Neural Computing and Applications (2021) 33:5069–5087
https://doi.org/10.1007/s00521-020-05302-x

ORIGINAL ARTICLE

Skills prediction based on multi-label resume classification using CNN with model predictions explanation

Kameni Florentin Flambeau Jiechieu[1,2] · Norbert Tsopze[1,2]

Received: 6 February 2020 / Accepted: 18 August 2020 / Published online: 28 August 2020

Abstract
Skills extraction is a critical task when creating job recommender systems. It is also useful for building skills profiles and skills knowledge bases for organizations. The aim of skills extraction is to identify the skills expressed in documents such as resumes or job postings. Several methods have been proposed to tackle this problem. These methods already perform well when it comes to extracting explicitly mentioned skills from resumes. But skills have different levels of abstraction: high-level skills can be determined by low-level ones. Instead of just extracting skill-related terms, we propose a multi-label classification architecture model based on convolutional neural networks to predict high-level skills from resumes even if they are not explicitly mentioned in these resumes. Experiments carried out on a set of anonymous IT resumes collected from the Internet have shown the effectiveness of our method reaching 98.79% of recall and 91.34% of precision. In addition, features (terms) detected by convolutional filters are projected on the input resumes in order to present to the user, the terms which contributed to the model decision.

Keywords Skill-gap · Resume · Skills extraction · Multi-label classification · Convolutional neural network · Model explanation

1 Introduction

The Association for Talent Development (ATD) defines a *skill-gap* as a significant gap between an organization's current capabilities and the skills it needs to achieve its goals and meet customer demand [7]. Organizations bridge the skills gap by hiring candidates with specific skills to perform critical tasks. Nowadays, hiring processes are often conducted through the Internet. Applications (resumes and cover letters) are sent via e-mails, web sites or job posting platforms (indeed.com, freelancer.com, upwork.com, ...). The number of applications for a particular job can be very important, making the candidates selection cumbersome. It is to solve this problem that job-resume matching algorithms have been developed in recent years in order to quickly find candidates whose skills match those required to perform the job. But, many of these algorithms require user inputs [13, 16, 27]: a user should input his skills as keywords list; as well, skills needed to perform the job are supposed to be known in advance. The matching then consists in computing a similarity score between competences of candidates with those required and ranking the candidates according to their scores. However, things become more challenging when dealing with raw text resumes. In that case, a prior work should consist in extracting topic words (skills in our context) from resumes before matching them to the ones extracted from jobs. It has been shown that using topic words in content-based recommendation is more accurate than using any other word [2].

Lots of works have been done to tackle the problem of skills extraction from resumes. Some researchers have proposed the use of ontology to identify skills from resumes [4, 5], while others have suggested to handle the skills extraction problem as a named entity recognition (NER) problem where skills are considered as named

✉ Kameni Florentin Flambeau Jiechieu
florentin.jiechieu@mintp.cm

Norbert Tsopze
norbert.tsopze@facsciences-uy1.cm

[1] Department of Computer Science, University of Yaounde I, Yaounde, Cameroon

[2] IRD, UMMISCO, F-93143 Bondy, France

entities [15, 30]. Moreover, skills can be organized into different levels of abstraction. There are low-level skills (e.g., css, html, php, ...) and high-level ones (e.g., web developer, front-end developer, ...). A huge work has already been done by some researchers to build skills taxonomies [9, 10] which represent the hierarchical relationship between skills of many domains including IT. We assume that high-level skills are characterized by a set of low-level ones. While existing methods [15, 26, 30] already perform remarkably well when extracting skill-related terms from resumes, they are limited by their inability to infer high-level skills not explicitly mentioned in resumes. In IT, for example, the observation of *html, css, javascript, etc.* could lead a recruiter to deduce that the candidate is a *front-end developer* even if this was not explicitly mentioned in his resume. On the other hand, a recruiter could doubt that a candidate has a certain competence despite the fact that it has been mentioned in his resume; just because there were no sufficient elements to convince him that the candidate truly has that expertise.

Artificial intelligence models have evolved over the years and offer excellent performances in solving varieties of problems in different fields (job recommendations, image processing, word processing, medical analysis, etc.). However, if these models are powerful in terms of the accuracy of predictions, they often suffer from the problem of opacity: in fact, they operate more like black boxes providing results without being able to explain them. However, entrusting an important decision to a system that cannot be explained presents an obvious danger [1]. This is the main reason why the construction of explainable artificial intelligence models has been gaining increasing attention in recent years.

Moreover, Deep learning models especially Convolutional Neural Networks (CNN) has shown surprising results in text processing, achieving states of the art results in many classification tasks. One advantage of CNN is that they are able to automatically discover features from inputs to perform classification. This property could be used to automatically identify terms (low-level skills) describing high-level skills. In addition, many research works have been conducted in the sense of analyzing CNN in order to explain predictions in text classification [14]

Following the above considerations, we propose in this article a method to build an explainable model based on Convolutional Neural Networks (CNN) to identify high-level skills from raw text resumes. The resulting model is capable of predicting the set of high-level skills expressed in an input resume, while highlighting the low-levels skills which have led to that prediction. We believe that the proposed method would be more trustful and transparent for recruiters. We also believe that this work could lay the foundations for the construction of transparent and explainable job recommender systems using CNN.

To achieve this goal, we first propose to handle the problem of skills extraction from resumes as a multi-label classification problem, where the output classes are high-level skills; then, we analyze the predictions in order to derive for each CNN filter the set of words that it is specialized in identifying, and to know which words contributed to the prediction of a particular competence for a given input resume (prediction explanation). The multi-label classification model is a binary relevance-based model [11] where we have as much binary classifiers as there are classes in the overall dataset. Each binary classifier is based on CNN. To explain the classifier results, we rely on a part of the work done by Jacovi et al. [14] to detect, project and visualize the words selected by filters on the original resume.

The main contributions of this article are summarized as follows:

1. The use of a multi-label classification model based on CNN to predict high-level competences from resumes. Resumes are transformed into matrices using a convenient method, and the resulting matrices are used as inputs to CNN;
2. The construction of a context-specific word embedding model for resumes of the IT domain;
3. The analysis of filters to explain predictions and to guarantee confidence in the classifier decisions;
4. We conducted several experiments to demonstrate the effectiveness of our method.

The rest of this article is structured as follows: the next section presents existing works about skills processing and model predictions explanation. The third section describes the proposed method. Then, the results of our experiments are presented before ending with a conclusion.

2 Related works

Skills extraction from resumes is a critical task when building job recommender systems. It is also useful for automatically building experts competence profiles and companies competences knowledge-bases.

2.1 Works about skills extraction

Many parsing algorithms have been developed to tackle the problem of skills extraction from resumes:

Shiqiang et al. [13] have proposed a personalized job-resume matching system, which could help job seekers to easily find appropriate jobs. They created a finite state transducer-based information extraction library to extract models from resumes and job descriptions. Then, they defined a new statistical-based ontology similarity measure to match the resume models and the job models. The extracted model consists of degrees, majors and skills.

Zhao et al. [30] proposed a combination of Named Entity Recognition (NER) and Named Entity Normalization (NEN) to identify skills from texts, considering them as named entities. The approach consists in annotating a set of resumes with skills they contain in order to build a dataset that will be used to train a Named Entity Recognition model capable of identifying the set of skills contained in a text resume. This approach has the potential to recognize all the skills mentioned inside the resume and can be used to build a kind of resume summary containing the list of skills that the expert possesses. But generally, Named Entities Recognition needs huge amount of annotated corpora to perform well; and it is difficult to manually build such datasets. To tackle this problem, they proposed a method to automatically annotate the resumes using Wikipedia Common Categories, which they consider as a taxonomy of skills. However, the problem now resides in the precision and the completeness of the labeling. Further in [15], the same authors added a skill entity word sense disambiguation component which infers the correct meaning of an identified skill by leveraging Markov Chain Monte Carlo (MCMC) algorithms.

Sayfullina et al. [26] proposed an ontology-based model to extract skills from resumes. To be tagged as a skill, a noun phrase in a resume must be found inside the ontology. This poses the problem of the completeness of the ontology in the sense that if the ontology is not enough complete, then many skills will not be identified. To deal with that, they proposed a way to find other skills that are not present in the ontology. This is done with the help of some specific and lexicalized multi-word expression patterns (i.e., specific contexts) that could surround new and unknown skills. New skills are validated by an expert and added to the ontology.

The common point between the preceding methods is that they all parse input resumes and extract skills from them. This means that the resume must actually contain those skills. But in practice, some skills can be deduced by the recruiters when reading the resume, even if they do not appear explicitly inside the resume description. For example, an expert's resume might contain basic skills such as (css, html, javascript, ...) which normally would lead a recruiter deducing that the expert has the profile of a web developer. The approaches which consist in just parsing the resume cannot be used to find implicit high-level skills.

Kiyimaki et al. [18] proposed *Elisit*, a graph-based approach to skills extraction from resume. They use the hyperlink graph of *wikipedia* and skills extracted from *linkedIn* to associate skills with documents. They first analyze the input document with a vector space model in order to associate it with a *wikipedia* article. From this initial article, they use Spreading Activation Algorithm [6] on the hyperlink graph of *wikipedia* in order to find articles that correspond to *LinkedIn* skills and are related to the initial pages. As stated by the authors, developing a completely automatic optimization scheme for this model selection task is difficult because of the number of different parameters, the size of the Wikipedia graph and the heuristic nature of the whole methodology. That is why they evaluated their model manually.

2.2 Related works about models explanation

The majority of works about CNN models explanation have been done for image processing [8, 12, 19, 21, 24, 25]. Until now, only few research works concern the interpretation of CNN models built for text classification tasks. Jacovi et al. in [14] have recently proposed an approach to explain CNN models built for text classification tasks. They show that filters can be classified into three categories based on their contribution in the classifier prediction: (1) accidental filters which do not influence the classifier decision; (2) filters which recognize good patterns (patterns which actually describe the target class); (3) finally, filters which recognize bad patterns (patterns which do not describe the target class). The main limitation of their approach is that it applies only to a strictly limited range of CNN architectures: architectures with just one layer (the output layer) in the fully connected stage. Indeed, the output layer should come immediately after the max-pooling layer. This is why they assumed that the class of a filter is the class which maximizes the weights of the connections between the max-pooled value corresponding to that filter and the outputs neurons.

