



ASSIGNMENT SUBMISSION FORM

This will be the first page of your assignment

Course Name: AMPBA Batch 16 (Summer)
Assignment Title: Foundation Project 1 (Group 06) – “Trending Jobs”
Submitted by: Group 06

(Student name or group names)

Student Name	PG ID
Anil Jhamb	12110076
Chitra Nair	12110058
Deepika Rawat	12110074
Namrata Nagaraj	12110045
Rohini Singh	12110094
Virendra Singh Shekhawat	12110040

ISB Honour Code

- I will represent myself in a truthful manner.
- I will not fabricate or plagiarise any information with regard to the curriculum.
- I will not seek, receive or obtain an unfair advantage over other students.
- I will not be a party to any violation of the ISB Honour Code.
- I will personally uphold and abide, in theory and practice, the values, purpose and rules of the ISB Honour Code.
- I will report all violations of the ISB Honour Code by members of the ISB community.
- I will respect the rights and property of all in the ISB community.
- I will abide by all the rules and regulations that are prescribed by ISB.

Note: Lack of awareness of the ISB Honour Code is never an excuse for a violation. Please go through the Honour Code in the student handbook, understand it completely. Please also pay attention to the following points:

- Please do not share your assignment with your fellow students under any circumstances if the Honour Code scheme prohibits it. The HCC considers both parties to be guilty of an Honour Code violation in such circumstances.
- If the assignment allows you to refer to external sources, please make sure that you cite all your sources. Any material that is taken verbatim from an external source (website, news article etc.) must be in quotations. A much better practice is to paraphrase the source material (it still must be cited).

(Please start writing your assignment below)

Trending Jobs

2022

Foundation Project – I
AMPBA 2021-22 (Summer)

Anil Jhamb – 12110076
Chitra Nair – 12110058
Deepika Rawat – 12110074
Namrata Nagraj – 12110045
Rohini Singh – 12110094
Virendra Singh Shekhawat – 12110040

Contents -

1. Business understanding	3
1.1 Determine business objectives.....	3
1.3 Set data mining goals	3
1.4 Project plan	3
2. Data understanding	4
2.1 Collect data.....	4
2.2 Data description	4
3. Data preparation	5
3.1 Clean data (User Profile)	5
3.1 Create new columns (User Profile)	5
3.1 Format data (User Profile)	5
3.2 Clean data (Job Description)	6
3.2 Create new columns (Job Description)	6
3.2 Format data (User Profile)	6
4. Modeling	7
4.1 Select modeling technique	7
4.2 Build model	7
4.3 Assess model	7
5. Evaluation	8
5.1 Evaluate results	8
6. Deployment	8
6.1 Deployment Plan	8
7. Appendix	9
7.1 Constraints	9
7.2 Data References	9

Business understanding –



Background –

The Pandemic has changed our working culture and has affected our professional careers. We have seen a lot of job cuts in some of the major IT companies and likewise in all the sectors. The employee churn has been at the peak due to the Coronavirus effect. HR teams have an additional responsibility to make sure employee retention is maintained and new hiring is done at a minimal cost.

1.1 Determining Business Objective –

The objective of this project is to minimize the cost of recruiting a candidate for the company. You have to scrape through the profiles which are publicly available on various job portals, and using NLP techniques extract the features from the resumes. Then arrange them in a structured form to create a database of required skill sets. Later develop a resume Automatic Tracking System (ATS) model which maps these profiles with the Job description using a score. Finally, with the help of the model, predict the candidate who would accept the offer.

1.2 Set data mining goals –

While starting the project, we had 3 options in our discussions. One was to find a source to scrap data, 2nd was to crowdsource and 3rd was to read data from resume docs and pdfs.

1.3 Project plan –

To start with, plan is to find a reliable and publicly available user resume profiles and scrap them. Once scrapped, filter through records and create a workable data-frame. Once we have a data-frame, we will cleaning/cleansing the data. We will be using 'nltk' techniques to remove stop-words and filtering emails and skills from it. We will take some job description and will create "Term Matrix" from both job description and user profiles. Once we have 2 vectors, we will match their similarity. We will get top 10 matches and to evaluate we will calculate their mean precision.

