Bài 20 - Recommendation Neural Network

26 Dec 2019 - phamdinhkhanh

Menu

1. Ưu điểm của Recommendation Neural Network

Ở bài trước chúng ta đã tìm hiểu về 2 thuật toán cơ bản trong recommendation đó là collaborative và content-based filtering (https://phamdinhkhanh.github.io/2019/11/04/Recommendation_Compound_Part1.html). Ý tưởng chung của các thuật toán này đều là tìm cách ước lượng các giá trị rating của những cặp user-item chưa được rate bởi người dùng sao cho sai số so với giá trị thực tế là nhỏ nhất.

Tuy nhiên những phương pháp này đều có một hạn chế đó là:

- Nếu một user hoặc item chưa từng xuất hiện trong cơ sở dữ liệu thì sẽ không thể xây dựng được phương trình hồi qui để dự báo rating cho sản phẩm. Do đó không thể khuyến nghị được cho khách hàng mới.
- Những sản phẩm được sử dụng phổ biến và sản phẩm có rating cao thì xuất hiện trong hầu hết các khuyến nghị. Điều đó chứng tỏ tính cá nhân hóa của những thuật toán này chưa cao.

Để khắc phục hạn chế này chúng ta dựa vào một phương pháp học sâu dựa trên mạng Neural Network giúp xây dựng một véc tơ embedding nhằm trích suất các đặc trưng của người dùng và sản phẩm. Từ đó có thể dễ dàng áp dụng để:

Top

- Tìm ra các sản phẩm tương đồng hoặc tập khách hàng tương đồng dựa trên các thước đo như hệ số tương quan cosine similarity hoặc dot product.
- Ước lượng được điểm số mà một khách hàng sẽ vote cho một sản phẩm để khuyến nghị những sản phẩm mà khách hàng yêu thích nhất.

Phương pháp neural network tận dụng được đồng thời đầu vào bao gồm các trường thông tin sản phẩm và các trường thông tin khách hàng nên tính cá nhân hóa cao hơn những phương pháp truyền thống. Bên cạnh đó, việc sử dụng đa dạng các trường thông tin cá nhân hóa giúp cho các khuyến nghị sản phẩm trở nên chính xác hơn. Chẳng hạn như hệ thống recommendation video của youtube có khả năng khuyến nghị video cho người dùng theo ngôn ngữ của họ.

Trong khi đó đối với mô hình collaborative-filtering ta chỉ sử dụng thông tin duy nhất là ma trận rating sản phẩm. Còn trong mô hình content-based chỉ tận dụng được chỉ một trong hai nhóm thông tin về khách hàng hoặc về sản phẩm.

Những thông tin về người dùng rất hữu ích trong nhận biết hành vi và sở thích của họ. Đặc biệt là các trường giới tính, độ tuổi, ngành nghề, quốc tịch,… sẽ cá nhân hóa người dùng rất cao. Minh chứng cho điều đó, tôi lấy ví dụ:

- Tùy theo quốc tịch mà người dùng sẽ có những thị hiếu khác nhau như người Việt Nam thích ăn thịt chó nhưng người phương Tây thì tuyệt đối không.
- Nam thích bóng đá, hút thuốc, đồ điện tử và công nghệ thông tin còn nữ thích thời trang, mỹ phẩm-làm đẹp, chăm sóc sức khỏe, nội trợ.
- Ta không thể khuyến nghị đồ chơi trẻ em cho người lớn và trái lại thuốc lá cho trẻ em. Hoặc với nam thì không nên recommend túi xách, váy (trừ khi họ mua cho bạn gái, vợ, người thân) hoặc nữ là áo sơ mi nam.

Trên thực tế, số lượng trường dữ liệu đối với khách hàng hoặc đối với sản phẩm ở một số hệ thống là rất lớn. Có thể lên tới vài trăm hoặc thậm chí vài nghìn chiều. Nếu sử dụng những mô hình truyền thống của collaborative filtering và content-based sẽ gặp hạn chế lớn là không tận dụng được toàn bộ các thông tin này.

Ngoài ra mô hình mạng neural network có ưu điểm đó là không chỉ tiếp nhận đầu vào là các biến numeric hoặc category. Các kiểu dữ liệu đặc biệt như các bức ảnh, các đoạn văn bản mô tả thuộc tính sản phẩm cũng sẽ được đưa vào mô hình. Và chúng cực kì có ích trong khuyến nghị sản phẩm. Bằng các phương pháp nhúng hình ảnh và văn bản, chúng có thể được dễ dàng đưa chúng vào mô hình như các chiều thông tin của sản phẩm. Các nghiên cứu về recommendation cho thấy những thuật toán sử dụng thêm những dữ liệu hình ảnh, mô tả sản phẩm lại có kết quả tốt hơn. Các bạn có thể tham khảo thêm tại Machine Learning for Recommender systems — Part 2 (Deep Recommendation, Sequence Prediction, AutoML and Reinforcement Learning in Recommendation) - Pavel Kordík (https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-2-deep-recommendation-sequence-prediction-automl-f134bc79d66b).

Chính vì những ưu điểm sử dụng đa dạng hóa dữ liệu, cả về phía người dùng và sản phẩm mà mô hình recommendation mạng neural network được ưa chuộng sử dụng hơn là các phương pháp cũ. Tiếp theo hãy cùng tìm hiểu về kiến trúc mô hình và nguyên lý hoạt động của một hệ thống recommendation qua mạng neural network như thế nào nhé.

2. Kiến trúc recommendation neural network

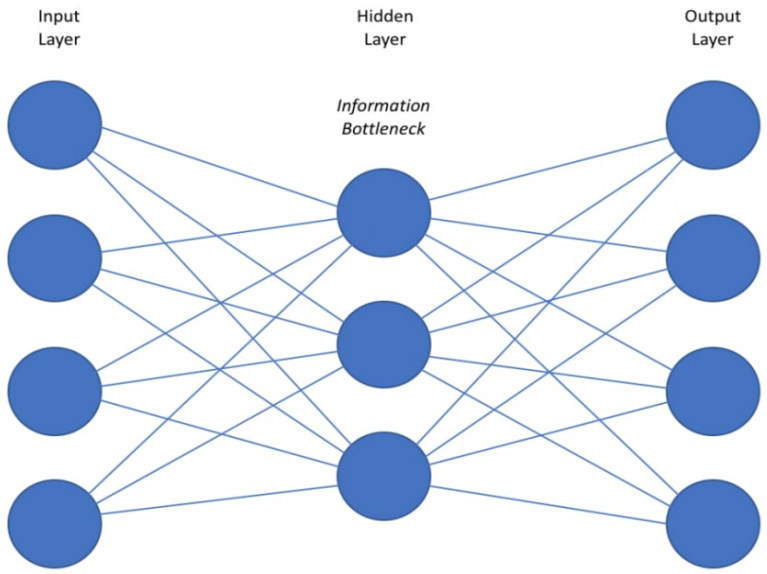
Top

Kiến trúc của recommendation neural network là một mạng neural khá đơn giản với input đầu vào của mô hình là:

- Các thông tin về khách hàng như lịch sử truy vấn, hành vi giao dịch, các biến demographic bao gồm giới tính, độ tuổi, nghề nghiệp, quốc gia,…
- Các thông tin về sản phẩm: Mô tả sản phẩm, hình ảnh, thể loại, mô tả sản phẩm,….

Đầu ra của mô hình là một véc tơ phân phối xác suất đánh giá khả năng tương tác của khách hàng đối với các sản phẩm. Một số trường hợp khác là phân phối điểm rating của sản phẩm mà khách hàng đã rate. Tùy vào cách xác định mục tiêu là dự báo xác suất hay dự báo điểm số mà chúng ta sẽ xác định output, hàm mục tiêu cho bài toán recommendation.

Kiến trúc của mô hình recommendation neural network là một mạng neural network đơn giản gồm input layer, hidden layer và output layer. Cụ thể, bên dưới là kiến trúc của một mô hình recommendation neural network:



Hình 1: Mô hình neural network recommendation. Ở mô hình này chúng ta sẽ có Input layer là những thông tin của khách hàng, thông tin sản phẩm, lịch sử giao dịch hoặc truy vấn. Những đầu vào này có thể được coi như là thông tin truy vấn (query). Một hidden layer trung gian để giảm chiều dữ liệu và chuyển đổi tính phi tuyến và output layer là một véc tơ phân phối xác suất về khả năng ưa thích hoặc một véc tơ embedding điểm rating của khách hàng.

Như vậy về bản chất mô hình này chính là một mạng MLP thông thường giúp dự báo sản phẩm khách hàng có khả năng tương tác dựa trên các dữ liệu lịch sử hành vi của khách hàng. Kiến trúc này khá đơn giản phải không? Đầu vào của mạng nơ ron là các véc tơ véc tơ $\mathbf{x}_i \in \mathbb{R}^{d\times 1}$ đại diện cho véc tơ concatenate các đặc trưng về sản phẩm và các đặc trưng về khách hàng dựa trên những gì khách hàng đã giao dịch, véc tơ này có kích thước d chiều. Khi đó thông qua một phép chiếu tuyến tính (linear projection) bằng cách nhân với ma trận hệ số $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times h}$ ta sẽ giảm chiều dữ liệu của véc tơ input về véc tơ $\mathbf{z} \in \mathbb{R}^{h\times 1}$, với h là số units của hidden layer.

$$\mathbf{z}_i = \mathbf{W}^{(1)^\top}\mathbf{x}_i, \mathbf{z}_i \in \mathbb{R}^{h\times 1}$$

Top

Để thay đổi tính phi tuyến thì hàm activation relu hoặc sigmoid sẽ được áp dụng ở ngay lên output của layer liền trước. Khi đó output sẽ có dạng:

$$\mathbf{a}_i = f(\mathbf{z}_i) \in \mathbb{R}^{h \times 1}$$

$f(\mathbf{x})$ chính là hàm relu hoặc softmax. Biến đổi trên có thể coi là một phép biến đổi không gian qua kernel hàm $f(\mathbf{x})$ để thay đổi mối quan hệ đầu vào và đầu ra từ tuyến tính sang phi tuyến. Khi đó ta có thể xem các giá trị $f(\mathbf{z}) \in \mathbb{R}^{h \times 1}$ như là một véc tơ user embedding đại diện cho hành vi, sở thích, thị hiếu của mỗi user.

Để tính toán phân phối xác suất ở đầu ra ta sử dụng hàm softmax áp dụng lên các đầu vào $\mathbf{a}_i$. Như vậy giá trị xác suất của khách hàng thứ i khi ưa thích sản phẩm thứ j sẽ chính là:

$$\hat{y}_{ij} = P(y_i = j | \mathbf{a}_i, j = \overline{1, o}) = \frac{\exp(\mathbf{a}_i^T \mathbf{w}_j^{(2)})}{\sum_{k=1}^{o} \exp(\mathbf{a}_i^T \mathbf{w}_k^{(2)})}$$

Ở đây $\mathbf{w}_k^{(2)} \in \mathbb{R}^{h \times 1}$ là những véc tơ cột của ma trận hình chiếu (projection matrix) $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times o}$ giúp biến đổi đầu ra của hidden layer sang output layer. o chính là số lượng sản phẩm ở output có kích thước rất lớn. Mỗi một cột $\mathbf{w}_i \in \mathbb{R}^{h \times 1}$ của ma trận hình chiếu được xem như là biểu diễn của một véc tơ item embedding đại diện cho một sản phẩm.

Sau khi huấn luyện mô hình ta thu được các véc tơ nhúng user embedding và item embedding của lần lượt khách hàng và sản phẩm. Điểm số yêu thích của một khách hàng đối với một sản phẩm có thể được đo lường thông qua `cosine similarity` hoặc `dot product`. Sẽ được trình bày cụ thể hơn ở phần thực hành mục 4.

