Data Science and Engineering (2018) 3:232–247
https://doi.org/10.1007/s41019-018-0076-2

What's Next? A Recommendation System for Industrial Training

Rajiv Srivastava[1,2] · **Girish Keshav Palshikar**[1] · **Saheb Chaurasia**[1] · **Arati Dixit**[3]

Received: 16 January 2018 / Revised: 2 August 2018 / Accepted: 11 September 2018 / Published online: 25 September 2018

Abstract
Continuous training is crucial for creating and maintaining the right skill-profile for the industrial organization's workforce. There is a tremendous variety in the available trainings within an organization: technical, project management, quality, leadership, domain-specific, soft-skills, etc. Hence it is important to assist the employee in choosing the best trainings, which perfectly suits her background, project needs and career goals. In this paper, we focus on algorithms for training recommendation in an industrial setting. We formalize the problem of next training recommendation, taking into account the employee's training and work history. We present several new unsupervised sequence mining algorithms to mine the past trainings data from the organization for arriving at personalized next training recommendation. Using the real-life data about trainings of 118,587 employees over 5019 distinct trainings from a large multi-national IT organization, we show that these algorithms outperform several standard recommendation engine algorithms as well as those based on standard sequence mining algorithms.

Keywords Personalized recommendation · Sequence matching · Sequence mining · Industrial training

1 Introduction

In knowledge economy, the skills and expertise of the employees directly impact the effectiveness and productivity of the workforce. Training is important for creating and maintaining the right skill-profile for the organization's workforce. In an industrial setting, for each employee, the learning and skill development often happens through a series of *training units* (aka *courses* or *trainings*). There is a tremendous variety in the available trainings within an organization: technical, project management, quality, leadership, domain-specific (e.g. insurance, banking, retail), soft-skills and so forth. Many different delivery methods are used: web-based training (WBT), classroom trainings, external certifications, external trainings, MOOC courses, etc. Many trainings have overlapping contents, though they may differ in quality of instruction, depth of coverage, duration, difficulty levels, etc.

✉ Rajiv Srivastava
rajiv.srivastava@tcs.com

[1] 54B, Tata Research Design and Development Center, Hadapsar Industrial Estate, Hadapsar, Pune 411013, India

[2] Department of Technology, Savitribai Phule Pune University, Ganeshkhind Road, Aundh, Pune 411007, India

[3] Pune, India

Sometimes an employee is mandated to undergo specific trainings, particularly when she joins a new project, to enable her reach the required expertise levels in tasks associated with the role assigned to her in that project. At all times, the employee can also voluntarily choose the trainings that she would like to undergo. Given all the diversity of available trainings, it becomes important to assist the employee in choosing the best *sequence* of trainings, which perfectly suits her background, project needs and career goals. While that is a long-term goal for our research, in this paper, we deal with a smaller part: how to recommend the *next* best training to any employee, taking into account her specific training history and work history. The goodness of a training c recommended at time t for a given particular employee e is measured in terms of the probability $P_e(c)$ that the employee will actually undergo c within some fixed time period starting from t. Thus, the best training c^* to be recommended for an employee e is one for which this probability is maximum:

$$c^* = \underset{c}{\text{argmax}}\, P_e(c)$$

Our focus is on recommending the *next* training for any given employee; we are not concerned with long-term learning objectives of the employee or the organization. The idea is to build a recommendation system that individual employees can use; such a system should make use of the past training-related

data within the organization to continually improve the quality and effectiveness of its recommendation. Since employees of large modern organizations spent considerable time and efforts in learning and development (L&D), such a recommendation system would prove to be very useful.

Our specific contributions in this paper are as follows. We formalize the problem of next training recommendation, taking into account the employee's training and work history. We present several new unsupervised algorithms to mine the past trainings data from the organization for arriving at personalized next training recommendation. We show that these algorithms outperform several standard recommendation engine algorithms as well as those based on standard sequence mining algorithms.

The paper is organized as follows. Section 2 reviews related work. Section 3 describes the real-life dataset containing training records from a large multi-national IT company. Section 4 describes the first set of our algorithms mainly based on sequence matching. Section 5 describes some baseline algorithms. Section 6 contains our algorithms which make use of two aspects which are specific to our problem, viz., change of project to which an employee is allocated and the trainings done by team members already in a project. Section 7 describes some standard recommendation engine algorithms. Section 8 further refines the proposed algorithms by making use of the fact that some trainings are very similar to each other. Section 9 contains experimental results of the algorithms on the case-study dataset. In Sect. 10, an approach based on Markov decision process for recommending next training is discussed. Section 11 contains conclusions and pointers to some future work.

2 Related Work

Recommendation systems is a well-established research area [1, 4, 5], with a wide range of applications including e-commerce and e-health. Applications of data mining and recommendation systems to e-learning are also a well-researched area, with its own conferences. Much of the work done in this area is focused on academic learning recommendations, whereas we are focused on learning recommendations in an industrial setting.

The survey paper [20] points out that the traditional collaborative, content-based, knowledge-based and hybrid recommender systems dealing with users and items do not fit well due to additional complexities of technology-enhanced learning (TEL)—an observation that we empirically support in this paper. TEL recommenders incorporate additional information about learners, teachers and their context in the recommendation process. The knowledge level of the learner, academic grades, learning goals, learning style, etc., need to be considered for personalization. Within an enterprise, the learning systems offer freedom to the employees to choose a career path. An appropriate mix of the contextual information such as learning sequence similarity, technology and project context, identified by our approach, improve the recommendation performance for enterprise learning. Manouselis et al. [10] observes that the top-down, closed-corpus-based formal learning and intra-enterprise informal learning are different as the user tasks and goals are different. They summarize the usefulness of recommendation techniques such as collaborative filtering (CF) and data mining techniques such as k-nearest neighbour (kNN) or vector-based models such as singular value decomposition (SVD) or matrix factorization (MF). In this paper, we have used many of these techniques as baselines and observe that these techniques do not consider the sequentiality of trainings and relations such as prerequisites among trainings, which reduces their effectiveness in enterprise learning scenario. Shani et al. [15] first propose the use of MDP, along with tolerances to the optimal reward values and high reward initialization for new items, to balance exploitation–exploration for recommendation of an item-set. In [12], the authors suggest an upper confidence bound-based multi-armed bandit learning algorithm for recommending an item-set in online setting where items are correlated, specifically to the first item. The employed rewarding scheme for the multi-arm bandit provides an increased reward for the first bandit so that it recommends the *right* first item followed by a set of items which have inter-dependence. Such approaches for an item-set recommendation differ from our problem as we recommend only one item which is the *next* training.

In [9], the authors provide for way-finding process among Moodle learning objects using personalized recommender systems (PRSs). The personalized recommendations are based on learning goals, learning activities and learner profiles. The next best learning activity is recommended by using a top-down formal knowledge model of activities, competences and the learner profiles along with informal bottom-up approach based on folksonomies and tagging data from social activity. In contrast to their model-based approach, which is difficult to define and measure in large organizations, we mine relevant patterns from a repository of learning data. [14] report a recommendation system for supporting students to better choose how many and which courses to enrol on, using the learning history of the students with similar achievements. The system suggests positive and negative recommendations to the students using a decision tree developed over attributes such as courses enroled in, name of the course, accumulative GPA, grade.

The authors in [16] propose individualized treatment effects (ITE) as a method to provide personalized feedback to students by using data obtained from evaluation of the pedagogical approaches and interventions as well as by identifying and characterizing at-risk students. The ITE quantifies the effectiveness of intervention and/or instructional

regimes for a particular student based on institutional student information and performance data. The random-forest-based regression and classification methods are used to identify factors for student success, characterize the benefits of a supplemental instruction section, and suggest intervention initiatives for at-risk groups in the course. In [19], the authors apply MF and CF to predict the rating or success/failure of a student for a given item or problem in a subject area which perform better than the linear regression or logistic regression methods. In this work, we show that the training recommendation using the mining of sequential patterns based on important domain knowledge elements performs better than MF and kNN user-based CF.

In [6], the authors observe, as part of their review of educational data mining (EDM) and learning analytics area, algorithms such as association rule mining, sequential pattern mining are used for group formation for successful academic project execution, finding patterns in concept maps associated with overall learning, etc. The authors argue the need of identifying appropriate methods for suitable applications in aiding learning in academic set-up. Intelligent tutoring systems form another rich source of related research work; however, not all such systems include a component for personalized training recommendation. Zhu et al. [21] describe one such system for school-based personalized education system (PeRES) for learners as well as instructors. The system comprises of software-based multi-agents for static and dynamic users modelling, learning plan generation and adjustment, personalized content search, personalized recommendation, as well as real-time evaluation of the learning progress. The personalized recommendation agent is driven by learners' interests and instructors' guidance. They use data of learners' activities such as page browsing, chatting, which invades privacy in the context of an enterprise. The learner's interests are also not explicitly available in an enterprise.

