Import config

Import glob

Import numpy as np

Import matplotlib.pyplot as plt

From FunctionsForJobRecommendation import FunctionsForJobRecommendation

Import os

Import json

Def main():

# The data scraped from web is obtained from reference dataset which is stored in JSON file

Exists = os.path.isfile(config.JOBS\_INFO\_JSON\_FILE)

If exists:

With open(config.JOBS\_INFO\_JSON\_FILE, ‘r’) as fp:

JobsInfo = json.load(fp)

# Initialize skill\_keyword\_match with JobsInfo

Skill\_match = FunctionsForJobRecommendation(JobsInfo)

# Extract skill keywords from job descriptions

Skill\_match.ExtractJobDescKeywords()

# Extract resume skills from given resume and store them in a list

For resumePDF in glob.glob(config.SAMPLE\_RESUME\_PDF\_DIR+”SampleResume\*.pdf”):

Print(“=======================================================”)

Print(“Processing the resume : “,resumePDF)

Print(“=======================================================”)

ResumeSkills = skill\_match.ExtractResumeKeywords(resumePDF)

ResumeSkills.reset\_index(inplace=True)

ResumeSkills.rename(columns={‘index’: ‘skillsinresume’}, inplace=True)

ResumeSkillList = ResumeSkills[‘skillsinresume’].tolist()

Resume\_skill\_list\_dummy = [‘azure’,’sql’,’mysql’,’c++’,’excel’,’power’,’keras’,’agile’,’r’,’tableau’,’google’]

Print(“Skills extracted from resume are : \n”,ResumeSkillList)

# Calculate similarity of skills from a resume and job post and get top10 job descriptions

MainTop10JDs = skill\_match.CalculateSimilarity(ResumeSkillList)

# copy of the dataframe as “MainTop10JDs2” to keep them different for static and dynamic approach

MainTop10JDs2 = MainTop10JDs.copy()

# Extract 20 similar Job description for each of the top10 job descriptions

# Explicit and Implicit skills extracted for static weight approach

ImplicitStatic,finalSkillWeightList = skill\_match.Extract20SimilarJDs(0,MainTop10JDs, ResumeSkillList)

# Calculating Final cosine score based on term frequency and weighted cosine similarity

FinalJDPrev = skill\_match.WeightedCosineSimilarity(ResumeSkillList, ImplicitStatic)

Print(“Below is the reference approach job listing ranking\n”,FinalJDPrev[[‘Jobid’,’final\_cosine’]])

# Extract 20 similar Job description for each of the top10 job descriptions

# Explicit and Implicit skills extracted for dynamic weight approach

ImplicitDynamic,finalSkillWeightList = skill\_match.Extract20SimilarJDs(1,MainTop10JDs2, ResumeSkillList)

# Calculating Final cosine score based on term frequency and weighted cosine similarity

FinalJD = skill\_match.WeightedCosineSimilarity(ResumeSkillList, ImplicitDynamic)

Print(“Below is the proposed approach job listing ranking\n”,FinalJD[[‘Jobid’,’final\_cosine’]])

topIndex = FinalJD[‘Jobid’][0]

allTopSkills = ImplicitDynamic.loc[topIndex][‘keywords’]

topExSkills = []

topImpSkills = []

for skill, weight in allTopSkills:

if weight ==1:

topExSkills.append(skill)

else:

topImpSkills.append(skill)

print(“Explicit skills to upskill : “,np.setdiff1d(topExSkills,ResumeSkillList))

diffImpSkills = np.setdiff1d(topImpSkills,ResumeSkillList)

if len(diffImpSkills)>5:

print(“Implicit skills to upskill : “,np.setdiff1d(topImpSkills,ResumeSkillList)[0:5])

else:

print(“Implicit skills to upskill : “,np.setdiff1d(topImpSkills,ResumeSkillList))

# Graph plot with explicit and implicit skills that match the resume for static approach

ImplicitStaticGraph = FinalJDPrev[[“Jobid”,”final\_cosine”,”exSkillCountResumeMatch”,”impSkillCountResumeMatch”]]

# skill\_match.GraphPlotsForEvaluation(ImplicitStaticGraph,finalSkillWeightList,0)

# Graph plot with explicit and implicit skills that match the resume for dynamic approach

# Graph plot to show how the ranking of the top10 job postings differ due to the Implicit weightage of skills

ImplicitDynamicGraph = FinalJD[[“Jobid”,”final\_cosine”,”exSkillCountResumeMatch”,”impSkillCountResumeMatch”]]

# skill\_match.GraphPlotsForEvaluation(ImplicitDynamicGraph,finalSkillWeightList,1)

If(resumePDF.count(r’SampleResume1’) == 1):

Plt.figure()

Skill\_match.AllGraphPlotsForEvaluation(ImplicitStaticGraph,ImplicitDynamicGraph,finalSkillWeightList,1)

If \_\_name\_\_ == “\_\_main\_\_”:

Main()