Hàm loss chính là cross entropy được sử dụng để đo lường sự khác biệt của ground truth với phân phối xác suất được dự báo có công thức như sau:

$$\mathcal{L}(\mathbf{X}; \mathbf{W}) = -\sum_{i=1}^{n} \sum_{k=1}^{o} y_{ik} . \log(\hat{y}_{ik})$$

Qúa trình tối ưu hóa hàm loss function sẽ cần phải trải qua 2 quá trình lan truyền thuận và lan truyền ngược. Lan truyền thuận sẽ tính toán phân phối xác suất ở output và lan truyền ngược cập nhật lại hệ số của các ma trận $\mathbf{W}$.

Để hiểu thêm về qúa trình lan truyền thuận và lan truyền ngược của thuật toán có thể xem tại Bài 14: mô hình MLP blog machine learning cơ bản (https://machinelearningcoban.com/2017/02/24/mlp/).

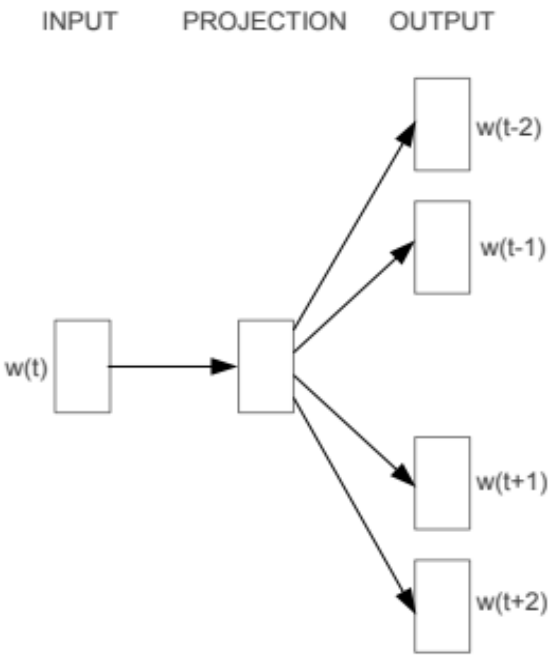
3. Negative sampling

Có một lưu ý khá quan trọng trong bài toán recommendation neural network đó là số lượng classes rất lớn. Thậm chí là lên tới vài triệu sản phẩm, dẫn tới chi phí tính toán xác suất ở đầu ra là rất cao. Để giảm thiểu chi phí tính toán, phương pháp Negative sampling ra đời giúp tính toán phân phối xác suất đầu ra và cập nhật nghiệm nhanh hơn.

Vậy Negative sampling là gì? Đây là một phương pháp huấn luyện nhanh các mô hình có nhiều classes ở đầu ra. Thường được sử dụng trong các mô hình word embedding như word2vec COB hay skip-gram.

Hãy tưởng tượng trong mô hình word2vec theo phương pháp skip-gram. Từ một từ bối cảnh (context word) ở vị trí t ta muốn xây dựng một phép chiếu sao cho thông qua phép chiếu ta có thể dễ dàng dự báo được những từ ở gần nó nhất, đó là các từ mục tiêu (target words) như hình bên dưới:

Top



Hình 2: Sơ đồ kiến trúc mô hình word2vec skip-gram.

Sau khi huấn luyện mô hình skip-gram ta sẽ thu được các véc tơ nhúng của từ thuộc không gian d chiều. Khi đó kí hiệu véc tơ biểu diễn từ input là $e_I \in \mathbb{R}^d$ và các từ output là các véc tơ $e'_i \in \mathbb{R}^d$. Như chúng ta đã biết ở Bài 3 - Mô hình Word2Vec (https://phamdinhkhanh.github.io/2019/04/29/ModelWord2Vec.html), output của mô hình skip-gram sẽ dự báo xác suất để 2 từ bất kì thuộc cùng một cặp. Khi đó đối với các cặp từ (w_I, w_O) đại diện cho (input, output), xác suất có điều kiện để chúng xuất hiện cùng nhau phải là lớn nhất:

$$P(y=1|(w_I, w_O)) = \frac{\exp(e_I^T e'_O)}{\sum_{j=1}^{V} \exp(e_I^T e'_j)} \quad (1)$$

Đây chính là một hàm softmax tính xác suất của từ w_O trong trường hợp đầu vào là từ w_I. Với mẫu số chính là tổng lũy thừa cơ số tự nhiên e của toàn bộ các từ xuất hiện trong bộ từ điển có kích thước là V.

Để thuận lợi cho việc tính đạo hàm, lấy logarit 2 vế và đổi dấu ta thu được hàm loss function dạng cross entropy như sau:

$$\mathcal{L}(w_I, w_O) = -\sum_{j=1}^{N}\sum_{i=1}^{V}[\log P(y_j=1|(w_I, w_O))]$$

Với N là số lượng các cặp (input, output) của mô hình. Thế phương trình (1) vào ta có:

$$\mathcal{L}(w_I, w_O) = -e_I^T e'_O + \log(\sum_{j=1}^{V} \exp(e_I^T e'_j))$$

Khi đó biến cần tinh chỉnh của hàm loss function chính là các véc tơ biểu diễn từ output e'_j. Đặt $e_I^T e'_j = f(w_j)$. Đạo hàm của hàm loss function tại một từ input sẽ có dạng:

Top

$$\mathcal{L}(w_I, w_O) = -f(w) + \log(\sum_{j=1}^{V} \exp(f(w_j)))$$

Theo công thức chain rule ta có $g'(f'(x)) = f'(x)g'(f)$. Do đó:

$$\frac{\delta(\log(g(x)))}{\delta x} = \frac{\delta \log(g(x))}{\delta g(x)} . \frac{\delta g(x)}{\delta x} = \frac{1}{g(x)} \frac{\delta g(x)}{x}$$

Thế vào tính đạo hàm loss function:

$$\begin{aligned}
\frac{\delta \mathcal{L}(w_I, w_O)}{\delta \theta} &= -\frac{\delta f(w)}{\delta \theta} + \frac{1}{\sum_{j=1}^{V}(\exp(f(w_j)))} . \frac{\delta \sum_{j=1}^{V} \exp(f(w_j))}{\delta \theta} \\
&= -\frac{\delta f(w)}{\delta \theta} + \sum_{j=1}^{V} \frac{\exp(f(w_j))}{\sum_{j=1}^{V} \exp(f(w_j))} \frac{\delta f(w_i)}{\delta \theta} \\
&= -\frac{\delta f(w)}{\delta \theta} + \sum_{j=1}^{V} P(w_i)[\frac{\delta f(w_i)}{\delta \theta}] \\
&= -\frac{\delta f(w)}{\delta \theta} + \mathbf{E}_{p \sim P(w)}[\frac{\delta f(w_i)}{\delta \theta}]
\end{aligned}$$

Dòng thứ 2 suy ra dòng thứ 3 là do $\frac{\exp(f(w_j))}{\sum_{j=1}^{V} \exp(f(w_j))}$ chính là hàm softmax tính xác suất để từ output là w_i.

Dòng thứ 3 suy ra dòng thứ 4 là do gía trị $\mathbf{E}_{p \sim P(w)}$ chính là kì vọng theo phân phối trọng số là $p \sim P(w)$.

Trong trường hợp bộ từ điển có kích thước rất lớn thì việc tính toán $\mathbf{E}_{p \sim P(w)}$ cho toàn bộ các từ vựng sẽ rất tốn chi phí tính toán. Khi đó, một phương pháp được Mikolov và các cộng sự giới thiệu để làm giảm chi phí tính toán mẫu số đó là chỉ lựa chọn ra một tợp hợp k hữu hạn các cặp mẫu âm tính (negative sample) kết hợp với 1 mẫu dương tính (positive sample). Mẫu âm tính được hiểu là các quan sát mà (w_I, w_O) không là một cặp đầu vào và đầu ra của nhau. Trường hợp ngược lại chúng là các mẫu dương tính (positive sample).

Các lượt cập nhật gradient descent, trong công thức đạo hàm loss function thì chỉ phải tính $\mathbf{E}_{p \sim P(w)}$ cho $k+1$ mẫu thay vì tính phân phối xác suất cho toàn bộ V mẫu. Sẽ đơn giản và tiết kiệm chi phí tính toán hơn rất nhiều.

Thông thường giá trị $k = 25$ sẽ là hiệu quả khi mẫu có kích thước nhỏ và giá trị $k = 5$ là hiệu quả với các mẫu kích thước lớn theo bài báo Negative Sampling - Distributed Representations of Words and Phrases and their Compositionality - Tomas Mikolov. etc (https://arxiv.org/pdf/1310.4546.pdf). Sau khi áp dụng kĩ thuật negative sampling thì kết quả được cải thiện đáng kể đồng thời tốc độ huấn luyện nhanh hơn gấp nhiều lần.

3. Các khó khăn khi xây dựng mô hình

Về cơ bản lý thuyết và kiến trúc của mô hình khá đơn giản. Tuy nhiên khi áp dụng vào thực tiễn sẽ có những vấn đề về dữ liệu, huấn luyện và dự báo mà chúng ta sẽ gặp phải. Đây là những trở ngại rất lớn để biến mô hình từ giai đoạn nghiên cứu phát triển đến ứng dụng. Bên dưới tôi sẽ lần lượt liệt kê ra những khó khăn theo các khía cạnh trên:

Top

3.1. Dữ liệu

Một mô hình sẽ không tốt nếu dữ liệu không được thu thập đầy đủ. Đầy đủ ở đây nghĩa là không có quá nhiều dữ liệu bị missing, số lượng các trường được thu thập là bao quát các nhóm dữ liệu gồm người dùng, sản phẩm, lịch sử tương tác. Nhưng dữ liệu đầy đủ cũng chỉ là một yêu cầu trong số các yêu cầu quan trọng. Xa hơn, chúng ta cần dữ liệu đó phải chính xác để phản ánh đúng thực tiễn. Tuy nhiên nhiều bộ dữ liệu sử dụng để huấn luyện recommendation thì thường không được như thế:

- Thực tế tại nhiều công ty, dữ liệu về người dùng chưa được thu thập chính xác và đầy đủ. Đặc biệt là khâu đăng kí tài khoản được thực hiện sơ sài nhằm tạo sự nhanh chóng, tiện lợi. Nhưng theo tôi thiệt hại cho cả khách hàng và doanh nghiệp lớn hơn nhiều. Vì thông tin không chuẩn xác và đầy đủ dẫn tới thuật toán hoạt động sai. Khách hàng sẽ không được recommend đúng sản phẩm mình cần. Doanh nghiệp tiếp cận không đúng đối tượng khách hàng.

- Các trường dữ liệu người dùng được thu thập để đưa vào mô hình cần phải được chuẩn hóa và thực sự có ý nghĩa phân biệt hành vi, sở thích, thị hiếu khách hàng. Về cơ bản chúng ta nên nhóm các thông tin khách hàng thành các nhóm trường:
    - Demographic: Biến nhân khẩu học là các trường xác định khách hàng như giới tính, độ tuổi, quê quán, nghề nghiệp, bằng cấp, thu nhập, thành phần gia đình,…. Tất nhiên để đảm bảo lấy được toàn bộ các thông tin nêu trên là rất khó. Tùy từng ngành nghề mà ta sẽ xác định đâu là thông tin cần để thu thập.

    - Behavioral: Các biến liên quan đến hành vi mua sắm, chi tiêu, giao dịch, lướt web của khách hàng. Đây là những biến rất quan trọng vì nó thể hiện trực tiếp những gì khách hàng đã trải nghiệm. Để dữ liệu này được đầy đủ thì bạn cần phải có một hệ thống tracking app, web đủ mạnh. Google Tag Management, Google Firebase và Mixpanel có thể là những lựa chọn mà bạn nên cân nhắc. Trong trường hợp công ty của bạn có điều kiện tài chính có thể thuê một bên tư vấn về SEO để thiết lập các cấu hình chuẩn xác và được tư vấn thiết kế báo cáo, phân tích kết quả tracking.