3 Case-Study Dataset

We have trainings-related past data from a business unit within a large multi-national IT services organization. We have retained only *technical* trainings related to IT domain, software packages from business domains and project management. We have excluded trainings from core business domains (insurance, telecom, etc.) and soft-skills. We have removed trainings related to soft-skills and those about business domains (insurance, retail, telecom, etc.), unless the trainings were about IT systems or software packages in those business domains. The dataset has 118,587 distinct training sequences (one for a particular employee), containing at least two trainings. There are 5019 unique trainings (unigrams), each of which is present in at least one sequence, and which occur a total of 934,167 times, giving an average of 186.13 occurrences per training. Table 1 shows the top 10 most frequent trainings. There are 192,168 unique bigrams (ordered pairs of consecutive trainings) in the data, which occur a total of 815,580 times, giving an average of 4.24 occurrences per bigram; Table 2 shows top 5 bigrams. The summary statistics for the sequence lengths (i.e. number of trainings per employee) is: average = 7.89 STDEV = 8.196, Q1:3, median = 5, Q3:10. There are about 2836 (2.39%) *outlier* sequences of unusually long lengths (> 32), which we have not removed. Figure 1 shows the histogram for sequence lengths—most employees, being junior, have shorter training sequences.

Table 1 Top 10 most frequent trainings

ID	Title	Count
PD0250	Critical thinking strategies simulation	21, 873
00073762	Software testing fundamentals	16, 965
78920_ENG	Introduction to UNIX	14, 251
COMM0332	Leading effective business meetings	12, 058
80533_ENG	Working with UNIX files and directories	11, 189
83855_ENG	Unix shell scripting basics	9304
00076833	Testing throughout the software life cycle	8301
31643_ENG	Java 2 language basics	7365
73491_ENG	Developing JSP	6890
00011300	Project life cycles and stakeholders	6080

Table 2 Top 5 most frequent trainings pairs (bigrams)

Title	Count
Introduction to UNIX:: Working with UNIX files and directories	4256
Project life cycles and stakeholders:: Project planning	3906
Project planning:: Project integration: executing and completing a project	3547
Working with UNIX files and directories:: Introduction to UNIX	2971
Project integration: executing and completing a project:: Project human resources management simulation	2893

Many trainings belong to a specific technology area. We manually identified 45 *Technology Groups* (or just *Groups*), and their characteristic keyphrases. For example, keyphrases for group JAVA included: `Java, JSP, "Core Java", Struts, J2EE, "Java Web"`. A simple algorithm then matched the training's title with this list of keyphrases and decided the appropriate technology group for it. For example, the training having the title `Core Java, GUI and Rich Clients` is automatically tagged to the group JAVA. We had a few simple conflict resolution rules when

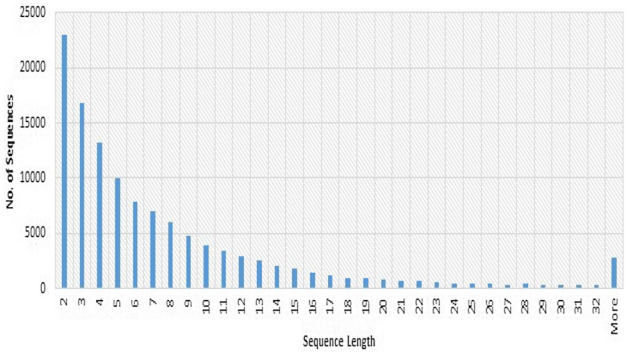

Fig. 1 Histogram for sequence lengths

keywords from multiple groups were present in a title. Table 3 shows top 10 technology groups, both in terms of number of trainings and number of training instances (i.e. *popularity*).

Let $C = \{c_1, c_2, \ldots, c_M\}$ denote the set of all available training ($M = 5019$ in our dataset). Each *training sequence* (i.e. the sequence of trainings done by an employee) is a finite ordered sequence of elements from C and will be denoted $\sigma = \langle a_1, a_2, \ldots \rangle$, where each $a_i \in C$. We use $|\sigma|$ to denote the length, i.e. the number of elements in σ. We assume that all elements of σ are distinct (i.e. an employee does a specific training at most once). Let $D = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ denote the set of all N training sequences, where σ_i is the training sequence of i-th employee; we have $|\sigma_i| \geq 2$ for each $1 \leq i \leq N$ and $N = 118{,}687$ in our dataset. Let $G = \{g_1, g_2, \ldots, g_L\}$ denote the set of all technology groups ($L = 45$ in our dataset). The notation $c_j.TG$ denotes the technology group associated with a particular training $c_j \in C$. A training in C may have multiple *occurrences* (or *instances*) among the training sequences in D. In addition to training sequences, we have the date on which each training in a given sequence was completed, along with the passing grade or marks (if applicable). Most employees are software engineers and work on software development and maintenance projects. We know the project to which an employee is allocated, along with her designated *role* in the project, when she completes a particular training. We know the technologies that each project uses, which is a non-empty subset of G. The role corresponds to the tasks that the employee is expected to handle in a project; e.g. *Developer, Designer, Tester, Analyst*.

4 Sequence Matching

Suppose a particular employee has the (temporally ordered) sequence of trainings $\sigma = \langle c_1, c_2, \ldots, c_m \rangle$. Our goal is to identify K trainings to be recommended to this employee. If the actual next training completed by this employee c_{m+1} (once it becomes known) is one of these K predicted (recommended) trainings, then we have a *hit*; else, we have a *miss*, i.e. we have made an error in prediction. Conceptually the simplest approaches for suggesting next training are based on sequence matching. Ignoring employee work history altogether, we view the past data as a set of sequences of trainings, each sequence corresponding to the training history of a particular employee. Given a *query sequence* $\sigma = \langle c_1, c_2, \ldots, c_m \rangle$, we propose several simple methods for predicting K trainings. Table 4 shows a small example dataset of sequences for $N = 15$ employees over the set of $M = 8$ trainings $\{a, B, c, d, e, f, G, H\}$. We will use the employee e having the sequence $\sigma_e = \langle a\ B\ c \rangle$ to illustrate how various algorithms recommend a next training for e.

- **Next-Frequent**: Recommend the K most frequent trainings that have appeared immediately after c_m in the training data. That is, find all bigrams of the form $\langle c_m, x \rangle$ in the training data, order them in terms of their frequencies and select the second elements from the top K bigrams.
- **NF-Lookback**(b): Recommend the K most frequent trainings that have appeared after the sequence

Table 3 Top 10 technology groups in terms of #available trainings and #trainings completed

Title	#Unique trainings	Title	#Training instances
MS_Platform_Architecture	952	Java	142, 557
Telecom	380	Assurance	86, 549
Oracle_DB	343	MS_Platform_Architecture	75, 892
.NET	299	.NET	71, 617
IBM_SW_Platform_Architecture	289	Programming_Languages	71, 023
SAP	249	Oracle_DB	62, 425
Oracle_Business_App	243	Software_Engg	57, 289
Java	190	Others	55, 109
BIPM_Solution_Architecture	181	OS_Unix	47, 997
Web_Technologies	163	Support_Services	30, 607

Table 4 Example sequences for 15 employees over $M = 8$ trainings

$\langle f\ e\ H\ B\ G\rangle$	$\langle f\ d\rangle$
$\langle a\ B\ c\ d\ e\ f\ G\ H\rangle$	$\langle d\ f\ B\rangle$
$\langle G\ c\ d\ H\rangle$	$\langle f\ d\ c\rangle$
$\langle c\ d\ H\ f\rangle$	$\langle c\ d\ G\rangle$
$\langle B\ H\ d\ e\ G\rangle$	$\langle B\ c\ a\ d\ e\rangle$
$\langle e\ B\ a\ H\ d\ f\rangle$	$\langle a\ c\ B\ d\ f\ G\ H\rangle$
$\langle a\ B\ c\ d\ e\ G\ f\ H\rangle$	$\langle e\ B\ c\ d\ f\rangle$
$\langle a\ B\ c\ e\ G\ f\rangle$	

$\langle c_{m-b+1}, \ldots, c_m\rangle$ in the training data; **Next-Frequent** is equivalent to setting $b = 1$. That is, find all $b+1$-grams of the form $\langle c_{m-b+1}, \ldots, c_m, x\rangle$ in the training data, order them in terms of their frequencies and select the last elements from the top K of $b+1$-grams.

