Job Function Recommender

By: Yomna Ahmed

Outline:

- Problem Definition
- Pre-Processing and Feature Extraction
- Classification Models
- Evaluation Metrics
- Deployment
  - As Rest API Service
  - Using Website Template
- Limitations
- Possible Extensions to the method used
- Project Folder Structure
- References

Problem Definition

Given any Job Title, Recommend suitable Job function(s) for that title.

This happened to be a **supervised classification** problem since we are provided a dataset that consists of three columns: Job Title, Job Function(s) and Job Industry. The 'title' column represents the input while the 'jobFunction' column represents the outcome.

In [54]:
```
import numpy as np
import pandas as pd
df = pd.read_csv('D:\jobs_data.csv')
df.head()
```

Out[54]:

	Unnamed: 0	title	jobFunction	industry
0	0	Full Stack PHP Developer	['Engineering - Telecom/Technology', 'IT/Softw...	['Computer Software', 'Marketing and Advertisi...
1	1	CISCO Collaboration Specialist Engineer	['Installation/Maintenance/Repair', 'IT/Softwa...	['Information Technology Services']
2	2	Senior Back End-PHP Developer	['Engineering - Telecom/Technology', 'IT/Softw...	['Computer Software', 'Computer Networking']
3	3	UX Designer	['Creative/Design/Art', 'IT/Software Developme...	['Computer Software', 'Information Technology ...
4	4	Java Technical Lead	['Engineering - Telecom/Technology', 'IT/Softw...	['Computer Software', 'Information Technology ...

'title' in each row is a single multi-word free text sentance that represents the input, This makes it a **Text Classification** problem.

'jobFunction' in each row is a list of string job functions that the title implies, This makes it a **MultiLabel Classification** problem.

'industry' in each row is a list of strings.

Pre-Processing

First, the **'jobFunction'** column needs to be cleaned and converted to numerical representation of the classes to be later fed into the classification models. There are several options for Categorical label encoding but considering that this is a MultiLabel Text Classification, OneHotEncoding proved to be the most robust and reliable in preserving the class distribution and occurence. MultiLabelBinarizer was used to achieve the desired result.

Since the Categorical Encoding process is a model fitting one, this meant that the dataset needed to be split into training and testing subsets before any further learning/pre-processing can be performed on the data (to keep the authenticity of the test data as unseen)

In [55]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
project_path = 'C:\\Users\\Dell\\PycharmProjects\\JobFunctionRecommender\\'
df = pd.read_csv(project_path + 'dataset\\jobs_data.csv')
df = df.drop(df.columns[0], axis=1)

train, test = train_test_split(df, random_state=42, test_size=0.1, shuffle=True)
print("Training set size: " + str(len(train)))
print("Testing set size: " + str(len(test)))
train.to_csv(project_path + 'dataset\jobs_data_train.csv')
test.to_csv(project_path + 'dataset\jobs_data_test.csv')
```

```
Training set size: 9783
Testing set size: 1087
```

After splitting the dataset into training set (90%) and testing set (10%), preprocessing and model fitting was applied on the training set.

First, since each row was a list of string seperated by commas and because to fit the binarizer,it needed an abstract view of all job functions, the column was stripped from all misleading symbols such as {",(,[,],)} and each job function extracted from each row and appended to another list encompassing all job functions without discrimination based on specific rows.

This list was fed to the MultiLabelBinarizer model and fitted.

In [56]:
```
from sklearn.preprocessing import MultiLabelBinarizer
functions_list = df['jobFunction']
functions_list = np.array(functions_list).reshape(functions_list.shape[0],1)
all_functions = []
for i in range(functions_list.shape[0]):
    row_functions = functions_list[i,0].translate(str.maketrans({"'":None," ":No
    for j in range(len(row_functions)):
        all_functions.append(row_functions[j])

mlb_job_functions = MultiLabelBinarizer()
mlb_job_functions.fit([all_functions])
print("Number of unique job functions is : " + str(len(mlb_job_functions.classes_
```

```
Number of unique job functions is : 38
```

In [57]:
```
print(mlb_job_functions.classes_)
```

```
['Accounting/Finance' 'Administration' 'Analyst/Research' 'Banking'
 'BusinessDevelopment' 'C-LevelExecutive/GM/Director'
 'Creative/Design/Art' 'CustomerService/Support' 'Education/Teaching'
 'Engineering-Construction/Civil/Architecture'
 'Engineering-Mechanical/Electrical' 'Engineering-Oil&Gas/Energy'
 'Engineering-Other' 'Engineering-Telecom/Technology' 'Fashion'
 'Hospitality/Hotels/FoodServices' 'HumanResources'
 'IT/SoftwareDevelopment' 'Installation/Maintenance/Repair' 'Legal'
 'Logistics/SupplyChain' 'Manufacturing/Production'
 'Marketing/PR/Advertising' 'Media/Journalism/Publishing'
 'Medical/Healthcare' 'Operations/Management' 'Pharmaceutical'
 'Project/ProgramManagement' 'Purchasing/Procurement' 'Quality'
 'R&D/Science' 'Sales/Retail' 'SportsandLeisure' 'Strategy/Consulting'
 'Tourism/Travel' 'Training/Instructor' 'Writing/Editorial' 'nan']
```

It turns out that there are 38 distinct job functions among the 9783 Job titles

After fitting the Binarizer, 38 columns for each job function class needed to be added and each row list of strings needs to be transformed into a 0/1 value for each job function class.

In [58]:
```
original_columns = df.shape[1]
resDF = df.join(pd.DataFrame(columns=mlb_job_functions.classes_).add_prefix('F_'
dummies_start_index = original_columns
dummies_end_index = resDF.shape[1]

