Resume Scorer (Information Retrieval Assignment)

Name - Anshuman Srivastava

Roll - BTECH/10408/18

Problem Statement and Dataset
A Hackerearth competition named "A Perfect Fit - ML Challenge" was held in August 2021. Broadly the problem solution was to retrieve the valuable information from a resume and score it according to a given Job Describtion. I have used the a set of 150 resumes to train the model which was available in a competition

Approach

- **Step 1 - Convert to Txt - Since the resume are in pdf formats first we need to convert them into readable formats (txt here).**
  For this purpose I have used "pdf2miner" python library and stored all resumes in txt format.

- **Step 2 - Cleaning & WordBreak - Now we have resume text and before moving further we must preprocess it removing all kinds of stopwords, URLs, whitespaces etc.. using "NLTK" library.After cleaning is done the whole text is split into words**

- **Step 3 - Splitting Into Sections - This is the main part where we divide words in 5 categories (["personal","projects","experience","skills","education"]) . We do a linear scan over all words and take find these 5 keywords in the resume and take correspoding lines after them.**

- **Step 4 - Vectorization and Calculating Cosine - For each section we transform each word to its "Glove vector of 100 dimenstions" and then accumulate their average over each dimension. Above 4 steps will be applied to JD also and then we will compare the cosing similarity of two vectors for each section**

- **Step 5 - Score from Cosine Similarities - Now we will have 5 cosine similaries according corresponding to a Job Description . Next step we will make a small Neural Network with 5 input layers 2 hidden layers and 1 output layers. We will train the network through the train data given. Hence obtaining a score of how much the resume is perfect fit for the corresponding Job Description**

CODE

Since Step 1 was very small I have started with Step2 here

In [1]:
```
'''
Importing Libraries
'''
import re
import os
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from numpy import dot
from numpy.linalg import norm
```

In [2]:
```
'''
Declaring Constants
'''
keywords = ["personal","projects","experience","skills","education"]
stop_words = stopwords.words('english')
MAX_SEQUENCE_LENGTH = 100
MAX_WORDS = None
```

In [3]:
```
'''
Cleaning Resume Text obtained after converting PDF to txt
'''
def cleanResume(resumeText):
    resumeText = re.sub('http\S+\s*', ' ', resumeText)  # remove URLs
    resumeText = re.sub('RT|cc', ' ', resumeText)  # remove RT and cc
    resumeText = re.sub('@\S+', '  ', resumeText)  # remove mentions
    resumeText = re.sub('[%s]' % re.escape("""!"#$%&'()*,-./:;<=>?@[\]^_`{|}~"""), ' ', resumeText)  # remove punctuations
    resumeText = re.sub(r'[^\x00-\x7f]',r' ', resumeText)
    resumeText = re.sub('\s+', ' ', resumeText)  # remove extra whitespace
    resumeText = [x for x in resumeText.split() if x not in stop_words]     #Removing stopwords
    return resumeText
```

In [5]:
```
'''
```

```
Function to divide the text into 5 sections as discussed
'''
def read_txt (path_to_folder, dir):
    indexes = []
    categories = {}
    for w in keywords:
        categories[w] = []
    for filename in dir:
        file = open(path_to_folder + filename,'r')
        read = file.read()
        read = read.lower()
        file.close()

        read = cleanResume(read)

        hash = {}
        hash["personal"] =0
        for word in keywords:
            if word in read:
                hash[word] = read.index(word)


        items = sorted(hash.items(), key = lambda x: x[1])
        for i in range(len(items)):
            start = items[i][1]
            end = None
            if (i+1)==len(items):
                end = len(read)
            else:
                end = items[i+1][1]

            categories[items[i][0]].append(read[start:end])

        for w in keywords:
            if w not in hash.keys():
                categories[w].append('None')

        indexes.append(filename)
    return  categories, indexes
```

In [6]:
```
dir = os.listdir('./train_txt/')
categories, indexes = read_txt('./train_txt/',dir)
```

In [7]:
```
data = []
words_dict = ['None']
for i in range(len(indexes)):
    row = []
    for w in keywords:
        row.append(categories[w][i])
        if categories[w][i] is not  None:
            words_dict+=categories[w][i]

    data.append(row)
```

In [17]:
```
'''
This is how our data looks after Step 2
'''
df = pd.DataFrame(data, index=indexes, columns=keywords)
df
```