- **NF-Lookback-U**(b): Recommend the union of $\lfloor K/b\rfloor$ most frequent trainings computed using **Next-Frequent** and $\lceil K/b\rceil$ most frequent trainings computed using **NF-Lookback(b)**.
- **NF-Lookback-SkipTG-U**: Same as **NF-Lookback-U** with $b = 2$, except that when forming the trigrams starting with $\langle c_{m-1}\ c_m$, instead of c_{m-1}, use that earlier training in σ which has a technology group different from that of c_m. If all trainings in σ have the same technology group, then use c_{m-1}. The intuition is that if an employee's training history sequence contains trainings of two (or more) technology groups, say JAVA and ORACLE, then recommend half the trainings from JAVA group and half from ORACLE group.
- **NF-Lookback-MatchTG**: Same as **NF-Lookback-SkipTG-U**, except that use the trigrams of the trainings having the same technology group as c_m (ignores all trainings in σ that have a different technology group than that of c_m).
- **NF-Lookback-BiMax**: Ignore all trainings in σ that have a different technology group than that of c_m, find the frequencies of occurrences of bigrams $\langle x, c_m\rangle$ where x is any training from 1 to c_{m-1} having the same technology group as that of c_m, and if $\langle x, c_m\rangle$ is the most frequent among these bigrams, then recommend the K most frequent trainings that have appeared after this bigram.
- **NF-Lookback-BiMax-U**: Recommend the union of $\lceil K/2\rceil$ trainings based on c_m and $\lfloor K/2\rfloor$ trainings based on c_{m-1}, computed using **NF-Lookback-BiMax**.

Using the example dataset in Table 4, and the test employee having the sequence of trainings $\langle a\ B\ c\rangle$, the first training recommended by the algorithm **Next-Frequent** is d, as among the bigrams starting with c, the bigrams $\langle c\ d\rangle$ has the highest count (6). The same training d is the top recommendation for this employee using **NF-Lookback(2)**, since the trigram $\langle B\ c\ d\rangle$ has the highest count (3) among those with c in the middle. Top 2 recommendations of **NF-Lookback-U(2)** are d and e, because the trigrams $\langle B\ c\ d\rangle$, $\langle B\ c\ e\rangle$ are the most frequent and bigrams $\langle c\ d\rangle$ and $\langle c\ e\rangle$ are the most frequent, so that their union leads to d and e.

Table 5 Top 10 outgoing edges from the node JAVA

Java	0.6631
Programming_Languages	0.0526
Oracle_Database	0.0492
Assurance	0.0485
Web_Technologies	0.0417
Others	0.0190
Software_Engineering	0.0143
OS_Unix	0.0112
BIPM_Solution_Architecture	0.0095
Databases	0.0088

Given a training bigram $\langle a\ b\rangle$, there is a considerable probability that a and b may have different technology groups: out of the total 815,580 bigrams, 419,738 (51.46%) show a technology group change. A *TG transition model* is a directed edge-labelled graph $G = (V, E, \lambda)$, where the vertices are technology groups, there is a directed edge from A to B labelled with a positive real number $0 < \lambda(A, B) < 1$ if a training of technology group A has the probability $\lambda(A, B)$ of being immediately followed by some training from technology group B. Self-loops correspond to no change in the technology group in a training bigram. In our data, the top 10 outgoing edges from the vertex for technology group JAVA are shown in Table 5. The edge probability is estimated by the fraction of the total number of edges outgoing from the vertex JAVA; e.g. out of 131,685 bigrams in which the first training is from technology group JAVA, in 87,322 (66.31%) bigrams, the second training is also from the technology group JAVA (i.e. the self-loop on the vertex JAVA has probability 0.6631).

Using this TG transition model, we can define another algorithm for predicting the next training.

- **NF-TG-Transition**: Recommend $\lceil K/2\rceil$ trainings using **Next-Frequent**. Let A denote the technology group for c_m. Let $B_1, \ldots, B_{\lfloor K/2\rfloor}$ be the $\lfloor K/2\rfloor$ most frequent out-edges from the vertex A in the TG transition model. For each $B_i, 1 \le i \le \lfloor K/2\rfloor$, select that training x_i such that $\langle c_m\ x_i\rangle$ is the most frequent bigram among the bigrams in which the second training is of technology group B_i. The remaining $\lfloor K/2\rfloor$ recommended trainings are $\{x_1, \ldots, x_{\lfloor K/2\rfloor}\}$.

5 Sequence Mining

The next approach that we adapted for predicting next training is based on sequence mining. The input data for a sequence mining algorithm typically consists of a set of *elements*, where each element is a sequence of sets. For example, in a market basket setting, each element corresponds to the sequence of purchases of a customer, where each purchase is an item-set. In our data, each training sequence can be considered as a sequence of *singleton* sets (i.e. we assume that the trainings of an employee are essentially sequential and she undergoes at most one training at a time). Suppose a sequence mining algorithm outputs a set of frequent sequences from a given set of training sequences (technically, each frequent sequence is a sequence of singleton sets). We assume that there are no repeated elements in any given sequence.

We say a sequence ρ is *very strictly right-aligned* (*VS-right-aligned*) with another sequence σ, denoted $\rho \dashv \sigma$, if ρ is a subsequence of σ, and the last element of both sequences is same. For example, $\langle a\ d\ g\rangle$ is VS-right-aligned with $\sigma = \langle a\ b\ c\ d\ e\ f\ g\rangle$. Note, the relation VS-right-aligned is not symmetric; i.e. $\rho \dashv \sigma$ does not necessarily imply $\sigma \dashv \rho$. This notion is very strict because of the following reasons. In our example, $\langle a\ x\ d\ g\rangle$ is not VS-right-aligned with σ because the training x is not present before g in σ. As another example, $\langle d\ a\ g\rangle$ is not VS-right-aligned with σ because the order of trainings d and a is not matching with that in σ.

Next, given a positive integer k, a *k-right-chop* of a sequence ρ, denoted $\rho_{(-k)}$, is the sequence obtained by removing k elements from the right end of ρ; the removed k element subsequence of ρ is called the *residue*. For example, for $\rho = \langle a\ b\ c\ d\ e\ f\ g\rangle$, $\rho_{(-1)} = \langle a\ b\ c\ d\ e\ f\rangle$ is the 1-right-chop, with residue $\langle g\rangle$. The 0-right-chop ($k = 0$) of a sequence ρ is ρ itself.

We say that a sequence ρ is *VS-feasible* for another sequence σ, if there exists a positive integer $k > 0$ such that the k-chop of ρ is VS-right-aligned with σ. For example, both $\langle b\ g\ p\ q\rangle$ and $\langle b\ d\ g\ h\rangle$ are VS-feasible for $\sigma = \langle a\ b\ c\ d\ e\ f\ g\rangle$. Note that $\langle b\ d\ g\rangle$ is *not* VS-feasible for this σ. We now discuss the idea of using frequent sequences mined from the given dataset D to predict the next training for any particular employee.

Let F denote the set of frequent sequences (at a fixed support s) obtained by a sequence mining algorithm from the given dataset D of training sequences. Let σ denote the training sequence of a particular employee. Let K denote the number of trainings we need to recommend for the employee corresponding to σ (e.g. $K = 5$).

The algorithm **VS_Feasible_Seq** works as follows. Let $F' = \{\rho_1, \rho_2, \ldots, \rho_m\} \subseteq F$ denote the subset of m frequent sequences such that each ρ_i is VS-feasible for the given σ. We output a set S of the union of first elements of the K residues of the most frequent sequences in F', where each residue has a minimum support in the data. If F' contains fewer than K sequences, then we continue to add second elements from the residues. For example, suppose $K = 5$, $\sigma = \langle a\ b\ c\ d\ e\ f\ g\rangle$ and $F' = \{\langle a\ b\ g\ h\ i\rangle, \langle b\ d\ g\ j\ w\rangle, \langle b\ c\ g\ r\ s\ t\rangle\}$. Then $S = \{h, j, r\}$, which has fewer than $K = 5$ elements. So we add second elements from the residues, keeping most frequent ones as required, and the final output is $S = \{h, j, r, i, w\}$ (assuming w is more frequent than s). If the actual next training done by this employee (after g) is in S then we have a hit; else we have a miss.

We can similarly define when a sequence ρ is *strictly right-aligned* (*S-right-aligned*) with another sequence σ. For example, $\langle x\ a\ y\ d\ z\ g\rangle$ is S-right-aligned with σ, even though it contains other elements (x, y, z) that do not occur in σ before g. But note that the elements in ρ that do occur in σ must form a subsequence of σ. Thus, $\langle x\ d\ y\ a\ z\ g\rangle$ is not S-right-aligned with σ, because $\langle d\ a\rangle$ is not a subsequence of σ. We say that a sequence ρ is *S-feasible* for another sequence σ, if there exists a positive integer $k > 0$ such that the k-chop of ρ is S-right-aligned with σ. In our example, $\langle x\ b\ y\ g\ p\ q\rangle$ is S-feasible for σ.