import time
start_time = time.time()
job_functions_column_index = resDF.columns.get_loc("jobFunction")
for i in range(functions_list.shape[0]):
    row_functions = resDF.iloc[i,job_functions_column_index].translate(str.maket
    row_values = mlb_job_functions.transform([row_functions])
    resDF.iloc[i, dummies_start_index:dummies_end_index] = row_values[0]
print("------------------------------ %s minutes for job functions pre-processin
print(resDF.iloc[0,dummies_start_index:dummies_end_index])
```

```
------------------------------ 0.5291524608929952 minutes for job functions p
re-processing-----------------------
F_Accounting/Finance                            0
F_Administration                                0
F_Analyst/Research                              0
F_Banking                                       0
F_BusinessDevelopment                           0
F_C-LevelExecutive/GM/Director                  0
F_Creative/Design/Art                           0
F_CustomerService/Support                       0
F_Education/Teaching                            0
F_Engineering-Construction/Civil/Architecture   0
F_Engineering-Mechanical/Electrical             0
F_Engineering-Oil&Gas/Energy                    0
F_Engineering-Other                             0
F_Engineering-Telecom/Technology                1
F_Fashion                                       0
F_Hospitality/Hotels/FoodServices               0
F_HumanResources                                0
F_IT/SoftwareDevelopment                        1
F_Installation/Maintenance/Repair               0
F_Legal                                         0
F_Logistics/SupplyChain                         0
F_Manufacturing/Production                      0
F_Marketing/PR/Advertising                      0
F_Media/Journalism/Publishing                   0
F_Medical/Healthcare                            0
F_Operations/Management                         0
F_Pharmaceutical                                0
F_Project/ProgramManagement                     0
F_Purchasing/Procurement                        0
F_Quality                                       0
F_R&D/Science                                   0
F_Sales/Retail                                  0
F_SportsandLeisure                              0
F_Strategy/Consulting                           0
F_Tourism/Travel                                0
F_Training/Instructor                           0
F_Writing/Editorial                             0
```

```
F_nan                                                 0
Name: 0, dtype: object
```

The same preprocessing technique was used for the **'industry'** column and will not be explaind in this report for the sake of being brief in the documentation.

In [12]:
```
with open(project_path + 'saved_models\\job_functions_binarizer.pkl', 'rb') as f
    mlb_functions_list_features = pickle.load(file)

job_functions_start_index = title_feature_end_index
number_of_job_functions_features = len(mlb_functions_list_features.classes_)
job_functions_end_index = job_functions_start_index + number_of_job_functions_fe
job_functions = df.iloc[:,job_functions_start_index: job_functions_end_index]
job_functions_occurences = np.sum(job_functions)
print("\nJob Function Occurences : \n\n" + str(job_functions_occurences))
```

```

Job Function Occurences : 

F_Accounting/Finance                                 440
F_Administration                                     577
F_Analyst/Research                                   248
F_Banking                                             10
F_BusinessDevelopment                                397
F_C-LevelExecutive/GM/Director                         2
F_Creative/Design/Art                                664
F_CustomerService/Support                            784
F_Education/Teaching                                 470
F_Engineering-Construction/Civil/Architecture        273
F_Engineering-Mechanical/Electrical                  451
F_Engineering-Oil&Gas/Energy                           9
F_Engineering-Other                                  153
F_Engineering-Telecom/Technology                    3492
F_Fashion                                              3
F_Hospitality/Hotels/FoodServices                     11
F_HumanResources                                     224
F_IT/SoftwareDevelopment                            3921
F_Installation/Maintenance/Repair                    513
F_Legal                                               18
F_Logistics/SupplyChain                              158
F_Manufacturing/Production                           113
F_Marketing/PR/Advertising                          1272
F_Media/Journalism/Publishing                        871
F_Medical/Healthcare                                 200
F_Operations/Management                              278
F_Pharmaceutical                                     116
F_Project/ProgramManagement                          279
F_Purchasing/Procurement                             127
F_Quality                                            361
F_R&D/Science                                        105
F_Sales/Retail                                      1643
F_SportsandLeisure                                     6
F_Strategy/Consulting                                 14
F_Tourism/Travel                                      10
F_Training/Instructor                                118
F_Writing/Editorial                                  178
F_nan                                                105
dtype: int64
```

Job Title Feature Extraction

The next step, was applying pre-processing and feature extraction on the **'title'** column. The same techniques used for 'jobFunction' and 'industry' columns cannot be used for the 'title' column since doing that would result in almost a class for each title since each title differs in at least one way or more from the others.

The approach then was to find commonalities between the job titles without losing the variation or distinction that each title holds apart from the others. And since this is not a case where each row consists of multiple free text sentances, a common method such as TF/IDF would ignore possibly important keywords in titles based alone on its low frequency in the training set.

An approach specially designed for extracting Job Title features was used where, first, each title is stripped from any concatetaions such as tool specifics or job locations that do not affect the assigned job functions recommendation.

In [59]:
```
import re
def get_first_title(title):
    # keep "co-founder, co-ceo, etc"
    title = re.sub(r"[Cc]o[\-\ ]","", title)
    split_titles = re.split(r"\,|\-|\||\&|\:|\/|and|\(", title)
    return split_titles[0].strip()

