TENSORFLOW IMPLEMENTATION OF RECOMMENDER SYSTEM

Thesis

Iosif M. Spartalis

Academic supervisor: Georgios Zois
Industry’s supervisor: Thanos Papaoikonomou

Athens
October, 2017
Abstract

Workable company is building recommender systems to aid recruiters and firms find the most suitable candidates for each position with the less human effort. So far, systems have been implemented using scikit-learn and Pytorch in Python. The topic of this thesis is to reimplement the existing machine learning algorithms in Tensorflow, Google’s open-source machine learning software. The objective is to optimize the performance and verify experimentally that the system performs at least as well as the existing implementation, while extending this framework with new features.

Two Machine Learning models are implemented on Python Tensorflow’s API r1.3; a Feed-Forward Neural Network Classifier and a hybrid model which combines a Feed-Forward Neural Network and a Logistic Regression Classifier. The performance of each model is tested by using a preprocessed and cleaned dataset provided by Workable.

In the first two chapters of this thesis, we present some of the fundamental theoretical concepts on Recommender Systems and Machine Learning techniques. In the third chapter, we describe in detail the main thesis topic, by presenting the dataset used, the two Classifiers that are implemented and their testing results. Finally, in the fourth chapter we present the final conclusions and remarks.
Acknowledgements

I have chosen this subject as my thesis topic because I wanted to delve more into the field of Machine Learning. In spite the short time, elaborating my dissertation project was indeed a fulfilling and didactic procedure. In addition, it was a great opportunity for me to collaborate with Workable, one of the fastest developing technology companies in Greece and have the chance to get my hands dirty with Google’s Tensorflow, which is on the cutting-edge of Deep Learning technology at the moment. At this point, I would like to thank my academic supervisor Georgios Zois for his valuable guidance throughout the project and his patience. I’ve would also like to thank Thanos Papaoikonomou in Workable for his spotless collaboration and friendliness. Last and not least, I’d like to thank professor Vassilis Vassalos for his overall supervision and contribution on this project and his valuable remarks.
# Table of Contents

Abstract ............................................................................................................................................................... iii

Acknowledgements ............................................................................................................................................. v

Table of Figures ............................................................................................................................................... viii

Table of Tables ................................................................................................................................................. viii

1 Recommender Systems fundamentals ............................................................................................... - 1 -
   1.1 Brief introduction ........................................................................................................................ - 1 -
   1.2 Content – based Information Filtering .................................................................................... - 3 -
   1.3 Collaborative Filtering ................................................................................................................ - 5 -

2 Machine Learning fundamentals ........................................................................................................ - 9 -
   2.1 Machine Learning (ML) techniques .......................................................................................... - 9 -
   2.2 Linear and Logistic Regression ................................................................................................ - 10 -
       2.2.1 Linear Regression ........................................................................................................... - 10 -
       2.2.2 Logistic Regression Classification .................................................................................. - 13 -
   2.3 Artificial Neural Networks (ANN) ......................................................................................... - 14 -
   2.4 Evaluation metrics in Classification ........................................................................................ - 17 -

3 Machine Learning implementation .................................................................................................. - 19 -
   3.1 Recommender system overview .............................................................................................. - 19 -
   3.2 Tensorflow – Google’s Machine Learning API ................................................................... - 19 -
   3.3 The entities of the Recommender System ............................................................................. - 19 -
   3.4 The dataset .................................................................................................................................. - 20 -
   3.5 The Machine Learning implementation ................................................................................. - 21 -
       3.5.1 Data preprocessing ........................................................................................................... - 21 -
       3.5.2 The Deep Model ............................................................................................................. - 22 -
       3.5.3 The Wide and Deep Model ............................................................................................. - 24 -
       3.5.4 Remarks on Models implementation in Python Tensorflow API ................................ - 26 -
Table of Figures

Figure 1-1 Phases of recommendation process. (source Folajimi, Isinkaye, Ojokoh, 2015) .......... - 3 -
Figure 1-2 The architecture of a content-based recommender system (Source: Ansgar Roald Koene, May 13, 2016) .................................................................................................................................................. - 4 -
Figure 1-3 User – item matrix, prediction and recommendation ...................................................... - 5 -
Figure 1-4 Collaborative filtering methods ............................................................................................. - 6 -
Figure 2-1 Machine Learning categories and application examples ..................................................... - 10 -
Figure 2-2 A simple example of Linear Regression with linear hypotheses function: \( y = f_{w_1, w_2}(x) = w_1 x + w_0 \) .................................................................................................................................................................... - 11 -
Figure 2-3 Gradient Descent example in two dimensions .................................................................... - 12 -
Figure 2-4 Linear classification of two classes in two dimensions ....................................................... - 13 -
Figure 2-5 Detail of an Artificial Neuron (Source: Vlahavas) ............................................................. - 14 -
Figure 2-6 Most commonly used activation functions (Source: Raschka 2016) ................................ - 15 -
Figure 2-7 Feed-Forward NN with one hidden layer ........................................................................... - 15 -
Figure 3-1 The Deep Model ....................................................................................................................... - 23 -
Figure 3-2 The Wide and Deep Model ..................................................................................................... - 25 -
Figure 3-4 Tensorboard’s representation of embedding keywords for “skill” category and the 12 closest embeddings for the keyword “sales” ........................................................................................................ - 28 -

Table of Tables

Table 3-1 Candidate features ..................................................................................................................... - 20 -
Table 3-2 Job features ................................................................................................................................. - 20 -
Table 3-3 Vocabulary minimum keyword frequency per category ....................................................... - 22 -
Table 3-4 Vocabulary sizes ......................................................................................................................... - 22 -
Table 3-5 Dimensions of embeddings vectors ........................................................................................ - 22 -
Table 3-6 The model function of the Deep Model ................................................................................ - 26 -
Table 3-7 The evaluation scores for each model .................................................................................... - 28 -
1 Recommender Systems fundamentals

1.1 Brief introduction

Recommender systems (RS) are software tools and techniques providing suggestions of items that may have a use to a user. Those suggestions are related to decision making processes like, what product to buy? what song to listen? what video to watch? what article to read online? or even what job position to apply for? The term “item” is referred to what the recommender system suggests to the users. In most cases, a recommender system focuses on a specific type of items such as books, movies, apps etc. In other words, a recommender system could be viewed as a ranking system, which utilizes a set of users, items and, in some cases, contextual information, to provide as an output a ranked list of items.

Such systems are widely used in e-commerce platforms, but their usage expands in other applications, like entertainment, content-personalized newspapers, recommendation of documents and webpages, travel services, consulting services etc. Highly rated internet sites as Amazon, YouTube, Netflix, TripAdvisor, Last.fm and media companies have developed and deployed recommender systems.

There are three major entities which play an important role in recommender systems: the items, the users, and the transactions. The items are the objects that are recommended by the system. On the other hand, the user is the subject that interacts with the items (rates a movie, purchases a product, clicks a link etc.) and this process is called a transaction which is logged by the system. All these elements can be modeled in various ways according to the application. For instance, in a movie rental recommender system, a movie (item) could have features for genre, director, cast, year of production etc. The transaction is the act of rating a movie by the user.