The algorithm **S_Feasible_Seq** works as follows. Let $F' = \{\rho_1, \rho_2, \ldots, \rho_m\} \subseteq F$ denote the subset of m frequent sequences such that each ρ_i is S-feasible for the given σ. We output a set S of the union of first elements of the K residues of the most frequent sequences in F'. If F' contains fewer than K sequences, then we continue to add second elements from the residues, just as in the algorithm **VS_Feasible_Seq**.

6 Effects of Project Change

As mentioned earlier, the trainings of an employee are both project dependent as well as driven by her own interest. The training sequence is interleaved with a Boolean flag indicating a project change, which we indicate by a semicolon. For example, the sequence $\langle a\ b\ ;\ c\ d\ e\ ;\ ;\ f\rangle$ indicates that this employee did two trainings while on first project, 3 on second project, 0 on third project and 1 in the last project. Different projects often are on different technology platforms, and hence when an employee joins another project, her trainings may change to reflect the new skill requirements. In this example, a and b may have the same technology group, whereas c, d and e may belong to a different technology group more aligned with the new project.

We tested the hypothesis whether change of project has a statistically significant impact on the technology group of the next training that an employee does, using the χ^2-test as follows. Consider a typical bigram $\langle a\ b\rangle$. Either there is

a project change between a and b or there is not; similarly, either a and b have the same technology group or not. The contingency table is shown in Table 6. The value of the χ^2 statistic is 12,103, which is too large, at degree of freedom = 1; hence, the null hypothesis is rejected and we conclude that project change and technology group change are *not* independent. The odds ratio is 3.07845 > 1, which gives the same conclusion.

This shows that we need a prediction algorithm that makes use of the project change information. Suppose the employee e for whom the prediction is to be made has the training sequence σ such that the last training is, say g, and the technology group of g is say *JAVA*. Now there are two possibilities: either the employee has changed the project after doing g, or she is in the same project (at the time when the prediction is to be made). Suppose we know the ground truth (whether there is a project change after g or not). Moreover, if she has joined a new project, we also assume we know the technology groups being used in the new project. We modify each of the above algorithms to predict the next training for e by making use of this additional knowledge. We illustrate the process by describing in detail for a couple of algorithms.

We first define the notion of a lenient bigram. A usual bigram $\langle a\ b\rangle$ occurs in data whenever b occurs immediately after a in some sequence. Assuming a and b belong to different technology groups, a *lenient bigram* allows other elements to come in between a and b, as long as they all belong to the same technology group as a. Thus $\langle a\ i\ j\ b\rangle$ is still counted as the bigram $\langle a\ b\rangle$, if both i and j have the same technology group as a. The notion of lenient trigram is defined similarly. Suppose trainings p and q belong to two different technology groups. Assuming p, q and r belong to three different technology groups, a *lenient trigram* allows other elements to come in between p, q, and r, as long as they all belong to the either the technology group of p or technology group of q. Thus $\langle p\ \ q\ i\ j\ r\rangle$ is still counted as the trigram $\langle p\ q\ r\rangle$, if i has the same technology group as that of either p or q, and so does j. The notion of lenient trigram $\langle p\ q\ r\rangle$ also works when p and q have the same technology group, which is different from that of r.

The algorithm **Next-Frequent-PC** works as follows. If there is no project change after g, then it makes the same predictions as **Next-Frequent**. Suppose the last training g of employee e belonged to the technology group JAVA. Now suppose there is a project change and also suppose that the new project for e makes use of two technology groups, say ASSURANCE and WEB_TECHNOLOGIES. Now find $\lceil K/4\rceil$ most frequent lenient bigrams of the type $\langle g\ x\rangle$, where the training x belongs to ASSURANCE technology group. Similarly, find $\lceil K/4\rceil$ most frequent lenient bigrams of the type $\langle g\ y\rangle$, where the training y belongs to WEB_TECHNOLOGIES technology group. Finally, find $\lfloor K/2\rfloor$ most frequent bigrams of the form $\langle g\ z\rangle$ using **Next-Frequent** i.e. without paying heed to their technology group of z. The final recommendation is the union of these sets. One complication is that the new project may include the technology group of g. In our example, the new project may use technology groups JAVA and ASSURANCE. In that case, we recommend the union of (i) $\lfloor K/4\rfloor$ trainings using **NF-Lookback-MatchTG(1)** (which will recommend only those trainings having the same technology group as that of g), (ii) $\lceil K/4\rceil$ most frequent lenient bigrams of the type $\langle g\ y\rangle$, where the training y belongs to ASSURANCE technology group, and (iii) $\lfloor K/2\rfloor$ most frequent bigrams of the form $\langle g\ z\rangle$ using **Next-Frequent**.

The algorithm **NF-Lookback-PC(** b **)** works as follows. With $b = 1$, it is the same as **Next-Frequent-PC**. We illustrate the algorithm assuming $b = 2$. If there is no project change after g, then it makes the same predictions as **NF-Lookback(2)**. Suppose the last two trainings of the employee are $\langle f\ g\rangle$. Now suppose there is a project change after g and also suppose that the new project for e makes use of two technology groups, say ASSURANCE and WEB_TECHNOLOGIES. Now find $\lceil K/4\rceil$ most frequent lenient trigrams of the type $\langle f\ g\ x\rangle$, where the training x belongs to ASSURANCE technology group. Similarly, find $\lceil K/4\rceil$ most frequent lenient trigrams of the type $\langle f\ g\ y\rangle$, where the training y belongs to WEB_TECHNOLOGIES technology group. Finally, find $\lfloor K/2\rfloor$ most frequent trigrams of the form $\langle f\ g\ z\rangle$ using **NF-Lookback(2)**. The final recommendation is the union of these sets. The case that the new project may include the technology group of either f or g or both is handled similarly.

We now present another algorithm, which we call *vTrain* (Algorithm 1), which takes into account the project P to which a user e is currently allocated to. The idea is that people in the same project use a common set of technologies and hence their trainings will also be similar. Thus, to recommend the next training to e on a particular date d, we look at all the trainings done (up to the date d) by team members who are already in the project P. The algorithm simply scans the sequences dataset and counts the frequencies of bigrams in different ways (explained shortly), combines these counts as a weighted average score and selects K trainings having the highest score as the recommendation.

Table 6 Contingency table to test whether project change and technology group are independent

	Technology group change	No technology group change	Total
Project change	37, 323	12, 164	49, 487
No project change	382, 415	383, 678	766, 093
	419, 738	395, 842	815, 580

The *skipped count* of an ordered pair of trainings $\langle a\ b\rangle$ (among a given set of sequences) is the sum of the counts of the trigrams of the form $\langle a\ x\ b\rangle$, where x is any training. We define the *co-occurrence count* of a bigram $\langle a\ b\rangle$ as the number of sequences in which both a and b occur, but not necessarily in the same order, and not necessarily next to each other (i.e. with any number of other elements between them). The *inverse count* of the bigram $\langle a\ b\rangle$ is the count of the bigram $\langle b\ a\rangle$. In our example, for the bigram $\langle c\ d\rangle$ the skipped count is 2, the co-occurrence count is 9, and the inverse count is 1.

In our dataset, we have both start and end dates of project allocation as well as the trainings for all employees. Thus, we can always find the subsequence of all the trainings that an employee did when she was allocated to a particular project P; we call such a subsequence as a *projection* onto a project P. Thus, if an employee's sequence is $\langle a\ b\ c\ d\ e\ f\rangle$ and if she did trainings c and d, while she was allocated to a project P, then her projection sequence for P is $\langle c\ d\rangle$. Given a specific project P, we first take the projection sequences for all employees who ever were allocated to P and then define the *in-project count* of a bigram $\langle a\ b\rangle$ as its count computed only using the set of projection sequences onto P. The *in-TG count* of a bigram $\langle a\ b\rangle$ is 0 if a and b belong to different technology groups and otherwise it is the count of the sequences in which a is followed by b with possibly other trainings of any *other* technology group occurring between them.

The weights for the bigram counts such as *co-occurrence count* or *in-project count* for use in *vTrain* (Algorithm 1) are learnt using a validation set carved out from the training data.

7 Baseline Recommendation Engine Methods

We have used several standard recommendation methods as baselines, in order to compare them with our algorithms, in this particular application of recommending the next training to a user.

