CSCI 4/5253 - Datacenter Scale Computing
Project Proposal

Team Members:
Ganesh Chandra Satish
Rahul Chowdhury
Si Shen

Date: 10/30/2018

1. ***Problem description***

Problem our group is trying to solve:

As students and potential job seekers either for an internship or a full-time position, we have all gone through the painful task of applying in a trillion(made up number) of applications. There certainly exists a wide gap in matching the right candidate to the right job vacancy.

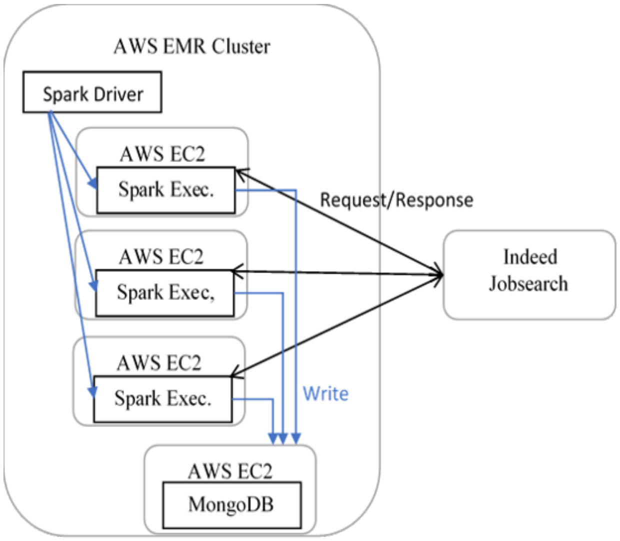
The primary step to solve our problem was to actually create a authentic dataset. Hence, instead of picking up the already existing datasets on the web(by the way, they are awesome too), we decided to make our own dataset by developing a crawler which gathers data according to the configuration set by us from Indeed.com. The base paper[1] leverages the power of deep learning to build an intelligent recommendation engine to fill this gap between candidates and vacancies There are also other similar research which we refer to as well [2] . Our primary goal for this project would be to replicate the base paper but on our own generated dataset from Indeed.com. A very simple example illustrating what our system can do would be: If the input to our system is - “we are looking for someone to manage a team of five accountants” our AI capable system after some under the hood sophisticated analysis should spit out “Finance Manager”. A more sophisticated system can take multiple inputs say like the job description before, preferred location, salary, work culture, etc. and predict the most desired job.

Why the problem is an important problem to solve:

According to our perspective, this problem of improving the mapping of jobs to the right candidates using the latest technology stack(Amazon Web Services - S3, EC2 instance, Spark, and Machine Learning/Data Science) is quite relevant to the existing current scenario of the job market. If our system is built successfully, it can be used in a fruitful way by every CU student who is potentially looking for a job in the market. Ultimately, our aim is to make the job hunting for students to be less daunting and hassle free.

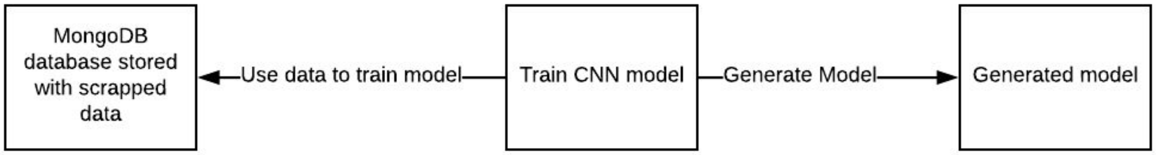
2. High-level solution architecture:

Dataset collection:



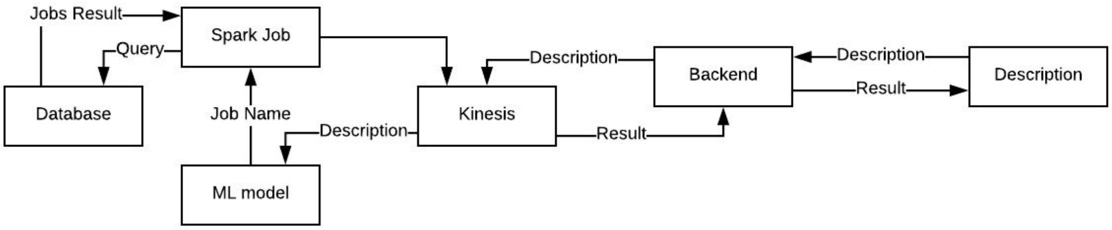
PySpark will be run in AWS EMR cluster, spanning several Spark executors on multiple EC2 instances. Each executor will run the crawling script to send requests containing certain queries to “Indeed.com”, with each request has 1-2 second interval. After getting the crawled data, the script will do some simple preprocessing and write to MongoDB. This data will be used to train the ML model.

Training the CNN model:



A CNN(Convolutional Neural Network) model will be generated with the data acquired from the previous step.

Final Architecture:



1. The description will be entered in the frontend.
2. The entered description will be taken in the backend.
3. The backend will put the data onto a kinesis stream
4. This data will be consumed by the ML model which is a python program. This ML model will get the job name that describes the input and send this to the spark job.
5. The spark job will retrieve all the data points regarding the jobs posted with that job name.
6. This data along with the job name is put into another kinesis stream.
7. The backend consumes this data and transfers it into the frontend.
8. The frontend will display the job name that best suits the description the user entered and provides a heat map of the number of jobs currently available on a world map.

3. *Dataset:*

The dataset we use is from Indeed.com. Indeed.com is an American worldwide employment-related search engine for job listings launched in November 2004. The site aggregates job listings from thousands of websites, including job boards, staffing firms, associations, and company career pages. It is one of highest-traffic job website in United States. [1]In Indeed.com, users can search job postings by location and keywords like company names, job title. Users can also refine their search by job type (full-time, internship, part-time, contract, temporary etc.), company and experience level (entry level, mid level, senior level). For example, a search like “software engineer” in “Denver, CO” will generate 5078 jobs. But we can only go up to 100 pages with each page have a list of 10-20 jobs. Each job has a job id, a job title, the company name, job location and a hyperlink to its full job description page.

As Indeed.com doesn’t provide API for individual users, we plan to crawl the job posting data. Here is a basic plan for the crawling:

1. set a list of job titles that we are most interested, like “software engineer”, “data scientist”, “data engineer” etc.
2. set a list of cities that we are most interested. Actually by searching every US state in Indeed.com, we can get a list of cities and number of the job postings in that state as a byproduct. In this way we filter by number of job postings and get a list of main cities.
3. We can set job type and experience level to the following combinations to further refine the results:
    a. job_type=“internship”
    b. job_type=“fulltime” and experience_level=”entry_level”
    c. job_type=”fulltime” and experience_level=”mid_level”
    d. job_type=”fulltime” and experience_level=”senior_level”

   This can help refine the jobs so that we can grab more job postings despite the 100 pages limitation. It can also give us more information about the jobs.
4. With all combinations of the above mentioned query options, we can have a list of Indeed.com querying URLs. An example of the URL is: http://www.indeed.com/jobs?q=software+engineer&l=New+York+NY&jt=internship. Starting from each URL of the list, we can start crawl job list and descriptions from Indeed.com
5. To avoid high traffic load caused to Indeed.com, we plan to make the script sleep every 1-2 second for each request.

By assuming 5 common job titles, 80 main cities, with 4 job_type/experience level combinations, with each query have 400 jobs in average, we will get around 500 k jobs. Each job will have the following fields:

- job id (get from Indeed.com, as a primary key), String format
- job title (e.g. “C++ software engineer”), String format
- company (e.g. “Google”), String format
- location (e.g. “San Francisco, CA”), String format
- job type+experience_level (e.g. “fulltime+entry_level”), String format
- job description, String format

The dataset will be fixed as it’s hard to get real-time job postings from Indeed.com and it would also create heavy burden on network traffic.

