Learning a 1-dimensional Mapping from State Transitions Data

Huichao Xue
huichao@linkedin.com
LinkedIn Corporation
Sunnyvale, CA, USA

Chao Wang
cowang@linkedin.com
LinkedIn Corporation
Sunnyvale, CA, USA

Xiaoqing Wang
xiaoqwang@linkedin.com
LinkedIn Corporation
Sunnyvale, CA, USA

Yan Zhang
yanzhang@linkedin.com
LinkedIn Corporation
Sunnyvale, CA, USA

ABSTRACT

Transition data exists in many domains, where people spend some time and effort in one state, and then move to the next. For example, career professionals move to their next positions after working at their previous positions for years; learners move to the next courses after finishing some other courses. To help people compare those states, and visualize their "levels of advancement", we propose a data-mining technique to project each state into a 1-dimensional real value. As an example, we apply our approach to mining seniority from career transition data collected from linkedin.com. Experimental results show that our model is able to learn the relative seniorities of positions within the same company, and also the seniorities of positions from different companies.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; • **Information systems** → *Web mining*.

KEYWORDS

dimensionality reduction, embeddings, career transitions, data mining

ACM Reference Format:
Huichao Xue, Chao Wang, Xiaoqing Wang, and Yan Zhang. 2022. Learning a 1-dimensional Mapping from State Transitions Data. In *Compjobs@WSDM '2022: The First International Workshop on Computational Jobs Marketplace, February 25, 2022, Virtual*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A transition is the process of changing from one state to another, such as changing from one job position (company + title) to another, either within the same company or in a different one. A common question is how to compare or order those states. For example (1) how hard is it to become a "senior software engineer" at company A comparing to a "staff software engineer" at company B? (2) is course C likely a prerequisite of course D?

Mathematically, one approach to answering such a question is to project each state (e.g., a company-title pair) into a one-dimensional value indicating the level of advancement. Intuitively, later-stage states (e.g., being senior staff engineers) should be projected into higher values than earlier-stage states (e.g. being entry level software engineers). Defining such a projection typically requires manual labeling. For example, levels.fyi[1] relies on crowd-sourcing to build such mappings for title comparisons across different companies [11]. However, due to the sparsity of manual labels, such approaches are subject to both noise and incomprehensiveness.

In this paper, we propose a data driven approach to automatically learn such mappings, by leveraging the large amount of existing data online. For example, linkedIn.com contains ≈800M members' working experiences consisting of job positions, and each generally includes the company, the job title, and the start and end time. Also coursera.com has ≈80M members' course learning history. By leveraging such data, we plan to build a machine learning model that automatically infers the level of advancement for each state, without the need for human interventions.

Specifically, we propose a statistical model that captures the positive trend and variance in state progression *velocities*. Intuitively, given a transition, for example, a person spent 3 years to transit from state A to state B, we can estimate the average velocity of the transition to be $\frac{B-A}{3}$. By assuming a Gaussian distribution for the velocities, and conducting a Maximum Likelihood Estimation (MLE), we can use transition data records to adjust the mapped value for each state, and hence learn a sensible mapping.

We claim two contributions. First, we propose a technique that processes millions of transition records and produces a one-dimensional mapping between states and real values, preserving the level of advancement for each state. Second, as an example, we apply the approach to learning job position mappings, using 0.1M transition records between seven

Compjobs@WSDM '2022, February 25, 2022, Virtual

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://levels.fyi

selected companies on linkedin.com. Our results show that (1) within the same companies, our model is able to associate seniorities to different positions in the correct order; (2) the associated seniorities can be also used to compare positions across different companies. We also discuss the inaccuracies of the model.

2 BACKGROUND

Formally speaking, let $s \in S$ represent a state (e.g., Software Engineer at Google). Our goal is to learn a reasonable mapping $f(s) \rightarrow \mathbb{R}$ from states to real values. The observed transitions in our work are in the format of $r = (s_1, t, s_2)$, indicating it takes time t for a person to transition from state s_1 to s_2. Though in general we need to learn the mapping function f, in practice we only need the mapping for a finite set of states. Therefore, f can be represented by a vector, where each dimension corresponds to the progression for a different job position.

One straightforward heuristic to summarize the level of advancements required for each position is to compute the average year-of-experience (YOE) required to get to a state. For example, in career transitions, if people have an average of 8 years of experience to get to s_1, but 6 years to get to s_2, we would conclude that s_1 is more advanced than s_2 . The drawback of this approach is it fails to account for the variance in the career growth velocity among the population. For example, if s_2 is a hot position in a new booming field (e.g. blockchain), then entering s_2 gives an edge to one's career advancement, but entering such fields does not necessarily require a long work experience.