- **Nearest Neighbours** $NN(N_0)$: Let $M = |C|$ denote the total number of distinct trainings available. We represent each employee e as a Boolean *training vector* T_e of size M, whose i-th entry is 1 if employee e has done the training c_i, and 0 otherwise. For each employee e in the test dataset, we find the most similar N_0 employees using cosine similarity, then we compute the set of trainings which are present in the training vectors of these N_0 employees but are not present in the training vector of e, and recommend the K most frequent trainings among these to e.
- **Bayesian Personalized Ranking (BPR)** [13]: Let $U = \{1, 2, \ldots, |U|\}$ denote the ordered set of users. Let $I = \{1, 2, \ldots, |I|\}$ be an ordered set of items. X is a matrix of dimensions $U \times I$ giving the ratings of each user for each item; this matrix X is, in general, quite sparse. A representation of the preferences of users is approximated by a $|U| \times k$-dimensional matrix. Similarly, an item-wise representation of the preferences of users is approximated by a $|I| \times k$-dimensional matrix. W and H are computed using SVD, and k is chosen in such a way that $W \times H^T$ is as close to the original X as desired. The Bayesian Personalized Ranking (BPR) method estimates

Algorithm 1: Algorithm $vTrain$

input : C = Set of all M trainings, D = set of sequences for all N employees, P = current project, weights $\{w_1, w_2, \ldots, w_6\}$, $\sigma = \langle c_1\ c_2\ \ldots\ c_m\rangle$ = sequence for a given employee who is allocated to P and for whom to recommend K trainings

output: S = Set of K recommended trainings for e

c_m = the last training done by e;

foreach *training* $x \in C, x \neq c_m$ **do**

  f_1 := count of bigram $\langle c_m\ x\rangle$ in D

  f_2 := skipped count of bigram $\langle c_m\ x\rangle$ in D

  f_3 := inverse count of bigram $\langle c_m\ x\rangle$ in D

  f_4 := in-TG count of bigram $\langle c_m\ x\rangle$ in D

  f_5 := in-project count of bigram $\langle c_m\ x\rangle$ in projection sequences onto P

  f_6 := co-occurrence count of bigram $\langle c_m\ x\rangle$ in D

  score $s_{c_m,x} := w_1 \cdot f_1 + w_2 \cdot f_2 + \ldots + w_6 \cdot f_6$ for bigram $\langle c_m\ x\rangle$

end

Sort the bigrams $\langle c_m\ x\rangle$ in descending order of the score $s_{c_m,x}$;

S := top K trainings x such that $\langle c_m\ x\rangle$ has the highest scores;

W and H, when the pairwise item preference data from each user is available. The BPR method finds a model parameter vector Θ, which has the maximum posterior probability: $p(\Theta|\succeq_u) = p(\succeq_u|\Theta) \cdot p(\Theta)$, where $\succeq_u$ is the preference relation over items given by user u. Here Θ consists of the elements of matrices W and H. They derive an estimator for the maximum posterior probability and use it for the optimal learning of the model parameter vector Θ. A general prior, $p(\Theta)$, is used which is a normal distribution with zero mean and the data-based variance–covariance matrix. The probability of a preference instance, such as user u prefers item i over item j, is modelled using the logistic sigmoid function parameterized by Θ. In our case, the items correspond to trainings and the preference relation of a user (employee) e is derived her sequence σ_e of trainings as follows: for every training $x \in \sigma_e$, and every training $y \notin \sigma_e$, we say $x \succeq_u y$. Given the $1 \times k$ row W_e of matrix W for employee e, we evaluate the score $W_e \times H_i$ for each training i not done by e and select the top K trainings with the highest value of this score as the K recommended trainings for e [7].

– **LDA**: In the seminal paper [3], the authors formalize the notion of a topic as a probability distribution over words. Based on the intuition that a document collection has multiple topics, and each document exhibits multiple hidden topics, they propose a generative graphical model called *latent Dirichlet allocation* (LDA) to model the (imaginary) process of how the documents are generated. LDA by Blei et. al. [3] is the simplest topic model of document collections based on the intuition that documents exhibit multiple hidden topics, where each topic is characterized by a distribution over words. LDA is a generative topic model, and it assumes that documents are generated by an imaginary random process. Figure 2 shows the graphical model of LDA. When applying LDA to the recommendation scenario, an item is akin to a word and user's set of preferred items form a document. In this graphical model, shaded and unshaded variables indicate observed and hidden discrete variables, respectively. The words in documents are observed variables and the probability distribution over words for each topic (Φ), the distribution over topics for each document (Θ), the per-document per-word topic assignments (Z) are hidden variables. These hidden variables form the hidden topic structure. Let $M_D = <\Phi, \Theta, Z>$ denotes the hidden structure of a LDA model on a document collection D. Generative process of LDA can be described as follows:

1. for $t = 1 \ldots T$(a) $\quad \phi_t \sim \text{Dirichlet}(\beta)$

2. for each document $d \in D$(a) $\theta_d \sim \text{Dirichlet}(\alpha)$
   (b) for each word w at position n in d
   i. $z_d^n \sim \text{Multinomial}(\theta_d)$
   ii. $w_d^n \sim \text{Multinomial}(z_d^n)$

where T is the number of topics, ϕ_t is the word probabilities for topic t, θ_d is the topic probability distribution, z_d^n is topic assignment and w_d^n is word assignment for nth word position in document d, respectively. α and β are topic and word Dirichlet priors, respectively. LDA finds a set of topics on the basis of word co-occurrence patterns in documents. In LDA, most probable words of a topic frequently co-occur with each other in the documents. The word co-occurrence patterns of a word give its contextual information, which can be used to disambiguate the word. Here, the words in documents are observed variables and the probability distribution over words for each topic, the distribution over topics for each document, and the per-document per-word topic assignments are hidden variables. Using Dirichlet priors for topic and word distributions, the LDA algorithm learns the hidden distributions using Gibbs Sampling or variational Bayesian methods. Thus, LDA effectively finds a set of topics on the basis of word co-occurrence patterns in documents, where the most probable words of a topic frequently co-occur with each other in the documents. In our case, trainings are words, and each sequence of trainings by a particular user is a document. A topic corresponds to a probability distribution over trainings. For a given test user u, a candidate training c and a topic t, we compute the score of c w.r.t. t, as the product of the probability that t occurs in the document (sequence of trainings) corresponding to u, and the probability of c in t. The final score of c for u is the sum of these scores over all topics t. The K trainings with highest scores are then recommended to u. To decide the optimal number of topics, we tested with possible values in the range of 5–45 and finalized 20 for its good performance.

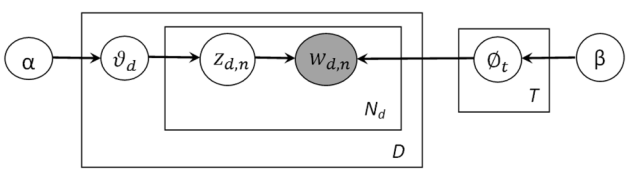

Fig. 2 The graphical model for latent Dirichlet allocation

– **Bigram-based topic models**: The bigram-based topic modelling [2] is extension of LDA, as it incorporates bigram-like subsequence information. The sequence-based topic models learn the distribution for each topic

over all the *training bigrams* in place of the distribution over individual trainings as in LDA. For computing the preference score of a training c_j for a user u after training c_i, the topic-specific probabilities associated with a training bigram ($\sigma_{u,c_i} \rightarrow \sigma_{u,c_j}$) are used along with the topic probabilities for the user u.

- **CARS**: Many recommendation systems take into account the context of the users to create recommendations. The CARS algorithm [8] adds keywords from user profiles as context to the LDA model for item recommendation. For applying CARS in our case, we used keywords from the names of the trainings done by a user as her *profile*. We additionally learn a probability distribution for each topic over the user's profile keywords. CARS adds an extra term to the formula for computing the score in LDA, which is the product of the probabilities of the keywords from the profile of u in the topic t. Both, LDA and CARS, ignore the *sequence* information, i.e. order among the trainings done by a user.
- **CSRS**: The CSRS system [17] is similar to CARS, except that it incorporates bigram-like subsequence information. CSRS learns the distribution for each topic over all the *training bigrams* in place of the distribution over individual trainings, as in CARS. The CARS method of computing the preference score of a training c for a user u is modified to work with probabilities associated with a bigram.

8 Training Similarity

One important factor that affects the accuracy of training recommendation is *training similarity*. If trainings a and b are "very similar", then at an abstract level it is less important which of these trainings is actually recommended, although practically there may be reasons to prefer say a over b. Thus, accurately measuring training similarity has implications for quality and accuracy of recommendations. Ideally, one should measure similarity of trainings in terms of their contents, i.e. the concepts that they cover. However, two trainings which are "very similar" in their contents, may differ in terms of other aspects such as duration, quality of instruction or difficulty level. (We ignore these other aspects to measure training similarity.) If we had access to the training description or syllabus/concepts covered in the training, then we could use text-mining techniques to measure their similarity. However, we assume that we do not have access to such text; if such text is available, then the similarity measurement can be considerably improved. In its absence, we need to depend only on the past training sequence data of employees to measure training similarity. We do not have the training description or syllabus or concepts covered; therefore, the use of text-mining techniques to measure training similarity is not feasible. Although content-wise two "very similar" trainings may differ in aspects of duration, quality of instruction or difficulty level. However, we assume that we do not have access to such text; if such text is available, then the similarity measurement can be considerably improved.