There will be no need to preprocess the data after we crawled the data. But we may need to do some filterings, preprocess in the crawling script, e.g. check job id duplication before request to avoid unnecessary traffic.

We plan to save the crawled job postings in MongoDB database, with 2 basic collections:

- collection Jobs, containing the above mentioned fields

- collection Locations, containing the location (e.g. “San Francisco, CA” and its geographic information for visualization). The geographic information can be got by Google Map API.

4. Challenges

The challenges we have in this problem is related with how to extract useful information from job descriptions. Useful information extraction is a key part of our project. It’s useful both for statistical analysis and job keyword search. For statistical analysis, we want to know:

- what the skills are required for each job title and/or location and/or job experience level, so that we can get a high level understanding of the job market
- what the culture keywords of each company are, so that we can have a quick view of whether my personality match the company culture or not

For job keyword search, we want to search for jobs that match my skills and knowledge and then highlight these keywords for each job posting we get. These tasks all require us to find and cluster the keywords or key sentences from the full job description text.This is a nontrivial NLP (natural language processing) question. Though there are several research paper related with this topic[1][2], they may not work well on the our specific data and tasks. There must be a lot of trial and error work on finding/creating a useful method for information extraction.

5. Timeline:

Team timeline

Week 10/29 - 11/02	Data Crawling, Setting up AWS Services - S3, Lambdas, Spark
Week 11/02 - 11/09	Data Preprocessing, Literature Review of Deep Learning
Week 11/12 - 11/16	Creating our Machine Learning Model
Week 11/19 - 11/23	Testing and Validating our Machine Learning Model
Week 11/26 - 11/30	Learning Kibana and creating heat maps in Kibana
Week 12/03 - 12/07	Writing up Results

Individual tasks and timeline

	Si Shen	Rahul Chowdhury	Ganesh Chandra
Week 10/29 - 11/02	data crawling part. setup Spark/Mongo write crawling script, start crawling data preparation	Statistical Analysis of data - creating word clouds, basic statistical summary of data to aid the ML model	Deep neural network literature study.
Week 11/02 - 11/09	create basic stats / analysis based on MongoDB data	Deep neural network literature review	Design CNN layers of architecture based on literature study
Week 11/12 - 11/16	Learn Deep learning framework and work with teammates	Deep neural network initial model set up and training.	CNN training of the model.
Week 11/19 - 11/23	Learn Kinesis and work with teammates	Setting up Kibana and creating the basic visualisations	Setting up spark job, kinesis and backend
Week 11/26 - 11/30	Prepare final product for web development, write up results	Prepare final product for web development, write up results	Prepare final product for web development, write up results
Week 12/03 - 12/07			

References:

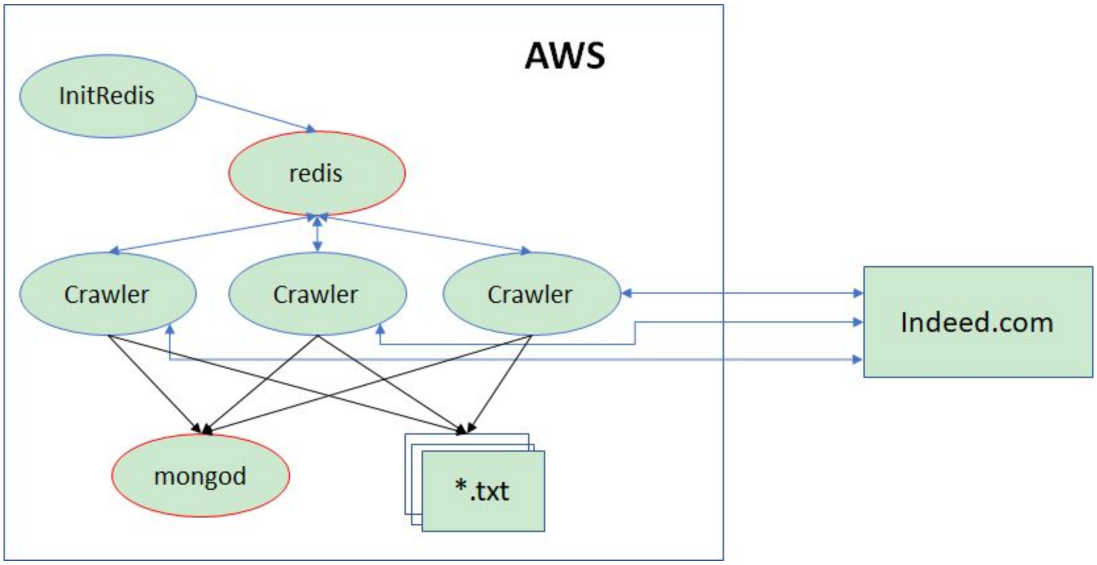
1. https://www.kdnuggets.com/2017/05/deep-learning-extract-knowledge-job-descriptions.html
2. https://arxiv.org/pdf/1707.09751.pdf
3. https://en.wikipedia.org/wiki/Indeed

