

Team Members:

Visheshwar Pratap Singh

Rohit Samant

Mehak Shamsi

Profile Matching and Recommendation for Recruiter

**Capstone Project Report**

April – July 2019

PGP Data Science, Business Analytics and Big Data

DF1809-CM

**ABSTRACT**

We present an evaluation of machine learning algorithms on a model prepared by us for improving the recruitment processes of organizations. The recruitment of candidates, being an important process for any organization, entails the hiring of employees that would be best fit for the job and ultimately beneficial for them. We have taken resumes of candidates of an organization and extracted the attributes (namely academics, qualifications, etc. to name a few) and assessed them according to a scale and a corresponding scoring system to train our system so that the candidates with the best scores can be shortlisted. We applied algorithms like decision tree, KNN, Content-Based Filtering, Collaborative Filtering.

**Problem Statement**

Which resume is the perfect match for our requirements?

Ranking the resumes that make the best fit to our Job Description on the basis of skills (Best to worst). Recommending the best candidates for the Job.

**Introduction/Abstract**

It is a huge task for recruiters to go through all the job applicants profiles manually and choose the ones which suit their requirements the best. Our application will simplify this task, thus, making talent acquisition much more efficient. It will assess the resumes and look for skills to match with the recruiter’s required skills for the job.

**Objective**

The general objective is to build a desktop application that will evaluate the resumes on the basis of skills required and skills present in the resume, and rank them accordingly. It performs the following tasks:

* Accept the skills required as input.
* Parse through the resumes to match the skills with the Job Description.
* Rank and Recommend Top 10 resumes from most suitable to least.

**Proposed System**

The main objective is to build a desktop application that will evaluate and rank the resumes on the basis of skills required, thus reducing the duration of the selection process, making recruitment simpler and more efficient. To make available the right person for the right job. We have currently used PDFMiner library to parse the resumes, we wish to use NER tagging in the future that help improve the model.

**Data Gathering.**

We collected the data manually from various people by asking out for the resumes. We collected resumes of different types of roles and fields, like IT, Engineering, Commerce, Finance, Banking, MBA etc.

In all we collected around 1400 resumes. Note all the resumes were in the PDF format, as it helped us for better processing.

**Processing and Working.**

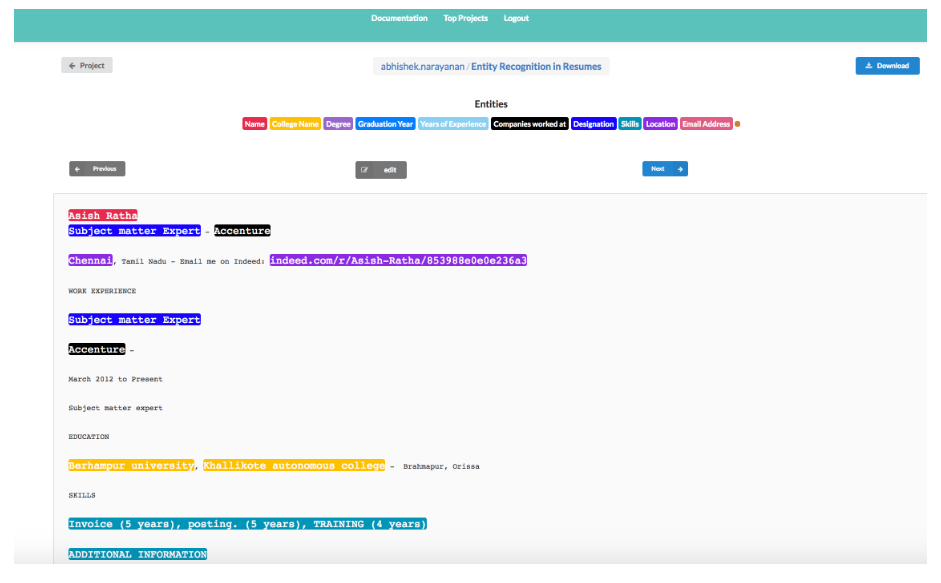
So for faster evaluation of the resumes we used NER(Named Entity Recognition) technique. Thereby helping us to shortlist the candidates among a pile of resumes.

**NER (Named Entity Recognition)**

Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a sub-task of information extraction that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

NER systems have been created that use linguistic grammar-based techniques as well as statistical models such as machine learning. Hand-crafted grammar-based systems typically obtain better precision, but at the cost of lower recall and months of work by experienced computational linguists.