print("Real Estates Sales Specialist - 10th of Ramadan ----->" + get_first_title
print("German Teacher - American School ----->" + get_first_title("German Teache
```

```
Real Estates Sales Specialist - 10th of Ramadan ----->Real Estates Sales Specia
list
German Teacher - American School ----->German Teacher
```

After getting the most important part of the title (the first sentance), The Features to be used in classification are extracted from this title after conversion to lower case and under the condition that the word is not an English stopword.

These features include:

- For each word in the remaining sentance, a new feature contains('word') is assigned to the row
- The last word in the title, this is significent because most titles have the most indicative word as the last. for example: 'Junior Software Engineer' has engineer as the last word which can be used as a strong indication for the job function recommendation (i.e any job that has engineer as the last word would get similar recommendations)
- The first word in the title.

In [60]:
```
import nltk
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
def get_title_features(title):
    features = {}
    title = get_first_title(title)
    word_tokens = nltk.word_tokenize(title)
    filtered_words = [w for w in word_tokens if not w in stop_words]

    for word in filtered_words:
        features['contains({})'.format(word.lower())] = True
    if len(filtered_words) > 0:
        first_key = 'first({})'.format(filtered_words[0].lower())
        last_key = 'last({})'.format(filtered_words[-1].lower())
        features[first_key] = True
        features[last_key] = True
    return features
get_title_features("Junior Software Engineer")
```

Out[60]:
```
{'contains(junior)': True,
 'contains(software)': True,
 'contains(engineer)': True,
 'first(junior)': True,
 'last(engineer)': True}
```

After applying feature extraction on the title column using these functions, they are transformed into numerical values using the same method as used previously for the 'jobFunction' and 'industry' columns.

In [62]:
```
job_titles_list = df['title']
title_features = []
for i in range(job_titles_list.shape[0]):
    row_features = get_title_features(job_titles_list[i])
    for feature in row_features:
        title_features.append(str(feature))

mlb_title_list = MultiLabelBinarizer()
mlb_title_list.fit([title_features])
print("Number of unique title features is : " + str(len(mlb_title_list.classes_)
```

```
Number of unique title features is : 1893
```

In [ ]:
```
original_columns = df.shape[1]
resDF = df.join(pd.DataFrame(columns=mlb_title_list.classes_).add_prefix('JT_'))
dummies_start_index = original_columns
dummies_end_index = resDF.shape[1]

import time
start_time = time.time()
job_titles_column_index = df.columns.get_loc("title")
for i in range(job_titles_list.shape[0]):
    title_features = []
    row_features_dict = get_title_features(resDF.iloc[i,job_titles_column_index]
    for feature in row_features_dict:
        title_features.append(str(feature))
    row_values = mlb_title_list.transform([title_features])
    resDF.iloc[i, dummies_start_index:dummies_end_index] = row_values[0]
```

After preprocessing and feature extraction are applied for all column and rows in the training set a new processed dataset is extracted which is used to experiment with classification models on.

In [63]:
```
project_path = 'C:\\Users\\Dell\\PycharmProjects\\JobFunctionRecommender\\'
df = pd.read_csv(project_path + 'dataset\\jobs_data_processed.csv')
print(df.shape)
```

```
(9783, 2002)
```

Other Pre-processing experiments that were applied:

```
- Stemming to reduce the number of features
- Allowing words in the english alphabet only to be acknowledged. <br>
```

The resulting classes of features for the titles was reduced from (1893) to (1618) but the classification accuracy did not improve that much.

In [13]:
```
#ignore industry features for now
all_training_data = df.iloc[:,0:job_functions_end_index]
print(all_training_data.shape)
```

