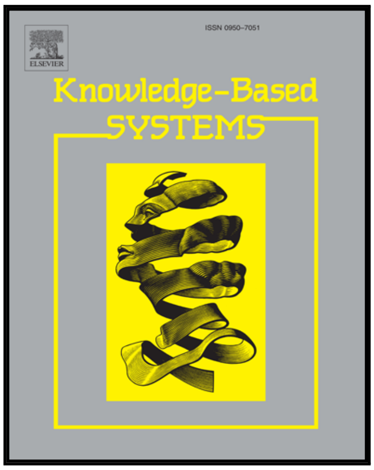
On the predictive analysis of behavioral massive job data using embedded clustering and deep recurrent neural networks

Citation for published version:
Benabderrahmane, SA, Mellouli, N & Lamolle, M 2018, 'On the predictive analysis of behavioral massive job data using embedded clustering and deep recurrent neural networks', *Knowledge-Based Systems*, vol. 151, pp. 95-113. https://doi.org/10.1016/j.knosys.2018.03.025

Digital Object Identifier (DOI):
10.1016/j.knosys.2018.03.025

Published In:
Knowledge-Based Systems

Accepted Manuscript



On the Predictive Analysis of Behavioral Massive Job Data Using Embedded Clustering and Deep Recurrent Neural Networks

Sidahmed Benabderrahmane, Nedra Mellouli, Myriam Lamolle

PII: S0950-7051(18)30157-6
DOI: 10.1016/j.knosys.2018.03.025
Reference: KNOSYS 4274

To appear in: *Knowledge-Based Systems*

Received date: 9 November 2017
Revised date: 16 March 2018
Accepted date: 17 March 2018

Please cite this article as: Sidahmed Benabderrahmane, Nedra Mellouli, Myriam Lamolle, On the Predictive Analysis of Behavioral Massive Job Data Using Embedded Clustering and Deep Recurrent Neural Networks, *Knowledge-Based Systems* (2018), doi: 10.1016/j.knosys.2018.03.025

On the Predictive Analysis of Behavioral Massive Job Data Using Embedded Clustering and Deep Recurrent Neural Networks.

Sidahmed Benabderrahmane [a], Nedra Mellouli [b], Myriam Lamolle [b].

(a): The University of Edinburgh, School of Informatics, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom.
sidahmed.benabderrahmane@gmail.com, Tel: 0044 131 651 5661
(b): LIASD (EA 4383), University of Paris 8 Saint-Denis, IUT de Montreuil, France.

Abstract

The recent proliferation of social networks as a main source of information and interaction has led to a huge expansion of automatic e-recruitment systems and by consequence the multiplication of web channels (job boards) that are dedicated to job offers disseminating. In a strategic and economic context where cost control is fundamental, it has become necessary to identify the relevant job board for a given new job offer has become necessary. The purpose of this work is to present the recent results that we have obtained on a new job board recommendation system that is a decision-making tool intended to guide recruiters while they are posting a job on the Internet. Firstly, the Doc2Vec embedded representation is used to analyse the textual content of the job offers, then the job applicant clickstreams history on various job boards are stored in a large learning database, and then represented as time series. Secondly, a deep neural network architecture is used to predict future values of the clicks on the job boards. Third, and in parallel, dimensionality reduction techniques are used to transform the clicks numerical time series into temporal symbolic sequences. Forecasting algorithms are then used to predict future symbols for each sequence. Finally, a list of top ranked job boards are kept by maximizing the clickstreams forecasting in both representations. Our experiments are tested on a real dataset, coming from a job-posting database of an industrial partner. The promising results have shown that using deep learning, the recommendation system outperforms standard multivariate models.

Keywords: Recommender System, Time Series, Deep Learning, Symbolic Sequences, Big Data, E-recruitment.

1. Introduction

This work concerns the recruitment market that is composed of three main players: the recruiter, who wishes to find the most suitable candidate with a desired profile; the candidate, looking for a job adapted to her/his profile and her/his professional perspectives; and the intermediaries, that mediate the relationship between the first two actors. Intermediaries in the labour market are the recruitment agencies, the temporary employment agencies, the human resources (HR) communication agencies, the press, the institutional networks, etc. Over the two last decades, another kind of intermediary appeared: the job boards (or job search websites). More formally, many job boards allow the dissemination of the job offers on different Web platforms (University websites, job social networks, business career websites, etc.). Since the arrival of the Internet, the use of web job boards has increased drastically. Between 2006 and 2009, the proportion of managerial positions that were diffused in the Internet has increased by 16%. In 2009, the Internet has been proved to be an essential medium for recruitment, with 82% of employment published therein [1]. Expanding the Internet media for recruitment has led to a multiplication of channels to find candidates. Current e-recruitment systems consider only a part of the recruitment process, concentrating on matching job offers with CVs. However, the selection of the most appropriate job board regarding an offer is also very important for the optimization of this fully digital recruitment process. This is our main contribution, in the SONAR research project (Sourcing and Automated Recruitment[1]). At the moment, various questions arise concerning the selection criteria for the relevance of a job board. For example, is the job board relevant if the numbers of offers are increasing in it? Or, simply if the number of visits and/or the number of clicks to view the offers by potential candidates tend to grow compared to those observed in the past? Our main goal is to provide a tool which can help recruiters to (i) select the most relevant job boards for a new job offer, (ii) diffuse more effectively job offers, that is to say at the right place at the right time, (iii) provide tools to connect candidates and job offers automatically.

In this paper, we propose *Deep4Job*, a job offer recommendation system in which the main contributions concern: (a) the representation of the job offers textual documents in a new embedded space model that allows extracting latent

[1] http://sonar-project.com

topics and for classifying business categories; (b) the consideration of contextual information such as the job applicants temporal behaviour through their clicks on different dissemination links as time series data; (c) by showing how interesting is the use of deep neural networks instead of the probabilistic models, to predict future clicks values; finally (d) by also proposing the use of symbolic temporal sequences that are obtained from the clicks time series using dimensionality reduction methods to analyse the trajectories of the job applicants. These new contributions were evaluated on a real job offers database provided by an industrial partner, as illustrated in 1. The results seem to be very interesting compared to the state of the art collaborative filtering analysis.

In the next section, we will firstly give a global overview on the existing recommendation systems with their advantages and limits, and afterward we will introduce the general architecture of our proposed Deep4Job system.

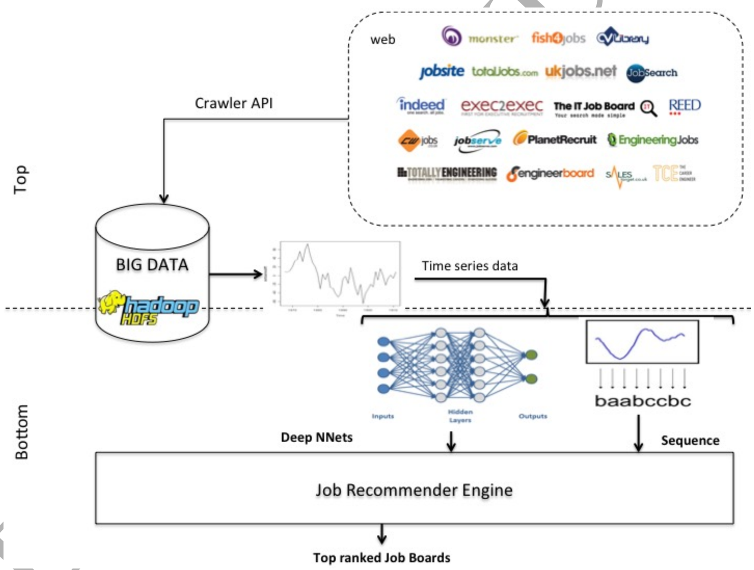


Figure 1: Top: The global architecture of the proposed recommender system with the used database. Job offers and their job boards are crawled from the Internet with appropriate APIs. Both textual content of the postings as well as the users' clickstreams are available. Bottom: Clickstreams data are later represented as time series, and new forecasting algorithms are used to predict future clicks values on each job board.

2. Related Work

2.1. Highlights on Recommender Systems

During the past decade, the variety and number of products and services provided by companies has increased dramatically. Companies produce a large number of products to meet the needs of customers. Although this gives more options to customers, it makes it harder for them to process the large amount of information provided by companies. Recommender systems are designed to help customers by introducing products or services. These products and services are likely preferred by users, based on their preferences, needs, and purchase history.
Nowadays, many people use recommender systems in their daily life for on-line shopping, reading articles, or watching movies. Usually, a recommender system recommends items either by predicting ratings or by providing a ranked list of items for each user.
Roughly speaking, there are three types of recommendation systems (excluding simple ranking approach) [2, 3, 4, 5, 6, 7]: Content-based (CB) recommendation, Collaborative filtering (CF), and Hybrid models.

A content-based recommendation system is a regression problem in which we try to make a user-to-item rating prediction using the content of items as features. On the other hand, for a collaborative filtering based recommendation system, we usually do not know the content of features in advance, and by using the similarity between different users (i.e. users may give similar ratings to the same items) and the similarity between items (similar movies may be given similar ratings by the users), we learn the latent features and make predictions on user-to-item ratings at the same time. Therefore, after we learn the features of the items, we can measure the similarity between items and recommend the most similar items to users based on their previous usage information. Content-based and collaborative filtering recommendation systems were the state of the art for the past 10 years ago. Apparently, there are many different models and algorithms to improve the prediction performance. For instance, for the case in which we do not have user-to-item rating information in advance, we can use the so-called implicit matrix factorization and replace the user-to-item ratings with some preference and confidence measures such as how many times the users click on the corresponding items to perform collaborative filtering. Furthermore, we can also combine content-based and collaborative filtering methods to utilize content as "side information" to improve the prediction performance. This hybrid approach is usually implemented by a "Learning to Rank" algorithm.

Other recent works considered the problems of data heterogeneity, data sparsity, cold-start initialization, and items' dynamic evolution process [8, 9, 10]

2.2. Deep Learning in Recommender Systems

2.2.1. A gentle introduction to deep learning

Deep learning is a special field of machine learning, putting the focus on the representation from the data, and adding successive learning layers to increase the meaningful representation of the input data. The "deep" notion in deep learning is not a reference to any kind of deeper understanding, rather it represents the great number of successive layers of neural representations [11], i.e. how many layers are used in the model for a suitable representation of data in the feature space. Other frequent names are also used to designate deep learning, such as: layered representations learning, or hierarchical representations learning [12]. In deep learning, the learning layers are trained via neural networks. Similarly to neurobiology, the learning process is inspired by human brain understanding. However, one should be aware of the pop-sciences articles, which claim that deep learning works perfectly like our brains. Indeed this is the usual approximation made by newcomers to the field [11]. Figure 2 gives a general framework of a deep neural network. Data samples are used as input of the learning process. They are then transformed through transformation layers that could be either embedding layers for textual data analysis, or convolutional layers for image processing, or recurrent layers for temporal and sequence data processing [13]. Finally, the transformed data are learned with deep (a large number of) fully connected layers to produce the output of the network.

By observing Fig. 2, we can see that deep learning is a complex process of multi-stage data transformation, with the goal of mapping inputs to targets, which is done by varying the weights of the network. The technical complexity resides in the huge number of parameters that are modified during the learning process.

2.2.2. Deep learning application domains

In the few years since 2010, deep learning has revolutionized the machine learning world, with very interesting results particularly in computer vision [14, 15, 16, 17] or Natural Language Processing (NLP) [18, 19, 20, 21, 22]. Breakthroughs have been observed in complex artificial intelligence problems such as image classification for digit recognition [23], handwriting transcription [24], signal processing [25], web mining [26, 27], or even autonomous systems [28], etc. Actually, deep learning is affecting everything from health-care to transportation to manufacturing, and more. Companies are turning to deep learning to solve

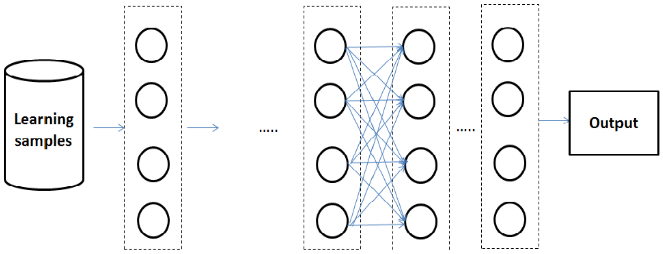


Figure 2: A deep neural network general model. Transformation layers could be either embedding layers for textual data, or convolutional layers for image processing or recurrent layers for temporal data processing. Transformed data are learned with deep fully connected layers.

hard problems, like speech recognition, object recognition, and machine translation. One of the most impressive achievements in 2017 was AlphaGo beating the best Go player in the world. With the victory, Go joins checkers, chess, and Jeopardy as games in which machines have defeated human champions [29].

