ISSN 2321 3361 © 2017 IJESC

Research Article Volume 7 Issue No. 2

Resume Parser with Natural Language Processing

Satyaki Sanyal[1], Souvik Hazra[2], Soumyashree Adhikary[3], Neelanjan Ghosh [4]
School of Electronics Engineering[1, 3]
School of Electrical Engineering[2, 4]
KIIT University, India

Abstract:
Parse information from a resume using natural language processing, find the keywords, cluster them onto sectors based on their keywords and lastly show the most relevant resume to the employer based on keyword matching. First, the user uploads a resume to the web platform. The parser parses all the necessary information from the resume and auto fills a form for the user to proofread. Once the user confirms, the resume is saved into our NoSQL database ready to show itself to the employers. Also, the user gets their resume in both JSON format and pdf.

Keywords: Resume parser, resume analyzer, text mining, natural language processing, resume JSON, semantic analysis

I. PROBLEM STATEMENT

To design a model this can parse information from unstructured resumes and transform it to a structured JSON format. Also, present the extracted resumes to the employer based on the job description.

II. INTRODUCTION

Corporate companies and recruitment agencies process numerous resumes daily. This is no task for humans. An automated intelligent system is required which can take out all the vital information from the unstructured resumes and transform all of them to a common structured format which can then be ranked for a specific job position. Parsed information include name, email address, social profiles, personal websites, years of work experience, work experiences, years of education, education experiences, publications, certifications, volunteer experiences, keywords and finally the cluster of the resume (ex: computer science, human resource, etc.). The parsed information is then stored in a database (NoSQL in this case) for later use. Unlike other unstructured data (ex: email body, web page contents, etc.), resumes are a bit structured. Information is stored in discrete sets. Each set contains data about the person's contact, work experience or education details. In spite of this resumes are difficult to parse. This is because they vary in types of information, their order, writing style, etc. Moreover, they can be written in various formats. Some of the common ones include '.txt', '.pdf', '.doc', '.docx', '.odt', '.rtf' etc. To parse the data from different kinds of resumes effectively and efficiently, the model must not rely on the order or type of data.

III. HISTORY OF HIRING

The process of hiring has evolved over the period of time. In the first generation hiring model, the companies would advertise their vacancies on newspapers and television. The applicants would send in their resumes via post and their resumes would be sorted manually. Once shortlisted, the hiring team would call the applicants for further rounds of interview. Needless to say, this was a time-consuming procedure. But the industries started growing and so did the hiring needs. Hence the companies started outsourcing their hiring process. Hiring consultancies came into existence. These agencies required the applicants to upload their resumes on their websites in particular formats. The agencies would then go through the structured data and shortlist candidates for the company. This process had a major drawback. There were numerous agencies and each had their own unique format. To overcome all the above problems an intelligent algorithm was required which could parse information from any unstructured resumes, sort it based on the clusters and rank it finally. The model uses natural language processing to understand the resume and then parse the information from it. Once information is parsed it is stored in the database. When the employer posts a job opening, the system ranks the resumes based on keyword matching and shows the most relevant ones to the employer.

IV. PREPROCESSING

Data preprocessing is the first and foremost step of natural language processing. Data preprocessing is a technique of data mining which transforms raw data into a comprehensible format. Data from the real world is mostly inadequate, conflicting and contains innumerable errors. The method of Data preprocessing has proven to resolve such issues. Data preprocessing thus further processes the raw data. Data is made to pass through a series of steps in the time of preprocessing:

Data Cleaning: Processes, like filling in missing values, smoothing noisy data or resolving inconsistencies, cleanses the data.

Data Integration: Data consisting of various representations are clustered together and the clashes between the data are taken care of.

Data Transformation: Data is distributed, assembled and theorized.

Data Reduction: The objective of this step is to present a contracted model in a data warehouse.

Data Discretization: In this step, the number of values of an uninterrupted characteristic is reduced by division of the range of intervals of characteristics.

Tokenization: Tokenization is the task of chopping off a provided character sequence and a detailed document unit. It does away with certain characters like punctuation and the chopped units are further called tokens. It can be illustrated as follows:

```
words = "The angry bear chased the frightened little squirel"
tokens = [i for i in words.split()]
print (tokens)
```

```
['The', 'angry', 'bear', 'chased', 'the', 'frightened', 'little', 'squirel']
```

FIGURE.1. TOKENIZATION

Tokens are usually referred to as terms or words, but sometimes fabricating a type/token distinction is essential. A specimen of an array of characters in a document that is assembled as a helpful acceptable unit for processing is called a token. Whereas, the group of tokens which consists of same character sequence is called type. And a type that is added to the dictionary of IR system is called term. We can completely differentiate a set of index terms from tokens. As an example we can say, they can be acceptable identifiers in taxonomy, but mostly in modern IR systems, they have a strong relation with tokens in the document. Nevertheless, as a substitute for being totally the tokens appearing in the document, they are mostly derived from them by various processes of normalization.

