THE AUTOMATIZATION OF THE CANDIDATE CHOICE FOR PROJECTS IN A CONSULTING FIRM

An extended framework for the tf-idf

LINKÖPING UNIVERSITY

DESCRIPTION

Application of the tf-idf algorithm to automatically recommend employees for project offers. This project offers are described in the e-mails a HHRR employee receives

Carles Sans Fuentes

Text Mining 732A92

Acknowledgments

My greatest gratitude to Urban Jonsson, Salesman and Consultant at *Knowit Decision Linköping AB* for spending time on the creation and the evaluation of the Gold Standard file and the data provided to do this project.

Index

1. Introduction

All consulting firms, without exception, have the need to know which skills their workers have to provide them with a suited project as fast as possible when the opportunity of a project arises. Thus, it is essential to have updated the skills of the employees in a CV to match the market. By doing that the first-mover advantage in the view of a project is reachable, which stands for a raise in the contracts and a higher utilization of your employee resources, translated in more inflows for the firm and thus the possibility of growing and getting a larger portion of the niche of the desired market. That is when I spoke with Urban Johnson, salesman in Linköping for the consulting firm *Knowit*, specialized in technological solutions. On a visit to evaluate possible topics for my master thesis, he claimed that evaluating daily about 20 e-mails concerning project opportunities for the consultants in Linköping was highly time consuming. For this reason, I thought I could try to automatize this part of his job in my text mining project, trying to produce a retrieving algorithm that matches emails with projects to consultants with certain skills gathered on their CVs, giving out the 3 best candidates for a specific offer.

2. Theory: Information Retrieval Systems

Information retrieval is the science of searching for information in a document or for documents themselves. The art of automate such a task has been crucial during this century, enabling the possibility of dealing with an overload of information. Thus, automated information retrieval systems such as search engines (e.g. Google, Yahoo search engines) has made huge impact in our daily life. For that, the exact algorithm behind these well performing searchers is a secret worth millions of dollars.

The theory behind automated retrieval information systems is quite wide and diverse. For that reason, I will only explain the one used in this project: The Term Frequency–Inverse Document Frequency retrieval model (tf-idf). The tf-idf is a numerical statistic number that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document, and it is often offset by the most frequent word in the corpus, which helps to adjust for the fact that some words appear more frequently in general terms.

The tf-idf

The tf part (term frequency) classifies each word in each document by how often does term k_i occurs in document d_j, normalizing by maximum term frequency in that document: $\mathrm{f}_{(\mathrm{i,j})} = \mathrm{freq}_{(\mathrm{i,j})} / \max_{\mathrm{l}} \mathrm{freq}_{(\mathrm{l,j})}$. The idf part (inverse document frequency) accounts for the number of times the term k_i appears in each document, making this be to a measure of importance of the word: the less times you find that word in different documents, the more important that word is. The mathematical expression stands for: $\log(N/n_i)$ where N = total number of documents and n_i = number of documents in which k_i occurs.

Nowadays, tf-idf is one of the most popular term-weighting schemes and 83% of the text-based recommender systems in the domain of digital libraries use tf-idf[1].

The tf-idf for the query

When creating the vector value for the query, the system used is normally slightly different to the tf-idf explained before. There exist several variants related to tf-idf, the most typical one illustrated below as number 1:

[1] *Breitinger, Corinna; Gipp, Bela; Langer, Stefan (2015-07-26). "Research-paper recommender systems: a literature survey". International Journal on Digital Libraries.* ***17****(4): 305–338. doi:10.1007/s00799-015-0156-0. ISSN 1432-5012*

Weighting scheme	Document term weight	Query term weight
1	$f_{i,j} * \log(N/n_i)$	$\left(0.5 + 0.5\left(f_{t,q}/\max_t f_{t,q}\right)\right)\log(N/n_t)$
2	$f_{i,j} * \log(N/n_i)$	$\left(f_{i,j} + \text{func}_{i,j}(\text{param})\right)/\max_i \text{freq}_{i,j}\log(N/n_i)$

However, this typical schema does not work well in our case since it is thought to base our answer on almost no words. For that, I have tried a new approach (see number 2 in the table), which sets additional weight to important words in the query by the *param* variable. These important words, selected by Urban Johnson, account for relevant words he may find in the query (e.g. skills, programs that people in the firm uses). The parameter value, which stands for extra weight given to important words found in the email that matches also their CVs, is optimized over a gold standard file (file of classification of 108 emails with/without attachments against the 18 CVs of the firm, outputting as maximum the three most relevant candidates for each email).

3. Data of the project

The data, provided by Urban Jonsson, consists of the 18 CVs of the employees in Linköping in word format (.docx) and 108 -emails (approximately between 1-2 weeks of job offers) in outlook format(.eml) with attachments in word (.docx) or pdf (.pdf) format.

Information details

CV information

The CVs contain:

- Experience people have had in previous jobs and projects
- Skills the employees have

Emails

Emails can contain diverse levels of information: from a sentence saying "I want a project leader" since e-mails with characteristics and skills the person should have plus requirements of the job that needs to be developed and information about the firm procedure.

