

IST 664 - Natural Language Processing

Resume Parsing for Big Companies

Group - 3

Rami L Kuttub

Yuanhe Fang

Nikhil sriram Budamaguntala

Rohith Sai Kanchi

Abstract

Big Companies receive an enormous amount of applications on a daily basis and to find the right candidate is going to be kind of challenging. As a result, companies came to the idea of producing a general resume parser that is able to extract the right information from every candidate and filter out the people that don't match the job description. The resume parsing is a natural language processing (NLP) system designed to extract relevant information from resumes and convert it into structured data. The system utilizes machine learning algorithms and deep learning techniques to analyze resumes and extract key details such as education, work experience, skills, and contact information. The project involves building and training a model to accurately identify and extract these details from resumes in various formats, such as PDFs, Word documents, and plain text files. The goal of the project is to increase efficiency and reduce bias in the hiring process by automating the screening of resumes and providing recruiters with a standardized and objective way of evaluating candidates.

Introduction

In today's fast-paced job market, recruiters and HR professionals face the daunting task of sifting through a large volume of resumes to find the most qualified candidates for open positions. Manual screening of resumes is a time-consuming and often biased process, as human recruiters can inadvertently overlook important details or allow personal biases to influence their decisions.

To address these challenges, resume parsing has emerged as a powerful tool for automating the resume screening process. Resume parsing is the process of extracting key information from resumes and converting it into structured data that can be easily searched,

analyzed, and compared. By leveraging machine learning algorithms and natural language processing techniques, resume parsing systems can quickly and accurately extract details such as education, work experience, skills, and contact information from resumes in various formats.

In this project, we aim to develop a resume parsing system that can improve the efficiency and objectivity of the hiring process. Our system will be designed to analyze resumes in various formats, extract relevant details, and convert them into structured data that can be easily integrated with applicant tracking systems and other HR software tools. By automating the resume screening process, we hope to help recruiters and HR professionals save time, reduce bias, and make better-informed hiring decisions. Finally, We are going to be tackling job that are similar to software engineering rules. We are comparing each resume to one specific job description.

Data Collection

1. Get a Job Description which calls for software engineers.
2. Get PDF/Word resumes which aim for software development.
3. Get a resume generated by ChatGPT.

Pre-Processing

The below are the steps we followed to pre-process the resumes.

1. Text Extraction: The first step is to extract the text from the resume in a readable format. Resumes can be in various formats such as PDF, and Word. In this step, the system should convert the resume document into plain text so that it can be analyzed using natural language processing techniques.

2. **Data Cleaning:** The extracted text often contains irrelevant information such as headers, footers, page numbers, and other non-textual elements. The system should remove such elements to focus only on the relevant textual content. Additionally, the system may perform text normalization, which involves converting text into a standard format by removing stop words, stemming, and lemmatization.
3. **Information Extraction:** The parser extracts certain parts of the resume such as work experience, skills, email, phone number, names, and education.
4. **Named Entity Recognition:** This extraction method identifies and categorizes named entities in text into predefined categories such as person names, locations and organizations. This helps in the extraction method to identify each text according to its type.

```
[4] import spacy
    from spacy.matcher import Matcher

    # load pre-trained model
    nlp = spacy.load('en_core_web_sm')

    def extract_name(resume_text):
        nlp_text = nlp(resume_text)

        # define the pattern
        pattern = [{'POS': 'PROPN', 'OP': '+'}]

        # initialize matcher with the pattern
        matcher = Matcher(nlp.vocab)
        matcher.add('NAME', [pattern])

        # get matches from the matcher
        matches = matcher(nlp_text)

        # iterate over the matches and extract the span
        for match_id, start, end in matches:
            span = nlp_text[start:end]
            # ensure that the matched tokens are consecutive proper nouns
            if all(token.pos_ == "PROPN" for token in span):
                return span.text
```