In [23], the authors proposed LIME, a model-independent approach to interpret the predictions. The idea is to consider the model to interpret as a black-box. Input features are varied to observe their impact on the outputs. Thus, the influence of input features on the outputs can be evaluated. This approach grows exponentially (in terms of time complexity) with respect to the number of features; and in text classification, we generally deal with more than dozens of thousands of features.

3 Proposed method

The method proposed in this article to deal with the problem of skills prediction as a multi-label classification task will be presented following the steps below:

1. First, the architecture of the multi-label skills prediction model is described along with its components;
2. Then, the methods used to "preprocess" the training data and build the components of the architecture are described ;
3. Finally, we describe how filters are analyzed to explain the predictions of the built model.

3.1 Multi-label classification architecture model for skills prediction

The first step in our method is to design a model capable of predicting a set of skills from an input resume. Since a resume might express multiple skills, we propose to handle the problem as a multi-label classification task. Figure 1 describes the architecture of the multi-label skills prediction model.

The architecture defined in Fig. 1 consists of two main components:

1. A preprocessing component which goal is to transform a raw text-resume, into a matrix which will serve as input for the CNN;
2. And a multi-label classifier component composed of a set of binary CNN classifiers.

3.1.1 Preprocessing component

When a raw text resume is passed as input to the model, the preprocessing component (preprocessor) transforms this resume into a matrix, before passing it as input to the CNN classifiers. The preprocessing consists in using a domain dictionary and a word embedding model to filter and embed the word vectors of a resume into a numerical matrix. We consider all the resume matrices to have the same length L which is defined empirically. The transformation of resume from text to matrix operates as follows:

1. Initially, an empty row matrix M is created;
2. Repeat the followings for each term w in the resume, until the length of the matrix is reached, or there is no other word to proceed:
   - if w is found in the domain dictionary and if there is an entry corresponding to the word w in the word embedding model then:
     (a) convert w into a numerical vector v using the embedding model;
     (b) append the corresponding vector v to the bottom of the matrix M;
3. If the length of the matrix is less than L then zero-padding is added to the bottom of the matrix to reach L.

At the end of these steps, we have the matrix of the resume which then goes as input to each base CNN classifier.

3.1.2 CNN classifiers

The second component of the architecture, consists of multiple independent binary CNN classifiers. Each CNN is used to classify resume according to a single class. In other words, a base CNN classifier f_j is designed to predict whether an input resume x belongs to the class c_j or not. The number n of base classifiers corresponds to the number of competences (classes) in the dataset (in practice, recruiters may know in advance, the sets of competences they would like to identify from resumes).

The prediction of the multi-label model then consists of the union of the competences predicted by the different base classifiers as specified in formula 1.

$$C_i = \bigcup_{j=1, f_j=1}^{j=n} c_j \tag{1}$$

We choose CNN as based classifiers for two main reasons:

1. Firstly, input of CNN generally consists of raw data which are less subject to information losses as compared to other models which generally required handcrafted features.
2. Secondly, convolutional filters can easily be analyzed for explanation purpose;

Figure 2 shows a view of the architecture of a base CNN classifier. This architecture is inspired from the architecture proposed by Kim, Yoon [17] and consists of a convolutional layer followed by a max-pooling layer, a ReLU layer and a fully connected layer. The model takes as input a matrix representation of a resume and predicts a value corresponding to the probability that the resume belongs to the target class or not.

Formally, a resume can be modeled as an n-words input text $w_1, w_2, \ldots, w_n$, and each word is embedded in a d dimension vector $w_i \in R^d$. So a resume can be represented as a matrix $M \in R^{d \times n}$. A convolution filter can be represented by a d dimension vector. For each filter f_j, the convolution among the resume matrix performs the scalar product $< w_i, f_j > (1 \leq i \leq n)$. The convolutions result in a matrix $F \in R^{n \times m}$ where m is the total number of filters. The columns of F represent the features maps. Applying max-pooling on the features maps results in a vector $p \in R^m$ which contains the maximum values of each features map. The components of the vector p are rectified by the ReLU activation unit before being passed as input to the fully connected neural network (FCNN) which computes the final decision. These steps are summarized as follows:

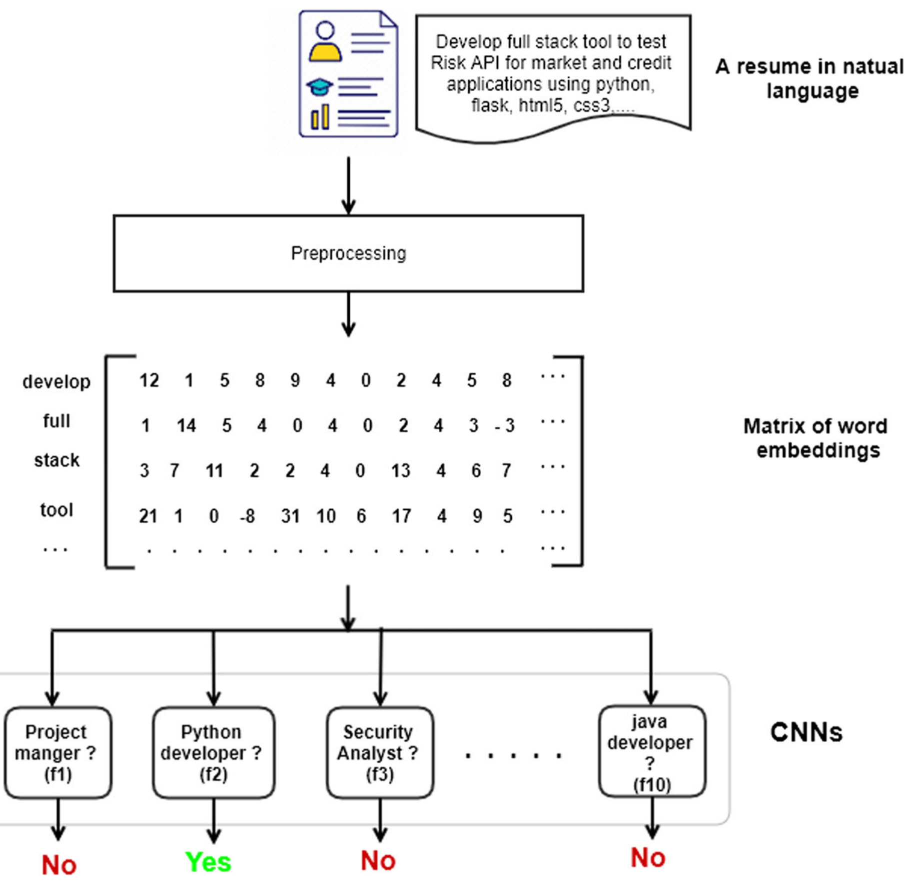

Develop full stack tool to test Risk API for market and credit applications using python, flask, html5, css3,....
A resume in natual language
Preprocessing
develop
full
stack
tool
Matrix of word embeddings
CNNs
Project manger ? (f1)
Python developer ? (f2)
Security Analyst ? (f3)
java developer ? (f10)
No
Yes
No
No

Fig. 1 Multi-label skills prediction architecture model

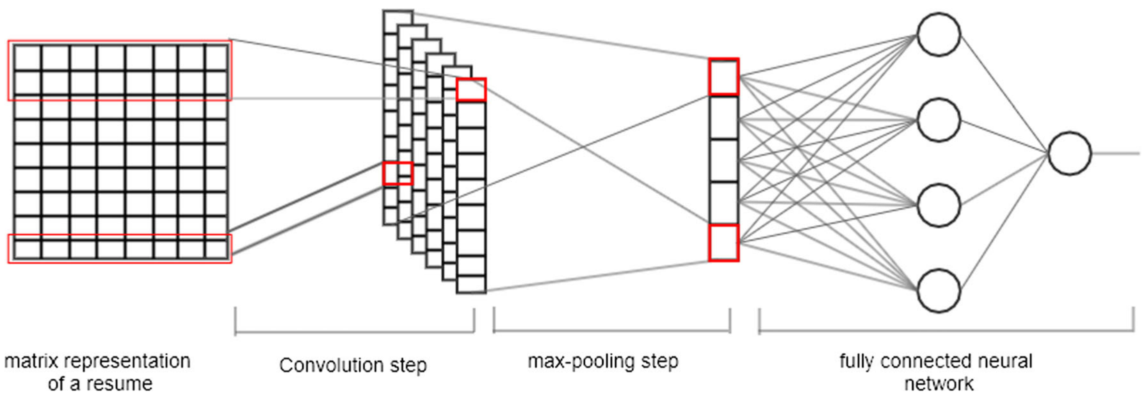

matrix representation of a resume
Convolution step
max-pooling step
fully connected neural network

Fig. 2 CNN architecture for text classification

1. **Convolution**: this operation is used to compute a similarity score between a filter and a word vector (Eq. 2). For simplicity, we considered filters with kernel size equal to 1 (filter height). It means that the convolution filters will specialize in recognizing single words. However, this work can also be easily adapted to work with variable kernel size filters.

$$F_{ij} = < w_i, f_j > \quad (2)$$

2. **ReLU (Rectified Linear Unit)**: this operation is used to rectify the convolution output by setting any negative value to zero. The ReLU function is defined by Eq. 3, where x is a real value.

$$y = \mathrm{ReLU}(x) = \max(x, 0) \quad (3)$$

3. **Max-pooling**: the operation performed here is a global max-pooling; meaning that the max-pool filter operates over the whole features map and selects its maximum value. This value is associated with the word which

produced the highest convolution score with the convolution filter. The component p_j of the max-pooled vector is given by Eq. 4, where F is the matrix of features maps and the column $F_{.j}$ is the features map associated with the filter f_j.

$$p_j = \mathrm{Max}_i(F_{ij}) \tag{4}$$

4. **Classification**: this is done by the fully connected neural network (FCNN). In the proposed architecture, the FCNN is composed by one hidden layer with the ReLU activation and one output unit (output layer) with a sigmoid activation. The hidden units receive as input, the max-pooled vector p and compute their activation h_i by Eq. 5 where W is the matrix of weights of the connections between hidden units and the input of FCNN (components of p); b_i are biases.

$$h_i = \mathrm{ReLU}(\sum_j W_{ij} \times p_j + bi) \tag{5}$$

The final output is calculated by Eq. 6.

$$y = sigmoid(W_i \times h_i + b) \tag{6}$$

The sigmoid function used is the logistic function: $sigmoid(x) = \frac{1}{1+e^{-x}}$, where W is the vector of synaptic weights of the connections between the output unit and the hidden units.