2.2.3. *Deep learning recommender examples*

Recently, deep learning technologies have seen considerable growth with many cases of application in the area of recommendation. Donghyun et al. in [30] proposed a context-aware recommendation model, in which a convolutional matrix factorization (ConvMF) integrates convolutional neural network (CNN) into probabilistic matrix factorization (PMF). The system captures contextual information of documents and enhances the rating prediction accuracy. Yin Zheng et al. in 2016 [31] proposed CF-NADE, a neural autoregressive architecture for collaborative filtering tasks, which is inspired by the Restricted Boltzmann Machine (RBM) based CF model and the Neural Autoregressive Distribution Estimator (NADE). This method is a tractable distribution estimator for high dimensional rating binary vectors. It tackles sparsity of the rating matrix. The performance of CF-NADE was tested on 3 real world benchmarks: MovieLens 1M, MovieLens 10M and Netflix database. Young-Jun et al. in their work presented in [32] proposed a song recommendation system that is based on language modeling and collaborative filtering combined with recurrent neural networks RNNs, to take into account the user's interaction and their contextual information for making the recommendation efficient. Strub et al. in [33] presented a recommender system using a stacked denoising Autoencoders Neural Network in which a loss function was adapted to input data with missing values. The main objective of their work was to alleviate the cold start problem by integrating side information, because when very little information is available on a user or item, Collaborative Filtering will have difficulties in infering its ratings. Note that, these models are not full deep architectures but one hidden layered neural architecture for CF. Van den Oord et al. in [34] tackled the problem of sound recommendation in social networks. They proposed to use a latent factor model for recommendation, and predict the latent factors from audio when they cannot be obtained from usage data. Their method used deep convolutional neural networks on the Million Song Dataset, for extracting local features from audio signals and aggregating them into a bag-of-words (BoW) representation. In [35], Almahairi et al. presented a work in which they have shown how a collaborative filter-based recommender system can be improved by incorporating side information, such as natural language reviews, as a way of regularizing the derived product representations. Instead of using a classical topic modeling of reviews (such as latent Dirichlet allocation (LDA)), the models they proposed are based on neural networks. Zheng et al. in [36] proposed DepCoNN, a Deep Cooperative Neural Network, to learn item properties and user behaviours jointly from review text. The proposed model consists of two parallel neural networks coupled in the last layers. One of the networks focuses on learning user behaviours exploiting reviews written by the user, and the other one learns item properties from the reviews written for the item. A shared layer is introduced on the top to couple these two networks together. The model was tested on several databases, for instance Yelp, which is a large-scale dataset consisting of restaurant reviews, and Amazon product reviews. Covington et al. in [37] proposed a sophisticated video recommender system on YouTube. The user preferences and search history are embedded into a latent space and then fed into the deep neural networks with additional side information such as demographics, geography, etc. The model generates a few best recommendations by assigning a score to each video according to a desired objective function using a rich set of features describing the video and user. The highest scoring videos are presented to the user, ranked by their score. The searched tokens on the platform as well as the watched videos are represented in an embedding space, and used later in a deep neural network to recommend new videos.

2.2.4. E-recruitment recommender systems

Recommender systems in automatic recruitment platforms allow HR agents to advertise a job offer on the relevant job board, which may attract the best candidates in a small temporal period. Actually, there are several thousands of dedicated job boards for broadcasting job offers, for instance Monster, indeed, iquesta, jobsite, parisjob, etc. Some of them charge subscription fee. Consequently, searching and identifying the best job board for a new job offer can be considered as a challenging and hard task. To this aim, several recommendation systems have been presented in the literature. These systems are generally classified into three main categories: textual recommendation systems [38], collaborative filtering recommendation systems, and hybrid recommendation systems [39],[40].

In the textual-based recommendation systems, the content of the job offers are analysed with the information provided by users to identify the semantic content. To that aim, two kinds of semantic analysis exist: the approaches based on ontologies [41] and those based on text mining [1].

Whatever the approach used in the purely textual recommendation systems, they have the weakness since they require manual annotations by the recruiters and the candidates to describe both the job offers and the CVs. Nevertheless, the volume and the complexity of the processed data are quite large and do require the use of highly optimized algorithms.

Collaborative filtering systems are based on the analysis of the opinions of a group of users. Their opinions are considered similar to those of an active identified user. These recommendation systems can target CVs only from related information (such as the title). The use of items certainly reduces the mass of processed data but with a loss of precision. As for hybrid systems, they combine the two previously mentioned categories.

In this context, other works focused on the analysis of the impact of the implicit relevance feedback on job recommender system. In particular, Hutterer et al. in [42] proposed some methods for monitoring and integrating the feedback of the users. Their project mainly focused on the Austrian job boards for which the recommendation system was designed. Basically the model remains simple and deeply dependent on the category of the jobs. In the same spirit, authors in [43] explore a specific approach to employ implicit negative feedback and assess whether it can be used to improve recommendation quality.

Most of these recommendation systems could be improved if the temporal information related to the job boards were taken into account. To that aim, we propose in this work, *Deep4Job*, a deep learning job offers recommender system, in which we are considering both temporal information relative to the dissemination and the clicks on job offers, as well as their textual content. We would like to show the importance of the temporal aspect of the job offers' dissemination process on different job boards, by creating a robust predictive model based on the quantity of the clicks on the job offers' URLs by job applicants, which represent their true behaviour on the web. We suggest representing this variation of the clicks, with time series data, for highlighting the trends and the seasonality in the recruitment process. The textual content of the job offers are used to discover latent semantic topics, using deep learning, word embedding and machine learning clustering. The time series data are used as learning samples to train a model for predicting future behavior of job applicants to support the recommender system. The forecasting is done with deep learning Long Short Term Memory neural networks (LSTM) [44]. A complementary solution suggests representing the numerical time series data, as temporal symbolic sequences, using efficient dimensionality reduction methods [45, 46]. The main interest of these transformations is the exploitation of robust symbolic data mining and natural language processing techniques to predict future symbols in a sequence. In the following section we will show the global architecture of *Deep4Job* recommender system, and present the learning database, as well as the global representation of our data. Section 4 presents the word embedding model that is used to discover potential projections and homogeneous clusters among the job offer textual documents. Section 5 presents the deep learning architecture that is used to forecast future click values on the numerical time series data. Section 6 presents the time series symbolic encoding approaches, followed by the deep learning model that is used to forecast future click values on the symbolic sequences. In section 7 we present the series of results that we have obtained when evaluating the model. Finally section 8 concludes this work with discussions and future perspectives.

3. Deep4Job: Using word embedding, Deep Learning and Temporal Sequences for Job Offers Recommendation

3.1. Project context and description of the Big Database

This work is the result of our participation in a FUI[2] project called SONAR (Sourcing and Automated Recruitment[3]), with an industrial partner (MultiPosting [4]) that is a leader in the French job market, which has provided us a big archive of job offers, that were disseminated in different job board websites, and also the relative quantity of their visits (clicks) by users (job applicants). The job boards in

[2]financed by the French government's FUI program

[3]http://sonar-project.com

[4]http://www.multiposting.fr/

this database DB are different and have multiple categories (social networks, specific to a category of business, with charge, with subscription, ... etc.). The heterogeneous big database saves the job offers and their relative job boards in both relational and NoSql schemas in a Hadoop cluster. The data concern backup archive of job offers, which were disseminated on different job boards (i.e. the textual content of the offers as json entries in a MongoDB), and also the relative quantity of their visits and clicks by users as a relational database (DB). The recorded data concern also candidates and their relative profiles on social networks (LinkedIn, Facebook, ...). The job boards in this archive are different and have multiple categories: social networks (e.g. LinkedIn), specific to a category of business (e.g. www.lesjeudis.com for IT jobs), free or with subscription (e.g. www.keljob.com), specific to a region (e.g. www.regionsjob.com), etc. The archives represent more than a six-year follow-up of data, that were scrapped from the Internet, and contain about ten thousand of job boards and more than three million of job offers and their daily relative clicks in these job boards, plus social networks posts, altogether making more than 3 TB of disk size. Each job board in our DB receives a lot of posting job offers each day. Fig. 3 gives an example of the top 10 most important job boards and their relative quantity of job offers which they disseminate. As illustrated, we have more than 1,6 million job ads from Facebook, approximately 1,2 million from Work4Us, about 600 thousand ads from Oodle, etc.
The global architecture of the proposed recommender system is illustrated in figure 4. The series of information in the database concern the textual content of the job offers and the temporal information of the job applicants behaviour. Thus, the first step in the system concerns the preparation and the formatting of these heterogeneous data. As a second step, we will use embedded layers of deep neural networks to represent the textual job offer documents in a sub dimensional word embedding space. Then we will apply a clustering procedure to discover similar classes among this representation of the job offers. After that, we consider each cluster of job offers separately, and create clicks time series data that can also be transformed to symbolic sequences using two dimensionality reduction techniques. Finally, forecasting algorithms are used to predict future clicks on the job offers, allowing the system to select the job boards in which the expected clicks can be maximized. All these general steps will be presented in detail in the next sections.

3.2. Job Offers Textual Representation

Each Job offer document in our database is represented as a list of structured items that includes the title of the job, the description, the required skills, the lo-

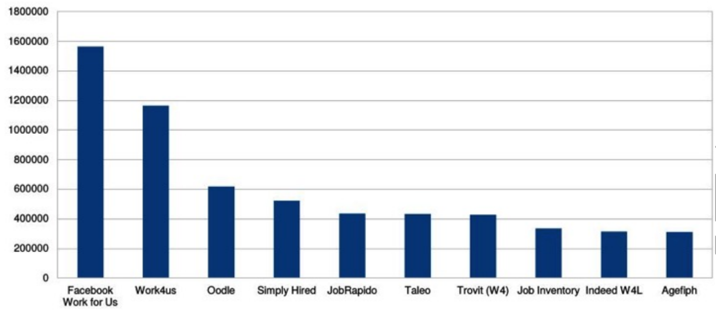


Figure 3: Example of some important job boards and their relative quantity of job offers which they disseminate.

cation, the salary, and so on. Each item is then transformed as a vector of frequent terms it contains. In addition, we have a job classification vocabulary, which is given by a public French organization that is called ROME code [5]. Therefore, it is necessary to represent these data adequately to process them with neural networks.

Text is one of the most widespread forms of sequence data. It can be understood either as a sequence of characters, or a sequence of words, albeit it is most common to work at the level of words. The deep learning sequence processing models that we will use to process the job offers, are able to leverage text to produce a basic form of natural language understanding, sufficient for applications ranging from document classification, sentiment analysis, author identification, or even question answering (in a constrained context) [11]. Deep learning for natural language processing is simply pattern recognition applied to words, sentences, and paragraphs, in much the same way that computer vision is simply pattern recognition applied to pixels. Like all other neural networks, deep learning models do not take as input raw text; they only work with numerical tensors. The Vector-

[5]http://www.pole-emploi.fr/candidat/le-code-rome-et-les-fiches-metiers-@/article.jspz?id=60702

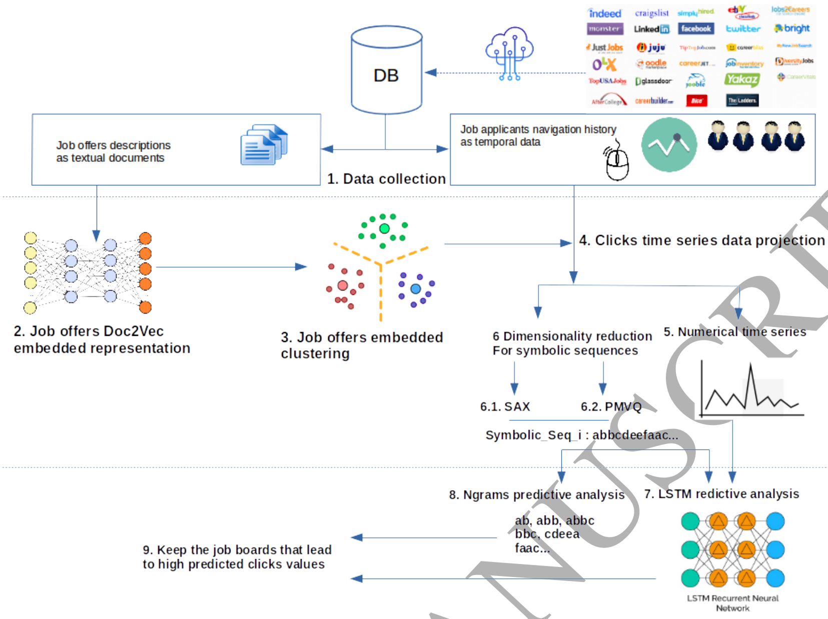


Figure 4: The global architecture of the job boards recommender system.

ization of text is the process of transforming text into numeric tensors. This can be done in multiple ways: (i) by segmenting text into words, and transforming each word into a vector; (ii) by segmenting text into characters, and transforming each character into a vector; (iii) by extracting ”n-grams” of words or characters, and transforming each n-gram into a vector. ”N-grams” are overlapping groups of multiple consecutive words or characters. Collectively, the different units into which one can break down text (words, characters or n-grams) are called ”tokens”, and breaking down text into such tokens is called ”tokenization”. All text vectorization processes consist in applying some tokenization scheme, then associating numeric vectors with the generated tokens. These vectors, packed into sequence tensors, are what gets fed into deep neural networks. There are multiple ways to associate a vector to a token. In this work we have used two major ones: one-hot encoding of tokens, and token embeddings (typically used exclusively for words, and generally called ”word embeddings”) [11, 47]. In the remainder of this paper, these techniques will be explained and we will show concretely how to use them to go from raw text to a tensor that we can send to the Keras API for deep network learning.

