản

Ta tiến hành huấn luyện một mô hình hồi quy tuyến tính với đầu vào là các vector nhị phân biểu diễn các mẫu ảnh, đầu ra là xác suất dự đoán thuộc lớp “tailed frog” và sử dụng trọng số  $w_i$  đã tính ở bước

trước.

## 6. Giải thích kết quả

Sau quá trình huấn luyện, các hệ số hồi quy thu được của mô hình sẽ thể hiện với mức độ quan trọng của superpixel tương ứng. Lưu ý rằng hệ số âm không nhất thiết biểu thị superpixel đó ảnh hưởng tiêu cực đến dự đoán của mô hình; trong bối cảnh giải thích, các superpixel có hệ số âm được xem là ít quan trọng hơn.



Hình 11: Hệ số và bias của mô hình hồi quy sau khi huấn luyện.

Hình 11 minh họa rằng superpixel 1 có hệ số hồi quy cao nhất, đồng nghĩa với việc nó tác động nhiều nhất đến quyết định của mô hình. Điều này chứng tỏ LIME đã xác định hiệu quả vùng ảnh có ảnh hưởng lớn nhất đến kết quả phân loại.



Hình 12: Kết quả giải thích của LIME.

## 4 Code

Sau khi đã hiểu rõ LIME thông qua ví dụ trên, chúng ta sẽ tìm hiểu triển khai thuật toán này trong lập trình. Trong Python, thư viện `lime` được phát triển bởi chính tác giả đề xuất phương pháp này, sẽ cung cấp các công cụ giúp giải thích dự đoán của các mô hình phân loại một cách trực quan và dễ sử dụng. Thư viện hỗ trợ giải thích cho nhiều loại dữ liệu khác nhau, bao gồm dữ liệu bảng (tabular), văn

bản (text) và hình ảnh (image). Các bạn có thể tìm hiểu thêm thông qua github và document của thư viện này.

#### 4.1 LIME cho dữ liệu bảng

Trong ví dụ này, chúng ta sử dụng bộ dữ liệu Iris để huấn luyện một mô hình RandomForestClassifier, sau đó sử dụng LIME để giải thích cho dự đoán cụ thể của mô hình.

```
1 from sklearn.datasets import load_iris
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.model_selection import train_test_split
4 from lime.lime_tabular import LimeTabularExplainer
```

Đầu tiên, chúng ta nhập các thư viện cần thiết:

- `sklearn.datasets` được sử dụng để tải bộ dữ liệu Iris.
- `RandomForestClassifier` là mô hình phân loại chúng ta sẽ huấn luyện.
- `train_test_split` giúp chia dữ liệu thành tập huấn luyện và tập kiểm tra.
- `lime.lime_tabular` cung cấp công cụ để giải thích mô hình trên dữ liệu bảng.

```
1 # Load Iris dataset
2 iris = load_iris()
3 X = iris.data
4 y = iris.target
5
6 # Split data into train and test sets
7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Bộ dữ liệu Iris được tải lên gồm thông tin về ba loài hoa khác nhau (`setosa`, `versicolor`, `virginica`), với bốn đặc trưng (`sepal length`, `sepal width`, `petal length`, `petal width`) và được chia thành tập huấn luyện và tập kiểm tra.

```
1 # Train RandomForest model
2 clf = RandomForestClassifier(n_estimators=100, random_state=42)
3 clf.fit(X_train, y_train)
```

Ở bước này, chúng ta khởi tạo và huấn luyện mô hình `RandomForestClassifier`.

```
1 # LIME Tabular Explainer
2 explainer_tabular = LimeTabularExplainer(
3     X_train,
4     feature_names=iris.feature_names,
5     class_names=iris.target_names,
6     discretize_continuous=True,
7     random_state=0
8 )
```

**Khởi tạo LIME Tabular Explainer:**

- `X_train`: Dữ liệu huấn luyện của mô hình.
- `feature_names`: Danh sách tên các đặc trưng.
- `class_names`: Danh sách tên các lớp.
- `discretize_continuous=True`: Chuyển đổi các đặc trưng liên tục thành các khoảng rác, giúp mô hình dễ giải thích hơn.

```

1 # Select a sample from the test set
2 sample_idx = 9
3 sample = X_test[sample_idx]

```

Tại đây, chúng ta chọn mẫu dữ liệu thứ 9 trong tập kiểm tra để phân tích.

```

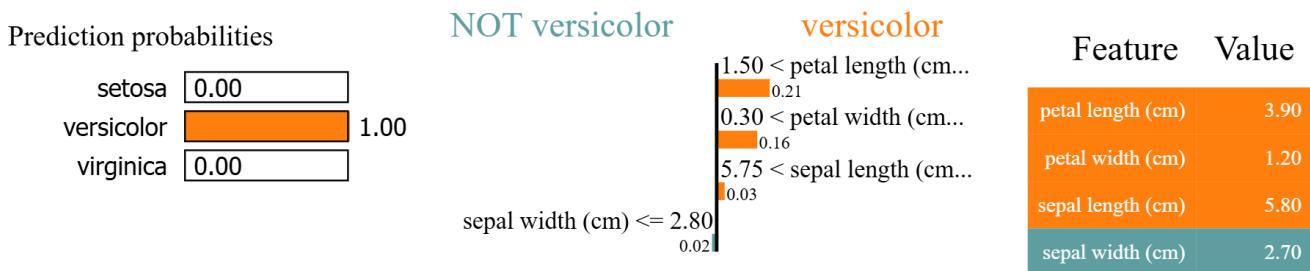
1 explanation = explainer_tabular.explain_instance(sample, clf.predict_proba, num_features=4)
2 explanation.show_in_notebook(show_table=True)

```

### Giải thích dự đoán với LIME:

- `explain_instance(sample, clf.predict_proba, num_features=4):`
  - `sample`: Mẫu dữ liệu cần được giải thích.
  - `clf.predict_proba`: Hàm dự đoán xác suất của mô hình.
  - `num_features=4`: Hiển thị bốn đặc trưng quan trọng nhất ảnh hưởng đến dự đoán.

LIME sẽ tạo ra nhiều điểm dữ liệu giả gần mẫu ban đầu, dự đoán kết quả với mô hình Random Forest, và huấn luyện một mô hình tuyến tính đơn giản để xấp xỉ cách mô hình ra quyết định. Cuối cùng, chúng ta sử dụng hàm `show_in_notebook` để hiển thị trực quan kết quả phân tích của LIME ngay trong môi trường Jupyter Notebook.



Hình 13: Kết quả giải thích của LIME với mẫu dữ liệu thứ 9

Trong hình 13, ta thấy rằng mô hình dự đoán mẫu dữ liệu có đặc trưng [5.8, 2.7, 3.9, 1.2] thuộc lớp `versicolor` với xác suất 100%.

- Bảng **Prediction probabilities** ở bên trái hiển thị xác suất dự đoán của mô hình đối với từng lớp: `setosa`, `versicolor`, và `virginica`. Mô hình dự đoán với xác suất tuyệt đối rằng mẫu này thuộc lớp `versicolor`.
- Biểu đồ giữa thể hiện những đặc trưng quan trọng nhất ảnh hưởng đến quyết định của mô hình. Các đặc trưng được phân thành hai nhóm:
  - Những đặc trưng giúp mô hình dự đoán rằng mẫu thuộc lớp `versicolor` (màu cam, nhãn `versicolor`). Trong trường hợp này, **petal length** lớn hơn 1.50 cm, **petal width** lớn hơn 0.30 cm và **sepal length** lớn hơn 5.75 cm góp phần chính vào dự đoán này, với trọng số ảnh hưởng lần lượt là 0.21, 0.16 và 0.03.
  - Những đặc trưng có tác động nhỏ khiến mô hình có thể phân loại mẫu này không thuộc lớp `versicolor` (màu xanh dương, nhãn **NOT versicolor**). Cụ thể, chỉ có một đặc trưng là **sepal width** nhỏ hơn hoặc bằng 2.80 cm, nhưng do đóng góp của nó rất nhỏ (0.02) nên gần như không ảnh hưởng đến quyết định cuối cùng của mô hình.

- Bảng **Feature - Value** bên phải hiển thị các giá trị của từng đặc trưng, với màu sắc tương ứng thể hiện lớp mà đặc trưng đó tác động.

Nhìn chung, LIME giúp chúng ta hiểu mô hình quyết định mẫu này thuộc lớp `versicolor` chủ yếu dựa vào độ dài và độ rộng của cánh hoa (`petal length` và `petal width`), vì các giá trị này nằm trong đặc trưng của lớp.

## 4.2 LIME cho dữ liệu văn bản

Tiếp theo, chúng ta dùng bộ dữ liệu đánh giá phim của `nltk` để huấn luyện mô hình `RandomForestClassifier` phân loại văn bản. Sau đó, LIME sẽ giúp giải thích tại sao mô hình đưa ra dự đoán đó.

```

1 from sklearn.ensemble import RandomForestClassifier
2 from lime.lime_text import LimeTextExplainer
3 import nltk
4 from sklearn.pipeline import make_pipeline
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.model_selection import train_test_split

```

Các thư viện cần thiết:

- `nltk` giúp tải tập dữ liệu đánh giá phim.
- `RandomForestClassifier` là mô hình phân loại văn bản chúng ta sẽ huấn luyện.
- `TfidfVectorizer` giúp trích xuất đặc trưng từ văn bản bằng phương pháp TF-IDF.
- `train_test_split` giúp chia dữ liệu thành tập huấn luyện và tập kiểm tra.
- `lime.lime_text` cung cấp công cụ để giải thích mô hình trên dữ liệu văn bản.