Stemming: According to linguistic morphology and retrieval of information, stemming is the mechanism of lowering altered or derived words to their word stem, root or base. The stem shouldn't always match the morphological root of the words. It is satisfactory that relevant words map to the same stem even when the stem is not even a valid root. Algorithms for this process are being studied in computer science from the 1960s. Conflation is when a number of search engines treat word with the same stem as synonyms as an example of query expansion.

```
#STEMMING
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
ps = PorterStemmer()
example_words = ['python', 'pythoned', 'pythons', 'pythoning', 'pythoner']
for w in example_words:
    print(ps.stem(w))
```

```
python
python
python
python
python
```

FIGURE.2. STEMMING

Lemmatization: In linguistics, lemmatization is a procedure of organising the altered form of a word, such that their analysis can take place as a single term, identified by the word's dictionary form (lemma). In computational linguistics, the procedure of concluding the lemma of a word depending upon its predetermined meaning. It depends on rightly identifying the predetermined part of speech and what a word in a sentence means, as well as in a bigger situation surrounding the sentences, which can include neighboring sentences, and even an entire document, which contradicts stemming. So, an algorithm of lemmatization is an open platform for research.

Parts of speech tagging: In corpus linguistics, the procedure of marking up a text (corpus) which is analogous to a particular part of speech depending on both its definition and context (like how it is related to the adjoining words in a paragraph, sentence or phrase) is called part-of-speech tagging (POS tagging or POST). It is also known as grammatical tagging or word-category disambiguation. School-age children are commonly taught a simplified version of this when they are taught to identify whether the word is a noun, verb, adjective, adverb, etc.

```
#POS TAGGING
words = "The angry bear chased the frightened little squirel"
tokenized = [i for i in words.split()]
for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            print(tagged)
```

```
[('The', 'DT')]
[('angry', 'JJ')]
[('bear', 'NN')]
[('chased', 'VBN')]
[('the', 'DT')]
[('frightened', 'VBN')]
[('little', 'JJ')]
[('squirel', 'NN')]
```

FIGURE.3. POS TAGGING

Chunking: Also known as shallow parsing, chunking is actually the recognition of parts of speech and short phrases. We can determine if the words are nouns, verbs, adjectives, etc by Parts of Speech tagging, but from this, we cannot get any clue about the sentence of phrase structure in the sentence. At times, some more information than parts of speech of words are useful, but the full parse tree that we would get from parsing is not needed. Named-entity recognition is a citation when chunking might be preferable. In NER, the goal is finding named entities, which mostly has a tendency to be noun phrases, so in the following sentence we would want to know if 'The angry bear' is there or not: "The angry bear chased the frightened little squirrel" But one wouldn't necessarily care if the angry bear is the subject of the sentence or not. Chunking is also commonly used in tasks like example-based machine natural language understanding, speech generation, and others as a pre-processing step.

```
#CHUNKING
words = "The angry bear chased the frightened little squirel"
tokenized = [i for i in words.split()]
for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            chunkGram = r"""Chunk: {<RB.?>*<VB.?>*<NNP>+<NN>?}"""
            chunkParser = nltk.RegexpParser(chunkGram)
            chunked = chunkParser.parse(tagged)
            print(chunked)
```

```
(S The/DT)
(S angry/JJ)
(S bear/NN)
(S chased/VBN)
(S the/DT)
(S frightened/VBN)
(S little/JJ)
(S squirel/NN)
```

FIGURE.4. CHUNKING

V. THE MODEL

We can classify the model into three major parts which we will be discussing in depth.

A. *The uploader*

B. *The parser*

A. THE UPLOADER

The client may be a giant corporate firm who wants to parse and rank their tens and thousands of unstructured resumes, or a student trying to beautify his/her unstructured text resume and convert into a beautiful pdf format. In either case, the algorithm remains same. First, the client uploads a file. The algorithm blocks any file having an extension other than '.pdf', '.doc', '.docx', '.odt', '.ods', '.txt'. After the client has successfully uploaded the file, the algorithm takes the file, reads the

contents and writes the content into a text file before passing on the data to the parser.

FIGURE.5. THE UPLOADER

B. THE PARSER

Once uploaded as a text file by the up loader, the parser comes in play. It parses all the relevant data from the uploaded resume including name, emails, contact numbers, social profile links, personal websites, years of work experience, work experiences, years of education, degrees, volunteer experiences, publications, skills, cluster(s) and languages through natural language processing and without any human interaction. Now, what exactly is natural language processing?