**How is the Annotation done? Or how the tagging is done?**



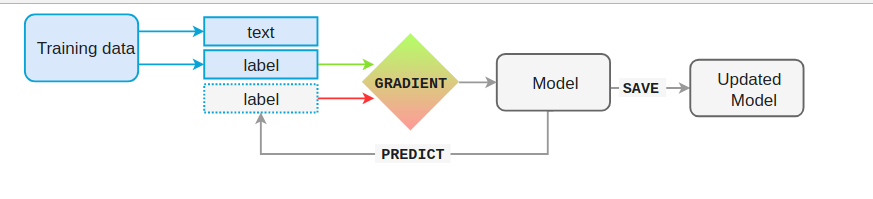
After this we get a JSON file which is then used for the further processing.

**Training the Model:**

We use python's spaCy module for training the NER model. spaCy's models are statistical and every "decision" they make - for example, which part-of-speech tag to assign, or whether a word is a named entity - is a prediction. This prediction is based on the examples the model has seen during training.

The model is then shown the unlabelled text and will make a prediction. Because we know the correct answer, we can give the model feedback on its prediction in the form of an error gradient of the loss function that calculates the difference between the training example and the expected output. The greater the difference, the more significant the gradient and the updates to our model.

When training a model, we don't just want it to memorize our examples - we want it to come up with a theory that can be generalized across other examples. After all, we don't just want the model to learn that this one instance of "Amazon" right here is a company - we want it to learn that "Amazon", in contexts like this, is most likely a company. In order to tune the accuracy, we process our training examples in batches, and experiment with minibatch sizes and dropout rates.



Of course, it's not enough to only show a model a single example once. Especially if you only have few examples, you'll want to train for a number of iterations. At each iteration, the training data is shuffled to ensure the model doesn't make any generalizations based on the order of examples.

**Testing the Model.**

We tested the model on 20 resumes and predicted the summarized resumes.

**Results we got:**

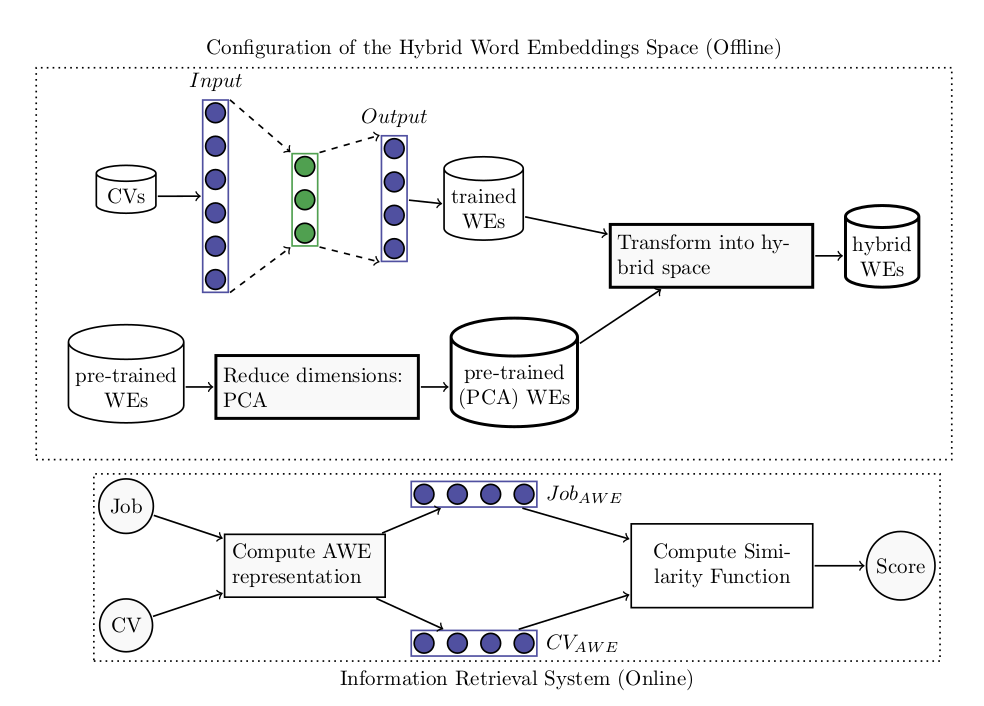
| Recognized Entity | Precision | Recall | F-Score |
| --- | --- | --- | --- |
| College Name | 100% | 100% | 100% |
| Location | 99.28% | 99.27% | 99.27% |
| Designation | 100% | 98.785 | 99.395 |
| Email Address | 100% | 99.43% | 99.71% |
| Name | 97.83% | 97.83% | 97.83% |
| Skills | 94.30% | 98.40% | 96.32% |