```

1 # Example text dataset
2 nltk.download('movie_reviews', download_dir='./nltk_data', quiet=True)
3 nltk.data.path.append('./nltk_data')
4 from nltk.corpus import movie_reviews
5
6 texts = [movie_reviews.raw(fileid) for fileid in movie_reviews.fileids()]
7 labels = [1 if fileid.split('/')[0] == 'pos' else 0 for fileid in movie_reviews.fileids()]
8
9 # Split data into train and test sets
10 X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)

```

Ta tải xuống bộ dữ liệu đánh giá phim, thực hiện lấy ra các văn bản nhận xét và nhãn của chúng, gồm tích cực (`pos`) hoặc tiêu cực (`neg`). Sau đó, chia bộ dữ liệu thành tập huấn luyện và kiểm tra theo tỷ lệ 80/20.

```

1 # Train text classifier
2 vectorizer = TfidfVectorizer(lowercase=True, stop_words='english', max_features=1000)
3 text_model = RandomForestClassifier(random_state=42)
4
5 pipeline = make_pipeline(vectorizer, text_model)
6 pipeline.fit(X_train, y_train)

```

Ở bước này, chúng ta tiến hành huấn luyện mô hình như sau:

1. Chuyển đổi văn bản thành dạng vector số bằng phương pháp **TF-IDF** (Term Frequency - Inverse Document Frequency), giúp xác định tầm quan trọng của từng từ trong văn bản dựa trên tần suất của chúng. Bên cạnh đó ta cũng sẽ thực hiện một số bước chuẩn hóa dữ liệu văn bản:

- `lowercase=True`: Chuyển tất cả chữ cái thành chữ thường.
  - `stop_words='english'`: Loại bỏ các stop word trong tiếng Anh như “the”, “is”, “and”.
  - `max_features=1000`: Giữ lại 1000 từ quan trọng nhất để giảm độ phức tạp của mô hình.
2. Khởi tạo mô hình `RandomForestClassifier` dự đoán nhãn của bài đánh giá dựa trên các đặc trưng được trích xuất từ văn bản.
  3. Tiến hành huấn luyện mô hình trên tập dữ liệu huấn luyện.

```

1 # LIME Text Explainer
2 explainer_text = LimeTextExplainer(class_names=['Negative', 'Positive'], random_state=42)

```

#### Khởi tạo LIME Text Explainer:

- `class_names=['Negative', 'Positive']`: Gán nhãn cho hai lớp phân loại, trong đó `Negative` là đánh giá tiêu cực, và `Positive` là đánh giá tích cực.

```

1 # Select a sample from the test set
2 sample_idx = 10
3 sample_text = X_test[sample_idx]

```

Tại đây, chúng ta chọn mẫu văn bản thứ 10 trong tập kiểm tra để phân tích.

```

1 # Generate explanation
2 explanation_text = explainer_text.explain_instance(sample_text, pipeline.predict_proba,
3                                                    num_features=10)
3 explanation_text.show_in_notebook(text=True)

```

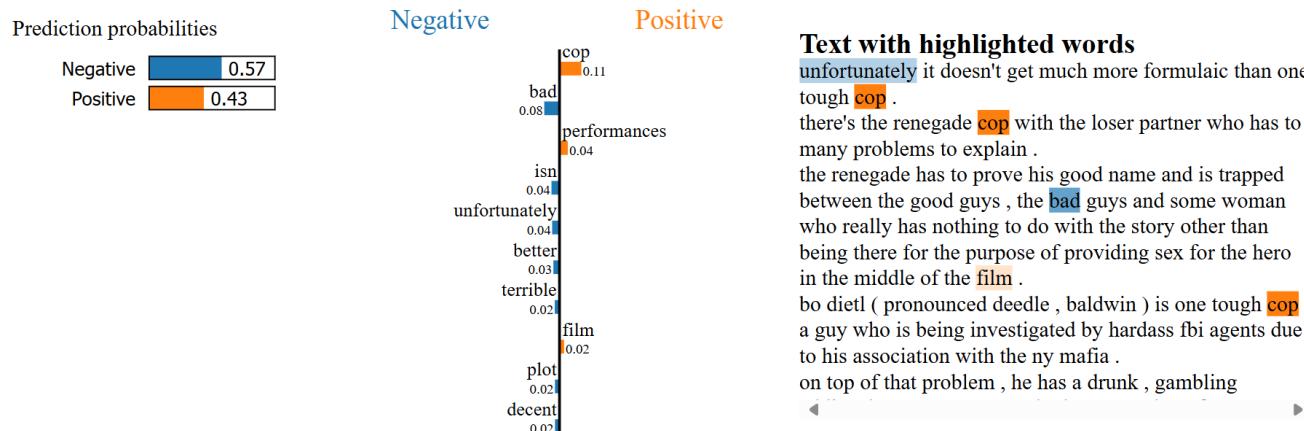
#### Giải thích dự đoán với LIME:

- `explain_instance(sample_text, pipeline.predict_proba, num_features=10)`:
  - `sample_text`: Văn bản cần được giải thích.
  - `pipeline.predict_proba`: Hàm dự đoán xác suất của mô hình.
  - `num_features=10`: Hiển thị 10 từ quan trọng nhất ảnh hưởng đến dự đoán.

LIME sẽ thực hiện các bước sau:

1. Tạo ra nhiều bản sao của văn bản bằng cách thay thế hoặc bỏ đi một số từ.
2. Sử dụng mô hình `RandomForest` để dự đoán xác suất cho từng phiên bản sửa đổi.
3. Xây dựng một mô hình tuyến tính đơn giản để tìm ra những từ có tác động lớn nhất đến kết quả phân loại.

Cuối cùng, chúng ta sử dụng hàm `show_in_notebook` để hiển thị trực quan kết quả phân tích của LIME về các từ quan trọng nhất ảnh hưởng đến dự đoán của mô hình.



Hình 14: Kết quả giải thích của LIME với mẫu văn bản.

Trong hình 14, ta thấy rằng mô hình dự đoán bài đánh giá này thuộc lớp **Negative** với xác suất 57%, trong khi xác suất thuộc lớp **Positive** là 43%.

- Bảng **Prediction probabilities** ở bên trái hiển thị xác suất dự đoán của mô hình đối với hai lớp: **Negative** (màu xanh) và **Positive** (màu cam). Mô hình dự đoán rằng văn bản này có khả năng cao thuộc lớp **Negative**.
- Biểu đồ giữa thể hiện những từ quan trọng nhất ảnh hưởng đến quyết định của mô hình. Các từ được phân thành hai nhóm:
  - Những từ có ảnh hưởng đến dự đoán **Negative** (màu xanh). Trong trường hợp này, các từ như **bad**, **unfortunately**, và **terrible** có trọng số lớn, góp phần chính vào dự đoán tiêu cực.
  - Những từ có ảnh hưởng đến dự đoán **Positive** (màu cam). Một số từ như **cop**, **performances**, và **film** ảnh hưởng không nhỏ vào khả năng mô hình phân loại bài đánh giá này là tích cực.
- Phần **Text with highlighted words** hiển thị nội dung bài đánh giá, trong đó các từ có ảnh hưởng đến dự đoán được tô màu tương ứng với biểu đồ giữa. Độ đậm nhạt của màu cũng thể hiện mức độ ảnh hưởng của từ.
  - Các từ có ảnh hưởng đến lớp **Negative** (như **unfortunately**, **bad**) được tô màu xanh.
  - Các từ có ảnh hưởng đến lớp **Positive** (như **cop**, **film**) được tô màu cam.

LIME đã giúp ta hiểu mô hình dự đoán bài đánh giá này thuộc lớp **Negative** chủ yếu dựa vào sự xuất hiện của các từ mang tính tiêu cực như **bad** và **unfortunately**. Bên cạnh đó vẫn có một số từ mang tính tích cực như **cop** và **film**, nhưng chúng không đủ mạnh để thay đổi kết quả phân loại cuối cùng.

### 4.3 LIME cho dữ liệu hình ảnh

Cuối cùng, chúng ta sẽ dùng LIME để giải thích cho một mô hình Inception v3 được huấn luyện trước để phân loại hình ảnh.

```

1 import torch
2 from torchvision import models, transforms
3 from torchvision.models import Inception_V3_Weights
4 from PIL import Image
5 import requests

```

```

6  from io import BytesIO
7  from torch.nn import Softmax
8  import numpy as np
9  from lime import lime_image
10 from skimage.segmentation import mark_boundaries
11 import matplotlib.pyplot as plt
12
13 # Set random seed for reproducibility
14 seed = 42
15 torch.manual_seed(seed)
16 np.random.seed(seed)
17 random.seed(seed)

```

Đầu tiên chúng ta xác định random seed để cố định kết quả và import các thư viện cần thiết:

- `torch, torchvision`: Thư viện chính để xây dựng và xử lý các mô hình học sâu, bao gồm cả mô hình Inception v3.
- `PIL.Image`: Xử lý và chuyển đổi hình ảnh.
- `requests, BytesIO`: Dùng để tải hình ảnh từ Internet.
- `lime.lime_image`: LIME dành cho dữ liệu hình ảnh.
- `matplotlib.pyplot`: Thư viện để trực quan hóa kết quả, vẽ biểu đồ và hiển thị hình ảnh.
- `skimage.segmentation.mark_boundaries`: Giúp đánh dấu các vùng quan trọng trong hình ảnh.

```

1 # Load pre-trained Inception v3 model
2 torch.hub.set_dir("./model")
3 model = models.inception_v3(weights=Inception_V3_Weights.DEFAULT, transform_input=False)
4 model.eval()
5
6 # Define image preprocessing pipeline
7 transform = transforms.Compose([
8     transforms.Resize((299, 299)),
9     transforms.ToTensor(),
10    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
11])

```

Tiếp theo, chúng ta cần tải mô hình Inception v3 đã được huấn luyện trước và định nghĩa hàm tiền xử lí ảnh.

