Automated Resume Screening

Shawon I. Kamal

Department of Computer Science, Memorial University of Newfoudland

COMP 4750: Natural Language Processing

Dr. Todd Wareham

December 8, 2021

Automated Resume Screening

To find the best candidate for the company while competing with others and trying to fill vacancies in a short period of time, today's recruiters need to consider innovative ways to reach professionals faster. This report introduces a prototype of a system that automatically extracts information from resumes and evaluates them based on the given job position. The main goal of this project is not to build the best automated resume screening application but to build a prototype which allows us to explore how to approach this problem using Natural Language Processing and Machine Learning techniques, understand the areas of difficulties which may arise and why this has been a difficult problem to solve till now. The application will consist of two parts: parsing and ranking of resumes.

Background

Large enterprise companies receive hundreds of CVs from applicants everyday (Kopparapu, n.d.). Among these a majority of candidates tends to be unqualified - upto 88% as revealed by the statistics from Glassdoor. If companies waste time manually searching for potential candidates in this large pool of candidates, they risk losing candidates to companies that use automated screening methods (Irina, 2019).

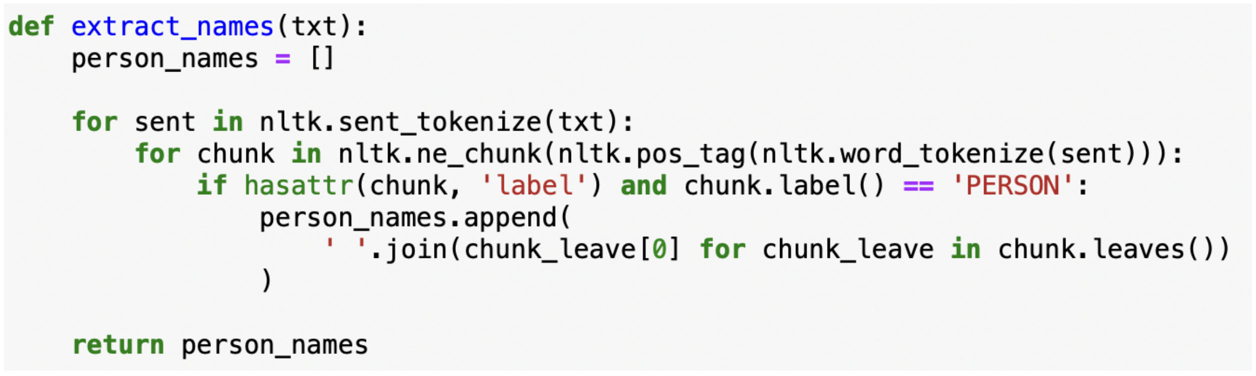
Recently, many recruitment tools have emerged to automate resume screening. However, most tools still struggle with processing text and matching candidates with the job requirements (Chen et al., 2018). Resumes are unstructured documents which can be created in many different formats which becomes very weary to process, select appropriate ones and store them in a unified database (Nimbekar et al., n.d.) This has led to the attention of researchers to come up with various methods to solve this problem.

Parsing

Resume parsing involves extracting relevant information from unstructured documents using linguistic analysis or pattern recognition techniques (Sinha et al., 2021). Resumes consist of many information like experience, education, personal information, skills, etc. Each resume is considerably different in these informations and overall structure (Nimbekar et al., n.d.) Working with such varied and ambiguous human language is very difficult.

Our prototype will work on unstructured resumes. The method used is called Entity Name Recognition (NER). NER is the method of converting unstructured resume text into structured text by classifying the noun-pronouns into separate categories such as persons name, addresses, phone numbers, education, experience, skills etc. (Nimbekar et al., n.d.)

I. Name Extraction



```
def extract_names(txt):
    person_names = []

    for sent in nltk.sent_tokenize(txt):
        for chunk in nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent))):
            if hasattr(chunk, 'label') and chunk.label() == 'PERSON':
                person_names.append(
                    ' '.join(chunk_leave[0] for chunk_leave in chunk.leaves())
                )

    return person_names
```

Fig 1. Extract name using NER model

Fig 1. shows a snippet of a NER model built with helper functions of python's nltk package. The first step for NER is tokenization which is the process of splitting raw strings into a list of meaningful substrings (Hardeniya et al., 2016). *sent_tokenize* splits text into sentences and *word_tokenize* similarly splits sentences into words.