The overall network is trained using the stochastic gradient descent (SGD) algorithm which consists in searching the best parameters (W, b) i.e., the parameters which minimize the gradient of the error function. During the backpropagation, the weights W are updated using the general equation.

$$W_i^{t+1} = W_i^t + \eta \frac{\partial E_i}{\partial W_i} \tag{7}$$

W is the matrix representing the weights of the units in the fully connected neural network, η the learning rate and E the classification error.

After having described the overall architecture model and its main components, we now focus on how the various models composing this architecture are built (trained) to solve the multi-label classification problem.

3.2 Building the architecture models

To build and train the models composing the architecture described in the above section, we have assembled a set of resumes collected from the Internet. Each resume is labeled with the set of professional occupations of its owner.

The following processing steps are applied in order to build and train the models used in the architecture:

1. First of all, classes (competences) are normalized to have a common representation for competences;
2. Next, a word embedding model is trained from the corpus of resumes using the skip-gram algorithm ;
3. Then, resumes are transformed into matrices using the trained word embedding model;
4. After that, the original multi-label dataset is splitted into n single-label dataset; each used to train a single base classifier;
5. Finally, the base CNN classifiers are trained using the appropriate sub-datasets.

3.2.1 Classes normalization

The first step before building the models is to normalize the competences. In fact, input resumes consist of raw text resumes from the IT domain and are written in HTML format. We consider occupation titles as high-level skills (java developer, project manager, ...): the resumes have been labeled with these titles. Since HTML is a structured format, it is easy to extract those occupation titles to automatically label resumes. But occupation titles are not normalized: experts might use different expressions to represent the same occupation. For example: experts might use *DBA, database admin, database administrator or database administration to represent the title “Database administrator”* or *java programmer, java coder, java developer, java engineer, python/java developer to represent the title “java developer”*. Since we have dozens of thousands of resumes, it would be fastidious to normalize all those titles manually. To deal with that problem, we wrote an automated “normalizer” algorithm which operates with a hand written dictionary. The dictionary contains for each normalized competence, a list of expressions which are generally used to identify that competence. Table 1 presents an overview of the dictionary.

The algorithm operates as follows:

1. Iterate over all the expressions in the dictionary and compare each of them to the occupation titles defined in resumes:
2. If an expression matches with one of the occupation titles of the resume then, the corresponding resume is labeled with the normalized competence associated with that expression.

Instead of comparing the whole expression, we compare words because expressions in the dictionary and those in the resume might not match exactly. For example: An expert with a title “developer (java, python)” will be labeled with the competences “java developer” and “python developer” since his occupation title contains all the words of the expression “java developer” and those of

Table 1 Different expressions of occupation titles

Normalized class	Expressions
Python_Developer	python django, python flask, python developer, python programmer, python engineer, python software developer, python software engineer, django developer, django programmer, django engineer, django software developer, django software engineer, python web developer, python application developer, python application engineer
Systems_Administrator	sys admin, sysadmin, systems engineer, system administrator, system admin, systems admin, systems administrator, systems specialist, sys administrator, system engineer, systems engineer
Database_Administrator	database administrator, database engineer, database programmer, database developer, database admin, dba

"python developer." Comparing words instead of the whole expressions also prevents us from putting inside the dictionary all the different expressions that experts might use to express the same occupation; what would be practically impossible.

In this step, we also consider the hierarchical relationship among competences [9]. Figure 3 shows an example of the hierarchical relationship between the high-level competences: *java developer, python developer, front-end developer, web developer and software developer*. It also shows examples of low-level features which describe each competence. We recall that if an expert is a java developer, then he is also a software developer. To handle the hierarchical relationship among competences, we use a simple taxonomy derived from [9] and restricted to the set of normalized competences that we have considered as the output classes. Using the skills taxonomy, the initial set of competences of a resumes is extended by adding their parent competences. For example, based on taxonomy presented in Fig. 3 "software developer" generalizes "python developer" and "java developer." Therefore, "Software Developer" skill will be added to the set of competences of resumes labeled with "java developer" or "python developer."

3.2.2 Training word embedding

Word embeddings are used to build the matrix representation of resumes. The interest of using word embedding mainly resides in the fact that it captures the semantic of words and preserves the semantic similarity between them. It also prevents the problem of data sparseness when it comes to representing text documents. The effectiveness of word vectors has been proven in natural language processing [3]. Embedding the semantic of words into vectors is particularly useful when working with CNN for text classification, because we want each filter to identify closely related words. Given a word vector, the cosine measure can be used to compute the similarity with other word vectors.

We use the skip-gram [20] algorithm to train our own word embedding model from the resume corpus. It has been proved that context-specific word embedding models better capture the relatedness between words than general models [22]. In fact, context-specific corpora are less subject to the problem of polysemous words [28, 29]: a word can have different meanings depending on the context. For example, the term java can mean a coffee or a programming language. Generally, when training word embedding models, two main parameters are considered: the dimension of the word vector and the windows size which refers to the number of words to consider before and after the target word to represent its context. The word embedding model used in this work was built using a dimension of 100 and a windows size of 5. The size of the dimension and the windows size were defined empirically. We observed satisfactory performances using these values. Example 1 shows the most similar words to the term

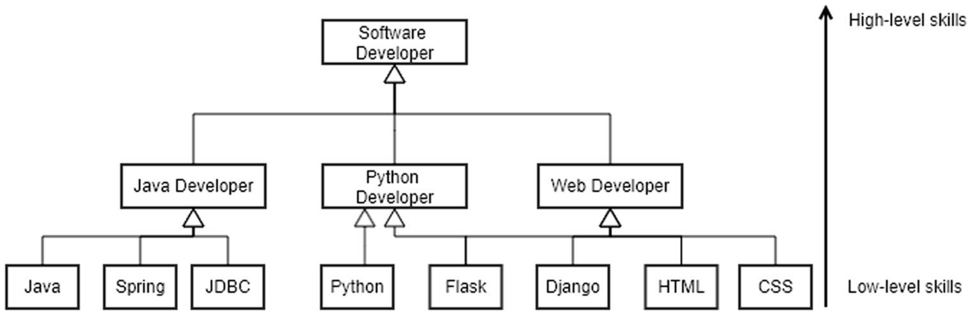


Fig. 3 Example of hierarchical relationship among competences

angular along with their similarity values. From this example, we observe that the most similar terms to angular (angularjs, zepto, vue, angular4, backbone, react, ...) out of hundreds of thousands of words in the vocabulary are either other written forms of angular (angularjs, angular2, angular4) or other javascript frameworks (zepto, backbone, react, ...). This illustrates the effectiveness of our self-trained word embedding model.

Example 1 Most similar words to the word angular in the resume corpus:

[('angularjs', 0.720), ('angular2', 0.718, ('zepto', 0.7039), ('vue', 0.686), ('angular4', 0.6773), ('backbone', 0.666), ('react', 0.663), ('skrollr', 0.658), ('angualr', 0.656), ('colorbox', 0.654)].

3.2.3 Words filtering and resumes transformation into matrices

CNNs were initially designed for images which are represented as 2-dimension matrices. If we want to use CNN to classify texts, then we will have to convert input texts to matrices. The common method to convert a text to a matrix is to convert each of its words in a vector and assemble them into a matrix. The detail method is described in Sect. 3.1.1.

In many text classification tasks, the length of the resulting matrix is generally fixed to the length of the resume which has the maximum number of words. If that method is practical when dealing with small documents like tweets, it can be problematic when dealing with huge documents in terms of number of words. Indeed, a resume could have up to 12 000 words. If we consider that the real numbers in the word vectors are represented in simple precision, and that the dimension of each word vector is set to 100, then we will need about 4.8Mo of memory to represent a resume matrix and 144Go to represent a dataset of about 30 000 resumes. To reduce the total amount of memory used, common methods suggest to fix the length of the resume to a smaller value than the maximum length. Therefore, if a resume has a length greater than that value, then extra words will be removed from it to reach the fixed size. If instead, its length is less than the fixed value, then 0-padding will be added at the bottom of the resume to reach the fixed length. However, the problem with just setting the length is that many relevant terms can be ignored. Instead, in this article, we empirically fixed the size of the resume matrices and considered a domain dictionary where relevant words are selected. The dictionary was established using IT skills gathered from *dice.com*[1] and other collected from the corpus of resumes.

[1] https://www.dice.com/skills.

3.2.4 Building sub-datasets to train base classifiers

To form the training set for each base classifier, the original multi-labeled dataset is divided into n single-labeled datasets (n being the total number of classes in the original dataset). Each resulting sub-dataset corresponds to a binary classification problem focused on a unique competence. To build the single-label dataset D_i, for a target class c_i, the class of each resume example x_j is set to *positive* (1) if in the original dataset the input x_j belongs to the class c_i and to *negative* (encoded by 0) else. This procedure is illustrated in Fig. 4.

Moreover, assuming that we have n classes, and that resumes are equally distributed among them, the dataset used to train a base classifier, will have in average $1/n$ of examples belonging to the target class and the rest belonging to the other classes. For example: if $n = 10$, then 10% of the examples will belong to the target class, and 90% will be of the other classes. That unequal distribution of examples among positive class (target class) and negative class (other classes except target class) can lead the model to a kind of overfitting: the model specializes in learning the features of the negative class in detriment of those of the positive class. To deal with that problem, we equilibrate the dataset accordingly: all the examples belonging to the target competence are retained, and an equal number of examples are randomly selected among the examples of the other classes. The sub-dataset formed is then shuffled before training the corresponding base classifier.