Out[17]:

	personal	projects	experience	skills	education
candidate_000.txt	[personal, profile, actively, seeking, opportu...	[projects, music, genre, classification, face,...	None	[skills, python, sql, mysql, tableau, power, b...	[education, b, tech, ece, vit, ap, university,...
candidate_001.txt	[brianna, williams, junior, developer, executi...	[projects, also, contribute, knowledge, logica...	[experience, curiosity, driven, data, scientis...	[skills, towards, consistent, growth, developm...	[education, teamwork, bsc, ca, mamco, universi...
candidate_002.txt	[mason, quadrado, associate, analyst, certifie...	None	[experience, analyzing, interpreting, data, go...	[skills, python, machine, learning, mysql, dat...	[education, b, tech, b, e, electronics, teleco...
candidate_003.txt	[associate, software, engineer]	[projects, koy, ok, 1e, im, ge, tena, wal, tur...	[experience, software, engineer, machine, lear...	[skills, ava, alo, avin, zt, od, al, ms, 1, da...	[education, b, tech, v, v, 2018, activities, a...
candidate_006.txt	[jennifer, armstrong, computer, vision, enthus...	[projects, understanding, images, gan, based, ...	[experience, currently, professional, experience]	[skills, machine, learning, deep, learning, co...	[education, b, tech, computer, science, iit, g...
...	...	...	...	...	...
candidate_144.txt	[benjamin, osta, fresher, developer, professio...	[projects, proficient]	None	[skills, software, engineer, software, develop...	None
candidate_145.txt	[jerome, pelinsky, big, data, analyst, big, da...	None	[experience, handling, kinds, data, also, used...	[skills, big, data, hadoop, hive, python, mapr...	[education, b, tech, electronics, amity, schoo...
candidate_147.txt	[jaroslav, chechnik, executive, profile, work]	[projects, music, genre, classification, face,...	[experience, looking, job, opportunity, expert...	[skills, b, tech, ece, vit, ap, university, 20...	[education]
candidate_148.txt	[data, scientist]	[projects, acvaline, daal, lan, el, kx, 1e, mm...	[experience, building, deploying, end, end, an...	[skills, dy, esxoll, alot, dy, ha, wy, val, hv...	[education, b, tech, b, e, computers, rajiv, g...
candidate_149.txt	[personal, profile, machine, learning, enginee...	[projects, wesbite, using, react, made, fully,...	None	[skills, artificial, intelligence, deep, learn...	[education, b, tech, hit, kancheepuram, chenna...

90 rows × 5 columns

In [18]:
```
'''
Lets do the same for our target resume/JD which is assumed to have a score of 1.
'''
```

```
dir = ['Job description.txt']
cat, _ = read_txt('./', dir)
```

In [19]:
```
'''
Next step is to transform the words to their glove vector
'''
embeddings_dict = {}
embed_keys = []
with open("glove.6B.100d.txt", 'r', encoding="utf-8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        embed_keys.append(word)
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
```

In [20]:
```
'''
Handling unknown tokens and taking the average over each dimension
'''
jd_vec = {}
unks = 0
for w in keywords:
    vecs = []
    for word in cat[w][0]:
        if word in embed_keys:
            vecs.append(embeddings_dict[word])
        else:
            unks+=1
            vecs.append(embeddings_dict['<unk>'])
    vecs = np.array(vecs)
    avg = np.average(vecs, axis = 0)
    jd_vec[w] = avg

print("Total unk tokens" , unks)
```

```
Total unk tokens 10
```

In [21]:
```
'''
Calculating cosing smilarity for each section of each candidate using JD
'''
sim_df = []
unks=0
all_words = []
for i in range(len(indexes)):
    row = []
    for w in keywords:
        vecs = []
        for word in categories[w][i]:
            all_words.append(word)
            if word in embed_keys:
                vecs.append(embeddings_dict[word])
            else:
                unks+=1
                vecs.append(embeddings_dict['<unk>'])
        vecs = np.array(vecs)
        va = np.average(vecs, axis = 0)
        cosine = None
        if np.isnan(va).any():
            cosine = 0
        else:
            vb = jd_vec[w]
            cosine = dot(va,vb)/(norm(va)*norm(vb))

        row.append(cosine)

    sim_df.append(row)

all_words = set(all_words)
print("Total words extracted are ", len(all_words))
print("Total unk tokens" , unks)
```

```
C:\Users\sriva\AppData\Roaming\Python\Python39\site-packages\numpy\lib\function_base.py:380:
RuntimeWarning: Mean of empty slice.
  avg = a.mean(axis)
C:\Users\sriva\AppData\Roaming\Python\Python39\site-packages\numpy\core\_methods.py:170: Run
timeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