Natural Language Processing:

Natural language processing is a branch of artificial intelligence and computational linguistics. It can be defined as the process which is involved in the interaction between a computer and natural language i.e the language, spoken by humans. It is directly related to the field of human-computer interaction. Now that natural language processing is properly defined, we will be using the following constraints of NLP to parse the information from the resumes:

I. Lexical Analysis

II. Syntactic Analysis

III. Semantic Analysis

I. LEXICAL ANALYSIS

The pilot stage of the compiler is lexical analysis. The altered source code is taken from the language preprocessor which writes in the form of sentences. The analyzer removes any comments or whitespace from the source code, breaking these syntaxes into a chain of tokens.

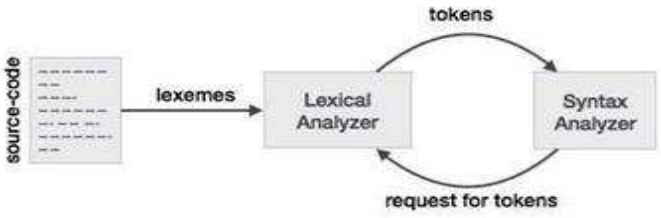

FIGURE.6. LEXICAL ANALYSIS

Considering our case, the resume is discriminated onto various segments including contact information, educational experiences, work experiences and more. We use a database or a data dictionary to hold the keywords or headings we find common in most of the resumes. Now when a new resume is taken, the parser searches for the keywords and extracts all the data between the starting and the ending of them, which we call as segments. Out of the many exceptions which might occur, one which is common is that the first segment generally contains the name as well as the contact information of the person. Now we program chunkers or Named Entity Recognizer to extract data from each segment specifically. This method makes the system efficient and reduces its complexity. Now if due to some reason the recognizer runs on a wrong piece of data, the system will produce unexpected results.

II. SYNTACTIC ANALYSIS

The syntactic analysis determines the structure of data. The architecture comprises of a hierarchy of expressions, the smallest being a basic symbol and the largest being sentences. We can visualise the architecture as a tree whose nodes represents the expressions. Values stored in the nodes represent the basic symbols. The root represents the sentence.

Parse Tree:

The parse tree is generated by the parser with syntactic analysis. A parse tree or a parsing tree is an organised, entrenched structure which we use to represent the syntactic analysis of a string. They categorically reflect the syntax of the input data, making them noticeable from the abstract syntax trees used in programming.

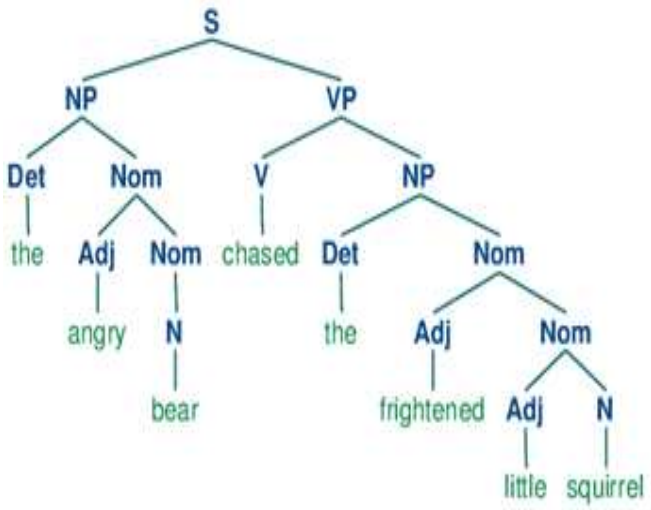

FIGURE.7. PARSE TREE REPRESENTING SYNTACTIC ANALYSIS

II. SEMANTIC ANALYSIS

Semantic analysis can be defined as the study of semantics i.e the structure and meaning of speech. This process relates syntactic structure to the level of the writing as a whole from the levels of clauses, phrases, paragraph and sentences. It relates to their language-independent meanings. Let's take an example. Person A has a resume which states he has graduated from the "University of Calcutta" and person B has a resume which says he has graduated from "Calcutta University". Essentially they both graduated from the same place. So what semantic analyzer does is convert "University of Calcutta" to "Calcutta University". In Information Retrieval research, text classification system is given the utmost focus which bounds the decisions to either relevant or non-relevant depending upon the information need of the user. It is not a hard task to get the user information need.