Gold Standard file

The Gold Standard file is an excel file containing in columns the name of the email as variable and 3 extra columns accounting for a manual classification of the potential employees that could do the project, having as maximum 3 possible candidates. This Gold Standard file can be found in the excel handed in with together with this report. Apart from the final Gold Standard file, it is also provided the evaluation of the Gold Standard file with the predictions got.

Preprocess of the data

Both CV and emails has been processed in the following way:

- **Tokenize**: Extraction and separation of the text into words using the TweetTokenizer from the nltk.tokenize library.
- **Lowercase**: All words have got replacement of capital letters.

- **Removal of Stop words**: Words that do not contain any relevant meaning has been removed. Those words are accounted in the library nltk from python being called stopwords ("Swedish").
- **Stem**: The remaining words has been shortened by the SnowballStemmer from the nltk library, getting just the root of the words.
- **Join:** Words has been rejoined using space to create a description text.
- **Word Corrections**: Words that appear to be sometimes shortened like projektledare as "pl" has been replaced by the word "projektled" which is the stemmed words for projektledare. On the other hand, pair of words that are important such as data warehouse has been replaced to "dw" to match the important words and deal with potential problems of the algorithm which performs under the assumption of independence of words. This is an interesting improvement since it allows to deal with word dependencies on algorithms that assumes independencies.

4. Methodology

The methodology of the project can be separated into 3 parts:

The treatment of the CV data

Firstly, the text of the CV must be retrieved and put into a dictionary. This can be done using the GetText() function in the script and then using the function zip() of Python to put your text together with the name of word document.

The retrieving process

The **methodology** of the **project** can be mainly **understood by** following the **QueryProcess**() function, which includes four arguments:

- **data**: an objected oriented set of variables that accounts for the store of the important variables:
  - *Emailpath*: list of emails (including the path) that has been passed through the argument *EmailPath* (explained below) of the QueryProcess() function.
  - *bestRec*: matrix in which the indexes of the 3 best candidates for a query are stored.
  - *cosSim*: matrix in which the cosine similarity of the 3 best candidates for a query are stored.
  - *Manip*: By default being False. If True, the parameter and the ImpWords variables will be taken into consideration
  - *ImpWords*: List of words already processed that result to be important for higher weighting on the tf and improvement
  - *Parameter*: float value that stands for the additional weight of the tf ImpWord in the query

- **Emailpath**: string accounting for the query names (including path) of the email that want to be read.
- **NtopResults**: Number of recommendations return for an email read (by default being 3).
- **myCVdict:** Dictionary having as *key* the name of the employee and *value* as Word document text.

Once the variables have been provided, the algorithm performs as following:

1. Preprocess of the CVs according to explanation in the *Preprocess of the data* section.
2. Creation of the tf-idf matrix for the CVs
3. Opening of email:
   a) Path already read:
      i. printing the best 3 recommendations for that email.
   b) Path not read:
      i. Store of different information variables (see QueryProcess() or cvSearcher() to retrieve and see the data stored).

ii. Preprocess of the email (query) according to explanation in the Preprocess of the data.
iii. Creation of the tf-idf matrix for the query.
iv. Store of the results in the variables defined in the cvSearcher()
v. Print of the k (in our case 3) top recommendations

The optimization of the parameter variable

First, a value is set to a variable which is going to be the input parameter for the OptRange() function (e.g. param = 1). Then, OptRange() works as follow:

1. Utilization of the QueryProcess for the specific parameter value provided
2. Comparison of how many matches between the Gold Standard and the prediction has been done, returning the percentage effectiveness of the algorithm between the prediction and Gold Standard. This process is carried out over a grid of values and then chosen the best one according to higher Output of the OptRange().
3. Evaluation of the Gold Standard across several weeks to understand better how the algorithm performs and recommends and evaluate and correct the initial Gold Standard. At the beginning, this evaluation was carried out thoroughly for 20 e-mails, evaluating several times their classification. After that, the other 88 emails were added, and only one general evaluation has been done.

5. Results

The algorithm prediction has been compared to the Gold Standard for a range of values (between 0 and 10) which stands for giving extra importance (and thus some extra weight) to key words. *Figure 2* shows the % of accuracy of the predictions over the grid previously mentioned. Additionally, general conclusions after the first evaluation for the Gold Standard has also been resumed in *Figure 1*.