3.3. Clickstreams Representation with Time Series

The job offers in our DB are periodically broadcast in one or more job boards on a given date. An offer disseminated in a job board has a finite life cycle. In such temporal periods, the number of clicked links of the job offers in different job boards can be easily known. Therefore, the daily number of clicks associated within an offer and job boards is available. This number can be known on different time scales: weekly, monthly, semi-annually, or even annually. We denote by T, the period or the time scale associated to the considered number of clicks. To formulate such data representation, in particular the number of clicks, we consider a job board, noted JB, as a set of offers o_j on a given period T: $JB_T = \bigcup o_j$ for $j = 1, ..., p$

For each job board, we then introduce a ratio X^{JB_T} calculated as the total number of relative clicks of offers in this job board in a period T: $X^{JB_T} = \frac{nb.click}{|JB_T|}$

In the following of this paper, we consider T as a discrete interval $[1, N]$. Since the relative clicks are numerical values, we can consider the ratio $X_t^{JB_T}$ as a temporal observation of the clicks given at the time t. Having a series of observations $X_1^{JB_T}, X_2^{JB_T}, ..., X_N^{JB_T}$ on a fixed period T, we propose the definition of previsions on a date N with a time series of observations, to estimate $\hat{X}^{JB_T}(N, h)$ on future dates within a given horizon h.

The objectives of the temporal analysis in our study are multiple. For instance, it concerns the prevision of future realization of a random variable X^{JB_T} using the previously observed values $X_1^{JB_T}, X_2^{JB_T}, X_N^{JB_T}$ for each job board JB. To that aim, we will use univariate time series only, and we notice the variable X^{JB_T} by x_t which is observed at time t. Fig. 5 gives an example of a time series of a given job board, where values x_t are the daily clicks ratios of all job offers which were disseminated in this job board, between 2008 untill 2014 (2190 days, i.e. the length of the series).

3.4. Deep4Job Recommendation Algorithm

The proposed recommender system *Deep4Job*, has two major stages (see Fig. 6), namely (i) learning the predictive model phase (the left frame), and (ii) the online recommendation phase (execution, in the right frame of Fig. 6). During the learning process, there are two main steps. Firstly, job offer documents are represented in a sub-dimensional space for topics discovery and business classification. Embedding deep networks are used to train the neuronal model and represent the documents in an embedded space. Then the projected job offers in this space are classified in order to regroup similar job offers on the basis of their textual content,

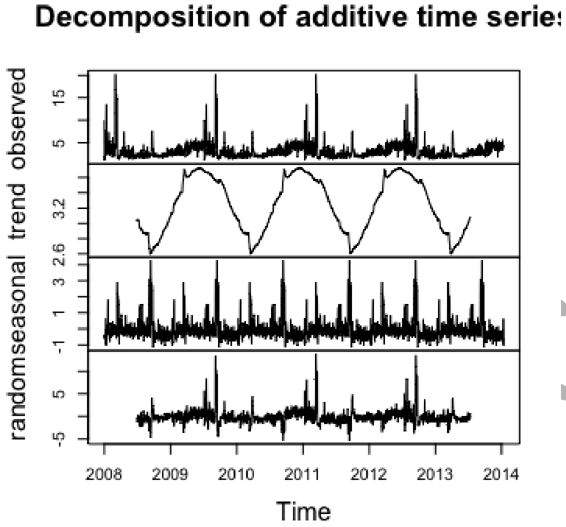


Figure 5: Example of the variation of the users clickstreams on some posting job offers which were disseminated in a job board.

and the similarity between the embedded vectors. The idea is to create a topology of job offer classes, that belong to similar job categories. The exact number of clusters can be obtained using the state of the art agglomerative clustering optimization techniques (e.g. Silhouette Index). Secondly, for each obtained cluster of job offers, and considering each job board in the DB, we build at each step of the algorithm, the clicks time series vectors (see Step 2 in Fig. 6) that represent the temporal behaviour of a job board by considering only the job offers that are disseminated in it, and belong to the current embedded cluster $\zeta^{posting}$. In other words, the observations (data points) of such time series are the ratio of clicks which were obtained through the job applicants URL visits on the offers that belong to the considered cluster, and which are daily clicked in this job board during a certain periodicity. Then, for each time series, a predictive model is learned, and future values of click ratios are predicted within a given horizon h (an average of 5 days). Finally, the job board(s) maximizing the different predicted ratio values are considered the most appropriate for the dissemination of the offers belonging to the considered cluster. A hash table is then created, containing key / values as cluster of offers, and the winner job boards.

During the recommendation step, and having a new incoming job offer, we want to disseminate it in the relevant job boards. Firstly the embedded representation of this job offer is created and is projected in the learned embedded space model to identify the closest class of job offers previously obtained. Thereafter, and visiting the hash table, this new job offer is recommended in the job boards that were associated to the closest cluster of job offers.

The contributions that concern the forecasting of future values of the clickstreams, use two complementary methods. The first one is based on the use of long short term memory (LSTM) deep neural network [48, 49] applied on the clicks numerical time series data. The second contribution suggests the transformation of the numerical time series to symbolic temporal sequences. Then symbolic data mining techniques such as N-grams [50] or sequence analysis are used to predict future symbols which represent a quantification of the clickstreams. Job Board time series data are the input of these two complementary methods (see Figure 1, Bottom, and figure 4, steps 4, 5 and 6). Future clickstream values are predicted with each method, and top ranked job boards, i.e. those which maximize at best the clicks, are kept for recommending new job offers.

Our idea is to consider each job category (cluster) as being different from the other ones hence analysing them separately. Thus the cluster of job offers in IT for instance, will be used as a homogeneous class to create the vectors of time series that will be used to make the prediction in this business category. This intuitive hypothesis was proposed here following many discussions with the HR experts who advised us to make the model mostly specific to each job category.

Each method, i.e., the embedding representation of the job offers and their clustering, the numerical time series forecasting, and the symbolic sequences prediction, are detailed in the next sections with the same order and separately to make this article easy to read.

4. Deep Learning and Doc2Vec for Job Offers Clustering

4.1. Embedded representation of job offers

As reported antecedently, we need to represent the textual job offer documents in a numerical way to make their manipulation with deep neural networks possible. One-hot encoding is the most common, and basic way to turn a token (word) into a vector. It consists in associating a unique integer index to every word, then turning this integer index i into a binary vector of size N, the size of the vocabulary, that would be all-zeros except for the ith entry, which would be 1.

Another popular and powerful way to associate a vector with a word is the use of "word vectors", also called "word embeddings". While the vectors obtained

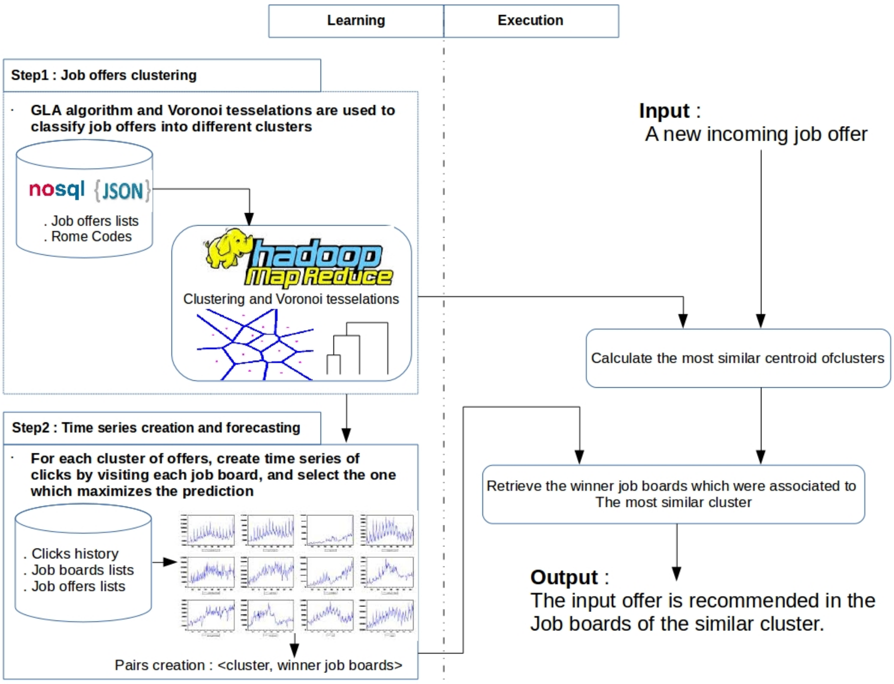


Figure 6: Overview on the clickstreams forecasting algorithm.

through one-hot encoding are binary, sparse (mostly made of zeros) and very high-dimensional (same dimensionality as the vocabulary), ”word embeddings” are low-dimensional floating point vectors (i.e. ”dense” vectors, as opposed to sparse vectors) [11, 47]. It is common to see word embeddings that are 256-dimensional, 512-dimensional, or 1024-dimensional when dealing with very large vocabularies. On the other hand, one-hot encoding generally leads to vectors that are 20,000-dimensional or higher (capturing a vocabulary of 20,000 token in this case). Therefore, word embedding can pack more information into far less dimensions. Fig. 7 gives an example of representing a text with numerical values.

There are two ways to use word embeddings: (i) learn word embeddings jointly with the main task (e.g. in our case for job offer documents classification), (ii) load pre-trained word embeddings into the model. The simplest way to associate a dense vector to a word would be to pick the vector at random. The problem with this approach is that the resulting embedding space would have no structure and no semantic relationship. For instance, the words ”job” and ”work” may end

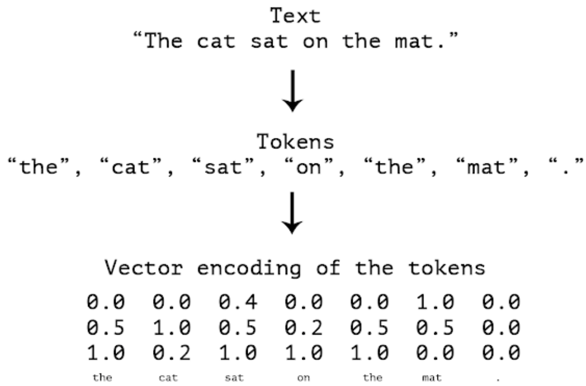


Figure 7: From text to tokens to vectors [11].

up with completely different embeddings, even though they are interchangeable in most sentences. It would be very difficult for a deep neural network to make sense of such a noisy, unstructured embedding space. To get a bit more abstract, the geometric relationships between word vectors should reflect the semantic relationships between these words. Word embeddings are supposed to map human language into a geometric space. For instance, in a reasonable embedding space, we would expect synonyms (e.g. job and work) to be embedded into similar word vectors, and in general we would expect the geometric distance (e.g. L2 distance) between them to be related to their semantic distance, that is to say words meaning very different things would be embedded to points far away from each other, while related words would be closer.

4.2. Word2vec and Doc2Vec representation of Job Offers

Natural language modelling technique like Word Embedding is used to map words or phrases from a vocabulary to a corresponding vector of real numbers. As well as being amenable to processing by Machine Learning (ML) algorithms, this vector representation has two important and advantageous properties: (i) Dimensionality Reduction - it is a more efficient representation, and (ii) Contextual Similarity - it is a more expressive representation. Previous works on Bag of Words (BoW) approach have shown that it often produces huge, very sparse vectors, where the dimensionality of the vectors representing each document is equal to the size of the supported vocabulary [27, 11]. Word Embedding aims to create a vector representation with a much lower dimensional space. In our case, Word Embedding is used for semantic parsing of job offers, to extract meaning from text to enable natural language understanding, and documents semantic classification. The vectors created by Word Embedding preserve the similarities, thus words that regularly occur nearby in text will also be in close proximity in vector space. Fig. 8 gives an example of an intuitive representation of some job titles in a word embedding space. Theoretically, jobs that belong to the same fields are supposed to be close to each other in the produced space model. Later, jobs represented with these titles are to be classified in the same clusters (e.g. data scientist and deep learning expert are in the same cluster of computer scientist jobs). This is the main interest of word embedding implementation in the first step of our learning algorithm.