The approaches that are used by recommender systems to make predictions are called recommendation techniques. A classical way of distinguishing between different recommender systems is based on those techniques [1] and there are six classes:

- **Content-based Information Filtering**: The system suggests items that are similar to the items that the user preferred in the past. The items are modeled with feature vectors and a similarity
measure (i.e. cosine, Euclidean distance) is calculated. The ranking is based on the user profile which is trained by using the history of user’s activity.

- **Collaborative Filtering:** The system suggests items to the active user that other users with similar tastes liked in the past. The similarity in taste of two users is calculated based on the similarity in the rating history of the users. Collaborative filtering is considered to be the most popular and widely implemented technique in Recommenders Systems [2].

- **Demographic:** The system recommends items based on the demographic profile of the user. For instance, different items would be recommended to different sets of users that are grouped either by their age, their nationality or their language.

- **Knowledge – based:** The system suggests products based on inferences about a user’s needs and preferences. The system learns those preferences by asking the user to give this information. Each item has certain features and the needs of a user could be modeled with certain methods. One of them is using a query, i.e., the set of preferred features for a product and another one is using an adapted similarity metric for matching [3].

- **Community(Social)-based:** This type of system recommends items based on the preferences of the users’ friends [4]. People tend to rely more on recommendations from their friends than on recommendations from similar but anonymous individuals. This observation, combined with the growing popularity of open social networks, is generating a rising interest in community-based systems or, as they usually referred to, social recommender systems.

- **Hybrid Recommender Systems:** These systems are based on a combination of the above mention techniques and attempt to compensate between the shortcomings and the advantages of those individual techniques [1].

However, most of these systems bear in their core an algorithm that can be understood as a particular instance of a Data Mining (DM) technique. Thus, a complete recommender system has procedures of data preprocessing (i.e. sampling, dimension reduction), data analysis (i.e. classification, clustering) and result interpretation. These three groups of procedures constitute the three major phases of the recommendation process: the information collection phase, the learning phase and the prediction/recommendation phase.

In information collection phase, the relevant information of users is collected and a user profile or model is built for the prediction tasks. This information may consist of the user’s attribute, behaviors or content of the resources the user accesses. A well-constructed user profile is crucial for the recommendation system to function accurately. In the learning phase, the system applies a learning algorithm to filter and exploit the user’s features from the feedback gathered in information collection phase. In the final phase of prediction/recommendation, the system predicts which items the user may prefer. This process can be accomplished either directly based on the dataset collected in information collection phase or through the system’s observed activities of the user.
Chapter 1: Recommender Systems fundamentals

As above-mentioned, a critical technical ingredient of this thesis is the process of classification, which is a central machine learning problem, in the special case of a Content-based Recommender System. Thus, the next two sections are dedicated in describing the major features, components, advantages and limitations of two commonly used techniques in recommender systems, Content-based and Collaborative Filtering.

1.2 Content – based Information Filtering

Content-based recommendation systems try to recommend items similar to those that a given user has liked in the past, whereas systems designed according to the collaborative recommendation paradigm identify users whose preferences are similar to those of the given user and recommend items that the users have liked [5]. The modeling of the items (content’s feature selection) plays a major role in this kind of systems, while the information for the users is limited in past ratings, likes, dislikes, etc.

The high-level architecture of a content-based recommender system is composed of three parts: the Content Analyzer, the Profile Learner and the Filtering Component.

**Content Analyzer** is the part where the data pre-processing is made. Usually, the information has no structure (i.e. text, webpage) and in this step the data items are analyzed by feature extraction techniques to shift from the original information space to the target one. The final structured data items that are produced at this step, are used as input in the Profile Analyzer and the Filtering Component.

**Profile Learner** is the module which collects the data that represent the user preferences, and tries to generalize this knowledge, in order to build the user profile. In most cases, this generalization strategy is accomplished through machine learning techniques and distinct separate models are produced for each user based on the items that liked and disliked in the past.

**Filtering Component** is the module that exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. In sort, this component makes the ranking of the suggested items.
One of the major advantages of the Content-based approach is that the system is able to recommend new items that haven't been rated by any user. Therefore, content-based systems do not suffer from the first-rater problem, which affects collaborative recommenders, which rely solely on users' preferences to make recommendations. Also, they have the ability to adapt when the user preferences change and has the capacity to adjust its recommendations in a short span of time. Finally, content-based systems can manage situations where different users do not share the same items, but only identical items according to their intrinsic features.

On the other hand, in a situation when a new user has few or not at all ratings, the system isn't able to make a reliable suggestion. Another limitation of content-based filtering techniques is that are highly dependent on items' metadata. That is, they require rich description of items and very well-organized user profile before recommendation can be made to users. This is called limited content analysis. So, the effectiveness of the contend-based system depends on the availability of descriptive data. Over-Specialization is another drawback of this approach. Content-based systems are not capable of suggesting something entirely unexpected, in other words the system suggests items which are similar to those already rated by the user in the past. But in real word, most humans like to discover new things, like watching a new movie from an unknown director or listening a new genre of music. Collaborative Filtering tries to tackle this problem.
1.3 Collaborative Filtering

Collaborative filtering is a prediction technique for content that cannot easily and adequately be described by metadata such as movies and music. It works by building a database (user-item matrix) of preferences for items by users and then it matches users with relevant interest and preferences by calculating similarities between their profiles to make recommendations [6]. Such users form a group called neighborhood. A user gets recommendations to those items that he has not rated before but that were already positively rated by users in his neighborhood. Recommendations that are produced by collaborative filtering can be of either prediction or recommendation. Prediction is a numerical value, $R_{ij}$, expressing the predicted score of item $j$ for the user $i$, while recommendation is a list of top $N$ items that the user will like the most (Figure 1-3).

Collaborative approaches can be grouped into two general classes of neighborhood-based (or memory-based) and model-based methods [7, 8, 9, 10]. In neighborhood based collaborative filtering methods, the user-item ratings are directly used to predict ratings for new items. Neighborhood-based methods can be accomplished in two ways known as user-based or item-based recommendation (Figure 1-4). In user-based systems, the interest of a user for an item is evaluated by using the ratings for this item by other users which they called neighbors. The metric that is used for quantifying this neighborhood is based on how similar are the ratings patterns of the current user with the rest of them. On the other hand, item-based approaches try to predict the ratings of a user for a certain item based on the ratings of the same user for items similar to that item. This similarity between two items is based on the way that several users rated those two items. It must be pointed out, that in a content-based method the similarity of two items is based on how close are the feature vectors that represent those items.

