NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados

Master Program in Advanced Analytics

Landing on the right job: a Machine Learning approach to match candidates with jobs applying semantic embeddings

Luís Matos Pombo

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Landing on the right job: a Machine Learning approach to match candidates with jobs applying semantic embeddings

Luís Matos Pombo

Work project presented as partial requirement for obtaining the Master's degree in Advanced Analytics

Title: Landing on the right job Luís Matos Pombo MAA

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

TITLE: LANDING ON THE RIGHT JOB: A MACHINE LEARNING APPROACH TO MATCH CANDIDATES WITH JOBS APPLYING SEMANTIC EMBEDDINGS

by

Luís Matos Pombo

Work project presented as partial requirement for obtaining the Master’s degree in Information Management, with a specialization in Advanced Analytics

Advisor: Leonardo Vanneschi

ABSTRACT

Job application’ screening is a challenging and time-consuming task to execute manually. For recruiting companies such as Landing.Jobs it poses constraints on the ability to scale the business. Some systems have been built for assisting recruiters screening applications but they tend to overlook the challenges related with natural language. On the other side, most people nowadays specially in the IT-sector use the Internet to look for jobs, however, given the huge amount of job postings online, it can be complicated for a candidate to short-list the right ones for applying to. In this work we test a collection of Machine Learning algorithms and through the usage of cross-validation we calibrate the most important hyper-parameters of each algorithm. The learning algorithms attempt to learn what makes a successful match between candidate profile and job requirements using for training historical data of selected/reject applications in the screening phase. The features we use for building our models include the similarities between the job requirements and the candidate profile in dimensions such as skills, profession, location and a set of job features which intend to capture the experience level, salary expectations, among others. In a first set of experiments, our best results emerge from the application of the Multilayer Perceptron algorithm (also known as Feed-Forward Neural Networks). After this, we improve the skills-matching feature by applying techniques for semantically embedding required/offered skills in order to tackle problems such as synonyms and typos which artificially degrade the similarity between job profile and candidate profile and degrade the overall quality of the results. Through the usage of word2vec algorithm for embedding skills and Multilayer Perceptron to learn the overall matching we obtain our best results. We believe our results could be even further improved by extending the idea of semantic embedding to other features and by finding candidates with similar job preferences with the target candidate and building upon that a richer presentation of the candidate profile. We consider that the final model we present in this work can be deployed in production as a first-level tool for doing the heavy-lifting of screening all applications, then passing the top N matches for manual inspection. Also, the results of our model can be used to complement any recommendation system in place by simply running the model encoding the profile of all candidates in the database upon any new job opening and recommend the jobs to the candidates which yield higher matching probability.

Keywords: job matching; recommendation systems; semantics; machine learning, word2vec

ACKNOWLEDGEMENTS

To my parents and brother, for their unconditional love.

A special thank you to Kira for her invaluable support, in so many ways.

To my dog Pierre, for being my company in the lonely days.

To Prof. Vanneschi for his inspirational classes.

To Kristen and Asimina, for their friendship and companionship through this master program and beyond.

To Illya for his Computation I classes, a huge asset which has opened my opportunities.

To my fellow colleagues Ariana, Esdras, Bernardo, Maria and Nicolas for the great experience of working with such smart and kind people.

To Landing.job for being open to the challenge and for facilitating the data for our work.

Index

Table of figures

A. Introduction

In the field of job recruitment, Internet has become a major channel for recruiters to publish job postings and attract candidates, whereas looking and applying for jobs have become mostly tasks that candidates perform online(Malinowski, Keim, Wendt, & Weitzel, 2006). While Internet certainly allows reaching a wider audience on the recruiters' standpoint and makes job applications more convenient on the candidates' standpoint, two problems arise: on one side, there is a tremendous amount of job postings online in different job portals and company's career sections. This situation makes the candidate's task of short-listing jobs a tedious and time-consuming one, often deriving in sub-optimal short-listing. On the other side, certain job postings can attract hundreds or thousands of applications, making the recruiter's task of screening all applications very challenging under businesses deadlines. To tackle the problem of job of application screening, numerous systems have been built which relied on Boolean search to filter out applications based on the inexistence of certain keywords. This technique presents several shortcomings such as ignoring the problems related with natural language including semantics and synonyms (Singh , Rose, Visweswariah , Vijil , & Kambhatla , 2010). Some more recent approaches to the problem relied on finding similarities between the applicant profile and the job profile across a multitude of dimensions such as skills, education or experience and ranking applications according to the overall similarity degree (Fazel-Zarandi & Fox, 2009), (Singh , Rose, Visweswariah , Vijil , & Kambhatla , 2010). Another approach, which uses Machine Learning was presented in (Faliagka, Ramantas, Tsakalidis, & Tzimas , 2012). However, we consider that considerable improvements to the proposed models could be reached by using start-of-the-art techniques to obtain a better representation of the jobs and the candidates' profiles.

With regard to candidate attraction, different job recommendation systems were developed. For instance, (Paparrizos, Cambazoglu, & Gionis, 2011) built a Content-Based Recommendation System to predict the next job for a candidate. In (Yuan, et al., 2016) the authors use a Collaborative Filtering approach while performing a semantic embedding of job profiles. Yet another example which uses a Hybrid Recommendation approach can be found in (Hong, Zheng, & Wang, 2013) where the authors utilize clustering technique for clustering users and build specific recommendation systems for each cluster of users.

In this work we present a Machine Learning approach for building a model to automatically screen job application but that can also be used to complement job recommendation system. Our approach can be regarded as a hybrid one. On one side, it incorporates aspects of content-based systems because we use attributes of jobs positions and candidates for building features. On the other side, it could also be seen as an Ontology-based system as we define a set of matchings between jobs and candidates' profiles based on the relationship that we manually draw between the attributes of these entities.

Our Machine Learning model learns what makes a successful application to a job. We achieve that by setting the target of our model as Boolean value encoding whether a certain application was pre-selected in the screening phase or not.

One of the major challenges is the proper encoding of jobs' and candidates' profiles given that many attributes are presented in natural language. To tackle this challenge, we apply tentatively a different set of strategies: TF-IDF, Latent Semantic Analysis and the start-of-the-art Word2Vec algorithm. The usage of Latent Semantic Analysis and Word2Vec has improved the overall results of our model because contrary to lexical features as in TF-IDF, these techniques explore the context of word, therefore being more resilient to problem such as synonyms which artificially degrade the similarity between the required profile for the job and the profile of the applicant.

Our model could also be deployed for purposes of job recommendation as well. In practice the difference between using the model for screening or for job recommendation is that in the former case there is an actual job application while in the latter case there is not an actual application but the potential candidate is already sitting in the database. Thus, one could ask the following question: what is the probability of person A to be a good match for job X, were person A willing to apply to it? By generalizing this question to every potential candidate in the database and every new job position, we end up with a list of top matches for each job position, therefore recommending the jobs of highest matching probability to the corresponding people.

Finally, we would like to mention that this work was based on the anonymous data provided by Landing.Jobs, an IT-specialized job portal. By using the model that we have developed, the company turn application screening both cheaper and more scalable while being able to integrate their current recommendation system with our model's output to further refine recommendations.

B. Literature review

The evolution of technology has changed that way people find and apply for jobs and the way organizations attract and receive applications. The recruitment process nowadays relies heavily on the internet as a means of communication to publish job openings, to attract applicants and to receive applications, be it through job portals or organizations' own websites career section (Malinowski , Wendt , Keim , & Weitzel, 2006), (Desai, Bahl, Vibhandik, & Fatma, 2017). At the same time, looking for jobs online became a natural approach for most people. In job portals such as LinkedIn or Landing.jobs people create their profiles and apply for jobs using their standard profile, sometimes adding motivation letter or their custom résumé/ CV (curriculum vitae). Often, organizations' own websites career section also allows people to create their profile to apply for multiple jobs within the organization using it. In both job portal or organizations' own websites career section it is common to be able to activate and receive alerts of jobs matching people's preferences and profile.

While the internet has made the application for jobs more convenient and candidates often regard the greater number of applications for jobs they make, the greater the chance of being selected for any, it sometimes leads to a huge amount of applications per job that the recruiter will have to shortlist from (Singh , Rose, Visweswariah , Vijil , & Kambhatla , 2010).

From another perspective, as the Internet has become a great channel for reaching and attracting a large audience, most jobs are nowadays published online, which leads to an amalgam of job postings through which the applicant has to browse through until he/she finds the ones that match his/her preferences and he/she sees has potential for success. This scenario often leads to applications for jobs that do not match so well applicant's profile (Alotaibi, 2012).

In the recruitment process context, some authors (Färber, Weitzel, & Keim, 2003) divide the recruitment process in two main phases: attraction and selection. Attraction is concerned with what happens before an application, including employer branding, job publishing and approaching potential candidates. Selection, on the other side, concern the steps after a candidate has applied. Upon the receival of applications what follows typically is a screening process intended to shortlist applications to pass to later stages where usually one or multiple rounds of interviews are conducted, tests are sometimes required and hiring decisions are made (Alotaibi, 2012). Figure X provides a generic example of a recruitment process.

In regard to the illustrated scenario, two problems arise: 1) how to efficiently screen a massive amount of applications and 2) how organizations can automatically recommend the right jobs openings to the right people. We call the former the screening problem and the later the recommendation problem.

The screening problem

To address the screening problem, organizations apply different strategies. Some opt for a purely manual approach where each application is reviewed by a human, sometimes outsourcing hiring firms to screen and shortlist candidates. To illustrate the challenge of screening applications, take the following example. Say candidate A possesses most skills required for a job but misses some critical ones, while candidate β has poorer skill set but is highly expert in some of the critical ones and yet candidate C is highly versatile, including critical skills but has only few years of experience in each. The decision to shortlist any of these candidates is not obvious and ideally should be only taken after reviewing all applications or otherwise the best application might not be found at all in the pile of all applications (Singh , Rose, Visweswariah , Vijil , & Kambhatla , 2010). Moreover, these decisions are taken under the pressure of business deadlines. This illustration presents three candidates but in practice it is hundreds or thousands of applications to be dealt by a human. That is why some organization introduce some form of Information Systems (IS) to assist on the task of screening applications. Numerous implementations relied on Boolean search, using keywords to search and filter out applications in the database which did not match those keywords (Alotaibi, 2012). However, this technique falls short on the task because it lacks the capabilities to deal with problems related with natural language such as synonyms or to capture underlying attributes such as personality trait, which according to (Singh , Rose, Visweswariah , Vijil , & Kambhatla , 2010) is the hidden reason why a large number of applications present such low compatibility with job search.

More recently, new systems have been developed. For instance, in (Singh , Rose, Visweswariah , Vijil , & Kambhatla , 2010) the authors present a system which uses Information Extraction techniques to automatically mine résumés and job profiles to later rank candidates. The ranking of candidates is based on the similarity between the job requirements and the applicant profile on different dimensions such as skills, education and experience. The recruiter can further refine the ranking by filtering per level of education or skills. While this system performs quite favorably in terms of feature extraction, it does not solve the problem of natural language. Another example of a screening system is the one introduced in (Faliagka, Ramantas, Tsakalidis, & Tzimas , 2012). The proposed model is built by means of Machine Learning, applying Support Vector Regression (SVR) algorithm to learn to rank candidates. For training the model, the authors used a set of features extracted from the candidate's LinkedIn profile, such as years of education, work experience, average number of years per job as well as extraversion which is a derived feature from mining applicant's blog posts. The model as a whole learns how to rank candidates using previous screening decisions, however given recent developments in the field of Natural Language Processing we consider that the usage of advanced techniques such as word2vec could improve results by dealing with problems such as synonyms and polysemy which are latent problems when it comes to use natural language in Machine Learning contexts.

The job recommendation problem

When a potential candidate is looking for a job, searching online has become for most people a natural way. However, searching through hundreds or thousands of jobs' postings, a situation designated as information overload, can be overwhelming. The potential candidate finds a huge collection of jobs in different career portals and recruiter's websites which makes the selections of positions to apply to a complex and time-consuming task. Job recommendation systems are meant to tackle that problem by providing a list of job positions to a candidate which best reflects his/her preferences and profile. Recommendation systems are widely employed for many applications, such as recommending books or movies –generally called 'items' – and they have been applied in the job market for more than a decade (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013). Technically, recommendation systems have been split in many categories but the following three categories have been the most common:

a) Content-based Recommendation (CBR) – the idea of CBR is to suggest items to users based on similarity between the user profile and the item information. (Paparrizos, Cambazoglu, & Gionis, 2011) used this approach to build a Machine Learning model to predict the next job of a candidate using both the candidate's profile information as well the information of the companies where the candidate had work previously. One of the challenges of CBR is the so-called overspecialization, the phenomenon in which candidatesl receive recommendations of jobs whose profiles contain multiple attributes similar to the candidates own profiles. profile while not receiving recommendation of other type of jobs which they may like more (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013).

b) Collaborative Filtering Recommendation (CFR) – this technique relies on finding people similar to the target person and recommending items which similar people have liked. The similarities in Collaborative Filtering technique are concerned with people's tastes, preferences and activities, contrary to CBR where similarity is built on top of the content of the person and the job profile. In the context of job recommendations, CFR usually relies on data about the person's activities such as job applications, job posting clicks and ratings. It is rare to find a recommendation system which relies solely on Collaborative Filtering. Among its challenges are the cold-start problems for new users and items. When a new person registers in a job portal she has never applied for a job there, therefore it is not possible to recommend her a job that someone as with the same preferences has applied to before. At the same time, when there is a new job opening, by definition no one has applied yet, so it is not possible to recommend that job to person A because no other person applied to it yet. A body of literature has attempted to solve the problem in different ways, one of which through deep learning as proposed in (Yuan, et al., 2016), where the authors build a model which learns the similarity between a new job profile and an existing one with prior applications utilizing doc2vec which is considered the current state-of-the-art deep learning algorithm for document embedding and matching. Therefore, the system can recommend the new job based on prior applications to similar jobs content-wise.

c) Hybrid Recommendation (HyR) – as seen already, all recommendation techniques have some shortcomings, therefore it is rare to find a recommendation system that uses only one technique. The hybrid approach combines different techniques to overcome the specific problems of each. In the work of (Hong, Zheng, & Wang, 2013), the authors propose clustering users into three groups based on their activity (pro-actives, passives and in-between) and apply a different recommendation technique on each group. In another work (Shalaby, et al., 2018), the authors attempt to address the challenge of cold-start in Collaborative Filtering and the rigidity of Content-based techniques. To achieve this, on one side, content-similarity between jobs is learnt by means of Machine Learning. On the other side, the authors use a statistical approach to estimate the likelihood of candidate applying for a job given their prior interactions. These intermediary results are then combined and correlated with the candidate profile and activity to provide the specific recommendations.

