Script báo cáo máy học (T4 – T123)

Chương 8:

Introduce

Trong thời đại của Internet và mạng xã hội, ý kiến của mọi người, review, và những đề xuất đã trở thành nguồn tài nguyên giá trị cho PS(Political Science) và các doanh nghiệp. Nhờ vào các công nghệ hiện đại, chúng ta có thể thu thập và phân tích dữ liệu một cách hiệu quả nhất. Trong bài báo cáo này thì chúng ta sẽ đào sâu trường Natural Language Processing (NPL – Ngôn ngữ xử lý tự nhiên) hay còn được gọi là Sentiment analysis (Phân tích quan điểm) và tìm hiểu xem làm thế nào để sử dụng giải thuật của Machine Learning để phân loại tài liệu dựa trên tính phân cực: thái độ của người viết. Bài báo cáo này sẽ bao gồm các nội dung dưới đây:

* Cleaning and peparing text data. (Clean và chuẩn bị data)
* Building feature vectors from text documents. (Xây dựng features vectors từ file text)
* Training a machine learning model to classify positive and negative movie reviews. (Huấn luyện một model để phân biệt được các bình luận tích cực và tiêu cực về phim)
* Working with large text datasets using *out-of-core* learning. (Làm việc với dữ liệu lớn sửa dụng phương pháp *out-of-core)*

Obtaining the IMDb movie review dataset

Sentiment analysis, đôi khi còn có thể gọi là opinion mining, nó là một nhánh phụ khá phổ biến của trường NPL; nó phân tích tính phân cực của một tài liệu. Một công việc rất phổ biến ở sentiment analysis là sự phân loại của tài liệu dựa trên các lựa chọn theo “bày tỏ” hoặc cảm xúc của tác giả liên quan đến một vấn đề cụ thể.

Trong bài báo cáo này, chúng ta sẽ làm việc với một lượng data(dữ liệu) khổng lồ về review phim từ Internet Movie Database (IMDb). Dữ liệu này bao gồm 50,000 phim phổ biến được gắn nhãn tích cực (positive) hoặc (negative). Ở đây tích cực nghĩa là bộ phim được đánh giá hơn 6\*(\*:sao) trên IMDb và tiêu cực thì được đánh giá thấp hơn 5\*. Ở phần tiếp theo chúng ta sẽ tìm hiểu ý nghĩa của những đánh giá từ tập hợp phim này và xây dựng một model để dự đoán một người reviewer sẽ đánh giá thích(like) hoặc dislike(không thích) một bộ phim.

Link download: [Sentiment Analysis (stanford.edu)](http://ai.stanford.edu/~amaas/data/sentiment/)

Sau khi tải về và giải nén dataset, chúng ta sẽ tích hợp lại các thư mục text từ file mới download về để được một file CSV. Để biết được quá trình chạy và thời gian chạy còn lại chúng ta sẽ cài PyPrind với câu lệnh : pip install pyprind

>>> import pyprind  
>>> import pandas as pd  
>>> import os  
>>> pbar = pyprind.ProgBar(50000)  
>>> labels = {'pos':1, 'neg':0}  
>>> df = pd.DataFrame()  
>>> for s in ('test', 'train'):  
... for l in ('pos', 'neg'):  
... path ='./aclImdb/%s/%s' % (s, l)  
... for file in os.listdir(path):

... with open(os.path.join(path, file), 'r') as infile:...  
txt = infile.read()  
... df = df.append([[txt, labels[l]]], ignore\_index=True)  
... pbar.update()  
>>> df.columns = ['review', 'sentiment']  
0% 100%  
[##############################] | ETA[sec]: 0.000  
Total time elapsed: 725.001 sec

Thực thi code, chúng ta sẽ khơi tạo một thanh progress mới với tên gọi là pbar, sử dụng loop lồng vào nhau chúng ta sẽ lặp đi lặp lại các thư mục train và test trong thư mục chính aclImdb và đọc các text file từ thư mục con pos và neg đã được thêm vào DataFrame df – với (1 = positive và 0 = negative)

