Recommendation based on sequence Item2vec

Technical Report · February 2017

DOI: 10.13140/RG.2.2.25358.97601

1 author:

Bereket Abera Yilma
Università degli Studi di Trento
1 PUBLICATION **0** CITATIONS

SEE PROFILE

Recommendation based on sequence
Item2vec

Yilma Bereket Abera
University of Trento
Trento,Italy
bereketabera.yilma@studenti.unitn.it

Abstract

Recommendation system play an important role in e commerce systems such as Amazon, ebay, Zalando etc. In this project we focused on making recommendation based on what items the user has used in the past and on what the other users have done. which are the most widely used functions in online services for presenting relevant items from a given item. We used an online retail dataset with 541,909 transactions and 2603 items and also synthetically generated transaction history dataset with 5 million transaction and 24,010 unique items we applied distributed representation method to be able to recommend the next possible item for the particular user. The key idea of our approach is treating items as words, and then training the Skip gram model based on those items and users information. Resulting item vectors from the model can be used to calculate the cosine similarity between items, and find the similar items for a given list of items. The experiments show that our system achieved an accuracy of 44.46 % using the real dataset and 61.03% using synthetically generated one.

Keywords: Recommender system, Item2vec, Word2vec, RDD (Resilient Distributed Dataset)

1. INTRODUCTION

A rapid growth of online information and e-commerce sites users need a system capable of recommending next set of items that a user will most probably would like. For the purpose of this some applications are developed based on last transaction information [2], based on purchase sequential history [1], based on hierarchical representation model [3] and Item sequences and ranking mechanisms [4].

1.1. Sequential vs. General Recommender

The most common approach to generate recommendations is to discard any sequential information and learn what items a user is typically interested in. On the other hand, recommendations of sequential methods are based only on the last user events by learning what an arbitrary user buys next when he has bought a certain item in the recent past. Both methods have their strengths and disadvantages. Imagine a user that in general buys movies like Star Trek and Star Wars. In contrast to his usual buying behavior, he recently has purchased Titanic and Dirty Dancing after that a sequential based recommender would only recommend movies like Notting Hill and other romantic movies. In contrast, a global personalized recommender would correctly account for the general taste of the user and recommend also movies like Back To the Future, Alien or other science fiction movies. But there are also examples where sequential recommenders have advantages: E.g. good recommendations for a user that has recently bought a digital camera are accessories that other users have bought after buying that camera. Global personalized recommender would not adapt directly to the recent purchase (the digital camera) but would recommend items this user likes in general. The main objective of this work is making recommendation of a ranked list of items based on what items the user has used in the past and on what the other users have done. This means by learning from the past history of the user himself and also by considering other users history. Whenever users buy items online, the underlying e-commerce system keeps track of the purchase history as a list of sequence of items in a transaction database. These sequence of items that are stored in the database are used for further recommendation. One challenge in such recommendation systems is synonimity. When an item is represented with one or more different names having similar meaning, In such cases, the recommender cannot identify whether the terms represent different items or the same item. To address the task of recommendation based on sequence we employ a strategy based on representation learning. More precisely we utilize the so called Skip-Gram model for word representation learning in natural languages that is introduced by Mikolov et. al [7]. This helps us to efficiently learn meaningful distributed word and phrase representation from un-annotated text. In the study of Mikolov et. al [7] they present two complementary techniques called Skip-Gram and CBOW. Given a target word, the Skip-Gram model learns word representation by predicting the context of the target word, whereas the CBOW does the exact opposite. word2vec [7] contains distinct models (CBOW and Skip-gram) each with different training methods. So, based on this concept we developed a technique called Item2vec representation. This method works based on items representation in a lower dimensional space. we treat each item as a word and then take advantage of the Skip-gram model to learn a representation for each item. Eventually we leverage the learned representation of items to perform the actual recommendation. In total the contributions of our work is as follows

- we train the Word2vec model to assign a vector for each item, and build a recommendation system and we used our own Item2vec model by adding additional techniques.

2. Related Work

Recommendation based on sequence is a typical application of recommender system which utilizes sequential data by predicting the next possible item by learning from users past action. There are several approaches that are developed to improve the quality of recommendations with a number of dimensions. In this section we briefly review the related work on recommendation system i.e sequential recommender, content-based filtering, collaborative filtering . Sequential recommender is a well-known method and has broad applications including web-log analysis, customer purchase behaviour analysis and medical record analysis. In the retailing business, sequential patterns can be mined from the transaction records of customers. For example di P Wang et.al developed learning hierarchical representation model for next basket item recommendation [3]. In this work sequential transaction data is given per user, where a transaction is a set of items (e.g. shoes or bags) bought at one point in time. The target is to recommend items that the user probably want to buy in his/her next visit. The other sequential recommender [12, 14], is mostly relying on Markov chains, which explores the sequential transaction data by predicting the next purchase based on the last actions. The other work that is related to ours is done with the concept of sequential pattern mining for food recommendation [15]. This basically works. In [23], they have considered the problem of generation of sequential recommendations and for that proposed Markov decision processes (MDPs). They have showed MDP model is more beneficial than other predictive models.

