Upgrade

Building NLP Content-Based Recommender Systems

A tutorial for a NLP recommendation engine using unsupervised learning

Armand Olivares Follow
Jul 7 · 8 min read

Let's understand how to do an approach for build recommender systems when you have text data.

Introduction

In this post we will be using datasets hosted by Kaggle and considering the content-based approach, we will be building **job recommendation systems**.

1. Getting Ready

For this post we will need **Python 3.6, Spacy, NLTK and scikit-learn,** If you do not have it yet, please install all of them.

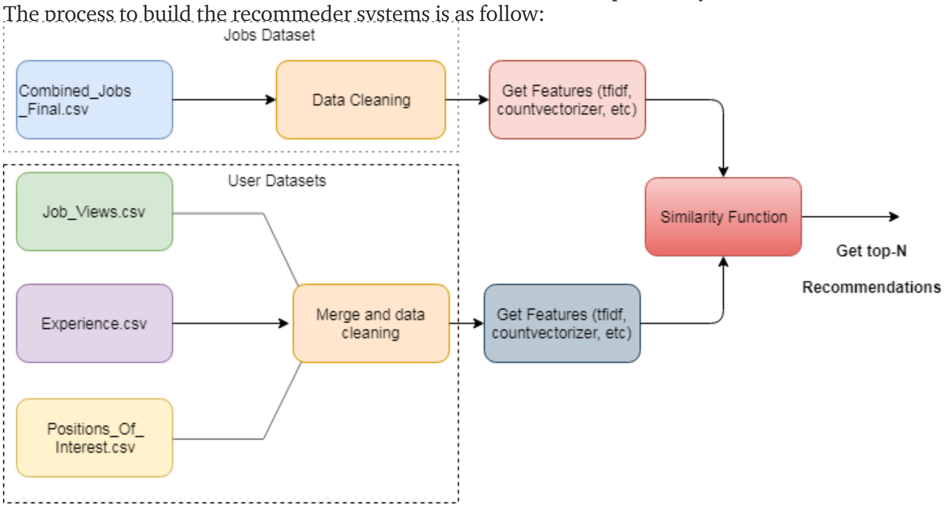
2. The proccess

You highlighted

Here, we are using the data from this challenge on kaggle . The 4 datasets are as follows:

- The Combined_Jobs_Final.csv file: has the main jobs data(title, description, company, etc.)
- The Job_Views.csv file: the file with the jobs seeing for the user.
- The Experience.csv: the file containing the experience from the user.
- The Positions_Of_Interest.csv: contains the interest the user previously has manifested.

The process to build the recommeder systems is as follow:



The process start by cleaning and building the datasets, then get the numerical features from data, after that we will apply a similarity function(*cosine similarity* for example) to get the similarity between **previous user jobs** or jobs which user has manifested interest **and the availables jobs**, finally get the top recommend jobs according to the score of the similarity.

You highlighted

2.1 Building the Datasets

In every data project, the fist step is to explore and clean the data we have, also as there are 4 dataset we are going to merge them in order to have **1** dataset for **jobs**, and **1** dataset for **users.**

2.1.1 for jobs Dataset:

Reading the data and get the info about it

```
df_jobs = pd.read_csv("Combined_Jobs_Final.csv")
df_jobs.info()
```

```
Provider             84090 non-null int64
Status               84090 non-null object
Slug                 84090 non-null object
Title                84090 non-null object
Position             84090 non-null object
Company              81819 non-null object
City                 83955 non-null object
State.Name           83919 non-null object
State.Code           83919 non-null object
Address              36 non-null object
Latitude             84090 non-null float64
Longitude            84090 non-null float64
Industry             267 non-null object
Job.Description      84034 non-null object
Requirements         0 non-null float64
Salary               229 non-null float64
Listing.Start        83407 non-null object
Listing.End          83923 non-null object
Employment.Type      84080 non-null object
Education.Required   83823 non-null object
Created.At           84090 non-null object
Updated.At           84090 non-null object
dtypes: float64(4), int64(2), object(17)
memory usage: 14.8+ MB
```


As we can see there are 23 columns, however for this article we only will use *'Job.ID', 'Title', 'Position', 'Company', 'City', 'Job_Description'.*
Then as part of the preprocessing we:

1. imputed the missing values if any.
2. remove stop words.
3. remove not alphanumeric characters.
4. lemmatize the columns.
5. finally we will merge all the columns in order to create a corpus of text for each job.

