INetwork

Job Functions for specific Job Title

Recommendation System

1-4-2020

Mohammad Elfarash

1 Introduction

First, let's conceptualize what a job title represents. Each job has a level of responsibility (job function/s) for some department. Responsibilities include maintenance tasks such as cleaning the floor, management task such as team leadership, or strategy tasks such as deciding on budgets. Departments include growth-based such as business development and marketing, compliance-based such as finance and legal, or execution-based such as product development and operations.

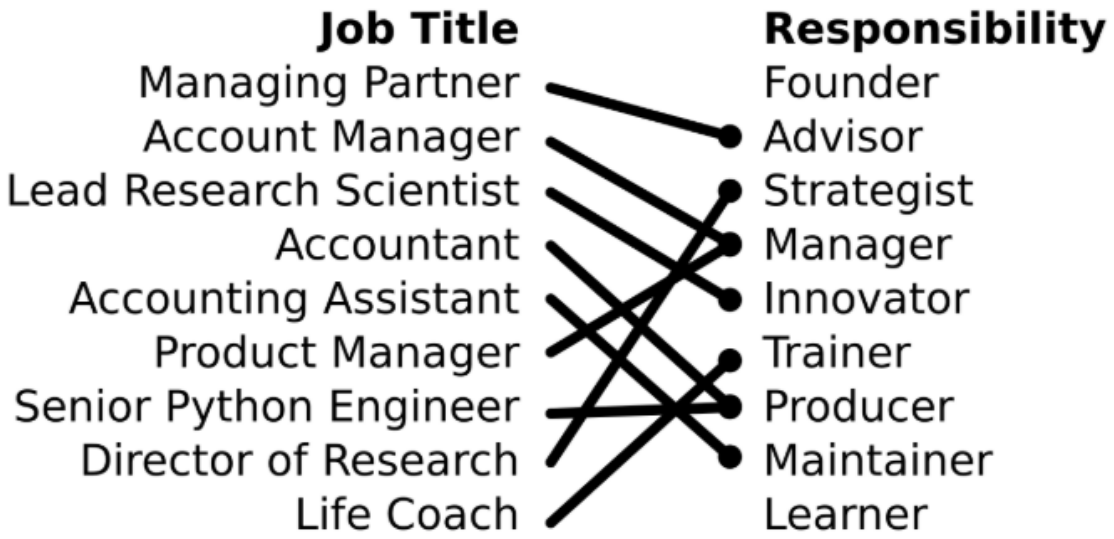

Figure 1 - Difference between job title & job functions

2 Data Preview

We are given dataset that has 3 columns (Job Title, Job Function, and Industry) the dataset contains more than 10,800 samples. As I handled it as a Machine Learning Classification problem it's a good practice to split the dataset into 70% training set and 30% test set.

3 Answering the Task Questions!

3.1 Which techniques you have used while cleaning the data if you have cleaned it?

Job Title Cleaning:

- Removing Punctuation.
- Removing Numbers.
- Removing all Non-English characters.
  - By using pandas and re dependencies
- Converting to Lower Case.
- Stop Words.
  - By using (words like a, the,...,etc) from each job title.
- Stemming
  - By using the Porter Stemmer in the NLTK package. The goal of stemming is to reduce the number of inflectional forms of words appearing in the title; it will cause words such as "developer" and "developing" into syntactically match one another by reducing them to their base word "develop". This helps decrease the size of the vocabulary space and improve the volume of the feature space in the corpus.
- Each corpus transformed into vector space model (VSM).
  - By using the tf-idf vectorizer in Python's SKlearn package to extract the features.

```
stemmer = PorterStemmer()
words = stopwords.words("english")
df['title'] = df['title'].apply(lambda x: " ".join([stemmer.stem(i) for i in re.sub("[^a-zA-Z]", " ", x).split() if i not in words]).lower())
```

Figure 2 - Removing Punk, Num, Stopwords and applying Stemming

Job Title Cleaning:

- Converting the string format to list and sorting the list.
  - By applying ast.literal_eval.
- Extracting the Classes.
  - Extracting 38 unique job function then add 38 columns (0/1) for ever class as its multi labeled classification problem.
- Removing Punctuation, Numbers, Non-English characters.
- Removing Stop Words. (if there are any)

```
df['jobFunction']= df['jobFunction'].apply(ast.literal_eval).apply(np.sort)
```

Figure 3 - Converting string into array and Sorting the array

3.2 How does your model work?

After cleaning the data and setting up all unique classes (38 classes).

	title	AnalystResearch	MediaJournalismPublishing	ProjectProgramManagement	LogisticsSupplyChain	R&DScience	AccountingFinance	WritingEditorial	OperationsManagement	IT
1348	mechan engin	0	0	0	0	0	0	0	0	
918	head of busi develop	0	1	0	0	0	0	0	1	
442	custom servic offic	0	0	0	0	0	0	0	0	
2405	senior nodej engin	0	0	0	0	0	0	0	0	
2399	senior network engin	0	0	0	0	0	0	0	0	

Figure 4 - Data Preview

The Binary relevance do its magic to divide it into 38 classifier, In other words, if there's q labels, the binary relevance method create q new data sets from the images, one for each label and train single-label classifiers on each new data set. One classifier may answer yes/no to the question "does it contain trees?", thus the "binary" in "binary relevance". This is a simple approach but does not work well when there's dependencies between the labels.

