Resume Parser

Jasmit Mahajan jsmahajan@cpp.edu
Rahul Nagarajan rnagarajan@cpp.edu
Malemsana Thokchom mthokchom@cpp.edu
Aarmin Alipour aalipour@cpp.edu

Abstract

Shortlisting resumes is a time-consuming process. HR professionals today struggle to find the right candidate for applied job roles. These professionals shift through thousands of resumes to filter out qualified candidates for the job. If the process of shortlisting resumes can be automated, it will make the hiring process easier. This is where our straightforward but effective solution comes in. We propose a custom NLP model that uses Named Entity Recognition to auto-detect entities in a resume in pdf or docx format towards shortlisting candidates for different job roles.

1 Introduction

A resume is a document that an applicant submits as their first step in applying for any job. It is also an important part of the candidate application because it briefly summarizes the candidate's qualifications. In general, resumes include important information such as skillset, professional experience, accomplishments, and hobbies. The resume is the foundation of any job application and one of the most important ways to make a good first impression towards a recruiter. However, not all resumes are well written, as many applicants submit resumes that are very verbose and ponderous, taking the recruiter a long time to go through and skim the important information. Because there are no standard rules for designing and writing resumes, poorly crafted resumes are less appealing, difficult to understand and is harder to catch the recruiter's attention to pick out important points from a candidate's profile.

Jobs today receive an abundance of resumes and are manually parsed by an HR professional. This is time-consuming and error-prone because an HR may overlook something important. As a result, our goal was to create a Resume Parser that could take an input resume (with popular extensions like pdf and docx) and extract job-relevant keywords like Name, Skill, Education, Experience, and so on, resulting in a short and concise version of the resume.

At a high-level, our project can be organized in four modules. The first module is responsible for importing the dataset. The second module handles data pre-processing. In the following module, we train our NER model for entity recognition and visualize the entities it generates. Our final module is dedicated to testing the model's accuracy.

2 Related Works

We have observed that Resume Parsing projects are typically hard-coded to work with a specific dataset. Such projects create their own dataset to match keywords such as skills, education, and so on. However, this is problematic if the keywords in the input resume do not match the hard-coded dataset, and thus the intended keyword is not parsed (Angelova 2022). Other approaches use the pyresparser library, which is a very simple approach and yet powerful at the same time as its uses multiple python libraries in conjunction (Suresh, 2022). One other approach is using Lexical Analysis, which the plain text input is segmented into words and paragraphs and then tokens are created, and Symantic Analysis, which the grammar and the arrangement of words in a meaningful manner are checked, to develop a Resume Parser system that users can put in various formats (Bhor, Gupta, Nair, Shinde, 2021).

Another popular approach is to use a custom NER model to parse a resume, which is the same approach as ours, although it uses a dataset that is already pre-processed to fit the model (Kant 2020). Our project stands out because we take on the challenge of using a different data set and pre-processing it ourselves. We adjusted the model's

hyper-parameters so that we get accurate results from training our convoluted dataset. We also use a different testing mechanism, having it tested using one of the resumes in the dataset as well as an unseen resume. We make the project more insightful and versatile (By displaying the entities recognized by the model and by having a function to deal with both pdf and docx files). We also make certain that our project is well documented and organized (via visualizations) so that it is easily understood even by a layman.

3 Method

We found that Named Entity Recognition is a viable solution to this problem because it can reliably extract the most important keywords from a resume. Named Entity Recognition (NER) is the concept of identifying named entities, such as 'Python,' 'Data Science,' 'California,' and categorizing them into specific categories, such as 'Skillset,' 'Location,' and so on. Using spaCy's custom NER model, we are able to accomplish our goals.