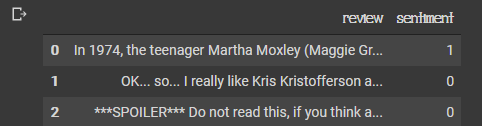
Các dataset ban đầu đã được sắp xếp, bây giờ chúng ta sẽ xáo trộn DataFrame để sử dụng hàm permutation từ câu lệnh np.random – công dụng là sẽ tách dataset thành 2 tập là training và test để ở phần tiếp theo chúng ta sẽ lấy data từ thư mục local. Chúng ta sẽ xáo trộn và hợp lại thành 1 file dataset dạng file CSV:

>>> import numpy as np  
>>> np.random.seed(0)  
>>> df = df.reindex(np.random.permutation(df.index))  
>>> df.to\_csv('./movie\_data.csv', index=False)

Chúng ta sẽ sử dụng datatset ở chương sau. Nhanh chóng kiểm tra lại chúng ta đã save data đúng ở dưới dạng CSV chưa và in 3 mẫu đầu tiên:

>>> df = pd.read\_csv('./movie\_data.csv')  
>>> df.head(3)

Ở đây nếu bạn chạy trên Google Collab, nó sẽ trông như thế này:



Introducing the bag-of-words model

Chúng ta phải convert phân loại dữ liệu như là text hoặc words sang dạng số trước khi chúng ta đưa vào giải thuật của machine learning. Trong phần này, nhóm mình sẽ giới thiệu bag-of-words model, nó cho phép chúng ta diễn tả text như là các vector đặc trưng số (numerical feature vectors). Nói nôm na là bag-of-words model khá đơn giản và có thể tóm tắt như sau:

1. Chúng ta có thể tạo từ vựng (vocabulary) của tokens riêng – ví dụ, words – cho toàn bộ tài liệu.
2. Chúng ta sẽ xây dựng một vector features từ mỗi tài liệu cụ thể mà nó chứa các từ ngữ xuất hiện với tuần suất thường xuyên.

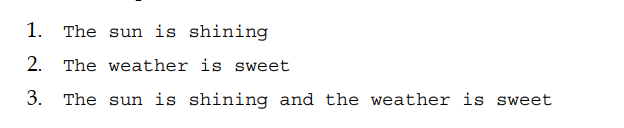
Unique words ở mỗi tài liệu chỉ biểu diễn một lượng nhỏ từ tập hợp của tất cả các từ trong bag-of-words, feature vector sẽ chứa hầu hết sẽ chứa giá trị 0 (zero), chúng ta sẽ gọi nó là thưa thớt (sparse).

Transforming words into feature vectors

Để tạo một bag-of-words model dựa trên đếm từ tương ứng trong data, chúng ta có thể sử dụng class CountVectorizer có trong scikit-learn. Chúng ta sẽ thấy class này sẽ lấy một array của text data, có thể là cả một thư mục hoặc 1 đoạn thôi và khởi tạo bag-of-word model cho chúng ta.

>>> import numpy as np  
>>> from sklearn.feature\_extraction.text import CountVectorizer  
>>> count = CountVectorizer()  
>>> docs = np.array([  
... 'The sun is shining',  
... 'The weather is sweet',  
... 'The sun is shining and the weather is sweet'])  
>>> bag = count.fit\_transform(docs)

Gọi fit\_transform của CountVectorizer, chúng ta sẽ tạo được từ (vocabulary) của bag-of-words model và chuyển hoá 3 đoạn ở dưới đây thành sparse (thưa thớt) feature vectors:



Sau đó sửa dụng lệnh:

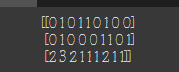
>>> print(count.vocabulary\_)

Sẽ in ra được màn hình là:



Như chúng ta thấy khi thực thi lệnh các từ sẽ được lưu ở thư viện Python, các từ (words) sẽ được nối tới giá trị số (integer). Cùng tiếp tục in feature vectors chúng ta vừa tạo bằng lệnh:

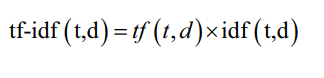
>>> print(bag.toarray())



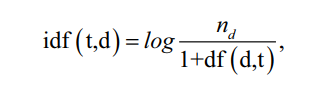
Các giá trị số ở mảng này tương ứng với các giá trị số nguyên (interger) được lưu trữ dưới dạng các mục từ điển trong thư viện CountVectorizer. Ví dụ như, giá trị 0 đầu tiên của mảng thì sẽ là từ and – chỉ xuất hiện ở cuối tài liệu, và từ is tại giá trị 1 thứ hai (của bag.toarray()) và thứ 2 của count.vocabulary\_) xuất hiện ở tất cả 3 đoạn trên. Những giá trị trong feature vectors được gọi là tuần suất thuật ngữ thô: tf(t,d) – số lần một thuật ngữ t xuất hiện trong một tài liệu d.