```
Total words extracted are  2319
Total unk tokens 450
```

In [22]:
```
'''
This is how our data looks after calculating 5 section similarity with the JD
'''
# Similartiy Table
similariy = pd.DataFrame(sim_df, index=indexes, columns=keywords)
similariy
```

Out[22]:

	personal	projects	experience	skills	education
candidate_000.txt	0.888378	0.295928	0.415252	0.636419	0.856955
candidate_001.txt	0.610979	0.234160	0.776972	0.762761	0.882273
candidate_002.txt	0.668103	1.000000	0.620666	0.924627	0.754305
candidate_003.txt	0.694706	0.378404	0.694232	0.825477	0.770898
candidate_006.txt	0.906714	0.300554	0.769333	0.881404	0.773082
...	...	...	...	...	...
candidate_144.txt	0.900714	0.164987	0.415252	0.853482	0.393262
candidate_145.txt	0.738020	1.000000	0.758193	0.920389	0.725882
candidate_147.txt	0.576090	0.273347	0.758924	0.730275	0.708900
candidate_148.txt	0.698836	0.395247	0.733054	0.694591	0.529111

candidate_149.txt	0.872008	0.358750	0.415252	0.663939	0.740257

90 rows × 5 columns

In [14]:
```
similariy.fillna(0, inplace=True)
```

In [23]:
```
xs = np.array(similariy)
```

Test Data

In [26]:
```
target = pd.read_csv('./dataset/train.csv', index_col='CandidateID')
target.head()
```

Out[26]:

CandidateID	Match Percentage
candidate_011	13.60
candidate_113	36.63
candidate_123	54.93
candidate_012	41.46
candidate_002	48.91

In [27]:
```
ys = []
for i in indexes:
    ys.append(target.loc[i[:-4]]['Match Percentage']/100)
ys = np.array(ys, dtype = np.float32)
```

In [28]:
```
xs = xs.astype(np.float32)
ys = ys.astype(np.float32)
print(xs.shape)
print(ys.shape)
```

```
(90, 5)
(90,)
```

Now we have caluclated cosines and we will feed it to our neural network

In [20]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(xs, ys, test_size = 0.1, random_state = 42)
```

In [21]:
```
from keras.models import Sequential, Model
from keras.layers import Dense , Dropout, Activation
```

In [22]:
```
'''
Building and Training Model
'''
def build_model():
    model = Sequential()
    model.add(Dense(8, input_shape =[5,]))
    model.add(Dense(8))
    model.add(Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics =['mse'])
    return model
```

In [23]:
```
model = build_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 48
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 129
Trainable params: 129
Non-trainable params: 0
_________________________________________________________________
```