- Nên sử dụng đa dạng các loại thông tin người dùng, sản phẩm. Các trường thông tin không chỉ giới hạn ở định dạng numeric hoặc category mà còn có thể là hình ảnh sản phẩm, nội dung mô tả sản phẩm, âm thanh, video. Những thông tin này có thể dễ dàng được nhúng dưới các véc tơ embedding thông qua các mô hình trong Computer Vision và NLP mà mang lại nhiều thông tin để khuyến nghị sản phẩm tới người dùng hiệu quả hơn.

3.2. Huấn luyện mô hình

Huấn luyện mô hình là một phần rất quan trọng. Bên cạnh việc lựa chọn kiến trúc mô hình nào mang lại hiệu quả cao, chúng ta cần phải liên tục huấn luyện mô hình. Dữ liệu thay đổi realtime nên việc liên tục huấn luyện mô hình sẽ kịp thời cập nhật những xu hướng hoặc hành vi tiêu dùng mới nhất của khách hàng. Qúa trình học này gọi là online learning, một trong những vấn đề rất quan trọng của recommendation. Việc học online learing để khuyến nghị sản phẩm cho khách hàng là rất cần thiết, đặc biệt là khi nhu cầu của khách hàng chỉ phát sinh trong một giai đoạn rất ngắn và trên thị trường có hàng trăm hãng cạnh tranh. Hiện tại facebook và google là những hãng áp dụng rất tốt quá trình học online learning. Bạn có thể tự cảm nhận điều này thông qua việc tìm kiếm sản phẩm và được đưa tin quảng cáo tràn ngập về sản phẩm đó chỉ sau vài giây.

Để học được online learning đối với các nền tảng hàng triệu người dùng thì chúng ta phải có một tài nguyên vật lý đủ lớn có thể đáp ứng được quá trình học liên tục. Sau mỗi một lượt click, mua sắm, thêm hàng vào giỏ hoặc thanh toán thì mô hình cần được ghi nhận những thay đổi. Để hạn chế số lượt kích hoạt huấn luyện mô hình quá nhiều dễ dẫn tới concurrent, chúng ta cần tổng hợp

Top

các dữ liệu mới vào một batch để huấn luyện mô hình một lần cho mỗi batch. Các phương pháp huấn luyện của mô hình deep learning được hướng dẫn khá chi tiết tại Bài 8: Gradient Descent-Blog machinelearningcoban (https://machinelearningcoban.com/2017/01/16/gradientdescent2/).

Ngoài ra, chúng ta nên kết hợp nhiều lớp mô hình khác nhau trên cùng một hệ thống recommendation, mỗi mô hình sẽ được sử dụng chuyên biệt cho một nhiệm vụ khác nhau như:

- Khuyến nghị sản phẩm người dùng có thể quan tâm dựa trên lịch sử các sản phẩm đã mua: Sử dụng các mô hình deep learning neural network.
- Khuyến nghị sản phẩm cùng loại với sản phẩm người dùng đang tìm kiếm: Có thể dùng các thuật toán similarity search về nội dung, hình ảnh hoặc đối chiếu similarity của các véc tơ embedding sản phẩm.
- Dự báo sản phẩm có khả năng mua tiếp theo dựa trên sản phẩm khác hàng đã mua gần đây: Sử dụng các thuật toán RNN, LSTM, GRU, BiLSTM.

Khi đó hệ thống recommendation sẽ giúp nâng cao trải nghiệm của khách hàng hơn nhiều so với chỉ có một mô hình recommendation duy nhất.

3.3. Dự báo

Để đưa ra dự báo cho mô hình đảm bảo chuẩn xác và nhanh chóng. Chúng ta phải giải quyết 2 bài toán lớn song song đó là: Nâng cao độ chính xác cho mô hình và nâng cao tốc độ xử lý. Cả 2 bài toán đều rất quan trọng.

Về độ chính xác thì cần phải chuẩn bị tốt khâu dữ liệu đầu vào và liên tục thử nghiệm những kiến trúc khác nhau. Đồng thời cũng cần thường xuyên nghiên cứu để cập nhật những phương pháp, xu hướng mới trong recommendation trên thế giới. Việc nghiên cứu đòi hỏi phải chuyên sâu và công phu về cả các khía cạnh lý thuyết, xây dựng mô hình, triển khai mô hình. Nếu có điều kiện, các doanh nghiệp nên xây dựng ngay từ đầu một đội ngũ R&D đủ mạnh và chuyên sâu nghiên cứu lớp mô hình này.

Để nâng cao tốc độ xử lý cho một hệ thống recommendation thì chúng ta cần áp dụng các kĩ thuật xử lý phân tán và xử lý song song. Đó là những kiến trúc dạng master-slaves. Master có vai trò như là một người quản lý kết quả công việc và giao việc tới các máy con là những slaves. Ngoài ra khi một xử lý bị lỗi sẽ hỗ trợ auto-retry. Do đó hệ thống xử lý phân tán sẽ có khả năng chịu tải và chịu lỗi tốt. Bạn đọc có thể tham khảo một bài viết rất hay về thiết kế một hệ thống triệu view có tên là Nghệ thuật xử lý background job - minhmomen (https://viblo.asia/p/nghe-thuat-xu-ly-background-job-07LKXjqJlV4).

Hi vọng là những khó khăn mà mình liệt kê trên không làm bạn nản chí. Không có gì là dễ dàng để có được một sản phẩm tốt. Chúng ta cần phải thấy trước các khó khăn, chuẩn bị các phương án có thể để giải quyết các vấn đề và sẵn sàng cải tiến, nâng cấp sản phẩm ngày càng tốt hơn.

Tiếp theo sẽ là phần rất thú vị, đó là thực hành xây dựng một mô hình recommendation liên quan chính đến nội dung bài viết ngày hôm nay.

4. Xây dựng mô hình

4.1. Dữ liệu

Dữ liệu được sử dụng trong thực hành là MovieLen-100k (https://grouplens.org/datasets/movielens/100k/). Đây là một bộ dữ liệu khá đầy đủ để thực hành các thuật toán recommendation bởi vì nó chứa đầy đủ cả 3 thông tin: Ratings, Users và Movies. Trong đó file ratings chính là ma trận tiện ích (utility) chứa 100.000 giá trị rating của các cặp (user,

Top

movie). File users là thông tin của 943 người dùng của trang phim MovieLen và file movies là thông tin về gần 1682 bộ phim được rating. Mỗi users sẽ rating ít nhất 20 bộ phim và cụ thể các trường thông tin trong từng file như sau:

RATINGS: Bao gồm các trường `UserID|MovieID|Rating|Timestamp` . Trong đó:

- UserIDs có giá trị nằm trong khoảng 1 đến 943 là id xác định người dùng.
- MovieIDs có giá trị nằm trong khoảng 1 đến 1682 là các id của các bộ phim.
- Ratings là giá trị đánh giá của người dùng theo thang đo 5 sao.
- Timestamp biểu diễn thời điểm tiến hành rating.

USERS: Thông tin người dùng bao gồm các trường `UserID|Gender|Age|Occupation|Zip-code` . Trong đó:

- Gender giới tính nhận 2 giá trị `F` là nữ, `M` là nam.
- Age khoảng tuổi của khách hàng.
- Occupation Nghề nghiệp của người dùng bao gồm các nghề: `'technician', 'other', 'writer', 'executive', 'administrator', 'student', 'lawyer', 'educator', 'scientist', 'entertainment', 'programmer', 'librarian', 'homemaker', 'artist', 'engineer', 'marketing', 'none', 'healthcare', 'retired','salesman', 'doctor'` .

MOVIE: Thông tin của các bộ phim gồm các trường `MovieID|Title|Genres` . Trong đó:

- Titles: Tiêu đề của bộ phim.
- Genres: Thể loại phim gồm các loại như: `["genre_unknown", "Action", "Adventure", "Animation", "Children", "Comedy", "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"]`

Mục tiêu của chúng ta là xây dựng một mô hình mạng neural dự báo xác suất để một khách hàng sẽ tương tác với một bộ phim nào đó. Đầu vào của mô hình sẽ là những thông tin về khách hàng và lịch sử rating các bộ phim của họ và các trường thông tin liên quan đến bộ phim. Đầu ra là giá trị index của bộ phim mà khách hàng có tương tác, giá trị index này sẽ được chuyển sang one-hot và sẽ được giải thích rõ hơn ở mục 4 thực hành.

Cụ thể các biến đầu vào của mô hình sẽ như thế nào?

Biến input của mô hình sẽ bao gồm:

- Nhóm các trường thông tin về khách hàng như: giới tính, độ tuổi và nghề nghiệp. Nếu là biến numeric sẽ được giữ nguyên, trái lại các biến category sẽ được biến đổi thành véc tơ one-hot.
- Nhóm các thông tin về lịch sử rating của khách hàng: Là các `movie_id` mà khách hàng đã rate. Những biến này được xem như các biến category và sẽ được nhúng thành one-hot véc tơ.
- Nhóm thông tin về bộ phim: Là các véc tơ liên quan đến bộ phim đó. Chủ yếu là các biến category bao gồm year (năm sản suất), genre (thể loại). Biến year sẽ được giữ nguyên, biến genre được chuyển sang one-hot véc tơ như xử lý category thông thường.

4.2. Khảo sát dữ liệu.

Bên dưới chúng ta sẽ cùng khảo sát dữ liệu theo vài chiều thông tin về người dùng, sản phẩm. Đầu tiên là đọc dữ liệu `users, movies, ratings`