Assessing word relevancy via term frequency-inverse document frequency

Khi phân tích text data, chúng ta thường gặp từ sẽ xuất hiện rất nhiều phần trong tài liệu từ 2 class (ở đây là test và train). Tuần suất xuất hiện của những từ đó thường sẽ không ảnh hưởng nhiều (hay gọi là không có ích) hoặc ít có giá trị. Ở phần này, chúng ta sẽ tìm hiểu về một kỹ thuật rất hữu ích được gọi là term frequency-inverse document frequency (tf-idf) có thể sửa dụng để giảm dung lượng tần số của các chữ xuất hiện thường xuyên như đã nói ở trên trong feature vectors. tf-idf có thể định nghĩa là sản phẩm của tần suất thuật ngữ (term frequency) và nghịch đảo tuần suất thuật ngữ (inverse document frequency):



Ở đây thì tf(t,d) chúng ta đã giới thiệu ở phần trước rồi và idf(t,d) được tính bằng công thức:



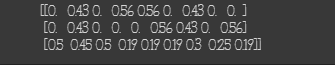
: là tất cả các tài liệu.

: là số lượng của thư mục d mà chứa thuật ngữ t.

Scikit-learn cũng chứa giải thuật này, được gọi là TfidTransformer, nó sẽ lấy tần số thuật ngữ thô (the raw term frequencies) từ CountVectorizer là đầu vào và chuyển hoá thành tf-idfs:

>>> from sklearn.feature\_extraction.text import TfidfTransformer  
>>> tfidf = TfidfTransformer()  
>>> np.set\_printoptions(precision=2)  
>>> print(tfidf.fit\_transform(count.fit\_transform(docs)).toarray())

-Kết quả-



Như chúng ta thấy từ phần trước, từ is có tuần suất xuất hiện ở hết 3 đoạn và nó trở thành từ được xuất hiện với tuần suất nhiều nhất. Tuy nhiên, sau khi chuyển hoá feature vector thành tf-idf, chúng ta sẽ xem sét xem hiện tại từ is đã được gán với số khá nhỏ qua tf-idf(0.19) ở dòng 3 và nó cũng xuất hiện ở dòng 1 và 2 vì vậy nó không hữu dụng và là giá trị không cần thiết. (đọc tiếp ở trang 239 sách tiếng Anh)

Cleaning text data

Trong phần trước chúng ta đã tìm hiểu về model bag-of-words, term frequencies và tf-idfs. Tuy nhiên, bước quan trọng đầu tiên của việc xây dựng bag-of-word model là chúng ta phải cần clean data bằng cách bỏ đi những ký tự làm nhiễu. Để hình dung việc này quan trọng như thế nào, chúng ta sẽ in ra màn hình 50 ký tự từ thư mục đầu tiên đã được xáo trộn trong review phim dataset bằng câu lệnh.

>>> df.loc[0, 'review'][-50:]

Kết quả:



Chúng ta có thể thấy ở đây, dòng chữ bị chèn vào thêm thẻ của HTML cũng như có chứa một vài ký tự khác không liên quan. Các từ của thẻ HTML thì là các từ vô nghĩa, không hữu dụng, dấu chấm ( . ) cũng có thể hữu ích -thông tin bổ sung trong các ngữ cảnh NLP nhất định. Tuy nhiên, để đơn giản hoá nó đi, chúng ta sẽ xoá hết mấy thứ này để lại các emoticon ( icon cảm xúc) như là “:)” nó cũng sẽ rất hữu dụng cho sentiment analysis. Để thực hiện phần này, chúng ta sẽ sử dụng thư viện Python’s được gọi là regular expression (regex).

>>> import re  
>>> def preprocessor(text):

... text = re.sub('<[^>]\*>', '', text)  
... emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)  
... text = re.sub('[\W]+', ' ', text.lower()) + \  
'.join(emoticons).replace('-', '')  
... return text

Với ký tự <[^>]\*> ở phần code section, chúng ta sẽ xoá được thẻ tag HTML có trong review phim. Sau khi chúng ta đã xoá được thẻ HTML< chúng ta sẽ chuyển sang tìm các biểu cảm icon và lưu trữ nó. Kế tiếp chúng ta loại bỏ các chữ cái vô nghĩa bằng [\W]+, chuyển text thành chữ thường (không in hoa), và thêm lưu trữ cho icon biểu cảm khi chạy tới cuối chuỗi.