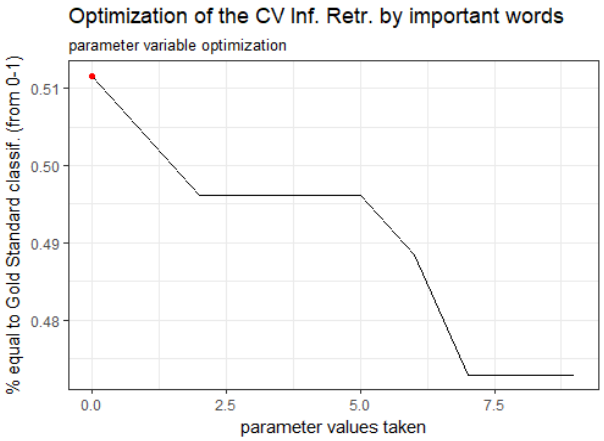

Figure 2 % of accuracy between Gold Standard file and Predictions done using the extended framework for the tf-idf over for the sequence 0-10 by 1 for the parameter value that inputs the WeightWordChan() function

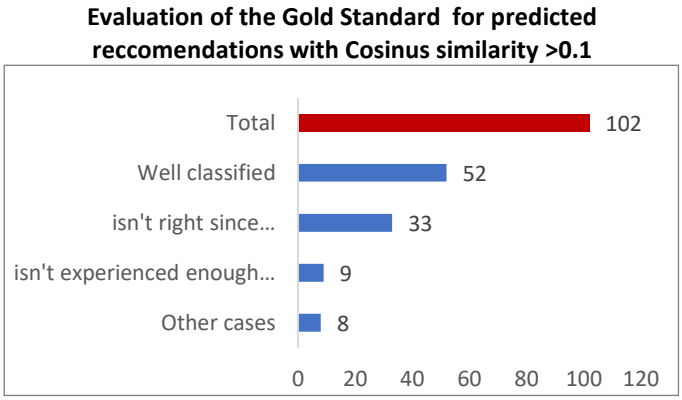

Figure 1 Evaluation of Gold Standard issues on predicted recommendations with Cosines similarity >0.1

6. Discussion

Results show that the tf-idf is already an optimized measure for the search of words. The Importance of knowing keywords for the algorithm to perform better is key to make good translations of words as well as finding relevant synonyms that impact the CV. Nevertheless, giving more importance (extra weight) to key words worsen the algorithm performance despite the time consumed on making this modification efficient.

The correction of the Gold Standard and its iterative evaluation is basic to ensure that the Gold Standard is correctly done. In this case, only one iteration has been made through the whole Gold Standard, and only once feedback has been received on a large scale to assess potential problems derived from either the algorithm or the Gold Standard. Nevertheless, at the beginning of the project (already mentioned in the methodology section), I worked with Urban side by side to evaluate 20 emails, included in the overall Gold Standard to have a first Gold Standard. Before the last correction of the Gold Standard (Provided in an excel file), only about 30% of the e-mails project were correctly allocated to candidates. Nevertheless, after the revision, more than 50% of them were correctly assessed candidates (see *Figure 1*, making reference to the 52 "well classified" reassessed candidates). These probably means that more time should be allocated in creating a good Gold Standard file, but because of time issues, Urban has not had time to evaluate them thoroughly. Getting 50% percent of matches means that, overall, at least 1 of the 3 possible candidates to a position is indeed relevant for that position. This is really good, since it already reduces a lot the time spend in CV finding, given that the search will most surely give you a good candidate for that position.

The problems arisen through this project are the following ones:

I. Emails received which are indistinctively in English and Swedish, and the CVs are only in Swedish. Generally, synonyms for some key words have been translated, but having also the CVs in English could improve the % of accuracy of the algorithm.
II. Candidate recommendation problem. The algorithm does not weight well experience on the job: even if someone is senior on their job and there is a requirement for seniority on a specific field, identifying specific years is something that my text mining procedure does not work well with. This is related to *Figure 1*, labeled as "isn't experienced enough".
III. Identifying pure roles versus non-pure roles: in the framework of tf-idf, separating pure roles from non-pure rules is challenging. "Pure roles" stands for technician versus non-totally technician offer/candidates. Since emails also ask for soft skills in unstructured text, it is difficult to separate skill demands that are related to a technician or vice versa. This is related to *Figure 1*, labeled as "isn't right since". Specific "pure roles" explanations can be found in the excel file, in the page where the Gold Standard evaluation has been done.
IV. Distinguish fundamental skills for an offer: the algorithm tries to match a general CV to an e-mail, but not to a specific skill. That is why I have tried to give more weight to key words, but it has turned out that with the data provided, the results have not improved. This is related to *Figure 1*, englobed in "Other cases".
V. Lack of information about an employee in the CV. It has happened trough the evaluation of the algorithm that sometimes-personal likes make a candidate not available for a position. Also, skills and experience are not fully described, neither included nor well assessed in the CV. Thus, candidates which seems the most optimal ones for the recruiter may not even appear on the recommendation system.

7. Conclusion

In view of the results, it is totally possible to automatize a big part of a recruiter candidate choice, but unless there is a big investment on optimizing a candidate recommendation system, it will just provide an initial point

that could lower the resources allocated on that task (if the firm is big enough) or reduce the amount of time a recruiter performs the task.

During this project, I have written my tf-idf, introducing an extended framework to improve the algorithm's accuracy by specific knowledge on the search, giving more importance to keywords provided by Urban. Nevertheless, it seems the algorithm has not improved accuracy by giving more importance to key words. However, since the dataset used is not really large, more data should be needed to confirm this statement.