Data understanding –



2.1 Collection of data (For User Profiles) –

We used <https://www.postjobfree.com> as our data source. For data crawling we used ‘BeautifulSoup’ and crawled through 50 pages recursively and collected them in a data frame. The result included Name, Resume and profile link. Once data is collected, we saved the records in a CSV file.

```
def data_scrap(base_url, p, links):
    parse_url = base_url+"&p="+str(p)
    response = requests.get(parse_url)
    soup = BeautifulSoup(response.content, 'html.parser')

    title_tags = soup.find_all('h3', attrs={'class': 'itemTitle'})

    for title_tag in title_tags:
        links.append("https://www.postjobfree.com"+title_tag.a['href'])

    if(p<50):
        p = p+1
        time.sleep(3)
        data_scrap(base_url, p, links)

    return links

def data_collection(url):
    links = []
    parse_links = data_scrap(url,1,links)
    data_results = []
    for link in parse_links:
        res = requests.get(link)
        print(res.status_code, link)
        content = BeautifulSoup(res.content, 'html.parser')

        nameDiv = content.find('div', attrs={'class': 'normalText'}).findAll('p')[0].text
        if(nameDiv.lower() == 'resume' or nameDiv.lower() == 'curriculum vitae' or nameDiv.lower() == '**/**/**' or nameDiv.lower() == 'profile'):
            nameDiv = content.find('div', attrs={'class': 'normalText'}).findAll('p')[1].text

        resume = content.find('div', attrs={'class': 'normalText'}).get_text()[:-23],

        collection_dict = []
        collection_dict = [nameDiv, resume, link, '', '']
        data_results.append(collection_dict)
        time.sleep(3)
    return data_results

results = data_collection("https://www.postjobfree.com/resumes?l=India&radius=25")
profile_df = pd.DataFrame(results, columns=['Name', 'resume', 'Profile Link', 'Skills'])
profile_df.head()
```

2.2 Data description (For Profiles) –

Post scrapping, we had 3 columns (data type - obj) - Name, Resume and Profile Link. Skills column was added for future use.

```
Name          object
resume         object
Profile Link   object
Skills         object
dtype: object
```

Data Preparation –



3.1 Clean Data / Create New Column(s) / Format Data –

Now that we had the scrapped data, the content we fetched is an unstructured data. We used ‘nltk’ to extract email from resume and also to remove stop words and tokenized the text. Then we created skill vectors using tokens and N-gram techniques. Once we have skill list, saved them in the csv.

```
# Libraries
import nltk
import re
from nltk.tokenize import word_tokenize
from collections import Counter
from nltk.corpus import stopwords
import string

class DataPreProcessing:
    def __init__(self, df):
        skill_set = []
        skill_data = pd.read_csv('skills.csv')
        for i,r in skill_data.iterrows():
            skill_set.append(r['skill'])

        self.dataframe = df
        self.skill_set = skill_set

    def get_email_entity(self):
        email_reg = re.compile(r'[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z0-9\.\-+_]+\.[a-z]+')
        user_email = []
        for index,row in self.dataframe.iterrows():
            user_email.append(re.findall(email_reg, row['resume']))
        return user_email

    def get_user_skills(self):
        skill_list = []
        stop_words = set(stopwords.words('english'))
        for index,row in self.dataframe.iterrows():
            tokens = word_tokenize(row['resume'])
            f_tokens = [w for w in tokens if w not in stop_words]
            f_tokens = [w for w in tokens if w.isalpha()]
            ngram_match = list(map(' '.join, nltk.everygrams(f_tokens, 2, 3)))

            skill_results = []

            for token in f_tokens:
                if token.lower() in self.skill_set:
                    skill_results.append(token)

            for ng in ngram_match:
                if ng.lower() in self.skill_set:
                    skill_results.append(ng)

            skill_list.append(skill_results)

        return skill_list

data_df = DataPreProcessing(profile_df)

skills = data_df.get_user_skills()
profile_df['Skills'] = skills

emails = data_df.get_email_entity()
profile_df['Email'] = emails
```