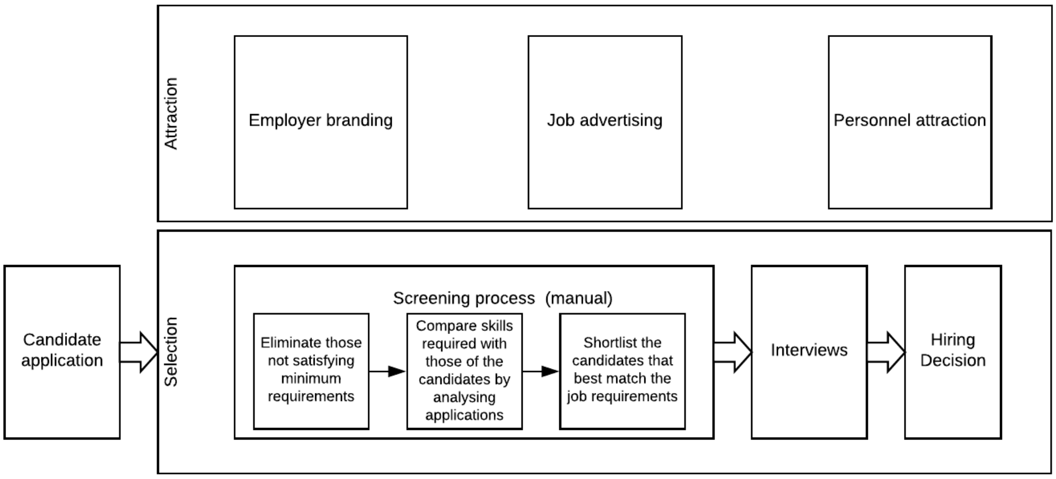

Attraction
Employer branding
Job advertising
Personnel attraction
Candidate application
Selection
Screening process (manual)
Eliminate those not satisfying minimum requirements
Compare skills required with those of the candidates by analysing applications
Shortlist the candidates that best match the job requirements
Interviews
Hiring Decision

Figure 1: Recruitment process adapted from PROSPECT and JRS Survey

Regardless of the conceptual division between screening and recommendation, in a broader picture, the recruitment goal is to bring valuable people for the organization to fulfill its needs. Clearly a good match between people and jobs needs to consider both the preferences of the recruiter and the preferences of the candidate. It is the perspective of this work that the screening problem and the recommendation problem should be tackled as a whole. In the work of (Malinowski , Wendt , Keim , & Weitzel, 2006) the authors followed a similar idea by developing two complementary models. The first one aims to recommend people profiles (CV-recommender) that are similar to other people's profiles previously selected by the recruiter and the second one (Job-recommender) aims to recommend jobs to people who have expressed their preference for similar jobs in the past. In the end, the authors acknowledge the need to aggregate the recommendations generated independently and propose an approach where one the first step the top candidates in terms of bilateral matching are chosen and on the second step these candidates are ranked based on their job preferences. Another work which relates to our approach is the one in (Fazel-Zarandi & Fox, 2009) who identified a set of features such as must-have skills, secondary skills, education and job experience and found the similarity degree between the job profile and applicant profile on each of these features. The work concludes by ranking the candidates based on an aggregation of the similarities through the different dimensions, whereas in our work we use machine learning to learn the parameters of a function to match the job profile and the candidate profile with a multi-dimensional input.

Lastly, in the work of (Yuan, et al., 2016), the authors find the similarities between job profiles based on the semantic representation of job profiles through means of the novel application of the doc2vec algorithm. This relates to our work in the sense that we also apply a variant of doc2vec, the word2vec algorithm to learn the semantic representation of skills and we use that representation to find the degree of similarity between applicant profile and job profile in terms of required/offered skills. Moreover, the similarity extracted from skills embedding among with a collection of other features are treated as input data for our Machine Learning application which learns what is a successful application from historical job applications.

C. Technical background

In this section we provide the technical background required in the context of this project. We describe in a high-level the variety of methods explored, the respective science fields and how these methods relate to each other. The topic-subtopic scheme that we lay here is one of many others schemes that could be drawn, as the respective science fields overlap in many domains. The details and the proper tuning of the hyper-parameter of each method in the context of our problem and data, are further explored in the experimental section.

1. Statistics

1.1. Latent Semantic Analysis (LSA)

LSA is a procedure which belongs to the Statistical subfield of Distributional Semantics. The goal of LSA is to extract the underlying concepts in a collection of documents and its respective words. LSA assumes that words that are close in meaning will occur in similar pieces of text. The word frequency (sometimes multiplied by the inverse of the number of documents where the words appear) is drawn from each document and built into a matrix A. This matrix is then decomposed through Single Vector Decomposition (SVD). The resulting matrix *M* is a new representation of the word/document matrix. Formally:

$$M = U\Sigma V^T,$$

where *U* is the unitary matrix of *A (eigenvectors)*, V^T is the conjugate transpose of the unitary matrix of A and Σ is the diagonal matrix composed with the eigenvalues of the unitary matrix of A.

To complete the LSA procedure, one applies a low-rank approximation of *M.* This is done by selecting the *k* highest eigenvalues in matrix Σ, and subject to a minimization procedure to recreate M with a *k* rank.

$$minimize\ over\ \widehat{M}\quad \| \mathrm{M} - \widehat{M} \|\ \ subject\ to\ rank\big(\widehat{M}\big) \leq k$$

The process of low-rank approximation of *M* (LSA) mitigates the problem of identifying synonymy, as the rank lowering is expected to merge the dimensions associated with terms that have similar meanings, and with limited results, to mitigate polysemy problems (Pottengerb & Kontostathis, 2006).

In this project we use LSA to tackle the problem of synonymies when matching the required skills by the job offer with the offered skills of the applicant.

1.2 Variance Inflation Factor (VIF)

VIF is used as a measure the severity of multicollinearity among variables. Multicollinearity causes biased estimation, coefficient estimation instability and is a considerable obstacle to most machine-learning techniques (Dumancas & Ghalib, 2015). Collinearity is most commonly intrinsic, meaning that collinear variables are different manifestations of the same underlying construct or latent variable (Dormann, 2013).

Formally,

$$VIF_m = \frac{1}{1 - R^2},$$

where VIF_m is the VIF of the m^{th} variable, and R^2 is the coefficient of determination computed from regressing variable m against the remaining ones.

The square root of VIF indicates how much larger the standard error of the variable coefficient estimation is in comparison to what it would be, were the variable uncorrelated with the other explanatory variables (Allison, 1999).

In this project, we use Variance Inflation Factor to assess the multicollinearity of our variables for the purpose of model stability.

2. Machine Learning

Machine learning is a field of statistics and computer science that gives computer systems the ability to progressively improve performance on a specific task with data, without being explicitly programmed. It explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions through building a model from sample inputs (Samuel, 1959).

2.1 Supervised Learning

Supervised learning is a subfield of Machine Learning which focus on the search of algorithms that learn from labeled data to produce general hypotheses, which then make predictions about future instances. Specifically, the goal of supervised learning is to model the distribution of the class labels given the input features. The resulting model is used to predict the class label (labels in multilabel classification) of unseen instances made of the same features (Kotsiantis, 2007).

2.1.1 Logistic Regression(LR) / Single Layer Perceptron (SLP)

Logistic Regression derive both from the field of Statistics and Machine Learning. In this work, we use LR and SLP terms interchangeably. Under the Machine Learning perspective, a Logistic Regression is a special case of the Perceptron where in Statistics it is considered a special case of the Generalized Linear Regression.

Logistic regression model computes the class membership probability for one of the two categories in input vector. In matrix notation:

$$p(Y = 1|X) = \frac{1}{1 + e^{-\theta X}},$$

where p$(Y = 1|X)$ represents the probability 1 given the input matrix X and θ is the matrix of estimation coefficients.

The problem can be represented as a minimization problem of distance/error between the logit and the true label. A common error function (also called cost function or fitness function) is the Cross-Entropy. Formally:

$$E = Y \log\left(\hat{Y}\right) + (1 - Y) \log\left(1 - \hat{Y}\right),$$

where E is the entropy, Y is the true label and $\hat{Y}$ is the class estimation. The function can be minimized by multiple mean, a common one being the gradient descent (Dreiseitl & Ohno-Machado, 2002).

The advantages of LR include its weighs interpretations and model readability. They can be considered a decent first-try when we don’t know whether the classes are linearly separable or not in terms of the explanatory variables. However, for the cases where it is not possible to come up with a straight line or plane to separate the classes, the Perceptron falls short, as the model will never be able to classify all instances properly (Kotsiantis, 2007).

In this project we use LR as our first approach to create a job matching model.

2.1.2 Multilayer perceptron (MLP) / Artificial Neural Networks (NN)

Multilayered Perceptron have been created to try to solve the problem of non-linearity of class separation. (Rumelhart, Hinton, & Williams, 1986). MLP can be considered a generic case of LR where between the input data and the logistic function we have intermediary/nested functions. In other words, a MLP consists of a set of functions joined together in a pattern of connections. To each function under the MLP context, the term neuron is frequently employed as the visual representation and inspiration is loosely associated with the brain. Input data is commonly called input layer, the model prediction called output layer and the intermediary layers called hidden layers. The term MLP derived afterwards in (Artificial) Neural Networks. There is a plentitude of Neural Networks topologies, but in the specific case of Feed Forward Neural Networks, where the output of each neuron travels only forward.

Learning with MLP was made possible, because of the Backpropagation algorithm (Rumelhart, Hinton, & Williams, 1986). It works by retro-propagating the error E between the model estimation $\hat{y}$ and the true value y, through the whole network.

In the output layer we differentiate the error function in order of x_j to obtain the gradient to update the respective parameter:

$$\frac{dE}{dx_j} = \frac{dE}{dy_j} \frac{dy_j}{dx_j}$$

For the hidden units, we don't know the value of x but we know the outputs of the previous layer neurons so we can compute how E is affected by the output of the previous layer and their parameters. Generically:

$$\Delta w = \alpha \frac{dE}{dw},$$

where Δw is the change in the model parameters, $\frac{dE}{dw}$ is the derivative of the error in order of w and α is a parameter controlling the update speed, so-called, learning-rate.

One way to see backpropagation is as a generalization of the delta rule, by means of the chain rule to iteratively compute gradients for each layer so as to adjust the model parameters.

MLP are a universal approximator. This means that they are capable of approximating virtually any function, with just a single hidden layer, provided that the function has a limited number of discontinuities and the number of neurons employed large enough, but not infinite (Hornik, 1989).

There is a plentitude of Neural Networks topologies. In this project we focus on the Feed Forward Neural Networks with a single hidden layer.

2.1.3 Decision trees

Decision trees is a learning algorithm which represents the mapping between inputs and outputs in a tree-like structure. This algorithm repeatedly splits the inputs using the feature that maximizes the separation of the data. Each node in a decision tree represents a feature and each branch represents a value that the node can assume. The feature that best divides the training data is the root node of the tree (Kotsiantis, 2007).

There are several methods to find the features that best splits the training data, such as Information Gain (Hunt, Martin, & Stone, 1966) and Gini Index (Breiman, Friedman, Stone, & Olshen, 1984).

$$Information\ Gain{:}\ \ G(T \mid a) = H(T) - H(T|a),$$

where $H(T)$ is the model entropy and $H(T|a)$ is the model entropy when further splitting based on the weighted entropy of the values of feature a.

$$Gini\ Index{:}\ \ I_G(p) = 1 - \sum_{i=1}^{J} p_i^2,$$

where p is the fraction of items labeled with class {\displaystyle i}ii in the set and J is the number of classes.

ReliefF algorithm is a splitting method which works a little different than the former two. ReliefF selects a feature not on the feature alone but in the context of other features. However, a majority of studies have concluded that there is no single best method (Murthy, 1998).

Decision Trees can be used both for classification and regression problems. Also, features can be either categorical or continuous. If a feature is continuous, it is implicitly discretized in the splitting process (Dreiseitl & Ohno-Machado, 2002).

By the nature of the algorithm, Decision Trees will keep on splitting the training data until all instances are correctly predicted which can create a very big model, with plenty of nodes and each node gradually with less and less instances. Such behavior can lead to overfitting so it is important to establish an adequate stop criterion. Common criteria include stopping when each child would contain less than five data points, or when splitting increases the information by less than some threshold (Shalizi, 2009).

On the other size, Decision Trees are robust to outliers, to monotonic transformations of input features, can deal with missing values an interpretable model, which is a considerable advantage in many domains.

There are multiples algorithms implementing decision trees, such as ID3, C4.5, C5.0 which use Information Gain as splitting criterion and CART which uses Gini Index.