In my opinion, the Gold Standard has not been revised thoroughly but just one time given time constraints. Given that a big portion of initial wrong candidates has been reclassified as potential good candidates with my revision, it is in my belief that this could still happen for a couple of more evaluations of the Gold Standard, improving indeed the percentage of accuracy of the algorithm. Thus, the process of candidate evaluation must be carried out slowly and in depth several times to get a fully reliable Gold Standard.

Also, a better choice of keywords could be done, evaluating which ones from the list improve indeed the algorithm and which ones worsen it. To do that, a bigger dataset should be used in order not to overfit the model by writing words for specific cases. Even if I have not written specific code to overfit the data, the data used should be considered training data. It is in my belief that the result on test data would at least maintain that 50% accuracy. Nevertheless, this is something that with more data could be well inferred.

During this project, even if it is not the main scope of the project, I have evaluated the impact of a the cosines similarity measure and the probability a recommendation is significant. On a rule of thumb, I have concluded that over 0.1 (10%) similarity, a recommendation is quite reliable, whereas under 0.08 (8%) similarity, a recommendation should not even be taken into consideration. These may look like a small difference, but it really makes a huge difference on the guess.

Leaving aside the technical part of the project, this project has also involved learning how to work with people that are not that technical in the sector (Urban Johnson), which stands for a really important skill to my future. This project is limited on 3 ECTS credits time, which makes the scope of this project really limited. Gathering more data and CVs to get a more reliable accuracy percentage, trying new variants of the tf-idf and my extended framework, doing model comparisons (e.g bigrams), improving the recommendation system for emails with more than one demand, acquiring more expertise on the field to improve the algorithm for keyword specific skills or experience requirements, setting more time to improve and understand better the Gold Standard file or getting more experts to evaluate the Gold Standard are some of the future steps that could be performed to evaluate and get both to improve and compare the efficacy of the algorithm on this field.

8. Appendix

Python Code

```
1.  # -*- coding: utf-8 -*-
2.  """
3.  Created on Mon Jan 15 12:33:46 2018
4.
5.  @author: Carles
6.  """
7.
8.  import numpy as np
9.  from docx import Document
10.
```

```
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords, words
from nltk.tokenize import TweetTokenizer
import re

from sklearn.metrics.pairwise import cosine_similarity

from email import policy
from email.parser import BytesParser
import datetime

######################################################
#############   Functions
################################################
class cvSearcher(object):
    def __init__(self, Listofemails):
            #Creating a list of ids, emails sent by, habilities and date
        self.count = 0
        self.id_unic=[]
        self.text = []
        self.dateEmail = []
        self.Emailsend=[]
        self.Emailpaths = []
        self.bestRec = np.zeros(shape=(len(Listofemails),3), dtype=np.int) #numpy.int, nump
y.zeros
        self.cosSim =  np.zeros(shape=(len(Listofemails),3))
        self.ImpWords =[]
        self.parameter = 0
        self.manip = False
        self.keyWord = ""
        self.listkeywords =[]
def GetText(filename):
    doc = Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)


###Getting date in proper format
def datetreat(date):
    import email.utils
    import time
    timetransf = email.utils.parsedate(date)
    timetransf = time.mktime(timetransf)
    timetransf = datetime.datetime.fromtimestamp(timetransf)
    return(timetransf)


###Part Process data

def remove_nonalphanum(words):

    words = [re.sub(r'[^a-zA-Z\[\]]',' ' , word) for word in words]

    return words


def tokenize(words):
    tknzr = TweetTokenizer()
    return [tknzr.tokenize(word) for word in words]

def lowercase(words):
    return [[word.lower() for word in lines] for lines in words]