Content based filtering [10] is a recommendation system based on a description of the item and a profile of the users preference. Basically content based filtering works recommendation in steps. First, an item profile consisting of a set of features is extracted for each purchased by each user, user profile are generated. Then similarity scores between user and item profiles are calculated to finally recommend items with top similarity scores. For example the recommendation that is developed by C. Basu et.al developed recommendation as classification [16]. This work states that the user of the system provides ratings of some items and the system makes informed guesses about what other items the user may like. Collaborative filtering [9, 12] methods are based on collecting and analysing a large amount of information based on users behaviours, activities or preferences and predicting what users will like based on their similarity to other users. There are many recommendations that are implemented using collaborative filtering approach. For example, Lee et al. [19] treat the market basket data as a binary user-item matrix and apply a binary logistic regression model based on Principal Component Analysis (PCA) for recommendation. Another example is paper by B. Sarwar analyzes different item based recommendation algorithms, like techniques for computing item-item similarities e.g item-item correlation, cosine similarities between items and it touches different techniques to make recommendation for them e.g weighted sum and regression. Hybrid approaches [11] can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them; by adding content-based capabilities to a collaborative-based approach (and vice versa); or by unifying the approaches into one model for a complete review of recommender systems). An example based hybrid approaches is an application that recommend mobile apps based what other users previously installed on their phone previously [21]. So the users can select among several content based and collaborative set of data items from the given recommendation.

3. Problem statement

In this section, we first introduce the problem formalization of recommendations based on sequences. we then describe the proposed Item2Vec in detail. After that, we talk about the learning and prediction procedure of Item2Vec. Given a users purchase history, usually a sequence of transaction data, one attempts to build a recommender system that can predict the next few items that the user most probably would like. A recommender system should be able to recognize a pattern in a given sequence the sequential behavior (i.e., buying one item leads to buying another next).

3.1. Formalization

Let U = $\{ u_1, u_2,..., u_n \}$ be a set of users and I =$\{ i_1, i_2,...,i_m \}$ be a set of items, where n and m denote the total number of unique users and items respectively. For each user u, a purchase history T^u of his transactions is given by $T^u := T^u_1 , T^u_2 , . . . ,T^u_t$, where $T^u_t \subseteq I$. The purchase history of all users is denoted as T:= $\{ T^1, T^2,..., T^n \}$.
our recommendation based on sequence task is to make recommendation of a ranked list of items to each user based on what items the user has used in the past (T^u) and on what the other users have done T (i.e purchase history of other users). Therefore to make recommendation of next item for a given list of items T^u which is bought by a particular user; we consider not only the transaction history or previous activity of that particular user but also the transaction history of the other users too. Since $T^u \subseteq T$ our task is based on a transaction history of all users T to make recommendation of ranked list of next possible item to be purchased after the purchase of a given list of items(transaction T^u). A single user might have multiple transactions ($T^u_1 , T^u_2 , . . . ,T^u_t$) but since we are also considering other users transaction history to make a recommendation, we focus on the whole transaction T, $T^u \subseteq T$. Afterwards in this document we refer T^u as a single transaction history.

4. SOLUTION

To address the above recommendation problem, here we present the proposed Item2vec in detail. The basic idea of

our work is to learn a recommendation model that involves user's sequential buying behavior.

Since we assume an equivalence relation between words in a natural language sentence and items in a transaction Tu our proposed approach, Item2vec for solving the formalized recommendation problem captures the intuition from distributed representations of words in a vector space. Such a representation has been proved to be helpful in downstream natural language processing tasks. Recently, Mikolov et al. (2013) have introduced the skip-gram text modeling architecture . It has been shown to efficiently learn meaningful distributed representations of words or phrases (also known as word embedding) from un-annotated text. In the distributed representation, similar words are projected into similar vectors. Vectors from Word2vec model conserve some of the semantic characteristics in operations regarding the semantic information that they capture. This idea of word representation has been applied to statistical language modeling, automatic speech recognition and machine translation, and a wide range of NLP tasks with considerable success. Skip-gram model is a two-layer neural network that processes text (i.e trained to reconstruct linguistic contexts of words). It enables us to learn a representation of words using the so called "learn by prediction" strategy. The shallow network learns a representation of words by way of predicting the context. That is, given a target word w, it iteratively tries to predict the context of w and adjust the model parameters (weights) by minimizing a cost function computed on the predicted context and the true context. Ultimately a high quality representation of words should be able to effectively predict their context.Word2vec takes as its input a large corpus of text and produces a vector space (feature vectors for words in that corpus), typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word2vec is not a deep neural network, it turns text into a numerical form that deep nets can understand. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. Word2vec can utilize either of two model architectures to produce a distributed representation of words: Continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. In this work we tried to adopt the skip-gram model architectures of word2vec which is an efficient method for learning high-quality vector representations of words from large amounts of unstructured text data. The skip-gram architecture weighs nearby context words more heavily than more distant context words. The formulation of skip gram described as follows:

$\Rightarrow$ Pr (context(Wt)|Wt)

This means predicting the surrounding using the current word. The word representations computed using neural networks are very interesting because the learned vectors explicitly encode many linguistic regularities and patterns.