```

1 # URL to fetch the ImageNet class labels
2 label_url = "https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt"
3
4 # Download and save the ImageNet class labels
5 def download_and_save_labels(url, save_path):
6     """Downloads and saves the ImageNet class labels."""
7     response = requests.get(url)
8     response.raise_for_status() # Check if the request was successful
9     with open(save_path, 'w') as f:
10         f.write(response.text)
11     return save_path
12
13 label_path = download_and_save_labels(label_url, 'imagenet_classes.txt')
14
15 with open(label_path) as f:
16     labels = [line.strip() for line in f.readlines()]
17

```

```

18 # Load the image from URL
19 image_url = 'https://github.com/EliSchwartz/imagenet-sample-images/blob/master/n01644900_tailed_frog
              .JPEG?raw=true'
20 response = requests.get(image_url)
21 image = Image.open(BytesIO(response.content)).convert('RGB')
22 input_tensor = transform(image).unsqueeze(0)

```

Kế đến chúng ta tải một bức hình và thông tin các label từ tập dữ liệu ImageNet để chuẩn bị dữ liệu cho ví dụ này.

```

1 # Predict the class probabilities
2 with torch.no_grad():
3     output = model(input_tensor)
4     probs = Softmax(dim=1)(output)
5
6 # Get the top predicted class
7 class_idx = torch.argmax(probs, 1).item()
8 print(f"Predicted class index: {class_idx}")
9 print(f"Predicted class label: {class_label}")

```

Ta dùng mô hình thực hiện dự đoán nhãn cho bức ảnh.

```

1 explainer_image = lime_image.LimeImageExplainer(random_state=seed)
2
3 def predict_fn(images):
4     model.eval()
5     batch = torch.stack([transform(Image.fromarray(img)) for img in images], dim=0)
6     with torch.no_grad():
7         outputs = model(batch)
8         probabilities = Softmax(dim=1)(outputs)
9     return probabilities.numpy()
10
11 explanation_image = explainer_image.explain_instance(
12     np.array(image),
13     predict_fn,
14     top_labels=5,
15     hide_color=0,
16     num_samples=1000
17 )

```

Tiếp theo, chúng ta sẽ giải thích cho quyết định của mô hình qua các bước:

1. Khởi tạo LimeImageExplainer để giải thích cho ảnh
2. Định nghĩa hàm dự đoán cho LIME để dự đoán cho nhiều phiên bản của ảnh gốc (sau khi thêm nhiều hoặc thay đổi vùng ảnh).
3. Tiến hành giải thích dự đoán với LIME bằng hàm `explain_instance`:
  - `np.array(image)`: Chuyển hình ảnh thành mảng NumPy để LIME xử lý.
  - `predict_fn`: Hàm dự đoán đã định nghĩa ở trên.
  - `top_labels=5`: Phân tích 5 nhãn dự đoán có xác suất cao nhất.
  - `hide_color=0`: Khi che khuất các vùng ảnh, LIME sẽ tô màu đen (0).
  - `num_samples=100`: Số lượng mẫu ảnh biến đổi mà LIME sẽ tạo để phân tích.

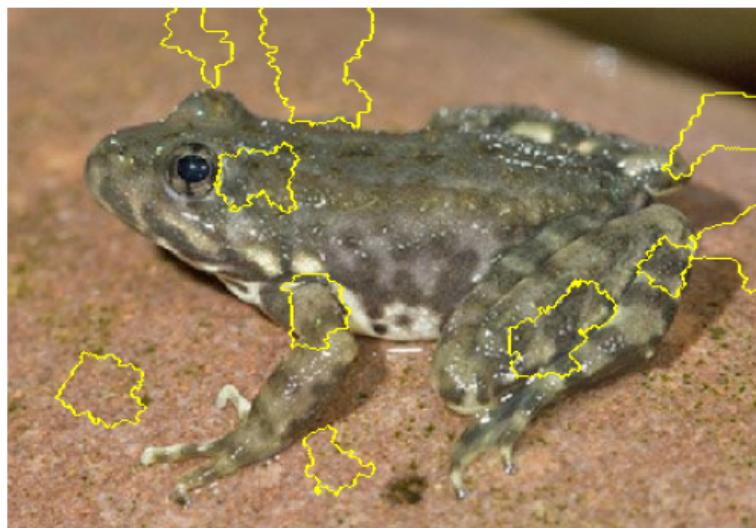
```