In [24]:
```
history = model.fit(X_train,y_train, epochs=100,validation_data=(X_test,y_test))
```

```
Epoch 1/100
3/3 [==============================] - 3s 146ms/step - loss: 0.8432 - mse: 0.8432 - val_los
s: 0.7007 - val_mse: 0.7007
Epoch 2/100
3/3 [==============================] - 0s 37ms/step - loss: 0.5713 - mse: 0.5713 - val_loss:
0.5889 - val_mse: 0.5889
Epoch 3/100
3/3 [==============================] - 0s 41ms/step - loss: 0.4523 - mse: 0.4523 - val_loss:
0.5309 - val_mse: 0.5309
Epoch 4/100
3/3 [==============================] - 0s 30ms/step - loss: 0.3858 - mse: 0.3858 - val_loss:
0.4918 - val_mse: 0.4918
Epoch 5/100
3/3 [==============================] - 0s 35ms/step - loss: 0.3401 - mse: 0.3401 - val_loss:
0.4664 - val_mse: 0.4664
Epoch 6/100
3/3 [==============================] - 0s 37ms/step - loss: 0.3108 - mse: 0.3108 - val_loss:
0.4479 - val_mse: 0.4479
Epoch 7/100
3/3 [==============================] - 0s 30ms/step - loss: 0.2898 - mse: 0.2898 - val_loss:
0.4330 - val_mse: 0.4330
Epoch 8/100
```

```
3/3 [==============================] - 0s 40ms/step - loss: 0.2775 - mse: 0.2775 - val_loss:
0.4166 - val_mse: 0.4166
Epoch 9/100
3/3 [==============================] - 0s 34ms/step - loss: 0.2616 - mse: 0.2616 - val_loss:
0.4046 - val_mse: 0.4046
Epoch 10/100
3/3 [==============================] - 0s 34ms/step - loss: 0.2484 - mse: 0.2484 - val_loss:
0.3900 - val_mse: 0.3900
Epoch 11/100
3/3 [==============================] - 0s 32ms/step - loss: 0.2377 - mse: 0.2377 - val_loss:
0.3813 - val_mse: 0.3813
Epoch 12/100
3/3 [==============================] - 0s 34ms/step - loss: 0.2269 - mse: 0.2269 - val_loss:
0.3705 - val_mse: 0.3705
Epoch 13/100
3/3 [==============================] - 0s 36ms/step - loss: 0.2190 - mse: 0.2190 - val_loss:
0.3512 - val_mse: 0.3512
Epoch 14/100
3/3 [==============================] - 0s 65ms/step - loss: 0.2079 - mse: 0.2079 - val_loss:
0.3393 - val_mse: 0.3393
Epoch 15/100
3/3 [==============================] - 0s 36ms/step - loss: 0.1986 - mse: 0.1986 - val_loss:
0.3259 - val_mse: 0.3259
Epoch 16/100
3/3 [==============================] - 0s 34ms/step - loss: 0.1910 - mse: 0.1910 - val_loss:
0.3135 - val_mse: 0.3135
Epoch 17/100
3/3 [==============================] - 0s 36ms/step - loss: 0.1802 - mse: 0.1802 - val_loss:
0.3005 - val_mse: 0.3005
Epoch 18/100
3/3 [==============================] - 0s 42ms/step - loss: 0.1718 - mse: 0.1718 - val_loss:
0.2883 - val_mse: 0.2883
Epoch 19/100
3/3 [==============================] - 0s 36ms/step - loss: 0.1633 - mse: 0.1633 - val_loss:
0.2768 - val_mse: 0.2768
Epoch 20/100
3/3 [==============================] - 0s 38ms/step - loss: 0.1559 - mse: 0.1559 - val_loss:
0.2642 - val_mse: 0.2642
Epoch 21/100
3/3 [==============================] - 0s 39ms/step - loss: 0.1460 - mse: 0.1460 - val_loss:
0.2537 - val_mse: 0.2537
Epoch 22/100
3/3 [==============================] - 0s 33ms/step - loss: 0.1380 - mse: 0.1380 - val_loss:
0.2459 - val_mse: 0.2459
Epoch 23/100
3/3 [==============================] - 0s 38ms/step - loss: 0.1325 - mse: 0.1325 - val_loss:
0.2336 - val_mse: 0.2336
Epoch 24/100
3/3 [==============================] - 0s 36ms/step - loss: 0.1236 - mse: 0.1236 - val_loss:
0.2270 - val_mse: 0.2270
Epoch 25/100
3/3 [==============================] - 0s 37ms/step - loss: 0.1175 - mse: 0.1175 - val_loss:
0.2177 - val_mse: 0.2177
Epoch 26/100
3/3 [==============================] - 0s 38ms/step - loss: 0.1100 - mse: 0.1100 - val_loss:
0.2059 - val_mse: 0.2059
Epoch 27/100
3/3 [==============================] - 0s 36ms/step - loss: 0.1050 - mse: 0.1050 - val_loss:
0.2057 - val_mse: 0.2057
Epoch 28/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0989 - mse: 0.0989 - val_loss:
0.1830 - val_mse: 0.1830
Epoch 29/100
3/3 [==============================] - 0s 42ms/step - loss: 0.0929 - mse: 0.0929 - val_loss:
0.1747 - val_mse: 0.1747
Epoch 30/100
3/3 [==============================] - 0s 41ms/step - loss: 0.0869 - mse: 0.0869 - val_loss:
0.1743 - val_mse: 0.1743
Epoch 31/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0822 - mse: 0.0822 - val_loss:
0.1567 - val_mse: 0.1567
Epoch 32/100
3/3 [==============================] - 0s 42ms/step - loss: 0.0770 - mse: 0.0770 - val_loss:
0.1488 - val_mse: 0.1488
Epoch 33/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0733 - mse: 0.0733 - val_loss:
0.1454 - val_mse: 0.1454
Epoch 34/100
3/3 [==============================] - 0s 41ms/step - loss: 0.0674 - mse: 0.0674 - val_loss:
0.1390 - val_mse: 0.1390
Epoch 35/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0638 - mse: 0.0638 - val_loss:
0.1247 - val_mse: 0.1247
Epoch 36/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0603 - mse: 0.0603 - val_loss:
0.1248 - val_mse: 0.1248
Epoch 37/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0556 - mse: 0.0556 - val_loss:
0.1185 - val_mse: 0.1185
Epoch 38/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0526 - mse: 0.0526 - val_loss:
0.1122 - val_mse: 0.1122
Epoch 39/100
3/3 [==============================] - 0s 32ms/step - loss: 0.0492 - mse: 0.0492 - val_loss:
0.1035 - val_mse: 0.1035
Epoch 40/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0478 - mse: 0.0478 - val_loss:
0.1002 - val_mse: 0.1002
Epoch 41/100
3/3 [==============================] - 0s 45ms/step - loss: 0.0431 - mse: 0.0431 - val_loss:
0.1014 - val_mse: 0.1014
Epoch 42/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0421 - mse: 0.0421 - val_loss:
0.1076 - val_mse: 0.1076
Epoch 43/100
```