```

```
76.
77.
78.
79.
80. ######################################TO DOOOO
81. def WordCorrections(words):
82. # ================================================================================
83.     ##Important word change manually
84.     words = [lines.replace("  ", " ") for lines in words]
85.     words = [lines.replace("  ", " ") for lines in words]
86.     words = [lines.replace(" erf ", " seni ") for lines in words] ## it is means erfaren (s
    enior)
87.     #words = [lines.replace(" seni ", " ") for lines in words] ## it is means erfaren (seni
    or)
88.     words = [lines.replace(" juni ", " ") for lines in words] ## it is means erfaren (junio
    r)
89.     words = [lines.replace(" cognos ", " cogno ") for lines in words]
90.     words = [lines.replace(" java ", " jav ") for lines in words]
91.     words = [lines.replace(" jira ", " jir ") for lines in words]
92.     words = [lines.replace(" kravspecialist ", " kravanalytik ") for lines in words]
93.
94. #   words = [lines.replace(" led ", " projektled ") for lines in words]
95. #    words = [lines.replace(" processled ", " projektled ") for lines in words]
96. #    words = [lines.replace(" projektadministratör ", " projektled ") for lines in words]

97.     words = [lines.replace(" release ", " releas ") for lines in words]
98.
99.
100.          words = [lines.replace(" pl ", " projektled ") for lines in words]
101.          words = [lines.replace( "data warehouse", "dw") for lines in words]
102.          words = [lines.replace( "teknisk projektled", "tpl") for lines in words]
103.          words = [lines.replace( "technical project manag", "tpl") for lines in words]
104.      #   words = [lines.replace( "lösningsarkitek", "systemutveckl") for lines in words]

105.          words = [lines.replace( "project manager", "projektled") for lines in words]
106.          words = [lines.replace( "project manag", "projektled") for lines in words]
107.
108.          words = [lines.replace( "architect", "arkitek") for lines in words]
109.          words = [lines.replace( "systemarkitek", "arkitek") for lines in words]
110.          words = [lines.replace( "arkitekur product plan", "tpl") for lines in words]
111.          words = [lines.replace( "plan technical arkitek", "tpl") for lines in words]
112.          words = [lines.replace( "arkitekur product plan", "tpl") for lines in words]
113.          words = [lines.replace( "systemansvar", "projektled") for lines in words]
114.      #    words = [lines.replace( "system responsibl", "projektled") for lines in words]

115.      #    words = [lines.replace( "projektledning", "projektled") for lines in words]
116.
117.          return words
118.
119.      ########################################TO DOO
120.      def stemming(words):
121.          st = SnowballStemmer("swedish")
122.          return [[st.stem(word) for word in lines] for lines in words]
123.
124.      def stoppingwords(wordes):
125.          stop= set(stopwords.words('swedish'))
126.          stop.update(['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}
    ', 'br', "st"]) # remove it if you need punctuation
127.          #stop2= set(stopwords.words('english'))
128.          wordes = [[word for word in lines if word not in stop] for lines in wordes]
129.          #wordes = [[word for word in lines if word not in stop2] for lines in wordes]
130.          return wordes
131.      def checkLanguage(data):
132.          checkWords = set(words.words())
133.          return [[word for word in lines if word in checkWords] for lines in data]
134.
```

```
135.        def joiner(words):
136.            return [" ".join(word) for word in words]
137.
138.        def cleaning(words):
139.            words = [re.sub(r'<.+?>', '', word) for word in words]
140.            words = [re.sub(r'\b\w{1}\b', '', word) for word in words] ## remove letters alo
    ne
141.            words = [lines.replace("\r","") for lines in words]
142.            words = [lines.replace("\n","") for lines in words]
143.
144.            return words
145.
146.        def processdata(functions):
147.            #functionList = [remove_nonalphanum, tokenize, lowercase, stoppingwords, WordCor
    rections,checkLanguage ,stemming , joiner]
148.
149.            functionList = [tokenize, lowercase, stoppingwords, stemming , joiner, cleaning,
     WordCorrections]
150.            for function in functionList:
151.                functions = function(functions)
152.            return functions
153.        ########################################################################
154.        ########################################################################3
155.        ## Tfidf implementation of the Searcher
156.        ########################################################################
157.
158.        def tf(documents):
159.            WordList =[]
160.
161.            ##Creating the list of words of all documents being WordList
162.            for document in documents:
163.                splitWords = document.split(" ")
164.                for Word in splitWords:
165.                    if Word not in WordList:
166.                        WordList.append(Word)
167.
168.            ##Matrix with colnames(Documents), rownames = Words
169.            tfperdoc = np.zeros([len(WordList), len(documents)])
170.            for i in range(len(documents)): #### CHange from here
171.                splitWords = documents[i].split(" ")
172.                for Word in splitWords:
173.                    tfperdoc[WordList.index(Word), i] +=1
174.            ##Dividing by the Columnsum and then divide by the larger term in the column
175.            for column in range(len(documents)):
176.                tfperdoc [:,column] = tfperdoc [:,column]/np.sum(tfperdoc [:,column])
177.                tfperdoc [:,column] = tfperdoc [:,column]/np.amax(tfperdoc [:,column])
178.            ###Giving more importance to words
179.            return [tfperdoc, WordList]
180.
181.
182.
183.
184.        def idf(tfres):
185.            #Number of documents in U that contain t
186.            nwords = tfres.shape[0]
187.            ndocs = tfres.shape[1]