Mặc dù thêm biểu tượng cảm xúc vào cuối chuỗi cũng không phải là cách tiếp cận hay, thứ tự của các từ trong bag-of-word model không quan trong nếu chúng ta cứ chứa 1-từ là 1-tokens.

Nhưng trước khi chúng ta nói thêm về tách thư mục, từ, hay token, hãy chắc chắn rằng code chúng ta chạy đúng

>>> preprocessor(df.loc[0, 'review'][-50:])

Kết quả:



>>> preprocessor("</a>This :) is :( a test :-)!")

Kết quả:



Cuối cùng, khi chúng ta sử dụng đi sử dụng lại clean text thì phần tiếp theo chúng ta sẽ áp dụng hàm preprocessor tới tất cả review phim vào DataFrame của chúng ta:

>>> df['review'] = df['review'].apply(preprocessor)

Processing documents into tokens

Sau khi thành công chuẩn bị reivew phim dataset, chúng ta có thể chia kho văn bản thành các “các yếu tố cá nhân” (Individual Elements). Một cách để token hoá tài liệu là chia nó thành các từ ngữ cá nhân bằng cách chia nhỏ tài liệu được xoá hết các ký tự khoảng trắng.

>>> def tokenizer(text):  
... return text.split()  
>>> tokenizer('runners like running and thus they run')

Kết quả:



Trong phần mã hoá, một kỹ thuật nữa được sử dụng là word stemming, là quá trình mà nó chuyển “từ” thành roots của nó, sau đó cho phép chúng ta ánh xa lên cùng 1 thành phần. Chúng ta sẽ sử dụng thư viện The Natural Langue Toolkit for Python (NLTK), để cài đặt chúng ta sử dụng câu lệnh: pip install nltk.

>>> from nltk.stem.porter import PorterStemmer  
>>> porter = PorterStemmer()  
>>> def tokenizer\_porter(text):  
... return [porter.stem(word) for word in text.split()]  
>>> tokenizer\_porter('runners like running and thus they run')

Kết quả:



Sử dụng PorterStemmer từ gói nltk, chúng ta xác định được hàm tokenizer để giảm từ về dạng gốc(roots), VD: running -> run ( the word running was stemmed to its root form run.).

Chúng ta sẽ huấn luyện một model để sử dụng bag-of-words model, chúng ta sẽ tóm tắt một chủ đề khá hữu dụng gọi là stop-word removal. Stop-words đơn giản là những từ cực kỳ phổ biến trong text (kể cả title), thông tin hữu ích có thể là được sử dụng để phân biệt giữa các lớp tài liệu khác nhau. Ví dụ stop-words là is,and,has. Loại bỏ stop-words có thể hữu dụng khi chúng ta làm việc với dữ liệu raw hoặc normalized term frequencies hơn là tf-idfs, thứ mà có thể giúp ta giảm dung lượng tuần suất xuất hiện của từ.

Để mà loại bỏ stop-words khỏi review phim, chúng ta sẽ sử dụng set của 127 English stop-words có sẵn trong NLTK library bằng dòng lệnh sau:

>>> import nltk  
>>> nltk.download('stopwords')

Sau khi chúng ta đã tải về stop-words xong, chúng ta có thể load và áp dụng English stop-word set theo code ở dưới đây:

>>> from nltk.corpus import stopwords  
>>> stop = stopwords.words('english')  
>>> [w for w in tokenizer\_porter('a runner likes running and runs a  
lot')[-10:] if w not in stop]

Kết quả:



Training a logistic regression model for  
document classification

Ở phần này, chúng ta sẽ huấn luyện một logistic regression model để phân loại các review phim ở dạng tích cực và tiêu cực. Đầu tiên, chúng ta sẽ chia DataFrame của clean text thành 25,000 mẫu để training và 25,000 mẫu để testing

>>> X\_train = df.loc[:25000, 'review'].values  
>>> y\_train = df.loc[:25000, 'sentiment'].values  
>>> X\_test = df.loc[25000:, 'review'].values  
>>> y\_test = df.loc[25000:, 'sentiment'].values

