

Automatic Resume Ranking using Machine Learning

JULY 6

Data Science Intern

Submitted by: Saifullah Rahimi



Index

1. Abstract
2. Introduction
3. First Approach
 - Data Collection
 - Training Word2Vec model
 - Extracting Sections
 - Assigning scores for ranking
4. Second Approach
 - Methodology
 - Deployment and Inference
 - CV Recommendation Model
 - k-Nearest Neighbours
5. Conclusion
6. References

Abstract

A huge number of applications received by the organization for every job post finding suitable candidates for an open role could be a daunting task, especially when there are many applicants. It can impede team progress in getting the right person at the right time. The process of classifying the candidate's resume is manual, time-consuming, and a waste of resources. To overcome this issue, I have proposed an automated machine learning-based model which recommends suitable candidates' resumes to HR based on the given job description. An automated way of "Resume Ranking and Filtering " could ease the tedious process of fair screening and shortlist, it would certainly expedite the candidate selection and decision-making process. Automatic resume ranking is the project which enables recruiters to wade through an ocean of resumes (especially during high-volume recruiting) to find the perfect candidates that match the job requirements. Two different approaches have been used for automatic resume ranking.

In the first approach, first classify the right categories using a different classifier, once classification has been done then as per the job description, top candidates could be ranked using k-NN to identify the CVs that are nearest to the provided job description.

The second approach filters applications based on skills, education, experience, or anything that is a requirement for an open role. The project uses techniques in Machine Learning and Natural Language Processing to automate the procedure using the trained model from datasets.

Involving domain experts like HR professionals would help to build a more accurate model, and feedback from the HR professional helps to improve the model iteratively.

Introduction

Talent acquisition is an important, complex, and time-consuming function within Human Resources (HR). The most challenging part is the lack of a standard structure and format for resumes which makes the short listing of desired profiles for required roles very tedious and time-consuming. Effective screening of resumes requires domain knowledge, to be able to understand the relevance and applicability of a profile for the job role. With a huge number of different job roles existing today along with the typically large number of applications received, short-listing poses a challenge for the human resource department. This is only further worsened by the lack of diverse skills and domain knowledge within the HR department, required for effective screening. Being able to weed out non-relevant profiles as early as possible in the pipeline results in cost savings, both in terms of time as well as money.

To overcome the mentioned issues in the resume short-listing process, in this paper we present an automated Machine Learning-based model and NLP technique. The model takes the features extracted from the candidate's resume as input and finds their categories, further based on the required job description the categorized resume is mapped and recommends the most suitable candidate's profile to HR. My main contributions are listed below:

1. Using StackExchange datasets trained my model which is the word2vec model used for resume ranking.
2. I developed an automated resume recommendation system which is a machine learning-based classification technique with similar functions used to find the most relevant resume.

The project aims to automate the filtering process of the influx of CVs/resumes for an IT company.

First Approach

Resume Ranking using Machine Learning and NLP techniques.

It filters applications based on skills, education, experience, or anything that is a requirement for an open role. The result was achieved by assigning a score to each CV by intelligently comparing them against the corresponding Job Description. The project uses techniques in Machine Learning and Natural Language Processing to automate the procedure.

Division of the report

The report has been divided into 4 sections:

1. Data Collection
2. Training Word2Vec Model
3. Extracting sections
4. Assigning scores for ranking

Section 1. Data Collection

Mainly used three datasets:-

1. Job Descriptions dataset
2. StackExchange Network Posts dataset
3. Resumes Collection

1.1 Job Description Dataset

A Kaggle dataset containing Job Descriptions for several job openings was used.

I have used NLP to filter out the Job Descriptions related to the IT industry.

Finally, 5000+ JDs including JDs for positions like 'Web developer', 'C++ software developer', 'Software developer', and 'Embedded Software Engineer' were filtered out and saved in CSV format.