CheckPoint 1

1. **Changes to Proposal**

(1) Crawler architecture is changed.

After further research online, we find Redis is more widely used in distributed crawling than Spark. So we build the architecture as follows:



In this architecture, Redis as a fast-access in-memory database, manages the URL queue to feed for each crawlers to work on. Redis also helps synchronize the job ids that are crawled already to avoid unnecessary duplications. All crawled job information will be saved in MongoDB and backed up as raw *.txt files.

(2) Dataset description is changed.

As we finished crawling, we get our dataset fixed now. The number of jobs we crawled is 334,507. We saved dataset in MongoDB on AWS EC2 instance. Below is the current fields of each document (job posting). We created indices for every field (except the long text "job_description" field) to allow fast query.

- "*_id*": job id, internal job id of Indeed.com
- "*query_keywords*": keyword used to query this job.
- "*query_location*": the location used to query this job.
- "*query_jobtyp_explvl*": job type and experience level used to query this job. We have 4 types here: "*internship*", "*fulltime+entry_level*", "*fulltime+mid_level*", "*fulltime+senior_level*"
- "*job_title*": the job_title shown in Indeed.com job list
- "*company*": company, shown in Indeed.com job list
- "*job_location*": location of the job, shown in Indeed.com job list
- "*job_description*": the long description of this job

2. New Timeline

Work that has been completed:

Si Shen

1. Developed the distributed crawling scripts and completed data crawling on AWS, including the following steps:
    - Wrote simple crawling scripts with Python/BeautifulSoup to crawl job postings starting from selected query keywords and query cities.
    - Integrated MongoDB into crawler to manage crawled data, and designed MongoDB schema
    - Integrated Redis into crawler to support multiple crawlers running on multiple EC2 instances to accelerate crawling

Note: According to the timeline, I was supposed to do create basic stats and analysis for MongoDB data. However I haven't got enough time to do that because:

a. It took much more time to crawl the data than I expected, and sometimes crawling stops because the disk space is run out..
b. After realizing single crawler was too slow, I spent some time to investigate distributed crawlings, including use of Redis.

Rahul Chowdhury

1. Statistical Analysis of data - creating word clouds, basic statistical summary of data to aid the ML model
    - Made word clouds for each job type - " internship", "fulltime+entry_level", "fulltime+mid_level", "fulltime+senior_level"
    - Made words clouds for each query word - "autonomous vehicle", "business analyst", "data analyst", "data engineer", "data scientist", "database administrator", "devops engineer", "embedded software engineer", "game software engineer", "machine learning", "network engineer", "product manager", "quality assurance", "robotics software engineer", "software engineer", "system administrator", "ui ux designer"
    - Basic count of each job type so that we know how to split our data for machine learning into training, validation and testing data.
    - Getting average count of words in job description for each job type. I think this will be useful information when building our predictive model.
2. Deep neural network literature review.

Note: According to the timeline, I was supposed to set up the deep learning model by this checkpoint. However, I was not able to do so as I busy with my course assistantship. I plan to pace up during this break and hopefully it should work out well as I have understood the working of the model as depicted in the literature review.