Learning the mapping from data points into real values resembles learning embeddings in many machine learning applications. The one-dimensional level-of-advancement measure can be seen as a special case of 1-dimension embedding. However, due to the differences in the learning goals, our scenario needs a customized setup.

The idea of embedding had been used in many approaches including Natural Language Processing [10, 13], Recommendation Systems [1, 2, 4], and in modeling career transitions [6, 9]. The fundamental idea is to represent items (e.g. words, or documents, job positions) into a low-dimensional vector, so that their dot-products form a low-rank factorization of a high-dimensional probability matrix [5]. The goal in most of the previous work is to preserve the probability distributions (e.g. the probability of the next word being "the", or the next position being a Software Engineer at Google) as much as possible. Our scenario, in comparison, is less concerned about predicting probabilities. To see the difference, we will take career transition as an example.

(1) Low distributional similarity, but high similarity in levels of advancement. Consider Google's Software Engineer and Facebook's Software Engineer, though being highly similar in levels of career advancement, they may often transit into very different positions (Google to Google, Facebook to Facebook).

(2) High distributional similarity, but low similarity in levels of advancement. Consider Software Engineer Interns and Software Engineers within the same company. Although people from these two positions often transit into similar other positions (e.g. software engineer in a different company), we would hope them to be mapped into far apart level-of-advancement values.

Our approach is directly inspired by visualization techniques which try to summarize vast amounts of data by projecting them into a low-dimensional (typically 2 or 3 dimensional) space. Such approaches include linear transformations such as PCA [3], and non-linear transformations including ISOMAP [14] and t-SNE [15]. These approaches represent data items as nodes, compute the distances (or similarities) between them, and then try to learn a low-dimensional projection to preserve the distances. This technique naturally translates into a graph-representation of the transition data, where

(1) States can be represented as nodes
(2) Each transition record $r = (s_1, t, s_2)$ can be seen as an edge starting from node s_1 to s_2, with a weight t

However, our graph structure is different from the previous approaches in that: (1) multiple edges can exist between different nodes (2) the graph is asymmetric (3) we learn a 1-dimensional embedding as opposed to multi-dimensional. Our approach applies a statistical model to aggregate the multiple edges between graph nodes, resulting in a 1-dimensional embedding.

3 MODEL

Our key modeling idea is to use state transitions to infer a 1-dimensional mapping for different states. Instead of training an embedding for downstream tasks [1, 2, 4, 10, 13], our 1-dimensional embedding is specifically aiming to capture the level of advancement for each state. This is achieved by designing a loss function to encourage a larger gap between embeddings only when we observe larger temporal gaps in transitions data. Intuitively, each state is represented as a dot on the 1-dimensional axis. Users' transitions can help us "connect the dots", and infer the relative positions of each state. If users typically spend less time transitioning from one position to another, then there should be a lower gap between their corresponding values, and vice versa. We use Figure 1 to illustrate such an example in career progression. We derive a mathematical model to capture this notion.

Suppose we have collected users' historical transition records. Each record is a tuple: $r = (s_1, t, s_2)$, indicating the user started position s_1 in the past, and then transitioned into position s_2 after time t (e.g., in years). Our learning goal is to infer a mapping f that maps each state s into a real number $f(s)$ which measures the *levels of advancement* of the state. Note that $f(s)$ is not observed in the training data, but is instead our learning goal.

With a given mapping f, we can estimate the user's progression velocity during the period as $v_r = \frac{f(s_2)-f(s_1)}{t}$. For different people, during different time periods, v is likely to

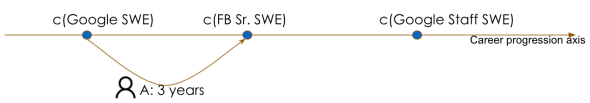

Figure 1: Representing position and transitions as nodes and edges on a one-dimensional axis. In this example, each job position (a title, company pair) is represented as a point on the career progression axis. Intuitively, more senior roles correspond to higher progression values. A user's career transition can be seen as an edge between points.

have variations. Our key modeling assumption is that users' progression velocities V are random variables with a Normal distribution $\mathcal{N}(1, \sigma^2)$ centered at 1:

(1) By assuming a normal distribution, we capture the variance of the velocities across the population
(2) By assuming a positive mean, we favor an overall positive trend in progressions over time.