In this project we use Decision Trees as the estimator for Recursive Feature Elimination. The Decision Trees algorithm we use is CART as it is the implementation on Scikit-learn.

2.1.4 Random Forests (RF)

Random forests originated from an ensemble approaches to improve the generalization ability of Decision Trees. RF are a collection of Decision Trees, each one growing fully independent of the others. Concretely, RF randomly select a random subset of features (Ho, 1995) and a subset of instances (as inspired by Bagging (Breiman, Bagging predictors, 1996)) and train as many Decision Trees as user-specified. Each Decision Tree learns a model and outputs a value which works as a vote. Several techniques exist to convert the several votes into a final decision, such as the mode or the average.

Although the generalization ability of Random Forests depends on the generalization ability of its trees, as more trees are added to it, the generalization error is limited to an upper bound as shown by the Theorem 1.2 in (Breiman, Random Forests, 2001).

In this project we use Random Forests to model job matching.

3 Variable Selection

Variable selection enables one to identify the input variables which separate the groups well and the corresponding model frequently has a lower error rate than the model based on all the input variables (Louw & Steel, 2006).

3.1 Recursive Feature Elimination (RFE)

Recursive feature elimination is to select features by recursively considering smaller and smaller sets of features. (Guyon, Weston, Barnhill, & Vapnik, 2002). First, the estimator is trained on the initial set of features and the importance of each feature is obtained. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached. The ideal number of features is then defined when adding a new feature, the score of the model deteriorates (Milborrow, 2017).

In this project we use RFE with cross-validation to automatically select the ideal number of features based on Decision Trees for the estimation of feature importance.

4. Filtering

Filtering is a set of techniques applied to smoothing the data while without greatly distorting the underlying distribution of the data. The goal with smoothing is to remove noise and better expose the signal of the underlying causal processes. Smoothing is applied to avoid overfitting, while it may also happen that while classifying documents a word is encountered but has not been in the training set.

In this project we apply the Savitzky–Golay filter. This filter work by as a convolution on the data. Specifically, each data point is regressed against its n adjacent points with a linear least squares regression of polynomial low-degree (Schafer, 2011).

Savitzky–Golay filter has been successfully applied on data which was then modeled by means of Neural Networks (Oliveira, Araujo, Silva, Silva, & Epaarachchi, 2018), (Wettayaprasit, Laosen, & Chevakidagarn, 2007).

In this project, we experiment the Savitzky–Golay filter to control the domain of values of the features in order to help the learning algorithm to better generalize.

5. Cross-validation (CV)

Cross-validation is a statistical method used to evaluate and compare models and to estimate the generalization ability of a model. Given a set of observations, one would pick a considerable portion to train the model with, while the remaining part to assess the model. However, there is no reason to believe the train data is a good representative of the test data and vice-versa. If we consider for a moment that they don't, and we make a model design decision based one evaluation set alone, the utility of the model is very much questionable. Therefore, multiple subsets of data can be chosen to train and evaluate a central moment of the distribution of the evaluation score. The method works by splitting the training set into k folds. An iterative process is followed where k-1 folds of the data are used to train a model which is then evaluated against the left-out fold. The number of repetitions is equal to k. When the process terminates, central moments of the evaluation results are computed, commonly mean and standard deviation (Refaeilzadeh, Tang, & Liu, 2008).

6 Natural Language Processing (NLP)

NLP is a field of application of Artificial Intelligence concerning the interactions between computers and humans via natural languages. It includes challenges such as parsing (grammatical analysis) natural-language understanding or natural-language generation.

Classical approaches commonly represented natural language as a unique dimension in a sparse vector of inputs (one-hot encoding) and applied linear learning algorithms on top of it (Goldberg, A Primer on Neural Network Models for Natural Language Processing, 2016).

6.1 Word embeddings - Word2vec

Concerning to the natural language representation, more recent approaches create the so-called word embeddings, where each feature is embedded into d dimensional space and represented as a dense vector.

The principal benefit of embedding is representing words as not features per se but as contexts, which should entail a higher level of information not only about the word, the how and where the word was employed.

A common approach for word embedding is referred as sliding window of 2k + 1 size, where k is user defined (in some techniques the actual k per iteration is randomly selected between 1 and k). The sliding window iterates through the words, where the middle word in the window is called the focus and the neighbors words called the context (Goldberg, A Primer on Neural Network Models for Natural Language Processing, 2016).

The information extracted from each word is distributed all along a word window in distributed representations (word2vec representation). For a word vector learning, given a sequence of T words $\{w_1, w_2, \dots\ w_T\}$ and a window size c, the objective function is as follows:

$$\frac{1}{T}\sum_{i=c}^{T-c}\log p(w_i|w_{i-c}, \dots, w_{i+c})$$

In order to maximize the objective function, the probability of w_i is calculated based on the softmax function as follows:

$$p(w_i|w_{i-c}, \dots, w_{i+c}) = \frac{e^{y_{w_i}}}{\sum_{j\in(1,\dots,T)} e^{y_{wj}}}$$

where the word vectors are concatenated or averaged for predicting the next word in the content.

One of the advantages of using word2vec for building word embedding is that, while it is a Machine Learning approach, it does not require annotations. One of the approaches is to predict the focus word based on the context word, the so-called Continuous Bag of Words. Another way is predicting the context words, given for a given focus word, named as Skip-gram model Words. Embeddings are then training by means of Neural Networks (Mikolov, Chen, Corrado, & Dean, 2013).

In this project, we apply Skip-gram flavor of word2vec, to create a semantic vector to represent the job skills required and the applicant skills.

D. Data

We have approached an international IT-specialized online job marketplace, presented our research idea and were given a sample of their data, anonymized, so as to not include any fields which could be used to identify either the applicants and the recruiting companies. The job marketplace works as a middle man, posting job ads in the platform and finding the best set of applicants. The job marketplace professionals, are in charge of pre-screening job applicants to find the best set to progress in the recruitment process.

The data we were given includes 7 tables, specifically:

- People: candidates database, with and without applications;
- Applications: application from someone applying for a Job Offer. Applications have a strong relation with people and job ads;
- Application Audit: Processual information from the application. Revision dates, rejection dates, etc.

- Job Ads: job offers since on the platform;
- Companies: pretty obvious, they're related with Job Ads;
- Tags: generic tags imported from LinkedIn and user defined;
- Skills: the relation between Tags and People.

Relations:

- An application has an Application Audit, a Person and a Job Ad;
- A Job Ad has a Company;
- A person has many skills which may have many tags;

Another set of data consists of the Master Data Management. In this file we find the business mapping with the state field of various tables including, job offers states (eg. Published, not published, closed) or application states (eg. Unreviewed, reviewed, engaged, pre-offer, rejected, hired).

In this section we'll go through the baseline transformations applied to the tables to create features describing people, job ads and output.

The data in csv is ingested in Python with the library Pandas which has a central object called dataframe which resembles a SQL table and a large variety of methods for manipulating the data in the dataframe, e.g. casting, aggregations and merge.

The complete list of fields per table can be found in the appendix.

E. Methodology

No Free Lunch Theorem

The reasoning behind the application of several learning methods and the calibration of the corresponding parameters lays on the *No Free Lunch Theorem For Optimization*, which states that given a finite set V and a finite set S of real numbers, assuming that $f: V \rightarrow S$ is chosen at random according to uniform distribution on the set S^V of all possible functions from $V$and to S, then for the problem of optimizing f over the set V, there no algorithm performs better than blind search (Wolpert & Macready, 1997). In other words, if one algorithm outperforms another for certain kinds of cost function, then the contrary must be observed for all other cost function dynamics.

Therefore, one must try multiple learning algorithms and parameters to find one that works best for a particular problem. It has been established a few approaches to standardize the process of finding the best combination of algorithms and parameters.

Data modelling in practice

The two most popular approaches for data modellings are SEMMA (Sample, Explore, Modify, Model, Assess), introduced by SAS Institute and CRISP-DM (Cross-Industry Standard Process for Data Mining), introduced by SPSS and NCR (Azevedo & Santos, 2008). In the project, follow loosely the latter because of its flexibility and easy customization. CRISP-DM follows six stages:

1. Business Understanding: understanding the business goals and converting it into a Data Mining problem;

2. Data Understanding: explore the data, understand quality issues, get the initial insights and draw an initial set of hypotheses;
3. Data Preparation: transformations required to build the dataset for training, including but not limited to outliers' inspection, data normalization, data filtering and smoothing, data impute, dimensionality reduction and feature selection.
4. Modelling: Applying a set of learning methods and the calibration of the corresponding parameters;
5. Evaluation: Assessing the results against the hypotheses drawn and business goals;
6. Deployment: Integration of the final model into the organization systems, in a way that the final user can take advantage of it.

This process can be sequential but also iteratively repeated in any of the stages (IBM, 2011).

Software

This project was fully implemented in Python language under the Jupyter Notebook framework. Data transformations are carried out using Pandas library (McKinney, 2010)whereas Machine Learning algorithms and Statistics methods applied come from Scikit-learn (Pedregosa, Varoquaux, & Gramfort, 2011), Scipy and Numpy libraries (Oliphant, 2007). Word2vec is applied following the implementation on Gensim library (Rehurek & Sojka, 2010).

F. Data transformation

1. Building the target variable

The target we will be building concerns to the binary outcome of an application screening: selected or rejected.

By selected we mean the application was reviewed by the HR team and considered a match, so the application moves to a next stage, which typically is a technical problem, an interview with the HR team or an interview with the employer, each depending on the prior stage results.

On the other side, a rejected application is one that was reviewed by the HR team but considered not a match, so the process is interrupted.

The total raw number of applications as denoted by the number of rows in table 'Applications' is 52.548.

Looking at table 'Applications' we find a field named 'state' which represents numerically the stage at which the application is currently at. The business mapping can be found in the Master Data Management file as we show in figure 8 in Annex.

We define our target in the following way: the applications that were reviewed and passed to the client are positive cases and are identified in Table 8 by id's 25, 27, 28, 29, 30 and 97. On the other hand, the negatives cases are identified with state id 99.

Id’s 60 and 95 concern to applications that were reviewed by cancelled. Looking at state alone we can’t conclude whether this are positives, negatives or not able to label. Therefore, after merging ‘Applications’ table with ‘Application Audit’ table, we use the field ‘**reviewed_at’** based on which we define the following: applications in stage 60 or 95 which have a date on the field ‘reviewed_at’ are considered positive cases. We can’t conclude whether the nulls are a rejection by the HR-team or simply the applicant has cancelled her application before the process moved to the recruiter.

In the aftermath we get 72% negatives and 28% positives out of 37.900 useful applications.

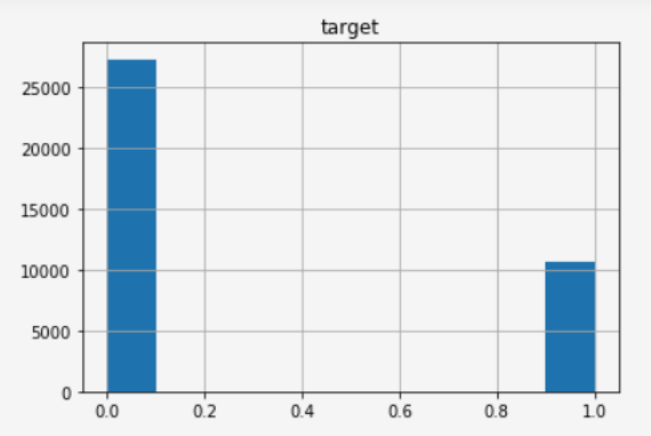


Figure 2 - Curator-defined target. On axis 0 we have the class and on axis 1 we have the number of observations

2.People features - create people dataset

People table is in large amount self-explanatory: each row represents a person in the database, and the various fields, such as ‘birth_year’, ‘relocation_countries’, or ‘salary_expectation’, represents the different dimensions of a person captured. Education and prior jobs are not asked when the person is building his profile in the Marketplace so these variables are not available in the database.

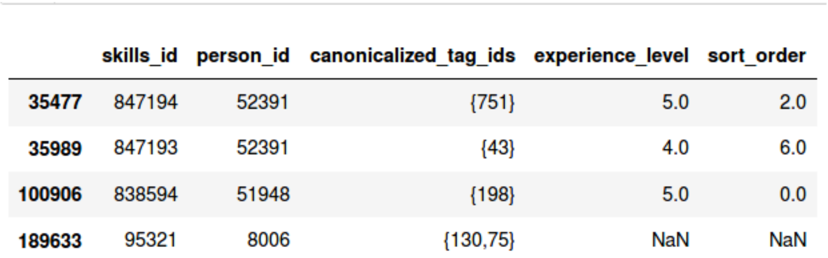
The person can attach the link to her Linkedin profile though, where she might or might not have declared prior job experience and education. For a matter of data privacy tough, the link to the Linkedin profile of the applicant was not provided in the data that we were given.

However, when the person builds her profile, she is asked to declare her skills in text boxes and the respective years of experience.

On the other side, job ads are built with the following sections: name of the position, description of the job, the city of the job or if it is a remote job, years of experience or level of seniority, required skills, nice-to-have skills (in some cases specifying the required experience per skill) and salary and perks of the job.

2.1. Extracting people skills and tags

Table ‘Skills’ holds the mapping between people and tags. A person has many skills and a skill can have many tags. Tags are text (skills name) and each one is unique in the database. However, the application has an engine which determines that when different tags possibly represent the same skill, then the skill points to a list of ‘tag_ids’ instead of a single ‘tag_id’. To the output of this engine is called ‘canonicalized skills’.