1 # Display explanation for the top label
2 image, mask = explanation_image.get_image_and_mask(
3     class_idx, positive_only=True, num_features=10, hide_rest=False
4 )
5
6 plt.imshow(mark_boundaries(image, mask))
7 plt.axis("off")
8 plt.show()

```

Cuối cùng chúng ta sẽ hiển thị kết quả giải thích lên trên ảnh:

- `get_image_and_mask()`: Tạo một phiên bản của hình ảnh với các vùng ảnh quan trọng nhất được làm nổi bật (mask).
  - `class_idx`: Lớp dự đoán cần được giải thích.
  - `positive_only=True`: Chỉ hiển thị các vùng ảnh có ảnh hưởng tích cực đến dự đoán của mô hình.
  - `num_features=10`: Hiển thị 10 vùng ảnh quan trọng nhất.
  - `hide_rest=False`: Giữ nguyên các vùng không quan trọng (không ẩn đi).
- `mark_boundaries()`: Vẽ các đường viền xung quanh các vùng quan trọng khi trực quan.
- `plt.show()`: Hiển thị hình ảnh kết quả với các vùng ảnh quan trọng đã được đánh dấu.



Hình 15: Kết quả giải thích của LIME với dữ liệu hình ảnh.

Trong hình 15, ta thấy các đường viền màu vàng xác định các vùng ảnh quan trọng mà mô hình Inception v3 sử dụng để đưa ra dự đoán. Các vùng nổi bật chủ yếu tập trung quanh phần đầu, chân và phần thân, đây là những điểm đặc trưng giúp mô hình phân biệt đối tượng. Tuy nhiên, một số vùng nhỏ xuất hiện ở phần nền cho thấy mô hình bị ảnh hưởng nhẹ bởi các yếu tố nền có kết cấu hoặc màu sắc tương đồng với đối tượng.

## 5 Hạn chế của LIME

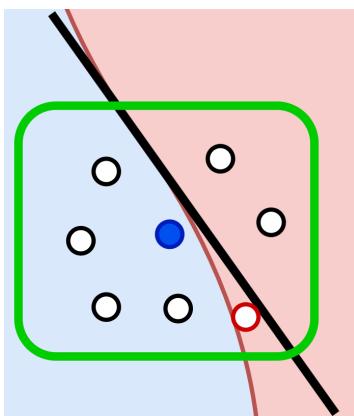
Mặc dù LIME được đánh giá cao nhờ khả năng giải thích cục bộ các dự đoán của mô hình hộp đen, nhưng thuật toán này vẫn tồn tại một số hạn chế nhất định:

- **Không ổn định (Inconsistency):** LIME dựa vào việc lấy mẫu ngẫu nhiên xung quanh điểm dữ liệu cần giải thích để huấn luyện mô hình tuyến tính cục bộ. Điều này có thể dẫn đến các giải thích khác nhau khi chạy lại nhiều lần, khiến kết quả không nhất quán và giảm độ tin cậy của quá trình giải thích.
- **Giới hạn cục bộ:** LIME tập trung xấp xỉ hành vi của mô hình tại một vùng lân cận nhỏ, nên giải thích chỉ phản ánh đúng hành vi của mô hình trong phạm vi hạn chế đó. Do đó, nó không thể nắm bắt được các đặc điểm tổng thể (tổng quát) của mô hình, đặc biệt khi mô hình có ranh giới quyết định phức tạp.
- **Phụ thuộc vào siêu tham số:** Kết quả của LIME phụ thuộc vào nhiều tham số như số lượng mẫu giả được tạo ra, cách tính khoảng cách (trong số) hay phương pháp chuyển đổi dữ liệu sang dạng biểu diễn dễ diễn giải. Nếu các tham số này không được điều chỉnh một cách cẩn thận, các giải thích có thể không phản ánh chính xác những đặc trưng quan trọng của mô hình.
- **Hiệu năng tính toán:** Việc tạo ra các mẫu dữ liệu biến đổi và huấn luyện mô hình giải thích cục bộ có thể tốn thời gian, đặc biệt khi làm việc với dữ liệu có kích thước lớn hoặc dữ liệu có tính chất phức tạp (như dữ liệu hình ảnh). Điều này làm hạn chế khả năng ứng dụng của LIME trong một số tình huống yêu cầu tốc độ tính toán cao.
- **Khó khăn trong việc chuyển đổi biểu diễn đặc trưng:** Để đưa ra được giải thích dễ hiểu cho con người, LIME cần chuyển đổi dữ liệu gốc sang một biểu diễn “diễn giải được” (interpretable representation). Quá trình này có thể làm mất đi một số thông tin quan trọng hoặc không phù hợp với một số loại dữ liệu phức tạp, từ đó ảnh hưởng đến độ chính xác của giải thích.

## IV ANCHOR

### 1 Động lực

Để khắc phục những hạn chế của LIME, chính nhóm tác giả đã đề xuất phương pháp ANCHOR nhằm giải thích quyết định của các mô hình phi tuyến tính – những trường hợp mà các lời giải thích dựa trên LIME (với giả định tuyến tính cục bộ) có thể không chính xác, minh họa ở hình 16.



Hình 16: Minh họa trường hợp không chính xác của LIME

Trong hình minh họa, khung xanh lá cây biểu thị vùng cục bộ mà LIME dùng để xây dựng mô hình tuyến tính (đường màu đen), trong khi ranh giới quyết định thực sự của mô hình là phi tuyến tính (đường cong màu đỏ). Điểm màu trắng viền đỏ cho thấy hạn chế của LIME: mô hình tuyến tính sẽ dự đoán điểm này thuộc class 1 (màu xanh) vì nằm ở phía bên trái của đường thẳng, nhưng thực tế nó thuộc vào class 2 (màu đỏ). Đây là minh chứng cho việc LIME có thể không chính xác khi mở rộng vùng cục bộ.

Điều này đặt ra câu hỏi: “Đâu là ranh giới mà tại đó lời giải thích của LIME bắt đầu trở nên không còn đáng tin cậy?”

Để minh họa rõ hơn vấn đề này, hãy xem xét từ “not”, một từ có thể thay đổi ý nghĩa của câu tùy vào cách kết hợp với các từ khác. Ví dụ ở hình 17 với lời giải thích của ANCHOR và LIME đối với bài toán dự đoán cảm xúc của văn bản, giả sử ta cần phân loại hai câu đánh giá thành tích cực và tiêu cực:

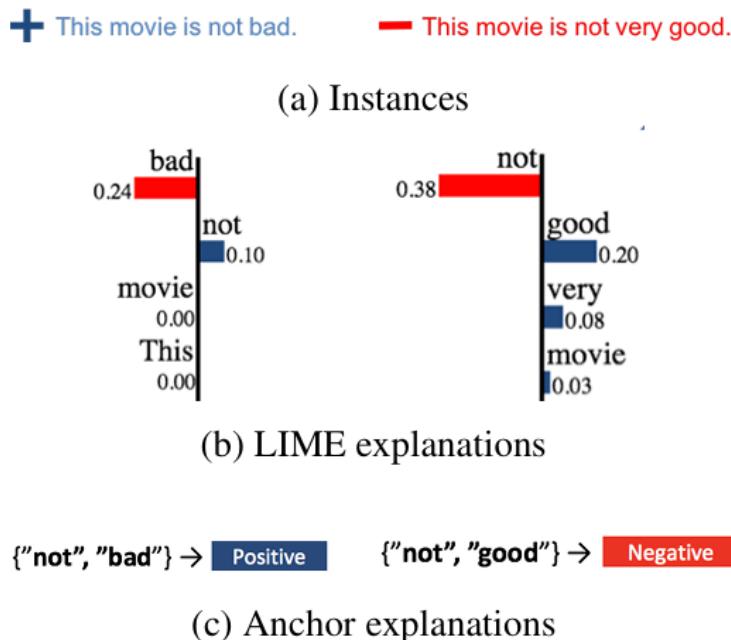
- **Positive:** “*The movie was not bad.*” (Bộ phim không tệ.)
- **Negative:** “*The movie was not good.*” (Bộ phim không hay.)

Khi áp dụng LIME:

- Câu thứ nhất: Từ “not” đóng góp 0.10 vào class Postive
- Câu thứ hai: Từ “not” đóng góp 0.38 vào class Negative

Nếu chỉ dựa trên trọng số của câu thứ nhất, ta có thể nhầm tưởng rằng từ “not” luôn làm tăng khả năng dự đoán tích cực. Tuy nhiên, điều này không chính xác:

- Trong “not bad”, từ “not” đảo ngược nghĩa tiêu cực của “bad”, khiến câu mang ý nghĩa tích cực.
- Trong “not good”, từ “not” đảo ngược nghĩa tích cực của “good”, khiến câu mang ý nghĩa tiêu cực.



Hình 17: Ví dụ về giải thích của LIME và ANCHOR.

Khác với LIME, ANCHOR không chỉ đánh giá tác động của một đặc trưng mà còn xác định một **ranh giới rõ ràng**, giúp diễn giải quyết định của mô hình theo cách dễ hiểu và đáng tin cậy hơn. Ví dụ, thay vì chỉ gán trọng số cho từ “not” như trong LIME, ANCHOR có thể tìm ra các quy tắc sau:

- Nếu câu chứa “not bad”, thì cảm xúc gần như chắc chắn là tích cực.
- Nếu câu chứa “not good”, thì cảm xúc gần như chắc chắn là tiêu cực.

Nhờ cách tiếp cận này, ANCHOR không chỉ mang lại lời giải thích trực quan về quyết định của mô hình mà còn chỉ ra rõ ràng khi nào lời giải thích đó có thể áp dụng cho các trường hợp mới, từ đó giúp người dùng có được cái nhìn toàn diện và tin cậy hơn về hành vi của mô hình.

## 2 Khái niệm cơ bản về ANCHOR

ANCHOR hoạt động bằng cách xác định một tập hợp các quy tắc (*rules*) hoặc điều kiện (*predicates*) dưới dạng **IF-THEN**, đảm bảo rằng khi một mẫu dữ liệu thỏa mãn các điều kiện này, mô hình sẽ duy trì dự đoán ban đầu với xác suất cao.

Hãy cùng nhìn qua kết quả trả về từ thuật toán, bảng dưới đây sử dụng một mẫu dữ liệu từ dataset về thu nhập cá nhân và các điều kiện giúp mô hình duy trì dự đoán mức lương của một người trên 50K với độ tin cậy cao.

Biểu diễn dưới dạng công thức toán học, một anchor  $A$  là một tập hợp các điều kiện trên các đặc trưng của mẫu dữ liệu  $x$  sao cho thỏa mãn  $\text{prec}(A) \geq \tau$ , trong đó:

$$\text{prec}(A) = \mathbb{E}_{\mathcal{D}(z|A)} [\mathbb{1}_{f(x)=f(z)}]$$

Cụ thể:

- $\text{prec}(A)$  là độ chính xác của anchor  $A$ , đo mức độ tin cậy của quy tắc.
- $\tau$  là ngưỡng độ chính xác tối thiểu.

Dữ liệu đầu vào và dự đoán	Giải thích của ANCHOR
$28 < \text{Age} \leq 37$ $\text{Workclass} = \text{Private}$ $\text{Education} = \text{High School grad}$ $\text{Marital Status} = \text{Married}$ $\text{Occupation} = \text{Blue-Collar}$ $\text{Relationship} = \text{Husband}$ $\text{Race} = \text{White}$ $\text{Sex} = \text{Male}$ $\text{Capital Gain} = \text{None}$ $\text{Capital Loss} = \text{Low}$ $\text{Hours per week} \leq 40.00$ $\text{Country} = \text{United-States}$  $P(\text{Salary} > 50K) = 0.57$	<b>IF</b> $28 < \text{Age} \leq 37$ <b>AND</b> $\text{Workclass} = \text{Private}$ <b>AND</b> $\text{Education} = \text{High School grad}$ <b>AND</b> $\text{Marital Status} = \text{Married}$ <b>AND</b> $\text{Occupation} = \text{Blue-Collar}$ <b>AND</b> $\text{Relationship} = \text{Husband}$ <b>AND</b> $\text{Race} = \text{White}$ <b>AND</b> $\text{Sex} = \text{Male}$ <b>AND</b> $\text{Capital Gain} = \text{None}$ <b>AND</b> $\text{Capital Loss} = \text{Low}$ <b>AND</b> $\text{Hours per week} \leq 40.00$ <b>AND</b> $\text{Country} = \text{United-States}$ <b>THEN PREDICT</b> $\text{Salary} > 50K$

- $\mathbb{E}_{\mathcal{D}(z|A)}$  là kỳ vọng trên các mẫu  $z$  được lấy từ phân phối dữ liệu  $\mathcal{D}$ , với điều kiện chúng thỏa mãn anchor  $A$ .
- $\mathbb{1}_{f(x)=f(z)}$  là một hàm chỉ (indicator) nhận giá trị 1 nếu mô hình phân loại hộp đen  $f$  dự đoán cùng nhãn cho cả  $x$  và  $z$ , ngược lại nhận giá trị 0.

Hay nói cách khác, chúng ta kỳ vọng khi áp dụng mô hình  $f$  lên dataset  $\mathcal{D}$ , mọi mẫu dữ liệu  $z$  thỏa mãn các điều kiện trong anchor  $A$  sẽ có cùng dự đoán với mẫu gốc  $x$  với xác suất ít nhất là  $\tau$ , đảm bảo rằng  $A$  thực sự là một quy tắc giải thích đáng tin cậy.

Trong thực tế, ta không thể biết được phân phối chính xác của  $\mathcal{D}$  và mô hình hộp đen  $f$  quá phức tạp để giải thích, nên việc tính toán trực tiếp độ chính xác này là bất khả thi. Thay vào đó, ta chỉ cần các anchor thỏa mãn ràng buộc về độ chính xác với xác suất ít nhất là  $1 - \delta$ , đây còn gọi là độ chính xác ước lượng hay độ chính xác tương đối:

$$P(\text{prec}(A) \geq \tau) \geq 1 - \delta \quad (1)$$

trong đó  $\delta$  cho biết mức sai số chấp nhận được, ta có thể xem  $1 - \delta$  là mức ý nghĩa của phương pháp lấy mẫu. Điều này đảm bảo rằng anchor có “độ chính xác cao” theo nghĩa rằng, với xác suất gần như chắc chắn, nếu thỏa mãn các điều kiện của anchor thì dự đoán của mô hình hầu như không thay đổi.