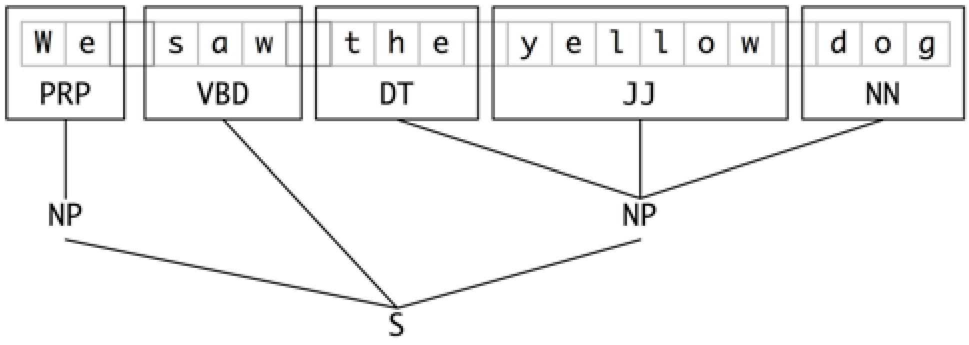


Fig 2. POS Tagging and chunking

The next step is to set a part-of-speech (POS) to each of these tokenized words generated from the first step. Once we tag the POS, *ne_chunk* uses Context-Free Grammar and Probabilistic Context-Free Grammar to club some chunks of the sentences that constitute grammar and meaning. *ne_chunk* also has a built-in trained classifier model that can recognize person names and organizations. However, it is not the best, and re-training the model with new datasets is advised.

II. Qualification Extraction

An easier yet effective approach to extracting qualifications from a resume is by scanning it to identify the presence of keywords from a reference data file. The knowledge-base will consist of all possible morphological forms of job titles and education background (Kopparapu, n.d.) These data files are built from archives of resumes contributed by both automatic methods and by person for validation.

III. Email Address and Phone Number Extraction

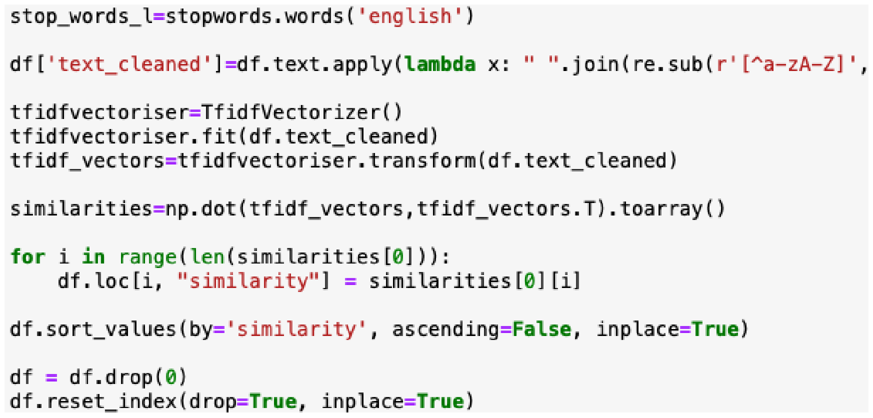
Other entity extractions such as email addresses and phone numbers can be extracted more easily by the help of regular expressions (Kopparapu, n.d.) Patterns such as "@" and (XXX) XXX-XXXX can be retrieved from resume. This process determines all email addresses or phone numbers, hence a post processing is required to pick the most probable entity and avoid the likes of contact of references (Kopparapu, n.d.)

Ranking

The final part of the application is ranking the resumes in the order of most qualified first. This will help the recruiters in making wise decisions and create an improved recruitment process in a shorter span of time (Nimbekar et al., n.d.)

One method of ranking resumes is to build an applicant learning model which requires training data to learn the algorithm for ranking. Given enough data from previous ranking decisions, it will eventually learn to rank resumes in order (Nimbekar et al., n.d.)

However, for our application we will be using an unsupervised method due to lack of dataset. The process is to rank the resumes by its relevance to the job description or requirements which is a process called Information Retrieval.



```python
stop_words_l=stopwords.words('english')

df['text_cleaned']=df.text.apply(lambda x: " ".join(re.sub(r'[^a-zA-Z]',

tfidfvectoriser=TfidfVectorizer()
tfidfvectoriser.fit(df.text_cleaned)
tfidf_vectors=tfidfvectoriser.transform(df.text_cleaned)

similarities=np.dot(tfidf_vectors,tfidf_vectors.T).toarray()

for i in range(len(similarities[0])):
    df.loc[i, "similarity"] = similarities[0][i]

df.sort_values(by='similarity', ascending=False, inplace=True)

df = df.drop(0)
df.reset_index(drop=True, inplace=True)
```