188.            wordDoc = np.zeros([nwords, ndocs])
189.            WordCount = np.zeros([nwords])
190.            for i in range(nwords):
191.                for j in range(ndocs):
192.                    if(tfres[i,j]==0):
193.                        wordDoc[i,j]=0
194.                    else:
195.                        wordDoc[i,j]=1
196.            for row in range(nwords):
197.                WordCount[row] = np.sum(wordDoc[row,:])
```

```
198.
199.            for number in range(nwords):
200.                if WordCount[number] == 0:
201.                    WordCount[number] = 0
202.                else:
203.                    WordCount[number] = ndocs/WordCount[number]
204.                    WordCount[number] = np.log(WordCount[number])
205.            return WordCount
206.
207.
208.        def tfidf(tf, idf):
209.            ndocs = tf.shape[1]
210.            for column in range(ndocs):
211.                tf[:,column] = tf[:,column]*idf
212.
213.            return(tf)
214.
215.        def WeightWordChan(data, listOfWords, alltf):
216.        # ================================================================================
217.        #     ### Args:
218.        #     # data.parameter : Set this to how much do you want to affect the effect of th
    at word to the final outcome
219.        #     # data.ImpWords:   List of Words that you want to change the weight in case th
    ey appear on the intersection with listOfWords
220.        #     # listOfWords:List of Words produce by the tf function, being result tf[1]
221.        #     # alltf:      A matrix or vector created in numpy format (also called Alltf)
222.        #     #             Recommended to be set between 0.2 and 1 (it will sum that much t
    o the place where this word appears on the Alltf matrix)
223.        #
224.        #     ###Return:
225.        #     # alltf with matrix modified by manual weights for a matrix
226.        # ================================================================================
227.            CommonWords = list(set(data.ImpWords) & set(listOfWords))
228.        #    print(CommonWords)
229.            if len(alltf.shape)== 2:
230.                if data.keyWord in CommonWords:
231.        #            print(data.keyWord+ "word in the email")
232.                    if alltf[data.keyWord.index(data.keyWord)]>0:
233.                        alltf[listOfWords.index(data.keyWord)] += data.parameter
234.                for word in CommonWords:
235.                    #Check the index on wordlist and pass it to Alltf being Alltf + paramete
    rs = 0.5 for example
236.                    for column in range(alltf.shape[1]):
237.                        if alltf[listOfWords.index(word), column]>0:
238.                            alltf[listOfWords.index(word), column] += data.parameter
239.
240.                return alltf
241.            else:
242.
243.                if data.keyWord in CommonWords:
244.        #            print(data.keyWord+ "word in the email")
245.                    if alltf[data.keyWord.index(data.keyWord)]>0:
246.                        alltf[listOfWords.index(data.keyWord)] += 2*data.parameter
247.                for word in CommonWords:
248.                    if alltf[listOfWords.index(word)]>0:
249.                        alltf[listOfWords.index(word)] += data.parameter
250.                return alltf
251.
252.
253.        def tfidfquery(data, query, wordlist, mytf):
254.        # ================================================================================
255.        #     ### Args:
256.        #     # query:    Message to be taken into account. Must be a string
257.        #     # wordlist: List of Words produce by the tf function, being result tf[1]
258.        #     # tf:       The matrix created in by tf being result tf[0]
```

```
259.       #     # data.manip: if True: the following variables will be passed exclusively to t
    he function WeightWordChan()
260.       #     #     data.ImpWords:   List of Words that you want to change the weight in cas
    e they appear on the intersection with listOfWords
261.       #     #     data.parameter : Set this to how much do you want to affect the effect o
    f that word to the final outcome
262.       #
263.       #     ###Return:
264.       #     # alltf with matrix modified by manual weights
265.       # ================================================================================
266.
267.           queryWord =np.zeros([len(wordlist)])
268.           QuerySplit= query[0].split(" ")
269.
270.           for Word in QuerySplit:
271.               if Word in wordlist:
272.                   queryWord[wordlist.index(Word)] +=1
273.           ##Dividing by the Columnsum and then divide by the larger term in the column
274.           if np.array_equal(queryWord ,np.zeros([len(query)])):
275.               return "No match found"
276.           if  data.manip == True:
277.               queryWord = WeightWordChan(data, listOfWords= wordlist, alltf= queryWord)
278.               queryWord  = queryWord/np.sum(queryWord)
279.
280.           else:
281.               queryWord  = queryWord/np.sum(queryWord)
282.               if  data.manip == True:
283.                       queryWord = WeightWordChan(data, listOfWords= wordlist, alltf= query
    Word)
284.
285.               queryWord = queryWord/np.amax(queryWord)  #query word = tf now
286.               ##Total tf
287.           Alltf = np.c_[mytf, queryWord] # insert values before column 3
288.
289.           ###Doing idf
290.           Allidf= idf(Alltf)
291.           ##tfidif =(0.5+0.5tf)*idf
292.           tfidfRes = queryWord*Allidf #####IF CHANGED TO queryWord*Allidf it works good
293.           return tfidfRes.reshape(-1,1)
294.
295.
296.       ##########################################################################3
297.       ##End of Search algorithm
298.       ###################################################################
299.
300.
301.       ##########################################################################3
302.       ## Main Query