The training objective is to learn word vector representations

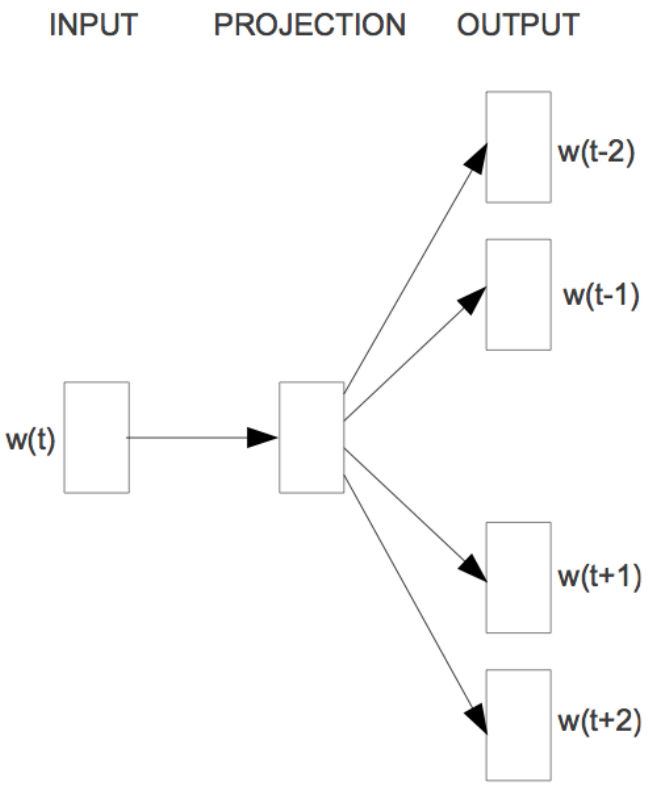
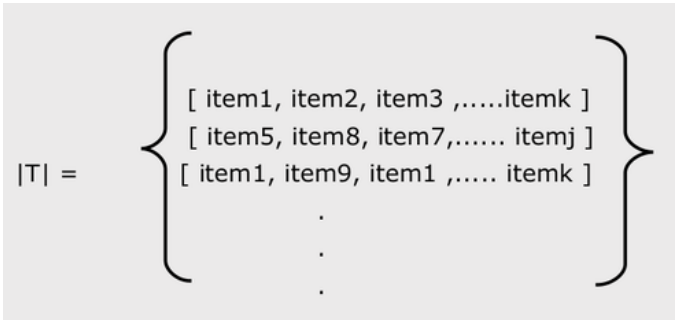


Fig. 1. The Skip-gram model architecture.

that are good at predicting the nearby words. Given enough data, usage and contexts, Word2vec can make highly accurate guesses about a words meaning based on past appearances. Those guesses can be used to establish a words association with other words (e.g. man is to boy what woman is to girl). The purpose and usefulness of Word2vec is to group the vectors of similar words together in vectorspace. That is, it detects similarities mathematically. Word2vec creates vectors that are distributed numerical representations of word features, features such as the context of individual words. It does so without human intervention. Because words are simply discrete state, and we are simply looking for the transitional probabilities between those states: the likelihood that they will co-occur. So Item2vec is possible.

4.1. Item2vec Algorithm

Traditionally, Word2vec models are trained on textual corpus data but here we utilize it on the purchase data of users. We are therefore treating each item as a word. Then the original Word2Vec method can be applied in the recommendation scenario. Based on the idea of utilizing cosine distance to measure the similarity between items, we apply distributed representation approach to build our recommender system. we used a transformed Online Retail dataset found at [22].



The transformed dataset contains 541,909 transactions T and 2603 items |I|. $T^u \subseteq T$, represents transaction history of an individual user

Item2vec algorithm first goes through each transaction (basket) and counts the number of items in each transaction during its first database scan if the number of items in the transaction is less than N it discards the transaction T^u. Then as skip gram architecture of word2vec for Natural language processing uses sentences to train a neural network model maximizing the conditional probability of context given the word, Item2vec also uses transactions to train a neural network model maximizing the conditional probability of context (similar items) given an item. And constructs a vocabulary from which it learns vector representation R^n for the items. Our skip gram model architecture of Item2vec enables us to learn a representation of items using the so called "learn by prediction" strategy. The shallow network learns a representation of items by way of predicting the context(similar items). That is, given a target item i, it iteratively tries to predict the context of i and adjust the model parameters (weights) by minimizing a cost function computed on the predicted context and the true context. Ultimately a high quality representation of items should be able to effectively predict their context. Item2vec takes as its input a large corpus of transactions and produces a vector space (feature vectors for items in each of the transactions), typically of several hundred dimensions, with each unique item in the transactions being assigned a corresponding vector in the space. Item2vec turns text(items) into a numerical form that deep nets can understand. Item vectors are positioned in the vector space such that Items that share common contexts(similar items) in the transaction database are located in close proximity to one another in the space. The formulation of our skip gram model architecture of Item2vec is described as follows:

$\Rightarrow$ Pr (context(Item)|Item)

This means predicting the surrounding (similar items) using the current item. We used Gensim which is an open-source vector space modeling and topic modeling toolkit, implemented in the Python programming language. It uses NumPy, SciPy and optionally Cython for performance. Similar items are located closer in the vector space.