Top

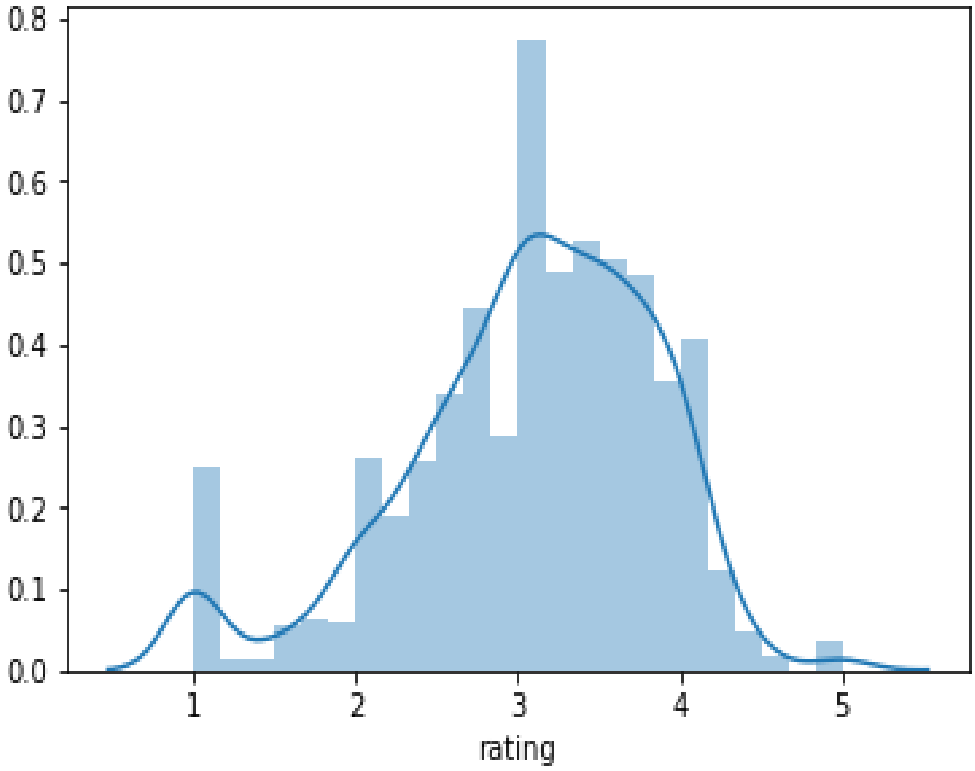
```
import pandas as pd
import numpy as np
import os

# Đọc dữ liệu users
def _read_users():
  users_cols = ['user_id', 'age', 'sex',
                'occupation', 'zip_code']
  users = pd.read_csv('ml-100k/u.user', sep='|',
                      names=users_cols, encoding='latin-1',
                      dtype={'user_id':str,
                             'age':str})
  return users

# Đọc dữ liệu ratings
def _read_ratings():
  ratings_cols = ['user_id', 'movie_id',
                  'rating', 'unix_timestamp']
  ratings = pd.read_csv('ml-100k/u.data', sep='\t',
                        names=ratings_cols, encoding='latin-1',
                        dtype={'user_id':str,
                               'movie_id':str})
  return ratings

# Đọc dữ liệu về các bộ phim
def _read_movies():
  genre_cols = ["genre_unknown", "Action", "Adventure",
                "Animation", "Children", "Comedy",
                "Crime", "Documentary", "Drama",
                "Fantasy", "Film-Noir", "Horror",
                "Musical", "Mystery", "Romance",
                "Sci-Fi", "Thriller", "War", "Western"]
  movies_cols = ['movie_id', 'title', 'release_date',
                 "video_release_date", "imdb_url"]
  all_movies_cols = movies_cols + genre_cols
  movies = pd.read_csv('ml-100k/u.item', sep='|',
                       names=all_movies_cols, encoding='latin-1',
                       dtype={'movie_id':str})
  # Đánh dấu các dòng có tổng các thể loại phim genres = 0.
  count_genres = [0 if count >= 1 else 1 for count in movies[genre_co
  # Cập nhật những dòng có tổng genres = 0 giá trị cột genre_unknow =
  movies['genre_unknown'].iloc[count_genres] = 1
  movies['year'] = movies['release_date'].apply(lambda x: str(x).spli
  movies['year'] = [str(year) if year != 'nan' else '1996' for year i
  # Lấy list các genre của mỗi movies
  all_genres = []
  for i in range(movies.shape[0]):
    dict_count = dict(movies[genre_cols].iloc[i])
    genre_row = [genre for genre in dict_count if dict_count[genre] !=
    all_genres.append(genre_row)
  movies['all_genres'] = all_genres
  return movies

users = _read_users()
ratings = _read_ratings()
movies = _read_movies()
```

Top

Thống kê và khảo sát dữ liệu

Phân phối điểm rating trung bình của mỗi bộ phim.

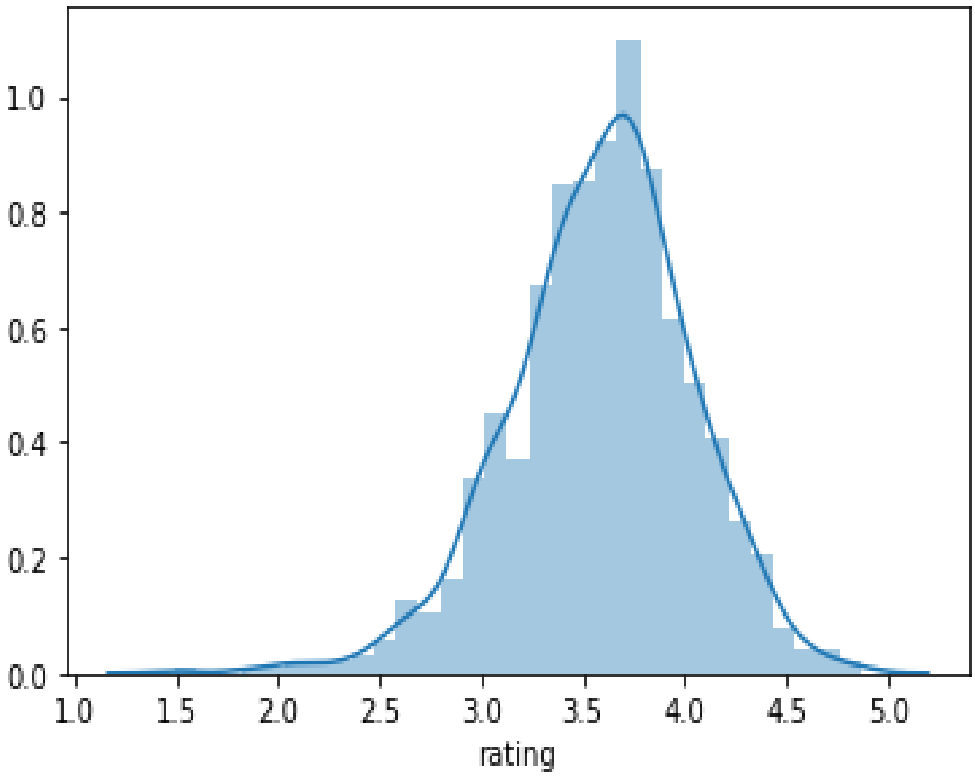
```
import seaborn as sns

avg_movie_rate = ratings.groupby('movie_id').rating.mean()
sns.distplot(avg_movie_rate)
```



Phân phối điểm rating trung bình của mỗi một người xem.

```
avg_user_rate = ratings.groupby('user_id').rating.mean()
sns.distplot(avg_user_rate)
```

Top

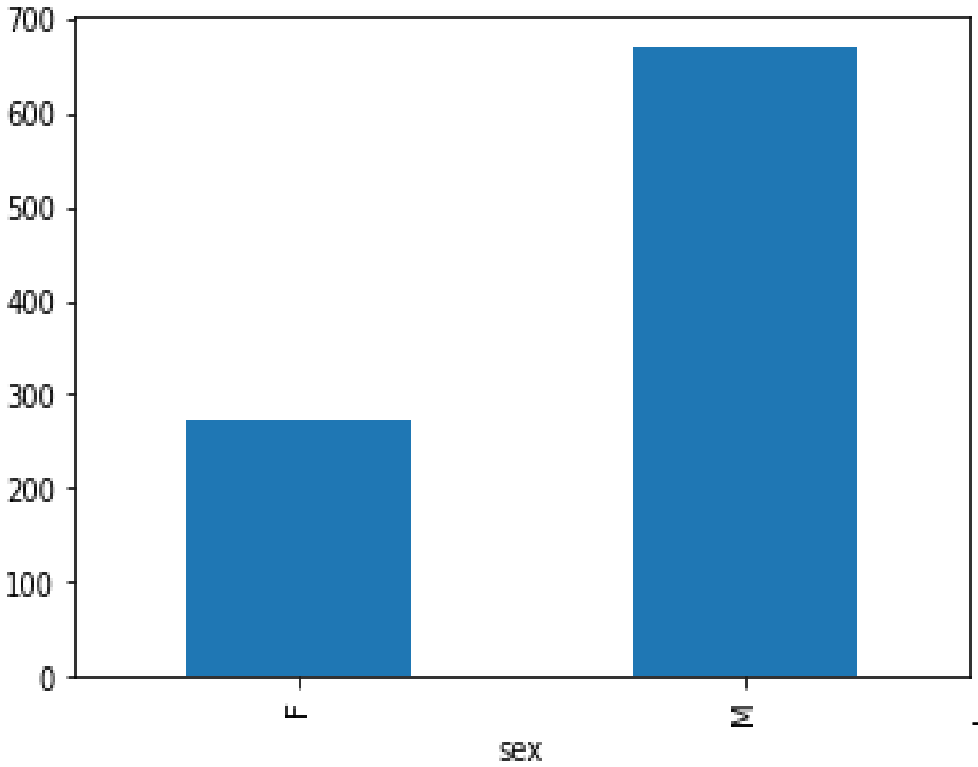


Nhận xét:

- Hầu hết các bộ phim có rating trung bình trong khoảng từ 3-4. Cũng có những bộ phim trung bình rate rất thấp (chỉ 1-2) và một số được rating khá cao (4-5).
- Các khách hàng cũng phân biệt thành khách hàng khó tính và dễ tính. Đối với khách hàng khó tính, điểm trung bình rate chỉ nằm trong khoảng từ 2-3 và khách hàng dễ tính là 4-5.

Số lượng người xem theo giới tính.

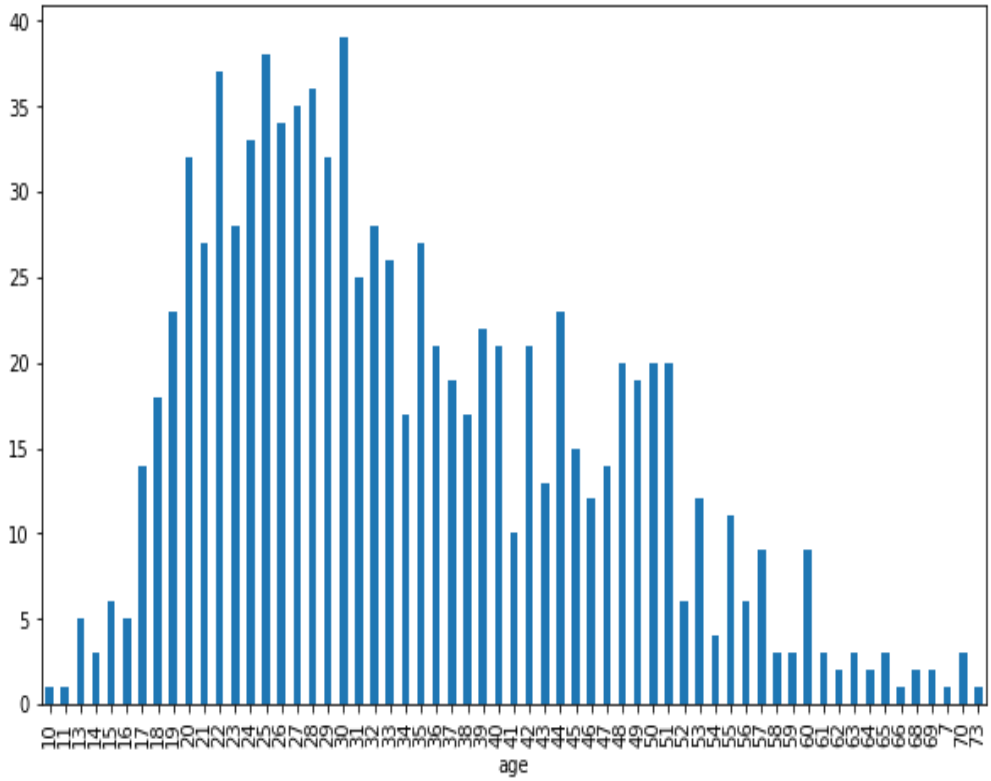
```
1    users.groupby('sex').user_id.count().plot.bar()
```



Top

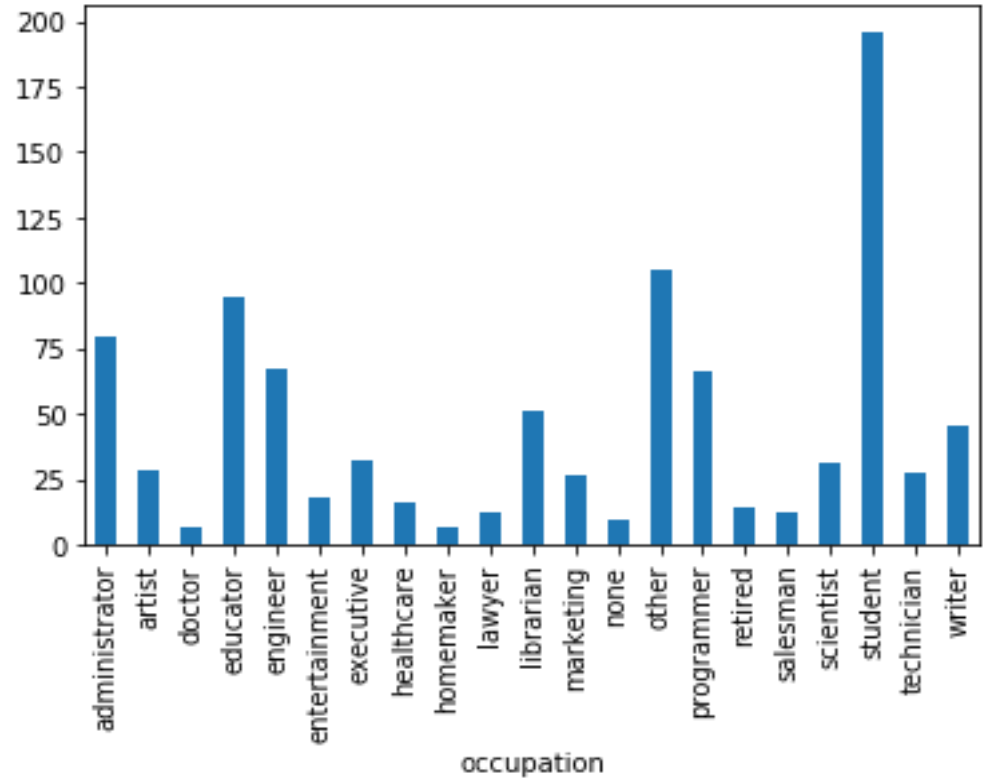
Số lượng người xem theo độ tuổi

```
1    users.groupby('age').user_id.count().plot.bar(figsize=(10, 6))
```



Số lượng người xem theo ngành nghề

```
1    users.groupby('occupation').user_id.count().plot.bar()
```



Nhận xét: Top

- Người xem là nam chiếm nhiều hơn nữ.