![User-item rating matrix](image)

*Figure 1-3 User–item matrix, prediction and recommendation*
In neighborhood-based systems the predictions are made using directly the stored ratings. This is not the case in model-based approaches which use the ratings in order to learn a predictive model. The model building process can be done using machine learning or data mining techniques. These techniques can quickly recommend a set of items for the fact that they use pre-computed model and they have proved to produce recommendation results that are similar to neighborhood-based recommender techniques. Models, like matrix factorization (SVD), transform both item and users to the same latent factor space and then try to explain the ratings by characterizing both items and users on factors automatically inferred from user feedback. In other words, the user-item interactions are modeled with factors representing latent characteristics of the users and items in the system. The model is trained using the available data and after that it is used to predict ratings of users for new items. Model-based approaches in collaborative filtering are numerous and include Bayesian Clustering [9], Latent Semantic Analysis [11], Latent Dirichlet Allocation [12], Maximum Entropy [13], Boltzmann Machines [14], Support Vector Machines [15], Singular Value Decomposition [16, 17, 18, 19], and Artificial Neural Networks [20].

Loosely speaking model-based approaches deliver high expressive ability in describing various aspects of the data and thus they tend to provide more accurate results than neighborhood models. However, accuracy alone does not guarantee users an effective and satisfying experience [21]. A factor that has been identified as playing an important role in the appreciation of users for the recommender system is its ability to provide serendipitous recommendations, which means that it can recommend items that are relevant to the user even without the content being in the user’s profile. Collaborative filtering and especially the neighborhood-based techniques have the ability of serendipity. Neighborhood-based methods have four main advantages over model-based ones: simplicity, justifiability, efficiency and stability. In respect with simplicity and justifiability, those methods are intuitive and relatively simple to implement and they provide a concise and intuitive justification for the computed predictions. As for their efficiency, unlike
most model-based systems, they require no costly training phases which need to be carried out at frequent intervals. Also, storing the nearest-neighbors (of user or items) requires very little memory which makes those approaches scalable to applications with millions of users and items. Lastly, the stability of recommender systems using neighborhood based techniques are little affected by the constant addition of users, items and ratings. In the scenario when a new item is inserted to the system, once its similarities have been computed, an item-based system has the ability to make recommendations to new users, without the need of re-training the system.

Collaborative Filtering has some major advantages over content-based techniques. For example, when the content of items is not available or difficult to obtain (i.e. features extraction of audio files), a collaborative system is able to recommend them to users through the feedback of other users. Another advantage of collaborative approaches is that, unlike content-based systems, they can recommend items with very different content, as long as other users have already rated them in the past. Also, in some cases the evaluation of an item’s quality is more reliable when it is based on user feedback instead relying on item’s content. As mentioned previously, collaborative techniques have the ability to provide serendipitous recommendations, which means that it can recommend items that are relevant to the user even without the content being in the user’s profile.

However, the widespread use of collaborative filtering techniques has revealed some potential problems. For instance, the cold-start problem is a situation where a recommender does not have adequate information about a user or an item in order to make relevant predictions [1]. A situation as this is one of the major issues that reduce the performance of recommendation systems. The data sparsity is another problem which occurs in occasions where only a few of the total number of items available in a database are rated by users [22]. Such occasions lead to a sparse user-item matrix and the system due to its inability to locate successfully neighbors generates weak predictions.

An additional issue that collaborative filtering techniques have to overcome is the one of scalability. While more users and items are inserted in the database, the algorithm computational cost is increasing in a linear way. Scaling up in a successful manner as the number of dataset in a database increases is crucial. Methods used for solving scalability problem and speeding up recommendation generation are based on Dimensionality reduction techniques, such as Singular Value Decomposition (SVD) method, which has the ability to produce reliable and efficient recommendations.

Finally, another problem that has to be overcome is synonymy. Synonymy is the tendency of very similar items to have different names or entries. Correlation based recommender systems can't find this latent association and treat these items differently. For example, let us consider two customers one of them rates 10 different recycled letter pad products as "high" and another customer rates 10 different recycled memo pad products "high". Correlation based recommender systems would see no match between product sets to compute correlation and would be unable to discover the latent association that both of them like recycled office products. Different methods, such as automatic term expansion, the construction of a
Chapter 1: Recommender Systems fundamentals

thesaurus, and Singular Value Decomposition (SVD), especially Latent Semantic Indexing are capable of solving the synonymy problem [23]. The shortcoming of these methods is that some added terms may have different meanings from what is intended, which sometimes leads to rapid degradation of recommendation performance.
2 Machine Learning fundamentals

2.1 Machine Learning (ML) techniques

Learning is the process of knowledge acquisition. Humans naturally learn from experience because of their ability to reason. In contrast, computers do not learn by reasoning, but learn with algorithms. Today, there are a significant number of ML algorithms proposed in the literature. They can be classified based on the approach used for the learning process. There are four main categories: supervised, unsupervised, semi-supervised, and reinforcement learning. Further sub-categories and some examples of their applications are shown in Figure 2-1.

**Supervised learning** happens when algorithms are provided with training data and correct answers. The task of this kind of ML algorithms is to learn based on the training data, and to apply the knowledge that was gained in real data. Some algorithms that belong to this family are:

- Linear and Logistic Regression/Classification
- Nearest Neighbor
- Naive Bayes
- Decision Trees
- Support Vector Machines (SVM)
- Artificial Neural Networks

In **unsupervised learning**, ML algorithms do not have a training set. They are presented with some data about the real world and have to learn from that data on their own. Unsupervised learning algorithms are mostly focused on finding hidden patterns in data. The most known algorithms in this category are:

- k-means clustering
- Association Rules
Semi-supervised learning falls in between the previous techniques. This method is used when algorithms work with a training set with missing information, and still need to learn from it.

Lastly, Reinforcement learning occurs when algorithms learn based on external feedback given either by a thinking entity, or the environment. Some of the algorithms that belong to this category are:

- Q-Learning
- Temporal Difference (TD)
- Deep Adversarial Networks

In this thesis, the techniques that are used are based on two of the supervised methods, more specifically on Linear Logistic Regression and on Neural Networks. In the next section, some of the fundamental ideas behind these two families of algorithms will be presented.

## 2.2 Linear and Logistic Regression

### 2.2.1 Linear Regression

The simplest linear model for regression is one that involves a linear combination of the input variables

\[
y(\tilde{x}, \tilde{y}) = w_0 + w_1 x_1 + \cdots + w_D x_D
\]

where \(\tilde{x} = (x_1, \ldots, x_D)^T\) is the features vector and \(y\) is a real number. This is often simply known as linear regression. The key property of this model is that it is a linear function of the parameters \(w_0, \ldots, w_D\), which are also referred as weights. It is also, however, a linear function of the input variables \(x\), and this imposes significant limitations on the model. We therefore extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

\[
y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \varphi_j(x)
\]
Where \( \varphi_j(\vec{x}) \) are known as basis functions. By denoting the maximum value of the index \( j \) by \( M - 1 \), the total number of parameters in this model will be \( M \). The parameter \( w_0 \) allows for any fixed offset in the data and is sometimes called a bias parameter.

By using nonlinear basis functions, function \( y(x, w) \) is allowed to be a nonlinear function of the input vector \( x \). Functions of the form (2-2) are called linear models, however, because they are linear in \( w \).