303.       ###################################################################
304.
305.
306.       def QueryProcess(data, Emailpath, NtopResults, myCVdict):
307.
308.           Description = list(myCVdict.values())
309.           IndivNames = list(myCVdict.keys())
310.
311.           k = NtopResults # Number of results to show
312.           ######################   Matrix numerical transformation
313.           Result = processdata(Description) ##Processing data calling function from part2

314.           tf1=tf(Result)
315.           idf1= idf(tf1[0]) ###tf1[0] outputs the tf, tf1[1] outputs the wordList
316.           tfidfMatrix=tfidf(tf1[0], idf1) ###
317.
318.           #Query numerical transformation
319.           with open(Emailpath , 'rb') as fp:
```

```
320.                msg = BytesParser(policy=policy.default).parse(fp)
321.                date =msg["Date"]
322.                EmailDescr = msg.get_body(preferencelist=('plain')).get_content()
323.
324.                if Emailpath in data.Emailpaths:
325.
326.                    print("It has already been evaluated")
327.                    k_rec = data.bestRec[data.Emailpaths.index(Emailpath),:]
328.                    k_recomendations = [IndivNames[i] for i in k_rec ]
329.                    return data, k_recomendations
330.                else:
331.
332.                    data.id_unic.append(data.count)
333.                    data.Emailsend.append(msg['From'])
334.                    data.dateEmail.append(datetreat(date)) ## Passing the function datetreat
    to get the data in a proper string
335.                    data.text.append(EmailDescr)
336.                    data.Emailpaths.append(Emailpath)
337.
338.                    Email = processdata([EmailDescr])###Processing query calling function fr
    om part2
339.                    key = re.search("(?<=sök )\w+", Email[0])
340.
341.                    if(type(key) != type(None)):
342.                        data.keyWord = key.group()
343.                        data.listkeywords.append(key.group())
344.
345.                    tfidfquest=tfidfquery(data,Email, tf1[1], tf1[0]) ##Passing own searcher
    functions
346.
347.                    ###Similarity comparison between matrix and query
348.                    search_result = cosine_similarity( tfidfquest.transpose(),tfidfMatrix.tr
    anspose())
349.                    SearchNiceArray= search_result.reshape(1,search_result .size)[0]
350.                    kTopIndex = SearchNiceArray.argsort(axis = 0 )[::-1][:k].flatten()
351.                    data.bestRec[data.count,:]= kTopIndex
352.                    k_recomendations = [IndivNames[i] for i in kTopIndex ]
353.                    data.cosSim[data.count,:]= ([SearchNiceArray[i] for i in kTopIndex])
354.                    data.count = data.count+1 ### upgrading the counter
355.                    #print([SearchNiceArray[i] for i in kTopIndex])
356.                    return data, k_recomendations
357.
358.
359.
360.        ##########################################################################################
    #########
361.        ############### Running code
362.        ##########################################################################################
    #######
363.                ############Importing libraries
364.
365.        from os import listdir
366.        from os.path import isfile, join, abspath
367.        import sys
368.        sys.path.append(abspath("C:/Users/Carles/Desktop/MasterStatistics-
    MachineLearning/Master_subjects/Text_Mining/Labs/Project"))
369.
370.
371.        ##########################################################################
372.        ##################     Reading the CVs (.docx)
373.        ##########################################################################
374.
375.        mypath = "C:/Users/Carles/Desktop/MasterStatistics-
    MachineLearning/Master_subjects/Text_Mining/Labs/Project/CV"
376.        ##Getting all the file docs from my folder and listing them
377.        mycvs = [f for f in listdir(mypath) if isfile(join(mypath, f))]
```

```
378.
379.
380.        CVexplanation=[] ## Variable list with all the descriptions of my data
381.
382.
383.        for cv in mycvs:
384.            totalpath = mypath +"/"+ cv
385.            CVexplanation.append(GetText(totalpath))
386.
387.        cvDict=  dict(zip(mycvs, CVexplanation)) ##Variable with all key = name of the perso
    n, value = description
388.
389.
390.        ###############################################################################
391.        ##################     Reading the e-mails
392.        ###############################################################################
393.
394.
395.        emailpath = "C:/Users/Carles/Desktop/MasterStatistics-
    MachineLearning/Master_subjects/Text_Mining/Labs/Project/Emails"
396.        ##Getting all the file docs from my folder and listing them
397.        ###############################################################################
398.        ##################     Reading excel gold standard(.docx)
399.        ###############################################################################
400.
401.        GoldFile = "C:/Users/Carles/Desktop/MasterStatistics-
    MachineLearning/Master_subjects/Text_Mining/Labs/Project/20180314FinalEmailEvaluation.xlsx"

402.        import pandas as pd
403.
404.        xl = pd.read_excel(GoldFile, sheetname ="FinalRec")
405.        GoldFramePandas =xl.iloc[:,1:]
406.        GoldEmailName =xl.iloc[:,0]
407.
408.        GoldEmailName = GoldEmailName.values
409.        GoldEmailName[0] =GoldEmailName[0]+".eml"
410.        myemailslist = GoldEmailName
411.
412.        myemailslist == GoldEmailName
413.        #Convert to numpy
414.        GoldNp = GoldFramePandas.values
415.        GoldNp  = np.where(np.isnan(GoldNp), -1, GoldNp)
416.        GoldNp = np.asarray(GoldNp, dtype = int)
417.
418.
419.
420.        ###############################################################################
421.        ##################     Reading excel gold standard(.docx)
422.        ###############################################################################