The learning becomes a maximum likelihood estimation, where we try to adjust the positions of the dots, so it's more likely to observe what we see. We also show that this boils down to minimizing the squared error between the observed velocity and one.

Specifically, during MLE, we learn the mapping f with maximum likelihood estimation over all the records:

$$\arg\max_f \prod_r \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{v_r - 1}{\sigma}\right)^2\right\}$$

This optimization problem boils down to minimizing the squared error between the observed velocity and one:

$$\arg\min_f \sum_{r=(s_1,t,s_2)} \left(\frac{f(s_2) - f(s_1)}{t} - 1\right)^2$$

The above optimization can be carried out by vanilla gradient descent algorithms.

Note how this optimization is different from directly aggregating on years of experience. This allows us to capture phenomena such as the bars for entering different companies are different, even if the Years of Experience for entering every company is the same, which is always zero. One limitation of the Normal distribution assumption is that it does not distinguish between personalized factors such as the prestige of each company, or the current position. As a result, our learned model does not provide personalized levels of advancement scores, but rather a summary across the population. We leave finer grained analysis as future work.

4 EXPERIMENTAL SETUP

Our experiments aim to test whether the automatically learned progression mapping can truly capture the global trends. We particularly have conducted the experiment on users' career progressions. We try to learn a mapping from job positions (e.g. Facebook's software engineer) into real values, using users' career transition data (e.g. John entered Google as Software Engineer in 2008, then went to Facebook as Senior Software Engineer in 2011).

From	To	Duration
Google,SWE	Other,SWE	1.0
Other,SWE	Facebook,Senior SWE	2.0

Table 1: Training data example. Consider a user who joined Google as a software engineer in 2000, then Uber as a software engineer in 2001, then Facebook as a senior software engineer in 2003, we will have two data records shown above for this user.

Microsoft	94K
Amazon	55K
Google	40K
Facebook	14K
Apple	11K
Linkedin	9K
Netflix	2K
Other	274K

Table 2: Breakdown in the number of companies in our collected transition records. We use seven tech companies as separate identities, whereas all other companies occurring in the data are labeled as "Other". The "Other" class shows up in a large portion of the transition records, and therefore helps to align the learned level of advancement for other companies. Further distinguishing between the "Other" class can lead to more accurate results, which we leave as future work.

We narrowed down our data collection into software engineering positions among tech companies, including Facebook, Google, Apple, Amazon, Microsoft and LinkedIn. We collected users' experiences from their LinkedIn profiles as the training data. Because the vast majority of users work outside of our study's scope, we further filtered our training data by:

(1) Only considering users who had ≥ 2 positions within the companies we consider
(2) Only consider positions that have "software engineer" in the title.
(3) We use a special token "other" to represent companies not in the short list
(4) We only keep the time duration between two positions

We show an example of the training data in Table 1. Our training data contains 0.1M transition records. Statistics of the data are shown in Table 2 and 3.

We use the training data to help estimate the mapping of each position into a 1-dimensional space using the approach discussed in Section 3. This allows us to visualize different positions' corresponding progressions in a single diagram.

5 EXPERIMENTAL RESULT

We try to answer two research questions:

Software Engineer	185K
Senior Software Engineer	65K
Staff Software Engineer	8K
Senior Staff Software Engineer	2K
Other	57K

Table 3: Breakdown in the number of titles in our collected data. In general, less data is observed for more senior positions.

(1) Is our approach able to correctly infer the relative seniority of positions within the same companies?
(2) Is our approach able to compare positions across different companies?

We present the main result in Figure 2, where each company's job positions are visualized in the format similar to levels.fyi.

We make the following observations:

First, the model is able to correctly learn the career progressions for the majority of the companies. All the companies' entry-level → Senior → Staff progressions were learned without explicitly encoding this domain knowledge.

Second, the model also learns the differences in entering different companies at similar positions. For example, Google and Facebook are well-known to have high bars for hiring at different levels, which was also picked up by the model.

Third, the model also shows obvious mistakes in certain cases. One example is Facebook's Senior Staff Engineer position, which was learned to fall between senior engineer and staff engineer. One main reason it happens is the missing career transition signals for this position. On one hand, many engineers in this role still use "software engineer" in their LinkedIn title; on the other hand, many engineers will consolidate all the previous positions in the company, and only keeping the last position. These data points violate our modeling assumptions, and therefore produce noise during model training.