Tiếp theo chúng ta sẽ sử dụng GridSearchCV để tìm giá trị tối ưu cho parameters của logistic regression model của chúng ta, sử dụng 5-fold stratified cross-validation

>>> from sklearn.grid\_search import GridSearchCV  
>>> from sklearn.pipeline import Pipeline  
>>> from sklearn.linear\_model import LogisticRegression  
>>> from sklearn.feature\_extraction.text import TfidfVectorizer  
>>> tfidf = TfidfVectorizer(strip\_accents=None,  
... lowercase=False,  
... preprocessor=None)  
>>> param\_grid = [{'vect\_\_ngram\_range': [(1,1)],  
... 'vect\_\_stop\_words': [stop, None],  
... 'vect\_\_tokenizer': [tokenizer,  
... tokenizer\_porter],  
... 'clf\_\_penalty': ['l1', 'l2'],  
... 'clf\_\_C': [1.0, 10.0, 100.0]},  
... {'vect\_\_ngram\_range': [(1,1)],  
... 'vect\_\_stop\_words': [stop, None],  
... 'vect\_\_tokenizer': [tokenizer,  
... tokenizer\_porter],  
... 'vect\_\_use\_idf':[False],  
... 'vect\_\_norm':[None],  
... 'clf\_\_penalty': ['l1', 'l2'],  
... 'clf\_\_C': [1.0, 10.0, 100.0]}  
... ]  
>>> lr\_tfidf = Pipeline([('vect', tfidf),  
... ('clf',  
... LogisticRegression(random\_state=0))])  
>>> gs\_lr\_tfidf = GridSearchCV(lr\_tfidf, param\_grid,  
... scoring='accuracy',  
... cv=5, verbose=1,  
... n\_jobs=-1)  
>>> gs\_lr\_tfidf.fit(X\_train, y\_train)

Chúng ta đã khởi tạo GridSearchCV và tham số của parameter bằng code phía trên, chúng ta phải giới hạn lại số lượng tổ hợp parameter vì số lượng feature vectors, cũng như do số lượng từ vựng rất lớn. Có thể mất tới 40p để hoàn thành.

Ở phần code mẫu trước, chúng ta đã thay thế CountVectorizer và TfidfTransformer từ phần phụ bằng TfidfVectorizer, sẽ kết hợp từ lại. param\_grid tồn tại 2 thư viện parameter. Ở thư viên thứ nhất, chúng ta sử dụng TfidfTransformer với thông số mặc định (use\_idf=True, smooth\_idf=Tru, và norm=’12’) để tính toán giá trị tf-idfs, ở thư viện thứ hai chúng ta sẽ set parameter thành use\_idf=False, smooth\_idf=False và norm = None để train model dựa trên raw term frequencies. Hơn nữa là cho bản thân của bộ phân loại logistic regression, chúng ta sẽ train model sửa dụng L2 và L1 regularization bởi penalty parameter và kết hợp với regularization strengths bởi định nghĩa vùng giá trị của inverse-regularization parameter C.

>>> print('Best parameter set: %s ' % gs\_lr\_tfidf.best\_params\_)

Kết quả:

Chúng ta có thể thấy ở đây, chúng ta sở hữu C=10

Sử dụng model tốt nhất từ grid search, in thử trung bình của 5-fold cross-validation điểm chính xác của training set và phân loại chính xác trên test dataset:

>>> print('CV Accuracy: %.3f'  
... % gs\_lr\_tfidf.best\_score\_)

Kết quả:

>>> clf = gs\_lr\_tfidf.best\_estimator\_  
>>> print('Test Accuracy: %.3f'  
... % clf.score(X\_test, y\_test))

Kết quả:

Working with bigger data – online algorithms and out-of-core learning

Khi xong phần này chúng ta nhận ra yếu điểm rằng khi xài grid thì chúng ta cần phải dùng một lượng lớn tài nguyên để tạo thành feature vectors đối với 50,000 mẫu review phim trong lúc search grid.

Chúng ta đã được tiếp cận stochastic gradient descent từ các chương trước, nó là một thuật toán tối ưu hoá update model weight sử dụng 1 mẫu tại một thời điểm. Ở phần này, chúng ta sẽ sử dụng hàm partial\_fit của SGDClassifier ở scikit-learn. Ta sẽ train logistic regression model thông qua cách này.