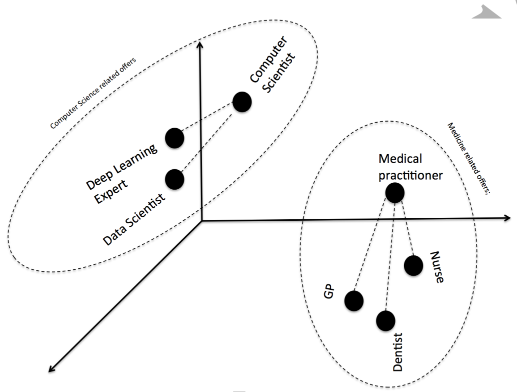


Figure 8: Example of a Word2vec representation of job offers.

One of the best known algorithms for producing word embedding models is Word2vec. This framework initially proposed by Mikolov et al. [20, 18, 22, 21] is based on their previous contribution called CBoW (Continuous Bag of Words). This model uses encoders neural networks to generate embeddings of a target word from an input context. While a language model is only able to look at the past words for its predictions, as it is evaluated on its ability to predict each next word in the corpus, a model that just aims to generate accurate word embeddings does not suffer from this restriction. Mikolov et al. thus use both the n words before and after the target word w_t to predict it as depicted in Fig. 9. They call this method the continuous bag-of-words (CBOW), as it uses continuous representations whose order is of no importance. The CBOW model tries to optimize an objective function defined as:

$$J_\theta = \frac{1}{T} log \quad p(w_t|w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n}) \tag{1}$$

Instead of feeding n previous words into the model, the model receives a window of n words around the target word wt at each time step t. In the deep neural network, this probability is calculated through the softmax layer(exp):

$$p(w_t|w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n}) = \frac{exp(h^T v'_{w_t})}{\Sigma_{w_i \in V} exp(h^T v'_{w_t})} \quad (2)$$

where, h is the intermediate state vector which is the word embedding v_{w_t} of the input word w_t of a vocabulary V.
The second contribution of Word2Vec is the Skip-Gram model (Fig. 10) which allows to do the inverse of CBoW, taking an input word and attempting to predict the words in the context. The skip-gram objective function sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t to produce the following objective:

$$J_\theta = \frac{1}{T} \Sigma_{t=1}^{T} \Sigma_{-n \leq j \leq n} log \quad p(w_{t+j}|w_t) \quad (3)$$

Similarly the skip-gram model computes $p(w_{t+j}|w_t)$, with the softmax layer as:

$$p(w_{t+j}|w_t) = \frac{exp(v_{w_t}^T v'_{w_{t+j}})}{\Sigma_{w_i \in V} exp(v_{w_t}^T v'_{w_i})} \quad (4)$$

Another word embedding algorithm worth knowing about is GloVe, which works slightly differently by accumulating counts of co-occurrences.

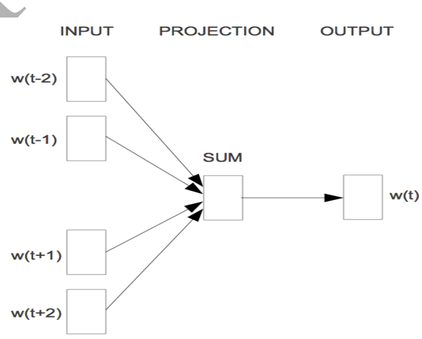


Figure 9: Continuous bag-of-words (Mikolov et al., 2013).

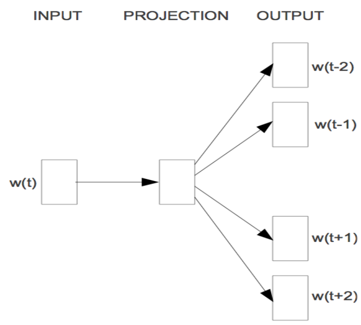


Figure 10: Skip-gram (Mikolov et al., 2013)

In 2014, Doc2vec that is an adaptation of Word2Vec, was introduced by Mikolov [20, 18, 22, 21] as a set of approaches to represent documents as fixed length low dimensional vectors that are document embeddings. Recent deep learning and NLP works have claimed that doc2vec outperforms other embedding schemes. Note that Word2vec is a three layers neural network with one input, one hidden and an output layer. The idea of CBOW architecture, one of the word2vec based algorithms, is to learn word representations that can predict a word given its surrounding words. The input layer corresponds to signals for context (surrounding words) and output layer correspond to signals for predicted target word. Doc2Vec explores the word context observation by adding additional input nodes representing documents as additional context. Each additional node can be thought of just as an id for each input document.

Doc2vec is a shallow neural net. Before implementing our embedding mode, we set up a work-flow as it is illustrated in Fig. 11. This process starts reading the job offer documents from the Hadoop HDFS disks and then applies successive analysis like tokenization, encoding, and embedding learning.

Once all NLP pre-processing terminated we used the corpus of job offers to represent each document in the Doc2Vec space model. To that aim, we have used a deep learning API implemented in Gensim open source library[6]. Fig. 12 represents the architecture of the neural network that was used to produce the embedding representation. The neural network takes as input each document as a sequence of words. As reported in the previous paragraphs, each word is represented as an encoding numeric vector. The documents sequences are then padded

[6]https://radimrehurek.com/gensim/models/doc2vec.html

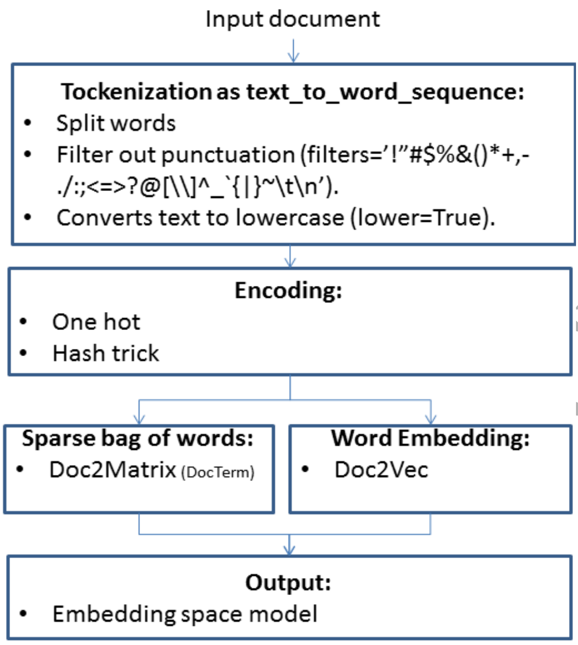


Figure 11: The followed workflow to produce the embedding of the job offer textual documents.

to a fixed-length sequence. The network has an embedding layer that produces the embedded representation of all job offer vectors. One important thing to note is that one can now infer a vector for any piece of text without having to re-train the model by passing a list of words to the model.infer_vector function implemented in Gensim. This vector can then be compared with other vectors via any similarity measure.

Once the embedding representation of the job offers terminated, we followed the first step of our learning algorithm, by classifying the documents vectors in different clusters, in-order to extract the emerging topics from the database. As we have explained it previously, the idea is to create a projection sub-space of job offers for performing the clicks forecasting using the job offers present in each embedded cluster.

Since each document is represented through a numerical embedding vector, we have tested a lot of clustering algorithms: Hierarchical, K-Means, and PAM (Partitions Around Medoids). The expected numbers of clusters were evaluated with a lot of well-known clustering optimization techniques, such as Silhouette Index,

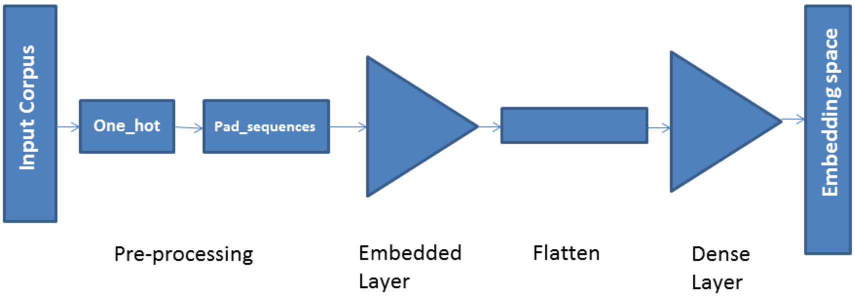


Figure 12: The architecture of the neural network that was used to produce the embedding representation of the job offer documents.

or Dunn Index, that compute the homogeneity of each clusters (i.e. the intra-set similarity) regarding the optimal separation (i.e. the inter-set similarity) [51]. The clustering results are presented in the evaluation section.

5. Deep Learning and Numerical Time Series For Clickstreams Forecasting

5.1. Preliminaries

The estimation of future values in a time series is a very interesting topic in data mining and machine learning. It is commonly done using past values of the same data. Given a job board time series, the forecasting here refers to the process of calculating one of several values ahead $\hat{X}^{JB_T}(N,h)$, using just the information given by the past values of the time series, $\hat{X}^{JB_T}(N,h)$= $\mathbf{f}(X_1^{JB_T}, X_2^{JB_T}, ..., X_N^{JB_T})$. Time series prediction issues are a difficult type of predictive modelling problem. Unlike regression predictive modelling, time series also add the complexity of sequence dependence among the input variables. In our context, we are interested by the prediction of clickstreams of the job applicants on different job boards. A powerful type of neural network designed to handle sequence dependencies are called recurrent neural networks (RNN) [47] [52]. They have the ability to connect previous information to the present task, such as using previous clicks values during the forecasting. However, the main drawback of RNN is that it is very difficult to get them to store information for long periods of time [53]. The Long Short-Term Memory Networks or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained. They were introduced by Hochreiter and Schmidhuber [49] as an improvement of RNNs, and were refined and popularized by many researchers in machine learning. Fig. 13 gives an example of a memory cell in a LSTM neural network.

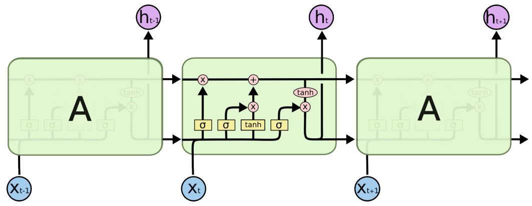


Figure 13: Example of a memory cell in a LSTM neural network.

It contains some multiplicative gates to keep constant error flow through the internal states of the special units. The three multiplicative gates are Input (I), Output (O) and Forget (F) gates. Their main role is to prevent memory contents from being perturbed by irrelevant inputs and outputs. The gates are used to save important information for each hidden layer from its previous layer, and vice versa with forget gates.

The simple version of a recurrent neural network owns an internal state h_t which is a summary of the sequence seen until the previous time step $(t1)$ and it is used together with the new input x_{t-1} [54], [49]:

$$h_t = \sigma(W_h x_t + U_h h_{t-1} + b_h) \tag{5}$$

$$y_t = \sigma(W_y h_t + b_y) \tag{6}$$

where W_h and U_h are respectively the weight matrices for the input and the internal state, W_y is the weight matrix for producing the output from the internal state, and the two b_y and b_h values are bias vectors.

The learning capability of this kind of RNNs is limited by the vanishing gradient problem, which prevent the learning of long term dependencies. Long Short-Term Memory (LSTM) has been hence proposed as a variant of RNN with the explicit intent of preventing the vanishing gradient, and it is defined by the following equations [54],[49]:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{7}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{8}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{9}$$

$$c_t = \sigma(W_c x_t + U_c h_{t-1} + b_c) \quad (10)$$

$$s_t = f_t.s_{t-1} + i_t.c_t \quad (11)$$

$$h_t = tanh(s_t).o_t \quad (12)$$

where i_t, f_t, o_t are, respectively, the input, forget and output gates, with values between 0 and 1, which decide what part of the input, of the previous hidden state and of the candidate output should low through the network. The vanishing gradient is due to the derivative of the $tanh$ function that is always strictly less than 1. In LSTM, the derivatives depend also on the gates, so that they are not anymore limited.

5.2. Architecture of the LSTM

The proposed Deep4Job clickstreams time series forecasting method is described in Algorithm 1. It takes as input the list of job boards, and a dictionary of the embedding clusters, where for each cluster we have the list of its job offers. The algorithm starts reading the time series of the clickstreams variation of the job offers that are disseminated in a job board JB_i and present in a cluster C_j. For each cluster of job offers, we will have then as much time series as the number of job boards. Then the algorithm applies the LSTM deep neural network, to train the temporal model on the time series and calculate $forecast(JB_i)_h$ the future values of clicks in an horizon h (average of 5 days). A maximum value of the forecasted clickstreams $Maxclick$ is calculated and its associated job board is identified as the winner job board, which will be associated in a hash table to the considered cluster. The algorith may generate a list of winner job boards on the ranked list of the predicted time series. This is the case when many job boards have led to similar $Maxclick$ values during the forecasting.
We have implemented our predictive model using Keras deep learning library to address the time series forecasting problem. The network has a visible input layer, 2 LSTM layers of size 32 units, each of which followed by 3 drop out layers, and 2 fully connected layers of size 64 units. The architecture is displayed in Fig. 14. The selection of the best architecture is still heuristic, even though we have tested very deep networks with more LSTM layers. However, the results were approximately close to those obtained with this architecture. The default sigmoid activation function is used for the LSTM blocks. The network is trained for 200

epochs and a batch size of 1 to 10 is used in the input. A sliding window of length 8 is used to address the problem as a regression. Concerning this look-back window we did several experiments and we found $w = 8$ as a best tuning parameter. The output of the network makes an estimation of the forecasting and the algorithm attempts to keep the maximum value $Maxclic = forecast(JB_i)_h$ and hence the good job board JB_i. The reader can access to the freely available code in our repository, in-order to test and evaluate the model [7].