There are many other possible choices for the basis functions, for example

\[
\varphi_j(x) = \exp \left\{-\frac{(x-\mu_j)^2}{2s^2}\right\} \quad (2-3)
\]

where the \( \mu_j \) govern the locations of the basis functions in input space, and the parameter \( s \) governs their spatial scale. These are usually referred to as “Gaussian” basis functions, although it should be noted that they are not required to have a probabilistic interpretation, and in particular the normalization coefficient is unimportant because these basis functions will be multiplied by adaptive parameters \( w_j \).

Another possibility is the sigmoidal basis function of the form

\[
\varphi_j(x) = \sigma \left(\frac{x-\mu_j}{s}\right) \quad (2-4)
\]

where \( \sigma(a) \) is the logistic sigmoid function defined by

\[
\sigma(a) = \frac{1}{1+e^{-a}} \quad (2-5)
\]

The objective is to find the best values of weights \( w \) through the process of training the model. But first, a criterion must be set in order to define the term “best”. For this purpose, a loss function is used and the best possible values of \( w \) are those that minimize this function. One of the most common loss functions is the squared error loss function.
Chapter 2: Machine Learning Fundamentals

\[ E(\vec{w}) = \frac{1}{2} \sum_{i=1}^{m} [f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2 \]  

where \( \vec{x}^{(i)} \) is the \( i \)-th training example, \( y^{(i)} \) is the correct (desired) output for \( \vec{x}^{(i)} \) and \( m \) is the number of training examples.

Because the problem of finding the minimum of the loss function either does not always have a closed-formed solution or it’s computational cost increases very quickly for high-dimensional spaces, it is commonly used the iterative optimization method of Gradient Descent. The vector of weights \( \vec{w} \) is updated iteratively with the update rule

\[ \vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \nabla E(\vec{w}) \]  

where in the simplest case, \( \eta \) is a small positive constant. In essence, this method tries to locate the local minima by taking small steps, each time in the opposite direction of the gradient of the loss function \( \nabla E(\vec{w}) \) at the current position \( \vec{w} \) (Figure 2-3). At the beginning of the iterative process, the weights are initialized with random values (usually from a gaussian distribution \( \mathcal{N}(0, s) \)). If all the training examples are used in every updating step, then the process is called batch Gradient Descent, but is not used in practice due to its computational cost. Stochastic Gradient Descent (SGD) on the other hand, uses only one training example at a time. SGD has much smaller computational cost which is preferable when the training dataset is large. The update steps do not always go towards the minimum of the total error \( E(\vec{w}) \), but go towards the local error \( E_i(\vec{w}) \) of the \( i \)-th training example. For this reason, a mini-batch of training example is usually used. Also, it is possible the algorithm may not reach the local minimum of \( E(\vec{w}) \). It may start wandering around the minimum, but in practice, it arrives close enough and much faster than the batch Gradient Descent.

Figure 2-3 Gradient Descent example in two dimensions
2.2.2 Logistic Regression Classification

In the case of linear regression, the objective is to predict a continuous value \( y(\vec{x}) \) when certain features \( \vec{x} \) are known. In classification, the goal is to take an input vector \( \vec{x} \) and to assign it to one of \( K \) discrete classes \( C_k \) where \( k = 1, \ldots, K \). Usually, the classes are taken to be disjoint, so that each input is assigned to one and only one class. The input space is thereby divided into decision regions whose boundaries are called decision boundaries or decision surfaces. A simple case of a binary classification (2 classes) is illustrated in Figure 2-4, with three potential decision boundaries \( H_1, H_2 \) and \( H_3 \). The goal is to find the straight line that separates successfully the two classes, which has the form

\[
w_2 x_2 + w_1 x_1 + w_0 = 0 \tag{2-8}
\]

For more features \( x_1, x_2, \ldots, x_n \), the decision surface is a hyperplane that separates the two classes.

\[
w_n x_n + \cdots + w_1 x_1 + w_0 = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x} = 0 \tag{2-9}
\]

The classification decision \( C \) is given by

\[
C = \text{sign}(\vec{w} \cdot \vec{x}) \tag{2-10}
\]

But, in many applications it is required a classifier to return a probability. In such case, the probability for \( \vec{x} \) to belong in the positive class is given by

\[
P(c_1|\vec{x}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}} \tag{2-11}
\]

While the probability of \( x \) to belong in the negative class is given by

\[
P(c_0|\vec{x}) = 1 - P(c_0|\vec{x}) = \frac{e^{-\vec{w} \cdot \vec{x}}}{1 + e^{-\vec{w} \cdot \vec{x}}} \tag{2-12}
\]
During training, the objective is to select the $w$ that makes the classifier more confident that the training examples belong in their correct classes. In other words, to find the weights' values that maximize the conditional likelihood of the examples

$$L(w) = P(\hat{y}^{(1)}, ..., \hat{y}^{(m)}|\hat{x}^{(1)}, ..., \hat{x}^{(m)}; w)$$  \hspace{1cm} (2-13)

Where $\hat{y}^{(1)}, ..., \hat{y}^{(m)}$, are the correct classes of the $m$ training examples with feature vectors $\hat{x}^{(1)}, ..., \hat{x}^{(m)}$. Assuming that the training example are independent and identically distributed the likelihood function (2-13) becomes

$$L(w) = \prod_{i=1}^{m} P(\hat{y}^{(i)}|\hat{x}^{(i)}; w)$$  \hspace{1cm} (2-14)

It is easier thought, instead of maximizing the $L(w)$, to maximize the (conditional) log-likelihood

$$l(w) = \log L(w) = \sum_{i=1}^{m} \log p(\hat{y}^{(i)}|\hat{x}^{(i)}; w)$$  \hspace{1cm} (2-15)

Often a regularization term is added for lowering the risk of over-fitting the training data

$$l(w) - \lambda \cdot \|w\|^2 = l(w) - \lambda \cdot \sum_{i=1}^{n} w_i^2$$  \hspace{1cm} (2-16)

where $n$ is the dimension of the vector $w$. (2-16) is called a L2 regularization and $\lambda$ is a positive real number.

Maximizing the (2-16) requires an optimization method like stochastic gradient ascent which does not have a closed-form solution.

### 2.3 Artificial Neural Networks (ANN)

The term “neural network” (NN) is inspired from the structure of the biological brain who’s the fundamental unit is the neuron. For instance, a tiny piece of the human’s brain, about the size of grain of rice, contains over 10,000 neurons, each of which forms an average of 6,000 connections with other neurons. It’s this massive biological network that enables us to experience the world around us. Artificial neural networks mimic this natural structure in order to help the machine to learn in analogous way. Although, it
must be noticed that Artificial Neural Networks are loosely inspired by natural NNs, and probably they are very different.