We put the step 2–5 into a function called "clean_txt":

```
def clean_txt(text):
  clean_text = []
  clean_text2 = []
  text = re.sub("'", "",text)
  text=re.sub("(\\d|\\W)+"," ",text)
  clean_text = [ wn.lemmatize(word, pos="v") for word in word_tokenize(text.lower()) if black_txt(word)]
  clean_text2 = [word for word in clean_text if black_txt(word)]
  return " ".join(clean_text2)
```

After made steps 1–5 we ended with a clean dataset with 2 columns: *Job.ID* and *text* (the corpus of the data) as we can see:

Job.ID	text
111	server tacolicious palo alto part time tacolic...
113	kitchen staff chef claude lane san francisco p...
117	bartender machka restaurants corp san francisc...
121	server teriyaki house brisbane part time serve...
127	kitchen staff chef rosa mexicano sunset los an...

2.1.2 for users Dataset:

For the "jobs_views" dataset:

```
df_job_view = pd.read_csv("Job_Views.csv")
df_job_view.head(2)
```

	Applicant.ID	Job.ID	Title	Position	Company	City	State.Name	State.Code	Industry	View.Start	View.End	View.Duration	Created.At	Updated.At
0	10000	73666	Cashiers & Valets Needed! @ WallyPark	Cashiers & Valets Needed!	WallyPark	Newark	New Jersey	NJ	NaN	2014-12-12 20:12:35 UTC	2014-12-12 20:31:24 UTC	1129.0	2014-12-12 20:12:35 UTC	2014-12-12 20:12:35 UTC
1	10000	96655	Macy's Seasonal Retail Fragrance Cashier - Ga...	Macy's Seasonal Retail Fragrance Cashier - Ga...	Macy's	Garden City	New York	NY	NaN	2014-12-12 20:08:50 UTC	2014-12-12 20:10:15 UTC	84.0	2014-12-12 20:08:50 UTC	2014-12-12 20:08:50 UTC

In this case we will use only the columns 'Applicant.ID', 'Job.ID', 'Position', 'Company','City', we select the columns and applying the *clean_txt* function we ended with an Id columns and a text column:

```
df_job_view = df_job_view[['Applicant.ID', 'Job.ID', 'Position', 'Company','City']]
df_job_view["select_pos_com_city"] = df_job_view["Position"].map(str) + "  " + df_job_view["Company"] +"  "+ df_job_view["City"]
df_job_view['select_pos_com_city'] = df_job_view['select_pos_com_city'].map(str).apply(clean_txt)
df_job_view['select_pos_com_city'] = df_job_view['select_pos_com_city'].str.lower()
df_job_view = df_job_view[['Applicant.ID','select_pos_com_city']]
df_job_view.head()
```

	Applicant.ID	select_pos_com_city
0	10000	cashier valet need wallypark newark
1	10000	macys seasonal retail fragrance cashier garden...
2	10001	part time showroom sales cashier grizzly indus...
3	10002	event specialist part time advantage sales mar...
4	10002	bonefish kitchen staff bonefish grill greenville

	Applicant.ID	Position.Name	Employer.Name	City	State.Name	State.Code	Start.Date	End.Date	Job.Description	Salary	Can.Contact.Employer	Created.
0	10001	Account Manager / Sales Administration / Quali...	Barcode Resourcing	Bellingham	Washington	WA	2012-10-15	NaN	NaN	NaN	NaN	2014-12- 20:10: U
1	10001	Electronics Technician / Item Master Controller	Ryzex Group	Bellingham	Washington	WA	2001-12-01	2012-04-01	NaN	NaN	NaN	2014-12- 20:10: U
2	10001	Machine Operator	comptec inc	Custer	Washington	WA	1997-01-01	1999-01-01	NaN	NaN	NaN	2014-12- 20:10: U

For this file we only use the *Position.Name* and the *Applicant.Id*, we select the columns and clean the data. we ended we an ID and a text column:

```
#taking only Position
df_experience= df_experience[['Applicant.ID','Position.Name']]
#cleaning the text
df_experience['Position.Name'] = df_experience['Position.Name'].map(str).apply(clean_txt)
df_experience.head()
```