```
(9783, 1901)
```

Classification Models

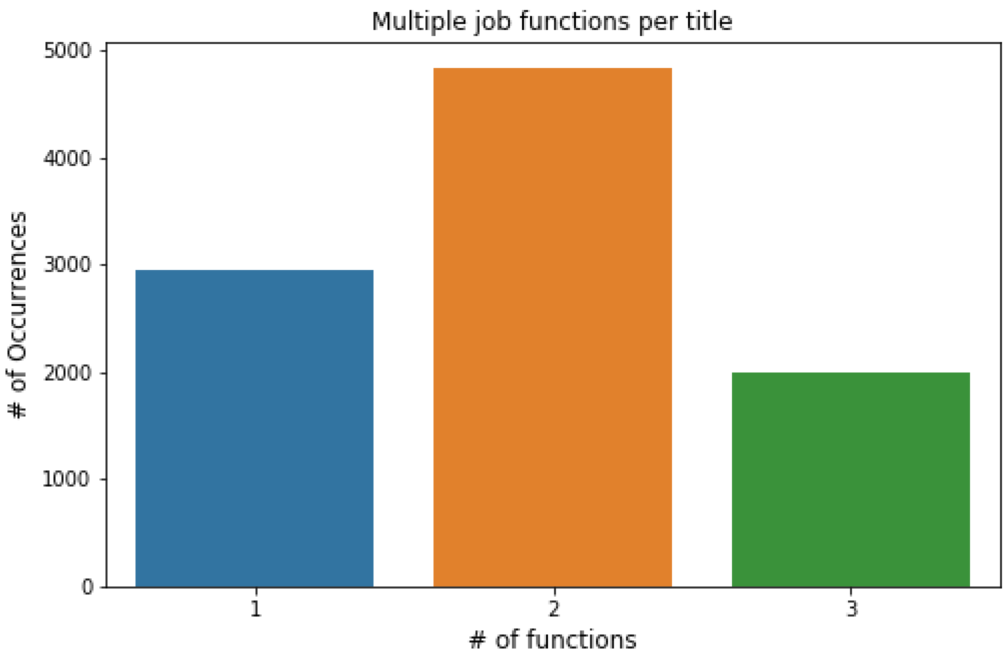
First, to attack the multilabel classification problem. OneVsRestClassifier was used where any classification algorithm could be used after casting it to a OneVsRest mode of operation.

Other methods for multilabel classification exist such as: BinaryRelevance and Classsifer chains were experimented with but produces almost the same results as using the OneVsRestClassifier technique.

> Traditional two-class and multi-class problems can both be cast into multi-label ones by restricting each instance to have only one label. On the other hand, the generality of multi-label problems inevitably makes it more difficult to learn. An intuitive approach to solving multi-label problem is to decompose it into multiple independent binary classification problems (one per category). In an “one-to-rest” strategy, one could build multiple independent classifiers and, for an unseen instance, choose the class for which the confidence is maximized.

In order to understand the multilabel classification problem further, this bar graph was used to understand the distribution of the outputs for each sample. This helped in undersstanding the importance of treating this as a multioutput problem since more than half the dataset's outputs are more than one target for the same input.

In [82]:
```
import seaborn as sns
import matplotlib.pyplot as plt
rowsums = all_training_data.iloc[:,job_functions_start_index:job_functions_end_i
x=rowsums.value_counts()
plt.figure(figsize=(8,5))
ax = sns.barplot(x.index, x.values)
plt.title("Multiple job functions per title")
plt.ylabel('# of Occurrences', fontsize=12)
plt.xlabel('# of functions', fontsize=12)
plt.show()
```



Multiple Classification algorithms were experimented on but mainly: Logistic Regression, Random Forest Classifier and K-Nearest Neighbour, The reason for choosing each one and its results are mentioned below:

- **K-Nearest Neighbour:** is what first comes to mind when building a recommendation system as it is robust in the face of an imbalanced dataset or an imbalanced problem such as Job function prediction from job title. However it faces a computational obstacle as the dataset size increases.

- **Random Forest Classifier:** Random forest is a time efficient algorithm and applies bagging so as not to fall into the trap of overfitting when using a simple decision tree. It is also adept at dealing with multilabel classification problems.
- **Logistic Regression :** Even though the Random Forest classifier generated the best training and testing results, however when hyperparameter tuning was applied on RFC to make sure that it the solution is generalized, the accuracy was substaintially reduced, so Logistic regression was experimented with to get a more generalized solution than what the Decision Tree got.

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
import time
start_time = time.time()
neigh = OneVsRestClassifier(KNeighborsClassifier(n_neighbors=3))
neigh.fit(X_train,y_train)
print("------------------------------ %s minutes for KNN fitting----------------
start_time = time.time()
KNN_Training_Score = neigh.score(X_train,y_train)
print(' KNN:', KNN_Training_Score)
print("------------------------------ %s minutes for KNN score------------------
```

For Random Forest Classifier, Hyper-parameter tuning was applied to get the best "max_depth" value, while "min_samples_leaf" was experimented with to understand how regularized the tree was.