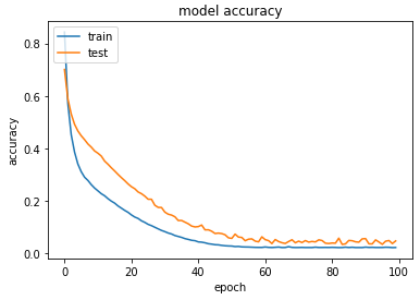
```
3/3 [==============================] - 0s 43ms/step - loss: 0.0397 - mse: 0.0397 - val_loss:
0.0890 - val_mse: 0.0890
Epoch 44/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0362 - mse: 0.0362 - val_loss:
0.0893 - val_mse: 0.0893
Epoch 45/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0346 - mse: 0.0346 - val_loss:
0.0838 - val_mse: 0.0838
Epoch 46/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0324 - mse: 0.0324 - val_loss:
0.0751 - val_mse: 0.0751
Epoch 47/100
3/3 [==============================] - 0s 41ms/step - loss: 0.0319 - mse: 0.0319 - val_loss:
0.0767 - val_mse: 0.0767
Epoch 48/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0298 - mse: 0.0298 - val_loss:
0.0748 - val_mse: 0.0748
Epoch 49/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0284 - mse: 0.0284 - val_loss:
0.0703 - val_mse: 0.0703
Epoch 50/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0276 - mse: 0.0276 - val_loss:
0.0588 - val_mse: 0.0588
Epoch 51/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0272 - mse: 0.0272 - val_loss:
0.0564 - val_mse: 0.0564
Epoch 52/100
3/3 [==============================] - 0s 43ms/step - loss: 0.0253 - mse: 0.0253 - val_loss:
0.0728 - val_mse: 0.0728
Epoch 53/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0259 - mse: 0.0259 - val_loss:
0.0615 - val_mse: 0.0615
Epoch 54/100
3/3 [==============================] - 0s 56ms/step - loss: 0.0241 - mse: 0.0241 - val_loss:
0.0601 - val_mse: 0.0601
Epoch 55/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0237 - mse: 0.0237 - val_loss:
0.0476 - val_mse: 0.0476
Epoch 56/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0235 - mse: 0.0235 - val_loss:
0.0534 - val_mse: 0.0534
Epoch 57/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0229 - mse: 0.0229 - val_loss:
0.0547 - val_mse: 0.0547
Epoch 58/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0223 - mse: 0.0223 - val_loss:
0.0461 - val_mse: 0.0461
Epoch 59/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0221 - mse: 0.0221 - val_loss:
0.0434 - val_mse: 0.0434
Epoch 60/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0220 - mse: 0.0220 - val_loss:
0.0625 - val_mse: 0.0625
Epoch 61/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0237 - mse: 0.0237 - val_loss:
0.0517 - val_mse: 0.0517
Epoch 62/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0222 - mse: 0.0222 - val_loss:
0.0479 - val_mse: 0.0479
Epoch 63/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0217 - mse: 0.0217 - val_loss:
0.0367 - val_mse: 0.0367
Epoch 64/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0225 - mse: 0.0225 - val_loss:
0.0521 - val_mse: 0.0521
Epoch 65/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0234 - mse: 0.0234 - val_loss:
0.0445 - val_mse: 0.0445
Epoch 66/100
3/3 [==============================] - 0s 41ms/step - loss: 0.0216 - mse: 0.0216 - val_loss:
0.0399 - val_mse: 0.0399
Epoch 67/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0219 - mse: 0.0219 - val_loss:
0.0371 - val_mse: 0.0371
Epoch 68/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0249 - mse: 0.0249 - val_loss:
0.0446 - val_mse: 0.0446
Epoch 69/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0222 - mse: 0.0222 - val_loss:
0.0514 - val_mse: 0.0514
Epoch 70/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0217 - mse: 0.0217 - val_loss:
0.0398 - val_mse: 0.0398
Epoch 71/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0218 - mse: 0.0218 - val_loss:
0.0460 - val_mse: 0.0460