3.3 Model predictions explanation

The main goal of model predictions explanation is to go over the black box predictions by providing a human understandable justification to the model predictions [21]. To justify the model result, we recover words detected by convolution filters and project them on the original resume. We assume that each filter recognizes a single word or an n-gram depending on the filter kernel size. The goal is then to recognize which words each filter identifies, or which terms have led the model to predict a particular class. At the end of this step, the algorithm provides the different terms that might have influenced the classifier output decision. In the case of images classification, this set of features, mainly responsible for the classifier decision is called saliency map [12]. The sentences (explanations) in the form "*the model predicts these classes because the following terms (basic skills) are present in the input resume*" are provided to the user.

Knowing that filters are used to recognize features from the input resumes, and that features can be assimilated to

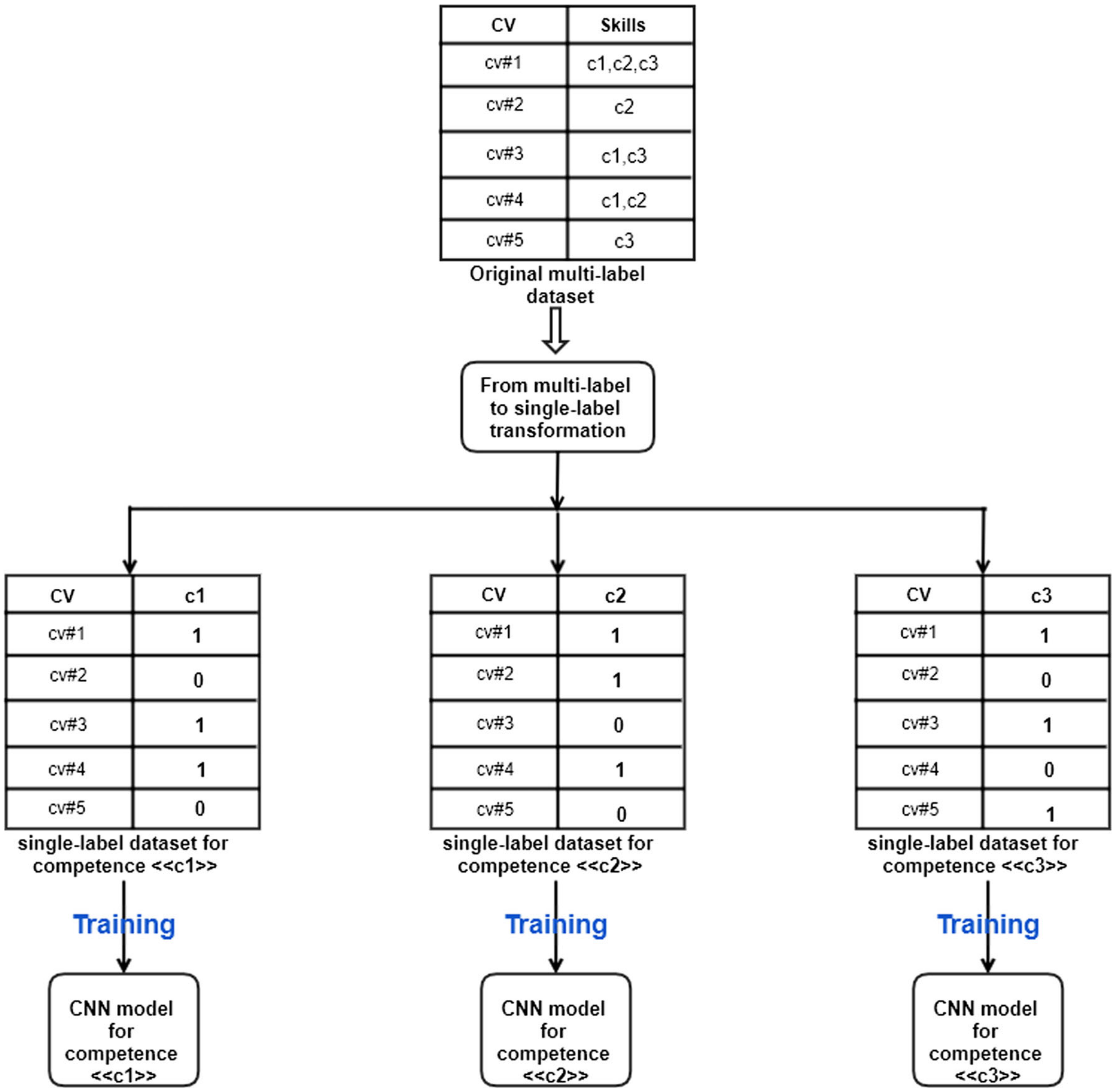

CV
Skills
cv#1
c1,c2,c3
cv#2
c2
cv#3
c1,c3
cv#4
c1,c2
cv#5
c3
Original multi-label dataset
From multi-label to single-label transformation
CV
c1
cv#1 1
cv#2 0
cv#3 1
cv#4 1
cv#5 0
single-label dataset for competence <<c1>>
CV
c2
cv#1 1
cv#2 1
cv#3 0
cv#4 1
cv#5 0
single-label dataset for competence <<c2>>
CV
c3
cv#1 1
cv#2 0
cv#3 1
cv#4 0
cv#5 1
single-label dataset for competence <<c3>>
Training
Training
Training
CNN model for competence <<c1>>
CNN model for competence <<c2>>
CNN model for competence <<c3>>

Fig. 4 From multi-label dataset to single-label datasets

words or n-grams regarding their shape, it is easy to find which words or n-grams were selected by a given filter. In fact, the words or n-grams which were not selected in the max-pooling layer do not influence the classifier decision since the values of their convolution with the filter will not be considered in the next stage (convolution and max-pooling are discussed in detail in Sect. 3.1.2).

Let f_j be a filter of kernel size 1, and $W = w_1, w_2, \ldots, w_n$ the set of words contained in the resume. The word detected by f_j is the word w_j^* which maximizes the scalar product $< w, f_j >$. w_j^* is determined by the formula 8.

$$w_j^* = \arg \max_w (< w, f_j >), w \in W \tag{8}$$

Only those words will influence the classifier decision since they are associated with the max-pooled values of the features maps, and they are those values which are used as input to the fully connected layer. The formula (8) is used to recover the words detected by filters which are then projected on the original resume. From these words, user might recognize the ones which have led the model to predict a particular competence.

4 Results

In this section, we first present the analysis and description of data before discussing the results of the experiments.

4.1 Descriptive data analysis

The experiments were carried out on a collection of 28 707 anonymous resumes available publicly on *indeed.com*[2] website and distributed in 10 classes with the proportions in Fig. 5. The number of pages of resumes ranges from 1 to 6. The extracted resumes are all from the IT domain and are related to the competences: *Software Developer, Front-End Developer, Network Administrator, Web Developer, Project Manager, Database Administrator, Security Analyst, Systems Administrator, Python Developer, Java Developer*. As shown in Fig. 5, the skill *Software developer* is the most represented, while *Python developer* is the least represented. We recall that *Software Developer* includes (*Java Developers, Python Developers, Web Developers* and *Front-End Developer*).

A descriptive text analysis has also been done on resumes of each class to find out the most relevant terms describing each class. To achieve this goal, we evaluated the importance of each term with respect to a competence. TF-IDF (formula 9) was used to evaluate the importance of a term with respect to a competence. The importance of a term w with respect to a competence c, is calculated by Eq. 9. The intuition behind this method of evaluating the importance of terms is that an important term for a competence is a term which appears frequently in resumes labeled with that competence and less frequently in resumes labeled with other competences. In fact, terms which appear frequently in all resumes, are not discriminant enough; that is why the standard frequency is not appropriate since it will highlight several non-discriminant terms.

Let D be the corpus of resumes. Let D_i be the corpus of documents which belongs to the class i. We consider the following definitions:

Definition 4.1 The importance of a term $w \in D$ with respect to a class i is the product of the frequency of that term with respect to the class i, and the inverted document frequency of the term with respect to the class i.

$$\mathrm{TF}-\mathrm{IDF}_i[w] = \mathrm{TF}_i[w] * \mathrm{IDF}_i[w]. \tag{9}$$

Definition 4.2 The frequency of the term w with respect to the class i ($\mathrm{TF}_i[w]$) is the frequency of w in the restricted corpus of resumes labeled with the class i. $\mathrm{TF}_i[w]$ is calculated by the formula 10.

$$\mathrm{TF}_i[w] = \frac{\text{Number of times } w \text{ appears in the corpus } D_i}{\text{Total number of terms in } D_i}. \tag{10}$$

Definition 4.3 The inverted document frequency of the term w with respect to the class i is calculated by the logarithm of the inverse of the document frequency of the term w in the counter corpus (corpus of documents which are not labeled with the class i).

IDF_i is calculated with respect to the counter corpus of D_i in order to penalize terms which appear frequently in resumes which are not of the target class i.

$$\mathrm{IDF}_i[w] = \log\left(\frac{|D|}{|d, d \in D - D_i|}\right). \tag{11}$$

Table 2 shows an overview of the 10 most important terms per class listed among 56000 terms. From this table, we observe that *System_Administrator* and *Network_Administrator* are closely related competences because several of their most important terms (*switch, cisco, firewall, directory, vmware*) are similar. On the other side, we observe that the top-words describing competences *python developer, java developer, security analyst* and *database administrator* do not overlap much with those of other competences. But, the competences *front-end* and *web_developer* have similar top-terms *(css3, responsive, html5, bootstrap*). These observations are confirmed by Table 3 which shows the degree of co-occurrences of the 100 most important terms of each competence. From this table, we can observe that 65 out of the 100 most important terms describing *Front-end Developer* and *Web developer* are the same; and 81 out of the 100 most important terms describing *Network administrator* and those describing *Systems administrator* are the same. *Software developer* has common descriptive terms with other developer-related competences because it generalizes them.

4.2 Evaluation of each CNN model

First, the accuracies of base classifiers are evaluated separately. After, the overall multi-label model is evaluated using appropriate measures. Table 4 shows the accuracy of each individual base classifier. These results were obtained with the following parameters: the number of filters is equal to 100; the filters kernel size set to 1; the stochastic gradient descent used as the optimizer; the batch size equal to 64 with 100 epochs, 0.01 as the learning rate and a momentum of 0.1. Those parameters were found through experiments, and they produced satisfactory results.