In [16]:
```
from sklearn.multiclass import OneVsRestClassifier
import time
start_time = time.time()

X_train, y_train, y_train_original = all_training_data.iloc[:,title_feature_star

from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
clf_rfc = OneVsRestClassifier(RandomForestClassifier(max_depth=100, random_state
start_time = time.time()
clf_rfc.fit(X_train,y_train)
print("------------------------------ %s minutes for RF fitting" % str((time.tim
RFC_Training_Score = clf_rfc.score(X_train,y_train)
print('RandomForestClassifier Training accuracy :', str(RFC_Training_Score) + '\
y_test_predicted_one_hot = clf_rfc.predict(X_train)
for i in range(5):
    title_features = np.array(X_train[i,:]).reshape(1,number_of_job_title_featur
    title = mlb_title_list_features.inverse_transform(title_features)
    predicted_job_functions = np.array(y_test_predicted_one_hot[i,:]).reshape(1,
    job_functions = mlb_functions_list_features.inverse_transform(predicted_job_
    print('Title:\t{}\nTrue labels:\t{}\nPredicted labels:\t{}\n\n'.format(
        str(title),(y_train_original[i]),(str(job_functions))
    ))
```

```
------------------------------ 0.8300332347551982 minutes for RF fitting
RandomForestClassifier Training accuracy : 0.8007768578145763

Title:  [('contains(digital)', 'contains(marketing)', 'contains(specialist)',
'first(digital)', 'last(specialist)')]
True labels:    ['Media/Journalism/Publishing', 'Marketing/PR/Advertising']
Predicted labels:       [('Marketing/PR/Advertising', 'Media/Journalism/Publi
shing')]


Title:  [('contains(application)', 'contains(programmer)', 'first(applicatio
n)', 'last(programmer)')]
True labels:    ['IT/Software Development']
Predicted labels:       [('Engineering-Telecom/Technology', 'IT/SoftwareDevel
opment')]


Title:  [('contains(designer)', 'contains(graphic)', 'contains(junior)', 'fir
st(junior)', 'last(designer)')]
True labels:    ['Creative/Design/Art']
Predicted labels:       [('Creative/Design/Art',)]


Title:  [('contains(customer)', 'contains(manager)', 'contains(service)', 'fi
rst(customer)', 'last(manager)')]
True labels:    ['Customer Service/Support']
Predicted labels:       [('CustomerService/Support', 'Operations/Managemen
t')]


Title:  [('contains(architect)', 'contains(microsoft)', 'first(microsoft)',
'last(architect)')]
True labels:    ['IT/Software Development', 'Engineering - Construction/Civi
l/Architecture', 'Engineering - Telecom/Technology']
Predicted labels:       [('Engineering-Construction/Civil/Architecture', 'Eng
ineering-Telecom/Technology', 'IT/SoftwareDevelopment')]
```

For Logistic Regression,Hyper-parameter tuning was applied to get the best "C" value.

In [89]:
```
from sklearn.linear_model import LogisticRegression
clf = OneVsRestClassifier(LogisticRegression(C = 5,solver='lbfgs', max_iter=1000
clf.fit(X_train,y_train)
print("------------------------------ %s minutes for logistic regression fit----
print(' Logistic Regression Training Accuracy:', clf.score(X_train,y_train))
```

```
------------------------------ 0.6417503237724305 minutes for logistic regressi
on fit-----------------------
 Logistic Regression Training Accuracy: 0.7264879600181735
```

Evaluation Metrics

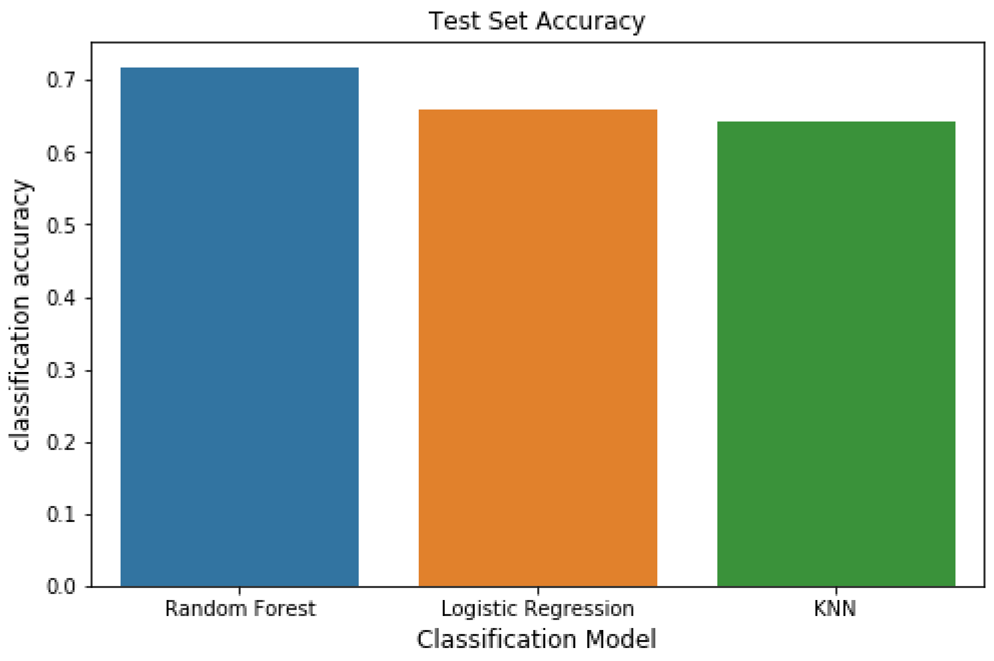
Each model was saved using pickle and tested on the testing set which was first subjected to the pre-processing process and then tested using each saved model.