For breaking down the over-fitting problems, we have used the dropout technique. This machine learning approach consists in randomly zeroing-out input units of layers in order to break happenstance correlations in the training data that the layers are exposed to. It has been known that applying dropout before a recurrent layer hinders learning rather than helping with regularization. In the case of LSTM networks, a temporally constant dropout mask is applied to the inner activations of the layers, in order to properly propagate its learning error through time [11].

The obtained results as well as the impact of the deep learning predictive network on the recommender system will be presented in the evaluation section.

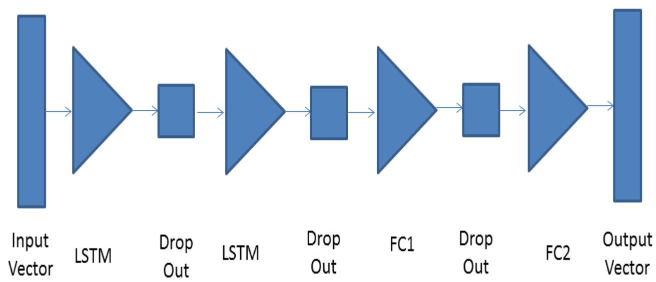


Figure 14: Overview on the architecture of the deep neural network implemented in our algorithm. An input layer is followed by 2 LSTM layers and 2 fully connected layers. Drop out layers were used to avoid over-fitting problem during the training.

[7] https://gitlab.com/opencver91/dl

Algorithm 1 Deep learning algorithm for numerical time series Clicks forecasting in each job-board.

Require: A collection of clusters C_{off} containing similar job offers, and a list of job boards JB, and an horizon value h.
Ensure: The appropriate job boards which maximize the predicted value of clicks ratio.

```
Begin
Maxclick = 0
By considering a cluster of job offers C_off at each time
for each jobboard JB_i in database DB do
   for each instant t ∈ Δ_t do
      Calculate the ratio x(t) = |clics| / |C_off|.
   end for
   Construct time series X(JB_i)={x(t)|t ∈ Δ_t}.
   Apply moving average filter on X(JB_i) to reduce noises.
   Use LSTM deep neural net to calculate forecast(JB_i)_h future values of
   clicks in an horizon h.
   if Maxclick ≤ forecast(JB_i)_h then
      Maxclick = forecast(JB_i)_h
      WinnerJobBoadsList.Add(JB_i)
   end if
end for
return Map(C_off, WinnerJobBoadsList).
End
```

6. Deep Learning and Temporal Sequences For Clickstreams Forecasting

6.1. Preliminaries

Predictive models with symbolic sequences concern generally 4 types of problems [47]: Sequence prediction, Sequence Classification, Sequence generation, Sequence-to-sequence Prediction. These models are different from set-based machine learning problems since in a sequence, the order of the observations is explicitly imposed.

Sequence prediction models, also known as sequence learning, involve the prediction of the next value for a given input sequence. They are still a big challenge in pattern recognition and machine learning. Weather forecasting is a good example of sequence prediction.

Sequence classification involves predicting a class label for a given input sequence. DNA sequence mining or sentiment analysis is a good example of sequence classification.

Sequence generation involves generating a new output sequence that has the similar features as the input sequence. Text generation or handwrite prediction are good examples of sequence generation.

Sequence to sequence prediction (or *seq2seq*) is an extension of sequence prediction models. Rather than predicting a unique value, a new sequence of length greater or equal to one is predicted. So-far multi-step time series forecasting is an example of *seq2seq* learning.

As in the previous section, we want to consider the clickstreams predictive model with an alternative way, using symbolic sequences instead of numerical time series. The symbolic encoding of time series with dimensionality reduction methods attempts to model trajectories of job applicants through their past visits and clicks, and might be useful to highlight their global behaviour for estimating what new job offers they want to apply for in the future. The remainder of this section presents firstly the two used time series encoding methods (SAX and PMVQ), then will show how it is possible to forecast future clicks with symbolic sequences using both probabilistic N-grams and deep learning.

6.2. Definitions

Nowadays, sources of information increased dramatically in different life domains, due to the availability of sensors in different systems. Time series data are occurring almost everywhere in various domains from medical (EEG, ECG, blood pressure), aerospace (satellite data), finance and business (stock market), meteorology (variation in temperature or pressure), sociology (crime figures, number of arrests), and others [55][56, 57]. Time series (TS) data mining methods are actually involved in many applications such as classification, clustering, similarity search, motif discovery, anomaly detection, and others [58, 56]. In practice, multi valued numerical TS suffer from high dimensionality, which is not convenient in the storage of this kind of data and the computational complexity of their manipulation. Such difficulties led to propose solutions involving dimensionality reduction. Many discretization methods have been proposed in the literature to encode time series in symbolic strings [59][60][61][62]. Among these methods, there is Fourier transform, PCA (Principal Component Analysis), Wavelet transform, SAX (Symbolic Aggregate Approximation) [61] and PMVQ (Parallel Multi-resolution Vector Quantization) [57]. All these methods have their advantages and some inconveniences. However, in a past work we have made an exhaustive evaluation, and have shown that SAX and PMVQ are very popular methods since they have been widely used for similarity search and clustering purposes [57]. We will present in a first step SAX and PMVQ methods, and then will show their involvement in our symbolic prediction application. Each predicted symbol is a quantification of the users' clicks on job offers. Hence we will study the behavioral trajectories of job applicants throughout these new representations.

6.3. Time Series Symbolic Aggregate Approximation: SAX

SAX maps a time series $T = (X_1^{JB_T}, X_2^{JB_T}, .., X_N^{JB_T})$ to a sequence of symbols from an alphabet of size $a = |\Sigma|$ [61]. The first step of this approach is to divide the time series of length *n* in w (codeword) frames of equal size and compute the mean value of the data falling within the window frame, and a vector of these values becomes the data-reduced representation. The sum of these averages is based on the PAA transformation (Piecewise Aggregate Approximation) where the i^{th} element is $C_i = \frac{w}{n} \Sigma_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} X^{JB_T}$. It should be noted that, before applying PAA, each time series is normalized to zero mean and standard deviation of 1, to avoid comparing time series with different offsets and amplitudes.

In the second stage, each segment is symbolized by strings of an alphabet. The conversion of the PAA representation of a time series into SAX is based on producing symbols that correspond to the time series features with equal probability. Keogh et al. [61] have shown that usually, the time series data follow a Normal distribution. With the normal distribution we can easily choose areas of equal size on the Gaussian curve, which define the breakpoints (quantiles) [61]. The same authors used a lookup table to determine breakpoints that divide a Gaussian distribution in an arbitrary number of equitable regions. The number of breakpoints

β_i is related to the size of the alphabet *a* (codebook), where *number (breakpoints) = alphabet size - 1*.
The interval between two successive breakpoints is assigned to a symbol of the alphabet, and each segment of the PAA within this interval is discretized by this symbol.
Fig. 15 gives an example of a numerical time series and its relative SAX sequence. In this example, the codeword length w=8 (8 window positions or splits along time dimension), and the codebook length $a = 3$ (three symbols of the alphabet). The SAX sequence of this series is $CBCCBAAB$. It appears clearly that such representation is very useful since data acquisition and their representation in numerical time series can intimately engender errors related to sensors or the acquisition protocol. It also appears evidently that with symbolic sequences, we can take the advantage of the robustness of the symbolic data mining and natural language processing methods, such as similarity search, pattern discovery, frequent motifs mining, behavioral trajectories construction, etc.

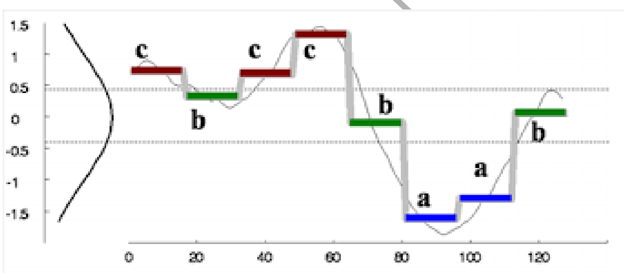


Figure 15: Example of a time series encoding to a SAX sequence. First, parameters such as the codeword (window length) as well as the codebook (number of symbols) are defined by the user. The temporal data are split down with a factor of codeword. At each position of the window, the mean value is calculated and then encoded with a symbol.

6.4. Time Series Parallel Multi-Resolution Vectors Quantization: PMVQ

Vector Quantization (VQ) is a wavelet transform that has been widely used in image processing for color image compression [63]. It is based on the extraction of the perceptual spatial correlation through wavelet transforms. Given a time series $T = (X_1^{JB_T}, X_2^{JB_T}, ..., X_N^{JB_T})$ with $X_i^{JB_T} \in R^N$ is the data point representing the relative click quantity of job applicants on the job offers at a date (i) in the job board JB. We define a vector quantizer Q of size K and dimension N as a mapping function of the data points $X_i^{JB_T}$ in one of the K output generated points

Y_j from C where: C= $\{Y_1, Y_2, ..., Y_K\}$ where $Y_j \in R^N$. Here C is called the codebook (CB) and Y is the codeword (CW).

Our implementation of the PMVQ method for the clicks time series symbolic representation is given in Fig. 16. First, job boards time series are extracted from the big database, and split down as subsequences of a given length that equals (CW) code word parameter (top right in Fig. 16). The obtained subsequences are clustered in different groups of a given size (CB) that represents the code book. The parameter CB represents the number of clusters of the parallel hadoop-based partitioning algorithm (bottom right of Fig. 16). After that, a sliding window alongside the original time series is analysed, and each position in the series is compared to the content of the learned CB. The most similar label of the clusters is identified as the symbolizing alphabet of the current window position. In other terms, the subsequence representing the current position of the sliding window is compared to the subsequences of CB which represent the centroids of the clusters, and then the pointed subsequence is labeled by the identifier of the most similar centroid. Parameters such as codeword (window length) as well as codebook (number of symbols) are defined by the user [57, 62].

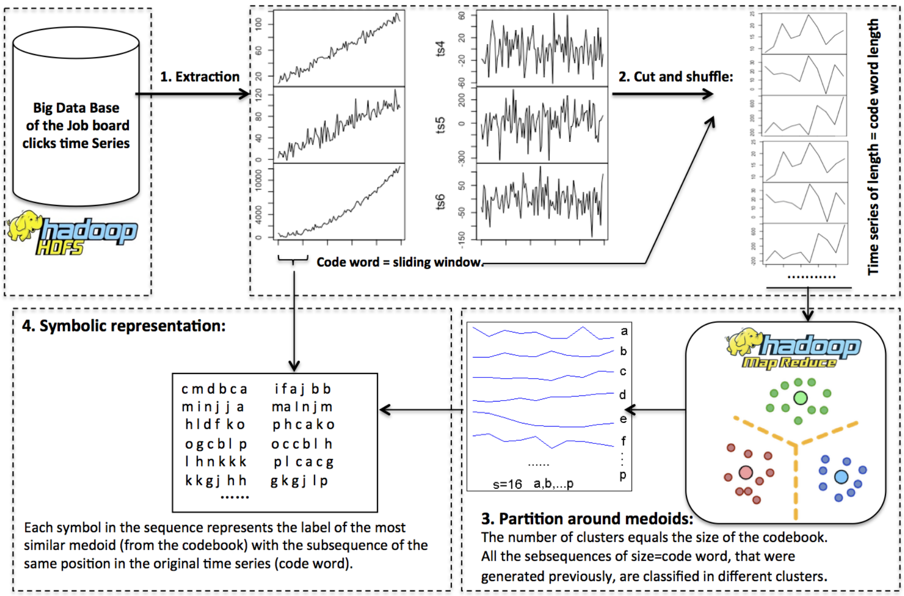


Figure 16: Implementation of the PMVQ method for time series symbolic representation.

6.5. Clicks Symbolic Time Series Prediction:

6.5.1. Prediction with N-grams

As illustrated in bottom of Fig. 1, each time series representing job boards will be encoded as a SAX or PMVQ symbolic sequence. The inputs of the encoding function are a job board series, the codeword (w), and the codebook (a). Note that the couple (w, a) is called the encoding resolution. Having a symbolic sequence of length w that we call S_w, Algorithm 1 is modified so that the prediction function becomes $forecast(JB_i)_h$= $predict(future\_symbol|S_w)$. The task of this sequence prediction function consists of forecasting the next symbol of a sequence based on the previously observed symbols. Recall that the predicted symbol represents in our case a future quantification of a clickstream value. To that aim, we propose here the use of Q-grams for generating sub-sequences from each sequence.