These results show that the model performs very well with an accuracy of about 99% for the competences: *Java Developer, Python Developer, Security analyst, Database administrator* and *Project Manager*. In the same time, the model predicts the competences: *Web developer, Front-end Developer, Network administrator* and *System administrator* with an accuracy of around 95%. The high accuracy of the first group can be explained by the fact that those competences can easily be distinguished from others because their low-level features do not overlap much with

[2] https://www.indeed.com (consulted on the 21th *Aug* 2019).

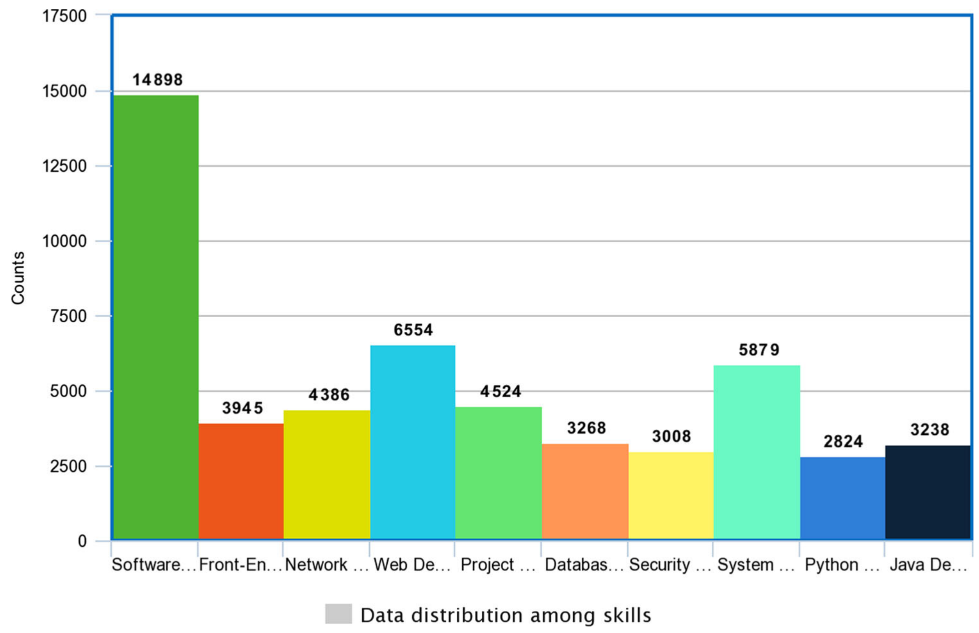


Fig. 5 Classes distribution in the dataset

Table 2 10 most important terms per classes

Classes	10 Most important terms
Software_Developer	('jquery', 0.709), ('git', 0.684), ('api', 0.679), ('ui', 0.679), ('css', 0.674), ('bootstrap', 0.673), ('html5', 0.672), ('front', 0.666), ('javascript', 0.663), ('ajax', 0.657)
Front_End_Developer	('responsive', 1.111), ('front', 1.047), ('sass', 1.031), ('css3', 1.026), ('photoshop', 0.976), ('browser', 0.945), ('html5', 0 .941), ('bootstrap', 0.932), ('react', 0.920), ('ui', 0.917)
Network_Admin	('switch', 1.050), ('cisco', 1.036), ('firewall', 0.953), ('lan', 0.937), ('router', 0.891), ('directory', 0.888), ('wan', 0.888), ('vpn', 0.848), ('vmware', 0.842), ('equipment', 0.805)
Web_Developer	('jquery', 0.813), ('html5', 0.789), ('bootstrap', 0.784), ('css3', 0.765), ('front', 0.754), ('responsive', 0.745), ('css', 0.744), ('website', 0.740), ('page', 0.736), ('wordpress', 0.724)
Database_Administrator	('dba', 1.303), ('tune', 1.015), ('rman', 0.967), ('tuning', 0.942), ('replication', 0.940), ('recovery', 0.908), ('index', 0.880), ('backup', 0.859), ('rac', 0.854), ('administrator', 0.799)
Security_Analyst	('vulnerability', 1.339), ('cyber', 1.141), ('threat', 1.119), ('nist', 1.077), ('analyst', 1.074), ('incident', 1.070), ('remediation', 1.021), ('assessment', 0.987), ('risk', 0.971), ('compliance', 0.970)
Systems_Administrator	('vmware', 0.935), ('directory', 0.918), ('active', 0.801), ('hardware', 0.791), ('administrator', 0.787), ('cisco', 0.784), ('switch', 0.761), ('dns', 0.751), ('backup', 0.738), ('firewall', 0.721)
Python_Developer	('django', 1.815), ('flask', 1.618), ('panda', 1.581), ('numpy', 1.546), ('python', 1.412), ('pycharm', 1.409), ('matplotlib', 1 .395), ('scipy', 1.258), ('2.7', 1.229), ('postgresql', 1.227)
Java_Developer	('hibernate', 1.647), ('j2ee', 1.629), ('jdbc', 1.566), ('servlet', 1.560), ('junit', 1.556), ('jsp', 1.522), ('strut', 1.495), ('maven', 1.490), ('log4j', 1.456), ('spring', 1.450)
Project_Manager	('budget', 1.025), ('leadership', 0.738), ('manager', 0.732), ('vendor', 0.730), ('strategic', 0.717), ('stakeholder', 0.708), ('cost', 0.705), ('risk', 0.693), ('scope', 0.684), ('executive', 0.683)

that of other competences (as discussed in Sect. 4.1). Indeed, many of the terms describing java developers are not the same as those describing python developers and others. But many of the terms (81 out of the 100 most important terms) describing network administrators overlap with those describing System administrators because those competences are closely related (Table 3). As well, terms describing web developers are much the same as those describing front-end developers. Following these observations, some *network administrators* will likely be classified as *systems administrators* and vice versa because both competences involve almost the same basic skills. The same reasoning applies for *web developers* and *front-end developers*. That situation will inevitably lead to ambiguities in both the learning and the evaluation processes because some examples will actually have the

Table 3 Overlapping of most important terms describing competences

	Software Developer	Front-End Developer	Network Admin	Web Developer	Project Manager	Database Admin	Security Analyst	Systems Administrator	Python Developer	Java Developer
Software Developer	100	45	0	76	0	1	0	0	37	37
Front-End Developer	45	100	0	65	1	0	0	0	13	10
Network Admin	0	0	100	0	14	13	21	81	0	0
Web Developer	76	65	0	100	1	0	0	0	30	28
Project Manager	0	1	14	1	100	3	17	15	0	0
Database Admin	1	0	13	0	3	100	3	16	1	3
Security Analyst	0	0	21	0	17	3	100	22	0	0
Systems Admin	0	0	81	0	15	16	22	100	0	0
Python Developer	37	13	0	30	0	1	0	0	100	17
Java Developer	37	10	0	28	0	3	0	0	17	100

Table 4 Accuracy of each base classifier

Classifiers	Accuracy (%)
Java Developer	99.18
Python Developer	99.20
Web Developer	96.30
Front-End Developer	95.18
Network Administrator	94.88
Systems Administrator	95.65
Software Developer	99.08
Security Analyst	99.30
Database administrator	99.11
Project Manager	99.29

characteristics of a competence but not labeled as such: the prediction will be evaluated as incorrect even if in fact it could be correct.

4.3 Evaluation of the overall multi-label model

Let n be the number of instances (resumes) in the test-set, A_i (resp. Z_i) the set of actual (resp. predicted) competences for the instance i. To evaluate the multi-label model, we use three measures generally used in the literature [11]:

4.4 The accuracy

The accuracy for an instance i measures the proportion of competences which were predicted correctly to the total number (predicted and actual) of competences of that instance. Then, the accuracy of the multi-label architecture is the average of accuracies calculated over all the instances. The global accuracy is calculated by the formula 12.

$$A = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i \cap Z_i|}{|A_i \cup Z_i|}. \tag{12}$$

4.5 The precision

The precision for an instance i measures the proportion of competences which were predicted correctly by the model to the total number of predicted competences of that instance. The overall precision is the average of the precisions evaluated over all the instances. The global precision is calculated by the formula 13.

$$A = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i \cap Z_i|}{|Z_i|}. \tag{13}$$

4.6 The recall

The recall for an instance measures the proportion of competences which were predicted correctly by the model to the total number of actual competences of that instance. The overall recall is the average of recalls evaluated over all the instances. The global recall is calculated by the formula 14.

$$A = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i \cap Z_i|}{|A_i|}. \qquad (14)$$

Table 5 shows a comparison of the results of our multi-label classification model with those obtained from other models which are based on different resumes' encoding methods. The recall of 98,79% indicates that our model almost always predicts all the expected competences. On the other hand, the value of the precision shows that 91,34% of competences are well predicted by the model. However, if we can rely on the fact that when a resume is labeled by a competence, it means its owner really has that competence, the opposite is not always true. More precisely, the fact that a resume has not been labeled by a particular competence does not necessarily mean that it does not express that competence. An expert can be labeled as a *System administrator* but actually has skills of a *network administrator* or *project manager*. This could explain the gap between the recall and the precision. Indeed, in many cases, the model predicts a competence based on words it has identified in the resume. But just because the resume was not labeled with that competence, the evaluation will mark it as incorrect. So, in practice, the precision of the model would be greater than that we evaluated on the test set due to incomplete resumes labeling by experts. In addition, even though a high recall and a high precision are preferable in all cases, in the context of job-resume matching, the recall seems to be more important than the precision because job-matching algorithms are not meant to recruit, but to produce a shortlist of resumes in order to ease the recruiter work. Thus, the fact that a job recommender system fails to select good candidates is more a major concern than the fact that it accidentally selects bad ones. In the latter case, we can still rely on human to filter and eliminate unfit candidates.