The main evaluation metric used was classification accuracy:

In [3]:
```
import pandas as pd
project_path = 'C:\\Users\\Dell\\PycharmProjects\\JobFunctionRecommender\\'
transformedTest = pd.read_csv(project_path + 'dataset\\jobs_data_test_processed.
X_test = transformedTest.iloc[:,5:1864]
y_test = transformedTest.iloc[:,1864:1902]
import time
start_time = time.time()
import pickle
with open(project_path + 'saved_models\\TrainedLogisticClassifier.pkl', 'rb') as
    clf_log = pickle.load(file)
Log_score = clf_log.score(X_test,y_test)
print(' Logistic Regression Test Score:', Log_score)
print("------------------------------ %s minutes for test set recommendation----

start_time = time.time()
with open(project_path + 'saved_models\\TrainedRandomForestClassifier.pkl', 'rb'
    clf_rfc = pickle.load(file)
RFC_score = clf_rfc.score(X_test,y_test)
print(' Random Forest Test Score:', RFC_score)
print("------------------------------ %s minutes for test set recommendation----

start_time = time.time()
with open(project_path + 'saved_models\\TrainedKNNClassifier.pkl', 'rb') as file
    clf_rfc = pickle.load(file)
KNN_Score = clf_rfc.score(X_test,y_test)
print(' KNN Test Score:', KNN_Score)
print("------------------------------ %s minutes for test set recommendation----
```

```
 Logistic Regression Test Score: 0.6605335786568537
------------------------------ 0.02883416016896566 minutes for test set recomme
ndation-----------------------
 Random Forest Test Score: 0.7175712971481141
------------------------------ 0.01745418707529704 minutes for test set recomme
ndation-----------------------
 KNN Test Score: 0.6430542778288868
------------------------------ 27.807108020782472 minutes for test set recommen
dation-----------------------
```

In [95]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
x=["Random Forest", "Logistic Regression", "KNN"]
y = [RFC_score,Log_score,KNN_Score]
plt.figure(figsize=(8,5))
ax = sns.barplot(x, y)
plt.title("Test Set Accuracy")
plt.ylabel('classification accuracy', fontsize=12)
plt.xlabel('Classification Model', fontsize=12)
plt.show()
```

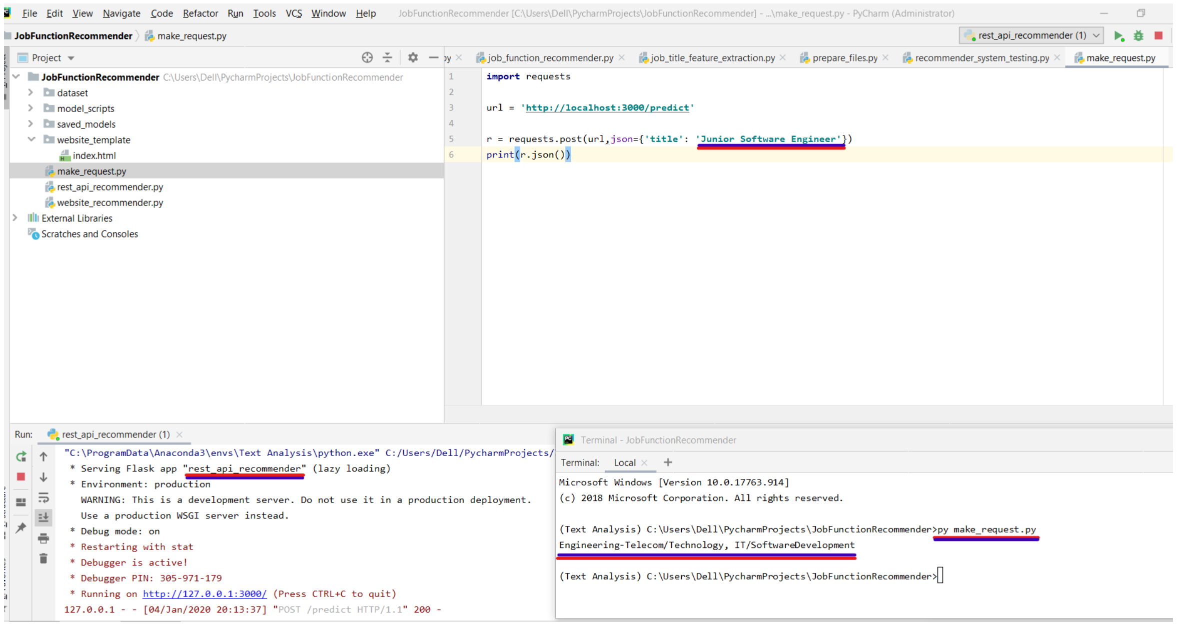


Deployment As Rest API Service

- Flask was used to setup the service

- A script called 'make_request' sends a string request to the URL where the service is setup, The string request contains the job title.
- Another script sets up the service at a URL '/predict' and when a request is sent, it calls the job_function_recommender which takes a title and returns a list of string of the recommended job functions.

In Action Screenshots:


File Edit View Navigate Code Refactor Run Tools VCS Window Help
JobFunctionRecommender [C:\Users\Dell\PycharmProjects\JobFunctionRecommender] - ...\make_request.py - PyCharm (Administrator)
JobFunctionRecommender 〉 make_request.py
rest_api_recommender (1)
Project
JobFunctionRecommender C:\Users\Dell\PycharmProjects\JobFunctionRecommender
dataset
model_scripts
saved_models
website_template
index.html
make_request.py
rest_api_recommender.py
website_recommender.py
External Libraries
Scratches and Consoles
job_function_recommender.py
job_title_feature_extraction.py
prepare_files.py
recommender_system_testing.py
make_request.py
Run: rest_api_recommender (1)
Terminal - JobFunctionRecommender
Terminal: Local