You highlighted

	Applicant.ID	Position.Name
0	10001	account manager sales administration quality a...
1	10001	electronics technician item master controller
2	10001	machine operator
3	10003	maintenance technician
4	10003	electrical helper

for the *position of interest* dataset:

```
#Position of interest
df_poi =  pd.read_csv("Positions_Of_Interest.csv", sep=',')
df_poi = df_poi.sort_values(by='Applicant.ID')
df_poi.head()
```

	Applicant.ID	Position.Of.Interest	Created.At	Updated.At
6437	96	Server	2014-08-14 15:56:42 UTC	2015-02-26 20:35:12 UTC
1156	153	Barista	2014-08-14 15:56:43 UTC	2015-02-18 02:35:06 UTC
1155	153	Host	2014-08-14 15:56:42 UTC	2015-02-26 20:35:12 UTC
1154	153	Server	2014-08-14 15:56:42 UTC	2015-02-26 20:35:12 UTC
1158	153	Sales Rep	2014-08-14 15:56:47 UTC	2015-03-02 02:13:08 UTC

We are going to select *Position.Of.Interest and Applicant.ID,* we clean the data and ended with an Id column and a text column:

```
df_poi['Position.Of.Interest']=df_poi['Position.Of.Interest'].map(str).apply(clean_txt)
df_poi = df_poi.fillna(" ")
df_poi.head(10)
```

	Applicant.ID	Position.Of.Interest
6437	96	server
1156	153	barista
1155	153	host
1154	153	server
1158	153	sales rep
1157	153	customer service rep
1952	256	host
1957	256	production area
1956	256	sales rep

Finally we **merge** the 3 dataset by the column **Applicant.ID**, the final dataset for user look like:

```
df_final_person.head()
```

	Applicant_id	text
0	2	volunteer writer uloop blog
1	3	market intern server prep cook
2	6	project assistant
3	8	deli clerk server cashier food prep order tak...
4	11	cashier

The code for tf-idf:

```
#initializing tfidf vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

tfidf_jobid = tfidf_vectorizer.fit_transform((df_all['text'])) #fitting and transforming the vector
tfidf_jobid
```

```
<84090x50755 sparse matrix of type '<class 'numpy.float64'>'
	with 8283370 stored elements in Compressed Sparse Row format>
```

Please refers to this page for check more about tfidf implementation.
For CountVectorizer:

```
from sklearn.feature_extraction.text import CountVectorizer
count_vectorizer = CountVectorizer()

count_jobid = count_vectorizer.fit_transform((df_all['text'])) #fitting and transforming the vector
count_jobid
```

```
<84090x50755 sparse matrix of type '<class 'numpy.int64'>'
	with 8283370 stored elements in Compressed Sparse Row format>
```

Please refers to this page for check more about count vectorizer implementation.

3. Recommender Systems

As this application has more textual data and there are no ratings available for any job, **we are not using** other matrix decomposition methods, such as SVD, or correlation coefficient-based methods, such as Pearsons'R correlation.
So we are only use content based filtering will show us how we can recommend items to people just based on the attributes of the items themselves.
In this post we are building 4 recommenders systems:

1. Content based Recomender with tfidf
2. Content Based Recomender with CountVectorizer
3. Content Based Recomender with Spacy
4. Content Based Recomender with KNN

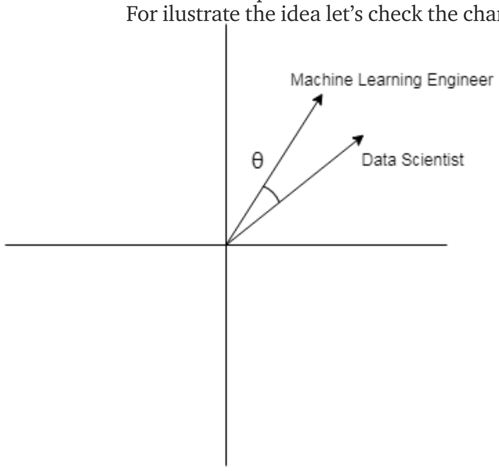
Let's start by thinking about how to measure the similarity between two jobs descriptions because we must find some sort of similarity measure that looks at how many in common have them.