Intuitively, two trainings a and b are "very similar" to each other if (i) they belong to the same subject (i.e. to the same technology group in our setting); and (ii) a is *replaceable* by b and vice versa. Informally, replaceability means that employees in the same context have no particular preference for a over b or vice versa. We formalize the concept of replaceability as follows. We say that a training a is *replaceable* by b and vice versa if: (i) the trainings that people take before a and the trainings that people take before b are the same; and (ii) the trainings that people take after a and the trainings that people take after b are the same; and (iii) not many people take both a and b, i.e. a and b are independent of each other. Given a training c and the set of all trainings C, we obtain a probability distribution $P_{\text{past},c}(x)$ over C, where $P_{\text{past},c}(x)$ gives the probability that a training $x \in C$ occurs before the training c. This probability distribution can be easily estimated from the set of sequences of trainings of all employees. In a similar manner, we can estimate the probability distribution $P_{\text{future},c}(x)$, which gives the probability that a training $x \in C$ occurs after the training c. Note that both these are discrete probability distributions, since C is a finite set. We compute *Jensen–Shannon divergence (JSD)* between $P_{\text{past},a}(x)$ and $P_{\text{past},b}(x)$ as well as $P_{\text{future},a}(x)$ and $P_{\text{future},b}(x)$ for any two given trainings a and b. If the average of the these two JSD values is below a user-specified threshold h_0, we say that a and b are *similar*. We use the χ^2 -test to test whether the occurrences of a and b are *independent*. Thus, a and b are *replaceable* if a and b are *similar* and *independent*. We now use *Jensen–Shannon divergence (JSD)* between probability distributions $P_{\text{past},c}(a)$ and $P_{\text{past},c}(b)$ to measure the similarity between the past contexts of the trainings a and b. We measure the similarity between $P_{\text{future},c}(a)$ and $P_{\text{future},c}(b)$ in a similar manner. We define the *overall JSD value* between a and b to be the average of the two JSD values: one between past distributions and the other between the future distributions. Given a user-specified threshold h_0, we say that a and b are *similar* if the overall JSD value for a and b is less than h_0. Note that the probability distribution $P_{\text{past},c}(a)$ ($P_{\text{future},c}(a)$) ignores the *order* of trainings that occur before (after) a. Finally, we use the χ^2-test to test whether the occurrences of a and b are independent. Thus, the formal definition that a and b are replaceable is that: (i) a and b are similar (as defined above); and (ii) occurrences of a and b are independent, as per χ^2-test. As an example, overall JSD value for trainings `Formatting E-mail and Configuring Message Options in Outlook 2013` and `Mail Automation, Cleanup, and Storage`

`in Outlook 2013` is 0.0067, whereas for `Managing E-mail in Outlook 2013` and `Working with Meetings in Outlook 2013` it is 0.0109, indicating that the latter pair of trainings has less similarity with each other.

Using this notion of similarity between trainings, we now re-measure the prediction accuracies of all algorithms discussed so far, as follows. If an algorithm predicts training a for an employee e, and the actual next training that e does is b, and a and b are replaceable as defined above, then mark this prediction is correct (i.e. a *hit*). We have empirically observed that the prediction accuracy of all algorithms improve to some extent, if we measure it using this relaxed notion of correct prediction. We report remeasured accuracies for a few algorithms namely **Next-Frequent** and **NF-Lookback(2)** reported as **Next-Frequent-TSim** and **NF-Lookback-TSim(2)**, respectively, in Table 7.

9 Experimental Evaluation

We split our dataset D of 118,587 sequences (one sequence of trainings for each employee) into two parts: training and test. The test dataset contains 23,718 sequences for 20% employees. From each sequence in the test dataset, we remove the last k elements (i.e. we take its k-chop), use the k-chop to predict a set S containing $K = 5$ trainings for that employee, and check if at least one of the k trainings in the residue is in S; if yes, we count that as a *hit* else a *failure*. We report the % of hits, which is the same as precision at 5 (P@5). We apply each of the proposed algorithms to this dataset, including the various baselines, and the results are shown in Table 7. As seen, most of our algorithms outperform the various baselines, the algorithm **vTrain** being the best overall, across the various values of the *horizon* k. For tuning the weights of the bigram counts based on the different criteria in **vTrain** (Algorithm 1), we use grid search in a separate run where the available data are divided into training (60%), validation (20%), and testing (20%) datasets. The discovered weights for various bigram counts are in range (0.8, 1). Among our sequence matching algorithms, **NF-Lookback(2)** works the best, closely followed by **Next-Frequent**. The domain information added in form additional weighing of the important patterns such as in-project training bigrams or co-occurrence-based bigrams improves the accuracy of prediction as seen in **vTrain**. It is evident that the project or the technology group are important criterion which contribute to the decision choices related to the trainings. The error is mainly due to the *exploratory nature of learning* specific to each individual employee and this behaviour does not conform to the *winner takes all* strategy which is favoured by the proposed algorithms. The sequence mining algorithm **VS_Feasible_Seq** has higher

Table 7 Comparison of various methods for predicting next training

Method	P@5 for $k = 1$	P@5 for $k = 2$	P@5 for $k = 3$	P@5 for $k = 5$
Next-Frequent	41.66	52.34	58.64	65.69
NF-Lookback(2)	41.93	52.79	59.00	65.92
NF-Lookback(3)	41.41	52.18	58.61	65.68
NF-Lookback-U(2)	40.43	51.39	57.90	65.29
NF-Lookback-U(3)	39.13	50.06	56.73	64.58
NF-Lookback-SkipTG-U(2)	39.90	50.83	57.31	64.77
NF-Lookback-MatchTG-U(2)	40.12	50.98	57.51	64.75
NF-Lookback-BiMax	38.61	49.43	55.69	63.06
NF-Lookback-BiMax-U	39.75	50.38	56.84	64.04
NF-TG-Transition	40.79	51.54	57.82	64.93
Next-Frequent-PC	41.28	51.95	58.41	66.10
NF-Lookback-PC(2)	42.14	53.74	59.67	67.10
vTrain	44.31	55.45	62.13	67.95
VS_Feasible_Seq	31.85	40.71	46.75	52.59
S_Feasible_Seq	22.23	28.27	33.92	35.68
k-NN(100)	27.26	37.09	42.14	50.69
BPR	1.37	4.48	8.27	9.31
LDA (#topic = 20)	17.17	22.16	25.46	–
Bigram topic model (#topic = 15)	1.71	2.61	3.64	5.67
CARS (#topic = 20)	17.42	23.25	26.00	–
CSRS (#topic = 20)	19.94	28.59	30.85	–
Next-Frequent-TSim	42.82	53.28	60.12	66.53
NF-lookback-TSim(2)	44.22	57.73	62.07	68.82

accuracy as compared to the abstract, latent feature-based methods involving matrix factorization and topic modelling. The learnt latent dimensions as in **BPR**, **LDA**, **CARS** or **CSRS** are also not refined enough due to exploratory nature of learning behaviour present in the training sequences, resulting in lower accuracies. As expected, adding training similarity to our algorithms generally improved the prediction accuracies where a maximum improvement of 4.94 in $P@2$ for $K = 2$ is noted for **NF-Lookback-TSim(2)**.

To be fair, **k-NN**, **MF**, **LDA** or **CARS**, do not make use of the sequence information inherent in our dataset. Moreover, our data does not include feedback (ratings) for any training, which is a crucial element of many recommendation algorithms. We also have some baseline algorithms, which make use of the sequence information such as **CSRS**; our algorithms outperform these as well. We observe broadly that simple sequence matching algorithms are very useful for training recommendations based on past data. We also observe that adding some specific contextual information such as the project allocation or similarity between trainings generally improves the prediction accuracy. The cold-start issue is also being addressed in algorithms such as **vTrain** which use technology group of the held skills of a new employee as well as the technologies used in the project for which she has been recruited.

Kindly note that we have addressed the problem of recommending *next* training or only one training and hence the temporal ordering or position in the recommended set of trainings is not considered in evaluation.

10 Sequence Prediction Using Markov Decision Process

As in case of strategy games such as Chess, the *next training* recommendation can be based on a look-ahead over actions and associated rewards which accrue over time through possible next trainings as shown in [?]. The strategy or policy discovery methods explore the state space for rewards while exploiting the available past data. For this purpose, the Markov decision process (MDP) framework provides a mechanism to define possible state space, actions, transitions and projection of rewards through multiple decision epochs. For defining an MDP, it is important to design and define the reward associated with each state which represents the true *utility* of a training. In our case, this *utility* can be based on the domain knowledge as seen in algorithms such as *vTrain*.