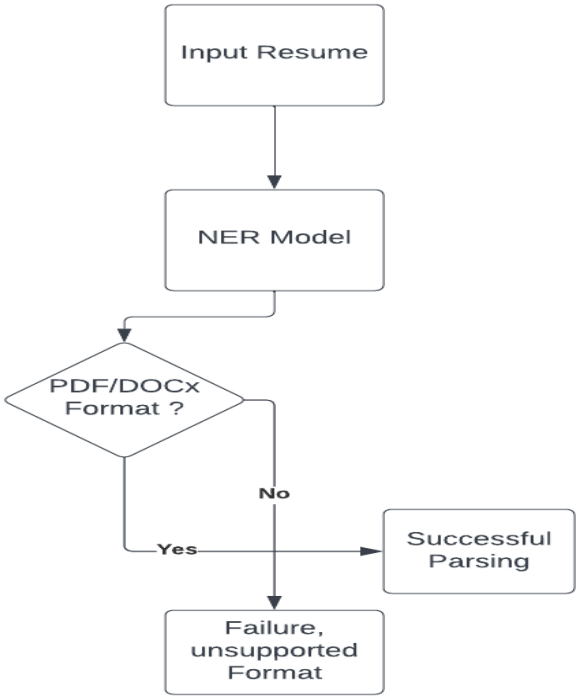

Figure 1: Project architecture

The architecture of our project is straightforward. The trained NER model will receive an input resume and process it. The format of the resume will be checked, and it will only be parsed if it is in pdf or docx format. In the following subsections, the project's modules will be thoroughly explained.

3.1 The Dataset

Finding a dataset was challenging as datasets for resume parsing projects are very scarce.We needed to make sure the dataset wasn't too sparse, because different resume formats could make our project impossible. We were fortunate enough to find a dataset on Kaggle. The dataset, authored by DataTurks contains 220 manually labeled Resumes, intended for use with the Named Entity Recognition technique. The labels in the dataset are divided into 10 following categories:

- Name
- College Name
- Degree
- Graduation Year
- Years of Experience
- Companies worked at
- Designation
- Skills
- Location
- Email Address

3.2 Dataset pre-processing

After importing the dataset, we begin by converting the JSON file to a pandas dataframe, completing the project's first module. In our second module, we begin by cleaning up the dataset's text, such as removing new lines and irrelevant symbols. We will then pre-process our dataset by structuring it in a way that the NER model will accept, which is the spacy BILOU (Beginning-Inside-Last-Outside-Unit) format/scheme. The latter is is essentially a list of tuples. Each tuple contains text and a dictionary. The key 'entities' in the dictionary will store the start and end indices, as well as the labels of entities found in the text. The following is a sample output of a chunk of the dataset after conversion:

```
{'entities': [(1901, 1905, 'Skills'),
  (1812, 1815, 'Graduation Year'),
  (1763, 1797, 'College Name'),
  (1756, 1760, 'Degree'),
```

```
  (803, 806, 'Graduation Year'),
  (653, 656, 'Graduation Year'),
  (645, 648, 'Graduation Year'),
  (469, 478, 'Companies worked at'),
  (461, 466, 'Designation'),
  (403, 406, 'Location'),
  (54, 97, 'Email Address'),
  (14, 17, 'Location'),
  (0, 12, 'Name')]}
```

3.3 Training the model

The conversion of the dataset concludes our second module and now we proceed feeding it to our model. The creation of a custom NER model is offered by the SpaCy library, an open-source library for Natural Language Processing in Python. SpaCy includes a pipeline 'ner' for Named Entity recognition tasks. It performs well by default, but not always. It is error-prone in the sense that it can misidentify entities; for example, a company's name can be misidentified as a person's name. To avoid such problems, we will customize the NER model for our Resume parsing project. To do so, we would need to provide training samples (such as sample resumes), which would allow the model to learn for predictions.

We initiate by creating a blank NER model, and passing a language ID, which is 'en' or English. As the model does not have any pipeline component by default, we add 'ner' to the pipeline using the add_pipe() method. We proceed to add the entity label to the entity recognizer using the add_label method. Before we train the NER model, we must disable other pipleline components as we are only concerned with the NER pipeline. We finally loop over our dataset and call nlp.update which essentially iterates through the input resume(s).

```
if 'ner' not in nlp.pipe_names:
    ner = nlp.create_pipe('ner')
    nlp.add_pipe(ner, last=True)

for eachEntity in entities:
    elist = eachEntity['entities']
    for ent in range(len(elist)):
      label = elist[ent][2]
      label = str(label)
      ner.add_label(label)

nlp.update([sentences[temp]],
                    [entities[temp]],
                    drop = 0.2,
                    sgd = optimizer,
                    losses = losses )
```