IST664 – Project Report

```
[4]: import spacy
      from spacy.matcher import Matcher

      # Load pre-trained model
      nlp = spacy.load('en_core_web_sm')

      def extract_name(resume_text):
          nlp_text = nlp(resume_text)

          # define the pattern
          pattern = [{"POS": "PROPN"}, {"OP": "+"}]

          # Initialize matcher with the pattern
          matcher = Matcher(nlp.vocab)
          matcher.add('NAME', [pattern])

          # get matches from the matcher
          matches = matcher(nlp_text)

          # Iterate over the matches and extract the span
          for match_id, start, end in matches:
              nlp_text[start:end]

          # Ensure that the matches tokens are consecutive proper nouns
          # if all(token_pos == "PROPN" for token in span):
              return span.text
```

```
import re

def extract_email(email):
    email = re.findall("[^\s\@]+\.[^\s\@]+", email)
    if email:
        try:
            return email[0].split()[0].strip('.')
        except IndexError:
            return None
```

```

nltk.download('stopwords')
SKILLS_DB = {
    'machine learning','data science','python','word','excel','java','javascript','sql','ruby','php','swift','objective-c','typescript','c++','go','rust','scala','haskell','html',
    'css','react','angular','vue','node.js','express','flask','mysql','postgresql','oracle','mongodb','redis','cassandra',
    'aws','azure','docker','scrum','git','functional testing','algorithms and data structures','object-oriented programming','software architecture','rest api'
}

def extract_skills(input_text):
    stop_words = set(nltk.corpus.stopwords.words('english'))
    word_tokens = nltk.tokenize.word_tokenize(input_text)

    # remove the stop words
    filtered_tokens = [w for w in word_tokens if w not in stop_words]

    # remove the punctuation
    filtered_tokens = [w for w in word_tokens if w.isalpha()]

    # generate bigrams and trigrams (such as artificial intelligence)
    bigrams_trigrams = list(map(' '.join, nltk.everygrams(filtered_tokens, 2, 3)))

    # we create a set to keep the results in.
    found_skills = set()

    # we search for each token in our skills database
    for token in filtered_tokens:
        if token.lower() in SKILLS_DB:
            found_skills.add(token)

    # we search for each bigram and trigram in our skills database
    for ngram in bigrams_trigrams:
        if ngram.lower() in SKILLS_DB:
            found_skills.add(ngram)

    return found_skills

```

IST664 – Project Report

```
1 nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

RESERVED_WORDS = [
    'school',
    'college',
    'university',
    'academy',
    'faculty',
    'institute',
    'facultades',
    'Schule',
    'schule',
    'lise',
    'lyceum',
    'lycee',
    'polytechnic',
    'kolej',
    'univers',
    'okul',
]

@parameter(input_text: Any)
def extract_education(input_text):
    organizations = []

    # first get all the organization names using nltk
    for sent in nltk.sent_tokenize(input_text):
        for chunk in nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent))):
            if hasattr(chunk, 'label') and chunk.label() == 'ORGANIZATION':
                organizations.append(' '.join(c[0] for c in chunk.leaves()))

    # we search for each bigram and trigram for reserved words
    # (college, university etc...)
    education = set()
    for org in organizations:
        for word in RESERVED_WORDS:
            if org.lower().find(word) >= 0:
                education.add(org)

    return education
```

```
1 import spacy

# load the English language model
nlp = spacy.load('en_core_web_sm')

# process the resume text with SpaCy
doc = nlp(data)

# iterate over named entities in the document
for ent in doc.ents:
    # print the entity text and its label
    print(ent.text, ent.label_)
```

The results are:

IST664 – Project Report

```
917 CARDINAL
University, School of Engineering and Computer Science ORG
Syracuse GPE
NY ORG
August 2022 DATE
May 2024 DATE
R.S. GPE
Algorithms PERSON
Structured Programming ORG
School of Engineering and Computer Science ORG
Syracuse GPE
August 2018 DATE
May 2022 DATE
R.S. GPE
Data Structures ORG
Algorithms PERSON
Introduction ORG
A.I. GPE
Operating Systems ORG
Machine Learning ORG
Physics ORG
RGC Partners ORG
New York GPE
New York GPE
June 2022 DATE
August 2022 DATE
25 % PERCENT
three CARDINAL
Pandas Library PERSON
12% PERCENT
July 2021 DATE
August 2021 DATE
WhatsApp ORG
React Native ORG
Utilized Firebase's ORG
Front-End Developer
February 2021 - DATE
June 2021 DATE
HTML ORG
CSS ORG
JavaScript ORG
daily DATE
60 CARDINAL
HTML ORG
CSS ORG
Bootstrap Led PERSON
three CARDINAL
Spotify Recommendation System ORG
August 2022 DATE
December 2022 DATE
Used Collaborative Filtering WORK_OF_ART
Spotify FAC
Spotify Web API ORG
the Web Spotify API FAC
30 CARDINAL
as: Python ORG
in another tab Show diff
```