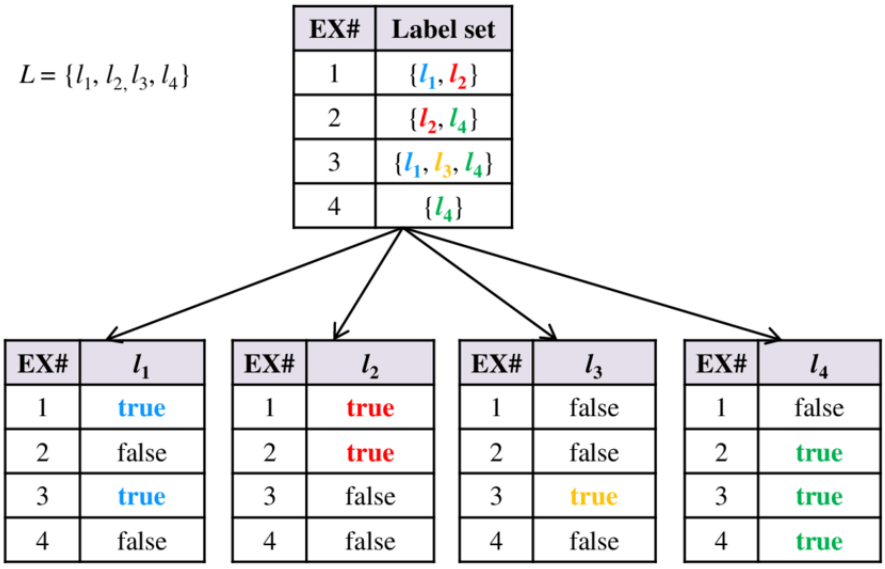

Figure 5 - Binary relevance

And for vector representation for the text, I use TF-IDF, which is an abbreviation for Term Frequency-Inverse Document Frequency and is a very common algorithm to transform text into a meaningful representation of numbers. The technique is widely used to extract features across various NLP applications. Words in the document with a high tf-idf score occur frequently in the document and provide the most information about that specific document. And pushes the output straight to the classifier (will discuss in the next question)

3.3 Why have you chosen this approach?

And after applying binary relevance and vectorization, the Classifier I use is Gaussian Naïve Bayes as it is a straightforward and powerful algorithm that gives great results when we use it for textual data analysis with more than one outcomes. To understand how Naïve Bayes work here's a simple example.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Figure 6 - Naive Bayes example

It converts the data set into a frequency table, then Create Likelihood table by finding the probabilities, then use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

3.4 How can you extend the model to have better performance?

Further improvements could be done on this problem by solving it using LSTMs in deep learning. For more speed we could use decision trees and for a reasonable trade-off between speed and accuracy we could also opt for ensemble models. Other frameworks such as MEKA can be used to deal with multi-label classification problems.

3.5 How do you evaluate your model?

Accuracy	Recall	Precision	F1score
0.657923	0.76330	0.5560797	0.6307075

3.6 What are the limitations of your methodology or where does your approach fail?

With no doubt the model can be improved as it fails with dealing with non-English job titles.

4 Testing

After setting everything we want to try out our model so if you give it a job title it starts giving you the job functions as each word in job title match to specific job functions