10.1 Modelling a Markov Decision Process for Training Recommendation

In this section, we formally define an MDP for our setting of next training recommendation. An MDP is defined as a five-tuple $\{S, A, R, P, T\}$ [11] where S represents a set of states, A is a set of possible actions, R represents rewards which is a set of real values associated with each pair of state and an associated action, P is the set of transition probabilities among states given each associated action, and T is the set of possible decision or time epochs.

At a decision epoch, the user is in a state, say $s \in S$ where S is of size M. In the training recommendation scenario, each state corresponds to a training where it represents a level of skill proficiency in related technology area. We assume that trainings are distinct in the available set. If we take into account cumulative learning (over the entire training history) of the user for defining state space over the possible proficiency levels of multiple skills, it will lead to a state-space explosion. In our model, we consider a state being defined by a training which represents proficiency level in a skill or a role. This assumption holds a simpler Markovian assumption for generation of training sequences.

The user can choose (and take) a training $c \in C$ as the *action*, where C is the set of available trainings. The S and C are finite and are assumed to be time invariant of size M. The training c taken as the action would transition the user to state associated with the training c.

When in state s_i for an action taken as training c_j the associated state transition probability from s_i to s_j is denoted by $p(s_j \mid c_j, s_i)$. In our model, there is one-to-one correspondence between a state s_i and a training c_i. We further simplify the notation for transition probability to $p(c_j \mid c_i)$ and further to $p_{i,j}$.

In our setting the transition probability between the states is calculated using frequency data of individual trainings from the bigram instances as follows:

$$p_{i,j} = \frac{(\text{Count of bigram } \langle c_i c_j \rangle)}{\sum_{c_q \in C} (\text{Count of bigram } \langle c_i c_q \rangle)} \tag{1}$$

We also assert that

$$\sum_{c_j \in C} p_{i,j} = 1 \quad \text{for all} \quad i \tag{2}$$

The associated reward with next training represents the gain in "utility" which includes increase in proficiency level in a technology or efficient role play in a project. In current manifestation, the reward is based on "popularity" of the training which serves as a proxy for its "utility". The rewards are associated with multiple factors such as frequency of training instances, frequency of transitions within a technology group or frequency of transitions within the project boundaries. In the model, this reward can be denoted by $r(i, j)$ or $r_{i,j}$. In current setting, it is associated with the target state of a transition, s_j hence we simplify the notation to r_j. The set of all rewards is $R = r_1, r_2, \ldots, r_M$.

For the reward initialization, the computation for each state is based on the domain knowledge where each state (training) will be assigned reward using following rules:

- For each second training in a bigram, the state corresponding to second training will earn a reward of + 1.
- If both the trainings in a bigram are taken during the same project, then the state corresponding to second training will earn a reward of + 3.
- If both trainings in a bigram are in the same technology group, then the state corresponding to second training will earn a reward of + 2.
- For a bigram, if the transition is from a "basic" level training to an "advanced" level training, then the state corresponding to the "advanced" level training will earn a reward of + 1. These levels of trainings are derived using the domain-specific keywords present in the names of the trainings such as *Beginner*, *Basic*, *Foundation*, *Advanced*, *Expert* as well as by considering technology features such as *Core Java*, *J2EE*.
- For a bigram, if the both trainings are available in the same training channel (i.e. same online learning system offers both the trainings), then the state corresponding to second training will earn a reward of + 1.

The time points at which decisions are made or the *decision epochs* are denoted by T. For the training, these epochs are discrete as the opportune time for training is selected by the user.

This collection of sets $\{S = C, A = C, R, p(\cdot|c_i \in C), T\}$ defines the Markov decision process.

10.2 Optimzing an MDP

In MDP, the user's goal is to maximize the reward stream which is the discounted sum of rewards or an average reward over multiple decision epochs. An optimal solution to the defined MDP is to learn the reward maximization behaviour of the system. A *policy*, π maps each state to an action or a training in our case. If we obtain an optimal policy then for a user in state c_i the next recommended training is $c_j = \pi(c_i)$.

There are various exact or approximate algorithms to compute the optimal policy which are selected based on the number of states in MDP, computing power or memory available, etc. The most common algorithms are (i) *value iteration*, (ii) *policy iteration* and (iii) *linear programming*. Most of these methods attempt to maximize a value function, V^{π} defined for each training c_i treated as the starting training, as follows:

$$V^{\pi}(c_i) = r_i + \gamma \sum_{c_j \in C} p_{ij} V^{\pi}(c_j) \tag{3}$$

where γ is the discount factor.

As we see, this is a recursive equation which traces a *Markov Chain* across trainings (states) accruing the reward value, where the next training c_j is as per the chosen policy. Thus, the value function of a policy assigns to each training c_i, the expected discounted sum of rewards over infinite horizon as per the policy when the starting training is c_i. The optimal value function, V^*, provides an optimal value to each state c_i as per the learnt optimal policy π^*, and is as follows:

$$V^*(c_i) = \max_{c_j \in C} \left[r_i + \gamma p_{ij} V^*(c_j) \right] \tag{4}$$

In our setting, after initialization of rewards for all trainings and computing the transition probabilities, the value function is stabilized using *value iteration*. Through this procedure, we compute the optimal value function (or reward) as well as the optimal policy. This iterative procedure uses the following formulae, 5 and 6:

$$V^{z+1}(c_i) = r_i + \gamma \sum_{c_j \in C} p_{ij} V^{z}(c_j); \quad \text{where number of iterations, } z \leq Z \tag{5}$$

$$\forall c_i \in C, \quad \left| V^{z+1}(c_i) - V^{z}(c_i) \right| \leq \epsilon \tag{6}$$

The variable z in Eqs. 5 and 6 denotes the number of decision epochs or iterations for computing optimal policy (π^*) along with value function for each state transition. Z is the maximum number of decision epochs.

For learning a stable policy, the error threshold value, ϵ, is set to 0.2 and the maximum number of iterations allowed are $Z = 2000$. An MDP model, MDP_{Gen} is constructed using complete training sequence of each employee from the dataset described in next section and its recommendation performance is reported in Table 8.

10.3 Factored Markov Decision Processes for Training Recommendation

As in [18], the factored Markov decision processes (fMDPs) assume that in place of transitions among items in an MDP,

Table 8 Comparison of various ensemble of fMDPs for predicting next training

Ensemble of MDP(s)	P@5 for k = 3
MDP_{Gen}	44.12
$MDP_{TechGrp}$	44.3
MDP_{Proj}	46.85
$MDP_{Gen} + MDP_{Proj}$	47.67
$MDP_{Gen} + MDP_{Proj}$ (considering last 2 trainings)	50.22

the transitions can be among the associated attributes. Assuming independence among attributes describing an item, per attribute or per factor, an independent MDP can be modelled leading to an ensemble of disjoint MDPs. Due to this change, the value function for MDP defined over items does not retain the structure for the factor-specific fMDPs. However, authors prove that a structured value function can be obtained for fMDPs having independent components for each factor. The fMDPs reduce the size of the state space and avoid the curse of dimensionality. Although we borrow the idea of fMDPs from [18], we define the factors more as influential drivers and important segments existing in the organizational domain of technical learning using trainings.

For training recommendation, the important factors identified are *projects*, *technology groups*, *training delivery channels*, *role of employees*, etc. For the training recommendation, we chose two most important factors, namely *projects* and *technology groups*. For the *project* factor, all training sequences are split along project boundaries, for example, for a training sequence $\langle a\ ;b\ c\ ;;d\ e\ f\ ;g\rangle$ only subsequences $\langle b\ c\rangle$ and $\langle d\ e\ f\rangle$ are considered where ; limits a project duration. Using these subsequences, which are observed during projects, a factored MDP model, MDP_{Proj}, is developed. Similarly the training subsequences of employees which have trainings from the same technology group are used to develop another factored MDP model, $MDP_{TechGrp}$. The method of reward initialization, transition probability and iterative value computation for MDP_{Proj} and $MDP_{TechGrp}$ is same as used for MDP_{Gen}.

10.4 Discussion on Experimentation and Results

We experimented with ensembles of the three models, MDP_{Gen}, MDP_{Proj} and $MDP_{TechGrp}$, and results are presented in Tables 8 and 9. For constructing MDPs, we have used a subset of the data available. The training dataset consisted of training sequences of 21,143 employees, having 186 distinct trainings, which are related to technology area of *Java and J2EE*. The test dataset consisted of training sequences of 2143 employees. There are 7741 unique bigrams in the training dataset and 3015 are in test dataset which are present in training dataset also. There were 9 technology subgroups defined within the technology area of *Java and J2EE* based on the domain knowledge. The training subsequences spanned 729 projects. We measure and report results as Precision@5 for horizon=3 for the individual as well as various ensembles of MDP models. For the best performing ensemble, we report Precision@5 for values of horizon from 1 to 5 in Table 9.