1.2 StackExchange Network Posts dataset

This dataset was required to be trained and loaded to the word2vec model. StackExchange network dumps its data in XML format under Creative Commons License. This is an anonymized dump of all user-contributed content on the Stack Exchange network. Each site is formatted as a separate archive consisting of XML files zipped via 7-zip using bzip2 compression. For some portions of the dataset contents I mentioned below, click the below link to view or download the dataset. Dataset link on Internet Archive.

1.3 Resumes

Around 250 resumes of applicants were collected for positions like 'Software Developer', 'Data Scientist', 'Web Developer' etc.

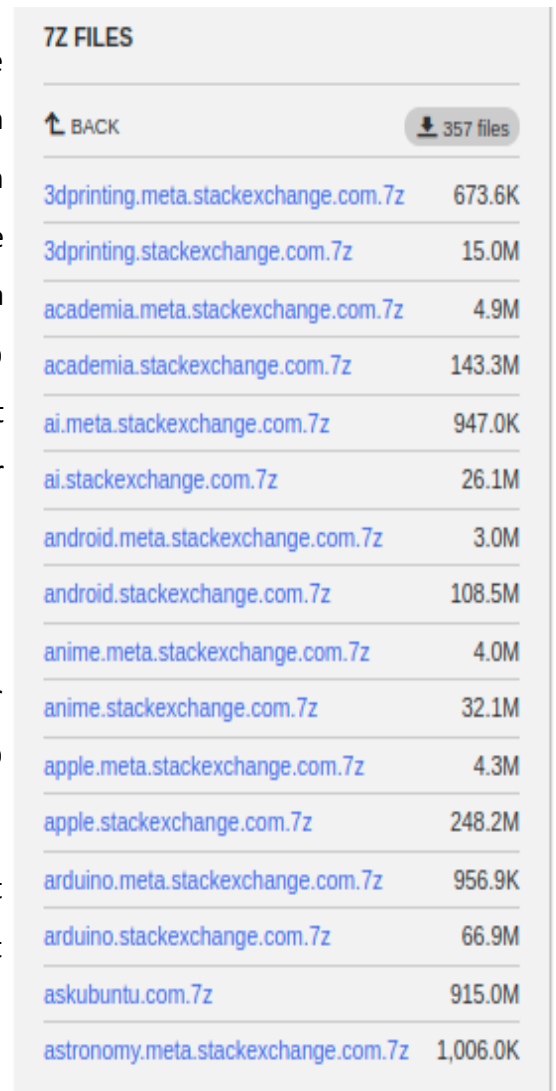
You can collect more Cvs from any free websites in text format using the python script provided in the project directory.

This dataset was required to test the trained word2vec

model. Among these resumes, best matching resumes should be filtered out. **Figure 1:** Datasets

Section 2. Training Word2Vec Model

Word2Vec models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share familiar contexts in the corpus are located close to one another in the space.



7Z FILES	
3dprinting.meta.stackexchange.com.7z	673.6K
3dprinting.stackexchange.com.7z	15.0M
academia.meta.stackexchange.com.7z	4.9M
academia.stackexchange.com.7z	143.3M
ai.meta.stackexchange.com.7z	947.0K
ai.stackexchange.com.7z	26.1M
android.meta.stackexchange.com.7z	3.0M
android.stackexchange.com.7z	108.5M
anime.meta.stackexchange.com.7z	4.0M
anime.stackexchange.com.7z	32.1M
apple.meta.stackexchange.com.7z	4.3M
apple.stackexchange.com.7z	248.2M
arduino.meta.stackexchange.com.7z	956.9K
arduino.stackexchange.com.7z	66.9M
askubuntu.com.7z	915.0M
astronomy.meta.stackexchange.com.7z	1,006.0K

Requirements of training our model

There are pre-trained models available both in gensim and spaCy packages in Python. These models are trained over Google News Data. This implies that they are not suitable for the technically aware context distinction required for this project.

For e.g. HTML and Ruby may have higher similarity value in these models than the model we trained. Therefore, a dataset was required which was both technically aware and also has sufficient amount of unique words present for the non-technical functioning of the model.