An n-gram is a succession of n characters or n words. A q-gram is a sub sequence of q consecutive characters in a given sequence. The n-gram method in our case will index and save all possible sub-sequences of length n.

The n-gram method was used by Claude Shannon who considered that it is possible to estimate the likelihood of observing a new symbol using the past observed symbols of a word. This modeling is a Markov model of order n where only the n last observations are used to predict future symbols [64].

For instance, having a symbolic sequence ($AABAACAAB$) of length $k \leq n$, the probability of having an element at position i depends only on the $n-1$ precedent elements, so that: $P(w_i|w_1, ..., w_{i-1})$=$P(w_i|w_{i-(n-1)}, w_{i-(n-2)}, ..., w_{i-1})$. For example with $n = 3$ we will have: $P(w_i|w_1, ..., w_{i-1})$=$P(w_i|, w_{i-2}, w_{i-1})$. With the precedent sequence $AABAACAAB$ we can have the possible n-grams depicted in table 1.

We can observe from the sequence the following occurring probabilities: $P(AAB) = 2$, $P(ABA) = 1$, $P(BAA) = 1$, $P(AAC) = 1$, $P(ACA) = 1$, etc. Hence we can estimate $P(B|AA)$=$\frac{P(AAB)}{P(AA)}$=$\frac{2}{3}$, and $P(C|AA)$=$\frac{P(AAC)}{P(AA)}$=$\frac{1}{3}$. Thus whenever we have the motif AA in a sequence we can expect the probability of having in the future the symbol B as $2/3$, and a symbol C with a probability $1/3$. As in the numerical time series case, the algorithm tries to identify the job board which will satisfy $Maxclic = forecast(JB_i)_h$. Here the variable $Maxclic$ is a symbol instead of a real value. For instance, regarding the SAX sequence displayed in Fig. 15, the greater values are those with the symbol C as breakpoint. Hence job boards that yield to predictions of sequences terminating with the symbol C are kept for recommendation, since it represents in this example the greater quantifi-

Table 1: The possible n-grams from the symbolic sequence $AABAACAAB$.

n=1	n=2	n=3	n=4	n=5
A	AA	AAB	AABA	AABAA
B	AB	ABA	ABAA	ABAAC
C	BA	BAA	BAAC	BAACA
	AC	AAC	AACA	AACAA
	CA	ACA	ACAA	ACAAB
		CAA	CAAB	

cation of the clicks. We consider all the n-grams of a sequence of a given job board as a database of sub-sequences. This database will be used as a training set of the sequence predictor that is implemented in [65][8] . The results of the prediction and the recommendation are discussed in the evaluation section.

6.5.2. *Prediction with Deep Neural Networks*

Sequence prediction in deep learning is a different issue from the other class of machine learning problems. It is mandatory to have an order on the observations that should be respected along the sequence during the learning process.
For our job offers symbolic time series forecasting we have considered Seq2Seq prediction model in which Encoder-Decoder LSTM are used to predict click-streams symbols. The architecture includes one layer for reading the input symbolic sequence and encoding it into a fixed length vector, in-order to learn the relationship between the symbols. The second layer (also known as the decoder) is used for decoding the pre-processed vectors to predict one or a set of symbols (sub-sequence). We implemented our model under Keras API (as in the case of numerical time series forecasting). The sources are given in the git repository of the project.
The architecture of the deep network used in Fig. 14 was also used here for symbolic trajectories prediction. However, we added additional layers to do one hot encoding of the input symbolic sequences. This involves converting each symbol of SAX or PMVQ to a binary vector. The decoder layer does the inverse, by converting the output vectors back into symbols. Results are presented in the next

[8]https://github.com/tedgueniche/IPredict

section for the remaining evaluation of our recommender system.

7. Evaluation and Results Discussion

In this section we will show the results of the evaluation of the different contributions that we have made in this paper. The assessment protocol involves in a first step the evaluation of the Doc2Vec job offers clustering. In a second step we will show the evaluation of the forecasting models on the numerical time series, as well as the symbolic sequences. Finally we will show the impact of each contribution on the recommendation system.

7.1. Evaluation of the Doc2Vec-Embedded job offers clustering

We present here the results of the job offers clustering. Recall that we have used Doc2Vec embedding representation for the projection of the job offers in an embedded space model. It was generated using Gensim API implementation of Doc2Vec. Then partitions around medoids as well as hierarchical clustering, methods were used to cluster job offers vectors. The inputs of Gensim are the three million textual documents that represent the job offers. For each document, we repeated the cleaning and tokenization procedures as illustrated in Fig. 11. Fig. 17, 18 and 19 show some dendrograms that were produced with hierarchical clustering using 1000, 10000 and 50000 random job offers documents through their embedded vectors. We didn't depict the dendrogram of the 3 million documents since it is not readable. The dendrogram can give a good clue on the partitioning clustering such as K-means or PAM algorithms. Indeed our global aim is to find the optimum number of clusters of our job offers documents, to make emerging the topics in our database. The optimum number of clusters can be obtained using the silhouette index method. In our case and after repeating the clustering process many times, we have observed that with $k = 663$, we obtained steady partitions of the textual job offers documents. These clusters will be then used for generating the time series and making possible the forecasting procedures.

7.2. Evaluation of the LSTM Networks for Numerical Time Series Forecasting

In this section we present the results of our experimentations on the LSTM neural networks that were implemented using Keras library. Each job board in the database is represented as a time series of 6 years (length= 6x365). Each series is divided into 2 subsets, 67% for training the LSTM model and 33% for the validation. Results are given in Fig. 20. Each job board is represented as a time

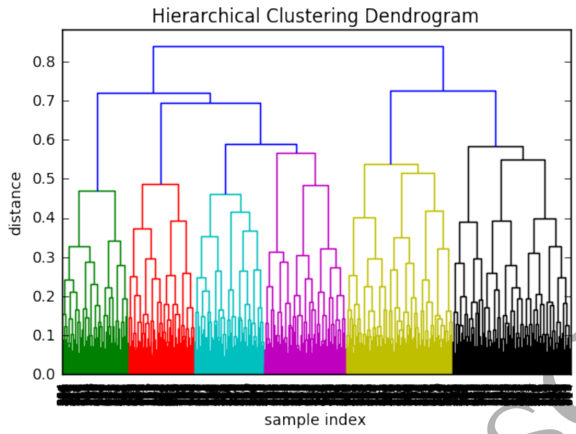


Figure 17: Dendrogram of some randomly chosen 1000 job offers documents using the similarity of their Doc2Vec representation.

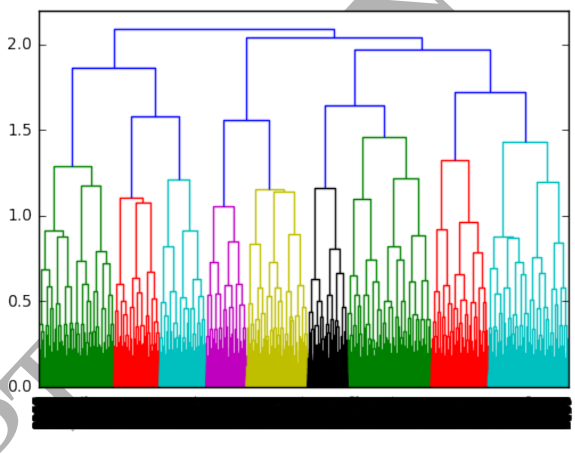


Figure 18: Dendrogram of some randomly chosen 10000 job offers documents using the similarity of their Doc2Vec representation..

series which is the input in the network. In blue we have the original data, green points represent the model fitted during the training, and red points represent the prediction of future clickstreams. We can observe that the predicted values in red fit well with the original time series in blue. To quantify these results, Fig. 21 gives an overview on the variation of the training error with the LSTM deep neural network using one job board from the precedent figure. Error values decrease after small number of iterations. This observation was checked for different job boards.

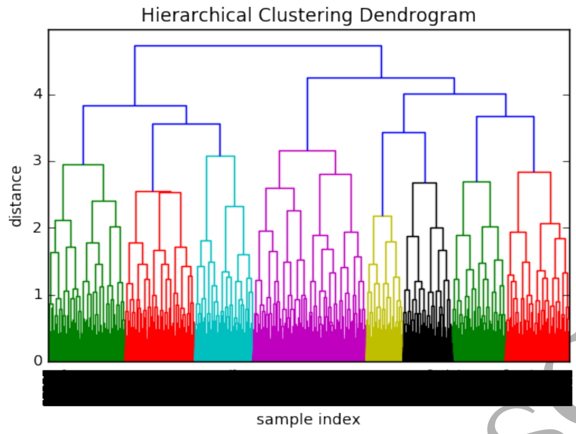


Figure 19: Dendrogram of some randomly chosen 50000 job offers documents using the similarity of their Doc2Vec representation..

For going a step forward in our evaluation, we calculated for each job board in the training DB the RMSE between the predicted values and the real time series data. Results are shown in Fig. 22. Error values are fluctuating with a global average of 0.14 which is very acceptable for a forecasting model.

7.3. Evaluation of the prediction with SAX and PMVQ Temporal Sequences

In this section we present the results of our experimentations that concern the use of the symbolic sequences for analysing the job applicant's trajectories in the database, and the prediction of future clickstreams symbols in the sequences. We have implemented both SAX and PMVQ dimensionality reduction methods using the same time series data.
Each job board time series is represented hence as a SAX and PMVQ sequence. Different resolutions, i.e. codeword and codebook values, were tested. Table 2 shows the used values in our experiment, which were in concordance with what were proposed in [61][57]. We expect that using high resolutions (large codeword splits, and great symbol codebook) the compression would be lossless, and inversely with small resolutions. For instance, with 1000 time series and using both SAX and PMVQ encoding methods, and for 8 resolutions we can obtain 1000x2x8 = 16000 symbolic sequences to train the models.
Fig. 23 and Fig. 24 illustrate the spectral representation of some job boards with the two encoding methods when producing the symbolic series. It is a new and innovative representation that we propose to have a global overview on the time

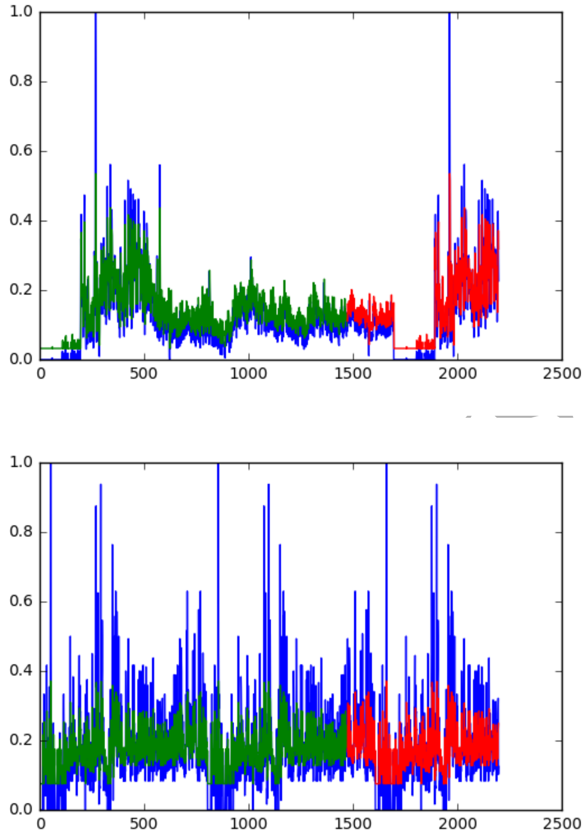


Figure 20: Some results of the training and test over the LSTM neural networks. Each job board is a time series which is the input the network. In blue we have the original clickstreams time series data, green points represent the model fitted during the training, and red points represent the prediction of future clickstreams.

series database, and for visually analysing the trajectories of the job applicants. Each vertical line represents a job board symbolic sequence. Each pixel of the line represents the quantification of the clicks with the encoding method.

Recall that we firstly applied N-grams as a first predictive method on the symbolic sequences. The global average RMSE values between the predicted symbols and real symbols in the sequence, are displayed in table 3 for each resolution. We can observe that the highest resolutions 7 and 8 have generated good RMSE for both encoding methods, with slight good results with PMVQ (0.25), whereas

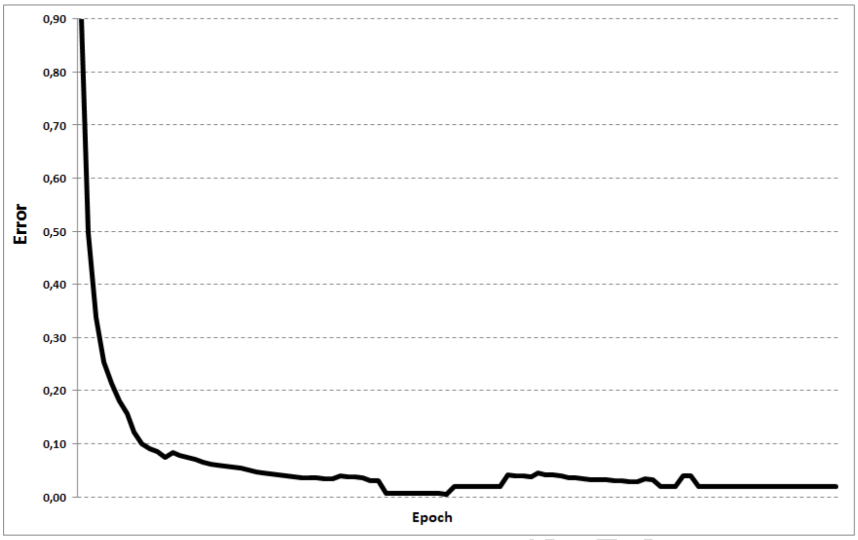


Figure 21: Variation of the error during the training of the LSTM neural network on a job board time series (Epoch 1 to 200). Error values decrease after small iterations.