Ganesh Chandra Satish

1. Data Cleaning and basic analysis:
    - Cleaned data that was collected and stored in the mongodb to remove basic tags.
    - Performed basic analysis on the existing number of job types present in the collected database.
2. Design and develop the machine learning model:
    - Designed and started implementing the clustering algorithm to generate job title clusters. The idea is to create job embeddings(or clusters) that can help us understand which words are used more with other job description titles. Currently I am implementing a clustering algorithm, I have also generated the embedding using the job descriptions.
3. Deep neural network literature review:
    - I have also been researching on implementing a LSTM model on the job descriptions with titles as labels, as LSTM's work well with continuous data and keeps the context intact.

Note: According to the timeline, I was to implement the CNN model to predict the job title from the prediction. But as the idea of clustering is more relevant to the data and the purpose we have. I was instead working on developing the clustering algorithm.

Work that needs to be completed:

Si Shen

1. Research on deep learning framework and deep learning on AWS
2. Conduct deep learning training together with teammates
3. Work on visualization, web development, report writing, together with teammates

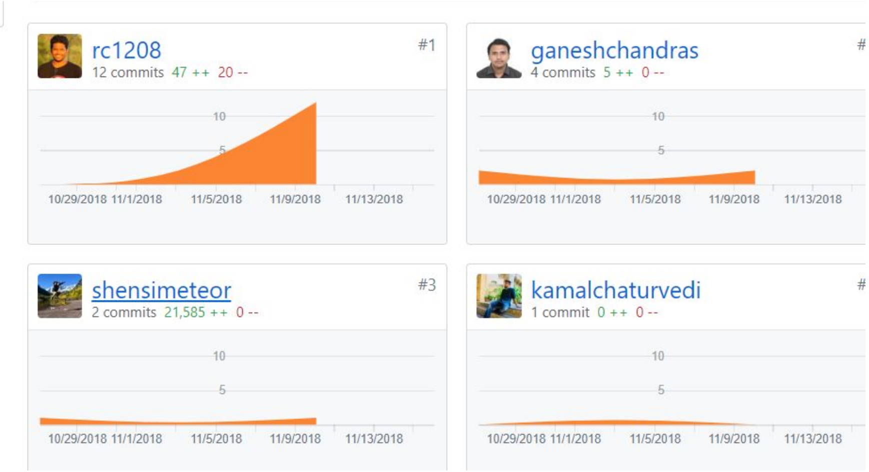
Rahul Chowdhury

1. Deep neural network initial model set up and training.
2. Setting up Kibana and creating the basic visualisations.
3. Prepare final product for web development, write up results

Ganesh Chandra Satish

1. Finish developing the ML model using wither clustering or LSTM
2. Setting up the backend architecture on aws

Github contributions:



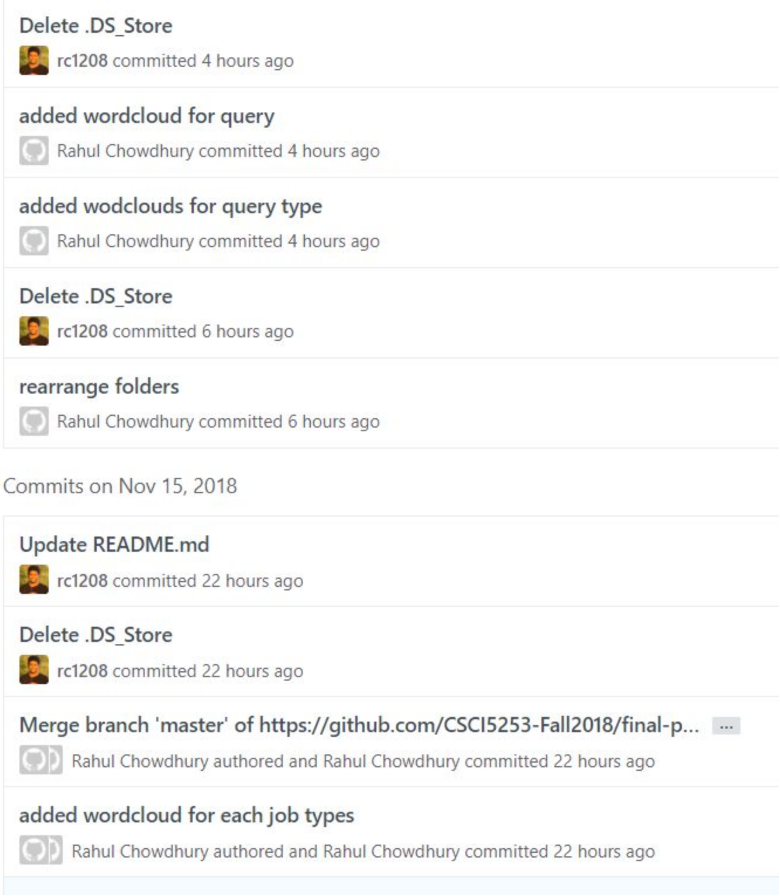
rc1208 #1
12 commits 47 ++ 20 --

ganeshchandras #
4 commits 5 ++ 0 --

shensimeteor #3
2 commits 21,585 ++ 0 --

kamalchaturvedi #
1 commit 0 ++ 0 --

Github check-ins:



Delete .DS_Store
rc1208 committed 4 hours ago

added wordcloud for query
Rahul Chowdhury committed 4 hours ago

added wodclouds for query type
Rahul Chowdhury committed 4 hours ago

Delete .DS_Store
rc1208 committed 6 hours ago

rearrange folders
Rahul Chowdhury committed 6 hours ago

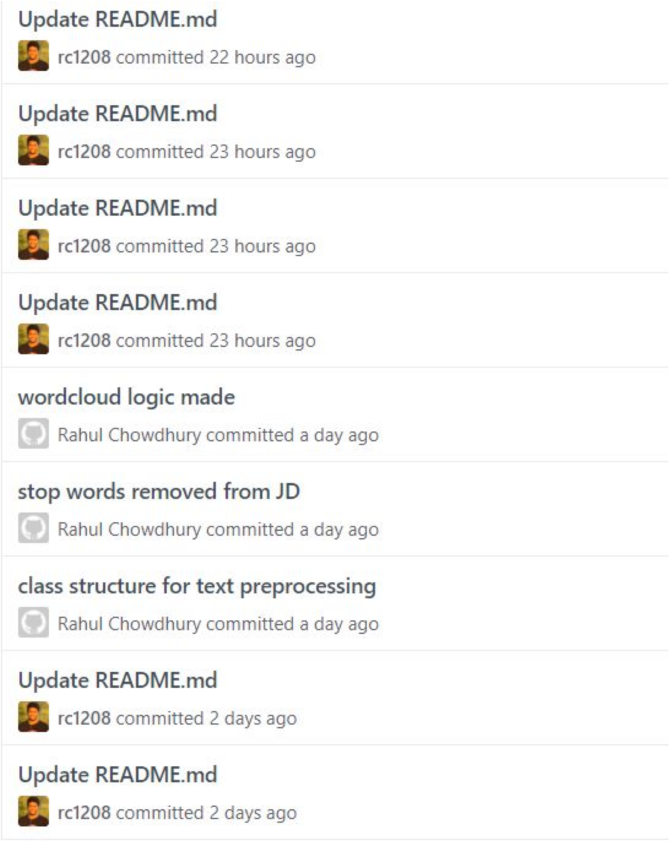
Commits on Nov 15, 2018

Update README.md
rc1208 committed 22 hours ago

Delete .DS_Store
rc1208 committed 22 hours ago

Merge branch 'master' of https://github.com/CSCI5253-Fall2018/final-p... ...
Rahul Chowdhury authored and Rahul Chowdhury committed 22 hours ago

added wordcloud for each job types
Rahul Chowdhury authored and Rahul Chowdhury committed 22 hours ago