Aside from building visualizations (e.g. levels.fyi) to help users compare positions, we also foresee the trained models helping recommendation/search systems. For example, in job recommendation (e.g. linkedin.com/jobs), the mismatch between user' and jobs' levels of advancements can be a useful signal for ranking or filtering the results.

6 RELATED WORK

Learning latent representations of state is a common technique used in industry. These embeddings are usually multi-dimensional, either pre-trained in an unsupervised fashion or computed in an end-to-end fashion. However, since we want to visualize the advancement comparisons among all the states, multi-dimensional embedding can be used as a feature but not generated as the output. Our work is to generate 1-dimensional mapping from transition data.

One approach to modeling the transitions is using graph embedding. A general approach to preserve the asymmetric transitivity is discussed in [12], where the authors proposed a High-Order Proximity preserved Embedding algorithm to compute the embeddings of directed graphs. To adapt the algorithm to the problem of transitions with time involved, one can construct the weight in such a way that it also captures the time elapsed for the transition. For example, the weight might include a term proportional to the inverse of the number of years spent in one job position before switching to the next. A more related work which uses graph embedding to learn job transitions has been explored in [8]. In the work, each job position is a node, and each transition is a directed edge connecting two nodes with a weight representing the frequency of the transition, and then computes the graph embedding. Differently, our work considers the number of years taken to move from one job position to another which is a strong indicator of level of advancement. For example, if a person spent one year at company A, and then moved to company B. In comparison, if the same person had worked at company A for five years before moving to company C. It is generally safe to assume the position at company C is higher than the one at company B.

There are other approaches explored other than modeling the transitions as graphs. Both [9] and [6] leverage sequence modeling algorithms, and static features are used beyond transitions themselves. The major focuses of them are on a particular area of transitions: career transitions, and the main goals are to predict next job positions rather than comparing and ordering those job positions across different companies. [7] manually groups positions into buckets with numerical labels and uses regression to predict the numerical label. The main difference from our approach is that it uses features like word and topic, while our method learns a single value without using any features for the position. That is, our method only has one feature, the position id and the embedding is only 1 dimensional. Our goal is to learn a value to compare career positions, so personalization with more features are not needed, unless we want to infer a new position (cold start problem), which is not a big concern in our use case.

7 CONCLUSIONS

Learning to map positions into progressions can provide a high-level visualization of the field, also to help people make comparisons. We propose a data-driven approach to automatically infer such mappings from users' position transition histories. We have conducted a study to apply the approach to career transitions data from software engineers among seven tech companies. The result indicates the approach can correctly infer relative progressions for many different positions. In the meantime, we also found the approach to be sensitive to the noise in the data.

REFERENCES

[1] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

Figure 2: Learned career progressions from transitions between example companies. Each column corresponds to a different company, where rows are different positions within the company. The width along the horizontal axis does not associate with any meaning. The vertical axis corresponds to the learned progressions for each position, pointing downwards (to simulate the visualization at levels.fyi). Our proposed model only learns the career progression values for starting each position, whereas the ending points are determined by the start of the "next" position inferred by the model. Note the model is able to associate different levels' of software engineering positions with different starting values. Note that the model is not able to infer the difference between different companies' "top-seniorities" at the bottoms of each column. We cut off the bottom arbitrarily for illustration purposes.

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[3] Harold Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology* 24, 6 (1933), 417.

[4] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.

[5] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems* 27 (2014), 2177–2185.

[6] Liangyue Li, How Jing, Hanghang Tong, Jaewon Yang, Qi He, and Bee-Chung Chen. 2017. NEMO: Next Career Move Prediction with Contextual Embedding. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 505–513.

[7] Ye Liu, Luming Zhang, Liqiang Nie, Yan Yan, and David S. Rosenblum. 2016. Fortune Teller: Predicting Your Career Path. In *AAAI*.

[8] Haiyan Luo, Shichuan Ma, Anand Joseph Bernard Selvaraj, and Yu Sun. 2019. Learning job representation using directed graph embedding. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 1–5.

[9] Qingxin Meng, Hengshu Zhu, Keli Xiao, Le Zhang, and Hui Xiong. 2019. A Hierarchical Career-Path-Aware Neural Network for Job Mobility Prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 14–24.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[11] Zuhayeer Musa. [n. d.]. Building Levels.fyi in Coffee Shops and Growing to Profitability. https://www.indiehackers.com/interview/building-levels-fyi-in-coffee-shops-and-growing-to-profitability-da7a4f5d63. Accessed: 2021-12-16.

[12] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1105–1114.

[13] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[14] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[15] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html