In similar fashion with natural neural networks, the fundamental unit in ANN is the artificial neuron. An artificial neuron takes as inputs two vectors with real values, a vector \( \vec{x} = [x_1, x_2, \ldots, x_n] \) and a weight vector \( \vec{w} = [w_1, w_2, \ldots, w_n] \). The Soma of the neuron compute the weighted sum \( S = \sum w_i x_i \) of the inputs, and then applies an activation function \( \Phi(S) \) to the sum. The most commonly used activations functions are shown in Figure 2-6.

<table>
<thead>
<tr>
<th>Activation function</th>
<th>Equation</th>
<th>Example</th>
</tr>
</thead>
</table>
| Unit step (Heaviside) | \( \phi(z) = \begin{cases} 
0, & z < 0, \\
0.5, & z = 0, \\
1, & z > 0, 
\end{cases} \) | Perceptron variant |
| Sign (Signum) | \( \phi(z) = \begin{cases} 
-1, & z < 0, \\
0, & z = 0, \\
1, & z > 0, 
\end{cases} \) | Perceptron variant |
| Linear | \( \phi(z) = z \) | Adaline, linear regression |
| Piece-wise linear | \( \phi(z) = \begin{cases} 
1, & z \geq \frac{1}{2}, \\
\frac{z + \frac{1}{2}}{\frac{1}{2}}, & -\frac{1}{2} < z < \frac{1}{2}, \\
0, & z \leq -\frac{1}{2}. 
\end{cases} \) | Support vector machine |
| Logistic (sigmoid) | \( \phi(z) = \frac{1}{1 + e^{-z}} \) | Logistic regression, Multi-layer NN |
| Hyperbolic tangent | \( \phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \) | Multi-layer Neural Networks |
| Rectifier, ReLU (Rectified Linear Unit) | \( \phi(z) = \max(0, z) \) | Multi-layer Neural Networks |
| Rectifier, softplus | \( \phi(z) = \ln(1 + e^z) \) | Multi-layer Neural Networks |

Figure 2-6 Most commonly used activation functions (Source: Raschka 2016)

A neural network is just a collection of units connected together; the properties of the network are determined by its topology and the properties of the “neurons”. One of simplest type of neural network is the Feed-Forward Neural Network (FFNN), which is usually arranged in layers, such that each unit (neuron) receives input only from units in the immediately preceding layer. A network with all the inputs connected directly to the outputs is called a single-layer neural network or a perceptron network. Such net-
work, as a classifier, is able to learn only linear separators, like the logistic regression classifier. Approximately, by adding in a FFNN one hidden layer the network can compute almost any bounded continuous function, and by two hidden layers can compute almost any bounded function \[24\].

A neural network is just a collection of units connected together; the properties of the network are determined by its topology and the properties of the “neurons”. One of simplest type of neural network is the Feed-Forward Neural Network (FFNN), which is usually arranged in layers, such that each unit (neuron) receives input only from units in the immediately preceding layer. A network with all the inputs connected directly to the outputs is called a single-layer neural network or a perceptron network. Such network, as a classifier, is able to learn only linear separators, like the logistic regression classifier. Approximately, by adding in a FFNN one hidden layer the network can compute almost any bounded continuous function, and by two hidden layers can compute almost any bounded function [24] depicts a simple Feed-Forward Neural Network Classifier with one hidden layer. The Forward Propagation procedure starts when the input vector \( \mathbf{x} = [x_1, \ldots, x_m] \) is fed to the hidden layer by multiplying it with the weight matrix \( W^{(1)} \).

Then, the hidden layer applies the activation function to the matrix-vector product \( \tilde{s}^{(1)} = W^{(1)} \mathbf{x} \) and feeds the activation vector \( \tilde{o}^{(1)} = [o_{1,1}, o_{1,2}, \ldots, o_{1,k_1}] = \text{tanh}(\tilde{s}^{(1)}) \) to the output layer by multiplying it with the second weight matrix \( W^{(2)} \). On the matrix-vector product \( \tilde{s}^{(2)} = W^{(1)} \tilde{o}^{(1)} \) it is applied a softmax function and the vector \( \tilde{o}^{(2)} = [o_{2,1}, o_{2,2}, \ldots, o_{2,k_2}] = \text{Softmax}(\tilde{s}^{(2)}) \) is produced. The values \( o_{2,1}, o_{2,2}, \ldots, o_{2,k_2} \) are the probabilities of the example with the input instance \( \mathbf{x} \) to belong to one of the \( K_2 \) classes.
\[ P(y = j | \tilde{x}) = \frac{e^{o^{(1)T}w^{(2)}_j}}{\sum_{k=1}^{K_2} e^{o^{(1)T}w^{(2)}_k}} \text{ for } j = 1, \ldots K_2 \]  

(2-17)

For training the FFNN classifier as loss function is commonly used the *cross entropy* function

\[ l(\tilde{x}) = -\sum_{j=1}^{K_2} t_j \log(o_2, j) \]  

(2-18)

The training of the network is done by minimizing this function using an optimization algorithm like SGD in combination with the *backpropagation*, which is a method that calculates the error contribution of each neuron after a batch of data is processed.

### 2.4 Evaluation metrics in Classification

The performance of a classifier is measured with certain evaluation metrics. The most common is the *accuracy* metric

\[ \text{Accuracy} = \frac{\text{correct decisions}}{\text{Total decisions}} \]  

(2-19)

Accuracy though is not always a good evaluation measure. For instance, if a binary classifier only classifies in the most frequent class and 80% of the instances belong to this class, then the accuracy of the classifier would be 80%, which is misleading.

Other evaluation metric are the *precision* and the *recall* of a class. Precision express how many of the instances classified in the class (true positives + false positives) are true members of the class (true positives). On the other hand, recall express how many of the true members of the class (true positives + false negatives) are classified in the class.

\[ \text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \]  

(2-20)

Another metric that is use is the *F-measure* (2-21) which combines precision and recall. When \( \beta = 1 \) it assigns equal weights to them and gives the \( F_1 \) measure.

\[ F_\beta = \frac{(\beta^2 + 1) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \]  

(2-21)

Because the above metrics are referred to one class, there some metrics that quantify the average performance of the classifier for all classes. *Macro-averaging* treats all classes equally by assigning equal weights to all of them. On the other hand, *micro-averaging* treats the frequent classes as more important.

The first category of average metrics includes

\[ \text{MacroPrecision} = \frac{1}{n} \sum_{i=1}^{n} \text{Precision} \]  

(2-22)

\[ \text{MacroRecall} = \frac{1}{n} \sum_{i=1}^{n} \text{Recall} \]  

(2-23)
The micro-averaging category includes

\[
\text{MicroPrecision} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} (TP_i + FP_i)} \tag{2-24}
\]

\[
\text{MicroRecall} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} (TP_i + FN_i)} \tag{2-25}
\]

Another measure that is used in evaluation, is the area under the curve (AUC) of a receiver operating characteristic curve, i.e. ROC curve. The ROC curve is created by plotting the recall against the false positive rate (FPR) at various threshold settings. The false-positive rate, which is also known as the fall-out or probability of false alarm, is calculated by

\[
FPR = \frac{FP}{FP + TN} \tag{2-26}
\]
3 Machine Learning implementation

3.1 Recommender system overview

The main goal of this thesis was to implement certain machine learning algorithms as part of an overall process of building a recommender system which purpose is to aid recruiters and firms find the most suitable candidates for each position with less human effort. So far, systems have been implemented using scikit-learn [24] and Pytorch [25]. The topic of this thesis is to reimplement the existing algorithms in Tensorflow, optimize their performance, verify experimentally that the system performs at least as well as the existing implementation, and extend these techniques with new features. The implementation is made in Python Tensorflow API r1.3.