Algorithm1: preprocessing transaction dataset

Input : Transactions dataset **T** , threshold (θ)
Output: Train transaction dataset $\mathbf{T}^{train}$, Test transaction dataset $\mathbf{T}^{test}$

Data: T
Read { train_data }
Split (**T**) =T^{tr}, T^{tes}
for each *transaction* T^u in T^{tr} do
  If ($|T^i| > \theta$)
    Keep T^i
  end if
  Output T^{train}
end
for each *transaction* T^u in T^{tes} do
  If ($|T^i| >$ n)
    Keep $\mathbf{T}^u$
  end if
  Output $\mathbf{T}^{test}$
end

ϕ is vector representation of T^{train}
T^u transaction(basket) in T^{train}
w window size - Maximum number words/items of context(nearby items for a given item)

Algorithm 2 is skip-gram model architecture of Item2vec which maximizes the co occurrence probability among the items that appear within a window, w, in a Transaction T^u. Algorithm 2 iterates over all possible items that appear within the window w (lines 2-3). For each, we map each item i_j to its current representation vector ϕ (i_j) $\in R^d$. Given the representation of i_j , we would like to maximize the probability of its neighbors.(line 4). it iteratively tries to predict the context of ij and perform stochastic gradient descent adjusting the model parameters (weights) by minimizing a cost function computed on the predicted context and the true context.(SGD update).

Algorithm2: skip-gram Item2vec training model(ϕ, $\mathbf{T}^u$,**w)**

Input: T^{train}
Output: F = IR^{u*d},where t =| T^{train} |

Data: T
Read { train_data }
for each *transaction* T^u in T^{train} do
  for each *item* i_j *in* T^u do
    for each *item* $i_k \in T^u$[j − w : j + w] do
      J(ϕ) = −log Pr(i_k | $\phi(i_i)$)
      $\phi = \phi - \alpha * \partial J / \partial \phi$
    end
  end
end

5. Experiments

we train the Item2vec model using the item sequences. Here, each item i in each transaction T^u corresponds to a

word in sentences. we keep only the meaningful transactions T^u that have at least N items, and experimentally set N = 3. Our Item2vec model of recommender system takes sequence of items as input, and outputs a set of similar items given the input. In order to give the item candidates, similar items are ranked from highest to lowest score, and top-k items with the highest scores will be stored as the output.

For training our model we used the word2vec python package using Gensim which is an open-source vector space modeling and topic modeling toolkit, implemented in the Python programming language. It uses NumPy, SciPy and optionally Cython for performance.

Before training the model we first split the transactions T patterns into training patterns and testing patterns, 80% and 20% respectively. we then train our Item2vec model on the training patterns of item sequences. we used the skip-gram architecture to train the Item2vec model. The combination of the following parameters is considered: *size, window, negative, min-count, sample and iteration.* Our experimental setup is available at: https://github.com/Bereket123/Recommendation-based-on-sequence-

Parameter	Explanation
Size	Vector dimension
Window size (w)	Maximum number words/items of context
Negative	Number of "noise words/items" should be drawn (train faster)
Min-count	Items appear less than this min-count value is ignored
Iteration	Training epochs
Sample	Sub-sampling of frequent words/items

Table 1: Parameters explanation for training Item2vec model

we experimentally tune the parameter values and we then perform a grid search to find the optimal parameters.

The parameter setting for best performance of our recommender system is given in Table3 .

5.1. Dataset

We evaluate our recommender system algorithm on a transformed Online Retail dataset found at [22]. The dataset we used is a 10- core subset, i.e. every user bought in total at least 10 items and vice versa each item was bought by at least 10 users. The statistics of the dataset can be found in table 4

Parameter	Values
Size	[10, **15**, 25, 50, 100, 200]
Window	[1, **3,** 5, 8, 10, 15]
Negative	[10, 15, **20**, 25, 30, 35]
Sample	[0, 1e-2, 1e-3, **1e-4**, 1e-5, 1e-6, 1e-7]
Iteration	[1, 2, **3**, ..., 20]
Min-count	[1, 2, **3**, ..., 20]

Table 2: Parameter settings for training Item2vec model

Parameter	Values
Size	15
Window	3
Negative	20
Min-count	3
Iteration	3
Sample	1e-4

Table3:Best parameter setting

Dataset	users |U|	items |I|	Baskets (transactions T^u)	avg. baskets per user
Online retail	169,346	2603	541,909	3.2

Table 4: Characteristics of the datasets in our experiments in terms of number of users, items, baskets and avg. baskets per user

5.2. Evaluation Method

We test the learned model on the test set by removing the last items from each user's transactions(baskets T^u) of the test patterns and asking our learned model to make recommendation of top K items located closer to each items in the learned vector representation model for each user's remaining basket $T^u = \{ i_1 , i_2 , \ldots , i_{m-1} \}$ we measured the performance by looking at the number of hits within the top-K items that were recommended by our recommender system. A method similar to the one introduced by Deshpande and George (2004). The number of hits is the number of items in the testing data that were

also present in the top-K recommended items returned. If test size is the total number of transactions T^u in the test data the Accuracy of our model can be computed as:

$$Accuracy = hit/test\ size$$

In all of experiments we set K =10 as the number of items to be recommended by the recommender system. A hit of 1.0 indicates that the system was able to recommend the relevant item (i.e the last item removed from the test pattern), whereas 0.0 indicates that the system was not able to recommend the relevant item.

The general procedure of learning and prediction is described as follows

1) Load sequence (T)
2) Split T into train_patterns and Test_patterns
3) Train Item2vec model on train_patterns
   a) Construct vocabulary from train_patterns
   b) Learn vector representation of items
4) Test model on Test_patterns
   a) Discard sets of patterns T^u from Test_patterns of length less than 2
   b) Reduce the length of the remaining patterns by removing their last items
   c) Discard items from each of the remaining patterns that are not in the models vocabulary
   d) For each of the patterns in the reduced Test_patterns predict top K items located closer to each items.
   e) Evaluate accuracy. Check if predicted set of items contains last item.