Epoch 72/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0221 - mse: 0.0221 - val_loss:
0.0406 - val_mse: 0.0406
Epoch 73/100
3/3 [==============================] - 0s 32ms/step - loss: 0.0216 - mse: 0.0216 - val_loss:
0.0482 - val_mse: 0.0482
Epoch 74/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0217 - mse: 0.0217 - val_loss:
0.0423 - val_mse: 0.0423
Epoch 75/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0216 - mse: 0.0216 - val_loss:
0.0448 - val_mse: 0.0448
Epoch 76/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0230 - mse: 0.0230 - val_loss:
0.0430 - val_mse: 0.0430
Epoch 77/100
3/3 [==============================] - 0s 48ms/step - loss: 0.0218 - mse: 0.0218 - val_loss:
0.0509 - val_mse: 0.0509
Epoch 78/100
3/3 [==============================] - 0s 46ms/step - loss: 0.0219 - mse: 0.0219 - val_loss:
```

```
0.0483 - val_mse: 0.0483
Epoch 79/100
3/3 [==============================] - 0s 27ms/step - loss: 0.0222 - mse: 0.0222 - val_loss:
0.0381 - val_mse: 0.0381
Epoch 80/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0219 - mse: 0.0219 - val_loss:
0.0369 - val_mse: 0.0369
Epoch 81/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0224 - mse: 0.0224 - val_loss:
0.0388 - val_mse: 0.0388
Epoch 82/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0220 - mse: 0.0220 - val_loss:
0.0382 - val_mse: 0.0382
Epoch 83/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0214 - mse: 0.0214 - val_loss:
0.0574 - val_mse: 0.0574
Epoch 84/100
3/3 [==============================] - 0s 42ms/step - loss: 0.0217 - mse: 0.0217 - val_loss:
0.0335 - val_mse: 0.0335
Epoch 85/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0230 - mse: 0.0230 - val_loss:
0.0357 - val_mse: 0.0357
Epoch 86/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0218 - mse: 0.0218 - val_loss:
0.0487 - val_mse: 0.0487
Epoch 87/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0227 - mse: 0.0227 - val_loss:
0.0476 - val_mse: 0.0476
Epoch 88/100
3/3 [==============================] - 0s 44ms/step - loss: 0.0216 - mse: 0.0216 - val_loss:
0.0434 - val_mse: 0.0434
Epoch 89/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0216 - mse: 0.0216 - val_loss:
0.0421 - val_mse: 0.0421
Epoch 90/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0215 - mse: 0.0215 - val_loss:
0.0548 - val_mse: 0.0548
Epoch 91/100
3/3 [==============================] - 0s 42ms/step - loss: 0.0232 - mse: 0.0232 - val_loss:
0.0556 - val_mse: 0.0556
Epoch 92/100
3/3 [==============================] - 0s 41ms/step - loss: 0.0220 - mse: 0.0220 - val_loss:
0.0357 - val_mse: 0.0357
Epoch 93/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0224 - mse: 0.0224 - val_loss:
0.0356 - val_mse: 0.0356
Epoch 94/100
3/3 [==============================] - 0s 43ms/step - loss: 0.0218 - mse: 0.0218 - val_loss:
0.0509 - val_mse: 0.0509
Epoch 95/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0218 - mse: 0.0218 - val_loss:
0.0426 - val_mse: 0.0426
Epoch 96/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0219 - mse: 0.0219 - val_loss:
0.0339 - val_mse: 0.0339
Epoch 97/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0228 - mse: 0.0228 - val_loss:
0.0459 - val_mse: 0.0459
Epoch 98/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0224 - mse: 0.0224 - val_loss:
0.0484 - val_mse: 0.0484
Epoch 99/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0214 - mse: 0.0214 - val_loss:
0.0367 - val_mse: 0.0367
Epoch 100/100
3/3 [==============================] - 0s 42ms/step - loss: 0.0218 - mse: 0.0218 - val_loss:
0.0468 - val_mse: 0.0468
```