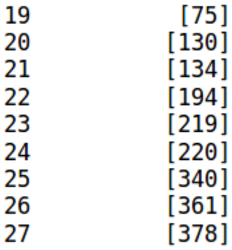
	skills_id	person_id	canonicalized_tag_ids	experience_level	sort_order
35477	847194	52391	{751}	5.0	2.0
35989	847193	52391	{43}	4.0	6.0
100906	838594	51948	{198}	5.0	0.0
189633	95321	8006	{130,75}	NaN	NaN

Table 1 - Skills table

For each applicant, we are interested in joining in the same table, skill-ids, skills-name(tags) and experience per skills.

Brief summary of the transformations applied:

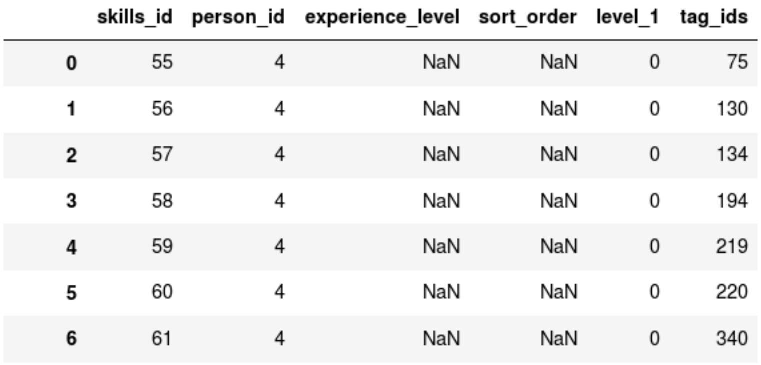
1) We are only interested in skills of people whose applications are part of out target dataset, so we start by filtering the ‘Skills’ table retaining only those of people in our target table.
2) Now we will work on the ‘canonicalized_tag_ids’ column. First we remove ‘{ }’ from all the rows. Then, we convert the string of tag ids into a list. String with multiple tag ids will return a list with multiple indices. Finally, we convert column data to integers.



```
19         [75]
20        [130]
21        [134]
22        [194]
23        [219]
24        [220]
25        [340]
26        [361]
27        [378]
```

Table 2 - canonicalized tags ids transformed

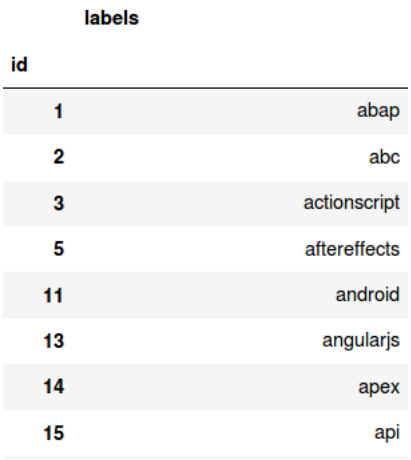
3) Now we breakdown the rows where skills have multiple tags into new rows and use a second level index to keep track of those rows whose skill ids are now duplicated. We have our skills dataframe ready.



	skills_id	person_id	experience_level	sort_order	level_1	tag_ids
0	55	4	NaN	NaN	0	75
1	56	4	NaN	NaN	0	130
2	57	4	NaN	NaN	0	134
3	58	4	NaN	NaN	0	194
4	59	4	NaN	NaN	0	219
5	60	4	NaN	NaN	0	220
6	61	4	NaN	NaN	0	340

Table 3 - Skills table after transformation on tag ids field

4) Import 'Tags' table.

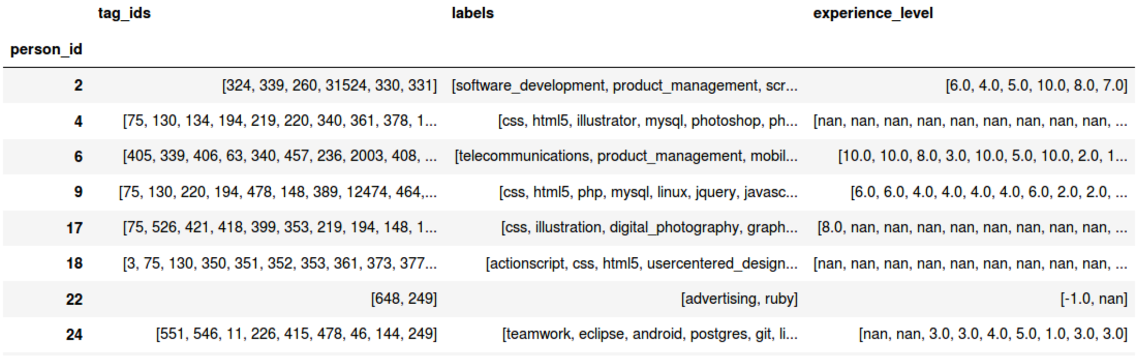


	labels
id	
1	abap
2	abc
3	actionscript
5	aftereffects
11	android
13	angularjs
14	apex
15	api

Table 4 - Tags table

5) We execute a left join between skills and tags dataframe, using 'tag_id' as key, resulting a new dataframe called 'skillTags'.

6) For last, we group 'skillTags dataframe by person_id concatenating in a single list all the values of a certain field related with the same person. The fields we are interested are 'tag_ids', 'label' and 'experience'.



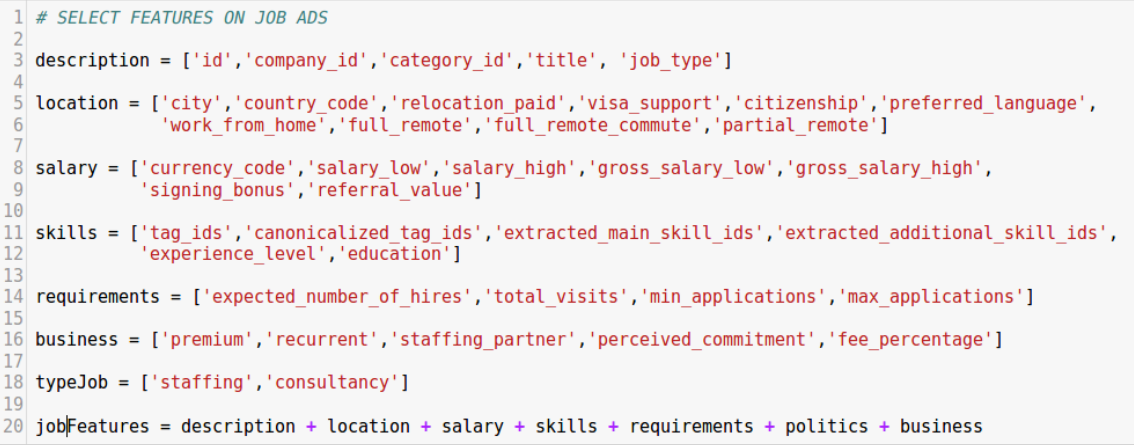
	tag_ids	labels	experience_level
person_id			
2	[324, 339, 260, 31524, 330, 331]	[software_development, product_management, scr...	[6.0, 4.0, 5.0, 10.0, 8.0, 7.0]
4	[75, 130, 134, 194, 219, 220, 340, 361, 378, 1...	[css, html5, illustrator, mysql, photoshop, ph...	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...
6	[405, 339, 406, 63, 340, 457, 236, 2003, 408, ...	[telecommunications, product_management, mobil...	[10.0, 10.0, 8.0, 3.0, 10.0, 5.0, 10.0, 2.0, 1...
9	[75, 130, 220, 194, 478, 148, 389, 12474, 464,...	[css, html5, php, mysql, linux, jquery, javasc...	[6.0, 6.0, 4.0, 4.0, 4.0, 4.0, 6.0, 2.0, 2.0, ...
17	[75, 526, 421, 418, 399, 353, 219, 194, 148, 1...	[css, illustration, digital_photography, graph...	[8.0, nan, nan, nan, nan, nan, nan, nan, nan, ...
18	[3, 75, 130, 350, 351, 352, 353, 361, 373, 377...	[actionscript, css, html5, usercentered_design...	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...
22	[648, 249]	[advertising, ruby]	[-1.0, nan]
24	[551, 546, 11, 226, 415, 478, 46, 144, 249]	[teamwork, eclipse, android, postgres, git, li...	[nan, nan, 3.0, 3.0, 4.0, 5.0, 1.0, 3.0, 3.0]

Table 5 - Skill tags per person

3. Job ad features – create job ads dataset

To build a dataset representing the attribute of a job ad, we need to pick attributes from tables 'JobAds', 'Companies' and 'Tags'. A job ad has a company while a company can have multiple job ads. A job ad requires multiple skills and each skill has a label.

Table "JobAds" has considerable number of fields. We get rid of all system columns as well as fields related with bureaucracy associated with the job ad.



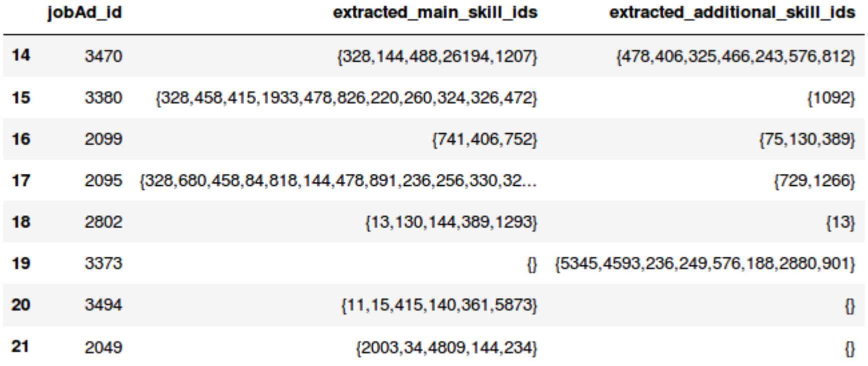
```
# SELECT FEATURES ON JOB ADS

description = ['id','company_id','category_id','title', 'job_type']

location = ['city','country_code','relocation_paid','visa_support','citizenship','preferred_language',
            'work_from_home','full_remote','full_remote_commute','partial_remote']

salary = ['currency_code','salary_low','salary_high','gross_salary_low','gross_salary_high',
          'signing_bonus','referral_value']

skills = ['tag_ids','canonicalized_tag_ids','extracted_main_skill_ids','extracted_additional_skill_ids',
          'experience_level','education']

requirements = ['expected_number_of_hires','total_visits','min_applications','max_applications']

business = ['premium','recurrent','staffing_partner','perceived_commitment','fee_percentage']

typeJob = ['staffing','consultancy']

jobFeatures = description + location + salary + skills + requirements + politics + business
```

Figure 3- Grouping job ads fields

From the remaining fields, we break them down into 7 groups, each one corresponding to a different perspective on the job ad based on our judgement. In any case, the purpose was merely to start having some hints on which feature could be used or built for the Machine Learning models that we will be later and to make the code more readable. Then, we merge jobAds table with Companies table.

3.1. Extracting required skills and nice-to-have skills

Job ads specify the required skills for the job and the additional or nice-to-have skills. These information is presented in the table ‘job ads’ in the fields ‘extracted_main_skill_ids’ and ‘extracted_additiona_skill_ids’. These fields come in the form showing in table 15. Please note that the ids of both of these fields point to Tag ids and not to Skills ids, contrary to what the name suggests.

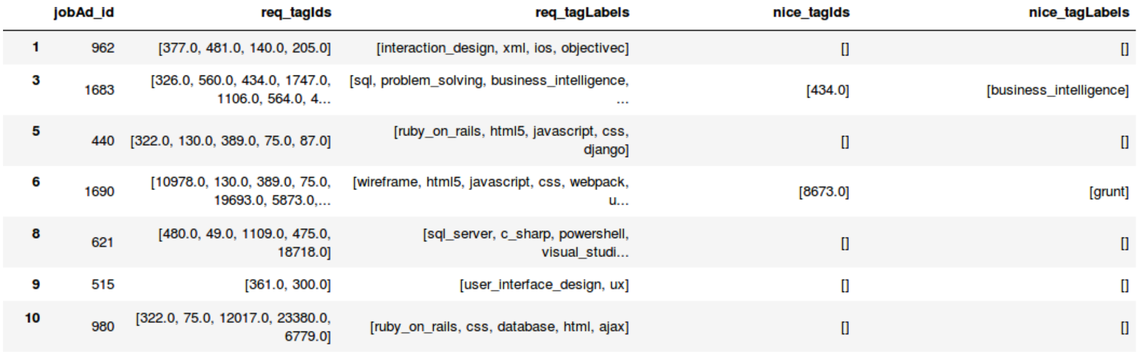


	jobAd_id	extracted_main_skill_ids	extracted_additional_skill_ids
14	3470	{328,144,488,26194,1207}	{478,406,325,466,243,576,812}
15	3380	{328,458,415,1933,478,826,220,260,324,326,472}	{1092}
16	2099	{741,406,752}	{75,130,389}
17	2095	{328,680,458,84,818,144,478,891,236,256,330,32...	{729,1266}
18	2802	{13,130,144,389,1293}	{13}
19	3373	{}	{5345,4593,236,249,576,188,2880,901}
20	3494	{11,15,415,140,361,5873}	{}
21	2049	{2003,34,4809,144,234}	{}

Table 6 - Job ad skills

Our goal is to obtain the list of required and nice-to-have skills per job ad and the corresponding labels. To do so, for both fields, we apply most of the same transformation that we briefed in section 2.1, which include:

1) Converting string of ids to list of integers;
2) Break down each list into the corresponding number of rows;
3) Merge with table Tags to get the labels.



	jobAd_id	req_tagIds	req_tagLabels	nice_tagIds	nice_tagLabels
1	962	[377.0, 481.0, 140.0, 205.0]	[interaction_design, xml, ios, objectivec]	[]	[]
3	1683	[326.0, 560.0, 434.0, 1747.0, 1106.0, 564.0, 4...	[sql, problem_solving, business_intelligence, ...	[434.0]	[business_intelligence]
5	440	[322.0, 130.0, 389.0, 75.0, 87.0]	[ruby_on_rails, html5, javascript, css, django]	[]	[]
6	1690	[10978.0, 130.0, 389.0, 75.0, 19693.0, 5873.0,...	[wireframe, html5, javascript, css, webpack, u...	[8673.0]	[grunt]
8	621	[480.0, 49.0, 1109.0, 475.0, 18718.0]	[sql_server, c_sharp, powershell, visual_studi...	[]	[]
9	515	[361.0, 300.0]	[user_interface_design, ux]	[]	[]
10	980	[322.0, 75.0, 12017.0, 23380.0, 6779.0]	[ruby_on_rails, css, database, html, ajax]	[]	[]

Table 7 - Transformed job ad skills

G. Experiments

We divide logically the experiments conducted into two groups. The techniques applied in both are highly intersected, however in experiment set A, we use a simple TF-IDF (Term Frequency-Inverse of Document Frequency)[1] approach to convert job and applicants’ skills into a term-

[1] Term-frequency matrix is a tabular representation of data where documents are usually assigned to the rows and the collection of terms through all documents are represented as columns. The values of the

document matrix and apply the cosine similarity on the applicant skills vector and job skills vector. In experiment set B, we test a set of different NLP techniques to further improve the ability to match the skills of the applicant with those required in the job ad. In experiment B, apart from the Skills-matching re-estimation, we rely on the same data-preprocessing techniques and features (except skills-matching) applied in experiment set A to generate the final dataset for training.