Update README.md
rc1208 committed 22 hours ago

Update README.md
rc1208 committed 23 hours ago

Update README.md
rc1208 committed 23 hours ago

Update README.md
rc1208 committed 23 hours ago

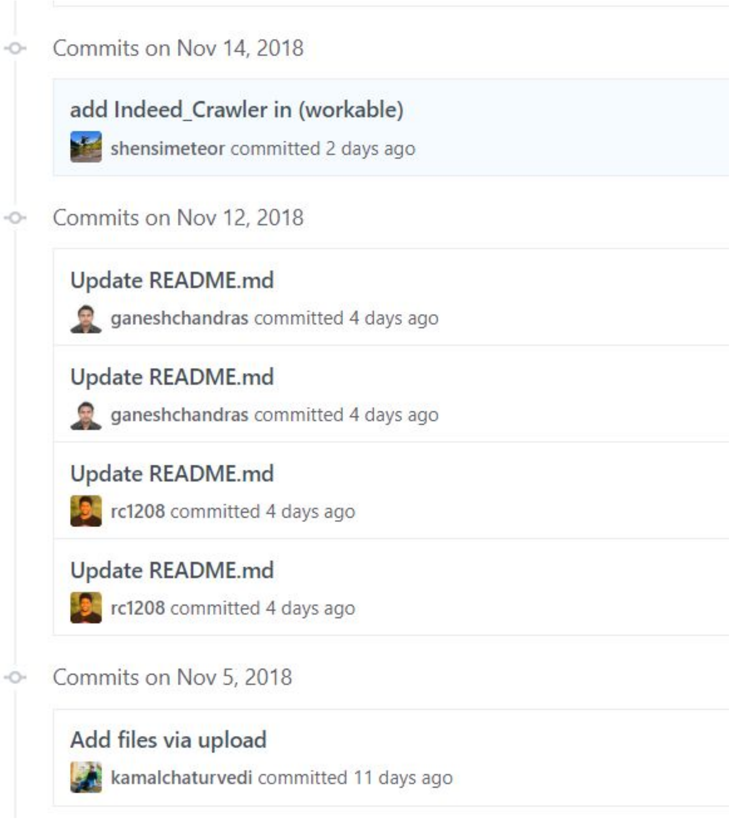
wordcloud logic made
Rahul Chowdhury committed a day ago

stop words removed from JD
Rahul Chowdhury committed a day ago

class structure for text preprocessing
Rahul Chowdhury committed a day ago

Update README.md
rc1208 committed 2 days ago

Update README.md
rc1208 committed 2 days ago



Commits on Nov 14, 2018

add Indeed_Crawler in (workable)
shensimeteor committed 2 days ago

Commits on Nov 12, 2018

Update README.md
ganeshchandras committed 4 days ago

Update README.md
ganeshchandras committed 4 days ago

Update README.md
rc1208 committed 4 days ago

Update README.md
rc1208 committed 4 days ago

Commits on Nov 5, 2018

Add files via upload
kamalchaturvedi committed 11 days ago

3. Costs

Task	Cloud Service	Cost ($)
Crawler, Mongod, Redis	EC2 instances	0 (still under free tier limit now)
Routine backup of crawled data	S3	2 (after realizing the potential high cost, S3 backup plan is stopped immediately)

The estimated costs for future development: $10-50 , depending on how complex our machine learning or deep learning method is, and how long the training takes.

4. Dataset Management

Dataset Size	We have a huge dataset(300k jobs) and this might be a factor to consider carefully when we train our deep learning NLP models. We look forward to leverage the power of cloud, primarily AWS for better computation resources.
Job description stop words removal	Job description of any job is filled with stop words which need to be essentially removed before we build our NLP models. We did it by removing the stop words from the list provided in our earlier projects.

Outstanding work that needs to be done:

Job description redundant information	One thing we realized while after creating the word clouds was that we get a lot of words such as “equal”, “color”,”religion” which are very frequent in number but might not help us when we try to map a candidate to a job application. We will have to think on how to extract skills from a job description.