- Độ tuổi phổ biến của người xem là từ 20-30 tuổi.
- Ngành nghề `student, educator, administrator` có tỷ lệ người xem đông nhất. Điều này cho thấy không hẳn cứ là quản lý thì sẽ không có thời gian xem phim. Trái lại họ sắp xếp công việc rất hợp lý nên vẫn có những thời gian giải trí, phim ảnh.

Tiếp theo chúng ta sẽ khởi tạo một hàm phân chia dataframe để lựa chọn ngẫu nhiên số lượng người dùng theo một tỷ lệ cho trước.

```
# Hàm chia data thành tập train/test
def split_train_test(df, split_rate=0.2):
  """Chia dữ liệu dataframe thành tập train và test.
  Args:
    df: dataframe.t
    holdout_fraction: tỷ lệ số dòng của dataframe được sử dụng trong
  Returns:
    train: dataframe cho huấn luyện
    test: dataframe cho kiểm định
  """
  test = df.sample(frac=split_rate, replace=False)
  train = df[~df.index.isin(test.index)]
  return train, test
```

4.2. Khởi tạo batch

Ở bước này ta sẽ ta sẽ xây dựng một mạng neural MLP đơn giản nhằm dự báo khả năng một user có tương tác (bằng cách rate) một bộ phim hay chưa?

Mô hình sẽ nhận đầu vào x biểu diễn list các bộ phim mà một user đã rate, thông tin users, thông tin movies. Chúng ta có thể lấy ra các movie_id mà user đã rate thông qua hàm groupby bảng `ratings` theo `user_id`.

```
user_rates = ratings.groupby('user_id').movie_id.apply(list).reset_ind
user_rates.head()
```

	user_id	movie_id
0	1	[61, 189, 33, 160, 20, 202, 171, 265, 155, 117...
1	10	[16, 486, 175, 611, 7, 100, 461, 488, 285, 504...
2	100	[344, 354, 268, 321, 355, 750, 266, 288, 302, ...
3	101	[829, 304, 596, 222, 471, 405, 281, 252, 282, ...
4	102	[768, 823, 70, 515, 524, 322, 625, 161, 448, 4...

Để thuận tiện cho quá trình huấn luyện ta sẽ tạo ra hàm `data_gen_batch()` giúp khởi tạo các batch đầu vào. Mỗi một batch sẽ lấy ra các thông tin bên dưới:

Nhóm thông tin về movies:

- movie_id: Một tensor string các movie_id mà một users đã rate.
- genre: Thể loại của bộ phim.
- year: Năm sản xuất của bộ phim.

Top

Nhóm thông tin users:

- age: Tuổi của người xem.

- sex: Giới tính của người xem.
- occupation: Nghề nghiệp của người xem.

Top

```
import tensorflow as tf

# Dictionary của bộ phim và năm sản xuất
years_dict = {
    movie: year for movie, year in zip(movies["movie_id"], movies["ye
}

# Dictionary của bộ phim tương ứng với thể loại
genres_dict = {
    movie: genres
    for movie, genres in zip(movies["movie_id"], movies["all_genres"]
}

# Dictionary của user:
user_dict = {
    user: {'age': age, 'sex': sex, 'occupation': occupation}
    for user, age, sex, occupation in zip(users["user_id"], users["ag
}

def data_gen_batch(user_rates, batch_size):
  """Khởi tạo ra một batch input chính là các movie embedding
  Argument:
    user_rates: DataFrame lịch sử các bộ phim đã rate của user sao ch
    batch_size: Kích thước mẫu.
  """
  # Hàm pad để đưa các véc tơ category không có cùng kích thước của 1
  def pad(x, fill):
    return pd.DataFrame.from_dict(x).fillna(fill).values

  # list danh sách các movie_id mà user đã rate
  movie = []
  # year là năm sản suất của movie
  year = []
  # genre là thể loại phim
  genre = []
  # label là movie_id của bộ phim
  label = []
  # Các thông tin user:
  age = []
  sex = []
  occupation = []
  for i, (movie_ids, user_id) in user_rates[["movie_id", "user_id"]].
    # Khởi tạo các thông tin liên quan đến movie
    n_movies = len(movie_ids)
    movie.append(movie_ids)
    genre.append([x for movie_id in movie_ids for x in genres_dict[mo
    year.append([years_dict[movie_id] for movie_id in movie_ids])
    # Lưu ý phải trừ đi label 1 đơn vị vì output mạng neural thì movi
    label.append([int(movie_id)-1 for movie_id in movie_ids])
    # Khởi tạo các thông tin liên quan đến user
    age.append([user_dict[user_id]['age']]*n_movies)
    sex.append([user_dict[user_id]['sex']]*n_movies)
    occupation.append([user_dict[user_id]['occupation']]*n_movies)
  # Khởi tạo batch từ feature huấn luyện
  features = {
      'movie_id': pad(movie, ""),
      'genre': pad(genre, ""),
```

Top

```
        'year': pad(year, ""),
        'age': pad(age, ""),
        'sex': pad(sex, ""),
        'occupation': pad(occupation, ""),
        'label': pad(label, -1)
    }
    batch = (
    tf.data.Dataset.from_tensor_slices(features)
        .shuffle(1000)
        .repeat()
        # Khai báo kích thước batch
        .batch(batch_size)
        # Khởi tạo vòng lặp iterator qua từng batch
        .make_one_shot_iterator()
        .get_next())
    return batch

def select_random(x):
    """Lựa chọn một số phần tử ngẫu nhiên từ mỗi một dòng của x."""
    def to_float(x):
      return tf.cast(x, tf.float32)
    def to_int(x):
      return tf.cast(x, tf.int64)
    batch_size = tf.shape(x)[0]
    # Trả về khoảng range từ 0 --> batch_size
    rn = tf.range(batch_size)
    # Đếm số lượng giá trị >= 0 của dòng x
    nnz = to_float(tf.count_nonzero(x >= 0, axis=1))
    # Trả về một chuỗi random có kích thước là batch_size và giá trị mỗ
    rnd = tf.random_uniform([batch_size])
    # Stack row indexes and column indexes
    # Khởi tạo list các ids gồm (index_row, index_col).
    ids = tf.stack([to_int(rn), to_int(nnz * rnd)], axis=1)
    # Trích suất các index ngẫu nhiên từ x theo ids. Mỗi dòng sẽ lấy 1 
    return to_int(tf.gather_nd(x, ids))
```

4.3. Xây dựng hàm loss function

Ta đã biết mô hình softmax sẽ map các dữ liệu input x là các thông tin liên quan tới movie tới một user thông qua hàm số $f(x) \in \mathbb{R}^d$ với d là số chiều embedding. Khi đó $f(x)$ chính là véc tơ embedding của user.

Véc tơ này sau đó sẽ nhân với ma trận embedding của một bộ phim là ma trận $V \in \mathbb{R}^{m \times d}$ với m là số lượng các bộ phim. Như vậy đầu ra cuối cùng của mô hình sẽ là:

$$\hat{p}(x) = \text{softmax}(f(x)V^\top)$$

Với một nhãn mục tiêu y, nếu chúng ta kí hiệu $p = 1_y$ là véc tơ one-hot của quan sát mục tiêu mà giá trị bằng 1 tại vị trí index của bộ phim mà khách hàng có tương tác, các vị trí khác bằng 0. Khi đó loss function sẽ chính là hàm cross-entropy đo lường khác biệt phân phối $\hat{p}(x)$ với véc tơ one-hot p.

Tiếp theo chúng ta sẽ xây dựng một hàm số có đầu vào là các embedding véc tơ của user $f(x)$, ma trận nhúng của các movies V và biến mục tiêu là véc tơ one-hot y để tính toán loss function.

Top

Để tính toán `cross_entropy` chúng ta có thể sử dụng hàm `tf.nn.sparse_softmax_cross_entropy_with_logits` (https://www.tensorflow.org/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits). Hàm số này sẽ lấy đầu vào là giá trị $f(x)V^\top$, hay còn gọi là giá trị `logits` .

```
def loss_cross_entropy(user_embeddings, movie_embeddings, labels):
  """Trả về hàm cross-entropy loss function.
  Args:
    user_embeddings: một tensor shape [batch_size, embedding_dim].
    movie_embeddings: một tensor shape [num_movies, embedding_dim].
    labels: một sparse tensor có kích thước [batch_size, 1], tensor n
    Chẳng hạn labels[i] là movie_id của bộ phim được rate của mẫu i.
  Returns:
    mean cross-entropy loss.
  """
  # Số chiều của user_embeddings phải bằng với movie_embeddings
  # assert user_embeddings.shape[1].value == movie_embeddings.shape[1
  # Tính giá trị logits
  logits = tf.matmul(user_embeddings, movie_embeddings, transpose_b=T
  # Hàm loss function
  loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logit
      logits=logits, labels=labels))
  return loss
```

4.4. Khởi tạo RecNeuralNet class

RecNeuralNet là một class có tác dụng hỗ trợ huấn luyện mô hình theo phương pháp stochastic gradient descent. Hàm số sẽ nhận giá trị đầu vào là các biến input của mô hình, từ đó tính ra được hàm loss function (được đo lường qua hàm `loss_cross_entropy()` ở bước trên) và tìm cách tối thiểu hóa nó.

Các biến constructor khởi tạo của RecNeuralNet class gồm có:

- Ma trận embedding của user U: Là ma trận embedding của user mà mỗi dòng tương ứng với một véc tơ embedding user.
- Ma trận embedding của movie V: Là ma trận embedding của các bộ phim, mà mỗi dòng tương ứng với một bộ phim.
- loss function: Là giá trị loss function cần tối ưu hóa. Thường là hàm `cross_entropy` .
- metrics: Một dictionary của các metrics, mà mỗi một metric được mapping đến một key là tên của metric. Những metrics này được đánh giá và vẽ biểu đồ trong suốt quá trình huấn luyện trên từng epochs.

Sau khi huấn luyện, chúng ta có thể truy cập vào các phần từ của user embedding và movie embedding bằng cách sử dụng `model.embeddings` dictionary.