Our Item2vec based recommender system achieved 44.46% accuracy for 108,381 users with the best settings shown above in table3. After that, we tune each parameter to inspect the best setting for each of them, and find the important parameters that affect recommenders performance. The experiment results are analyzed in Figure2. All the best settings for parameters result in the best performance (Accuracy), except min-count. If min-count value is increased, e.g., min-count=20 ,the accuracy will be higher. However, the item quantity will be significantly reduced. We therefore, set min-count value to 3 for keeping the balance between the number of items and the systems performance. Figure 2 shows the impact of parameters Item2vec model. Interestingly, min-count had little effect on our recommender system's accuracy, while *sample* and *size* are two most important parameters that control the performance.

All experiments were run on Intel i3 processor @ 2.60GHz with 4 cores and 4 GiB of memory.

We compared our results with conventional approaches, including item-similarity based similarity [24] method and matrix factorization [25]. The item similarity based model first computes the similarity between items using the obser-

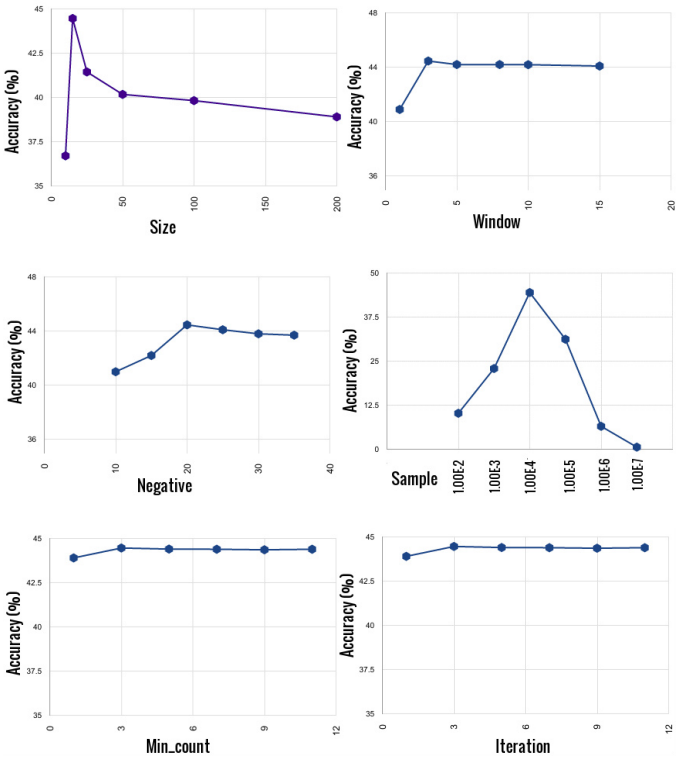


Fig. 2. Parameter optimization for Item2vec model

vations of users who have interacted with both items. Given a similarity between item i and j, it scores an item j for user u using a weighted average of the users previous observations. There are several choices of similarity function to use, e.g., Jaccard, cosine or pearson. In our case, we choose Jaccard to compute the item similarities. Matrix factorization method is another well-developed method in recommendation scenario. This model tries to learn latent factors for each user and item and then uses them to make recommendations. We compared the quality in terms of accuracy and the training time of our Item2vec recommender system with the two collaborative filtering based systems, item-similarity and matrix factorization methods in table 5. The three columns in Figure 3 show the accuracy of two collaborative filtering based systems, and our Item2vec system. It indicates that our Item2vec recommender system performed significantly better than those two collaborative filtering based recommender systems, with an accuracy of 44.46% compared to 8.35% and 19.59%Ȯur system has proved to be effective by significantly improving the accuracy score.

6. Parallel Implementation of Item2vec

Since our Item2vec model is a shallow neural network the accuracy depends heavily on the amount of the training data. Unfortunately, when the dataset size is huge, both the memory use and computational cost willl be prohibitively expensive. In this section, we propose to parallelize Item2vec algorithm distributing the work fairly among processors. we implemented a parallel version of Item2vec which distributes the workload and hence more scalable than a single-machine

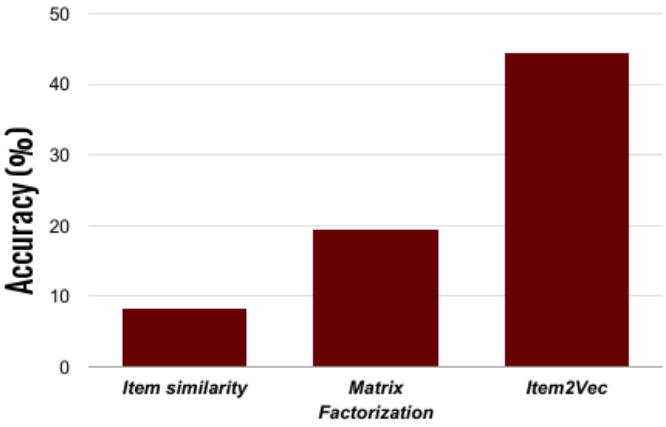


Fig. 3. Prediction accuracy of Item similarity, Matrix Factorization and Item2vec recommender system

System	Accuracy(%)	Training time
Item2vec	44.46	7.6 sec.
Item-similarity	8.35	0.9 sec.
Matrix Factorization	19.59	1.8 sec.