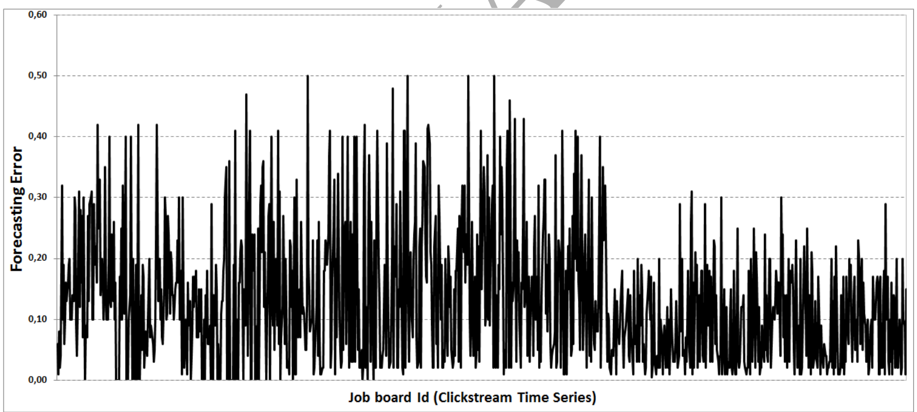


Figure 22: Variation of the prediction error using the LSTM deep neural network. The RMSE error is calculated between the predicted values and the real time series data. X-axis: job boards identifiers in the DB. Y-axis: the RMSE values.

small resolutions have led to bad predictions. The simplest way to interpret these observations is that with low resolutions, the symbolic encoding is lossy. By consequence the forecasting may have weaknesses due to the low discriminative power thay may exist between the observed sequences. These results confirm what we have already observed in a previous work on sensors data classification,

Table 2: The used resolutions for the temporal sequences generation. Values are varying from high resolutions (high codeword splits, and high symbol codebook) to small resolutions.

Resolution	**Codeword**	**Codebook**
1	8	8
2	8	16
3	128	8
4	128	16
5	256	8
6	256	16
7	512	8
8	512	16

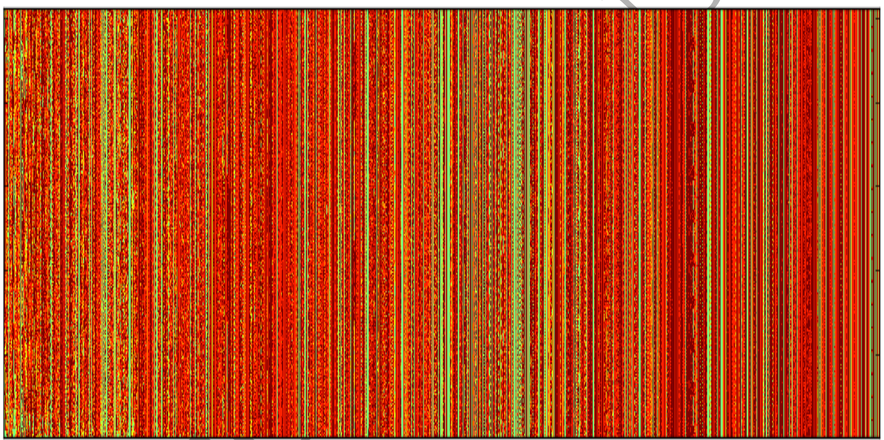


Figure 23: Spectral representation of some job boards with the SAX symbolic series. Each vertical line represents a job board symbolic sequence. Each pixel of the line represents the quantification of the clicks with SAX.

where we have shown that with high resolutions we can expect good classification and vice-versa [57].

To enhance the analysis we continued our evaluation protocol by testing the same symbolic sequences database on the proposed seq2seq deep neural network for sequence prediction that we have presented in the previous sections as a second predictive model on the symbolic sequences. Fig. 25 gives an example of the variation of the loss function, during the training of the deep neural network, on a PMVQ symbolic sequence of a given job board clicks data. Prediction error tends

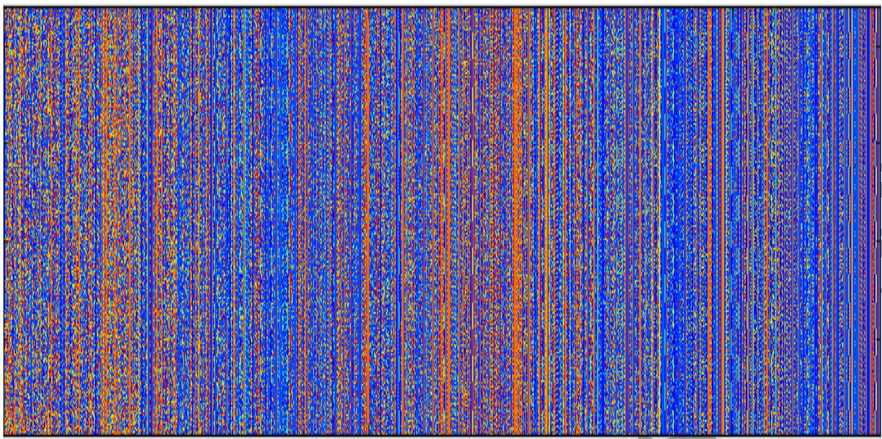


Figure 24: Spectral representation of some job boards with the PMVQ symbolic series. Each vertical line represents a job board's symbolic sequence. Each pixel of the line represents the quantification of the clicks with PMVQ.

Table 3: The obtained average RMSE values during the prediction with N-grams, for each resolution.

Res.	1	2	3	4	5	6	7	8
SAX RMSE	0.55	0.5	0.45	0.39	0.4	0.33	**0.31**	**0.30**
PMVQ RMSE	0.5	0.45	0.42	0.39	0.39	0.3	**0.27**	**0.25**

to zero after 200 epochs which is a good clue for convergence.
As in the previous case we have split down the sequences between learning and validation sub-sequences to make comparison between predicted and real symbols. Fig. 26 and Fig. 27 show the variation of the prediction accuracies which were obtained with deep LSTM on the SAX and PMVQ job board sequences respectively. The results concern sequences of resolution 8 since we obtained weak RMSE errors using this resolution. We can observe here that with PMVQ the prediction of future clicks quantification symbols is more efficient than SAX method. A global accuracy average of 0.89 was observed for PMVQ versus 0.73 for SAX. We have calculated the accuracy averages for the remaining resolutions (R1 to R8) with SAX and PMVQ, and the results are given in Fig. 28. Here we can also see that PMVQ is more efficient for predicting new symbols than SAX. Moreover, we can observe that with deep neural networks the prediction results are better than

those obtained with N-grams.

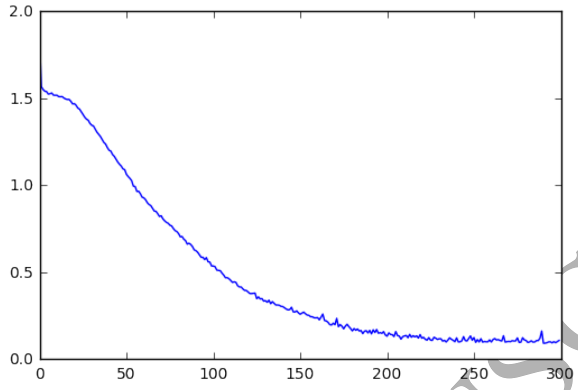


Figure 25: Variation of the loss function during the training of the deep neural network on a symbolic sequence. Prediction error tends to zero after 200 epochs.

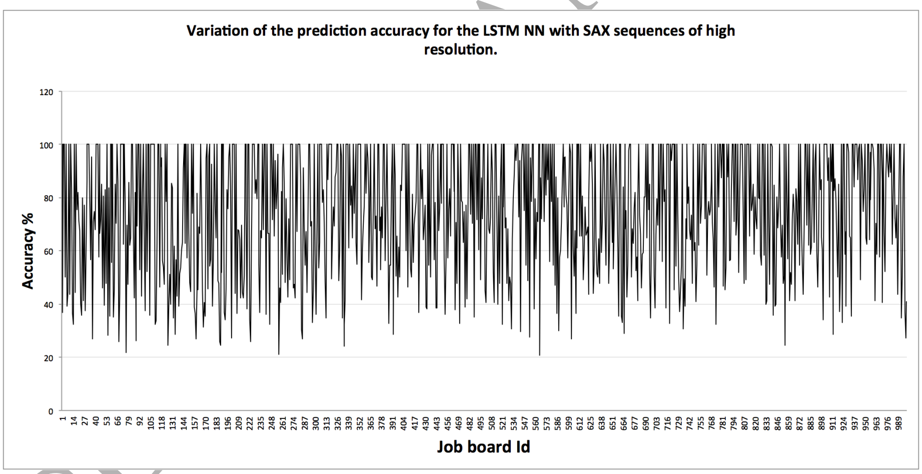


Figure 26: Variation of the prediction accuracy obtained with deep LSTM on the SAX job boards sequences. X axis: job boards IDs representing the SAX symbolic sequences of the clicks. Y axis: prediction accuracy on each job board. Results concern sequences of resolution 8.

7.4. Evaluation of Deep4Job During the Recommendation

As our work concerns a job offers recommender system that uses many temporal prediction models, we decided to evaluate the impact of each proposed technique on the recommendation performances of Deep4Job, that means for both

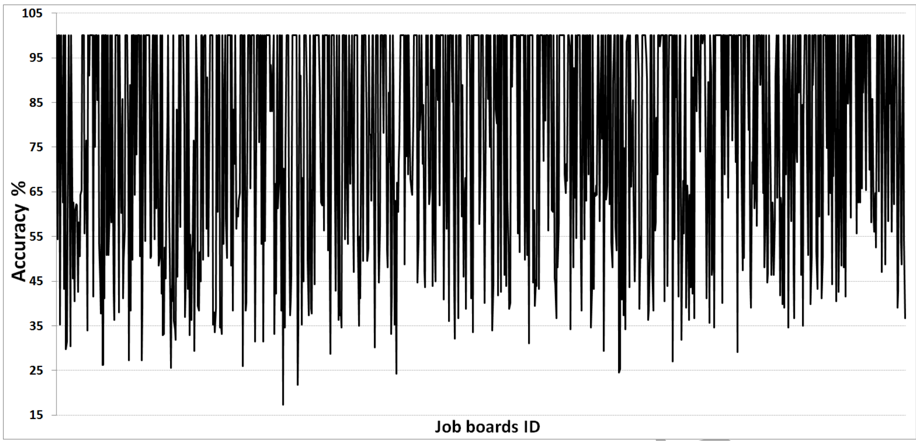


Figure 27: Variation of the prediction accuracy obtained with deep LSTM on the PMVQ job boards sequences. X axis: job boards IDs representing the PMVQ symbolic sequences of the clicks. Y axis: prediction accuracy on each job board. Results concern sequences of resolution 8.

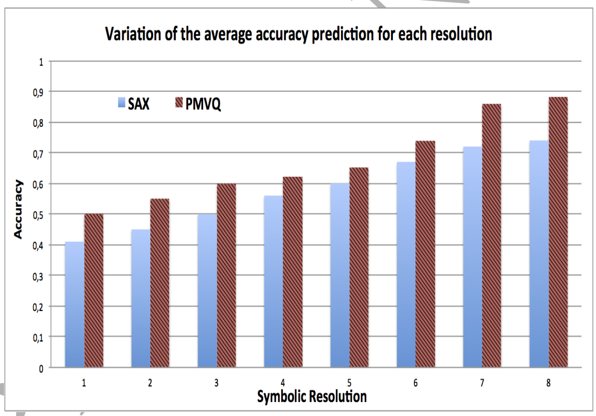


Figure 28: Average accuracy values of the prediction with the deep neural networks using the symbolic sequences of the job boards. Results are displayed for each resolution (R1 to R8) with SAX and PMVQ.

neural networks-based numerical time series prediction (Deep4Job LSTM-NN) as well as the symbolic sequences using the best resolution that equals 8 (with smallest RMSE) for SAX and PMVQ. The results are compared to a collaborative filtering (CF) method which corresponds to a baseline implementation of a previous work that we have proposed in [40]. This CF implementation is a memory-based approach, where the job offer documents are represented as vectors of frequent terms (TF), and the similarity between items is calculated with a weighted cosine measure. We have used a ground truth validation dataset of job offers with their supposed best job boards in which they should be disseminated. Processes were repeated in 10-fold cross validation, and the results are displayed in Table 4. The average F1-Score observed with deep learning and the numerical time series prediction equals 95% (Deep4Job LSTM Num TS column). The F1-Score results of the deep learning prediction using both PMVQ and SAX encoding methods are equal to 0.90 and 0.85 respectively (Deep4Job LSTM PMVQ and SAX Sym TS in Table 4). The results concerning N-grams prediction method for PMVQ and SAX are equal to 0.83 and 0.79 respectively (called Deep4Job NGrams PMVQ and SAX Sym TS in Table 4). The average F1-Score for the baseline recommender collaborative filtering (CF) is equal to 91%.