Chẳng hạn như sau:

```
# Truy cập user_embedding
model.embeddings['user_id']
# Truy cập movie_embedding
model.embeddings['movie_id']
```

Top

Bên dưới là class `RecNeuralNet()` hỗ trợ khởi tạo loss function và huấn luyện mô hình.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import collections

# Khởi tạo class RecNeuralNet hỗ trợ huấn luyện mô hình
class RecNeuralNet(object):
  """Một class biểu diễn mô hình collaborative filtering model"""
  def __init__(self, embedding_vars, loss, metrics=None):
    """Khởi tạo RecNeuralNet.
    Args:
      embedding_vars: Từ điển của các biến đầu vào. Định dạng tf.Vari
      loss: Hàm loss function cần được tối ưu hóa. Định dạng float Te
      metrics: Một list các dictionaries của các tensors. Các metrics
      trong suốt quá trình huấn luyện.
    """
    self._embedding_vars = embedding_vars
    self._loss = loss
    self._metrics = metrics
    # Khởi tạo none embeddings dictionary
    self._embeddings = {k: None for k in embedding_vars}
    self._session = None

  @property
  def embeddings(self):
    """trả về embeddings."""
    return self._embeddings

  def train(self, num_iterations=100, learning_rate=1.0, plot_results=
            optimizer=tf.train.GradientDescentOptimizer):
    """Huấn luyện model.
    Args:
      iterations: số lượng iterations huấn luyện.
      learning_rate: learning rate optimizer.
      plot_results: có vẽ biểu đồ sau khi huấn luyện xong mô hình hay
      optimizer: object optimizer được sử dụng để huấn luyện mô hình.
    Returns:
      Dictionary là các metrics được đánh giá tại vòng lặp cuối cùng.
    """
    # Khởi tạo các train_operation là quá trình tối thiểu hóa hàm los
    with self._loss.graph.as_default():
      opt = optimizer(learning_rate)
      train_op = opt.minimize(self._loss)
      local_init_op = tf.group(
          tf.variables_initializer(opt.variables()),
          tf.local_variables_initializer())
      if self._session is None:
        self._session = tf.Session()
        with self._session.as_default():
          self._session.run(tf.global_variables_initializer())
          self._session.run(tf.tables_initializer())
          tf.train.start_queue_runners()

    # Huấn luyện mô hình trong session
    with self._session.as_default():
      # Kích hoạt khởi tạo các biến local
      local_init_op.run()
      iterations = []
```

Top

```
58          metrics = self._metrics or ({},)
59          metrics_vals = [collections.defaultdict(list) for _ in self._me
60
61          # Huấn luyện mô hình và append results
62          for i in range(num_iterations + 1):
63            _, results = self._session.run((train_op, metrics))
64            # In ra kết quả các metrics sau mỗi 10 vòng lặp.
65            if (i % 10 == 0) or i == num_iterations:
66              print("\r iteration %d: " % i + ", ".join(
67                    ["%s=%f" % (k, v) for r in results for k, v in r.item
68                    end='')
69              iterations.append(i)
70
71              for metric_val, result in zip(metrics_vals, results):
72                for k, v in result.items():
73                  metric_val[k].append(v)
74
75          # Lưu các giá trị user_embedding, movie_embedding vào embedding
76          for k, v in self._embedding_vars.items():
77            self._embeddings[k] = v.eval()
78          # Vẽ biểu đồ các metrics được đo lường.
79          if plot_results:
80            num_subplots = len(metrics)+1
81            fig = plt.figure()
82            fig.set_size_inches(num_subplots*10, 8)
83            for i, metric_vals in enumerate(metrics_vals):
84              ax = fig.add_subplot(1, num_subplots, i+1)
85              for k, v in metric_vals.items():
86                ax.plot(iterations, v, label=k)
87              ax.set_xlim([1, num_iterations])
88              ax.legend()
89          return results
```

4.5. Xây dựng softmax recommendation neural network model

Mô hình sẽ có input bao gồm các biến liên quan đến bộ phim mà khách hàng đã rate đó là:

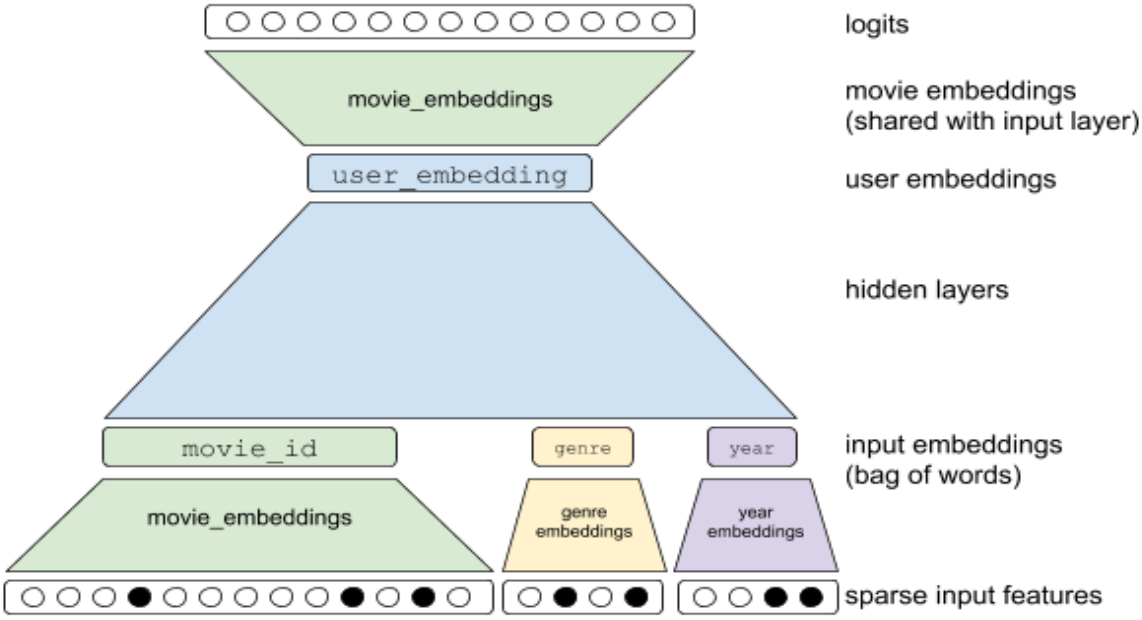
- movie_id: List các bộ phim mà khách hàng đã rate.
- genre: Thể loại tương ứng với list movie_id ở trên.
- year: Năm sản xuất của các bộ phim tương ứng với list movie_id ở trên.

Những biến đầu vào trên sẽ được concatenate lại thành một tensor input và tiếp theo chúng ta sẽ có một hidden layer có số units bằng đối số `hidden_dims` trong hàm `init_softmax_model()` bên dưới để map véc tơ input sang véc tơ nhúng của user là `user_embedding` . Cuối cùng từ `user_embedding` ta sẽ đi qua một layer softmax để tính toán phân phối xác suất ở output.

Mỗi một user có thể từng rating nhiều sản phẩm khác nhau. Tuy nhiên chúng ta sẽ chỉ lựa chọn ngẫu nhiên một nhãn mục tiêu là `movie_id` nằm trong số các bộ phim mà user đã rate trong 1 quan sát huấn luyện. Điều này để đảm bảo chắc chắn rằng ground truth p là một véc tơ one-hot. Hàm `select_random()` sẽ có tác dụng giúp ta thực hiện việc này dễ dàng.

Bạn đọc có thể dễ hình dung hơn những giải thích phía trên thông qua kiến trúc mô hình Softmax recommendation neural network.

Top



Hình 3: Kiến trúc mô hình softmax recommendation neural network.

Top

```
def init_softmax_model(rated_movies, embedding_cols, hidden_dims):
  """Xây dựng mô hình Softmax cho MovieLens.
  Args:
    rated_movies: DataFrame chứa list các bộ phim mà user đã rate.
    embedding_cols: là list các dictionary sao cho mỗi một dictionary
    list các dictionaries này sẽ được sử dụng trong tf.feature_column
    mapping sparse input features --> input embeddings như hình vẽ ki
    hidden_dims: list các kích thước của lần lượt các hidden layers.
  Returns:
    một RecNeuralNet object.
  """
  def user_embedding_network(features):
    """Maps input features dictionary với user embeddings.
    Args:
      features: một dictionary của input dạng string tensors.
    Returns:
      outputs: một tensor shape [batch_size, embedding_dim].
    """
    # Khởi tạo một bag-of-words embedding cho mỗi sparse features thô
    # Ở đây ta sử dụng một thủ thuật nhúng các biến category thành cá
    inputs = tf.feature_column.input_layer(features, embedding_cols)
    # Hidden layers.
    input_dim = inputs.shape[1].value
    for i, output_dim in enumerate(hidden_dims):
      w = tf.get_variable(
          "hidden%d_w_" % i, shape=[input_dim, output_dim],
          initializer=tf.truncated_normal_initializer(
              stddev=1./np.sqrt(output_dim))) / 10.
      outputs = tf.matmul(inputs, w)
      input_dim = output_dim
      inputs = outputs
    return outputs

  train_rated_movies, test_rated_movies = split_train_test(rated_movi
  # Khởi tạo các movie embedding trên train/test
  train_batch = data_gen_batch(train_rated_movies, 128)
  test_batch = data_gen_batch(test_rated_movies, 64)

  # Khởi tạo variable_scope để tái sử dụng lại các biến của nó
  with tf.variable_scope("model", reuse=False):
    # Tạo véc tơ nhúng user embedding
    train_user_embeddings = user_embedding_network(train_batch)
    # Lấy ngẫu nhiên label từ batch size
    train_labels = select_random(train_batch["label"])
  with tf.variable_scope("model", reuse=True):
    # Tạo véc tơ nhúng user embedding
    test_user_embeddings = user_embedding_network(test_batch)
    # Lấy ngẫu nhiên label từ test batch
    test_labels = select_random(test_batch["label"])
    # Trích xuất các movie_embeddings là layer cuối cùng của mạng neu
    # Lấy variable movie_embeddings có tên như bên dưới hoặc tạo movi
    movie_embeddings = tf.get_variable(
        "input_layer/movie_id_embedding/embedding_weights")

  # Xây dựng các hàm loss function trên train và test từ các đầu vào
  train_loss = loss_cross_entropy(train_user_embeddings, movie_embedd
  test_loss = loss_cross_entropy(test_user_embeddings, movie_embeddin
```

Top