Table 9 Performance of ensemble of MDPs: $MDP_{Gen} + MDP_{Proj}$ (considering last two trainings)

P5 for k = 1	20.47
P5 for k = 2	30.81
P5 for k = 3	50.22
P5 for k = 4	62.14
P5 for k = 5	70.81

For generating recommendations, the selection of next training or next action is as per the learnt optimal policy for an MDP model. For making recommendations using an ensemble, we sum the values associated with each of the next trainings (possible from current state) as per the optimal policy learnt for each of the member MDP model. For testing the performance of an MDP, we consider the last training of the employee as the current state after performing a *right k-chop* operation where k is the horizon value used for verification. In one of the ensembles, we have considered last two *(after right k-chop)* trainings for making predictions independently. In this case, we consider union of ordered sets of recommendations based on each of these two trainings. The top five of these trainings are recommended based on the sum of associated values from optimal policies of individual MDPs participating in the ensemble.

Kindly note that the results are reported on a smaller dataset for the MDP-based approaches. Interestingly, as shown in Tables 8 and 9, the $P@5$ values for the residue sizes of $k = 1, 2, 3$, are much lower than what is being achieved by for $P@5$ for $k = 4, 5$. This is due to the higher reward being associated with the trainings towards the end of the residue size of 5 in a sequence rather than at the lower values of 1, 2 or 3. This is due to the *exploratory nature of learning* exhibited by employees and the MDPs learn exactly this optimality. It means that the recommended trainings with higher rewards are actually present towards the end of the larger residues of size 4 or 5 in the past data of training sequences. This leads to a finding that most of the employees explore different trainings at relative positions of 1, 2, 3 on their learning path to reach a higher rewarding training at relative position of 4 or 5. If training at relative position of 4 or 5 is taken *directly*, it provides higher optimal reward and hence such trainings are recommended by MDPs even for the lower values of k resulting in more *misses*. This effect is amplified in our case as we use past data to validate instead of a live system wherein the MDP-based recommendation system would have guided the employees to the trainings with optimal rewards avoiding exploratory trainings.

We have proposed an approach for next training recommendation using the ensemble of factored MDPs which shows promise though it is implemented for a smaller dataset. As seen from the prior work, the MDP models and the learnt optimal policies are used to initialize a recommendation system which is further tuned using reinforcement learning in an online setting. The current implementation of MDP and the computed optimal policy can be used for recommending a sequence of trainings based on the higher value gain after a look-ahead of n training predictions as

a sequence. The solution can be further mined to search popular sequences of trainings achieving movement into higher roles or attaining higher proficiencies in a skill for an employee. The fMDP implementation can be further improved using other factors such as role as well as by improving the value function by including the interaction among attributes.

11 Conclusions and Further Work

In knowledge economy, the skills and expertise of the employees directly impact the effectiveness and productivity of the workforce. Training is important for creating and maintaining the right skill-profile for the organization's workforce. There is a tremendous variety in the available trainings within an organization: technical, project management, quality, leadership, domain-specific, soft-skills, etc. Hence, in this paper, we presented several new unsupervised sequence mining algorithms for training recommendation in an industrial setting, to assist the employee in choosing the best trainings, which perfectly suits her background, project needs and career goals. The input to the algorithms is just the database of sequences of past trainings done by the employees. Using the real-life data about trainings of 118,587 employees over 5019 distinct trainings from a large multi-national IT organization, we showed that these algorithms outperform several standard recommendation engine algorithms as well as those based on standard sequence mining algorithms. We observed that our simple sequence matching algorithms are very useful for personalized training recommendations based on past data. We also demonstrated that adding some specific contextual information (such as the project that an employee currently allocated to, or similarity between trainings) generally improves the prediction accuracy. We have also formulated the next training recommendation as an ensemble of MDPs with encouraging results.

For future work, we are exploring an ensemble-based approach to combine predictions of the individual algorithms. We are enhancing our recommendation algorithms to make use of a detailed textual description of the contents of trainings. For example, using such descriptions, we can better represent the concepts covered in the training leading to a better similarity measure for trainings. We are incorporating feedback from people who have already completed a training to improve the recommendation algorithm, along with other information about the training such as its duration, or difficulty level. We are also exploring how more contextual information about the employee can be used to enhance the recommendations. We are exploring planning techniques to recommend a *sequence* of trainings to an employee in order to meet her long-term career aspirations, for example, aiding a programmer to move to the role of database administrator or GUI designer.

References

1. Aggarwal CC (2016) Recommender systems: the textbook. Springer, Cham
2. Barbieri N, Manco G, Ritacco E, Carnuccio M, Bevacqua A (2013) Probabilistic topic models for sequence data. Mach Learn 93(1):5–29
3. Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3:993–1022
4. Bobadilla J, Ortega F, Hernando A, GutiéRrez A (2013) Recommender systems survey. Knowl Based Syst 46:109–132
5. Chen BC, Agarwal DK (2015) Statistical methods for recommender systems. Springer, Berlin
6. de Baker RSJ, Shum SB, Duval E, Stamper JC, Wiley D (2012) Educational data mining meets learning analytics. In: Second international conference on learning analytics and knowledge, LAK 2012, Vancouver, April 29–May 02, p 20
7. Guo G, Zhang J, Sun Z, Yorke-Smith N (2015) Librec: a Java library for recommender systems. In: Posters, demos, late-breaking results and workshop proceedings of the 23rd conference on user modeling, adaptation, and personalization (UMAP 2015), Dublin, June 29–July 3
8. Hariri N, Mobasher B, Burke RD (2013) Query-driven context aware recommendation. In: RecSys, pp 9–16
9. Hummel HGK, van den Berg B, Berlanga AJ, Drachsler H, Janssen J, Nadolski R, Koper R (2007) Combining social-based and information-based approaches for personalised recommendation on sequencing learning activities. IJLT 3(2):152–168
10. Manouselis N, Drachsler H, Verbert K, Duval E (2012) Recommender systems for learning. Springer, New York
11. Puterman ML (2014) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York
12. Rahman M., Oh J.C. (2015) Fast Online Learning to Recommend a Diverse Set from Big Data. In: Ali M, Kwon Y, Lee CH, Kim J, Kim Y (eds) Current Approaches in Applied Artificial Intelligence, International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. IEA/AIE 2015. Lecture Notes in Computer Science, vol 9101. Springer, Cham. pp. 361–370. https://doi.org/10.1007/978-3-319-19066-2_35
13. Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2009) BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of uncertainty in artificial intelligence, pp 452–461
14. Sacín CV, Agapito JB, Shafti L, Ortigosa A (2009) Recommendation in higher education using data mining techniques. In: Proceedings of the 2nd international conference on educational data mining-EDM 2009, Cordoba, July 1–3, pp 191–199
15. Shani G, Brafman RI, Heckerman D (2002) An MDP-based recommender system. In: Proceedings of the eighteenth conference on uncertainty in artificial intelligence, UAI'02. Morgan Kaufmann Publishers Inc., San Francisco, pp 453–460

16. Spoon K, Beemer J, Whitmer JC, Fan J, Frazee JP, Stronach J, Bohonak AJ, Schmidt-Thieme L (2016) Random forests for evaluating pedagogy and informing personalized learning. J Educ Data Min 8(2):20–50
17. Srivastava R, Hingmire S, Palshikar GK, Chaurasia S, Dixit A (2016) CSRS: a context and sequence aware recommendation system. In: Proceedings of 8th annual meeting of the forum on information retrieval evaluation, FIRE 2016, Kolkata, December 8–10, pp 8–15
18. Tavakol M, Brefeld U (2014) Factored MDPS for detecting topics of user sessions. In: Proceedings of the 8th ACM conference on recommender systems. ACM, pp 33–40
19. Thai-Nghe N, Drumond L, Krohn-Grimberghe A, Schmidt-Thieme L (2010) Recommender system for predicting student performance. In: Proceedings of the 1st workshop on recommender systems for technology enhanced learning, RecSysTEL 2010, Barcelona, September 29–30, vol 2, pp 2811–2819
20. Verbert K, Manouselis N, Ochoa X, Wolpers M, Drachsler H, Bosnic I, Duval E (2012) Context-aware recommender systems for learning: a survey and future challenges. IEEE Trans Learn Technol 5(4):318–335
21. Zhu F, Ip HH, Fok AWP, Cao J (2007) Peres: a personalized recommendation education system based on multi-agents & SCORM. In: 6th international conference on advances in web based learning-ICWL 2007, Edinburgh, August 15–17, revised papers, pp 31–42