3.2 Tensorflow – Google’s Machine Learning API

TensorFlow is Google’s open-source and powerful artificial intelligence software, which powers many services and initiatives from Google. It is the second generation of a system for large-scale machine learning implementations, built by the Google Brain team.

Tensorflow represents computation as stateful data flow graphs. What makes this technology unique is its ability to model computations on a vast range of hardware, from consumer-level mobile devices to world-class multi-GPU servers. It can run on different GPUs and CPUs and promises the scalability of machine learning among the various devices and gadgets without having to alter a significant amount of code. This feature has played decisive role in Workable’s decision for reimplementing the existing machining learning algorithms in Tensorflow.

3.3 The entities of the Recommender System

Each recommender system has three major entities: users, items and transactions. The purpose of the recommender system is to suggest a list of items to an active user according to the particular user’s profile and needs. In our case, the job position has the role of the user and the candidate has the role of the item. A transaction is considered to have a positive outcome, which is encoded as one if the candidate who had applied for the job position is called for an interview by the firm, otherwise is encoded as zero.
Both, job position and candidate are modeled with certain features. This approach is similar with the one used in the content-based recommender systems, but in this case both users and items are modeled based on their content. The candidate has three features: skills, experience and education. Those three features include a list of various keywords describing the various skills, education titles and previous working experience candidate may have (Table 3-1). In a similar way, the job position has two features: title and keywords (Table 3-2). The title contains the title of the job position and the keywords feature contains list of keywords that describe the job position tasks. It must be noticed that those keywords may contain more than one word.

<table>
<thead>
<tr>
<th>Skills</th>
<th>Experience</th>
<th>Education</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leadership, microsoft_exchange, computer_hardware, hyper-v, office_365,...</td>
<td>sales_representative, customer_service, it_systems_administrator</td>
<td>electronics</td>
</tr>
<tr>
<td>mobile_design, layout, actionscript, graphic_design, video_editing, css, 3d_studio_max, web_development,...</td>
<td>designer, content_specialist, graphic_designer &amp; marketing, admin_assistant, team_lead</td>
<td>artes_plasticas_e_multimedia</td>
</tr>
<tr>
<td>stocking_items, strong_computer_and_internet_skills, using_a_cash_register, manual_labor, cleaning_duties</td>
<td>sandwich_artist, stock_associate, crew_member, pre-loader</td>
<td>business_management</td>
</tr>
</tbody>
</table>

Table 3-1 Candidate features.

<table>
<thead>
<tr>
<th>Title</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_engineer</td>
<td>bgp, cabling, data_center, cisco, mpls</td>
</tr>
<tr>
<td>driver</td>
<td>Courier, delivery_job, delivery, Gloucester, dispatch_rider</td>
</tr>
<tr>
<td>brand_ambassador</td>
<td>Commission, golf, sales, promotions</td>
</tr>
</tbody>
</table>

Table 3-2 Job features

### 3.4 The dataset

The dataset that was used, had the form of a csv file with 6 columns: Skills, Experience, Education, Title, Keywords and Label. Each row represents a transaction which a candidate with a certain skills, experience and education applies for a job position with the certain title and keywords. The label takes two values, one if the firm calls the candidate for an interview and zero if it doesn’t. The size of the dataset was around 800,000 entries.

The dataset was created by employing existing real-world data of successful cases where a firm has evaluated the candidate’s data and the candidate is called for an interview. The number of those entries was around 200,000. Next, by randomly picking candidate ids and job ids from the initial real-world data, another 600,000 artificial entries were produced. Due to the random way of producing these entries, it was
assumed that the outcome of the job application will be negative (label = 0). Although, it is possible that some combinations of job ids and candidate ids could exist in the initial real-word data with a positive outcome, but for simplicity it is assumed that this possibility is neglectable. Eventually, the final ratio of the negative (label=1) and positive (label=0) outcomes was 3 over 1.

3.5 The Machine Learning implementation

The main objective of the current machine learning task is to have a trained model that would be able to predict if a certain candidate is suitable for a given job position. Candidate’s and job’s feature vectors would be used as input data and the result would be either positive (label = 1) or negative (label=0). In other words, it is a binary classification problem. Two models are implemented in Tensorflow python API r1.3: A simple feed-forward neural network classifier and a model which is a combination of a linear and a feed-forward neural network classifier.

3.5.1 Data preprocessing

A major issue that had to be tackled first was finding a way of transforming the candidate’s and job’s features, so that they would be suitable to be used as inputs. Therefore, a certain data preprocessing had to be done. In machine learning problems, input features must be transformed into numbers so the machine can make the necessary computations. In most ordinary and simple cases, features have already a numerical form (person’s age, income, price etc.), but categorical types are also frequent (marital status, nationality, hair color etc.). Thus, the problem of transforming categorical type into a numerical type is quite common. Two of the most used methods is the label encoding and the one-hot encoding. In label encoding each category is represented by an integer number. In the case of the one-hot encoding, each category is represented by a vector of the same size as the number of the categories and has the value one in the position that corresponds to the certain category and zeros to the rest positions.

Returning to the project case, a potential approach would be to treat the 5 input features (skills, experience, education, title, keywords) as categorical variables. But, these features consist of a various number of keywords, so a different approach had to be taken. To this direction, a method that originates from the text analytics field was adopted in order to resolve this problem of variant input length.

The overall concept that lies behind this approach is that the current machine learning problem could be treated like a special case of document classification, where there are five different texts, instead of one, as inputs. But these five texts are actually lists of keywords where the order of ‘words’ is playing no role, which is not the case in texts with natural language where order of the words has an impact in meaning and follows grammar and syntax rules. Another difference between a list of keywords and a text, is that a keyword may appear only once in a list.
Chapter 3: Machine Learning implementation

Considering the above-mentioned facts, we decide to adopt the simplest model of text representation, which is the bag of words [26] where the order of the words is ignored. Five different vocabularies should be created in order to encode the keywords of the five features. In text analytics a large corpus is needed for completing this task, but in our case, the provided dataset is used for this role. Certain minimum keywords frequencies are used for each feature vocabulary (Table 3-3).

These frequencies represent the minimum times a keyword must be encountered in the corpus in order to be stored in the vocabulary. The values of those frequencies originated from the experience of Workable’s existing recommender system. The final size of each vocabulary is shown in Table 3-4.