Nếu nhiều anchor thỏa mãn tiêu chí này, ta sẽ ưu tiên chọn những anchor mô tả hành vi của mô hình trên không gian đầu vào lớn hơn, tức là những anchor có độ phủ cao nhất. Về mặt toán học, độ bao phủ của anchor  $A$  được tính là xác suất mà một mẫu ngẫu nhiên từ không gian các perturbation (các mẫu dữ liệu giả được tạo ra) thỏa mãn điều kiện của anchor  $A$ .

$$\text{cov}(A) = \mathbb{E}_{\mathcal{D}(z)}[A(z)]$$

với  $A(z) = 1$  nếu mẫu  $z$  thỏa mãn anchor  $A$  và 0 khi mẫu  $z$  không thỏa mãn anchor  $A$ . Anchor càng có độ bao phủ cao, lời giải thích càng tổng quát và có khả năng áp dụng cho nhiều mẫu dữ liệu khác nhau, giúp người dùng hiểu được phạm vi mà lời giải thích có hiệu lực.

Do đó, bài toán có thể được phát biểu dưới dạng một bài toán tối ưu như sau:

$$\max_{A \text{ s.t. } P(\text{prec}(A) \geq \tau) \geq 1 - \delta} \text{cov}(A)$$

### 3 Quy trình cơ bản

Để tìm ra anchor phù hợp cho một mẫu dữ liệu cần giải thích, chúng ta thực hiện các bước sau:

- Sinh ra các luật ứng viên:** Khởi tạo tập hợp anchor ban đầu  $A = \emptyset$ . Dựa trên giá trị của các đặc trưng trong dữ liệu, ta xây dựng tập hợp các predicate có thể có, chẳng hạn như “Giá trị của đặc trưng X bằng x”, “Độ tuổi lớn hơn 30”,...

$$A' = A \cup \{a_i\}$$

Các predicate này sau đó được kết hợp dần dần để tạo tập hợp các luật ứng viên, nhằm kiểm tra khi áp dụng các điều kiện này, mô hình có duy trì dự đoán ban đầu hay không.

- Chia thuật toán tìm kiếm thành những beam khác nhau:**

Để tăng tốc độ tìm kiếm và đảm bảo rằng thuật toán không mắc kẹt trong cực trị địa phương, ta chia quá trình tìm kiếm thành các beam (chùm tìm kiếm). Trong mỗi beam, ta có vòng lặp như sau:

- Tạo mẫu dữ liệu lân cận:** Giống như LIME, chúng ta tạo các điểm dữ liệu mới bằng cách thay đổi một số giá trị của mẫu gốc nhưng vẫn đảm bảo rằng chúng thỏa mãn điều kiện của anchor  $A$ . Cách tạo dữ liệu được thực hiện như sau:
  - Dữ liệu văn bản: Thay thế từ ngẫu nhiên bằng từ đồng nghĩa, hay đơn giản hơn là token [MASK].
  - Dữ liệu ảnh: Thay đổi các vùng ảnh nhưng đảm bảo rằng vùng chứa thông tin quan trọng của anchor không bị ảnh hưởng
  - Dữ liệu bảng: Lấy mẫu từ phân phối xác suất của tập dữ liệu gốc để thay đổi các giá trị cột mà không vi phạm điều kiện anchor.
- Dự đoán bằng mô hình gốc:** Sử dụng mô hình phức tạp ban đầu để dự đoán nhãn cho các điểm dữ liệu đã tạo, ta chỉ quan tâm nhãn của chúng có trùng với nhãn của mẫu gốc hay không.
- Tính độ chính xác và độ bao phủ**

Độ chính xác của một anchor  $A$  được tính bằng:

$$\text{prec}(A) = \frac{\text{Số lượng mẫu có cùng dự đoán với mẫu gốc}}{\text{Tổng số mẫu sinh ra từ phân phối điều kiện của anchor}}$$

Độ bao phủ của anchở được tính bằng:

$$\text{cov}(A) = \frac{\text{Số lượng mẫu trong tập kiểm tra thỏa mãn anchor}}{\text{Tổng số mẫu trong tập kiểm tra}}$$

Giả sử ta có một mô hình phân loại email là “Spam” hay “Không Spam”. Ta xét một anchor “Email chứa từ free và win”, và sau khi sinh ra 1000 email biến đổi vẫn giữ hai từ này, trong đó 950 email vẫn bị phân loại là Spam, thì ta có:

$$\text{prec}(A) = \frac{950}{1000} = 0.95,$$

Nếu trong toàn bộ tập kiểm tra có 10,000 email, và 1,500 email chứa “free” và “win”, thì:

$$\text{cov}(A) = \frac{1500}{10000} = 0.15.$$

- (d) **Thuật toán KL-LUCB:** Sau khi tính toán độ chính xác và độ bao phủ của các anchor ứng viên, thuật toán KL-LUCB sẽ được sử dụng để tìm ra anchor tốt nhất. Quá trình tìm kiếm có thể rơi vào một trong ba trường hợp sau:
1. **Anchor đạt yêu cầu về độ chính xác:** Với mỗi beam, nếu có ít nhất một anchor có độ chính xác đạt hoặc vượt mức ngưỡng cho trước ( $\tau$ ), thuật toán sẽ chọn anchor có độ phủ lớn nhất. Sau khi tất cả beam hoàn thành, thuật toán sẽ chọn anchor có độ bao phủ cao nhất trong các beam làm kết quả cuối cùng.
  2. **Không có anchor nào đạt yêu cầu về độ chính xác:** Nếu toàn bộ các anchor hiện tại đều không đủ chính xác (tức là không đạt ngưỡng  $\tau$ ), thuật toán quay lại bước 1 và mở rộng các anchor bằng cách thêm điều kiện mới. Do thêm điều kiện có thể làm giảm độ bao phủ, thuật toán sử dụng cách tiếp cận bottom-up (tạo anchor từ đơn giản đến phức tạp) để tránh làm anchor quá cụ thể và mất tính tổng quát.
  3. **Số lượng mẫu sinh ra đạt giới hạn:** Để tránh việc thuật toán kiểm tra quá nhiều mẫu và làm chậm quá trình tìm kiếm, ta đặt một giới hạn về số lượng mẫu tối đa. Nếu KL-LUCB vượt quá giới hạn này mà vẫn chưa tìm được anchor thỏa mãn, thuật toán sẽ chọn anchor tốt nhất hiện có và ghi nhận xác suất  $P(\text{prec}(A) \geq \tau)$ . Nếu không có beam nào tìm thấy anchor phù hợp, thuật toán sẽ quay lại bước 1 và mở rộng tập hợp anchor như trong trường hợp 2.

## 4 Từng bước triển khai ANCHOR cho dữ liệu ảnh

Tương tự như LIME, chúng ta sẽ sử dụng ResNet50 đã được huấn luyện sẵn để dự đoán lớp của một bức ảnh éch, sau đó áp dụng Anchor để giải thích kết quả này.

### 4.1 Sinh ra các luật ứng viên:

Trước tiên, ta cần chia ảnh thành các superpixel hay segment. Đối với dữ liệu ảnh, một anchor được xem là tập hợp các superpixel giữ lại trong ảnh. Sau khi khởi tạo  $A = \emptyset$ , ta tạo ra anchor đầu tiên với 1 điều kiện ứng với 1 superpixel được “bật” như bảng 2.

	Superpixel 0	Superpixel 1	Superpixel 2	Superpixel 3
$A_1$	1	0	0	0
$A_2$	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>
$A_3$	0	0	1	0
$A_4$	0	0	0	1

Bảng 2: Các anchor được sinh ra ở vòng lặp đầu tiên.

### 4.2 Chia thuật toán tìm kiếm thành những beam khác nhau:

Bước tiếp theo, từng beam nhận toàn bộ các anchor được sinh ra ở bảng 2. Để đơn giản, ta sẽ chọn anchor thứ 2 làm minh họa. Số lần chạy beam search sẽ là len(features) = 4 vì mỗi thuộc tính chỉ có thể được thêm 1 lần vào anchor.

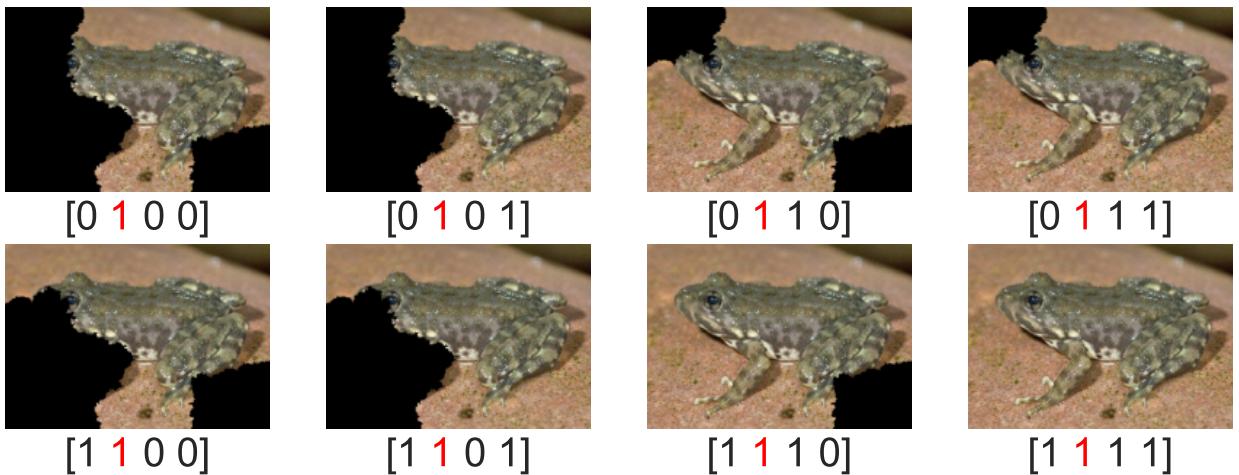
### 4.3 Tạo mẫu dữ liệu lân cận và dự đoán bằng mô hình gốc:

LIME tạo ra các mẫu dữ liệu mới bằng cách ngẫu nhiên chọn các phần của dữ liệu gốc để giữ lại hoặc loại bỏ. Trong khi đó, ANCHOR cố định một số superpixel quan trọng và thay thế các vùng còn lại bằng dữ liệu nhiều từ tập dữ liệu gốc. Cách tiếp cận này giúp đảm bảo rằng các mẫu sinh ra vẫn giữ được ý nghĩa ban đầu mà không bị biến đổi quá mức. Ở ví dụ này, để đơn giản, ta thay đổi các thành

phần cần loại bỏ thành nền đen. Bảng 3 và hình 18 minh họa một vài mẫu được sinh ra khi ta lấy anchor  $A_2 = [0, 1, 0, 0]$  làm ví dụ.

Superpixel 0	<b>Superpixel 1</b>	Superpixel 2	Superpixel 3	$f(x) = f(z)$
0	<b>1</b>	0	1	True
0	<b>1</b>	1	0	True
0	<b>1</b>	0	0	True
1	<b>1</b>	0	0	True

Bảng 3: Một vài mẫu sinh ra khi cố định Superpixel 1. Cột  $f(x) = f(z)$  so sánh dự đoán của mô hình với ảnh biến đổi có giống dự đoán của ảnh gốc không.



Hình 18: Một vài mẫu được sinh ra bởi anchor  $A_i = [0, 1, 0, 0]$ .

#### 4.4 Đánh giá độ chính xác của luật và tính toán độ bao phủ:

Anchor sẽ được tính precision và coverage để có thể áp dụng vào thuật toán tìm kiếm KL-LUCB.

Lấy ví dụ với anchor  $A_2 = [0, 1, 0, 0]$ . Giả sử ta tạo ra 10 mẫu dữ liệu mới thỏa mãn anchor này, trong đó 6 mẫu được mô hình dự đoán giống như ảnh gốc. Khi đó, ta tính độ chính xác của anchor như sau:

$$\text{prec}(A) = \frac{\text{Positive sample}}{\text{Total samples}} = \frac{6}{10} = 0.6$$

$$\text{cov}(A) = 1 - \left( \frac{\text{Number of elements}}{\text{Total elements}} \right) = 1 - \frac{1}{4} = 0.75$$

Có một điều cần lưu ý là định nghĩa của hàm cov có thể thay đổi tùy thuộc vào cách định nghĩa của ta với kiểu dữ liệu, không nhất thiết phải giống công thức trên. Ví dụ:

$$\text{cov}(A) = 1 - \left( \frac{\text{Number of pixels}}{\text{Total pixels}} \right)$$

hoặc

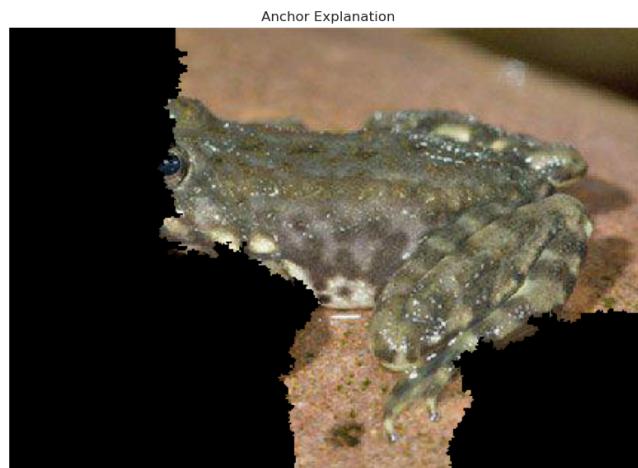
$$\text{cov}(A) = \frac{\text{Number samples contains } A \text{ of } A_\emptyset}{\text{Number samples of } A_\emptyset}.$$

#### 4.5 Thuật toán KL-LUCB:

Để đảm bảo mỗi anchor đạt độ chính xác mong muốn, thuật toán KL-LUCB sẽ tăng dần số lượng mẫu kiểm tra và sử dụng khoảng tin cậy để quyết định có tiếp tục lấy thêm mẫu hay không. Quá trình thực hiện như sau:

- Bắt đầu với một số lượng mẫu nhỏ cho mỗi anchor ứng viên.
- Ước lượng độ chính xác của từng anchor dựa trên các mẫu đã thu thập.
- Xác định khoảng tin cậy cho độ chính xác của từng anchor.
  - Nếu lower bound của độ chính xác của một anchor vượt qua ngưỡng  $\tau$ , anchor đó được chọn.
  - Nếu upper bound của một anchor không thể đạt được ngưỡng  $\tau$ , anchor đó bị loại.
- Nếu chưa tìm được anchor phù hợp, tiếp tục lấy thêm mẫu cho đến khi có một anchor thỏa mãn hoặc đạt giới hạn số mẫu tối đa.

Thuật toán kết thúc khi chọn được anchor có độ bao phủ cao nhất nhưng vẫn đảm bảo độ chính xác yêu cầu. Trong trường hợp này, anchor tối ưu được chọn là  $A_i = [0, 1, 0, 0]$ , được minh họa như hình 19.



Hình 19: Giải thích được đưa ra bởi ANCHOR.

### 5 Code

Trong phần này, chúng ta sẽ khám phá cách triển khai thuật toán ANCHOR trong lập trình. Giống như LIME, thuật toán này cũng có một thư viện riêng mang tên `anchor`, giúp giải thích mô hình máy học trên dữ liệu dạng bảng, văn bản và hình ảnh. Các bạn có thể tìm hiểu thêm thông tin về thư viện ở đây.

#### 5.1 ANCHOR cho dữ liệu dạng bảng

Trước tiên ta cần import các thư viện cần thiết và đồng thời load dataset cần dùng. Ở ví dụ này chúng ta dùng tập dataset Adult để dự đoán lương của một người có trên \$50,000 một năm hay không.