Table 5 Comparison of our method with other text encoding methods applied on resumes

	Recall (%)	Precision (%)	Accuracy (%)
word2vec+CNN	98.79	91.34	90.22
doc2vec+LR	94.68	81.45	78.63
one-hot+LR	92.28	80.11	78.07

In Table 5, our model (word2vec+CNN) is compared to some models built using other resumes' encoding methods and logistic regression as classifier: (1) one-hot encoding + logistic regression(LR) as base classifiers and (2) doc2vec + logistic regression (LR) as base classifiers. We observe that our model outperforms the model based on "doc2vec encoding" where resumes are transformed into vectors using a self-trained doc2vec model. Our model also outperforms the one based on "one-hot encoding." This is likely because the matrices of words' vectors better preserve the semantic of words and reduce information loss as compared to "doc2vec encoding" where the whole semantic of the resume is embedded into a single vector or to the "one-hot encoding" where the semantic relationship between words is not taken into account. The classifier used (CNN vs LR) also plays a significant role in the model performance.

These results demonstrate the effectiveness of embedding resumes into matrices and using CNN as base classifiers in the multi-label architecture.

4.7 Contribution of word filtering

Here, we analyze the contribution of word filtering in the model performance. When describing the proposed method, we said that instead of selecting any word from the resume, we selected domain related words because they might be more informative than common language words. We will call the method which consist in selecting domain words **DWS** which stands for domain words selection in opposition to the standard method where only common stop words are removed. Figures 6 and 7 show, respectively, the comparisons of the evolution of the overall recall and the precision of both methods (DWS and standard) with respect to the length of the resume. From these figures, we observe that when using the standard filtering method the recall is high while the precision is very low. This can be explained by the fact that many common words were selected to encode the resume into a matrix. This assumption is confirmed by Table 6 which shows the number of words per resumes when no filtering is applied, after stopwords filtering is applied, and after only domain words are selected. Those common words a priori cannot suitably discriminate resumes according to their competences; instead, they might introduce noises and classification errors when training the base classifiers. We also observe that the performances of both method increases with the length of the resume: that means a resume representation with much words increases the number of features used for classification thus improving the overall model performance. In conclusion, we can say that the DWS filtering method presents a better trade-off between

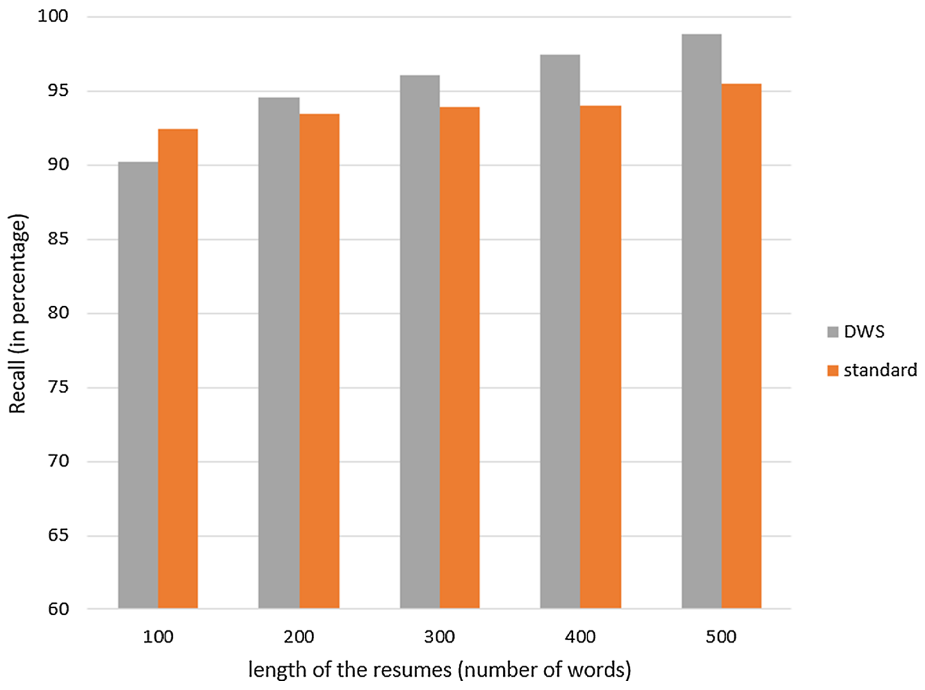


Fig. 6 Recall of DWS filtering versus that of standard filtering

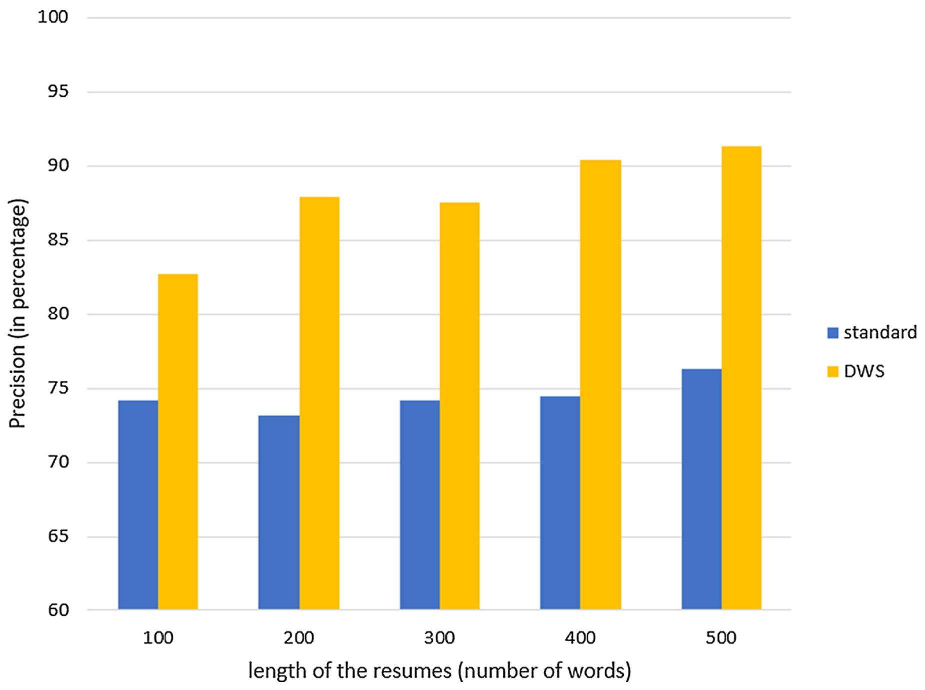


Fig. 7 Precision of DWS filtering versus that of standard filtering

Table 6 Word counts (wc) per resume after filtering

Filtering method	min wc	max wc	avg wc
No filtering applied	17	12479	948
Stopwords filtering	0	9809	724
Domain words selection	0	5526	450

the precision and the recall than the standard filtering method.

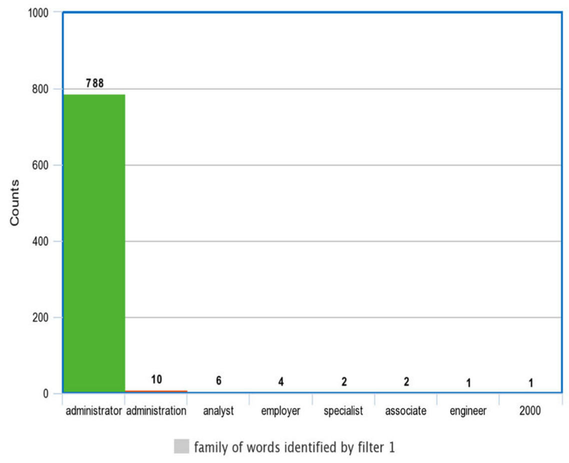
4.8 Understanding the model predictions

In this section, we analyze filters to understand the predictions of each single base CNN classifier. Two different levels of analysis are considered:

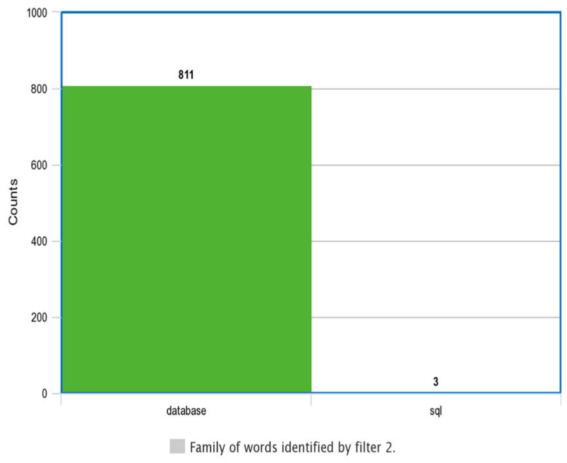
1. First, a filter-level analysis is done to understand which words each filter is specialized in identifying;
2. Secondly, an analysis at the resume level is performed to understand which words were responsible for the classifier decision for an input resume.

4.8.1 Filter-level analysis

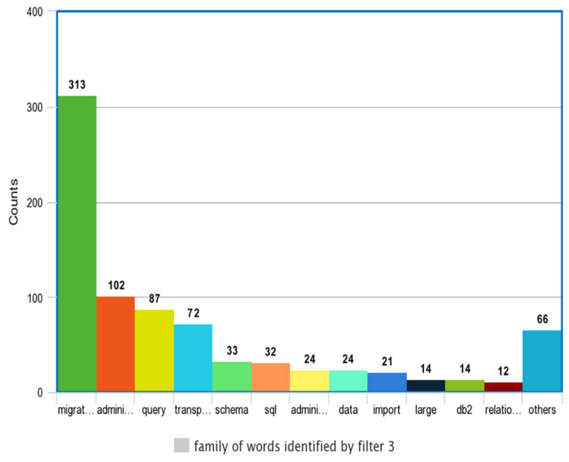
First of all, we would like to determine which words each filter is specialized in detecting. To find out those words, we ran the model on a test set of resumes of a specific competence. Then, we computed the number of times each word was selected by each filter. Figure 8a–d shows the words identified by four filters using a test set of resumes labeled with the competence "database administrator." We observe in Fig. 8a that the filter 1 has detected the word *administration* 788 times over 814 resumes. By observing this result, we can assume that filter 1 is specialized in the detection of the word *administration*. The other words *(analyst, employer, specialist, ...)* were identified in resumes where the word *administration* was not found. Similarly, the filter 2 (Fig. 8b) is specialized in the detection of the word *database* following the same reasoning. Since the test set was restricted to resumes labeled with the "database administrator" competence, it is normal that the term "database" appears in almost all the resumes. That is why that filter selected the word database 811 times over 814. When the word database is not present in a resume, the filter still selects a term closely related to it (sql). Filter 3 (Fig. 8c) mainly detects the word *migration*. In case, this word is not in the resume, the filter selects other words such as *administration, query,...* Filter 4 (presented in Fig. 8d) specializes in detecting the words: *tune, database, tuning, oracle*. We observe that those filters mainly select words closely related to the competence *database administrator*. The filter-level analysis can be used to identify the sets of low-level features that explain each competence.