## **Architecture Description**

Information retrieval (IR) models are composed of an indexed corpus and a scoring or ranking function. The main goal of an IR system is to retrieve relevant documents or web pages based on a user request. During the retrieval, the scoring function is used to sort the retrieved documents according to their relevance to the user query. The classic IR models such as BM25 and language models are based on the bag-of-words (BOW) indexing scheme. BOW models have two major weaknesses: they lose the context where a word appears and they also ignore its semantics. Latent semantic indexing (LSI) is a technique used to handle this problem but when the number of documents increases, the process of indexing becomes computationally expensive. The standard technique used to overcome this is to train word or paragraph embeddings over a corpus or use pre-trained embeddings.

Word embeddings (WE) are distributed representations of terms obtained from a neural network model. These continuous representations have been used recently in different natural language processing tasks. The average word embeddings (AWE) is a popular technique to represent long sequences of text, not just a term.

In our case, a set of CVs is available, but job descriptions are priorly unknown and we need to provide a solution based on an unsupervised learning approach. Thus, word embeddings seem to be a good starting point for our experiments.Architcture is defined in the next figure.



**Working:**

As a first step, we build a balanced corpus of CVs from four known job profiles: Java, Tester, SAP HCM, and SAP SD. As CVs are written in several formats and with different styles and vocabulary, we decide to use only nouns and verbs in order to obtain just the important and relevant information from the CV. Once the corpus is built, we pass it through Word2vec with the configuration parameters as follows: window size is 5, minimum word count is 3, and dimensions are 200. CBOW is the default Word2vec model used.

We use Python 3.6.1 with Anaconda 64 bit for Linux Ubuntu 16.04 LTS. To install several libraries, the pip install command must be run as follows:

pip install gensim

pip install pattern3

pip install textract

pip install numpy

pip install scipy

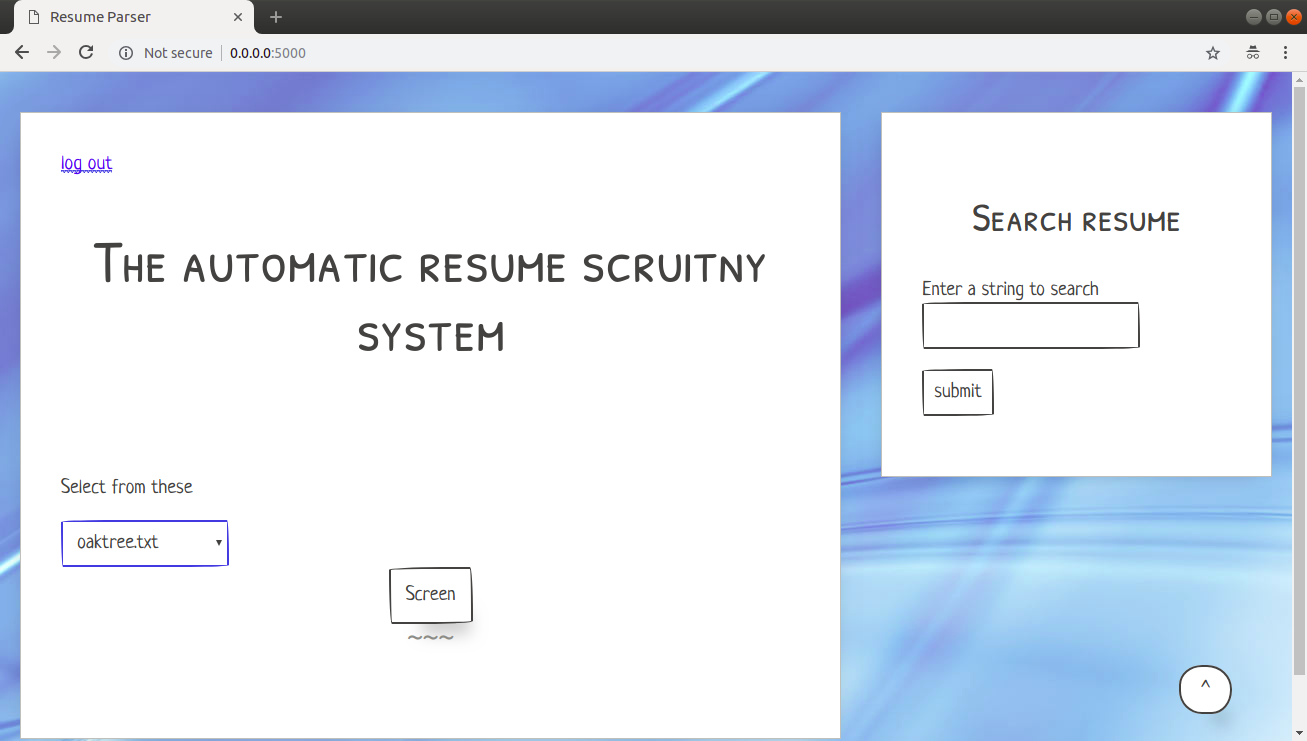
pip install sklearn

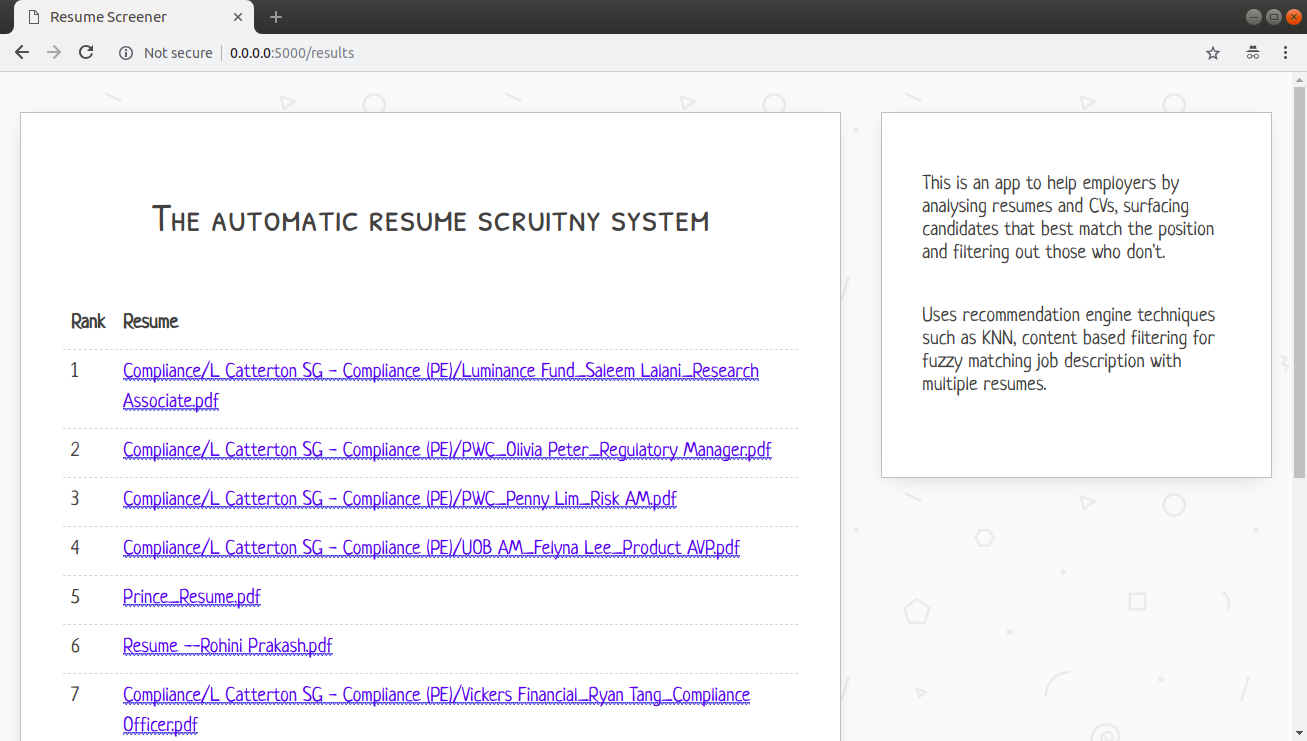
pip install pickle

After all needed packages are installed, we create a function to retrieve all CVs from a specific folder, read them (using textract), lemmatize them (using pattern3), and finally create the word embeddings (using gensim). The python function responsible for extracting the text from CVs (PDF, TXT, DOC, DOCX) is defined as follows:

**Deployment.**

The application made for the deployment of the project was on Flask. The snapshots of the same are:





**Challenges Faced.**

* Data Collection issues due to privacy.
* Annotations were very time consuming, hence we had to use alternative PDFMiner.
* Too many errors while setting environment due to PDFminer and Textract libraries.

**Conclusion.**

We implemented a flask application making it easy for the Recruiter to get best to worst candidates for the required job description.

Using modules like PDFMiner and Textract to parse through the resumes and match them with the JD’s using Knn algorithm.

**Reference.**

[**https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/**](https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/)