Experiment-set A

Feature engineering

In this section we introduce the features we generate to serve as input for the Machine Learning algorithms we apply later. The set of features we generate can be split into two groups: the matching group and the descriptive group. The matching group refers to the set of features generated in order to allow a comparison of similar dimensions between the job requirements and the candidate profile, for instance the matching between the skills required and the skills presented in the candidate's profile. The descriptive group are features for which either was not possible to draw a match or are purely one-sided features. One example of the former is where candidates present their experience years as a continuous variable, while the job postings state the required experience level as a category representing ranges of years of experience. For the latter case an example is whether Landing.Jobs considered the company offering the job position as strategic or not strategic.

1. Skills-matching

Let A be the set of all applications in our data D and A_n be the n^{th} application.

Let R be the set of all skills required in the job ad and O be the set of all skills offered by the applicants.

Skills are represented as a vector of strings encoded as a TF-IDF matrix.

Each application A_n contains a job ad j and a person/applicant p.

The skills-matching coefficient is the similarity between the vector of skills required R_j in the job ad j and the vector of skills offered O_p by the applicant p.

Let:

$$K = R \cup O$$

representing the union of all skills vector into a single set.

To encode the vectors of skills, we transform K into a TF-IDF sparse matrix $\overline{K}$. In text-mining terms, we implicitly consider each vector of skills a document and each skill a term.

The similarity measure used is the commonly applied cosine-similarity.

matrix represent the frequency each term appears per document. Usually it is multiplied by the inverse of the frequency of the term in the documents so as to give less important to terms which are very common and appear in most documents while stressing the importance of less common terms. In this case, the matrix is called Term-Frequency Inverse of Document Frequency (TF-IDF) matrix.

Therefore, the similarity S_n computed for each pair ($\overline{\overline{R_j}}$, $\overline{O_p}$) comes as:

$$S_n = \text{cosine}\,(\overline{\overline{R_j}}\,, \overline{O_p}) \ \in A_n, \quad \forall\, n = 1, 2 \dots N,$$

where N is the total number of applications in the data.

2. Profession-matching

Professional-matching represents how well the person's professional title, denoted in our data as the headline, matches with the name of the job role. The first impression a person conveys in her headline when compared with the name of the job role, might provide some insights on the degree of matching between a candidate and a job position.

We start by considering candidate's professional title and name of job role as strings and the first metric we extract is the distance between those string. For that we use Levenshtein distance metric which counts the number of character edits needed in one strings for it to become like the other one (Navarro, 2001). While Levenshtein distance metric is intuitive and easy to understand, we acknowledge some shortcomings such as ignoring the semantics of words. For instance, headline 'Experienced Software Developer' would match highly with job role 'Software Developer', however 'Front-end Developer' matches very poorly with 'UI Engineer, even though one can consider that the two strings are semantically connected.

Let's use the same notation as in the skills-matching, where, in this profession-matching context, R_j is name of job role presented on job ad j and O_p is the applicant p headline.

Therefore, the distance S_n computed for each pair $(R_j,\ O_p)$ comes as:

$$S_n = \text{levenshtein}\,(R_j, O_p) \ \in A_n, \quad \forall\, n = 1, 2 \dots N$$

The final match value is computed by dividing the distance S_n by the length of the longest of the two strings.

3. Location-matching

Location-matching intends to find the degree of matching between the applicant location or a location the applicant considers moving to and the job location. As an illustration for the pertinence of this feature consider that a candidate from a certain country applies for a job in another country for which a visa is required but the recruiter company is not sponsoring the visa application. As briefed by a Landing.Job official and through our domain knowledge, we suspect that these applicants would be in a less favorable situation and that this may be embedded in the selection criteria.

The following variables were defined before estimating the location-matching coefficient.

Let C represent the country-matching, then:

$$C = \begin{cases} 1 \;\; if\ the\ applicant\ is\ from\ the\ same\ country\ as\ the\ job \\ 0 \;\; otherwise \end{cases}$$

Let L represent the relocation-matching, then:

$$L = \begin{cases} 1 \;\; if\ job\ location\ is\ in\ the\ relocation\ countries\ list\ of\ the\ applicant \\ \quad 0 \;\; otherwise \end{cases}$$

Let V represent the visa-matching, then:

$$V = \begin{cases} 1 \quad if\ visa\ support\ and\ relocation\ paid, \\ 0.5 \quad if\ visa\ support\ or\ relocation\ paid, \\ 0 \quad if\ not\ visa\ support\ nor\ relocation\ paid \end{cases}$$

Let R represent the remote-matching, then:

$$R = \begin{cases} 1 \quad if\ full\ remote\ job, \\ 0.5 \quad if\ partial\ remote\ or\ WFH2\ or\ full\ remote\ commutable, \\ 0 \quad if\ not\ remote \end{cases}$$

We then compute the location-match coefficient by applying a weight vector to C, L, V and R

Let's represent C, L, V and R as a vector $\vec{l}$. We use our domain knowledge to create a vector of weights as

$$\vec{w} = (0.3, 0.3, 0.25, 0.15)$$

Recall from the previous notation that p be the applicant, j the job ad and $(p, j) \in A_n$, where A_n is the n^{th} application.

Hence, the location-match coefficient comes as:

$$LocMatch_n = \overrightarrow{l_n} \cdot \overrightarrow{w_n}$$

Other feature-matchings were considered, for instance, Salary-matching and Experience-matching. However, in the former case, most of the people don't specify their salary expectation

[2] Work-from-home job modality abbreviated to WFH.

nor is used a consistent scale used (e.g. units or thousands) and in the latter case, the experience required in the job ad is presented as classes with a non-homogenous scale (eg. Class 3 corresponds to 5 years of experience and class 4 corresponds to 7+ years of experience including leading).For the presented reasons, we decide to introduce salary and experience features in our feature set but we do not create a matching feature ourselves.

In the following, we present a set of descriptive features we generate to complement the aforementioned ones.

4. Additional features

Job experience level - Categorical

Job experience required in the job ad, converted to four dummy variables.

Applicant experience - Continuous

Years of experience of the applicant.

Strategic – Categorical / Boolean

Is a flag which encodes whether the company recruiting is strategic.

Premium – Categorical / Boolean

Is a flag which encodes whether the job ad is premium.

Perceived Commitment - Categorical

Represents the commitment of the recruiting company as perceived by the account manager. Three categories, converted to dummy variables.

Availability – Categorical

Encodes whether the applicant availability to start on the job, were she selected.

Offered salary – Continuous

Salary offered for the job. Is truncated with a lower bound of 10.000 and a higher bound of 150.000. Null values are imputed using the intra-class mean salary for classes country, role and experience.

Expected salary – Continuous

Salary expected by the applicant. Is truncated with a lower bound of 10.000 and a higher bound of 150.000. Null values are imputed using the intra-class mean salary for classes country and experience.

Number of applications per job – Continuous

Self-explanatory.

Number of expected hiring – Continuous

Number of people the company is recruiting for that job.

Total visits – Continuous

Number of times a job ad has been viewed.

Fee percentage – Continuous

Represents the price the recruiting company is paying to the Marketplace for their work.

Data preprocessing and feature selection

We begin our preprocessing phase by testing multicollinearity. Then we scale continuous features and applied a filtering algorithm.

For multicollinearity testing we used the VIF (variance inflation filter) algorithm as explained in Technical Background section. We suspected that our data could contain a high level of multicollinearity because whenever we transformed a categorical variable into dummy variables, we did not remove one of the dummies which should be considered the baseline dummy variable. In table 8 we can find the multicollinearity measurement scores where all features were considered and in table 9 we made available the multicollinearity measurement scores where we have removed a baseline class per categorical feature. The results are significantly different, with the latter set of features resulting in much lower scores. Though in the latter case some features show a VIF score over 20, we have considered the results acceptable, as there is not a consensus in the literature in regards to the acceptable values for the higher bound VIF score but these scores could be considered within an acceptable range (Dumancas & Ghalib, 2015).

Variable	VIF	type
jobExperLvl_1	5.035	categorical
jobExperLvl_2	605	categorical
jobExperLvl_3	25.146.274	categorical
jobExperLvl_4	1.925.703.577	categorical
is_strategic_job_0	34	categorical
is_strategic_job_1	21.609	categorical
premium_job_0	5.080	categorical
premium_job_1	346	categorical
perceived_commitment_jobAds_0	184.941.364.079	categorical
perceived_commitment_jobAds_1	32	categorical
perceived_commitment_jobAds_2	5.058.016.391	categorical
availability_ppl_0	13.350.010	categorical
availability_ppl_1	2.519.106	categorical
availability_ppl_2	379	categorical
availability_ppl_3	7.758	categorical
skillsMatch	2	interval
pplExperience	6	interval
professionMatch	7	interval
offeredSalary	7	interval
expectedSalary	7	interval

locationMatch	4	interval
nAppsPerJob	4	interval
expectedNumberOfHires	2	interval
totalVisits	5	interval
feePercentage	9	interval

Table 8 - VIF scores per feature before removing baseline classes in categorical variables

Variables	VIF	Type
'jobExperLvl_2	2	categorical
jobExperLvl_3	7	categorical
jobExperLvl_4	8	categorical
is_strategic_job_1	10	categorical
premium_job_1	8	categorical
perceived_commitment_jobAds_1	4	categorical
perceived_commitment_jobAds_2	4	categorical
availability_ppl_1	2	categorical
availability_ppl_2	6	categorical
availability_ppl_3	24	categorical
skillsMatch	9	interval
pplExperience	3	interval
professionMatch	2	interval
offeredSalary	2	interval
expectedSalary	5	interval
locationMatch	54	interval
nAppsPerJob	3	interval
expectedNumberOfHires	8	interval
totalVisits	10	interval
feePercentage	3	interval

Table 9 - VIF scores per feature after removing baseline classes in categorical variables

Moving on to feature scaling, it is particularly beneficial when the domain range among variables differ substantially, as is the case here since we have features such as skills-matching which take values between zero and one and other feature such as number of job posting views which can take virtually any positive natural number. Feature scaling results in faster convergence during training and there are some algorithms which give substantially better results with scaling (Hall, 1999). Continuous features are scaled by means of z-score3.

Regarding data filtering we have applied Savitzky–Golay filter to smooth the continuous variables in our feature-set. As indicated in the Technical Background section, Savitzky–Golay filter has been successfully applied in Machine Learning problem to reduce that noise in the data. Through visual inspection using a boxplot chart (Figure 4) we noticed that the distribution

[3] Z-score is computed by subtracting for each variable observation the sample mean and divided by the standard deviation of the sample (Kreyszig , 1979).

of our features presented notable differences. Savitzky–Golay filter has two main parameters a) window size *W* and b) polynomial degree *D*. Usually, the *D* is much smaller than *W*, so as to obtain a reasonable smoothing. Conversely, the larger the *W*, the smoother the data. The estimation of parameters *W* and *D* was made by visual inspection, comparing the box-plot of each feature prior to and after smoothing for each combination of parameters. We have chosen a window size of 11 and a polynomial degree of 2 as it provided an acceptable balance across all continuous features. Boxplot after applying filter can be found in Figure 5.