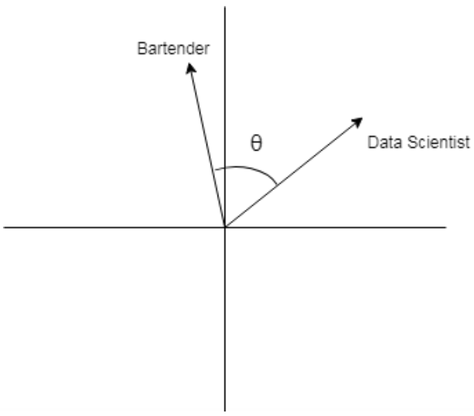
So what's a good way of doing that mathematically?:

Cosine similarity

3.1 Cosine Similarity

Is the most common metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.
For ilustrate the idea let's check the charts below:





The jobs *machine learning enginner* and *data Scientist* are quite similar so θ close to *0°* (zero) and cos(θ) close to 1.

(close to 1 more similar items)
The jobs *bartender* and *data Scientist* are not similar so θ close to 90° and cos(θ) close to *0*(zero)
The general idea for this case is if the cosine is close to 1 the items are similar, if is close to 0 not similar, there is another case when cosine equal to -1 meaning similar but oposite items.
Please refer to this link tho review more about cosine similarity.

3.2 Content based Recomender with tfidf

For calculate the cosine similarity in python we will use **cosine_similarity** from **sklearn package**, the following code for a given user's job ilustrated that.
Using tfidf:

M

Upgrade

```
#computing cosine similarity of user with job corpus
from sklearn.metrics.pairwise import cosine_similarity
user_tfidf = tfidf_vectorizer.transform(user_q['text'])
cos_similarity_tfidf = map(lambda x: cosine_similarity(user_tfidf, x),tfidf_jobid)
```

In this, scores **close to one** means **more similarity** between items.

3.3 Content Based Recomender with CountVectorizer

using countvectorizer:

```
from sklearn.metrics.pairwise import cosine_similarity
user_count = count_vectorizer.transform(user_q['text'])
cos_similarity_countv = map(lambda x: cosine_similarity(user_count, x),count_jobid)
```

Again, scores **close to one** means **more similarity** between items.

3.4 Content Based Recomender using Spacy

For this we are not using cosine similarity but we will using pre-trained word vectors in Spacy, which can help to get better results, to compute similarity between the text.

First, for each text in jobs we need to build an *spacy doc*:

```
list_docs = []
for i in range(len(df_all)):
  if df_all['text'][i] != '':
    doc = nlp("u'" + df_all['text'][i] + "'")
    list_docs.append(doc)
```

Then we use the **spacy's similarity function**, which constructs sentence embedding by averaging the word embeddings and computing the similarity, the function below compute the similarity:

```
def calculateSimWithSpaCy(nlp, df, user_text, n=6):
    # Calculate similarity using spaCy
    list_sim =[]
    doc1 = nlp("u'" + user_text + "'")
    for i in df.index:
      try:
            doc2 = list_docs[i]
            score = doc1.similarity(doc2)
            list_sim.append((doc1, doc2, score))
          except:
            continue

    return  list_sim
```

For spacy similarity, scores **close to one** means **more similarity** between items.

3.5 KNN Recomender System

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

The code below, compute the 10 nearest neighbors for a given user job, using **tfidf** as features:

```
from sklearn.neighbors import NearestNeighbors
n_neighbors = 11
KNN = NearestNeighbors(n_neighbors, p=2)
KNN.fit(tfidf_jobid)
NNs = KNN.kneighbors(user_tfidf, return_distance=True)
```

This is a particular case which scores **close to zero** means **more similarity** between items.