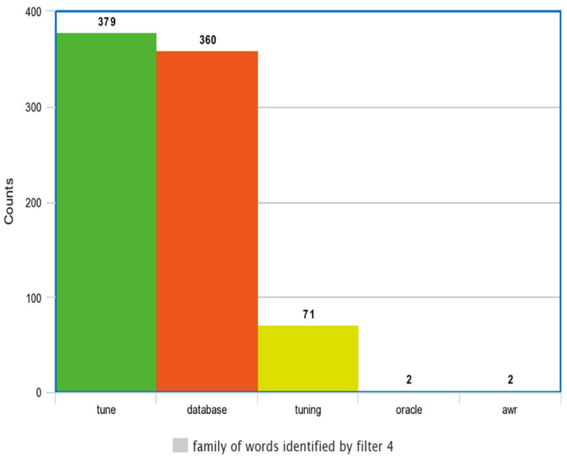
(a) Words selected by filter 1



(b) Words selected by filter 2



(c) Words selected by filter 3



(d) Words selected by filter 4

Fig. 8 Filter analysis for the "Database administrator" specialized model

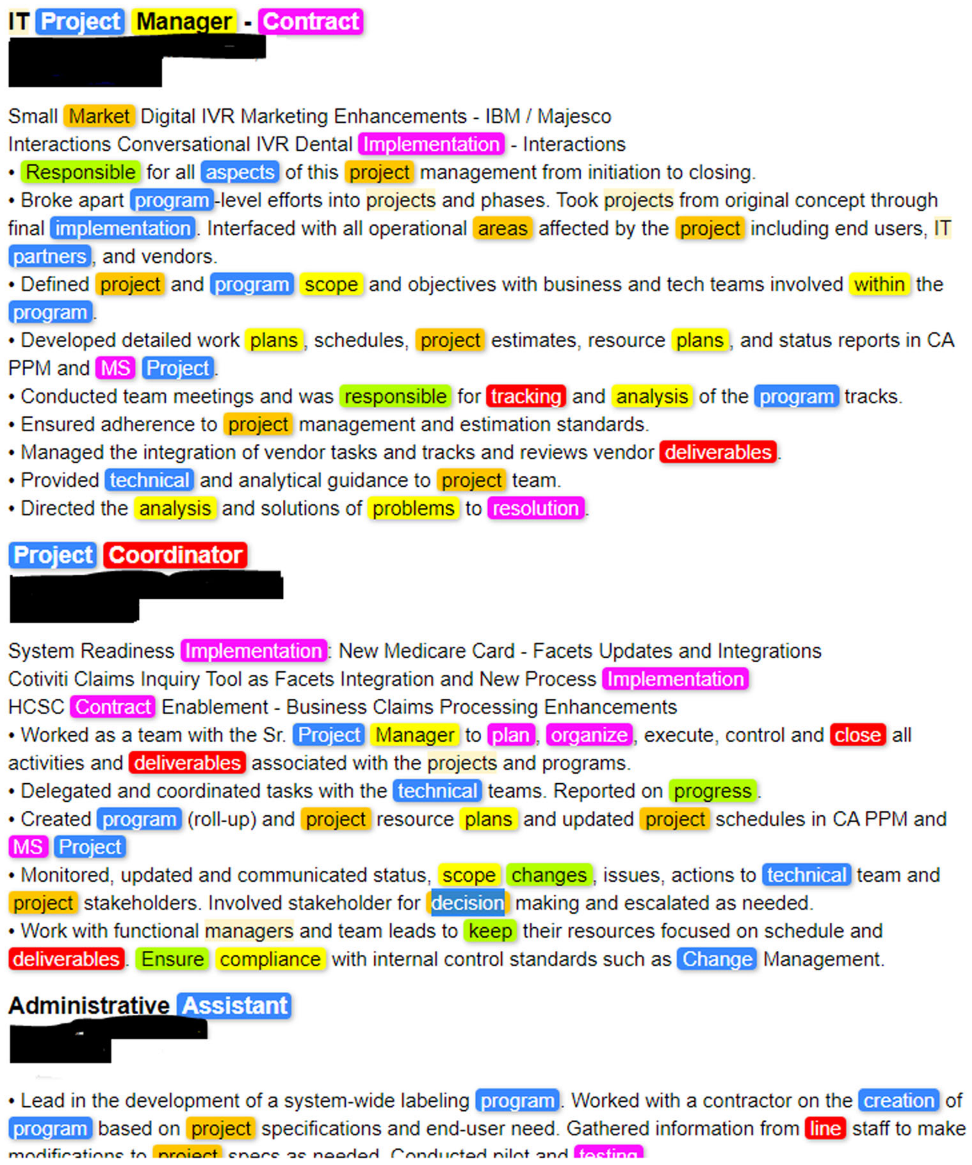


Fig. 9 Words identified by filters in a project manager resume

IT Project Manager - Contract

[redacted]

Small Market Digital IVR Marketing Enhancements - IBM / Majesco
Interactions Conversational IVR Dental Implementation - Interactions
• Responsible for all aspects of this project management from initiation to closing.
• Broke apart program-level efforts into projects and phases. Took projects from original concept through final implementation. Interfaced with all operational areas affected by the project including end users, IT partners, and vendors.
• Defined project and program scope and objectives with business and tech teams involved within the program.
• Developed detailed work plans, schedules, project estimates, resource plans, and status reports in CA PPM and MS Project.
• Conducted team meetings and was responsible for tracking and analysis of the program tracks.
• Ensured adherence to project management and estimation standards.
• Managed the integration of vendor tasks and tracks and reviews vendor deliverables.
• Provided technical and analytical guidance to project team.
• Directed the analysis and solutions of problems to resolution.

Project Coordinator

[redacted]

System Readiness Implementation: New Medicare Card - Facets Updates and Integrations
Cotiviti Claims Inquiry Tool as Facets Integration and New Process Implementation
HCSC Contract Enablement - Business Claims Processing Enhancements
• Worked as a team with the Sr. Project Manager to plan, organize, execute, control and close all activities and deliverables associated with the projects and programs.
• Delegated and coordinated tasks with the technical teams. Reported on progress.
• Created program (roll-up) and project resource plans and updated project schedules in CA PPM and MS Project.
• Monitored, updated and communicated status, scope changes, issues, actions to technical team and project stakeholders. Involved stakeholder for decision making and escalated as needed.
• Work with functional managers and team leads to keep their resources focused on schedule and deliverables. Ensure compliance with internal control standards such as Change Management.

Administrative Assistant

[redacted]

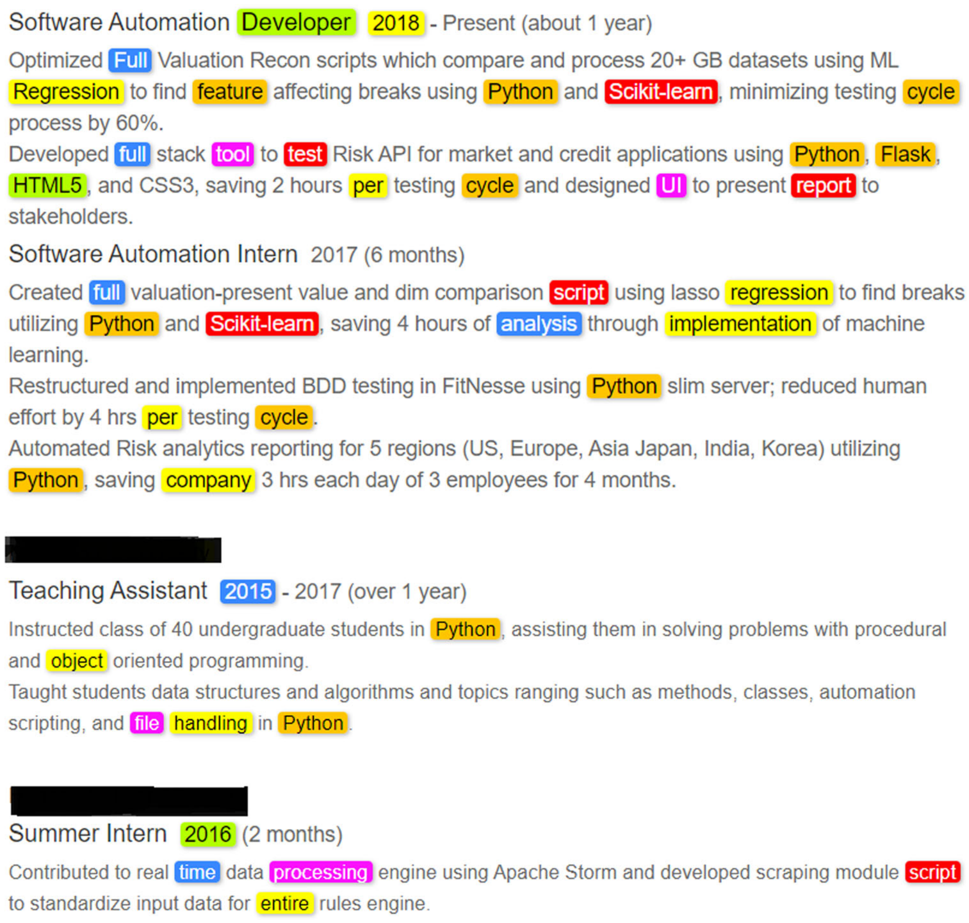
• Lead in the development of a system-wide labeling program. Worked with a contractor on the creation of program based on project specifications and end-user need. Gathered information from line staff to make modifications to project specs as needed. Conducted pilot and testing

4.8.2 Resume-level analysis

After having analyzed the general behavior of filters, we now focus on the explanation of the model prediction for a given input resume. Figure 9 shows the resume of a *project manager* expert where the words selected by the convolution filters have been projected. The label of that resume has been correctly predicted by the classifier. The *multi-highlight browser plugin*[3] was used to highlight the words selected by the filters on the original resume. The colors are related to the frequency of the highlighted words in the resume. We also observe that many words related to project management (project, program, implementation, analysis, resolution, manager, deliverable, plan, organize, ...) were selected by filters. And those words can give to the user an explanation of the classifier decision. From the analysis, the model provides the following explanation to the user: *the expert is classified as a project manger because the terms: project, program, implementation, analysis, resolution, manager, deliverable, plan, organize ...were found in his resume*.