Đầu tiên chúng ta sẽ định nghĩa hàm tokenizer để clean data từ file movie\_data.csv file mà chúng ta đã làm từ đầu chương và tách ra các word token trong khi xoá các stop-words.

>>> import numpy as np  
>>> import re  
>>> from nltk.corpus import stopwords  
>>> stop = stopwords.words('english')  
>>> def tokenizer(text):  
... text = re.sub('<[^>]\*>', '', text)  
... emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',  
... text.lower())  
... text = re.sub('[\W]+', ' ', text.lower()) \

... + ' '.join(emoticons).replace('-', '')  
... tokenized = [w for w in text.split() if w not in stop]  
... return tokenized

Bây giờ chúng ta khởi động hàm, stream\_docs, là đọc một lần và trả về giá trị tài liệu trong một khoảng thời gian.

>>> def stream\_docs(path):  
... with open(path, ‘r’, encoding=‘utf-8’) as csv:  
... next(csv) # skip header  
... for line in csv:  
... text, label = line[:-3], int(line[-2])  
... yield text, label

Để biết stream\_docs hoạt động đúng hay không, chúng ta sẽ đọc dòng đầu tiên của tài liệu từ file movie\_data.csv và nó sẽ trả lại ta các dòng chữ review phim cũng như nhãn tương ứng.

>>> next(stream\_docs(path='./movie\_data.csv'))

Kết quả: 

Bây giờ chúng ta sẽ định nghĩa hàm, get\_minibatch, nó sẽ lấy dòng tài liệu từ hàm stream\_docs và trả về một số của tài liệu được chỉ định bởi size của parameter

>>> def get\_minibatch(doc\_stream, size):  
... docs, y = [], []  
... try:  
... for \_ in range(size):  
... text, label = next(doc\_stream)  
... docs.append(text)  
... y.append(label)  
... except StopIteration:  
... return None, None  
... return docs, y

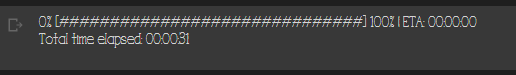
Chúng ta sẽ sử dụng một vectorizer khác để thực thi quá trình chạy, nó nằm trong scikit-learn được gọi là HashingVectorizer

>>> from sklearn.feature\_extraction.text import HashingVectorizer  
>>> from sklearn.linear\_model import SGDClassifier

>>> vect = HashingVectorizer(decode\_error='ignore',  
... n\_features=2\*\*21,  
... preprocessor=None,  
... tokenizer=tokenizer)  
>>> clf = SGDClassifier(loss='log', random\_state=1, n\_iter=1)  
>>> doc\_stream = stream\_docs(path='./movie\_data.csv')

>>> import pyprind  
>>> pbar = pyprind.ProgBar(45)  
>>> classes = np.array([0, 1])  
>>> for \_ in range(45):  
... X\_train, y\_train = get\_minibatch(doc\_stream, size=1000)  
... if not X\_train:  
... break  
... X\_train = vect.transform(X\_train)  
... clf.partial\_fit(X\_train, y\_train, classes=classes)  
... pbar.update()

Kết quả:



Chúng ta cũng sử dụng PyPrind thôi, giống như phần đầu ta làm vậy.

Ta sẽ đặt vòng for chạy 45 lần lặp, và trong vòng lặp đấy chúng ta sẽ lặp đi lặp lại 45 lần minibatches của tài liệu nơi mà mỗi minibatches chứa 1000 mẫu.

Sau khi chạy xong, chúng ta sẽ sử dụng 5000 mẫu cuối cùng để đánh giá hiệu suất mô hình.

>>> X\_test, y\_test = get\_minibatch(doc\_stream, size=5000)  
>>> X\_test = vect.transform(X\_test)  
>>> print('Accuracy: %.3f' % clf.score(X\_test, y\_test))

Kết quả:



Như chúng ta có thể thấy độ chính xác là 87%. Hoá ra phương pháp học out-of-core rất hiệu quả về mặt bộ nhớ và tốn rất ít thời gian để hoàn thành, không lâu như grid search. Cuối cùng ta có thể sửa dụng 5,000 mẫu cuối cùng để update model của chúng ta

>>> clf = clf.partial\_fit(X\_test, y\_test)

--Hết--