Table5: Comparison of Item similarity and Matrix Factorization systems with Item2vec recommender system

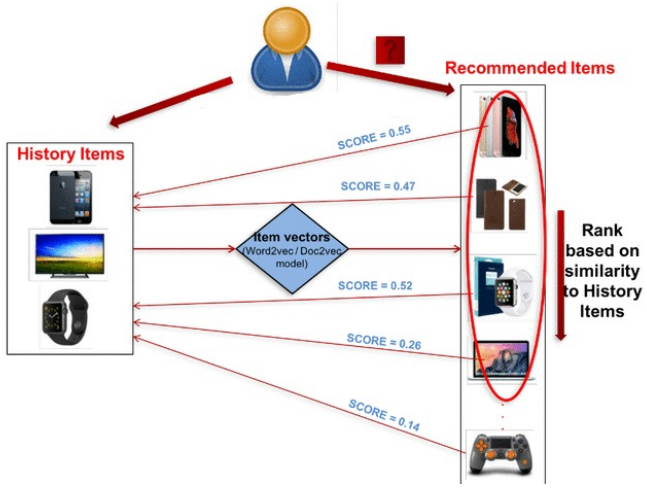


Fig. 4. Illustration of Item2vec system

implementation. All experiments have been executed using spark-2.1.0 by dividing jobs into smaller sets of stegs that can be parallelized and depend on each other (similar to the map and reduce stages in MapReduce). But before explaining about our parallel implementation of Item2vec let us show an overview of parallel computation and Apache Spark.

6.1. Parallel Computing

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. In parallel computing a problem is broken into discrete parts that can be solved concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different processors and an overall control and coordination mechanism is employed. The computational problem should be able to be broken apart into discrete pieces of work that can be solved simultaneously and execute multiple program instructions at any moment in time and also be solved in less time with multiple compute resources than with a single compute resource. The compute resources are typically a single computer with multiple processors/cores and an arbitrary number of such computers connected by a network.

6.1.2. Why Use Parallel Computing?

In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence. Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

Main Reasons

SAVE TIME AND/OR MONEY: In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. Parallel computers can be built from cheap, commodity components.

SOLVE LARGER / MORE COMPLEX PROBLEMS: Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory. For example: ”Grand Challenge Problems” requiring PetaFLOPS and PetaBytes of computing resources. And also Web search engines/databases processing millions of transactions every second. PROVIDE CONCURRENCY: A single compute resource can only do one thing at a time. Multiple compute resources can do many things simultaneously. For example: Collaborative Networks provide a global venue where people from around the world can meet and conduct work ”virtually”. TAKE ADVANTAGE OF NON-LOCAL RESOURCES: Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient. For example: SETI@home (setiathome.berkeley.edu) over 1.5 million users in nearly every country in the world. Source: www.boincsynergy.com/stats/ (June, 2015). And also Folding@home (folding.stanford.edu) uses over 160,000 computers globally (June, 2015) MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE: Modern computers, even laptops, are parallel in architecture with multiple processors/cores. Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc. In most cases, serial programs run on modern computers ”waste” potential computing power.

6.2. Apache Spark

Apache Spark is an open-source cluster-computing framework in which applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in main program (called the driver program). Specifically, to run on a cluster, the SparkContext can connect to several types of cluster managers (either Sparks own standalone cluster manager, Mesos or YARN), which allocate resources across applications. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for the application. Next, it sends the application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run.

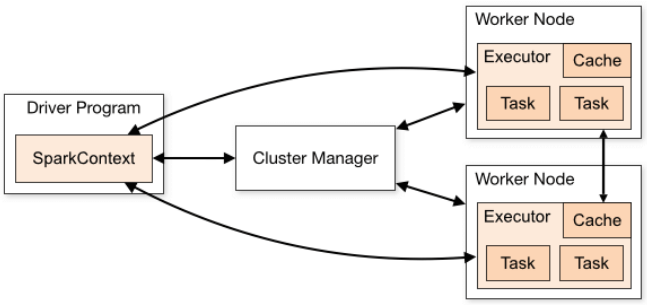


Fig. 5. Spark cluster computing master/slave architecture

Spark uses a master/slave architecture. As it is seen in the figure, it has one central coordinator (Driver) that communicates with many distributed workers (executors). The driver and each of the executors run in their own processes.
DRIVER: The driver is the process where the main method runs. First it converts the user program into tasks and after that it schedules the tasks on the executors.
EXECUTORS: Executors are worker nodes' processes in charge of running individual tasks in a given Spark job. They are launched at the beginning of a Spark application and typically run for the entire lifetime of an application. Once they have run the task they send the results to the driver. They also provide in-memory storage for RDDs that are cached by user programs through Block Manager.

APPLICATION EXECUTION FLOW

Applications can be submitted to a cluster of any type using the spark-submit script. When an application is submitted to a cluster with spark-submit this is what happens internally:

1) A standalone application starts and instantiates a SparkContext instance (and it is only then is possible to call the application a driver).
2) The driver program ask for resources to the cluster manager to launch executors.
3) The cluster manager launches executors.
4) The driver process runs through the user application. Depending on the actions and transformations over RDDs task are sent to executors.
5) Executors run the tasks and save the results.
6) If any worker crashes, its tasks will be sent to different executors to be processed again.
7) With SparkContext.stop() from the driver or if the main method exits/crashes all the executors will be terminated and the cluster resources will be released by the cluster manager.