```

1 import numpy as np
2 import sys
3 import sklearn
4 import sklearn.ensemble

```

```

5 import sklearn.metrics
6 from anchor import utils
7 from anchor import anchor_tabular
8 np.random.seed(1)
9
10 dataset_folder = './datasets/'
11 dataset = utils.load_dataset(
12     'adult',
13     balance=True,
14     dataset_folder=dataset_folder,
15     discretize=True
16 )

```

Trong đoạn code trên:

- Các thư viện như `numpy`, `sklearn` và các module từ `anchor` được import để xử lý dữ liệu và thực hiện giải thích.
- Hàm `utils.load_dataset` tải tập dữ liệu Adult với các tham số:
  - `balance=True`: Cân bằng số lượng mẫu giữa các lớp.
  - `discretize=True`: Rời rạc hoá các thuộc tính liên tục, giúp tạo mẫu lân cận dễ dàng hơn. Việc rời rạc này được thực hiện bằng cách chia các thuộc tính định lượng đó vào các “bin”.

Tiếp đến, chúng ta sẽ khởi tạo và huấn luyện mô hình `RandomForestClassifier` với dataset trên. Kết quả độ chính xác ở tập train và test lần lượt là: 0.94 và 0.85. Nhưng liệu mô hình có thực sự dự đoán đúng với logic bài toán hay không? Chúng ta sẽ triển khai giải thích cho một ví dụ cụ thể để tìm hiểu

```

1 clf = sklearn.ensemble.RandomForestClassifier(n_estimators=50, n_jobs=5)
2 clf.fit(dataset.train, dataset.labels_train)
3
4 train_acc = sklearn.metrics.accuracy_score(dataset.labels_train, clf.predict(dataset.train))
5 test_acc = sklearn.metrics.accuracy_score(dataset.labels_test, clf.predict(dataset.test))
6
7 explainer = anchor_tabular.AnchorTabularExplainer(
8     dataset.class_names,
9     dataset.feature_names,
10    dataset.train,
11    dataset.categorical_names
12 )
13
14 idx = 0
15 prediction = clf.predict(dataset.test[idx].reshape(1, -1))[0]

```

Giải thích đoạn code trên:

- Khởi tạo và huấn luyện mô hình `RandomForestClassifier` trên tập dữ liệu huấn luyện.
- Độ chính xác của mô hình được tính toán trên tập huấn luyện và tập test bằng hàm `accuracy_score`.
- Đối tượng `AnchorTabularExplainer` được khởi tạo với:
  - `dataset.class_names`: Danh sách tên các lớp (ví dụ: "`<=50K`" và "`>50K`").
  - `dataset.feature_names`: Danh sách tên các thuộc tính của dữ liệu.
  - `dataset.train`: Dữ liệu huấn luyện, dùng để tạo các mẫu lân cận cho việc giải thích.
  - `dataset.categorical_names`: Các thuộc tính rời rạc trong dataset.

- Cuối cùng, mẫu dữ liệu thứ 0 trong tập test được chọn để tiến hành giải thích dự đoán.

Mô hình dự đoán lương của người này thuộc nhóm có thu nhập trên \$50,000 một năm. Dựa vào dự đoán này, ANCHOR sẽ xác định các điều kiện (anchor) quan trọng giúp mô hình đưa ra quyết định. Đối với dữ liệu dạng bảng, các mẫu lân cận được tạo ra bằng cách cố định một số cột và thay thế các cột khác bằng các giá trị được lấy từ tập huấn luyện, từ đó đảm bảo rằng các mẫu tạo ra có ý nghĩa thực tế.