The dataset used to train Word2Vec model becomes more crucial considering the fact that Word2Vec models can be retrained over and over, however, new Vocabulary cannot be added to the model. Therefore, stackexchange network data was used and trained my own model.

Cleaning and Extracting data

From the stackexchange/ dataset the Posts.xml for each site was used to extract each Post irrespective of whether it's a Question or an Answer. These Posts were extracted as HTML para tags and saved as paras.txt in the corresponding subfolder of the site.

At this stage, each subdirectory of StackExchange/ which corresponds to the site under StackExchange network, has a new file called paras.txt .

For training the Word2Vec model, we required a sequence of sentences to be streamed from the disk. Each sentence is represented as a list i.e. each element of this list is the word of the sentence. So, the paras.txt files were used to extract sentences using BeautifulSoup(a Python Library), and saved into sentences.txt (for each site), such that the final result is free of formatting and mathematics, code etc.

These sentences were streamed into the Word2Vec train method for training the model.

Section 3. Extracting sections

I have some collection of words that are usually the heading in the resumes. For example 'education', 'academic', 'school', 'study', etc will mark the start of the education section, iterate over all lines of all resumes, one by one. For each line, first, we remove all the blank lines or the lines containing just symbols. Some resumes have a line denoted to just asterisks or dashes.

Next, we categorize each line into one of the four sections. This is done by calculating its similarity to the existing words. If the similarity is higher than the threshold, we update the section and mark that point, on the other hand, if the similarity is below the threshold, we continue with the previous section.

This enables us to separate the sections with good enough accuracy.

Finally, we write each section of a resume in a .csv file after removing the stop words and doing lemmatization.

Section 4. Assigning scores for ranking

For a given Job Description, we remove all the stop words and do lemmatization, to get a selected few keywords.

For each keyword found, we find 5 similar words and their corresponding similarity.

Now, we find tf-idf for each word, that we got in step 2.

The score of the CV is the sum of $\text{tf-idf} * \text{similarity}$ for all words we got in step 2.

How to run the project :

unzip all the downloaded dataset of stackexchange network Posts dataset using zipextract.py and place the unzipped files inside a directory named stackexchange.

Each subdirectory inside stackexchange folder includes Posts, Users, Votes, Comments, PostHistory and PostLinks (all in .xml files).

Next, Follow the following steps:

1. Download and unzip the StackExchange Network Posts dataset(Follow the above instruction).
2. Each subdirectory of dataset contains posts.xml therefore Generate text file named (paras.txt) using Extraction_from_posts.ipynb file.
3. Generate another text file named (sentences.txt) using Sentence_Extraction.ipynb file.
4. Train the model using Model_Training.ipynb file after successfully run it will generate a trained model named stackexchange_model in CWD.

How to use trained model for CV Ranking and Word2Vec

Generate CSV file named prc_data.csv using Section_Extraction.ipynb

All the model and cleaned data are ready therefore for cv ranking.

You can rank the resumes by CV_ranking.ipynb file.

Run With Word2Vec.ipynb to understand better concept of word2vec using trained model stackexchange_model model.