4 Evaluating the recomendations

As we build recommendations systems using TF-IDF, count vectorizer, cosine similarity, spacy etc, i.e using mainly text data and because there is not predefined testing matrix available for generating the accuracy score we need to check our **recommendations relevance manually.**

For test all the recommenders we selected **random users** from the user dataset:

```
#taking a user
u = 326
index = np.where(df_final_person['Applicant_id'] == u)[0][0]
user_q = df_final_person.iloc[[index]]
user_q
```

You highlighted

	Applicant_id	text
186	326	java developer

As you can see we selected the user with **Applicant_id 326**, and text corpus related to **java developer,** let's check the recommendations for this user:

4.1 Using TFIDF

The results:

JobID	title	score
294684	Java Developer @ Kavaliro	0.740035
269922	Entry Level Java Developer / Jr. Java Develope...	0.737007
303112	Java Developer @ TransHire	0.734574
141831	Lead Java/J2EE Developer - Contract to Hire @ ...	0.671667
270171	Senior Java Developer - Contract to Hire - Gre...	0.645037

245753	Java Administrator @ ConsultNet	0.530231
146640	Jr. Java Developer @ Paladin Consulting Inc	0.510534
150882	Java Consultant - Mobile Apps Development @ Co...	0.486789
251696	Java Developer @ ConsultNet	0.414257

The recommendation looks pretty good based in the data we have.

4.2 Using CountVectorizer

JobID	title	score
294684	Java Developer @ Kavaliro	0.59588
269922	Entry Level Java Developer / Jr. Java Develope...	0.571726
303112	Java Developer @ TransHire	0.559017
141831	Lead Java/J2EE Developer - Contract to Hire @ ...	0.496907
270171	Senior Java Developer - Contract to Hire - Gre...	0.481757
309945	Java Software Engineer @ iTech Solutions, Inc.	0.454673
305264	Sr. Java Developer @ Paladin Consulting Inc	0.406017
245753	Java Administrator @ ConsultNet	0.378968
150882	Java Consultant - Mobile Apps Development @ Co...	0.363216
146640	Jr. Java Developer @ Paladin Consulting Inc	0.323381
294489	Magento Developer (ONSITE) @ Creative Circle	0.303239

Pretty **similar** results compares to **tfidf**.

4.3 Using Spacy

Spacy uses vector embedding to compute similarity, this are the results:

JobID	title	score
250216	Microstrategy Developer @ Kavaliro	0.684521
294489	Magento Developer (ONSITE) @ Creative Circle	0.654219
257251	Front End Developer @ ConsultNet	0.63
294684	Java Developer @ Kavaliro	0.620511
257437	Drupal Developer-offsite @ Creative Circle	0.617237
316365	Jr. Ruby on Rails Developer @ ConsultNet	0.617211
257439	Drupal Developer-offsite @ Creative Circle	0.612062
257438	Drupal Developer-offsite @ Creative Circle	0.611012
257440	Drupal Developer-offsite @ Creative Circle	0.610952
302425	Software Developer @ OfficeTeam	0.608529

In this case the results are not looking so much similar, the system recommend some magento and drupal jobs (mainly for php devs).

4.4 Using KNN

The top 10 recomendation is the table below:

JobID	title	score
269922	Entry Level Java Developer / Jr. Java Develope...	0.725249
303112	Java Developer @ TransHire	0.728596
141831	Lead Java/J2EE Developer - Contract to Hire @ ...	0.810349
270171	Senior Java Developer - Contract to Hire - Gre...	0.842571
305264	Sr. Java Developer @ Paladin Consulting Inc	0.865411
309945	Java Software Engineer @ iTech Solutions, Inc.	0.903005
245753	Java Administrator @ ConsultNet	0.969298
146640	Jr. Java Developer @ Paladin Consulting Inc	0.98941
301776	Dock Worker / Part Time	1
263462	Accounting Clerk	1

You can see that, a little bit diferent from the previous recomendations in fact, the position 9 and 10 is like quite diferent(remember score close to 1 means totally diferent), so the system for this user

to decide what you're optimizing for, because in a recommender system you care about your ability to show new things that users will love.
In this post we builded several contend-based recommender systems and for **this particular case** the recomendations based on **cosine similarity** seems to show the best results.

The code can be found on this Jupyter notebook, and you can browse for more projects on my Github.

Data Science NLP Spacy Machine Learning Nltk

WRITTEN BY
Armand Olivares
Engineer, Data Science — NLP Practitioner
https://armandds.github.io/#projects

You highlighted

You highlighted

You highlighted