3.2 Clean Data / Create New Column(s) / Format Data (Job description) –

When we downloaded job description sheet, that was also an unstructured data so we decided to use same techniques as we used for user profiles. So, we used 'nltk' for extracting skill set from the text and created a new column as 'bag_of_words'.

```
class DataPreProcessing_jd:
    def __init__(self, df):
        skill_set = []
        skill_data = pd.read_csv('skills.csv')
        for i,r in skill_data.iterrows():
            skill_set.append(r['skill'])

        self.dataframe = df
        self.skill_set = skill_set

    def get_user_skills(self):
        skill_list = []
        stop_words = set(stopwords.words('english'))
        for index,row in self.dataframe.iterrows():
            tokens = word_tokenize(row['job_description'])
            f_tokens = [w for w in tokens if w not in stop_words]
            f_tokens = [w for w in tokens if w.isalpha()]
            ngram_match = list(map(' '.join, nltk.everygrams(f_tokens, 2, 3)))

            skill_results = []

            for token in f_tokens:
                if token.lower() in self.skill_set:
                    skill_results.append(token)

            for ng in ngram_match:
                if ng.lower() in self.skill_set:
                    skill_results.append(ng)

            skill_list.append(skill_results)
        return skill_list

#job_queries = pd.read_csv('job_List.csv')
df_jd = DataPreProcessing_jd(job_queries)
keywords = df_jd.get_user_skills()
job_queries['bag_of_words'] = keywords

job_queries
```

Unnamed: 0			job_description	bag_of_words
0	300	Needs someone that is a self starter, can work...	[accounting, Tax, Tax, Word, tax, Email, Micro...	
1	301	As the manager of copywriting on the Walden Un...	[marketing, presentations, marketing, content,...	
2	302	Bojangles' is expanding and opening new locati...	[Scheduling, Inventory, compliance, Communicat...	
3	303	The Enrollment Agent I (EA I) supports Enrollm...	[Operations, hardware, database, policies, sec...	
4	304	Progressive Design, Inc. is a Heavy Industrial...	[Design, Design, Consulting, AutoCAD, AutoCAD,...	
...
195	495	The Judge Group is actively seeking a QA Analy...	[testing, migration, test plans]	
196	496	Report this job About the Job Work Schedule: ...	[Schedule, schedules, Transportation, schedule...	
197	497	Report this job About the Job We are a locally...	[Operations, Operations, Routing, Scheduling, ...	
198	498	Security Officer - Northern Kentucky/OTR 3861B...	[Security, SECURITY, security, security, Hospi...	
199	499	Job Description Position Summary: Provides pro...	[hospital, testing, testing, training, trainin...	

200 rows × 3 columns

Modeling –



4.1 Select Modeling Techniques –

After trying all possible ways to create an efficient model, we decided to go for cosine similarity. We thought of creating term vectors and then comparing job vector with profile vector and compare their similarity and on the basis of them get top records.