The first observation that we can made is that using the LSTM neural nets, the recommendation performances have been improved significantly compared to classical used methods (CF). The second observation concerns the high F-scores values when using numerical time series rather than symbolic sequences prediction. Even though the encoding methods reduce the dimensionality and the complexity of the data, the loosed information can penalise the performance of the recommender system. We can also see that deep learning (LSTM) prediction methods are very efficient compared to other prediction techniques such as N-grams. This is also a confirmation of what we asserted in the state of the art section where we have discussed the robustness and the strength of the new deep learning methods compared to the classical machine learning approaches. These satisfactory results come as a support to our preliminary idea with which we wanted to show that it is possible to improve the efficiency of a job offer recommendation system by analysing the temporal behaviour of job applicants, through their historical navigation data on the Internet. This work is also a pioneer example of the usefulness of the deep learning paradigm with a job offer recommender system, which is at our best knowledge the first work that includes this technology in such application.

Table 4: Evaluation of the recommendation results.

Algorithm	Deep4Job LSTM Num TS	Deep4Job LSTM-PMVQ R8 Sym TS	Deep4Job LSTM-SAX R8 Sym TS	Deep4Job NGrams-PMVQ R8 Sym TS	Deep4Job NGrams-SAX R8 Sym TS	Baseline CF
F1-Score	0.95	0.90	0.85	0.83	0.79	0.91

8. Conclusion and Perspectives

In this work, we have presented *Deep4Job*, a big data recommendation system based on the temporal prediction of the clickstreams with time series representation. The system analysis the historical behavior of job applicants in the Internet. We have shown how it was possible to use Doc2Vec embedding representation for extracting topics from large scale job offer documents. Then, we have proposed many prediction algorithms, using deep learning methods. We have implemented two complementary forecasting methods. The first approach uses LSTM neural networks (LSTM-NN) with numerical clicks time series data, while the second one uses multiple resolution time series symbolic encoding in the context of job offers dissimination. The proposed system suggests the recommendation of posting job offers in the top ranked job boards which may maximize at best the prediction of future clicks values. Each approach was separately evaluated on real datasets obtained from our industrial partner. The results were compared with the state of the art collaborative filtering recommender system. *LSTM-NN* Deep neural networks showed good performances compared to the rest of the methods.
As future work, we envisage including job applicants reviews on job market social networks such as Linked-in or job forums, in-order to take into account the sentiment analysis during the process of decision making. Indeed we want to use such information as a feedback that can be used to endorse the job boards recommendation. We also envisage to adapt and improve other prediction techniques that were used in the financial prediction problems [66].

Acknowledgment

The authors would like to thank Multiposting start-up for data sharing.
Thanks to Dr. James Cheney for proofreading the article

Funding

This work was supported by the French government and Ile de France region under a grant for FUI SONAR Project (FUI-AAP15-SONAR) for automatic recruitment tasks.

Availability:

The sources and the additional materials are available in https://gitlab.com/opencver91/dl.

Compliance with Ethical Standards

The authors declare that there is no conflict of interest.
Ethical approval: This article does not contain any studies with human participants or animals performed by the author.

References

[1] J. Sgula. *Fouille de donnes textuelles et systmes de recommandation appliqus aux offres d'emploi diffuses sur le web*. PhD thesis, CEDRIC Laboratory, Paris, France, 2012.

[2] Xu Yu, Yan Chu, Feng Jiang, Ying Guo, and Dunwei Gong. Svms classification based two-side cross domain collaborative filtering by inferring intrinsic user and item features. *Knowledge-Based Systems*, 141(Supplement C):80 – 91, 2018.

[3] Haifeng Liu, Zheng Hu, Ahmad Mian, Hui Tian, and Xuzhen Zhu. A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-Based Systems*, 56(Supplement C):156 – 166, 2014.

[4] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[5] Mariem Bambia, Mohand Boughanem, and Rim Faiz. Exploring current viewing context for TV contents recommendation. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2016, Omaha, NE, USA, October 13-16, 2016*, pages 272–279. IEEE Computer Society, 2016.

[6] Mohamed Nader Jelassi, Sadok Ben Yahia, and Engelbert Mephu Nguifo. Étude du profil utilisateur pour la recommandation dans les folksonomies. In Nathalie Pernelle, editor, *IC 2016 : 27es Journées francophones d'Ingénierie des Connaissances (Proceedings of the 27th French Knowledge Engineering Conference), Montpellier, France, June 6-10, 2016.*, pages 181–192, 2016.

[7] recommender systems. `http://www.datasciencecentral.com/profiles/blogs/5-types-of-recommenders`. Accessed: 2010-09-30.

[8] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Jialie Shen, Shunxiang Wu, and Tat-Seng Chua. Version-sensitive mobile app recommendation. *Inf. Sci.*, 381(C):161–175, March 2017.

[9] Da Cao, Xiangnan He, Liqiang Nie, Xiaochi Wei, Xia Hu, Shunxiang Wu, and Tat-Seng Chua. Cross-platform app recommendation by jointly modeling ratings and texts. *ACM Trans. Inf. Syst.*, 35(4):37:1–37:27, July 2017.

[10] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. Embedding factorization models for jointly recommending items and user generated lists. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 585–594, New York, NY, USA, 2017. ACM.

[11] F. Chollet. *Deep Learning with Python*. Manning Publications Company, 2017.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[13] Neeraj Kumar, Ruchika Verma, and Amit Sethi. Convolutional neural networks for wavelet domain super resolution. *Pattern Recognition Letters*, 90(Supplement C):65 – 71, 2017.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[16] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):664–676, April 2017.

[17] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 2017–2025, Cambridge, MA, USA, 2015. MIT Press.

[18] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[19] Tomas Mikolov. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[22] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification, 2016. cite arxiv:1607.01759.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[25] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[26] Changliang Li, Bo Xu, Gaowei Wu, Saike He, Guanhua Tian, and Hongwei Hao. Recursive deep learning for sentiment analysis over social data. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02*, WI-IAT '14, pages 180–185, Washington, DC, USA, 2014. IEEE Computer Society.

[27] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[28] Vishnu Nath and Stephen E. Levinson. *Autonomous Robotics and Deep Learning*. Springer Publishing Company, Incorporated, 2014.

[29] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016.

[30] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 233–240, New York, NY, USA, 2016. ACM.

[31] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 764–773. JMLR.org, 2016.

[32] Young-Jun Ko, Lucas Maystre, and Matthias Grossglauser. Collaborative recurrent neural networks for dynamic recommender systems. In Robert J. Durrant and Kee-Eung Kim, editors, *Proceedings of The 8th Asian Conference on Machine Learning*, volume 63 of *Proceedings of Machine Learning Research*, pages 366–381, The University of Waikato, Hamilton, New Zealand, 16–18 Nov 2016. PMLR.

[33] Florian Strub, Romaric Gaudel, and Jérémie Mary. Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 11–16, New York, NY, USA, 2016. ACM.

[34] Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2643–2651, USA, 2013. Curran Associates Inc.

[35] Amjad Almahairi, Kyle Kastner, Kyunghyun Cho, and Aaron Courville. Learning distributed representations from reviews for collaborative filtering. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 147–154, New York, NY, USA, 2015. ACM.

[36] Lei Zheng, Vahid Noroozi, and Philip S. Yu. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 425–434, New York, NY, USA, 2017. ACM.

[37] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 191–198, New York, NY, USA, 2016. ACM.

[38] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[39] Mamadou Diaby and Emmanuel Viennet. Développement d'une application de recommandation d'offres demploi aux utilisateurs de facebook et linkedin. In *Atelier Fouille de Données Complexes de la 14e Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances (EGC'14)*, Rennes, jan 2014.

[40] Sidahmed Benabderrahmane, Nedra Mellouli, Myriam Lamolle, and Patrick Paroubek. Smart4job: A big data framework for intelligent job offers broadcasting using time series forecasting and semantic classification. *Big Data Research*, 7:16–30, 2017.

[41] V. Radevski, Z. Dika, and F. Trichet. Common: A framework for developing knowledge-based systems dedicated to competency-based management. In *Information Technology Interfaces, 2006. 28th International Conference on*, pages 419–424, 2006.

[42] Matthias Hutterer. *Enhancing a Job Recommender with Implicit User Feedback*. PhD thesis, University of Wien, May 2011.

[43] Danielle H. Lee and Peter Brusilovsky. *User Modeling, Adaptation, and Personalization: 17th International Conference, UMAP 2009, formerly UM and*

AH, Trento, Italy, June 22-26, 2009. Proceedings, chapter Reinforcing Recommendation Using Implicit Negative Feedback, pages 422–427. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[44] Eleni Tsironi, Pablo Barros, Cornelius Weber, and Stefan Wermter. An analysis of convolutional long short-term memory recurrent neural networks for gesture recognition. *Neurocomputing*, 268:76–86, 2017.

[45] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007.

[46] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with i SAX2+. *Knowl. Inf. Syst.*, 39(1):123–151, 2014.

[47] Brownlee Jason. *Deep Learning With Python*. 2006.

[48] Yuzhen Lu and Fathi M. Salem. Simplified gating in long short-term memory (LSTM) recurrent neural networks. *CoRR*, abs/1701.03441, 2017.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[50] William M. Fisher. A statistical text-to-phone function using ngrams and rules. In *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '99, Phoenix, Arizona, USA, March 15-19, 1999*, pages 649–652, 1999.

[51] Brijnesh J. Jain. Consistency of mean partitions in consensus clustering. *Pattern Recognition*, 71(Supplement C):26 – 35, 2017.

[52] Xueliang Liu. Deep recurrent neural network for protein function prediction from sequence. *CoRR*, abs/1701.08318, 2017.

[53] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[54] Mattia Antonino Di Gangi, Salvatore Gaglio, Claudio La Bua, Giosuè Lo Bosco, and Riccardo Rizzo. A deep learning network for exploiting positional information in nucleosome related sequences. In Ignacio Rojas and

Francisco M. Ortuño Guzman, editors, *Bioinformatics and Biomedical Engineering - 5th International Work-Conference, IWBBIO 2017, Granada, Spain, April 26-28, 2017, Proceedings, Part II*, volume 10209 of *Lecture Notes in Computer Science*, pages 524–533, 2017.

[55] Gaber Mohamed et al. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005.

[56] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 31(3):606–660, 2017.

[57] Sidahmed Benabderrahmane, Rene Quiniou, and Thomas Guyet. Evaluating distance measures and times series clustering for temporal patterns retrieval. In James Joshi, Elisa Bertino, Bhavani M. Thuraisingham, and Ling Liu, editors, *Proceedings of the 15th IEEE International Conference on Information Reuse and Integration, IRI 2014, Redwood City, CA, USA, August 13-15, 2014*, pages 434–441. IEEE, 2014.

[58] Ralanamahatana ChotiratAnn et al. Mining time series data. In Oded Maimon and Lior Rokach, editors, *DMKD Handbook*, pages 1069–1103. Springer US, 2005.

[59] Kin-Pong Chan and A.W.-C. Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133, 1999.

[60] Chakrabarti Kaushik and Keogh Eamonn et al. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM TDS.*, 27(2):188–228, June 2002.

[61] Jessica Lin and Eamonn Keogh et al. A symbolic representation of time series, with implications for streaming algorithms. In *In Proceedings of the 8th ACM SIGMOD RIDMKD Workshop*, pages 2–11. ACM Press, 2003.

[62] Vasileios Megalooikonomou, Qiang Wang, Guo Li, and Christos Faloutsos. A multiresolution symbolic representation of time series. In *ICDE*, pages 668–679, 2005.

[63] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symp. on MSP*, pages 281–297, 1967.

[64] Nicolas Turenne. *Analyse de donnees textuelles sous R*. editions COLLECTION SCIENCES COGNITIVES, 2016.

[65] Open data mining library, howpublished = `http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php#cptplus`.

[66] Yauheniya Shynkevich, T. Martin McGinnity, Sonya A. Coleman, Ammar Belatreche, and Yuhua Li. Forecasting price movements using technical indicators: Investigating the impact of varying input window length. *Neurocomputing*, 264:71–88, 2017.