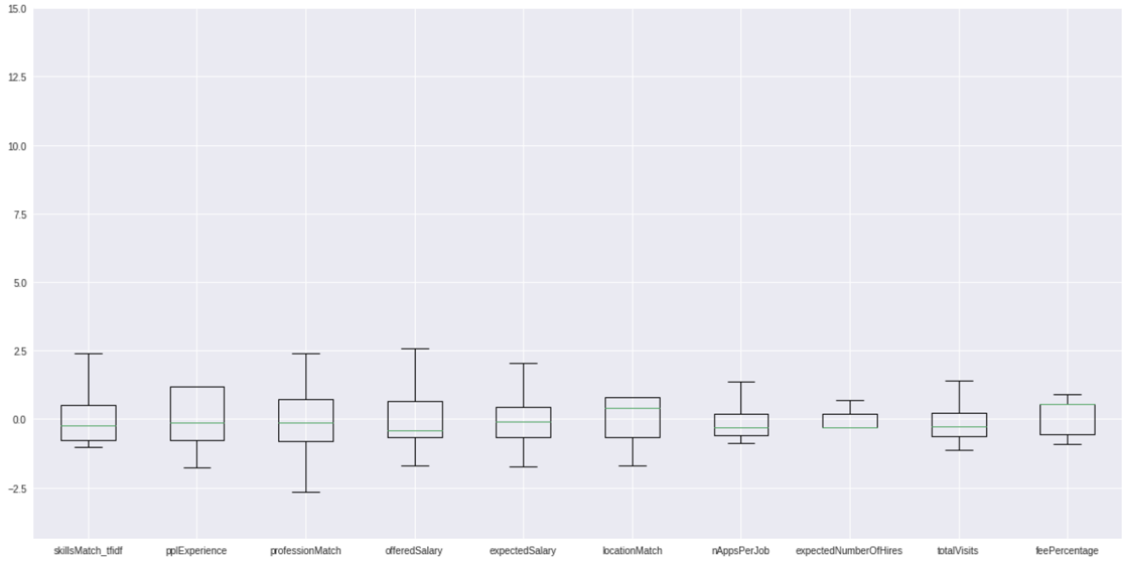


Figure 4- Boxplot chart representing variables prior applying Savitzky–Golay filter

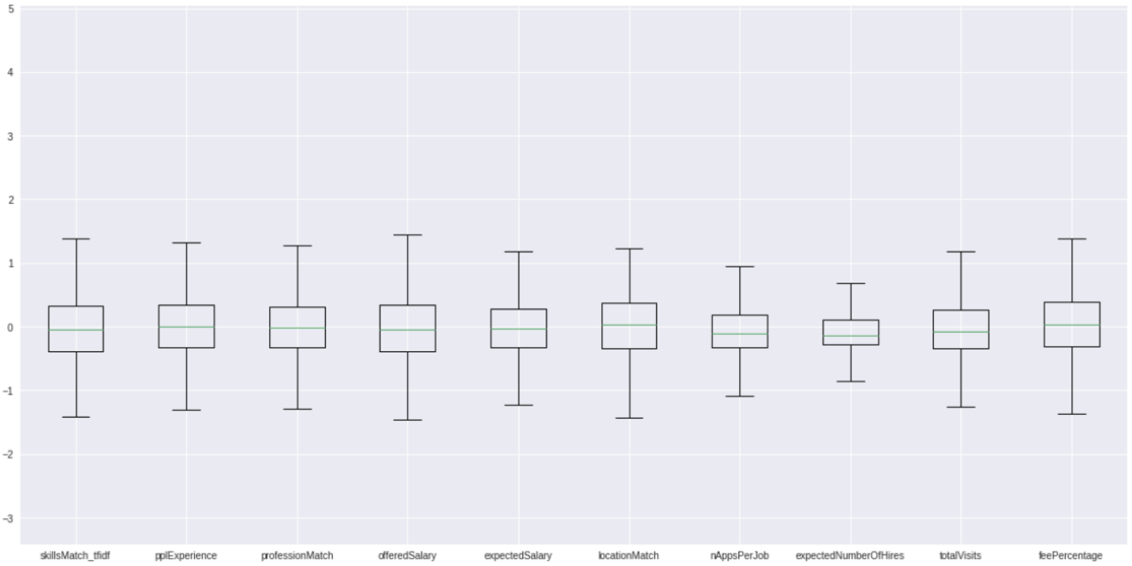


Figure 5 - Boxplot chart representing variables after applying Savitzky–Golay filter

Feature selection

For feature selection we have applied Recursive Feature Eliminative algorithm in conjunction with Cross Validation with 30 folds (RFE-CV). This algorithm intends to find the optimal number of features in regard to an evaluation metric. Please consider the Technical Background section for details about the algorithm. The evaluation metric we use is the Receiver operating characteristic – Area Under the Curve (ROC-AUC). In the Results Discussion section the reader can find a discussion for the usage of this metric. To understand the impact of data filtering we have applied RFE-CV in the data both prior applying Savitzky–Golay filter and after applying it. In both cases, the ideal number of features was twelve, however when applied to the data without the filtering algorithm the result was 0.60 while when applied to the data with the filtering algorithm the result was 0.62. Based on these results, we decided to maintain the filtering algorithm.

In Table 10 the reader can find the set of features selected by the RFE-CV algorithm.

Feature names
Skills-matching
Professional-matching
Location-matching
Job experience level_2
Job experience level_3
Job experience level_4
Applicant experience
Offered salary
Expected salary
Number of expected hiring
Total visits
Number of applications per job
Strategic

Table 10 - Final set of features

Modelling

We have modelled the preprocessed data with three learning algorithms: Logistic Regression, Multilayer Perceptron and Random Forests.

Each learning algorithm has its own set of parameters that we need to tune. We have specified a grid of possibilities for each parameter and each algorithm.

The model is learnt with cross-validation with 30 folds.

Logistic Regression

Regarding Logistic Regression we tune the regularization parameter C. Lower values of C represent a stronger regularization. Regularization is a technique in which the prediction error of a Machine Learning model is artificially inflated to help the algorithm to learn the right set of parameters that not only fit the training data but most importantly that better generalizes for unseen data.

Regularization parameter testing values are: 0.003, 0.01, 0.03, 0.1, 0.3 and 1.

The most performant parameter value for each measure is provided in the results section.

Multilayer Perceptron

The parameters we tune are:

- Neurons per layer: 3 and 5
- Initial learning rate: 0.01, 0.05, 0.1
- Learning rate strategy: “constant” and “invscaling”
- Momentum: 0.01, 0.02 and 0.03
- Optimizer: ‘SGD’ , ‘Adam’ and ‘LBFGS’
- Epochs: 50, 100, 200 and 400

When choosing the learning rate strategy “constant”, the initial learning rate is used during the whole training. Then set to “invscaling”, the learning rate is update at each epoch. The effective learning rate is then:

$LRe = {}^{1}/_{\sqrt{t}}$, where t the training epoch.

‘SGD’ stands for ‘Stochastic Gradient Descent’, a popular optimization algorithm. ‘Adam’ a is relatively new algorithm. It is an optimizer of first-order gradient based on adaptive estimates of lower-order moments, which is computational and memory-efficient, making then fast and well-suited for problems with large amounts of data (Kingma & Ba, 2015).

Momentum is used to avoid the algorithm getting stuck in a local minimum. Momentum is a term which is multiplied by the delta of the previous epoch and is summed to the current delta, providing an extra push to leave a potential local minimum.

The number of hidden layers is 1 (Hornik, 1989) and the activation function we use is the Logistic. The training has another parameter controlling the interruption of the training called ‘early stopping’, which is set to stop training when the model evaluated on the validation test worsens score to avoid overfitting. Also, the parameter controlling the regularization (L2) is set to 0.001.

Apart from momentum and ‘invscaling’ which are only combined with ‘SGD’, the remaining parameters are combined as a cartesian product.

Random Forests

The parameters we tune are:

- Number of estimators: 10, 30 and 100
- Minimum samples on the trees leaves: 1, 20 and 50
- Maximum number of features: All and the square root of the number of features
- Maximum tree depth: No limit and 5

Defining a minimum number of leaves in each tree and/or controlling the depth of the trees are ways to control overfitting, avoiding the model to retain all the small details of the training data, which may not be representative of true feature distribution. Splitting criterion is set to the default Gini Index.

The top results for each metric is presented in results section and the remaining in the appendix.

Results: experiment-set A

Results are provided in terms of learning algorithm with a specific set of parameters. The metric we use for assessing the quality of our models is the AUROC curve (Area Under the Receiver Operating Characteristic curve) (Cortes & Mohri, 2003).

The best results were found with the following parameters:

Logistic Regression

- Regularization parameter: 0.01

Multilayer Perceptron

- Neurons per layer: 3
- Initial learning rate: 0.01
- Learning rate strategy: constant
- Momentum: 0.03
- Optimizer: LBFGS
- Epochs: 200

Random Forests

- Number of estimators: 100
- Minimum samples on the trees leaves: 50
- Maximum number of features: Square root of the number of features
- Maximum tree depth: 5

Model	Algorithm	ROC-AUC Sample Mean	ROC-AUC Sample Std. Dev.	95% CI
LR_{TF-IDF}	LR	0,797	0,048	[0,780 – 0,814]
RF_{TF-IDF}	RF	0,805	0,051	[0,787 – 0,823]
MLP_{TF-IDF}	MLP	0,814	0,031	[0,803 – 0,825]

Table 11 - Best results per Machine Learning algorithm using the TF-IDF encoding of skills

In terms of sample mean, the best performing algorithm is the Multi-layer Perceptron (MLP) which reaches 0,814 in terms of ROC-AUC. However, for the three algorithms explored, the 95% confidence interval reveals an overlap in the results, which means that no algorithm performs statistically better than the others. Even after considering that, we decide to accept the MLP as the baseline model to compare with the next set of experiments. The result of the Logistic Regression are the poorer ones, for reasons we suspect are related with the problem we are solving not being linear. However, it is often a good idea in practice to start with a simple model such as Logistic Regression and use it as baseline for further refinements. With regard to Random Forests, the algorithm performed better than the Logistic Regression and worse than the Multi-layer Perceptron. Eventually, a better result could be found by increasing the number of estimators or by increasing the tree depth, however this would impact training times which should remain in a reasonable time frame, considering that when deployed to production, the model is going to be updated often with new data.

Experiment-set B

Feature engineering – semantic embedding

In this section we present two approaches to improve the skills-matching feature: LSA and Word2vec. The remaining variables are left untouched.

Latent Semantic Analysis (LSA)

Consider $\overline{K}$ the union of R (the vectors of required skills) and O (the vectors of applicant skills) encoded as term-frequency matrix. In some preliminary work we have found that applying LSA to the TF matrix yielded better results than when we applied LSA to the TF-IDF matrix.

When applying LSA, we choose 300 latent variables as to include 80% of the explained variance in the new set of features. We denote the resulting LSA matrix as L.

The new skills-matching coefficient is the similarity between the LSA-transformed vector of skills required $L(R_j)$ in the job ad j and the vector of skills offered $L(O_p)$ by the applicant p.

The similarity S_n^L computed for each pair $\overline{\mathrm{L}(\overline{\overline{R}}_j)\ ,}\ \mathrm{L}\left(\overline{O}_p\right)$ comes as:

$$S_n^L = \text{ cosine } (\mathrm{L}(\overline{\overline{R_j})}\,, \mathrm{L}(\overline{O}_p\,)) \ \in A_n, \quad \forall\, n = 1, 2 \dots N$$

Word2vec

Concerning to word2vec skills embedding, we model K with the Skip-gram flavor of word2vec, denoted as W. Specifically:

$W_{ji} = word2vec(R_{ji}),$

Where W_{ji} is the semantic encoding of the skill i of job j and R_{ji} is the skill i of job j, as a string.

Likewise,

$W_{pk} = word2vec(O_{pk}),$

Where W_{pk} is the semantic encoding of the skill k of applicant p and O_{pi} is the skill k of applicant p.

The similarity S_{nik}^{W} computed for each pair W_{ji} , W_{pk} is the cartesian product of the cosine similarity among each pair W_{ji} and W_{pk} belonging to the same application A_n.

This results in a $i * k$ vector of similarities per An.

To aggregate these similarities into a single coefficient, we compute the mean of S_n^W , denoting it as $\bar{S}_n^W$.

Two important parameters of Word2vec are the minimum count of word frequency and the window size. We set the minimum count as 5 and use experimentally a window size of 2 and 3, where we denote the similarity with a window size of 2 as $\bar{S}_n^{W2}$ and the similarity with a window size of 3 as $\bar{S}_n^{W3}$.

Results – Experiment-set B

Following the semantic embedding as outlined above, we build three new input datasets, similar to the one we used in experiment-set A, except that we change the original skills-matching feature. In the first e of the new input datasets, the skills-matching feature we use is the one following the application of LSA. In the second one, the skills-matching feature is the result of the application of word2vec algorithm with the window size of two and in the third one we test setting the window size to three.

We finally apply the best performing algorithm found in experiment-set A in each of the new datasets.

The results are displayed in Table 12.

Model	Skills semantics	ROC-AUC Sample Mean	ROC-AUC Sample Std. Dev.	95% CI
MLP_{LSA-TF}	S_n^L	0.807	0.032	[0.796 – 0.818]
MLP_{w2v2}	$\bar{S}_n^{W2}$	0.822	0.049	[0.804 – 0.840]
MLP_{w2v3}	$\bar{S}_n^{W3}$	0.838	0.048	[0.821 – 0.855]

Table 12 - Results in terms of ROC-AUC after applying the MLP fine-tuned where skills are represented as semantic vectors

Discussion of results

Comparing the results of models prior and after semantic embedding we notice a considerable improvement with respect with the target metric, the ROC-AUC. Even though we can observe an overlap to some extent of the confidence intervals between the first set of experiments and the second one where skills where semantically embedded, the sample mean score yields a consistently better score after embedding.

With regard to the semantic embedding techniques, the word2vec algorithm provided better results than the LSA technique. Actually, when setting the window parameter of word2vec to three, the difference in the results is statistically significant considering an alpha of 5%. Perhaps the result of the experiment using LSA could be optimized by increasing the percentage of explained variance, which would affect the length of the skills vector, however, the same type of argument could be applied to the word2vec where we did not fine tune neither the number of iterations of the algorithm nor the minimum frequency of words which are both hyper-parameters of the algorithm. Therefore, our final model is the one where we applied the Multi-layer Perceptron with 3 neurons in a single hidden layer, a constant learning rate of 0.01, a momentum of 0.03, optimizer LBFGS and 200 epochs of training and where we used the word2vec algorithm for skills embedding with minimum word frequency of 5 and a window size of 3.