4.2 Build Model –

We created an Information retrieval system to get highest matching data by comparing vectors.

```
class RecommendationModel:
    def __init__(self, skills, df):
        self.dataframe = df
        self.skill_set = skills

    def recommendation_vectors(self):
        final_list = []
        counter1 = Counter(self.skill_set)
        for index, row in self.dataframe.iterrows():
            if(len(row['skills']) > 0):
                counter2 = Counter(row['skills'])
                all_items = set(counter1.keys()).union( set(counter2.keys()) )

                vector1 = [counter1[k] for k in all_items]
                vector2 = [counter2[k] for k in all_items]

                similarity = 1 - spatial.distance.cosine(vector1, vector2)
                if(similarity > 0):
                    #similarity = cluster.util.cosine_distance(vector1,vector2)
                    new_dict = [row['Name'], row['Profile Link'], row['skills'], similarity]
                    final_list.append(new_dict)
        return final_list

#job_queries = pd.read_csv("job_list.csv")

model_class = RecommendationModel(job_queries.iloc[7]['bag_of_words'], profile_df)
recommendation_df = model_class.recommendation_vectors()

print("Job description -- "+job_queries.iloc[7]['job_description'])
print("--")
print(job_queries.iloc[7]['bag_of_words'])

recommendation_df = sorted(recommendation_df, key=lambda x: x[3], reverse=True)
filtered_profiles = pd.DataFrame(recommendation_df, columns=['Name', 'profile_link', 'skills', 'similarity'])
relevant_profiles = filtered_profiles.head(10)
relevant_profiles
```

4.3 Assess Model –

To assess the model, we randomly checked parsing job description and manually assessed if the returned response was correct or not and how similar the vector was computing them.

Evaluation –



5.1 Evaluate Results –

To evaluate we randomly chose JDs randomly and calculated mean average precision of the responses. And we checked if the retrieved results were upto the mark or not.

```
sample = job_queries.sample(n=10)
vector_similarity = []
for index, rows in sample.iterrows():
    model_class = recommendationModel(rows['job_description'], profile_df)
    recommendation_df = model_class.recommendation_vectors()
    for row in recommendation_df:
        vector_similarity.append(row[3])

vector_similarity = sorted(vector_similarity, reverse = True)[:10]
print('Mean Average Precision=>', np.mean(vector_similarity))
```

Mean Average Precision=> 0.12762012256187502

Deployment –



6.1 Deployment Plan –

For deployment we chose to go with FLASK app to deploy in local machine.

Employee Search Model

This model is to search employees who are most likely to accept a job profile.

Job Description

Submit

Name	profile_link	similarity
V.Ajay	https://www.postobfrees.com/resume/ador07jally-vista-liruvalur-tamil	0.5499719405228703
Akash Shah	https://www.postobfrees.com/resume/ador04chartered-accountant-as-vadodara-gujarat	0.5451842731386104
Ramesh M. A.	https://www.postobfrees.com/resume/ador06co-ld-thysoor-karnataka	0.4871794871794872
P.Brinwasara	https://www.postobfrees.com/resume/ador05accountant-jally-hydrabad-india	0.42700841014889605
D. Shankar Kumar	https://www.postobfrees.com/resume/ador03an/accountant-jally-coimbatore-tamil	0.340206087198858
Kuldeep Verma	https://www.postobfrees.com/resume/ador06w/accountant-jally-delhi-india	0.3299631645537221
Pankaj	https://www.postobfrees.com/resume/ador02z/account-officer-pune-maharashtra	0.3187276291558383
Deeksha Gaur	https://www.postobfrees.com/resume/ador03nback-office-manager-new-delhi-india	0.316227766016838
Sk Saddam Hossain	https://www.postobfrees.com/resume/ador04lgoa-sk-new-delhi-india	0.2721655269759087
Ashu Kumar	https://www.postobfrees.com/resume/ador04sk/supervisor-store-dewa-sitar-india	0.2872612419124244

Appendix-

7.1 Constraints –

- (a) Social media sources like LinkedIn etc. couldn't be exploited fully due to privacy constraints. However, open-source websites were used to get suitable data set for building the model.
- (b) Our data was unstructured so deep data cleaning was done before utilizing the same for further processing. Apart from outliers, redundant data fields like website's email address in place of candidate's email address etc., needed to be removed.
- (c) Crowdsourcing didn't give a usable dataset thus sources from open source were only used for data input.

7.2 Data References –

- (a) **For Data Scrapping -**
<https://www.postjobfree.com>
- (b) **For Job descriptions -**
<https://www.kaggle.com/residentmario/exploring-monster-com-job-postings/data>
- (c) **For Skill sets -**
<https://www.kaggle.com/arbazkhan971/allskillandnonskill>