Model Training

Model_Training.ipynb : Notebook to train and save the word2vec model. After successfully run the model(stackexchange_model) will be saved in ./Model/ subdirectory (locally).

```
model = gensim.models.Word2Vec(sentences, workers=4, size=300, min_count = 1, window = 15, sample = 1e-3)
```

```
2022-05-05 12:26:22,341 : INFO : EPOCH 5 - PROGRESS: at 94.10% examples, 603222 words/s, in_qsize 8, out_qsize 1
2022-05-05 12:26:22,487 : INFO : INITIATED: Processing sentences for dataset29
2022-05-05 12:26:22,590 : INFO : INITIATED: Processing sentences for dataset9
2022-05-05 12:26:22,688 : INFO : INITIATED: Processing sentences for dataset39
2022-05-05 12:26:22,761 : INFO : INITIATED: Processing sentences for dataset27
2022-05-05 12:26:23,327 : INFO : INITIATED: Processing sentences for dataset2
2022-05-05 12:26:23,345 : INFO : EPOCH 5 - PROGRESS: at 96.60% examples, 601606 words/s, in_qsize 7, out_qsize 0
2022-05-05 12:26:23,384 : INFO : INITIATED: Processing sentences for dataset10
2022-05-05 12:26:23,852 : INFO : INITIATED: Processing sentences for dataset18
2022-05-05 12:26:24,362 : INFO : EPOCH 5 - PROGRESS: at 99.01% examples, 601318 words/s, in_qsize 8, out_qsize 1
2022-05-05 12:26:24,517 : INFO : INITIATED: Processing sentences for dataset33
2022-05-05 12:26:24,666 : INFO : worker thread finished; awaiting finish of 3 more threads
2022-05-05 12:26:24,672 : INFO : worker thread finished; awaiting finish of 2 more threads
2022-05-05 12:26:24,679 : INFO : worker thread finished; awaiting finish of 1 more threads
2022-05-05 12:26:24,696 : INFO : worker thread finished; awaiting finish of 0 more threads
2022-05-05 12:26:24,697 : INFO : EPOCH - 5 : training on 33867415 raw words (25231308 effective words) took 41.9
s, 602208 effective words/s
2022-05-05 12:26:24,697 : INFO : training on a 169337075 raw words (126161472 effective words) took 210.4s, 59949
0 effective words/s
```

```
In [15]: # save the model for later use. You can load it later using Word2Vec.load()
model_name = "stackexchange_model"
model.save(model_name)
```

```
2022-05-05 12:46:03,612 : INFO : saving Word2Vec object under stackexchange_model, separately None
2022-05-05 12:46:03,615 : INFO : storing np array 'vectors' to stackexchange_model.wv.vectors.npy
2022-05-05 12:46:03,698 : INFO : not storing attribute vectors_norm
2022-05-05 12:46:03,699 : INFO : storing np array 'synlneg' to stackexchange_model.trainables.synlneg.npy
2022-05-05 12:46:03,773 : INFO : not storing attribute cum_table
2022-05-05 12:46:03,946 : INFO : saved stackexchange_model
```

Figure 2: Model Training

Extraction_from_posts.ipynb : Notebook for extracting paragraphs(paras.txt) from Posts.xml.

Sentence_Extraction.ipynb : Notebook for extracting cleaned sentences(sentences.txt) from extracted paragraphs (paras.txt).

paras.txt : Each and every subdirectory of dataset in stackexchange directory contains this file which is paragraph in html tags file format, It was extracted from Posts.xml using the code Extraction_from_posts.ipynb

stackexchange/ : Its a directory that contains all the unzip stackoverflow datasets.

sentences.txt : Each and every subdirectory of the dataset in stackexchange directory contains this file, It was extracted from the corresponding paras.txt which was generated earlier using the code sentence_Extraction.ipynb. The process took around 24.5 hours to complete.

Using Spacy Model.ipynb : Demonstrates the need for a custom Word2Vector model rather than a general model trained otherwise. The similarity values generated by en_core_web_md spaCy model trained on Google News articles, do not reflect the technological sharpness required for the project.

With Word2Vec.ipynb : Demonstrates how to use word2vec to get similar words by words and similar words by vector. It also implements sent2vec() function. This function takes a sentence as an argument and returns an average vector for the sentence. Root Mean Square is used to average the vectors. The advantage of this function is to use it to find similar words for phrases that makes more sense while searching for roles etc.