<table>
<thead>
<tr>
<th>Vocabulary size</th>
<th>Skills Vocabulary</th>
<th>Experience Vocabulary</th>
<th>Education Vocabulary</th>
<th>Title Vocabulary</th>
<th>Keywords Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>15341</td>
<td>3798</td>
<td>14095</td>
<td>4780</td>
<td>7173</td>
<td></td>
</tr>
</tbody>
</table>

Table 3-4 Vocabulary sizes

These five vocabularies are used to transform those lists of keywords into list of numbers and vice versa. Overall, a vocabulary performs a mapping between a word and a unique word id which is basically a label encoding procedure. If a word is not included in the vocabulary, then it returns a word id that corresponds to a special token of 'unknown word'.

3.5.2 The Deep Model

The first model that was implemented was a Feed Forward Neural Network with one ReLU layer and a sigmoid output layer which is used for binary classification. Each of the five bag of words (BOWs) inputs first is converted into a list of low-dimensional and dense real-valued vectors, which are also known as embedding vectors. Those vectors are initialized randomly and then are trained to minimize the final loss function during the model training. As a result, after this conversion, there are five lists of embedding vectors, one for each category. The dimensions of each embedding vector per category are shown in the Table 3-5. The dimension was calculated using the formula $d_{embedding} = \log_2(n)$, where n is the size

<table>
<thead>
<tr>
<th>Embedding size</th>
<th>Skills</th>
<th>Experience</th>
<th>Education</th>
<th>Title</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 3-5 Dimensions of embeddings vectors
Chapter 3: Machine Learning implementation

of the corresponding vocabulary.

Before they are fed to the hidden layer, these lists of vectors have to be aggregated into a single vector by using a function (e.g. sum, mean etc.), resulting five summarized vectors; one for each five categories. In this model, the function \( \sqrt[n]{\sum_{k=1}^{n} V_{k,d}} \), \( d \in [0, D] \)

\[
V'_{d} = \frac{1}{\sqrt{n}} \sum_{k=1}^{n} V_{k,d}, \quad d \in [0, D]
\]

where \( n \) is the total number of vectors in the list, \( D \) is the dimension of the embedding vector and \( V_{k,d} \) is
the d-th value of the k-th vector. This function was selected because it is claimed, in the Tensorflow’s documentation, that gives better results for cases where the feature vectors are BOWs.

Those 5 vectors are concatenated, resulting a 73-dimension vector which is fed to a rectified linear unit layer (ReLU) with 1000 dimensions. As activation function except from ReLU, we tried the Hyperbolic Tangent using a small dataset, but it did not perform well in evaluation scores. The output layer has two dimensions with a softmax function which returns the probabilities of each result. More specifically, the model prediction is:

\[ P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}_{\text{deep}}^T \mathbf{a}^{(lF)} + b) \quad Y \in \{0,1\} \]  \hspace{1cm} (3-2)

Where \( Y \) is the binary class label, \( \sigma(\cdot) \) is the sigmoid function, and \( \mathbf{w}_{\text{deep}} \) are the weights applied on the final activations \( \mathbf{a}^{(lF)} \). Those two probabilities are feed to a Max function for hard binary classification which returns 1 if the candidate is relevant with the current job position or 0 in is not. For loss function we use the Cross-Entropy function and for optimizer, the Adam Optimizer [27].

3.5.3 The Wide and Deep Model
The model, as its name suggest, has a wide and a deep component [28]. The wide component is a generalized linear model of the form \( y = \mathbf{w}^T \mathbf{x} + b \), where \( y \) is the prediction, \( \mathbf{x} = [x_1, x_2, ..., x_d] \) is a vector of \( d \) features, \( \mathbf{w} = [w_1, w_2, ..., w_d] \) are the model parameters and \( b \) is the bias. The feature set that we use in the wide component contains only transformed features. More specifically, it is a transformation which returns the crosses between the categories and is defined as:

\[ \varphi_k(\mathbf{x}) = \prod_{i=1}^d x_i^{c_{ki}} \quad c_{ki} \in \{0,1\} \]  \hspace{1cm} (3-3)

Where \( c_{ki} \) is a Boolean variable that is 1 if the \( i \)-th feature is part of the \( k \)-th transformation \( p_k \), and 0 otherwise. The total number of the transformed features derived from all the possible combinations between the candidate’s and the job’s features, so that their interactions can be captured. The resulting six crossed columns were:

1. skills_x_title
2. skills_x_keywords
3. experience_x_title
4. experience_x_keywords
5. education_x_title
6. education_x_keywords

The deep component of the model is the same feed forward neural network that is used in the previous model with the exception that the last layer with the sigmoid function is fed by both components. The combined model is illustrated in Figure 3-2.
The model's prediction is:

\[
P(Y = 1 | x) = \sigma (w_{\text{wide}}^T \varphi(x) + w_{\text{deep}}^T a^{(1)} + b) \quad Y \in \{0,1\} \tag{3-4}
\]

Where \( Y \) is the binary class label, \( \sigma(\cdot) \) is the sigmoid function, \( \varphi(x) \) are the cross-product transformations of the original features \( x \), and \( b \) is the bias term. \( w_{\text{wide}} \) is the vector of all wide model weights, and \( w_{\text{deep}} \) are the weights applied on the final activations \( a^{(1)} \). Those two probabilities, as in the Deep model, are feed to a Max function for hard binary classification.
Chapter 3: Machine Learning implementation

The two components are jointly trained by backpropagating the gradients from the output to both the wide and deep part of the model simultaneously using mini-batch stochastic optimization. For the wide component we use the Follow the Regularized Leader (FTRL) optimization algorithm [29] with $L_1$ regularization (although it is used the default value = 0), and for the deep part we apply the Adam optimizer [27]. We select the FTRL optimizer because it is also used in the Google’s paper where the concept of Wide & Deep model is introduced [28]. Although in the same paper, it is used the Adagrad optimizer for the deep component, in our case we choose the Adam optimizer because after testing the two methods with a small dataset, the latter has given better results in terms of convergence. As a loss function, similarly with the first model, we use the Cross-Entropy function which is fed with the weighted sum of the two components.