All of the results are presented in terms ROC-AUC. ROC curves come from the field of signal processing and are used, for instance, in medicine to evaluate the validity of the diagnostic tests (Ferri, Flach, & Hernández-Orallo, 2002). The ROC curves show the rate of true-positives in the y-axis against the false-positives rate in the x-axis (Fawcett, 2005). The AUC represents the area that is below the ROC curve and can be interpreted as the effectiveness of a measurement of interest. The results of our final model are given in probability terms, i.e., for each application, the model returns the probability of the application being successful. The case of a perfect ROC curve corresponds to obtaining all successful applications at the beginning of the list of results and all unsuccessful applications at the end. This situation corresponds to an AUC equal to one. The contrary situation corresponds to a random system where the progress in the rate of successful applications is accompanied by an equivalent degradation in the rate of unsuccessful application and corresponds to an AUC equal to 0.5 (Fawcett, 2005).

Therefore, given our results, in light of the ROC-AUC metric we can read our best results (0.838) as the average successful application has approximately 16% probability of having an unsuccessful application scoring higher than it.

When setting ROC-AUC as our evaluation metric we achieve two goals: 1) maximizing the number of job applications correctly selected and 2) minimizing the number of incorrectly selected ones, which happens to be just as important as 1). Also ROC curves are resistant to imbalance (Fluss , Reiser, Faraggi, & Rotnitzky, 2009) which is relevant for our case since we have roughly three times more negative examples than positive ones.

The models we have built in our experiments correspond to the mapping between the input data (skills-matching, profession-matching, etc.) and the output data which encodes whether an application was selected in the screening phase of a recruitment process. However, we claim that our models could be deployed for purposes of job recommendation as well. In practice the difference between using the model for screening or for job recommendation is that in the former case there is an actual job application while in the latter case there is not an actual application but the potential candidate is already sitting in the database. Thus, one could ask the following question: what is the probability of person A to be a good match for job X, were person A willing to apply to it? By generalizing this question to every potential candidate in the database and every new job position, we end up with a list of top matches for each job position, therefore recommending the jobs of highest matching probability to the corresponding people. Some advantages of this approach are that instead of multiple models to maintain in production, the recruiter would only need to maintain ones which decrease maintenance effort while any improvements in the current model would benefit both the recommendation and the screening processes. Another advantage of using a single model is that the jobs that the recruiter recommends to a person are likely to be a good match with her. Ideally a recruiter should recommend a job to a person that is likely to be interested in it and also has a profile that gives a good level of confidence that if the person were actually to apply, there would be a good fit. By using features such as skills matching, location matching or salary offered and demanded as we do in our model, it gives us a fair expectation that a job recommendation to a person which yields a high probability level in the screening it would peak the attention of the recommendation recipient. On the disadvantages side, since our approach can be considered to some extent a content-based approach, people with uncomplete profiles in the job portal, possibly applying to jobs using their own CV, are likely to not be recommended to many jobs. To tackle this disadvantage the recruiter could set the matching probability to a lower threshold when recommending jobs while setting it to a higher threshold when screening applications.

H - Future work

Preliminary work

Our research proposal for future work is based on the premise that people who apply for the same kind of jobs share some attributes. In the data we use in this work we find peoples' profiles, including their skill set which is presented in natural language. For instance, we have applicant A who writes that she knows about HTML, CSS and Javascript and applicant B writes he knows about front-end development. From what we know about the world, these two people are likely to have many similarities in their technical abilities but if we were to represent their skill-set with algorithms such as TF-IDF and compute their similarity they are likely to be quite distant.

Also, if we try to match the skill set of each candidate against a job posting which requires knowledge of HTML and CSS, candidate A would come up with much higher similarity than candidate B, even though they might be both as good. Therefore, our research idea is to virtually extend peoples' skill-set by adding the top n mentioned skills that the k most similar people have.

In our preliminary work, which has been conducted in order to guide this discussion of future work, we start by identifying the top five most similar people with the target person in terms of the co-occurrence of applications for the same jobs. After that, we find the top three skills most frequent in the skill-set of the most similar people. Finally, we increment the skill-set of the target person by adding up any of these top three skills whenever they are not already presented in her skill-set.

Likewise, if a pool of candidates applies for similar positions multiple times, the job positions themselves are likely to share attributes amongst them. Extrapolating our first idea to the job side, we intended to extend the required set of skills by adding up the top n required skills that the k most similar jobs have. For that, we first found the top five most similar jobs with the target job in terms of the co-occurrence of application and we increment the required skills-set for the job position adding up the top three most frequent required skills whenever they are not already requested in the job profile.

Upon the extension of applicants' skill-set and job required skills, a new skill matching was computed by applying cosine similarity measure on the vectors of the extended applicant skill-set and job's extended required skills and finally we trained a Neural Network with the best performing set of parameters on the whole set of features, replacing the former skills-matching by the new skill-matching extended.

Preliminary results show an improvement on the results. We suspect the gain comes from tacking the sparsity in skills presented by applicant and required for the job position.

We consider that results could be improved by fine-tuning the parameters n and k, the number of most similar people/jobs and the number of considered top skills, respectively. Moreover, given the ability of the word2vec algorithm to predict a word in a context of a window of surrounding words, we contemplate the possibility to train a Neural Network using the word2vec algorithm using as input the skill-set of the most similar people and predict the next k words/skills, which would apply for extending job's required skills in the same fashion.

One critique of the aforementioned idea is that by extending people's skill-set and job required skills, the similarity degree skill-wise will grow more strongly for people with less declared skills and that could reduce that discriminant ability of the skill-matching feature. While that is potentially a problem not yet explored, we also consider that the set of skills which were artificially incremented on people's profile could be recommended to people when writing or updating their profile or to serve as a recommendation for continued learning, as people applying for the same jobs tend to have them.

Other future work

Moreover, we consider that our model could be improved by using semantic embedding not only for skills but also for other aspects such candidate summary, job description, company summary, etc. The semantic representation of these aspects could then be used as additional features to improve the model.

Finally, the target variable we used in our model was derived from the pass or fail decision in the screening process. Application screening by HR experts is conducted in a manner to optimize the subset of applicants which offer the greatest probability of hiring. This selected group of applicants then proceed to additional elimination phases such as written tests and interviews. Another path for improving results could be training a model based on the outcome of the whole recruitment process, i.e. hired or anything else, rather than the outcome of the screening phase, as hiring is the true desired outcome.

I – Conclusion

A challenge faced in the era of Internet is the huge volume of information available. Job market is no exception. On one side, there a tremendous number of job postings published online everyday which makes the task of the candidate to short-listing jobs a time-consuming and tedious one. On the other side, many job postings receive hundreds or thousands of applications which make the recruiter task of screening application a very challenging one given the businesses deadlines. In this work, we tried tackled these problems by means of Machine Learning.

The raw data we use in this work is the transactional database data of the job portal Landing.Jobs. It consists of multiple tables which store information such as people's profiles (skills, experience, location…) jobs' profiles, applications and the result of applications.

Based on the raw data we build a set of features based on literature and domain knowledge and we apply the Recursive Feature Elimination algorithm to arrive to the final set of features which includes a) skills-matching (the cosine similarity between the vector of required skills for the job and the vector of skills possessed by the applicant), b) professional-matching (similarity between the job title and the professional title of the applicant), c) location-matching (measure of geographic and bureaucratic matching in terms of the location of the applicant and the location of the job position), d) required experience, e) applicant experience, f) the salary offered for the job, g) the required salary by the applicant, h) the number of expected hiring for the job position, i) the total number of times the job posting was viewed, j) the number of applicants for the job and k) whether the respective company is seen as strategic or not for the job portal.

In our first set of experiments, we model the data using three algorithms – Logistic Regression, Random Forests and Multilayer Perceptron and we calibrate the most important hyper-parameters of each of the algorithms by iterating through a set of potential values (grid-search). The model that provided the best results in the first set of experiments is the one where we modelled the data using MLP with the following hyper-parameters: single hidden layer, 3 neurons, constant learning rate of 0.01, momentum of 0.03, LBFGS optimizer and 200 epochs of training.

In the second set of experiments, we apply the MLP algorithm with the best hyper-parameter values found in the first experiment and we use all features that we have used to model it, except for skills-matching. Concerning skills-matching, we have adopted a semantic embedding approach using two different techniques. One was Latent Semantic Analysis with 300 latent features and the other was Word2Vec algorithm with window size of value 2 or 3 depending on the experiment.

The best results were found with the application of the Word2Vec algorithm with window parameter set to three. Our results are expressed in terms of ROC-AUC which yielded a value of 0.838 that can be read as the average successful application has around 16% probability of having an unsuccessful application scoring higher than it. This metric is relevant in the context of job recruitment where one wants to maximize the number of job applications correctly selected and minimize the number of incorrectly selected ones, which is what ROC-AUC optimizes for.

We further consider that our model can be deployed in production for purposes of automatically screening applications and recommending jobs to candidates. While the former application of our model is a natural fit because after all our Machine Learning model is learning what is a successful application, for the latter application, i.e., recommending jobs to candidates, we consider that there are points in its favor. For one, if we apply a screening model for recommendation purposes, we are likely to be recommending jobs to people which have a good probability of being selected in the screening phase, which can be positive for managing candidates’ expectations and improving the overall recruitment process. Another reason is a maintenance one, since maintaining a single model is easier and cheaper and any improvements on the model would result in benefits for both recommendation and screening tasks.

The work we presented in the paper has the potential to give greater scalability power to the business of Landing.Jobs by automatizing (at least partially) application screening while being an additional option for building recommendation by complementing the current job recommendation results with the ones which our model can provide.

J - Bibliography

Allison, P. (1999). What can be done about multicollinearity. In P. Allison, *Multiple Regression: A Primer.* (pp. 137-150). Thousand Oaks: Pine Forge Press.

Alotaibi, S. (2012). A survey of job recommender systems. *International journal of physical sciences*.

Azevedo, A., & Santos, M. (2008). KDD, semma and CRISP-DM: A parallel overview. *IADIS European Conference on Data Mining 2008* (pp. 182-185). Amsterdam: IADIS.

Berge, J. M., & Kiers, H. A. (1999). Retrieving the correlation matrix from a truncated PCA solution: The inverse principal component problem. *Psychometrika*, 317–324.

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, 109-132.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 123–140.

Breiman, L. (2001). Random Forests. *Machine Learning*, 5-32.

Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and Regression Trees.* Taylor & Francis.

Cortes, C., & Mohri, M. (2003). AUC Optimization vs. Error Rate Minimization. *Proceedings of the 16th International Conference on Neural Information Processing Systems* (pp. 313-320). Whistler: MIT Press Cambridge.

Desai, V., Bahl, D., Vibhandik, S., & Fatma, I. (2017). Implementation of an Automated Job Recommendation System Based on Candidate Profiles. *International Research Journal of Engineering and Technology*.

Diaby, M. (2015). *Methods for Job Recommandation on Social Networks. Social and Information Networks* . Paris: Université Sorbonne Paris Cité.

Dormann, C. F. (2013). Collinearity: a review of methods to deal with it and a simulation study evaluation their performance. *Dormann, C. F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carré, G., Marquéz, J. R. G., Gruber, B., Lafourcade, B., Leitão, P. J., Münkemüller, T., McClean, C., Osborne, P. E., Reineking, B., Schröder, B., Skidmore, A. K., Zurell, D. and Lautenbach,*, 27.46.

Dreiseitl, S., & Ohno-Machado, L. (2002). Logistic regression and artificial neural network classification. *Journal of Biomedical Informatics*, 352–359.

Dumancas, G., & Ghalib, B. (2015). Comparison of Machine-Learning Techniques for Handling Multicollinearity in Big Data Analytics and High Performance Data Mining. *Supercomputing 2015: The International Conference for High Performance Computing, Networking, Storage, and Analysis.* Austin.

Faliagka, E., Ramantas, K., Tsakalidis, A., & Tzimas , G. (2012). Application of Machine Learning Algorithms to an online Recruitment System. *ICIW 2012 - The Seventh International Conference on Internet and Web Applications and Services.* Stuttgart.

Fang, M. (2015). *Learning To Rank Candidates For Job Offers Using Field Relevance Models.*

Färber, F., Weitzel, T., & Keim, T. (2003). An Automated Recommendation Approach to Selection in Personnel Recruitment. *9th Americas Conference on Information Systems.* Tampa, Florida.

Fawcett, T. (2005). An introduction to ROC analysis. *Elsevier*, 861-874.

Fazel-Zarandi, M., & Fox, M. (2009). Semantic Matchmaking for Job Recruitment : An Ontology-Based Hybrid Approach.

Ferri, C., Flach, P., & Hernández-Orallo, J. (2002). Learning Decision Trees Using the Area Under the ROC Curve. *The Nineteenth International Conference on Machine Learning.* Sidney.

Fluss , R., Reiser, B., Faraggi, D., & Rotnitzky, A. (2009). Estimation of the ROC curve under verification bias. *Biometrical journal*.

Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research*, 345–354.

Guo, S. (2015). *RésuMatcher: A Personalized Résumé-Job Matching System.* Texas: Texas A&M University.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 389–422.

Ho, T. (1995). Random decision forests. *Proceedings of the Third International Conference on Document Analysis and Recognition*, (p. 278). Washington, DC.

Hong, W., Zheng, S., & Wang, H. (2013). A Job Recommender System Based on User Clustering. *JOURNAL OF COMPUTERS*, 1960-1967.

Hornik, K. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 359-366.

Hunt, E., Martin, J., & Stone, P. (1966). *Experiments in Induction.* New York, Academic Press.

IBM. (2011). *IBM SPSS Modeler CRISP-DM Guide.* IBM Corporation.