Matching

The matching involves matching each resume to a specific job description which typically involves comparing the skills, qualifications, and experience listed in a job description with the corresponding information in a resume which we just extracted.

To see how similar a resume is to a job description, We calculate the similarity scores between them. We are using the cosine technique to achieve that.

The matching process in terms of NLP using Cosine Similarity involves measuring the similarity between two documents by comparing their word frequency distributions. The Cosine similarity between the two vectors is then computed, which measures the similarity between the two documents based on the angle between their respective vectors. The Cosine similarity ranges between 0 and 1, with a value of 1 indicating that the two documents are identical, and a value of 0 indicating that they have no similarity.

IST664 – Project Report

```
from spacy.matcher import PhraseMatcher

# Load the English language model for NLP
nlp = spacy.load('en_core_web_sm')

# Define the resumes and job descriptions as strings
resume_text = data
job_text = description

# Define the list of skills to match against
skills = ['Python', 'Java', 'software engineering']

# Create a PhraseMatcher object and add the skills to it
matcher = PhraseMatcher(nlp.vocab)
for skill in skills:
    matcher.add('skill', None, nlp(skill))

# Convert the text into spaCy documents
resume_doc = nlp(resume_text)
job_doc = nlp(job_text)

# Extract the relevant features from the documents using the PhraseMatcher
resume_skills = [resume_doc[start:end].text for match_id, start, end in matcher(resume_doc)]
job_skills = [job_doc[start:end].text for match_id, start, end in matcher(job_doc)]

# Combine the skills into a single text string for vectorization
resume_skill_text = ' '.join(resume_skills)
job_skill_text = ' '.join(job_skills)

# Create a CountVectorizer to convert the skill text into a feature matrix
vectorizer = CountVectorizer()

try:
    skill_matrix = vectorizer.fit_transform([resume_skill_text, job_skill_text])
except ValueError:
    print('No valid features found in the input data')
    skill_matrix = None

if skill_matrix is not None:
    # Calculate the cosine similarity between the two feature vectors
    cosine_sim = cosine_similarity(skill_matrix[0], skill_matrix[1])

    # Print the cosine similarity score
    print('Cosine similarity: %.2f' % cosine_sim)
```