6.3. Parallel Implementation approach

In the Proposed parallel implementation of Item2vec algorithm the first stage is preprocessing of the transaction dataset T by the master (a sequential operation). Which is a similar operation represented by algorithm 1. The operation at this stage is transformation of the transaction dataset and the output is two transformed datasets; training transaction dataset T^{train}, and testing transaction dataset T^{test}. The full execution for the parallel implementation of the Item2vec model training looks something like the following:

Procedure: Full execution:
Broadcast (Initialization mapper)
for i : 1 to number of iterations
Broadcast (Training mapper)
end

During training the master broadcasts the transformed training transaction dataset T^{train} to the Executors/slaves (worker nodes). Each executor gets a portion of T^{train}. We have two map tasks to be executed in parallel by executors during training. The first parallel operation of the training phase at each worker node is random vector representation of the items. Taking each transaction T^u in T^{train} it assigns initial vector representation $\phi_i(\ i\ ) \in R^d$ for each item in each transaction. The pseudo code for the first map stage looks like the following:

Procedure:
Initialization Mapper (transaction_index, transaction)
for item i in transaction T^u
emit(item, initial vector representation $\phi_i(\ i\ ) \in R^d$

The second map stage is a training mapper which takes the output of the initialization mapper as its input. Given initial vector representation $\phi_i(\ i\ ) \in R^d$. Here Item2vec constructs a vocabulary of items by way of predicting the context(similarity) it iteratively tries to predict the context of i and perform stochastic gradient descent(SGD) adjusting the model parameters (weights) by minimizing a cost function computed on the predicted context and the true context. SGD update updates initial feature vectors for items, typically of several hundred dimensions, with each unique item in the transactions being assigned a corresponding vector in the space such that Items that share common contexts(similar items) in the transaction database are located in close proximity to one another in the space. Item2vec turns (items) into a numerical form that deep nets

can understand. The pseudo code for the training mapper stage looks like the following:

Procedure:
Training mapper (item, initial vector representation ϕ_i(i) $\in$ R^d
    for each item i do
        SGD update ϕ_i(i) $\in$ R^d.
        emit(item, SGD updated ϕ(i))

Testing Phase:

In the parallel implementation of the testing phase of Item2vec the first stage is broadcasting the transformed testing transaction dataset T^{test} containing list of transactions T^u to Executors/slaves (worker nodes) by the master. The parallel operation executed at each worker node during the testing phase is the following: first taking each transaction T^u in T^{test} we split each transaction into the first N items and last item. In other words here we Reduce the length of each transaction T^u in T^{test} by removing their last item and we then make a prediction of top K similar items to the reduced transaction from the trained model. There are several choices of similarity function to use for predicting nearest neighbours, e.g., Jaccard, cosine or pearson. In our case, we choose cosine to compute the item similarities. After prediction is made we implement a hit counter to compare prediction and last item. A hit of 1.0 indicates that the system was able to recommend the relevant item (i.e the last item removed from the test transaction T^u), whereas 0.0 indicates that the system was not able to recommend the relevant item. The pseudo code for this stage of test phase looks like the following:

(N) First N items , (T^u_id) transaction index, transaction(T^u), Top K items (top K)

Input: T^{test},
Procedure: Test Mapper (T^u_id, T^u)
    (N, last item) = split(T^u)
    Pred = predict (N, top K)
    Hit = compare(Pred, last item)
        emit ((T^u_id), Hit)

After prediction is made and hit count is generated at each worker nodes parallely the next step is aggregation of hit counts and checking accuracy.. The master will compute the prediction accuracy in (%) by collecting and summing up the hit counts from each executors/slaves(worker nodes) and dividing it with the test size. The pseudo code for this stage looks like the following:

Procedure: Hit count ((T^u_id), Hit)
    Hit = collect from test mappers
for test transaction T^{test} do
    Accuracy = Hit/test size
    return Accuracy
end

6.4. Parallel implementation Experiment

Parallel experiments have been executed using spark-2.1.0, and we used 2 nodes, each equipped with a 4-core Intel i5 processor @ 2.60GHz,4 GB of memory.

Dataset

For the experiment in this section we synthetically generated a transaction history dataset with 5 million transaction., 24,010 unique items and 1,562,495 users. And the generated dataset have similar distribution with the real online retail dataset we used for the experiment in section 5. Figure 6, illustrates the distribution of the real and the synthetically generated datasets.

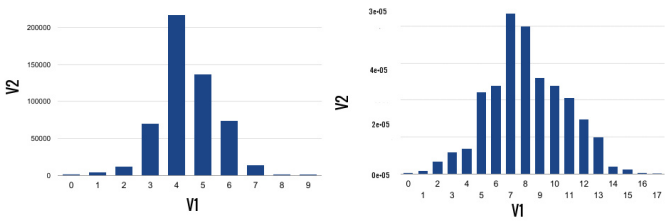


Fig. 6. Distribution of the real and the synthetically generated datasets.