```

1 explanation = explainer.explain_instance(dataset.test[idx], clf.predict, threshold=0.95, tau=0.15)
2
3 print('Anchor: %s' % (' AND '.join(explanation.names())))
4 print('Precision: {:.2f}'.format(explanation.precision()))
5 print('Coverage: {:.2f}'.format(explanation.coverage()))
6 >>> Anchor: Education = Bachelors AND Relationship = Husband AND Occupation = Sales
7 >>> Precision: 0.97
8 >>> Coverage: 0.02

```

Giải thích đoạn code trên:

- Hàm `explain_instance` tạo lời giải thích cho mẫu dữ liệu được chọn, sử dụng hàm `clf.predict` để dự đoán nhãn cho các mẫu lân cận.
- `threshold=0.95` là tham số để thuật toán KL-LUCB xác định khi nào việc lấy mẫu hoàn tất, để có thể xác định động chính xác ước lượng mà ta cần.
- `tau=0.15` đặt ngưỡng độ tin cậy, nghĩa là anchor phải tạo ra dự đoán giống với mẫu gốc trong ít nhất 95% các mẫu lân cận.
- Hàm `explanation.names()` trả về danh sách các điều kiện của anchor dưới dạng chuỗi, các điều kiện này được nối với nhau bởi chuỗi "AND".
- `explanation.precision()` trả về độ chính xác của anchor, cho biết tỉ lệ các mẫu lân cận thoả mãn điều kiện mà mô hình dự đoán giống với mẫu gốc.
- `explanation.coverage()` cho biết tỷ lệ mẫu trong tập huấn luyện mà anchor có thể áp dụng được.

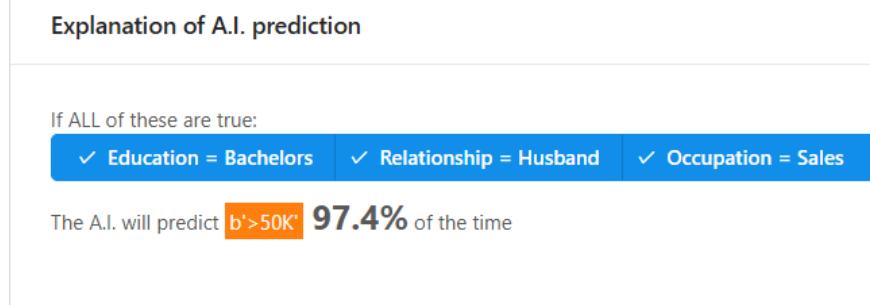
Trong ví dụ, anchor được tìm ra là: `Education = Bachelors AND Relationship = Husband AND Occupation = Sales` với độ chính xác khoảng 97% và độ bao phủ 2%. Mặc dù độ bao phủ của anchor chỉ là 2%, điều này cho thấy rằng các điều kiện này chỉ áp dụng cho một tập con nhỏ của dữ liệu, nhưng lại rất đặc thù và có độ chính xác rất cao.

Cuối cùng, để trực quan hóa lời giải thích, ta sử dụng hàm `show_in_notebook()` để hiển thị giao diện tương tác.

```

1 explanation.show_in_notebook()

```



Hình 20: Dự đoán của thư viện `anchor` cho dữ liệu dạng bảng.

## 5.2 ANCHOR cho dữ liệu văn bản

Ở ví dụ này, chúng ta áp dụng ANCHOR để giải thích quyết định của mô hình LogisticRegression trong bài toán phân loại cảm xúc được huấn luyện trên tập dữ liệu đánh giá phim rt-polaritydata, bao gồm các tệp rt-polarity.neg và rt-polarity.pos chứa các câu đánh giá tiêu cực và tích cực. Các đoạn mã dưới đây thực hiện các bước chuẩn bị dữ liệu, huấn luyện mô hình và tạo lời giải thích cho một câu văn mẫu.

```

1 import os
2 import numpy as np
3 import sklearn
4 import sklearn.model_selection
5 import sklearn.linear_model
6 import spacy
7 from sklearn.feature_extraction.text import CountVectorizer
8 from anchor import anchor_text
9
10 def load_polarity(path='./datasets/sentiment-sentences/rt-polaritydata'):
11     data, labels = [], []
12     files = ['rt-polarity.neg', 'rt-polarity.pos']
13     for label, file_name in enumerate(files):
14         file_path = os.path.join(path, file_name)
15         with open(file_path, 'rb') as f:
16             for line in f:
17                 line_str = line.decode('utf8').strip()
18                 if line_str:
19                     data.append(line_str)
20                     labels.append(label)
21     return data, labels
22
23 nlp = spacy.load('en_core_web_sm')
24
25 data, labels = load_polarity()
26 train, test, train_labels, test_labels = sklearn.model_selection.train_test_split(
27     data, labels, test_size=0.2, random_state=42)
28 train, val, train_labels, val_labels = sklearn.model_selection.train_test_split(
29     train, train_labels, test_size=0.1, random_state=42)
30 train_labels = np.array(train_labels)
31 val_labels = np.array(val_labels)
32
33 vectorizer = CountVectorizer(min_df=1)
34 vectorizer.fit(train)
35 train_vectors = vectorizer.transform(train)
36 val_vectors = vectorizer.transform(val)
37
38 clf = sklearn.linear_model.LogisticRegression()
39 clf.fit(train_vectors, train_labels)
40 preds = clf.predict(val_vectors)
41 print('Validation accuracy: {:.2f}'.format(
42     sklearn.metrics.accuracy_score(val_labels, preds)))
43 >> Validation accuracy: 0.75

```

Trong đoạn mã trên:

- Hàm `load_polarity` đọc dữ liệu từ các tệp `rt-polarity.neg` và `rt-polarity.pos`, gán nhãn 0 cho tiêu cực và 1 cho tích cực.
- `spacy` được sử dụng để tải mô hình ngôn ngữ tiếng Anh `en_core_web_sm` nhằm hỗ trợ quá trình tokenization và xử lý văn bản.

- Dữ liệu được chia thành các tập huấn luyện, kiểm tra và validation bằng `train_test_split` của `sklearn`.
- `CountVectorizer` chuyển đổi văn bản thành dạng vector đặc trưng dựa trên tần suất xuất hiện của các từ.
- Mô hình `LogisticRegression` được huấn luyện trên tập huấn luyện và đánh giá trên tập validation với độ chính xác đạt khoảng 0.75.

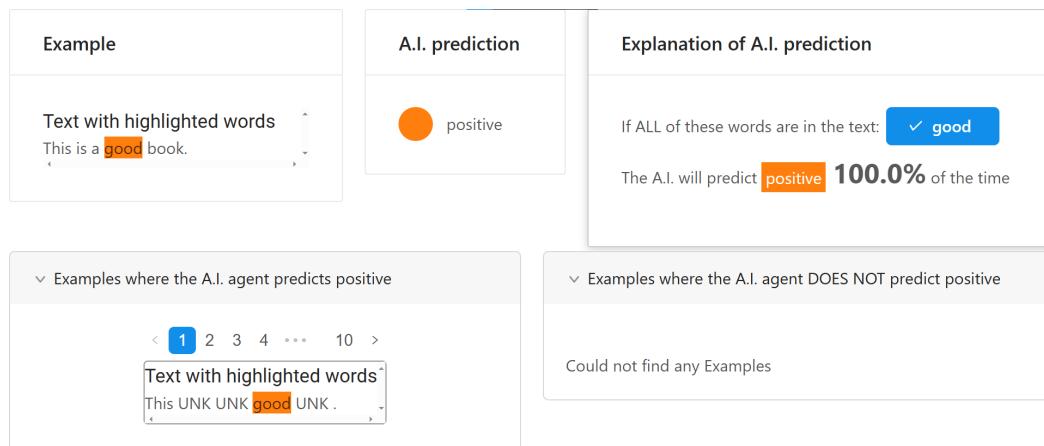
```

1 def predict_lr(texts):
2     return clf.predict(vectorizer.transform(texts))
3
4 explainer = anchor_text.AnchorText(nlp, ['negative', 'positive'],
5                                     use_unk_distribution=True)
6
7 np.random.seed(1)
8 text = 'This is a good book.'
9 pred_class = predict_lr([text])[0]
10 pred = explainer.class_names[pred_class]
11 print('Prediction:', pred)
12 >>> Prediction: positive
13
14 exp = explainer.explain_instance(text, predict_lr, threshold=0.95, tau=0.15)
15 exp.show_in_notebook()

```

Sau đó, chúng ta tiến hành giải thích cho mô hình theo các bước:

- Định nghĩa hàm `predict_lr` nhận đầu vào là một danh sách các câu văn và trả về nhãn dự đoán từ mô hình `LogisticRegression` sau khi chuyển đổi văn bản sang vector đặc trưng.
- Khởi tạo đối tượng `AnchorText` với các tham số:
  - `nlp`: Mô hình ngôn ngữ của `spacy` để thực hiện tokenization và xử lý cấu trúc câu.
  - `['negative', 'positive']`: Danh sách các nhãn cảm xúc.
  - `use_unk_distribution=True`: Cho phép sử dụng phân phối của token không xác định (unknown) khi một từ bị che bằng token [MASK].
- Sử dụng hàm `explain_instance` để tạo lời giải thích cho câu văn mẫu:
  - `text`: Câu văn cần được giải thích.
  - `predict_lr`: Hàm dự đoán được định nghĩa ở trên.
  - `threshold=0.95`: Đây là giá trị  $1 - \delta$ , nghĩa là các từ (anchor) được chọn phải duy trì độ chính xác của mô hình ít nhất 95% (sai số tối đa có thể chấp nhận được là  $\delta = 5\%$ ).
  - `tau=0.15`: Tham số dùng để xác định độ chính xác tuyệt đối.
- Cuối cùng, lời giải thích được hiển thị thông qua hàm `show_in_notebook()`, cho phép trực quan các token quan trọng mà mô hình dựa vào để đưa ra dự đoán.



Hình 21: Lời giải thích của ANCHOR đối với dữ liệu văn bản. Ta thấy khi có sự xuất hiện của từ “good” trong văn bản, mô hình chắc chắn sẽ đánh giá đây là một câu có cảm xúc tích cực.

Khi phân tích câu "This is a good book.", hệ thống xác định từ **good** là yếu tố quan trọng nhất quyết định nhãn **positive**.

- Màu cam: Đánh dấu từ có ảnh hưởng mạnh đến dự đoán.
- Mô hình Logistic Regression: Dự đoán **positive** với xác suất **100%** khi từ **good** xuất hiện.
- Các ví dụ khác: Khi từ **good** xuất hiện trong câu nhưng chứa thêm các token không xác định (**UNK**), mô hình vẫn dự đoán kết quả tích cực, chứng tỏ **good** là từ khóa then chốt.
- Không có ví dụ nào: Hệ thống không tìm thấy trường hợp nào mà mô hình không dự đoán **positive** khi từ **good** có mặt.

Qua đó, ANCHOR giúp chúng ta xác định các từ hoặc cụm từ quan trọng trong câu có ảnh hưởng lớn đến dự đoán của mô hình, từ đó cung cấp cái nhìn sâu sắc về cơ chế hoạt động của mô hình phân loại cảm xúc.

### 5.3 ANCHOR cho dữ liệu hình ảnh

Cuối cùng, chúng ta dùng ANCHOR để giải thích quyết định của mô hình Inception v3 đã được huấn luyện sẵn để dự đoán cho ảnh 22. Các đoạn mã dưới đây thực hiện các bước tiền xử lý ảnh, dự đoán nhãn và trực quan hóa các vùng ảnh quan trọng được mô hình dựa vào đó để đưa ra dự đoán.



Hình 22: Ảnh được dùng để lời giải thích trong ví dụ này.