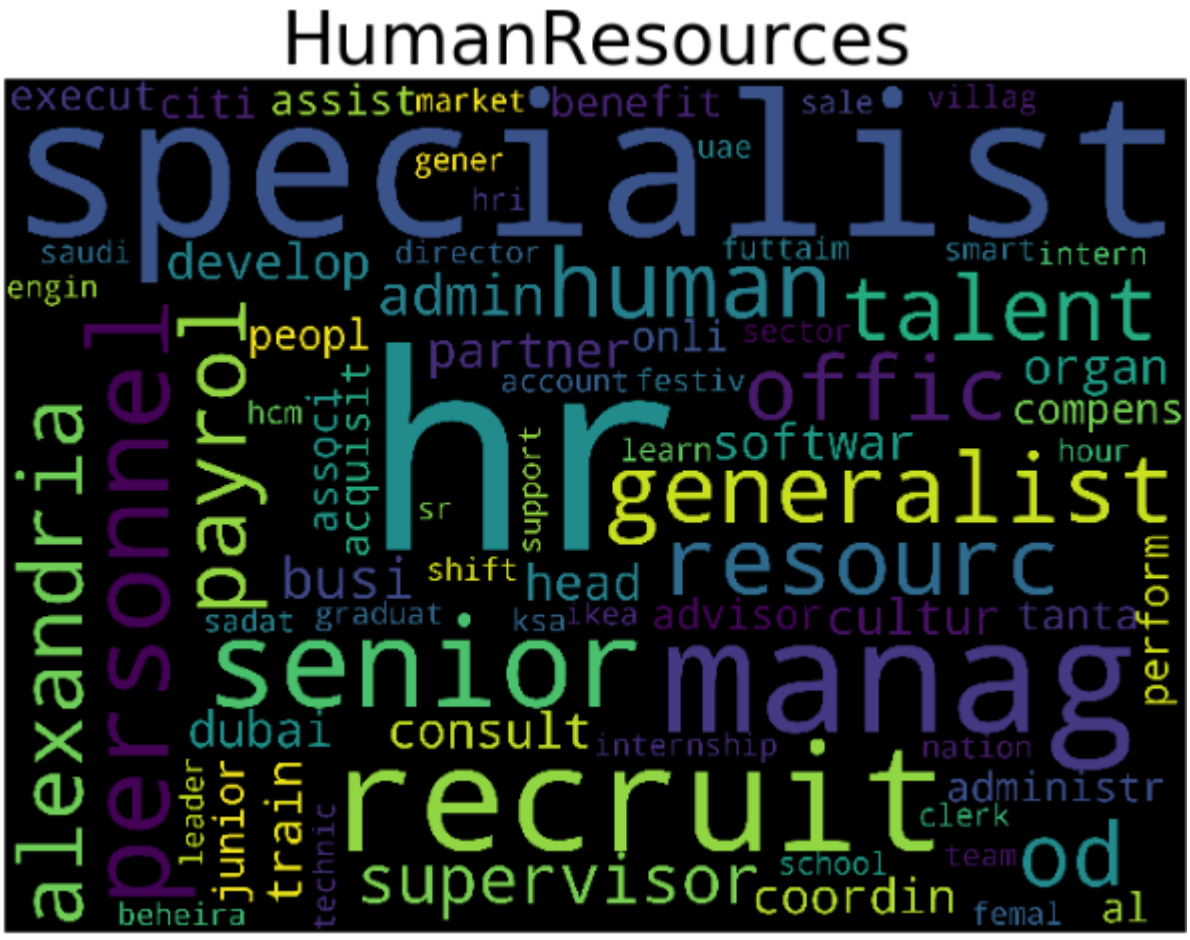

Figure 7 – WordCloud Rep These are The Most Common Words that Maps to HumanResources Class

So for example:

```
example = "hr specialist"
vec_example = vectorizer.transform([example , "-"])
```

```
predic = classifier.predict(vec_example)
output = predic.toarray(order = {2})
print(output[0])
```

```
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
```

```
predicted_jobfunctions = []
for i in range(0,len(mapper)):
  if output[0][i] == 0 :
    continue
  else :
     predicted_jobfunctions.append(mapper[i])
```

```
for i in predicted_jobfunctions:
  print(i)
```

```
HumanResources
Administration
```

Figure 8 - further example to test model

4.1 Testing on RESTful API service

By using Flask and Request libraries.

```
import flask
from flask import request


app = flask.Flask(__name__)
app.config["DEBUG"] = True

@app.route('/inetworks')
def inetworks():
    jobtitle = request.args.get('jobtitle')
    jobfunctions = model(jobtitle)
    return '<h1>the job functions is : {}</h1><h2>INetworks Task</h2>'.format(jobfunctions)
```

Figure 9 - Flask Code

On Local Host

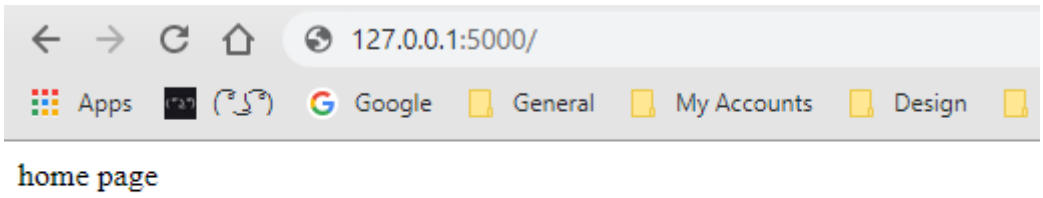

Figure 10 - Local Host Home-Page

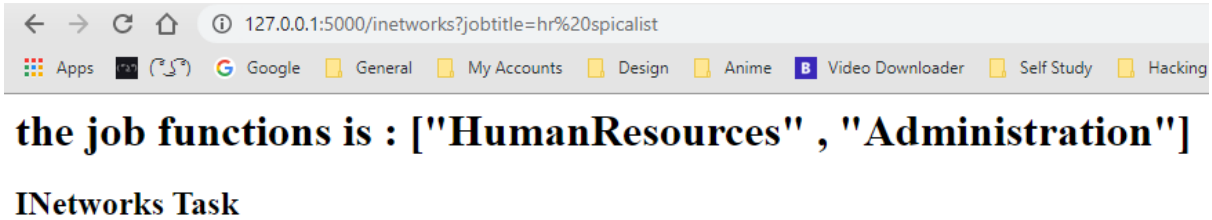

Figure 11 - Local Host INetword-Page