Johnson, R., & Wichern, D. (2007). Principal Components. In R. Johnson, & D. Wichern, *Applied Multivariate Statistical Analysis* (pp. 430-465). New Jearsey: Pearson Education Inc.

Kingma, D., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *ICLR2015.* San Diego.

Kotsiantis, S. (2007). Supervised Machine Learning: A Review of Classification. *Informatica*, 249-268.

Lazarsfeld, P., & Henry, N. (1968). *Latent structure analysis. Boston: Houghton Mifflin.*

Louw, N., & Steel, S. (2006, December 1). Variable selection in kernel Fisher discriminant analysis by means of recursive feature elimination. *Computational Statistics & Data Analysis*, pp. 2043-2055.

Lu, Y., El Helou, S., & Gillet, D. (2013). A recommender system for job seeking and recruiting website. *22nd International Conference on World Wide Web* (pp. 963-966). International World Wide Web Conferences Steering Committee.

Malinowski, J., Keim, T., Wendt, O., & Weitzel, T. (2006). Matching People and Jobs: A Bilateral Recommendation Approach. *Proceedings of the 39th Annual Hawaii International Conference on System Sciences* . Kauia, HI, USA, USA: IEEE.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, (pp. 51 - 56).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at ICLR.*

Milborrow, S. (2017, December 13). *Notes on the earth package.* Retrieved from milbo.org: http://www.milbo.org/doc/earth-notes.pdf

Murthy, S. (1998, December 2). Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, pp. 345–389.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, 31-88.

Oliphant, T. (2007). Python for Scientific Computing. *Computing in Science & Engineering*.

Oliveira, M., Araujo, N., Silva, R., Silva, T., & Epaarachchi, J. (2018). Use of Savitzky-Golay Filter for Performances Improvement of SHM Systems Based on Neural Networks and Distributed PZT Sensors. *Sensors*, 152-169.

Paparrizos, I. K., Cambazoglu, B. B., & Gionis, A. (2011). Machine learned job recommendation. *Proceedings of the 2011 ACM Conference on Recommender Systems.* Chicago, USA: RecSys.

Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 559–572.

Pedregosa, F., Varoquaux, G., & Gramfort, A. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2825--2830.

Pottengerb, W., & Kontostathis, A. (2006). A Framework for Understanding Latent Indexing (LSI) Performance. *Elsevier*, 56-73.

Refaeilzadeh, P., Tang, L., & Liu, H. (2008). *Cross-validation.* Phoenix: Arizona State University.

Rehurek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (pp. 45--50). Valletta: ELRA.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning Representations by Back-Propagating the erros. *Nature*, 533-535.

Samuel, A. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 335-365.

Schafer, R. (2011). What Is a Savitzky-Golay Filter? *IEEE Signal Processing Magazine*, 111-117.

Shalaby, W., AlAila, B., Korayem, M., Pournajaf, L., AlJadda, K., Quinn, S., & Zadrozny, W. (2018). Help Me Find a Job: A Graph-based Approach for Job Recommendation at Scale. *2017 IEEE International Conference on Big Data.* IEEE.

Shalizi, C. (2009, November 6). Classification and Regression Trees. *Data Mining*.

Singh , A., Rose, C., Visweswariah , K., Vijil , E., & Kambhatla , N. (2010). PROSPECT: A system for screening candidates for recruitment. *Proceedings of the 19th ACM Conference on Information and Knowledge Management* (pp. 659-668). Toronto: ACM.

Wettayaprasit, W., Laosen, N., & Chevakidagarn, S. (2007). Forecasting, Data Filtering Technique for Neural Networks. *7th WSEAS International Conference on Simulation, Modelling and Optimization*, (pp. 225-230). Beijing.

Wolpert, D., & Macready, W. (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions On Evolutionary Computation*, 67.82.

Yuan, J., Shalaby, W., Korayem, M., Lin, D., AlJadda, K., & Luo, J. (2016). Solving Cold-Start Problem in Large-scale Recommendation Engines: A Deep Learning Approach. *2016 IEEE International Conference on Big Data.* IEEE.

Zalta, E., Nodelman, U., Allen, C., & Anderson, L. (2014, 8 19). *Philosophy of Statistics.* Retrieved from www.stanford.edu: https://plato.stanford.edu/entries/statistics/

Appendix

Metrics/ variables	skillsMatch_tfidf	pplExperience	professionMatch	offeredSalary	expectedSalary
count	32634	32634	32634	32634	32634
mean	0,12	6,40	0,53	31554,45	41915,53
std. dev.	0,12	3,06	0,20	12666,42	18486,37
min	0,00	1,00	0,00	10000,00	10000,00
25%	0,03	4,00	0,37	23200,00	30000,00
50%	0,09	6,00	0,50	26420,53	40000,00
75%	0,18	10,00	0,67	40037,84	50000,00
max	1,00	10,00	1,00	92000,00	150000,00

Table 13 - Data summarization - part I

Metrics/ variables	locationMatch	nAppsPerJob	expectedNumberOfHires	totalVisits	feePercentage
count	32634	32634	32634	32634	32634
mean	0,20	41,74	1,63	7327,50	0,09
std. dev.	0,12	47,05	2,01	6501,01	0,03
min	0,00	1,00	1,00	0,00	0,00
25%	0,13	14,00	1,00	3293,00	0,08
50%	0,25	28,00	1,00	5484,00	0,11
75%	0,30	51,00	2,00	8738,00	0,11
max	0,30	345,00	30,00	52927,00	0,12

Table 14 - Data summarization - part II

id	macro_state	state
-2	Internal	Pre-draft (internal)
-1	Draft	Draft
0	Unreviewed	Unreviewed
10	Unreviewed	Pending Information
11	Unreviewed	Triaged
20	Reviewed	Reviewed
22	Reviewed	Seen by Employer
23	Reviewed	Employer Reviewing
25	Engaged	Engaged
27	Offer	Pre Offer
28	Hired	Pre Hire
29	Offer	Offer Made
30	Hired	Hired
50	Cancelled	Cancelled Before Review

60	Cancelled	Cancelled After Review
94	Cancelled	Cancelled Before Review
95	Cancelled	Cancelled After Review
96	Declined	Closed Before Review
97	Declined	Rejected by Client
98	Declined	Closed After Review
99	Declined	Rejected by HR team

Table 15 - Application states

People (n=53.073)	
Attribute	**Example**
id	6956
city	Lisbon Area, Portugal
created_at	2015-04-09 09:42:25.1160450
updated_at	2016-03-18 13:46:08.0681510
user_id	7014
availability	2
country_code	PT
birth_year	1979
headline	Consultant at Audaxys
relocation_countries	{}
rating	0
how_we_met	6
how_we_met_other	
salary_expectation	NULL
currency_code	EUR
weekly_email	1
cv_content_type	NULL
cv_file_size	NULL
cv_updated_at	NULL
experience_level	10
full_remote	0
full_remote_commute	0
partial_remote	0
freelance	0
recent_grad	0
match_key	cfa0adc1debbba5951afd9edeeb3df47
talent_advocate_id	NULL
google_place_id	NULL
needs_refresh	0
companies_types	{}
consulting	NULL
staffing	NULL

share_profile	NULL
github_hireable	NULL
relocation	0
citizenships	{}

Table 16 - Fields of table People

Applications (n=52.531)	
Attribute	**Example**
id	10647
text	Being enthusiastic to solve problems via Ruby, I ….
job_ad_id	443
person_id	20827
created_at	2015-12-01 13:03:39.0913700
updated_at	2016-03-18 13:56:33.5017440
state	99
starts_on	NULL
skill_assessment_report_file_name	NULL
skill_assessment_report_content_type	NULL
skill_assessment_report_file_size	NULL
skill_assessment_report_updated_at	NULL
skill_assessment_notes	NULL
uuid	D3B8BD17-A116-4F72-943A-0E37FFC23CFF
application_owner_id	49
request_information	NULL
left	NULL
has_work_permit	NULL
seen_by_employer_at	NULL
application_curator_id	49
tracking_codes_raw	
tracking_referrer	NULL
tracking_referrer_domain	NULL
tracking_utm_term	NULL
tracking_utm_medium	NULL
tracking_utm_source	NULL
tracking_utm_content	NULL
tracking_utm_campaign	NULL
tracking_ldref	NULL
tracking_date	NULL
state_comments	NULL
reconsider_message	NULL
candidate_feedback	NULL
is_shortlist	0
match_score	0.2210
submitted_at	2015-12-01 13:03:39.0913700
exclude_website_ids	{}

cv_file_name	NULL
cv_content_type	NULL
cv_file_size	NULL
cv_updated_at	NULL
availability	NULL
availability_detail	NULL

Table 17-Fields of table Applications

Application Audit (n=52.548)	
Attribute	**Example**
id	15147
application_id	15194
pending_info_by_id	61
triaged_by_id	NULL
reviewed_by_id	NULL
rejected_by_id	NULL
rejected_by_type	AdminUser
engaged_by_id	NULL
engaged_by_type	AdminUser
employer_reviewing_by_id	NULL
pre_offer_by_id	NULL
pre_hire_by_id	NULL
offer_by_id	NULL
hired_by_id	NULL
canceled_by_id	NULL
closed_by_id	NULL
pending_info_at	2016-03-22 01:17:39.7529890
triaged_at	NULL
reviewed_at	NULL
rejected_at	NULL
pre_offer_at	NULL
pre_hire_at	NULL
employer_reviewing_at	NULL
engaged_at	NULL
offer_at	NULL
hired_at	NULL
canceled_at	NULL
closed_at	NULL
engaged_date	NULL

Table 18 - Fields of table Application Audit

Job Ads (2.807)	
Attribute	**Example**
id	3266
title	Java Developer

city	Amsterdam
expires_at	NULL
company_id	1677
created_at	2017-03-25 15:20:42.6600580
updated_at	2017-03-26 14:22:08.3325810
slug	draft-3266
employer_fee	0
perks	We have an awesome start-up culture …
job_type	1
country_code	NL
state	0
poster_id	53289
show_salary	0
currency_code	EUR
salary_low	NULL
salary_high	NULL
tag_ids	{144,2880,2003,552,576}
first_published_at	NULL
relocation_paid	1
work_from_home	0
featured	0
category_id	3
visa_support	1
video_marketing	0
gross_salary_low	30000
gross_salary_high	55000
type	Offer::LandingApplications
voucher_id	NULL
jobbox_value	NULL
referral_value	NULL
signing_bonus	0.00
premium	0
experience_level	2
full_remote	0
full_remote_commute	0
partial_remote	0
google_place_id	NULL
closed_by_id	NULL
closed_by_type	NULL
closed_at	NULL
closed_reason	NULL
closed_reason_detail	NULL
citizenship	0
expected_number_of_hires	NULL
info_to_ask_the_candidate	NULL
private_notes_for_landing	NULL

canonicalized_tag_ids	{144,552,576,2003,2880}
application_curator_id	NULL
preferred_language	en
social_title	NULL
social_description	NULL
social_image_file_name	NULL
social_image_content_type	NULL
social_image_file_size	NULL
social_image_updated_at	NULL
total_visits	0
discount_pp	0.0
sale_closed_by_id	NULL
payment_option	1
fee_percentage	0.00
last_published_at	NULL
close_feedback	NULL
staffing	0
consultancy	0
exigency_level	1
perceived_commitment	1
post_wizard_step	hiring_process
education	Master's degree
hiring_process_steps	NULL
extracted_main_skill_ids	{}
extracted_additional_skill_ids	{}
hidden	0
min_applications	5
max_applications	20
review_fee_cents	3000
retainer_fee_cents	0
post_fee_cents	15000
retainer	0
payment_method	NULL
terms_id	NULL
payment_card_id	NULL
tracking_codes	
lead_source	NULL
lead_content	NULL
sale_referral_name	NULL
sale_referral_email	NULL
sale_amount	NULL
recurrent	0
offer_manager_id	88
staffing_partner	0
promo_code	NULL

Table 19 - Fields of table Job ads

Companies (n=1.465)	
Attribute	**Example**
id	993
short_pitch	It, programmer, recruitment IT
created_at	2016-06-03 09:16:51.7596650
updated_at	2016-07-20 14:31:01.4042440
website_url	http://www.bebjobs.pl
notifications_email	NULL
country_code	PL
city	Wroc┼éaw
address	NULL
postcode	NULL
company_category_id	NULL
premium	0
industry_id	10
company_size_id	2
slug	bebjobs-sp-z-o-o
is_strategic	0
how_we_met	9
how_we_met_other	
ats_name	NULL
ats_url	NULL
ats_other	NULL
next_report_delivery_on	NULL
has_activity_for_report	0
staffing	0
consultancy	0
onboarded_at	NULL
onboarded_by_id	NULL
onboarding_comments	NULL
onboarded	0
exigency_level	1
rating	NULL
rating_metadata	{}
perceived_commitment	1
email_deliveries	post_offer_reminder=>"2016-07-20T14:31:01+00:00"
tracking_codes	
lead_source	NULL
lead_content	NULL
segment	0
staffing_partner	0
seg_business_potential	NULL
seg_engagement	NULL
seg_brand_awareness	NULL
seg_strategic_fit	NULL

seg_score	NULL
ats_reject_notify_candidate	1

Table 20 - Fields of table Companies

Tags (n=38.869)	
Attribute	**Example**
id	13
name	AngularJS
created_at	2014-02-15 18:35:11.0815180
updated_at	2016-06-24 15:34:57.0416380
garbage	0
internal_representation	angularjs

Table 21 - Fields of table Tags

Skills (n=683.340)	
Attribute	**Example**
id	169750
person_id	14431
tag_id	458
canonicalized_tag_ids	{458}
experience_level	8
sort_order	4

Table 22 - Fields of table Skills