```

1 import os
2 import time
3 import numpy as np
4 import torch
5 import urllib.request
6 import matplotlib.pyplot as plt
7 import skimage.io
8 import skimage.transform
9 from skimage import color
10 import anchor.anchor_image
11
12 model = torch.hub.load('pytorch/vision:v0.10.0', 'inception_v3', pretrained=True)
13 model.eval()
14 if torch.cuda.is_available():
15     model.to('cuda')

```

Đầu tiên, ta thực hiện:

- Import các thư viện cần thiết như numpy, torch, skimage và matplotlib để xử lý ảnh và xây dựng mô hình.
- Mô hình Inception v3 được tải từ PyTorch Hub với các trọng số đã được huấn luyện sẵn, sau đó được đặt ở chế độ đánh giá bằng eval().
- Nếu có GPU, mô hình sẽ được chuyển sang thiết bị cuda để tăng tốc tính toán.

```

1 def transform_img_fast(path):
2     img = skimage.io.imread(path)
3     if len(img.shape) != 3:
4         img = skimage.color.gray2rgb(img)
5     if img.shape[2] == 4:
6         img = color.rgb2rgb(img)
7     short_edge = min(img.shape[:2])
8     yy = (img.shape[0] - short_edge) // 2
9     xx = (img.shape[1] - short_edge) // 2
10    crop_img = img[yy:yy + short_edge, xx:xx + short_edge]
11    return (skimage.transform.resize(crop_img, (299, 299)) - 0.5) * 2
12
13 img_filename = "dog.jpg"
14 if not os.path.exists(img_filename):
15     urllib.request.urlretrieve(
16         "https://github.com/pytorch/hub/raw/master/images/dog.jpg",
17         img_filename
18     )
19
20 def predict(images):
21     images = images.transpose((0, 3, 1, 2))
22     input_tensor = torch.FloatTensor(images)
23     if torch.cuda.is_available():
24         input_tensor = input_tensor.to('cuda')
25     with torch.no_grad():
26         output = model(input_tensor)
27     probabilities = torch.nn.functional.softmax(output, dim=1)
28     return probabilities.cpu().numpy()
29
30 def show_image(image, title=""):
31     plt.figure()
32     plt.imshow(image / 2 + 0.5)
33     plt.axis("off")

```

```

34     plt.title(title)
35     plt.show()
36
37 images = transform_img_fast(img_filename)
38 show_image(images, "Input Image")

```

Tiếp đến, ta định nghĩa các hàm cần thiết:

- Hàm `transform_img_fast` nhận đầu vào là đường dẫn đến ảnh, thực hiện biến đổi ảnh để phù hợp với đầu vào của Inception v3.
- Hàm `predict` thực hiện xử lý và đưa ảnh cho mô hình dự đoán xác suất thuộc mỗi lớp.
- Hàm `show_image` trực quan hóa ảnh đầu vào.

```

1 explainer = anchor.anchor_image.AnchorImage(
2     dummys=torch.tensor(torch.ones(10, 3, 299, 299)),
3     white=True,
4     transform_img_fn=transform_img_fast,
5     n=5000
6 )
7
8 segment, exp = explainer.explain_instance(
9     images,
10    predict,
11    threshold=0.50,
12    batch_size=50,
13    tau=0.10,
14    verbose=True,
15    min_shared_samples=100,
16    beam_size=2
17 )

```

Ở bước này, ta tiến hành giải thích:

- Khởi tạo đối tượng của lớp `AnchorImage` với các tham số:
  - `dummys`: Một tensor chứa 10 ảnh giả dùng để khởi tạo các ảnh nền để thay thế vào các mẫu từ ANCHOR.
  - `white=True`: Thiết lập nền trắng cho các vùng ảnh bị che khi tạo anchor.
  - `transform_img_fn`: Hàm tiền xử lý ảnh đã định nghĩa ở trên.
  - `n=5000`: Số lượng mẫu ảnh được tạo ra để tìm kiếm các anchor.
- Sử dụng hàm `explain_instance` thực hiện giải thích cho ảnh đầu vào:
  - `images`: Ảnh cần được giải thích.
  - `predict`: Hàm dự đoán đã định nghĩa, được sử dụng để đưa ra dự đoán của mô hình.
  - `threshold=0.50`: Ngưỡng xác suất dùng để xác định ảnh hưởng của một anchor, hay còn gọi là độ chính xác ước lượng.
  - `batch_size=50`: Số lượng mẫu được xử lý trong một batch.
  - `tau=0.10`: Tham số điều chỉnh độ chính xác của anchor cần có.
  - `verbose=True`: In ra các thông tin chi tiết trong quá trình giải thích.
  - `min_shared_samples=100` và `beam_size=2`: Các tham số điều chỉnh quá trình tìm kiếm anchor tối ưu của thuật toán beam search KL-LUCB.

- Hàm trả về hai đối số:
  - **segment**: Mảng đánh dấu các phân đoạn (segment) của ảnh.
  - **exp**: Danh sách các anchor cùng các thông số liên quan.

```

1 def show_anchor(segment, exp, image):
2     mask = np.zeros(segment.shape, dtype=bool)
3
4     for seg_idx, _, _, _, _ in exp:
5         mask |= (segment == seg_idx)
6     anchor_image = image.copy()
7     anchor_image[~mask] = 0
8     show_image(anchor_image, "Anchor Explanation")
9
10 show_anchor(segment, exp, images)

```

Cuối cùng ta thực hiện trực quan kết quả giải thích bằng hàm `show_anchor`:

1. Tạo một `mask` ban đầu với giá trị `False` cho toàn bộ ảnh.
2. Duyệt qua các anchor được trả về trong `exp` để cập nhật `mask`, đánh dấu các phân đoạn ảnh có ảnh hưởng tích cực đến dự đoán.
3. Ảnh gốc được sao chép, sau đó các vùng không thuộc anchor được che bằng cách đặt giá trị về 0.
4. Ảnh kết quả được hiển thị để trực quan hóa các vùng ảnh quan trọng.



Hình 23: Kết quả của ANCHOR cho bức hình.

- Vùng màu sáng hiển thị các pixel có ảnh hưởng lớn đến quyết định của mô hình. Các phần có màu xám minh họa rằng chúng không đóng vai trò quan trọng trong quá trình nhận diện.
- Trong ảnh này, mô hình chủ yếu dựa vào khu vực đầu và tai của con vật để đưa ra dự đoán.

## 6 Hạn chế của ANCHOR

Tuy ANCHOR là một phương pháp cải tiến của LIME, nhưng cũng không thể thay thế hoàn toàn LIME và cũng chịu một vài nhược điểm như sau:

- **Khó để cài đặt:** ANCHOR có nhiều tham số ảnh hưởng lớn đến kết quả giải thích. Ngoài ra, với từng loại dữ liệu, chúng ta cần thiết kế hàm lấy mẫu phù hợp, làm cho quá trình cài đặt trở nên phức tạp.
- **Đánh đổi giữa việc phân rã, gom nhóm và độ trung thực của lời giải thích:** Nhiều tình huống đòi hỏi phải phân rã (discretization) hoặc gom nhóm (aggregation) các điều kiện vì nếu không, kết quả sẽ quá cụ thể, có độ bao phủ thấp và không góp phần vào việc hiểu mô hình. Có những trường hợp yêu cầu quá nhiều tiêu chuẩn cụ thể, ví dụ như lời giải thích ANCHOR đưa ra trong dataset người sống sót trên Titanic: “Nếu giới tính là nữ, hạng vé là thương gia, độ tuổi là 20, không đi chung với gia đình thì dự đoán sẽ là sống sót”. Câu nói trên chỉ có độ bao phủ là 2,4%, quá thấp để ta hiểu về cách mô hình hoạt động, vậy nên ta phải áp dụng các phương pháp kể trên để giúp cho lời giải thích có độ bao phủ cao hơn. Phân rã các điều kiện có thể cải thiện lời giải thích, nhưng nếu làm không đúng cách, các đặc trưng quan trọng có thể bị gộp chung hoặc tách nhỏ quá mức, khiến mô hình không thể hiện rõ ràng cách nó đưa ra quyết định. Vì không có kỹ thuật phân rã tối ưu, người dùng cần phải nắm rõ dữ liệu trước khi quyết định cách phân rã để không thu được kết quả kém.
- **Yêu cầu nhiều lần gọi mô hình:** ANCHOR cần chạy mô hình nhiều lần để đưa ra lời giải thích, giống như các phương pháp dựa trên lấy mẫu khác. Dù thuật toán có sử dụng MAB giảm số lần gọi, nhưng thời gian chạy vẫn phụ thuộc rất nhiều vào tốc độ của mô hình, đôi khi rất chậm và không ổn định.
- **Thiếu sự tường minh trong định nghĩa:** Khái niệm độ bao phủ (coverage) không được định nghĩa rõ ràng trong một số loại dữ liệu. Ví dụ, không có định nghĩa rõ ràng hay phổ quát nào về cách so sánh superpixel giữa các bức ảnh khác nhau, dẫn đến việc giải thích đôi khi không nhất quán hoặc khó hiểu.

- *Hết* -