```
import requests

url = 'http://localhost:3000/predict'

r = requests.post(url,json={'title': 'Junior Software Engineer'})
print(r.json())
```

```
"C:\ProgramData\Anaconda3\envs\Text Analysis\python.exe" C:/Users/Dell/PycharmProjects/
 * Serving Flask app "rest_api_recommender" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 305-971-179
 * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
127.0.0.1 - - [04/Jan/2020 20:13:37] "POST /predict HTTP/1.1" 200 -
```

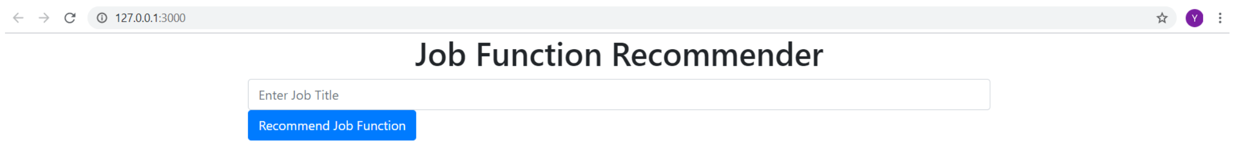
```
Microsoft Windows [Version 10.0.17763.914]
(c) 2018 Microsoft Corporation. All rights reserved.

(Text Analysis) C:\Users\Dell\PycharmProjects\JobFunctionRecommender>py make_request.py
Engineering-Telecom/Technology, IT/SoftwareDevelopment

(Text Analysis) C:\Users\Dell\PycharmProjects\JobFunctionRecommender>
```

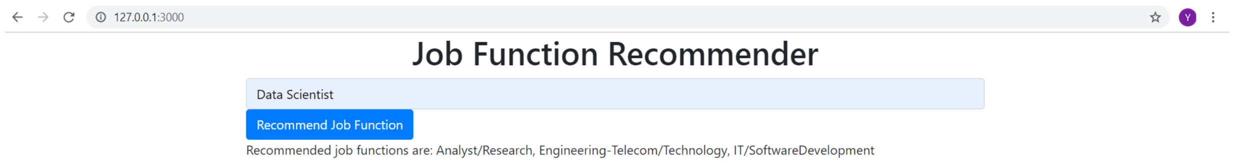
In [1]:
```python
from flask import Flask, request, jsonify # Loading in Flask
app = Flask(__name__)

from model_scripts.job_function_recommender import job_function_recommender

@app.route('/predict', methods=['POST'])
def predict():
    # Get data from Post request
    data = request.get_json()
    title = str(data['job title'])
    # making predictions
    pred = job_function_recommender(title)
    pred_str = ''
    for i in pred:
        if i > 0:
            pred_str = pred_str +', '
        pred_str = pred_str + i
    # returning the predictions as json
    return jsonify(pred_str)