Training Results

Loss - 0.02
MSE - 0.02

In [25]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['mse'])
plt.plot(history.history['val_mse'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [30]:
```
#Testing
dir = os.listdir('./test_txt/')
test_cat, test_index = read_txt('./test_txt/',dir)
```

In [31]:
```
'''
Performing Tests and Calculating Score
'''
test_df = []
for i in range(len(test_index)):
    row = []
    for w in keywords:
        vecs = []
        for word in test_cat[w][i]:
            if word in embed_keys:
                vecs.append(embeddings_dict[word])
            else:
                vecs.append(embeddings_dict['<unk>'])
        vecs = np.array(vecs)
        va = np.average(vecs, axis = 0)
        cosine = None
        if np.isnan(va).any():
            cosine = 0
        else:
            vb = jd_vec[w]
            cosine = dot(va,vb)/(norm(va)*norm(vb))

        row.append(cosine)

    test_df.append(row)
```

```
C:\Users\sriva\AppData\Roaming\Python\Python39\site-packages\numpy\lib\function_base.py:380:
RuntimeWarning: Mean of empty slice.
  avg = a.mean(axis)
C:\Users\sriva\AppData\Roaming\Python\Python39\site-packages\numpy\core\_methods.py:170: Run
timeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

In [32]:
```
test_sim = pd.DataFrame(test_df, index=test_index, columns=keywords)
test_sim.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60 entries, candidate_004.txt to candidate_146.txt
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   personal    60 non-null     float64
 1   projects    60 non-null     float32
 2   experience  60 non-null     float32
 3   skills      60 non-null     float32
 4   education   60 non-null     float32
dtypes: float32(4), float64(1)
memory usage: 1.9+ KB
```

In [33]:
```
txs = np.array(test_sim)
print(txs.shape)
```

```
(60, 5)
```

In [72]:
```
ans = model.predict(txs)
ans = np.reshape(ans, (ans.shape[0],1))
ans = ans*100
```

In [35]:
```
def write(ans,test_index):
    ans = np.reshape(ans, (ans.shape[0],1))
    ans_index = np.array([x[:-4] for x in test_index])
    ans_index = np.reshape(np.ravel(ans_index) , ans.shape)
    ans = np.concatenate([ans_index,ans] , axis = 1)
    ans_df = pd.DataFrame(ans, index=None, columns=['CandidateID','Match Percentage'])
    print("Written new file")
    ans_df.to_csv('submission.csv',index=None)
```

RESULTS

In [73]:
```
# ## Trying other models
# from sklearn.svm import SVR
# svr = SVR(C=1.0,epsilon=0.2)
# svr.fit(xs,ys)
# ans_ =svr.predict(txs)
# ans_ = np.reshape(ans_, (ans_.shape[0],1))
# ans_ = ans_*100

# # write(ans_, test_index)
```

In [81]:
```
# take = np.copy(ans_)
# for c in [1,5,10,80,110,120]:
#     for e in [0.1, 0.002, 0.3, 0.83, 0.99]:
#         svr = SVR(C = c, epsilon=e)
#         svr.fit(xs,ys)
#         ansx = svr.predict(txs)
#         ansx = np.reshape(ansx, (ansx.shape[0],1))
#         ansx = ansx*100
#         take = np.concatenate([take,ansx],axis = 1)


# print(take.shape)
```

```
(60, 31)
```

In [82]:
```
# xx = np.concatenate([ans,ans_], axis = 1)
```

```
# xx = np.max(xx, axis = 1)
# xx
# take = np.min(take, axis =1)
# write(take, test_index)
```

In [77]:
```
# write(xx, test_index)
```

In [ ]:
```

```