```
    # Kiểm tra độ chính xác trên tập test chỉ tại k class có xác suất d
    _, test_precision_at_10 = tf.metrics.precision_at_k(
        labels = test_labels,
        predictions = tf.matmul(test_user_embeddings, movie_embeddings,
        k=10
    )

    metrics = (
        {"train_loss": train_loss, "test_loss": test_loss},
        {"test_precision_at_10": test_precision_at_10}
    )
    embeddings = {"movie_id": movie_embeddings,
                  "user_id": train_user_embeddings}
    return RecNeuralNet(embeddings, train_loss, metrics)
```

4.6. Huấn luyện model softmax

Bây h chúng ta sẽ cùng huấn luyện mô hình softmax bằng cách thiết lập các tham số cho mô hình như:

- Tham số huấn luyện learning_rate.
- Số lượt huấn luyện number of iterations.
- Kích thước chiều embedding của cả user và story.
- Số lượng hidden layer và kích thước units của mỗi layers.

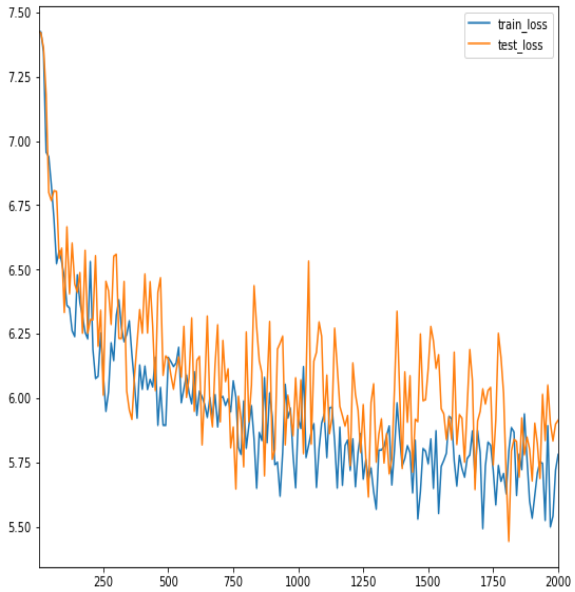
Lưu ý: Bởi vì đầu vào của chúng ta là những giá trị category (movie_id, genre, year) nên phải map chúng vào các id số nguyên. Chúng ta có thể thực hiện bằng cách sử dụng hàm `tf.feature_column.categorical_column_with_vocabulary_list` (https://www.tensorflow.org/api_docs/python/tf/feature_column/categorical_column_with_vocabulary_list), nó sẽ khởi tạo một list từ điển cho mỗi một feature. Sau đó mỗi một id sẽ được map vào một embedding véc tơ bằng cách sử dụng `tf.feature_column.embedding_column` (https://www.tensorflow.org/api_docs/python/tf/feature_column/embedding_column).

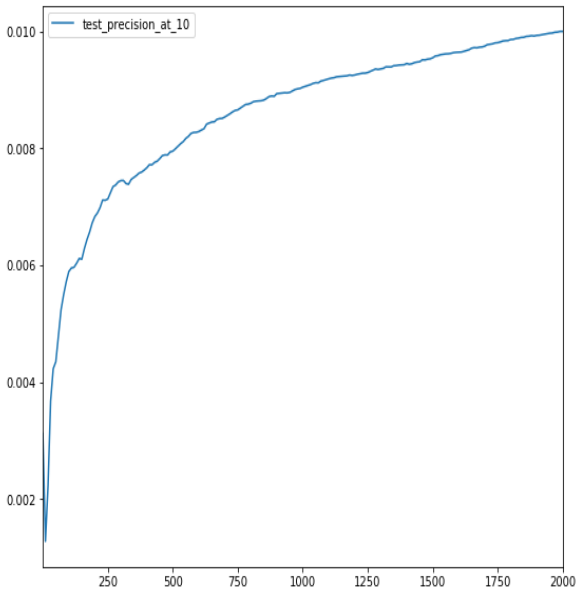
Top

```
# Khởi tạo hàm số giúp embedding các category sang véc tơ embedding.
def _embedding_category_col(key, embedding_dim, table_name = 'movies'
  """
  key: tên cột dạng category cần khởi tạo vocabulary, tên cột phải tr
  embedding_dim: số chiều dữ liệu được sử dụng để nhúng các categoric
  """
  if table_name == 'movies':
    # Khởi tạo vocabulary cho các giá trị của column key.
    if key == 'genre':
      list_gens = list(set([gen for movie_gens in movies['all_genres'
      categorical_col = tf.feature_column.categorical_column_with_voc
        key=key, vocabulary_list=list_gens, num_oov_buckets=0)
    else:
      categorical_col = tf.feature_column.categorical_column_with_voc
          key=key, vocabulary_list=list(set(movies[key].values)), num_
  elif table_name == 'users':
    categorical_col = tf.feature_column.categorical_column_with_vocab
        key=key, vocabulary_list=list(set(users[key].values)), num_oo
  # Trả về embedding véc tơ từ dictionary.
  return tf.feature_column.embedding_column(
      categorical_column=categorical_col, dimension=embedding_dim,
      combiner='mean')

with tf.Graph().as_default():
  softmax_model = init_softmax_model(
      user_rates,
      embedding_cols=[
          _embedding_category_col("movie_id", 35),
          _embedding_category_col("genre", 3),
          _embedding_category_col("year", 2),
          _embedding_category_col("age", 3, "users"),
          _embedding_category_col("sex", 2, "users"),
          _embedding_category_col("occupation", 3, "users")
      ],
      hidden_dims=[35])

softmax_model.train(learning_rate=5., num_iterations=2000, optimizer=
```

```
iteration 2000: train_loss=5.780039, test_loss=5.913449, test_precisi
({'test_loss': 5.9134493, 'train_loss': 5.780039},
 {'test_precision_at_10': 0.010001249375312344})
```

Top





Ta nhận thấy giá trị của hàm loss function giảm dần qua các step huấn luyện trên đồng thời cả tập train và test. Đồng thời tỷ lệ precision của nhãn được đo lường trên chỉ 10 nhãn dự báo có xác suất lớn nhất cũng tăng dần trên tập test.

Hàm `_embedding_category_col()` sẽ lần lượt thực hiện các bước sau:

- Khởi tạo một vocabulary (từ điển) cho một column dạng category bằng hàm `tf.feature_column.categorical_column_with_vocabulary_list()`. Khi đó vocabulary sẽ là một dictionary dạng `{'cate_value': index}`. Trong đó `cate_value` là giá trị của nhóm và index là số nguyên mapping tương ứng của nó.
- Tiếp theo map mỗi một giá trị index vào véc tơ one-hot có số chiều bằng với kích thước của `vocabulary`. Để điều chỉnh lại độ dài output, chúng ta truyền one-hot véc tơ qua một layer dense có số units bằng `embedding_dim`. Đầu ra sẽ là một véc tơ embedding có số chiều là `embedding_dim`. Tất cả quá trình này được thực hiện thông qua hàm: `tf.feature_column.embedding_column()`

Như vậy thông qua hàm `_embedding_category_col()` ta có thể map một trường đầu vào dạng category sang một không gian véc tơ với số chiều `embedding_dim` tùy ý.

5. Sử dụng mô hình để recommendation

5.1. Tìm kiếm sản phẩm tương đồng.

Để đo lương mức độ tương đồng giữa 2 sản phẩm ta có thể tính toán các chỉ số `cosine similarity` hoặc `dot product` giữa các véc tơ nhúng của chúng. Véc tơ embedding được trích xuất từ chính các cột của ma trận hệ số của layer cuối cùng mạng neural network. Bên dưới lần lượt là các công thức dot product và cosine similarity giữa 2 véc tơ $\mathbf{v}_i$ và $\mathbf{v}_j$.

- Dot product:

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{m=1}^{k} v_{im} v_{mj}$$

Top

- Cosine similarity:

$$\text{cosine\_similarity}(\mathbf{v}_i, \mathbf{v}_j) = \frac{\langle \mathbf{v}_i, \mathbf{v}_j \rangle}{\|\mathbf{v}_i\| \|\mathbf{v}_j\|}$$

Hàm `_compute_similarity()` sẽ có tác dụng tính ra chỉ số tương quan giữa một sản phẩm với toàn bộ các sản phẩm thông qua véc tơ nhúng đại diện cho sản phẩm đó (đối số `query_embedding`) với ma trận nhúng gồm toàn bộ các sản phẩm (đối số `item_embeddings`) theo công thức `dot product` hoặc `cosine similarity`.

```
DOT = 'dot'
COSINE = 'cosine'
def _compute_similarity(query_embedding, item_embeddings, measure=DOT
  """Tính điểm số giữa câu query và item embedding.
  Args:
    query_embedding: là một vector nhúng của query kích thước [k].
    item_embeddings: là ma trận nhúng véc tơ các items kích thước [N,
    measure: là một chuỗi string xác định kiểu đo lường tương đương đ
  Returns:
    scores: một véc tơ kích thước [N], sao cho score[i] là điểm của i
  """
  u = query_embedding
  V = item_embeddings
  if measure == COSINE:
    V = V / np.linalg.norm(V, axis=1, keepdims=True)
    u = u / np.linalg.norm(u)
  scores = u.dot(V.T)
  return scores
```

Tiếp theo dựa trên hàm `_compute_similarity()`, ta có thể xây dựng một hàm tìm kiếm những sản phẩm gồm thông tin về thể loại, tiêu đề, movie_id tương đồng nhất với một sản phẩm bất kì.

```
def _movie_similarity(model, movie_id, measure=DOT, k=10):
  # Lọc ra title của bộ phim
  titles = movies[movies['movie_id']==str(movie_id)]['title'].values
  print('movie title: {}'.format(titles))
  if len(titles) == 0:
    raise ValueError("Found no movies with movie_id %s" % movie_id)
  print("Nearest neighbors of : %s." % titles[0])
  scores = compute_scores(
      model.embeddings["movie_id"][movie_id], model.embeddings["movie_
      measure)
  score_key = measure + ' score'
  df = pd.DataFrame({
      score_key: list(scores),
      'titles': movies['title'],
      'genres': movies['all_genres']
  })
  return df.sort_values([score_key], ascending=False).head(k)
```

Thử nghiệm chút nào. Hãy cùng tìm ra top 10 các bộ phim liên quan nhất đến bộ phim kinh dị `Blood For Dracula (Andy Warhol's Dracula) (1974)` có `movie_id=666`. Đây rõ ràng là một bộ phim về ma cà rồng đáng sợ.

Top

```
_movie_similarity(softmax_model, movie_id=666, measure=COSINE, k=20)
```

```
movie title: ["Blood For Dracula (Andy Warhol's Dracula) (1974)"]
Nearest neighbors of : Blood For Dracula (Andy Warhol's Dracula) (1974
```

	cosine score	titles	genres
666	1.000000	Audrey Rose (1977)	[Horror]
444	0.920122	Body Snatcher, The (1945)	[Horror]
564	0.900972	Village of the Damned (1995)	[Horror, Thriller]
668	0.884781	Body Parts (1991)	[Horror]
853	0.872900	Bad Taste (1987)	[Comedy, Horror]
1272	0.864216	Color of Night (1994)	[Drama, Thriller]
634	0.850382	Fog, The (1980)	[Horror]
859	0.845901	Believers, The (1987)	[Horror, Thriller]
439	0.836006	Amityville II: The Possession (1982)	[Horror]
589	0.835799	Hellraiser: Bloodline (1996)	[Action, Horror, Sci-Fi]
674	0.805313	Nosferatu (Nosferatu, eine Symphonie des Graue...	[Horror]
636	0.802819	Howling, The (1981)	[Comedy, Horror]
572	0.784273	Body Snatchers (1993)	[Horror, Sci-Fi, Thriller]
858	0.774721	April Fool's Day (1986)	[Comedy, Horror]
443	0.765144	Blob, The (1958)	[Horror, Sci-Fi]
852	0.761631	Braindead (1992)	[Comedy, Horror]
799	0.761129	In the Mouth of Madness (1995)	[Horror, Thriller]
440	0.754666	Amityville Horror, The (1979)	[Horror]
757	0.747806	Lawnmower Man 2: Beyond Cyberspace (1996)	[Sci-Fi, Thriller]
772	0.728285	Mute Witness (1994)	[Thriller]

Ta nhận thấy toàn bộ các bộ phim liên quan nhất đều thuộc về thể loại `Horror` hoặc `Thriller` và cũng đáng sợ không kém. Điều này cho thấy thuật toán đã giúp tìm ra các sản phẩm tương đồng khá chính xác.

5.2. Phân cụm các sản phẩm

Để nhận biết thuật toán recommendation có tính cá nhân hóa cao hay không, ta sẽ giảm chiều dữ liệu của những véc tơ nhúng xuống không gian 2 chiều và tìm cách biểu diễn chúng theo các nhóm thể loại phim. Nếu các points phân bố tập trung theo cụm tương ứng với từng thể loại thì chứng tỏ thuật toán giúp nhận biết khá tốt tính chất khác biệt giữa các bộ phim. Để giảm chiều dữ liệu từ 35 chiều xuống 2 chiều ta sử dụng thuật toán TNSE (https://distill.pub/2016/misread-tsne/).

Top

```
from sklearn.manifold import TSNE
import time

X = softmax_model.embeddings['movie_id']

time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
X2D = tsne.fit_transform(X)

print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 1682 samples in 0.003s...
[t-SNE] Computed neighbors for 1682 samples in 0.239s...
[t-SNE] Computed conditional probabilities for sample 1000 / 1682
[t-SNE] Computed conditional probabilities for sample 1682 / 1682
[t-SNE] Mean sigma: 0.575077
[t-SNE] KL divergence after 250 iterations with early exaggeration: 68
[t-SNE] KL divergence after 300 iterations: 1.419350
t-SNE done! Time elapsed: 5.158601522445679 seconds
```

Khi đó dữ liệu của các bộ phim đã được giảm về còn 2 chiều là biến `X2D` . Tiếp theo ta sẽ xác định nhãn thể loại cho từng bộ phim. Do nhãn là duy nhất nên đối với những bộ phim được gán nhiều thể loại thì ta sẽ lấy ra ngẫu nhiên 1 làm đại diện.