Using Word2Vec to get similar words

```
In [7]: #Get 5 most similar words by words
model.wv.most_similar(positive=['android'],topn=5)

Out[7]: [('ios', 0.7613162994384766),
          ('iphone', 0.7476478815078735),
          ('ipad', 0.7242550849914551),
          ('mobile', 0.7198758125305176),
          ('mac', 0.6739652156829834)]

In [8]: #Get 5 most similar words by vector.
# We can obviously get similar words by providing a vector using the format below:
'''model.similar_by_vector(vector,topn = 10)'''
#But we can also use it cleverly to find a similar word to a phrase, using my sent2vec method:

model.wv.similar_by_vector(softdev, topn =5)

Out[8]: [('software', 0.8723065853118896),
          ('developer', 0.7645061016082764),
          ('hardware', 0.6354432702064514),
          ('programmer', 0.613621711730957),
          ('linux', 0.5968120694160461)]
```

Figure 3: Word2Vec

Section Extrraction

Section_Extraction.ipynb : Notebook for extracting main sections from different resumes.

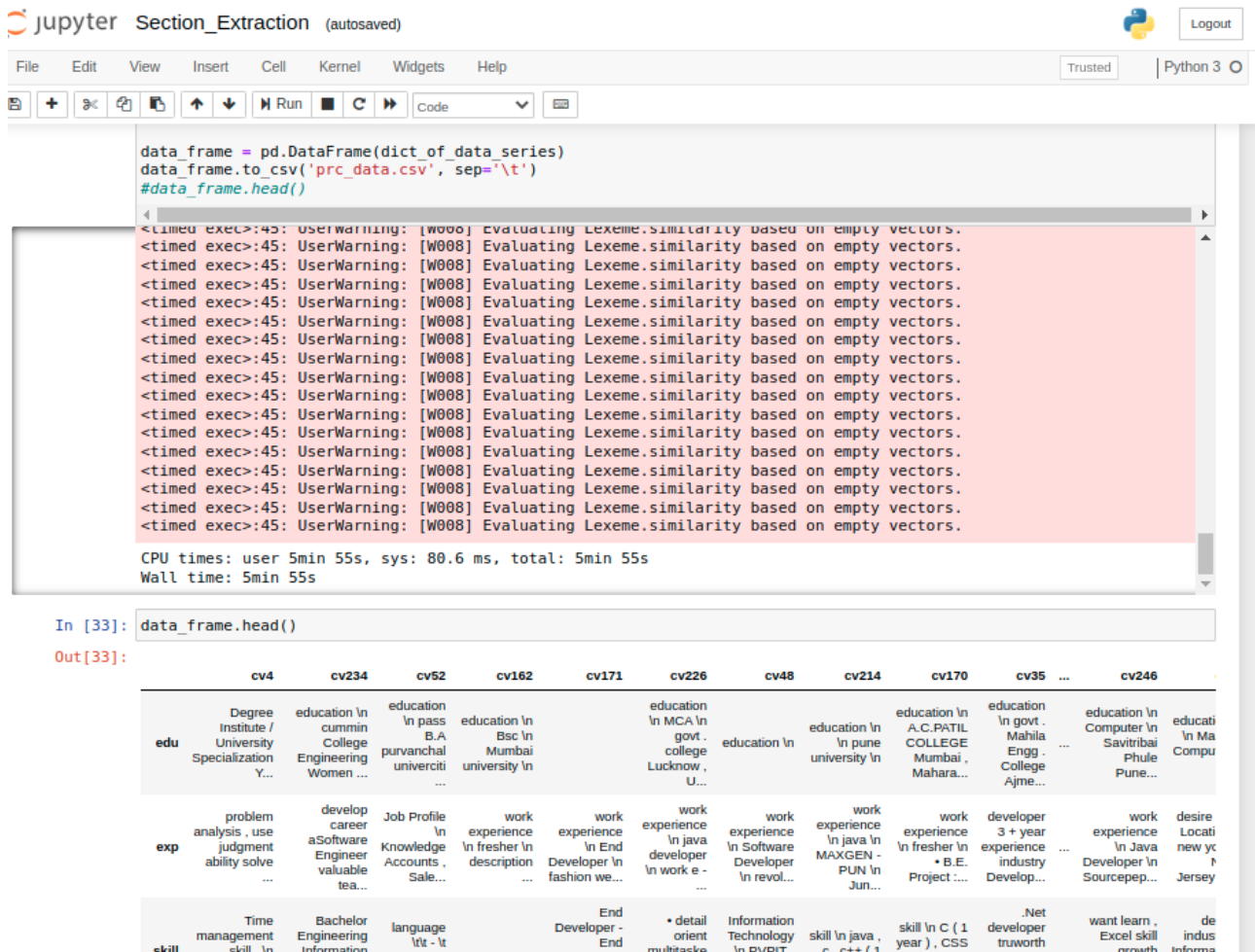
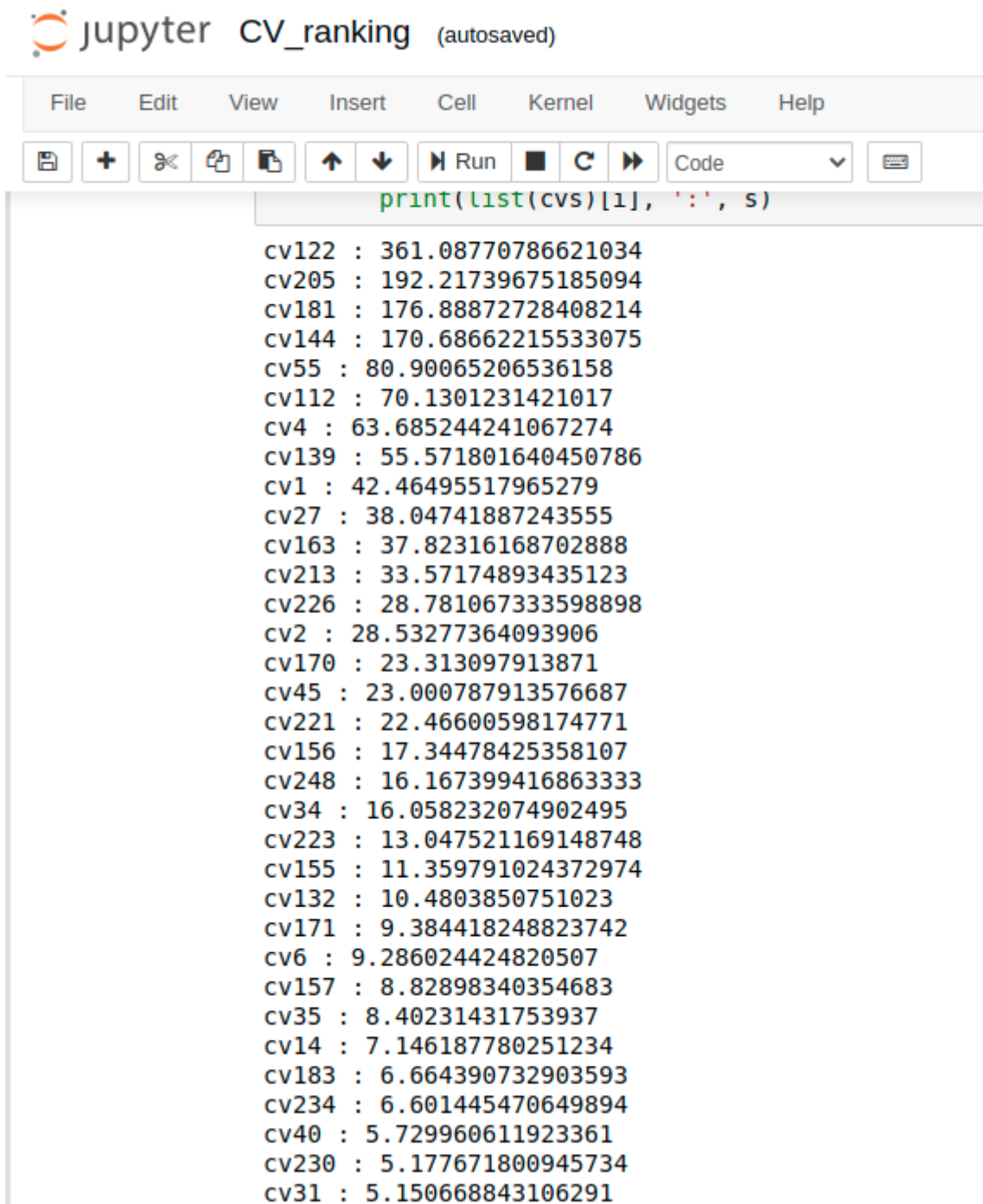


Figure 4: Section Extraction

Resume Ranking

CV_ranking.ipynb : Notebook for ranking the CVs according to Job Description.