We also consider the case of an incorrect prediction. Figure 10 is a resume predicted by the model as *python developer*, but the title *python developer* does not appear in that resume. We recall that they are those occupation titles that were used to label the resume for training. Figure 10 also shows the words selected by the filters to produce the final decision. In fact, even though the resume was not labeled as a *python developer*, the resume actually

[3] https://chrome.google.com/webstore/detail/multi-highlight/pfgfgjlejbbpfmcfjhdmikihihddeeji: a browser extension used to highlight words in web pages.



Software Automation Developer 2018 - Present (about 1 year)

Optimized Full Valuation Recon scripts which compare and process 20+ GB datasets using ML Regression to find feature affecting breaks using Python and Scikit-learn, minimizing testing cycle process by 60%.
Developed full stack tool to test Risk API for market and credit applications using Python, Flask, HTML5, and CSS3, saving 2 hours per testing cycle and designed UI to present report to stakeholders.

Software Automation Intern 2017 (6 months)

Created full valuation-present value and dim comparison script using lasso regression to find breaks utilizing Python and Scikit-learn, saving 4 hours of analysis through implementation of machine learning.
Restructured and implemented BDD testing in FitNesse using Python slim server; reduced human effort by 4 hrs per testing cycle.
Automated Risk analytics reporting for 5 regions (US, Europe, Asia Japan, India, Korea) utilizing Python, saving company 3 hrs each day of 3 employees for 4 months.

[redacted]

Teaching Assistant 2015 - 2017 (over 1 year)

Instructed class of 40 undergraduate students in Python, assisting them in solving problems with procedural and object oriented programming.
Taught students data structures and algorithms and topics ranging such as methods, classes, automation scripting, and file handling in Python.

[redacted]

Summer Intern 2016 (2 months)

Contributed to real time data processing engine using Apache Storm and developed scraping module script to standardize input data for entire rules engine.

Fig. 10 A resume which was not labeled as "python developer" but predicted by the model as such

expresses python developers skills: *developer, regression, python, scikit-learn, tool, test, script, flask (a python framework)*, etc. So, the model has identified some words characteristic of a *python developer* and has produced his decision based on those words. That is why concerning the model evaluation, we said that, in practice, the precision might be greater than that we evaluated on the dataset due to incomplete resumes labeling. In fact, it is difficult to manually label resumes with all the competences. But when a recruiter reads the resume, he can identify certain competences that might not have been explicitly mentioned by the expert but might be deduced from his experience or other skills he has mentioned. The high recall rate that we obtained, guarantees that whenever a resume will have elements describing a certain competence, the model will almost always be able to predict that competence. Even though the prediction has been marked as incorrect, the following explanation can still be provided to the user: *this expert has python developer competence because the terms: developer, regression, python, scikit-learn, tool, test, script, flask, etc ...are present in his resume*. These terms are in fact closely related to "python development".

5 Conclusion and future work

We have shown the effectiveness of using a multi-label architecture based on CNN to predict high-level competences from resumes: the overall model performs very well reaching 98,79% of recall and 91,34% of precision, achieving more than 99% of accuracy for certain competences. As we have demonstrated in our experiments, in practice, the precision of the overall multi-label architecture might be greater than the evaluated value due to incomplete resume labeling. We also showed that convolutional filters are helpful to conveniently extract terms (low-level features) which explain target skills (classes). Finally, the visualization of words selected by the filters allows human to understand why the model produced its decision.

As for our future work, in addition to the filter's visualization, we will analyze and compute the importance or the contribution of each selected term in the classifier decision. The evaluation of contributions will enable us to sort out selected features with respect to their contribution to the value predicted by the classifier, thus identifying salient features among others.

Data availability statement Resumes repository: https://github.com/florex/resume_corpus.

Code availability The preprocessing codes: https://github.com/florex/preprocessing. The CNN classifiers codes: https://github.com/florex/cnn_classifiers.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Adadi A, Berrada M (2018) Peeking inside the black-box: a survey on explainable artificial intelligence (xai). IEEE Access 6:52138–52160
2. Bansal S, Srivastava A, Arora A (2017) Topic modeling driven content based jobs recommendation engine for recruitment industry. Proc Comput Sci 122:865–872
3. Bian J, Gao B, Liu TY (2014) Knowledge-powered deep learning for word embedding. In: Joint European conference on machine learning and knowledge discovery in databases, pp 132–148. Springer
4. Çelik D, Elçi A (2012) An ontology-based information extraction approach for résumés. In: Joint international conference on pervasive computing and the networked world, pp 165–179. Springer
5. Çelik D, Karakas A, Bal G, Gültunca C, Elçi A, Buluz B, Alevli MC (2013) Towards an information extraction system based on ontology to match resumes and jobs. In: 2013 IEEE 37th annual computer software and applications conference workshops, pp 333–338. IEEE
6. Cohen PR, Kjeldsen R (1987) Information retrieval by constrained spreading activation in semantic networks. Inform Process Manag 23(4):255–268
7. Development AFT (2015) Bridging the skills gap: workforce development is everyone's business n2
8. de Jesús Rubio J (2017) Usnfis: uniform stable neuro fuzzy inference system. Neurocomputing 262:57–66
9. Faliagka E, Liadis L, Karydis I, Rigou M, Sioutas S, Tsakalidis A, Tzimas G (2013) On-line consistent ranking on e-recruitment; seeking the truth behind a well-formed cv. Artif Intell Rev. https://doi.org/10.1007/s10462-013-9414-y
10. Ganzeboom HB, Treiman DJ (1996) Internationally comparable measures of occupational status for the 1988 international standard classification of occupations. Soc Sci Res 25(3):201–239
11. Gibaja E, Ventura S (2014) Multi-label learning: a review of the state of the art and ongoing research. Wiley Int Rev Data Min Knowl Disc 4(6):411–444. https://doi.org/10.1002/widm.1139
12. Guidotti R, Monreale A, Ruggieri S, Turini F, Giannotti F, Pedreschi D (2018) A survey of methods for explaining black box models. ACM Comput Surv 51(5):93:1–93:42. https://doi.org/10.1145/3236009
13. Guo S, Alamudun F, Hammond T (2016) Resumatcher: a personalized resume-job matching system. Expert Syst Appl 60:169–182. https://doi.org/10.1016/j.eswa.2016.04.013
14. Jacovi A, Sar Shalom O, Goldberg Y (2018) Understanding convolutional neural networks for text classification. In: Proceedings of the 2018 EMNLP workshop blackbox NLP: analyzing and interpreting neural networks for NLP, pp 56–65. Association for Computational Linguistics. http://aclweb.org/anthology/W18-5408
15. Javed F, Hoang P, Mahoney T, McNair M (2017) Large-scale occupational skills normalization for online recruitment. In: Twenty-ninth IAAI conference
16. Kenthapadi K, Le B, Venkataraman G (2017) Personalized job recommendation system at linkedin: practical challenges and lessons learned. In: Proceedings of the eleventh ACM conference on recommender systems, pp 346–347. ACM
17. Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1746–1751. Association for computational linguistics, Doha, Qatar. https://doi.org/10.3115/v1/D14-1181
18. Kivimäki I, Panchenko A, Dessy A, Verdegem D, Francq P, Bersini H, Saerens M (2013) A graph-based approach to skill extraction from text. In: Proceedings of TextGraphs-8 graph-based methods for natural language processing, pp 79–87
19. Meda-Campaña JA (2018) On the estimation and control of nonlinear systems with parametric uncertainties and noisy outputs. IEEE Access 6:31968–31973
20. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) Advances in neural information processing systems, vol 26, pp 3111–3119. Curran Associates, Inc. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
21. Miller T (2019) Explanation in artificial intelligence: insights from the social sciences. Artif Intell 267:1–38. https://doi.org/10.1016/j.artint.2018.07.007
22. Nooralahzadeh F, Øvrelid L, Lønning JT (2018) Evaluation of domain-specific word embeddings using knowledge resources. In: Proceedings of the eleventh international conference on language resources and evaluation (LREC 2018)
23. Ribeiro MT, Singh S, Guestrin C (2016) "Why should i trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22Nd ACM SIGKDD international conference on knowledge discovery and data mining, KDD '16, pp 1135–1144. ACM, New York, NY, USA. https://doi.org/10.1145/2939672.2939778
24. Rubio DJJ (2009) Sofmls: online self-organizing fuzzy modified least-squares network. IEEE Trans Fuzzy Syst 17(6):1296–1309. https://doi.org/10.1109/TFUZZ.2009.2029569
25. Rubio J (2017) Usnfis: uniform stable neuro fuzzy inference system. Neurocomputing. https://doi.org/10.1016/j.neucom.2016.08.150
26. Sayfullina L, Malmi E, Kannala J (2018) Learning representations for soft skill matching. In: International conference on analysis of images, social networks and texts, pp 141–152. Springer
27. Shalaby W, AlAila B, Korayem M, Pournajaf L, AlJadda K, Quinn S, Zadrozny W (2017) Help me find a job: a graph-based approach for job recommendation at scale. In: 2017 IEEE international conference on big data (big data), pp 1544–1553. IEEE
28. Shi H, Wang X, Sun Y, Hu J (2018) Constructing high quality sense-specific corpus and word embedding via unsupervised elimination of pseudo multi-sense. In: Proceedings of the eleventh international conference on language resources and evaluation (LREC 2018)

29. Zhang L, Wang S, Liu B (2018) Deep learning for sentiment analysis: a survey. Wiley Interdiscip Rev Data Min Knowl Discov 8(4):e1253
30. Zhao M, Javed F, Jacob F, McNair M (2015) SKILL: a system for skill identification and normalization. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence, January 25–30, 2015, Austin, Texas, USA, pp 4012–4018