```
# Lựa chọn ngẫu nhiên một thể loại phim trong trường hợp thể loại phim
y = [np.random.choice(genres) for genres in movies['all_genres']]
print('y.shape: {}'.format(len(y)))
print('X2D.shape: {}'.format(X2D.shape))
```

```
y.shape: 1682
X2D.shape: (1682, 2)
```

Vẽ biểu đồ scatter biểu diễn tọa độ 2 chiều của các bộ phim theo các nhóm thể loại.

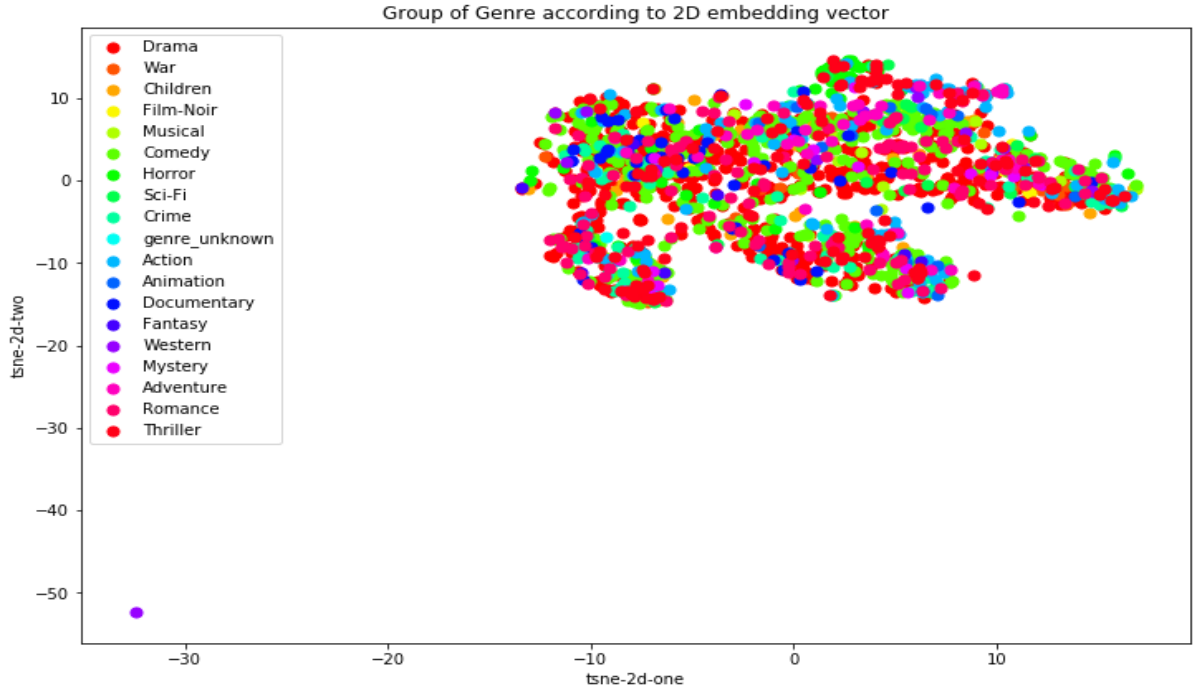
Đầu tiên ta sẽ khởi tạo ngẫu nhiên các mã màu sắc bằng cách sử dụng bằng hàm `get_cmap()`

```
genres = list(set(y))
N = len(genres)

def get_cmap(n, name='hsv'):
    '''Trả về một hàm số map mỗi index trong khoảng từ 0, 1, ..., n-1
    Giá trị của đối số name phải trùng với tên colormap của mã màu chu
    return plt.cm.get_cmap(name, n)
cmap = get_cmap(N)
```

Sau đó vẽ đồ thị trong không gian 2 chiều

Top

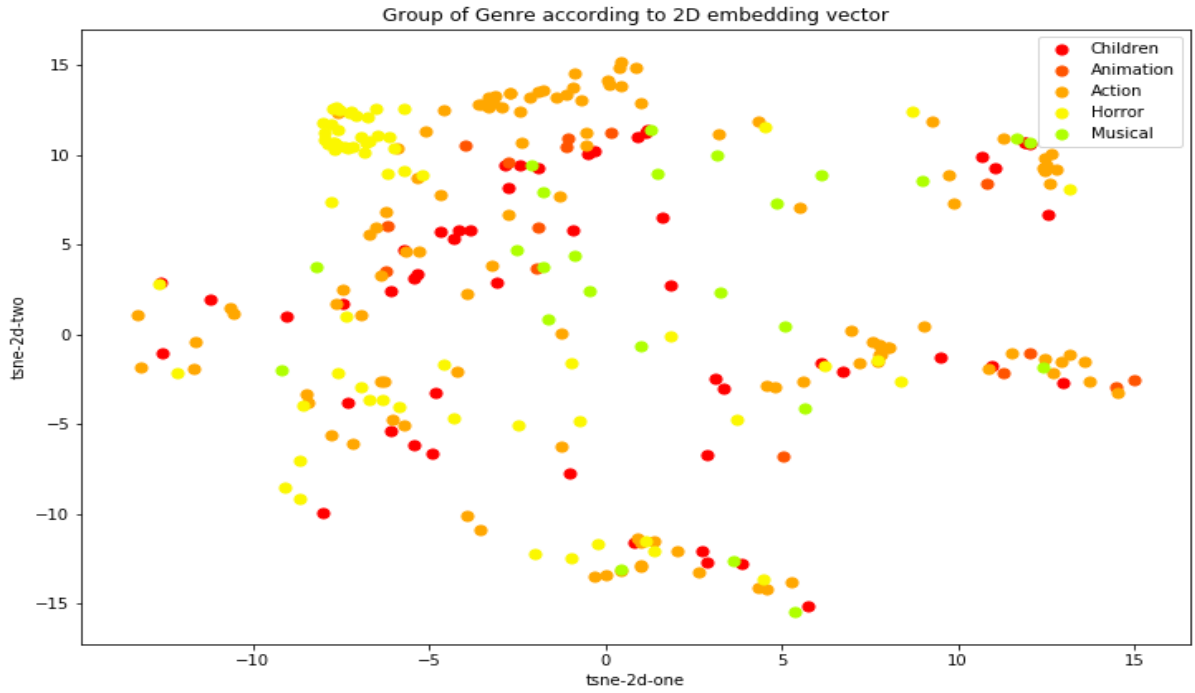
```
# Hàm lọc các index thuộc về một thể loại phim genre
def _filter_genre_id(y, genre):
  return [True if item == genre else False for item in y]

# Vẽ biểu đồ
plt.figure(figsize = (12, 8))
for i, genre in enumerate(genres):
  ids = _filter_genre_id(y, genre)
  plt.scatter(X2D[ids, 0], X2D[ids, 1], s=50, c=cmap(i), label=genre)
plt.title('Group of Genre according to 2D embedding vector')
plt.xlabel('tsne-2d-one')
plt.ylabel('tsne-2d-two')
plt.legend()
plt.show()
```



Visualize chỉ riêng một số thể loại phim gồm `['Children', 'Animation', 'Action', 'Horror', 'Musical']`.

```
genres=['Children', 'Animation', 'Action', 'Horror', 'Musical']

plt.figure(figsize = (12, 8))
for i, genre in enumerate(genres):
  ids = _filter_genre_id(y, genre)
  plt.scatter(X2D[ids, 0], X2D[ids, 1], s=50, c=cmap(i), label=genre)
plt.title('Group of Genre according to 2D embedding vector')
plt.xlabel('tsne-2d-one')
plt.ylabel('tsne-2d-two')
plt.legend()
plt.show()
```

Top



Ta nhận thấy đối với một số loại hình phim có số lượng quan sát nhỏ, thể loại phim đặc thù và kén khách thì phân bố của chúng khá tập trung như: `['Children', 'Animation', 'Action', 'Horror', 'Musical']`. Điều này cho thấy thuật toán có hiệu quả trong việc phân tích sự khác biệt giữa các thể loại phim đặc thù.

Một số cụm của cùng một thể loại phim bị tách rời có thể là do rất nhiều các bộ phim thuộc về ít nhất từ 2 thể loại trở lên nên dẫn tới véc tơ nhúng của chúng chứa đặc trưng của vài loại hình phim. Tùy vào việc ta lựa chọn ngẫu nhiên nhãn thể loại của chúng mà biểu diễn của point tương ứng sẽ thuộc về một cụm tách biệt với các cụm khác trong cùng một thể loại phim.

6. Tổng kết

Trên đây tôi đã giới thiệu đến các bạn một trong những ứng dụng cơ bản nhất của recommendation sử dụng deep learning đó là xây dựng một mạng neural network đơn giản gồm toàn bộ các fully connected layer. Kiến trúc này có thể tận dụng được đồng thời rất nhiều các thông tin cả về phía user, sản phẩm và lịch sử tương tác giữa user và sản phẩm. Do đó thông tin chúng học được để biểu diễn các véc tơ user và items có tính cá nhân hóa cao. Khi sử dụng thuật toán để khuyến nghị tìm kiếm các sản phẩm tương đồng sẽ cho kết quả chuẩn xác hơn. Ngoài ra quá trình xây dựng và triển khai thuật toán sẽ gặp phải những khó khăn nhất định về dữ liệu, huấn luyện, dự báo mà ta cần phải khắc phục, đặc biệt là đối với những hệ thống lớn lên tới vài triệu người dùng.

Ngoài phương pháp nêu trên chúng ta còn có thể sử dụng thêm rất nhiều các phương pháp khác trong deep learning. Các bạn có thể xem ở phần giới thiệu `các phương pháp recommendation` tại Bài 15 - collaborative và content-based filtering (https://phamdinhkhanh.github.io/2019/11/04/Recommendation_Compound_Part1.html). Có thể ở những bài sau tôi sẽ tiếp tục giới thiệu một số phương pháp deep learning khác nữa nếu có đủ thời gian viết lách. Hi vọng rằng các bạn có thể áp dụng bài viết này để tự xây dựng các thuật toán recommendation cho doanh nghiệp hoặc website của bạn.

Bài viết có sử dụng rất nhiều các tài liệu tham khảo được liệt kê mục 7.

7. Tài liệu tham khảo.

Top

1. Negative Sampling - Distributed Representations of Words and Phrases and their Compositionality - Tomas Mikolov. etc (https://arxiv.org/pdf/1310.4546.pdf)
2. applying word2vec to recommenders and advertising - Chris McCormick (http://mccormickml.com/2018/06/15/applying-word2vec-to-recommenders-and-advertising/)
3. Deep Learning based Recommender System: A Survey and New Perspectives - shuai zhang. etc (https://arxiv.org/pdf/1707.07435.pdf)
4. Approxiamting softmax technical - Sebastian Ruder (https://ruder.io/word-embeddings-softmax/)
5. Negative sampling - Jason Tam (https://tech.hbc.com/2018-03-23-negative-sampling-in-numpy.html)
6. recommendation - google course (https://developers.google.com/machine-learning/recommendation/content-based/basics)

Top