The image shows a Jupyter Notebook interface with the title "jupyter CV_ranking (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar contains icons for saving, adding cells, undo, redo, copy, paste, and navigation. The code cell contains the following Python code:

```
print(list(cvs)[1], ': ', s)
```

The output of the code is a list of 36 CVs, each followed by a colon, a space, and a numerical value. The CVs are sorted in descending order of their numerical values. The values range from 361.08770786621034 for cv122 to 5.150668843106291 for cv31.

CV ID	Value
cv122	361.08770786621034
cv205	192.21739675185094
cv181	176.88872728408214
cv144	170.68662215533075
cv55	80.90065206536158
cv112	70.1301231421017
cv4	63.685244241067274
cv139	55.571801640450786
cv1	42.46495517965279
cv27	38.04741887243555
cv163	37.82316168702888
cv213	33.57174893435123
cv226	28.781067333598898
cv2	28.53277364093906
cv170	23.313097913871
cv45	23.000787913576687
cv221	22.46600598174771
cv156	17.34478425358107
cv248	16.167399416863333
cv34	16.058232074902495
cv223	13.047521169148748
cv155	11.359791024372974
cv132	10.4803850751023
cv171	9.384418248823742
cv6	9.286024424820507
cv157	8.82898340354683
cv35	8.40231431753937
cv14	7.146187780251234
cv183	6.664390732903593
cv234	6.601445470649894
cv40	5.729960611923361
cv230	5.177671800945734
cv31	5.150668843106291

Figure 4: CV Ranking

Second Approach

Resume Ranking using KNN

A python web app for analyzing resumes/ CVs that best match the position. which helps employers by analyzing resumes and CVs, and ranking the candidates' resumes based on the estimated job description. In this project, I have used the KNN concept along with recommendation engine techniques such as Collaborative, Content-Based filtering for fuzzy matching job descriptions with multiple resumes.

Methodology

This work aims to find the right candidates' resumes from the pool of resumes.

Preprocessing

In this process, the CVs being provided as input would be cleansed to remove special or any junk characters that are there in the CVs. In cleaning, all special characters, numbers, and single-letter words are removed. I got the clean dataset after these steps having no special characters, numbers, or single letter words. The dataset is split into the tokens using the NLTK tokenizes. Further, the preprocessing steps are applied to tokenized datasets such as stop word removal, stemming, and lemmatization. The raw CV file was imported and the data in the resume field was cleansed to remove the numbers and the extra spaces in the date.

Stop words removal: The stop words such as and, the, was, etc. are frequently appeared in the text and not helpful for the prediction process, hence it is removed. Steps to filter the Stop Words:

1. I have to tokenize the input words into individual tokens and stored them in an array.
2. Now, each word matches with the list of Stop Words present in the NLTK library:

(a) `from nltk.corpus import stopwords /*Imported Stop Word module from NLTK corpus*/`

(b) `StopWords[]= set(stopwords.words('english')) /* Get set of English Stop Words*/`

3. If the words present in the list of StopWords[], are filtered from the main sentence array.
4. The same process is repeated until the last element of the tokenized array is not matched.
5. Resultant array does not have any stop words.

Stemming: Stemming is the method of decreasing word inflection to its root forms such as mapping a group of words to the same stem even though the stem itself is not a valid term in the language. Stem (root) is the part of the word to which you add inflectionally (changing/deriving) affixes such as (-ed, -ize, -s, -de, -ing, mis). For example, the words like Playing, Plays, and Played are mapped to their root word Play, and the words like python, pythoner, pythoning, pythoned are mapped to their root word python.