if __name__ == '__main__':
    app.run(port=3000, debug=True)
```

...

Deployment using website template

In Action Screenshots:





Recommended job functions are: Analyst/Research, Engineering-Telecom/Technology, IT/SoftwareDevelopment

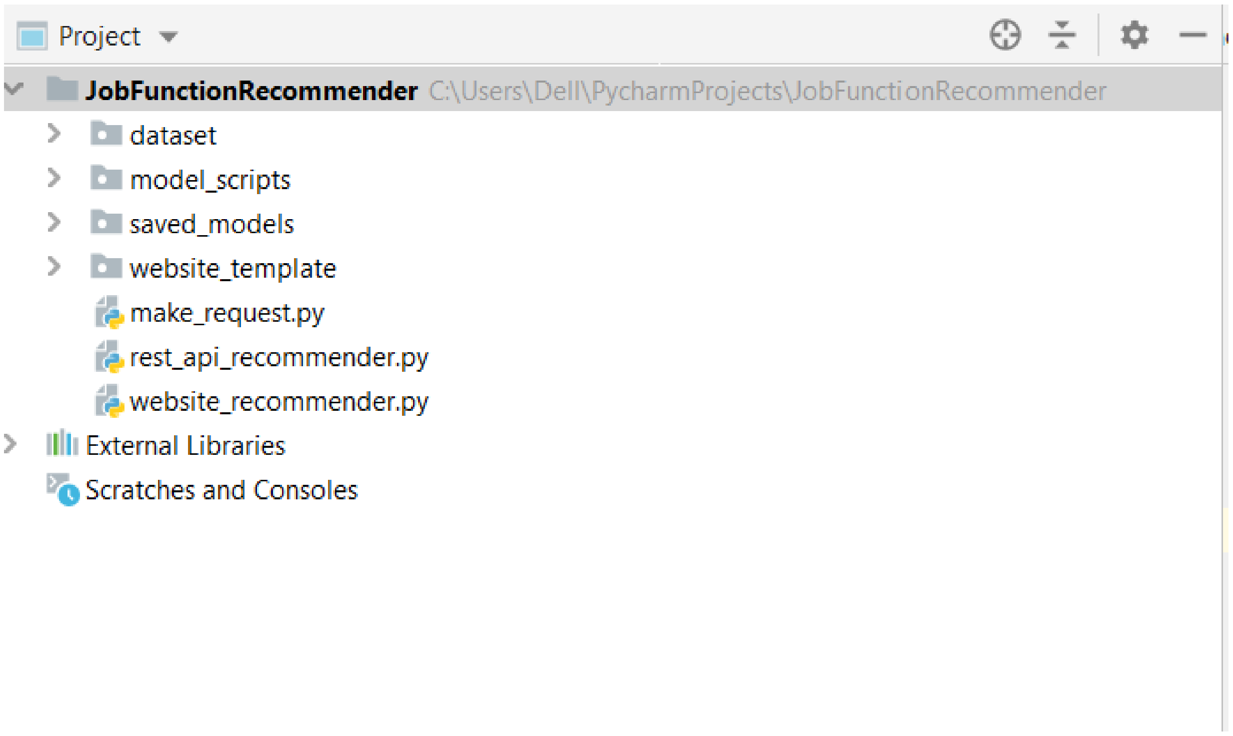
Limitations:

- The number of features extracted from the job titles is still too many, noisy features still exist, such as some arabic words and abreviations.
- The model does not always return a recommendation even when lexically an input is similar to some of the samples in the dataset.

Possible extensions to the method used:

- Use the 'industry' column to improve recommendation. (for example: predict industry from job title and then using both title and industry, predict job functions)
- Experiment with other classification approaches such as CNN.
- Further study the feature extraction process for this dataset to drop noisy features.

Project Folder Structure



JobFunctionRecommender C:\Users\Dell\PycharmProjects\JobFunctionRecommender
- dataset
- model_scripts
- saved_models
- website_template
- make_request.py
- rest_api_recommender.py
- website_recommender.py

External Libraries
Scratches and Consoles

JobFunctionRecommender : (folder)

> dataset : (folder) for saving the preprocessed .csv files to accelerate the classification experiments (avoid waiting for preprocessing each time a model needs to be fitted).
> make_request : (script).
> rest_api_recommender : (script)
> website_recommender : (script)
> website template : (folder)
>
> > index.html

> > saved_models : (folder) pickle saved encoding and classification models
> > model_scripts : (folder)
> >
> > > prepare_files : (script) split dataset into two files one for training and one for testing
> > > job_title_feature_extraction (script)
> > > preprocessing : (script) apply pre-processing and feature extraction on the training set and save the processed features to csv
> > > recommender_system_training : (script) read the processed features from training file and apply different classification algorithms
> > > recommender_system_testing : (script) apply pre-processing and prediction on the testing set
> > > job_function_recommender : (script) contains function that given a single input recommends a list of corresponding job functions

References:

Deploying your ML model - Building a simple website and rest-api using Flask [https://www.youtube.com/watch?v=tjSV6pzJsGg] (https://www.youtube.com/watch?v=tjSV6pzJsGg%5D)
Deep Dive into MultiLabel Classification [https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff] (https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff%5D)
Job title analysis in Python [https://towardsdatascience.com/job-title-analysis-in-python-and-nltk-8c7ba4fe4ec6] (https://towardsdatascience.com/job-title-analysis-in-python-and-nltk-8c7ba4fe4ec6%5D)