At each word of the resume, it makes a prediction and seeks annotation for verification. If the prediction was wrong, the weights are adjusted accordingly such that a better score is achieved in the subsequent iterations.

```
Iteration 0:
Loss: {'ner': 10771.07683084369}
Iteration 1:
Loss: {'ner': 12394.833622767888}
Iteration 2:
Loss: {'ner': 13346.55944420256}
Iteration 3:
Loss: {'ner': 10055.561846235418}
Iteration 4:
Loss: {'ner': 10536.417472127132}
Iteration 5:
Loss: {'ner': 9268.727689689495}
Iteration 6:
Loss: {'ner': 7263.374293092989}
Iteration 7:
Loss: {'ner': 6481.302725220139}
Iteration 8:
Loss: {'ner': 8506.186095533074}
Iteration 9:
Loss: {'ner': 8805.119950339247}
```

As we can see, loss increased from iteration 0 to 3, then gradually decreased to 8805.11, which is a good sign because loss is directly proportional to bad prediction, and we want it to be as low as possible.

3.4 Testing the model

Finally, we use nlp.to disk to save the trained model and then test its accuracy. Before we begin the testing phase, we will ensure that our model was able to recognize the entities as intended by dataset labeling, as shown below:

```
10 Entities found:

['College Name',
 'Companies worked at',
 'Degree',
 'Designation',
 'Email Address',
 'Graduation Year',
 'Location',
 'Name',
 'Skills',
 'Years of Experience']
```

Following the verification, we proceed to the testing phase. This module is completed in two stages. We will first randomly select a resume from the dataset and use it to make a prediction, and then we will use an unseen resume to further test the model's ability to extract an unseen data source based on the 'resume-specific' keywords it has learned to recognize and extract. As our dataset is complex due to overlapping entities, we were unable to use spaCy's built-in function for accuracy scores, so we conducted a manual evaluation of the model's performance.

4 Results

We conducted a manual evaluation of our model by eyeballing the keywords extracted by the model on one of the resumes in the dataset and comparing the keywords extracted by the model to an 'unseen' external resume in either pdf or docx format. In our observations, the model performs better when tested with one of the resumes from the trained dataset than when tested with an unseen resume. For example, in one of our cases, the model was able to extract all keywords from one of the datasets, but it missed some keywords in our unseen resume, such as name, and misidentified the degree as college name. The results of testing the model with one resume from the dataset is shown below:

```
Name            :    Geeta
Degree          :    PGDM in Marketing
College name    :    Goa Institute
Degree          :    B.Tech in CS
Skills          :    Adobe Photoshop, BI
```

The results of testing the model with an unseen resume:

```
Designation     :    Software Engineer
Degree          :    IIT Mumbai
Skills          :    Machine Learning, NLP
Others          :    Problem Solving
```

Our model was initially inaccurate, so we had to adjust the hyperparameters to improve accuracy. We reasoned that our model was inaccurate due to the low number of initial iterations we used. After much trial and error, we determined that iterations in multiples of ten are ideal for our project; in this case, we stick to ten iterations. We also decreased our dropout rate from 0.5 to 0.2 because a higher value makes data memorization more difficult. We also thought it would be a good idea to fine-tune the model by randomly shuffling the resumes using the random.shuffle() function, to ensure that the model does not generalize based on the order of the resumes.

5 Conclusions and Future Work

Our model performs quite well when testing it with BILOU structured resume, and shows satisfactory results when testing it against an unseen resume. The latter's inaccuracy could be attributed to not being structured in accordance to SpaCy's BILOU scheme. Our future enhancements will be aimed at achieving this goal. In terms of future work, we want to completely restructure our project so that, given an input resume, it can summarize it in the manner of a real person. We discovered that abstractive summarization with transformers is a viable solution to this problem.

References

Angelova, V., 2022. Build your own Resume Parser Using Python and NLP. [online] API Layer. Available at: ¡https://blog.apilayer.com/build-your-own-resume-parser-using-python-and-nlp/¿ [Accessed 19 May 2022].

Suresh, V., 2022 RESUME PARSER-PYTHON https://vidhusuresh16325.medium.com/resume-parser-python-57bc8a81cce0. [Accessed 18 May 2022].

Kant, L., 2022. Resume and CV Summarization using Machine Learning in Python. [online] Medium. Available at: ¡https://kgptalkie.medium.com/resume-and-cv-summarization-using-machine-learning-in-python-394ea49fba3e¿ [Accessed 19 May 2022].

Bhor, S., Gupta, V., Nair, V., Shinde, H., 2021. Resume Parser Using Natural Language Processing Techniques. https://www.ijres.org/papers/Volume-9/Issue-6/Ser-8/A09060106.pdf [Accessed 18 May 2022].