Lemmatization: Unlike Stemming, lemmatization decreases the inflected phrases to ensure that the root word belongs to the language correctly. Lemmatization comprises the following routine steps1 Transform the corpus of text into a list of words.

The next step is feature extraction. On preprocessed dataset, I have extracted the features using the tf-Idf .

The cleansed data was imported and feature extraction was carried out using Tf-Idf. The machine learning-based classification model or learning algorithms need a fixed-size numerical vector as input to process it. ML-based classifiers did not process the raw text having the variable size in length. Therefore, the texts are converted to a required equal length of vector form during the preprocessing steps. There are many approaches used to extract the features such as BoW(Bag of Words), tf-idf (Term Frequency, Inverse Document Frequency), etc. In the BoW model, for each document, a complaint the narrative in our case, the presence (and often the frequency) of words is taken into consideration, but the order in which they occur is ignored. Specifically, we have calculated tf-idf (term frequency, and inverse document frequency)

Deployment and Inference

In this process, the tokenized CV data and the job descriptions (JD) would be compared and the model would provide CVs relevant to the job description as an output.

CV Recommendation Model

The recommendation model is designed to take the job description and CVs as input and provide the list of CVs which are closest to the provided job description. This is done using the k-Nearest Neighbours approach.

k-Nearest Neighbours

In this model, k-NN is used to identify the CVs that are nearest to the provided job description, in other words, the CVs that are a close match to the provided job description. First, to get the JD and CVs to a similar scale, we have used an open-source library called “gensim”, this library generates the summary of the provided text in the provided word limit. So to get the JD and CVs to similar word scales this library was used to generate a summary of JD and CVs and then k-NN was applied to find the CVs which are closely matching the provided JD.

Conclusion

However, there is room for improvement, the result is satisfactory enough for the first iteration of the project. I have learned a lot during the project and hopefully, the project will serve its purpose as well. The filtering up of CVs has always been a subjective process, although, the use of Machine Learning can certainly reduce the unnecessary amount of human effort.

References

1. spaCy Documentation: <https://spacy.io/>
2. spaCy GitHub Issue Page: <https://github.com/explosion/spaCy/issues>
3. Gensim Word2Vec Documentation: <http://radimrehurek.com/gensim/models/word2vec.html>
4. Gensim Word2Vec GitHub repository: [link](#)
5. Google Word2Vec: <https://code.google.com/archive/p/word2vec/>
6. GitHub Repository for Doc2Vec Illustration: <https://github.com/linanqiu/word2vec-sentiments>
7. Al-Otaibi, S.T., Ykhlef, M., 2012. A survey of job recommender systems. *International Journal of Physical Sciences* 7, 5127–5142.
8. Breugh, J.A., 2009. The use of biodata for employee selection: Past research and future directions. *Human Resource Management Review* 19, 219–231.
9. Breiman, L., 2001. Random forests. *Machine learning* 45, 5–32.
10. Carrer-Neto, W., Hernández-Alcaraz, M.L., Valencia-García, R., García-Sánchez, F., 2012. Social knowledge-based recommender system application to the movies domain. *Expert Systems with applications* 39, 10990–11000.
11. Celma, O., 2010. Music recommendation, in: *Music recommendation and discovery*. Springer, pp. 43–85.
12. Das, A.S., Datar, M., Garg, A., Rajaram, S., 2007. Google news personalization: scalable online collaborative filtering, in: *Proceedings of the 16th international conference on World Wide Web*, ACM. pp. 271–280.
13. Diao, Q., Qiu, M., Wu, C.Y., Smola, A.J., Jiang, J., Wang, C., 2014. Jointly modeling aspects, ratings and sentiments for movie recommendation
14. (jmars), in: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM. pp. 193–202.
15. Färber, F., Weitzel, T., Keim, T., 2003. An automated recommendation approach to selection in personnel recruitment. *AMCIS 2003 proceedings*, 302.