### 3.5.4 Remarks on Models implementation in Python Tensorflow API

Because of the thesis’ elaboration strict timetable, we make use of high-level Tensorflow API functions [30]. These functions are very suitable for rapid implementation and easy experimentation. For the Deep Model, we use the generic `tf.Estimator` class with a custom model function (Table 3-6).

```python
def deep_fn(features, labels, mode, params):
    # Bag of words columns with integer words ids
    bow_column_skills = tf.feature_column.categorical_column_with_identity(
        "skills", num_buckets=n_skills)
    bow_column_exp = tf.feature_column.categorical_column_with_identity(
        "experience", num_buckets=n_experience)
    bow_column_title = tf.feature_column.categorical_column_with_identity(
        "title", num_buckets=n_title)
    bow_column_kwd = tf.feature_column.categorical_column_with_identity(
        "keywords", num_buckets=n_keywords)

    # columns with embedding of keywords
    with tf.name_scope("Embeddings_input"):
        bow_embedding_column_skills = tf.feature_column.embedding_column(
            bow_column_skills, dimension=EMBEDDING_SIZE_SKILLS, combiner="sqrtn")
        bow_embedding_column_exp = tf.feature_column.embedding_column(
            bow_column_exp, dimension=EMBEDDING_SIZE_EXP, combiner="sqrtn")
        bow_embedding_column_title = tf.feature_column.embedding_column(
            bow_column_title, dimension=EMBEDDING_SIZE_TITLE, combiner="sqrtn")
        bow_embedding_column_kwd = tf.feature_column.embedding_column(
            bow_column_kwd, dimension=EMBEDDING_SIZE_KWORDS, combiner="sqrtn")

    bow = tf.feature_column.input_layer(
        features,
        feature_columns=(bow_embedding_column_skills, bow_embedding_column_exp, 
        bow_embedding_column_title, bow_embedding_column_kwd))

    first_hidden_layer = tf.layers.dense(bow, 1000, activation=tf.nn.relu, 
        name="hiddel_layer1")

    logits = tf.layers.dense(first_hidden_layer, MAX_LABEL, activation=None)

    return estimator_spec_for_softmax_classification( 
        logits=logits, labels=labels, mode=mode, params=params)
```

Table 3-6 The model function of the Deep Model
Chapter 3: Machine Learning implementation

For the Wide and Deep model we use the canned Estimator Classifier `tf.estimator.DNNLinearCombinedClassifier`.

### 3.6 Training and Experiment Results

Both models were trained by using the same dataset. Due to the fact that the keyword embeddings for each category are not available, it is necessary to be trained too. This is one of the major differences between the current Workable’s implantation, where the whole process has two parts; the first part trains the keyword embeddings by using the `word2vec` [31] algorithm, while the second part is a Feed-Forward Neural Network Classifier which utilizes those embeddings.

The initial dataset is divided to two parts; 80% of the dataset is used as training dataset and 20% is held as testing data. The Deep model is trained in 6500 steps, with a mini-batch size of 1,500 examples per batch, while the Wide & Deep model is trained in 2,000 steps with the same mini-batch size.

The parameters that used for the Adam optimizer in both models were:

- Learning rate = 0.01
- $b_1 = 0.9$
- $b_2 = 0.99$
- $\epsilon = 10^{-8}$

The parameters that used for the FTRL in the wide component of the second model were:

- Learning rate = 0.01
- $L_1 = 0$

The evaluation scores for both models are presented in Table 3-7. The Deep model was also trained in a Workable’s server with a bigger dataset and the results after 10,000 steps is almost identical with the ones presented here. It is evident that the Wide & Deep model doesn’t have better performance from the simple FWNN model in concern of Precision and Recall, and is ten times slower than the Deep model.

Considered the fact that interviewing a candidate who is not relevant with the job position is more expensive than not interviewing a relevant candidate, we assume that the precision of the class “1” is more important than its recall. Thus, using a threshold value instead of a Max function for hard classification we can probably achieve higher precision at the expense of lower recall.

From the aspect of getting good quality keyword embeddings, the results are ambiguous. Tensorboard, which is Tensorflow’s visualization tool, has a very easy way to represent 3d graphs of the trained embeddings vectors. It has also the ability to calculate the nearest neighbors of a selected embedding vector, by using as measure either the cosine or the Euclidian distance. In order to test the coherence of the embeddings, some keywords like (“Cook”, “Sales”) were picked manually and by applying a filter, ten of the
closest embedding vectors were selected (Figure 3-3). In most cases, the keywords that correspond to those vectors are irrelevant to the one that has been selected.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DEEP</th>
<th>WIDE N DEEP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“1”</td>
<td>“0”</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td>Precision</td>
<td>0.64</td>
<td>0.76</td>
</tr>
<tr>
<td>Recall</td>
<td>0.62</td>
<td>0.89</td>
</tr>
<tr>
<td>F1</td>
<td>0.63</td>
<td>0.88</td>
</tr>
<tr>
<td>AUC_ROC</td>
<td>0.75</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Table 3-7 The evaluation scores for each model

This is a concerning fact that need to be examined further in the future. At this point, it is necessary to make a comment about the mechanism with which the embeddings vectors are trained in word2vec and in this particular case. Roughly speaking, in the word2vec algorithm, the meanings of two words are closer if those two words happen to be in the same sentence frequently. Approximately, the model is trained by feeding it with a very large corpus and thus numerous sentences are provided as training examples. If there is a sentence where those two words exist, the algorithm attempts to correct the corresponding embedding vectors by bringing them closer.

In the current case, the “sentences” are the five lists of keywords; the candidate’s skills, experience and education and the title and keywords for the job position. It is highly probable in case of single training
example, that the keywords describing the candidate are correlated. This is also the case for the keywords
describing the job position. But, only in the case where the candidate is relevant to the job position the
algorithm attempts to bring closer the embeddings vectors. If a training example has a negative outcome,
then those keywords although may be related to each other, the algorithm attempts to separate those vec-
tors in order to minimize the cost function.

Therefore, considering also the fact that the negative examples of the dataset was three times more fre-
quent than the positive ones, the poor performance on producing good quality keyword embeddings
could be interpreted by the above explanation.
4 Conclusions

Taking into account the results of the evaluation of the two models, it is clear that the Wide & Deep model doesn’t have better performance from the simple FFNN model, however, there is room for further experimentation and improvement. Because of the strict timetable, no hyperparameter tuning has been performed. It is highly probable that the performance of both models would be further improved. Those hyperparameters are listed below:

- Hidden layer size
- Number of hidden layers
- Activation function of the hidden layers
- Embedding keywords sizes
- Regularization parameters for the optimization functions

In addition, it is quite possible to achieve further improvement for the Wide & Deep model, if only use a subset of the six crossed columns. Also, there is high probability of improving the performance of both classifiers, if we use a threshold value for hard binary classification, instead of using a Max function. This threshold value can be estimated with a cross-validation procedure.

In the case of concurrent training of keyword embeddings, for the reasons that are analyzed in the previous chapter, a much larger and more balanced training set might be needed for better results.

Finally, for more direct comparison between the performance of the models that have already implemented in Pytorch and the currently proposed ones, the use of the same pre-trained keyword embeddings would be preferable.

As regard Tensorflow, it is a very powerful machine learning software which is ideal for rapid deployment, while models can be deployed easily on a variety of devices. Although it comes at the cost of a much steeper learning curve and solid understanding of machine learning and mathematical concepts especially linear algebra and calculus. Also, in terms of programmability, the syntax is not as straightforward as PyTorch. On the other hand, Tensorflow has a very simple way of managing the execution devices (CPUs, GPUs, etc), while Tensorboard is a very easy-to-use visualization tool.
5 References


Chapter 5: References


Chapter 5: References


Chapter 5: References

predictive-modeling/.


Appendix

Github link for the Deep model:

https://github.com/iosifsp/MSs_DataScience-Thesis-project/deep.py

Github link for the Wide & Deep model:

https://github.com/iosifsp/MSs_DataScience-Thesis-project/widedeep.py