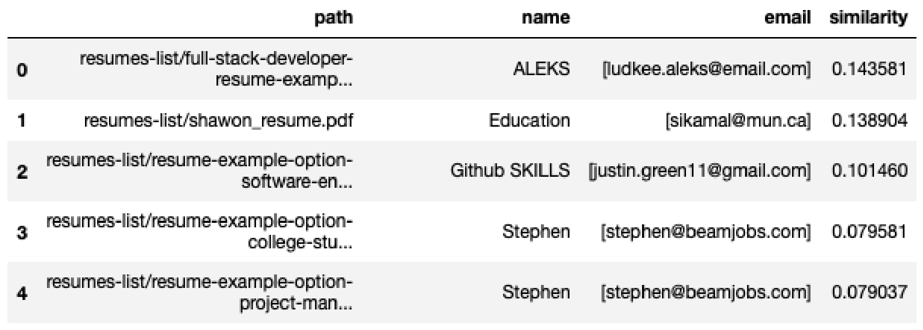
Fig 3. Calculate similarity between job description and resumes

In order for computers to understand the similarity between resume and job description, we need to mathematically define how these documents should compare (Nishimura, 2020). This brings to the first step of converting the texts into vectors. As seen in Fig 3, preprocessing is required before we do the conversion such as removing any stop words and punctuations from text. Stop words are defined as commonly used words such as “the”, “a”, “an” etc. that can interfere with the calculation. Using *TfidfVectorizer()* from the scikit-learn package, we convert the cleaned text to vectors of Term Frequency - Inverse Document Frequency (TF-IDF) features. TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

$$cos\,(\Theta) \;=\; \frac{A \cdot B}{|A|\,|B|}$$

Fig 4. Cosine similarity

The next step is to calculate the similarity of the vector of job description and resume. We do that using cosine similarity as seen in Fig 4, where we take the dot product of the resume vector and the job description vector. In our system, we get an output as shown in Fig 5 while using a job description of Verafin's software developer position.



	path	name	email	similarity
0	resumes-list/full-stack-developer-resume-examp...	ALEKS	[ludkee.aleks@email.com]	0.143581
1	resumes-list/shawon_resume.pdf	Education	[sikamal@mun.ca]	0.138904
2	resumes-list/resume-example-option-software-en...	Github SKILLS	[justin.green11@gmail.com]	0.101460
3	resumes-list/resume-example-option-college-stu...	Stephen	[stephen@beamjobs.com]	0.079581
4	resumes-list/resume-example-option-project-man...	Stephen	[stephen@beamjobs.com]	0.079037

Fig 5. Ranking output

Conclusion

To sum up, this application does an average job of parsing and screening resumes. There are several areas where the parsing could be improved. Currently the NER model is not perfect, candidate's names are not picked correctly in a few resumes. The lack of successfully parsed information disallowed from building a good evaluation module. Additionally, there are a few areas where evaluation could be independently improved such as by retrieving information from linkedin, or relevant social media, by verifying correctness etc. Regardless, this is a good start in recognizing the points of failures in NLP and resume screening which inspires us to research on improvements.

References

Chen, J., Niu, Z., & Zhang, J. (2018). A Two-Step Resume Information Extraction Algorithm. *Hindawi - Mathematical Problems in Engineering*.

Hardeniya, N. et al. (2016). Natural Language Processing: Python and NLTK. *Packt Publishing Ltd.*

Irina, M. (2019). Screening the Candidates in IT Field Based on Semantic Web Technologies: Automatic Extraction of Technical Competencies from Unstructured Resumes. *Bucharest University of Economics Studies*.

Kopparapu, K. (n.d.). Automatic Extraction of Usable Information from Unstructured Resumes to Aid Search. *TCS Innovation Labs*.

Nimbekar, R., Patil, Y., Mulla, S., & Prabhu, R. (n.d.). Automated Resume Evaluation System using NLP. *Don Bosco Institute of Technology.*

Sinha, A., Kumar, A., & Akhtar, A. (2021). Resume Screening Using Natural Language Processing and Machine Learning: A Systematic Review. *Springer Nature Singapore Pte Ltd*.

Nishimura (2020). The Best Document Similarity Algorithm: A Beginner's Guide. *https://towardsdatascience.com/the-best-document-similarity-algorithm-in-2020-a-beginners-guide-a01b9ef8cf05*