THE MODEL

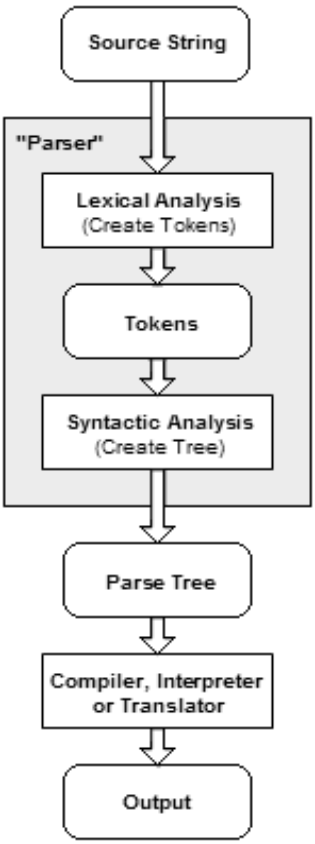

FIGURE.8. THE MODEL

Now that we understand what Lexical, Syntactic and Semantic analysis does, let us see how the system works. The data is given to the system as a raw string. The Lexical analyzer pre-processes the data and tokenizes them. The Syntactic analyzer takes the tokens and finds the structure in it. The parse tree diagrammatically represents the syntactic structure in the form of a tree. The Semantic analyzer studies the structure of the data to find their language-independent meaning.

VI. EXPECTED OUTCOME

Both employers and candidates will be appeased by our system. Lots of strain on the head of candidate or employee in Online Recruitment System will have been reduced by this online tool. The system will parse all the resumes and store them in the database. Then it will rank them using Artificial Intelligence or AI and predict which candidate is best suited for the job, thus making the hiring system authentic. Some screen-shots of the result of our resume parser are portrayed below. Once the user confirms the result of our parser the system generates a JSON resume and stores it in the NoSQL database.