Dataset	*Synthetic*
users |U|	*1,562,495*
items |I|	*24,010*
Baskets (transactions T^u)	*5,000,000*
avg. baskets per user	*29.5*

Table 6: Characteristics of the synthetically generated datasets in our experiment in terms of number of users, items, baskets and avg. baskets per user

We used the parameters in table 7 for training the model. We did both serial and parallel experiments using the synthetically generated dataset. We achieved an accuracy of 61.03 % and we observed that the the parallel implementation provides faster training time. Since our Item2vec model is a shallow neural network the accuracy depends heavily on the

Parameter	Values
Size	15
Window	3
Negative	20
Min-count	3
Iteration	3

Table7:Best parameter setting

Implementation	*Accuracy(%)*	*Training time(sec)*
serial	*61.03*	*886.0*
parallel	*61.03*	*330.9*

Table8: Comparison of serial and parallel version

amount of the training data. We also observed a significant improvement in the accuracy with a larger dataset.

CONCLUSION

In this project we focused on recommendation of a ranked list of items for users based on what items the user has used in the past and on what the other users have done. We developed the distributed representation-based recommender systems and we applied this approach to an online retail dataset with 541,909 transactions T^u, 169,346 users $|U|$ and 2603 items $|I|$. Experiment results illustrated that our proposed systems achieved an accuracy of 44.46% for recommending items to users in testing data and outperformed collaborative filtering methods. Then we applied both serial and parallel version of our Item2vec approach to a synthetically generated dataset with 5,000,000 transactions T^u, 1,562,495 users $|U|$ and 24,010 items $|I|$. Experiment results illustrated that our proposed systems achieved an accuracy of 61.03% Ẇe also observed that the parallel implementation scales the the training time.

REFERENCES

[1] Haiyun Lu Recommendations Based on Purchase Patterns International Journal of Machine Learning and Computing, Vol. 4, No. 6, December 2014

[2] Sonali Khandagale, Sneha Mallade, Krunali Kharat and Vishkha Food Recommendation System using Sequential Pattern Mining Imperial Journal of Interdisciplinary Research (IJIR) 2016

[3] Pengfei Wang, J. Guo, Y. Lan, Jun Xu, S. Wan and X. Cheng Learning Hierarchical Representation Model for Next basket Recommendation

[4] Ms. Manali K. Patil a, Miss A. A. Manjrekar, * Recommender Systems Using Item Sequences and Ranking Mechanisms

[5] Yann LeCun, Yoshua Bengio and Geoffrey Hinton Deep learning

[6] Jrgen Schmidhuber Deep learning in neural networks: An overview The Swiss AI Lab IDSIA, Istituto Dalle Molle di Studi sullIntelligenza Artificiale, University of Lugano and SUPSI, Galleria 2, 6928 Manno-Lugano, Switzerland

[7] Tomas Mikolov, Ilya Sutskever, Greg Corrado, Jeffrey Dean and Kai Chen Distributed Representation of words and phrases and their compositionality

[8] Greg Linden, Brent Smith, and Jeremy York Amazon .com recommendation: Item-to-Item collaborative filtering

[9] John S. Breese; David Heckerman and Carl Kadie (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (UAI'98).

[10] Peter Brusilovsky (2007). The Adaptive Web. p. 325. ISBN 978-3-540-72078-2.

[11] Adomavicius, G.; Tuzhilin, A. (June 2005).: "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions". IEEE Transactions on Knowledge and Data Engineering. 17 (6): 734749. doi:10.1109/TKDE.2005.99.

[12] Sarwar Badrul et al.: Item-based collaborative filtering recommendation aligorithms. Proceedings of the 10th international conference on World Wide Web. ACM, 2001.

[13] S. Chen J. L. Moore, D. Turnbull, and T. Joachims: Playlist prediction via metric embedding. In proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 12, pages 714-722, New York, NY, USA, 2012. ACM.

[14] R. Srikant and R. Agrawal. Mining sequential Patterns: Generalizations and performance improvements. In proceedings of the 5th International Conference on extending Database Technology, EDBT 96 pages 317, London, UK, UK, 1996. Springer-Verlag.

[15] Sonali Khandagale, Sneha Mallade, Krunali Kharat and Vishakha Bansode: Food Recommendation System Using Sequential Pattern Mining. Imperial Journal of Interdisciplinary Research

[16] Chumki Basu, Haym Hirsh and William Cohen: Recommendation as Classification: Using Social and Content-Based Information in Recommendation. AAAI-98 Proceedings. 1998

[17] R. J. Mooney and L. Roy, Content-based book recommending using learning for text categorization, in Proc.ACM SIGIR 99 Workshop on Recommender Systems: Algorithms and Evaluation, Berkeley,CA, 1999.

[18] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, Grouplens: An Open Architecture for Collaborative Filtering of Netnews, in Proc. the ACM 1994 Conference on Computer Supported Cooperative Work, 1994.

[19] S. K. Jong-Seok Lee, Chi-Hyuck JuncnJaewook Lee. Classification-based collaborative filtering using market basket data. Expert Systems with Applications, 2005.

[20] Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." Proceedings of the 10th international conference on World Wide Web. ACM, 2001.

[21] Wolfgang Woerndl1, Christian Schueller2, Rolf Wojtech: A Hybrid Recommender System for Context-aware Recommendations of Mobile Applications

[22] .https://archive.ics.uci.edu/ml/datasets/Online+Retail

[23] G. Shani, D. Heckerman, and R. Brafman, An MDP-based recommender system, Journal of Machine Learning Research, 6, pp. 1265-1295, 2005.

[24] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. Springer US, 2011.

[25] Koren, Yehuda, Robert Bell and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems.Computer Volume: 42, Issue: 8 (2009): 30-37.

[26] .https://computing.llnl.gov/tutorials/parallelcomp