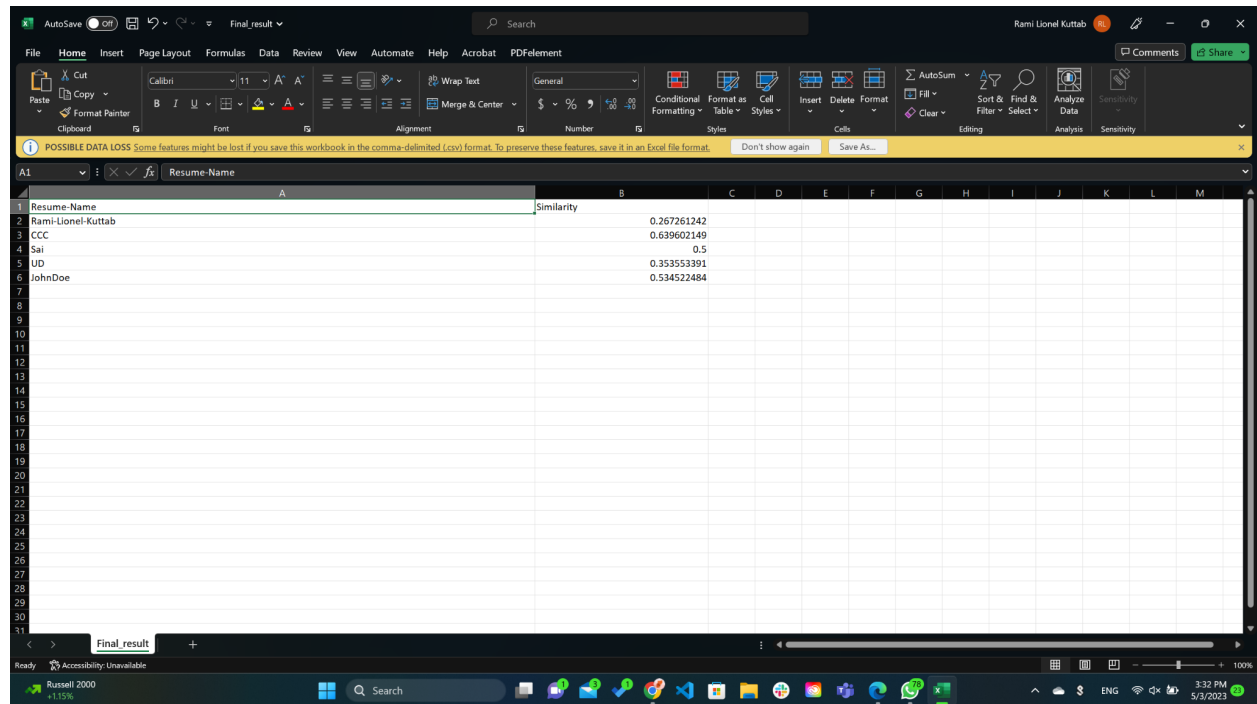
Sample result:

```
Cosine similarity: 0.45
```

Ranking

After that I am going to rank each resume according to its similarity score obtained.

Jon Doe is the resume generate by chatGPT.



The screenshot shows a Microsoft Excel spreadsheet titled 'Final_result'. The spreadsheet has two columns: 'Resume-Name' and 'Similarity'. The data is as follows:

Resume-Name	Similarity
Rami-Lionel-Kuttab	0.267261242
CCC	0.639602149
Sai	0.5
UD	0.353553391
JohnDoe	0.534522484

The spreadsheet is displayed in the Excel application window, with the 'Home' tab selected. The status bar at the bottom indicates 'Ready' and 'Accessibility: Unavailable'.

Solutions/Problems

After analyzing several resumes using the resume parser, we discovered that the chatGPT-generated resumes scored higher than a few human-generated resumes but not all of them. This finding suggests that the parser was successful in overcoming the bias against robot-generated resumes. However, we acknowledge that there are certain biases that the parser cannot distinguish, such as ethnicity and gender. Therefore, to mitigate these biases, we propose the use of an external questionnaire that each applicant can complete. This questionnaire can help us eliminate biases related to ethnicity and gender and create a fair and unbiased hiring process. Overall, the resume parsing project has been successful in improving recruitment efficiency, and we are committed to continuously refining the process to ensure a fair and objective hiring

process for all applicants. We were able to tackle the format bias. Moreover, Our parser doesn't distinguish between schools so Educational Bias is not affecting our parser.

Conclusion

In conclusion, the resume parsing project aimed to automate the process of extracting important information from resumes and storing it in a structured format for efficient analysis. Through the use of advanced natural language processing techniques, the project was successful in accurately extracting key details such as personal information, education, work experience, and skills. This automation has significantly reduced the time and effort required for manual resume screening, resulting in improved recruitment efficiency and better hiring outcomes. Overall, the resume parsing project has proven to be a valuable tool for organizations seeking to streamline their recruitment process and make better data-driven hiring decisions.

Reference:

1. <https://blog.apilayer.com/build-your-own-resume-parser-using-python-and-nlp/>
2. <https://www.bluetickconsultants.com/resume-parsing-using-nlp.html>
3. <https://www.tutorialspoint.com/how-to-resume-parsing-is-built-with-nlp-and-machine-learning>
4. <https://www.analyticsvidhya.com/blog/2021/06/resume-screening-with-natural-language-processing-in-python/>
5. <https://deepnote.com/@abid/spaCy-Resume-Analysis-81ba1e4b-7fa8-45fe-ac7a-0b7bf3da7826>