FIGURE.9.1. BASIC INFORMATION

FIGURE.9.2. WEBSITE AND EMAIL

FIGURE.9.3. ADDRESS

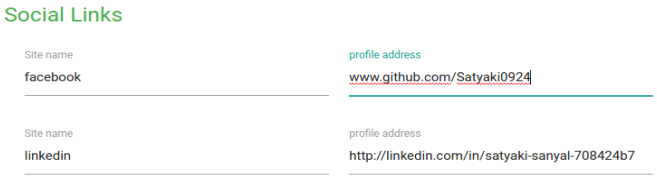

FIGURE.9.4 SOCIAL LINKS

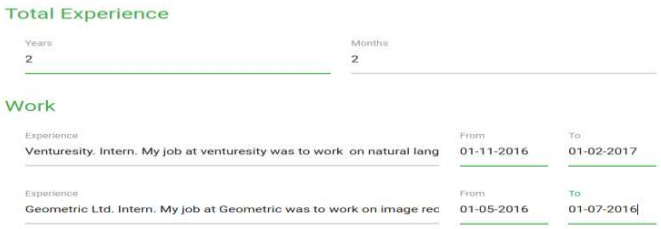

FIGURE.9.5. WORK EXPERIENCE

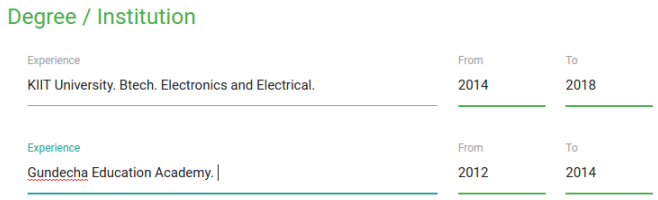

FIGURE.9.6. EDUCATIONAL EXPERIENCE

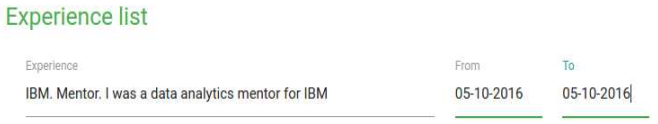

FIGURE.9.7. VOLUNTEER EXPERIENCE

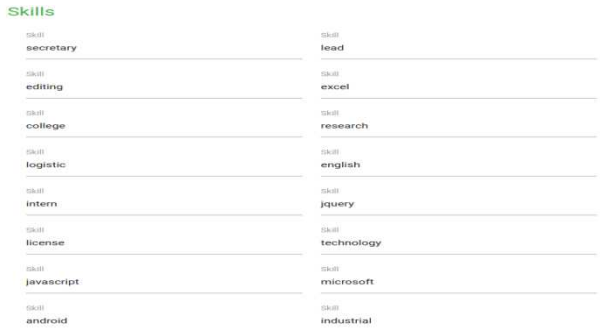

FIGURE.9.8. SKILLS

VII. JSON OUTPUT


```
{
  "basics": {
    "name": "Satyaki Sanyal",
    "label": "Programmer",
    "picture": "",
    "email": "sanyal.satyaki09@gmail.com",
    "phone": "(+91) 9178449492",
    "website": "http://www.satyakisanyal.com",
    "summary": "A summary of John Doe...",
    "location": {
      "address": "Acharya Prafulla Chandra Road",
      "postal Code": "700020",
      "city": "Kolkata",
      "country": "India",
    },
    "profiles": [{
      "Twitter": "http://www.twitter.com/Satsan95",
      "Github": "www.github.com/Satyaki0924",
      "LinkedIn": "http://linkedin.com/in/satyaki-
sanyal-708424b7"
    }]
  },
  "work": [{
    "company": "Venturesity",
    "position": "Intern",
    "start Date": "01-11-2016",
    "end Date": "01-02-2017",
    "summary": "My job at venturesity was to work on
natural language processing and make online parsing
systems",
  },
{
    "company": "Geometric Ltd.",
    "position": "Intern",
    "start Date": "01-05-2016",
    "end Date": "01-07-2016",
    "summary": "My job at Geometric was to work
          on image recognition and simulate
self driving with reinforcement learning",
  },
],
  "volunteer": [{
    "organization": "IBM",
    "position": "Mentor",
    "start Date": "05-10-2016",
    "end Date": "05-10-2016",
    "summary": "I was a data analytics mentor for
IBM",
  }],
  "education": [{
    "institution": "KIIT University",
    "area": "Electronics and Electrical",
    "study Type": "Btech",
    "start Date": "2014",
    "end Date": "2018",
  },
{
    "institution": "Gundecha Education Academy",
    "area": "Science",
    "study Type": "Indian School Certificate",
    "start Date": "2012",
    "end Date": "2014",
  }],
  "awards": [{
    "title": "First prize in robotics",
    "date": "15-08-2016",
    "awarder": "VIT Vellore",
  }],
  "publications": [{
    "name": "recruitment predictions with id3 decision tree",
    "publisher": "international journal of advanced engineering
and research development",
    "release Date": "22-10-2016",
    "website": "http://www.ijaerd.com",
    "summary": "we have tried to solve the problem
          of recruitment with cognitive
          computing. we have used decision trees to predict the
candidates best suited for the job and we have used random
forest for better pedictions .."
  }],
  "skills": [{
      "HTML",
      "CSS",
      "Javascript",
      "Python",
      "Machine Learning",
      "Deep Learning"
  }],
  "languages": [{
    "name": "English",
    "level": "Expert"
  },
{
    "name": "Hindi",
    "level": "Expert"
  }],
  "interests": [{
    "Swimming",
    "Reading books",
    "Watching movies",
    "Coding"
    ]
  }]
  "references": [{
    "name": "John Doe",
    "reference": "Satyaki was an asset to our
company.."
  }]
}
```


VIII. CONCLUSION AND FUTURE WORK

We successfully converted different formats of resumes to text and parse relevant information from there. We also were able to scrape keywords from different social networking sites including Stack Overflow, LinkedIn, etc and find the similarity between them with which we could determine the genre of the resume (e.g: Computer science, Management, Sales, human resource, etc). Future work includes ranking the resume and analysing information about the candidate from social networking sites like Face book and Twitter so that we can decide more accurately and authentically whether or not to offer the candidate, a job.

IX. ACKNOWLEDGEMENT

We are grateful to all anonymous reviewers for their valuable feedback.

X. REFERENCE

[1]. F. Ciravegna, "Adaptive information extraction from text by rule induction and generalisation," in Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI2001), 2001.

[2]. A. Chandel, P. Nagesh, and S. Sarawagi, "Efficient batch top-k search for dictionary-based entity recognition," in Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE), 2006.

[3]. S. Chakrabarti, Mining the Web: Discovering Knowledge from Hypertext Data. Morgan-Kauffman, 2002

[4]. M. J. Cafarella, D. Downey, S. Soderland, and O. Etzioni, "KnowItNow: Fast, scalable information extraction from the web," in Conference on Human Language Technologies (HLT/EMNLP), 2005.

[5]. M. J. Cafarella and O. Etzioni, "A search engine for natural language applications," in WWW, pp. 442–452, 2005.

[6]. https://www.ijircce.com/upload/2016 /april/218_ Intellig ent.pdf

[7]. https://www.tutorialspoint.com/compiler_design/images /token_passing.jpg

[8]. http://www.nltk.org/book/tree_images/ch08-tree-6.png