423.
424.        matches =np.zeros(GoldNp.shape[0], dtype = int) ## Total number of possible matches
    per query
425.        for i in range(GoldNp.shape[0]):
426.            for j in range(GoldNp.shape[1]):
427.                if GoldNp[i,j] >= 0:
428.                    matches[i]+=1
429.
430.        def intersection(pred, Gold):
431.            nrows = pred.shape[0]
432.            intersectionRes =np.zeros([nrows], dtype = int)
433.            for i in range(nrows):
434.                intersectionRes[i] =len(set(pred[i,]) & set(Gold[i,]))
435.            return intersectionRes
436.
437.
438.
```

```
439.        def optRange(numParam):
440.            # ================================================================================
441.        #     ### Args:
442.        #     # sequence:
443.        #     #
444.        #     ###Return:
445.        #     # PercWellClass: percentage between matches/totalgoodpredictions
446.        # ================================================================================
447.            import time
448.            start_time = time.time()
449.
450.
451.
452.            data2 = cvSearcher(myemailslist)
453.            data2.ImpWords = set(['analytic', 'apex','arkitek', 'configuration', 'datamodell
    ering',
454.                                  'dax', 'django', 'excel',  'express',  'förvaltningsled',
455.                                  'integration', 'jir', 'lösning', 'management', 'microsoft'
    ,
456.                                  'mongodb', 'mysql', 'net',  'oracl', 'pow',
457.                                  'prediktiv', 'projektled', 'python', 'qlikview', 'spss', '
    sql',
458.                                  'ssas', 'ssis', 'tableau', 'vba',
459.                                  'visual', 'warehous', 'wcf', 'weblogic', "dw", "nosql", "n
    odej",
460.                                  "bi", "postgresql", 'almexpert', 'alm', 'cam', 'frontend',
461.                                  'it-
    arbetsplat', 'marknadsanalytik', 'sas', 'sap', 'testled',"arkitektur",
462.                                  "pss", "tpl", 'cogno',"jav", 'javintegration', 'informatio
    nsarkitek',
463.                                  'kravanalytik','releas',"systemutveckl","seni"]) ##Definin
    g Words
464.            data2.manip = True
465.
466.            data2.parameter = numParam
467.
468.            for email_id in myemailslist:
469.                totalpath = emailpath +"/"+ email_id
470.                QueryProcess(data2, totalpath, 3, cvDict)
471.
472.            PredIntersect =  intersection(data2.bestRec,GoldNp)
473.
474.
475.            PercWellClass =np.sum(PredIntersect)/np.sum(matches)## Percentage of good
476.            print ("My program took "+ str(time.time() - start_time) + " seconds to run")
477.
478.            return PercWellClass  , data2.bestRec, data2.cosSim, data2.listkeywords
479.
480.        paramRange= np.arange(0, 10, 1)
481.
482.
483.
484.
485.        yRes =[]
486.        for i in range(len(paramRange)):
487.            yRes.append(optRange(paramRange[i])[0])
488.        yRes
489.
490.        # ================================================================================
491.        # result=optRange(0)
492.        # result[0]
493.        # result[1]
494.        # result[2]
```

```
495.       # result[3]
496.       #
497.       # mylist = set(result[3])
498.       #
499.       #
500.       # ================================================================================
501.
502.       ##########################################################################
503.       ##################     Storing variables
504.       ##########################################################################
505.       ##exporting to R
506.               #bestRec
507.               #cosSim
508.               #myemailslist
509.       ##########################################################################
510.
511.       # ================================================================================
512.       # ImpWords = ["deploy", "release","projektledare", "förbättringsarbete", "cognos", "
    wcf",
513.       #             "integration", "lösning", "django","mysql", "microsoft", "oracle","pow
    er", "bi","spss", "tableau",
514.       #             "net", "apex", "c#","configuration", "management",  "warehouse", "data
    modellering",
515.       #             "dax", "excel", "förvaltningsledare","jira", "mongodb", "mysql", "node
    js", "nosql",
516.       #             "express", "pentaho", "pl/sql", "postgresql", "power", "prediktiv", "p
    ython", "qlikview",
517.       #             "sql", "ssas","ssis","vba", "visual", "analytics", "weblogic", 'pow',
    'förbättringsarbet',
518.       #             'jir', 'oracl', 'releas',  'analytic',  'cogno',  'förvaltningsled',
    'warehous', "data warehouse",
519.       #             "plsql", "dw", "nodej", 'projektled', 'dat', 'rele', ]
520.       #
521.       # MatchedWOrds = ['ssis', 'wcf', 'förvaltningsled', 'dax', 'prediktiv', 'visual', 'i
    ntegration', 'warehous', 'oracl', 'projektledare',
522.       #                 'pow', 'weblogic', 'qlikview', 'django', 'warehouse', 'microsoft',
     'analytic', 'vba', 'management', 'spss', 'configuration', 'jir', 'mysql', 'ssas', 'bi', 't
    ableau', 'lösning', 'excel', 'python', 'apex', 'net', 'mongodb', 'express', 'datamodellerin
    g', 'sql']
523.       #
524.       # ================================================================================
525.       #functionList = [tokenize, lowercase, stoppingwords, stemming , joiner, WordCorrecti
    ons]
526.       # ================================================================================
527.       # TotalAMount = set(["deploy", "release","projektledare", "förbättringsarbete", "cog
    nos",
528.       #                           "wcf", "integration", "lösning", "django","mysql", "micr
    osoft", "oracle",
529.       #                           "power", "bi","spss", "tableau", "net", "apex", "c","con
    figuration",
530.       #                           "management",  "warehouse", "datamodellering","dax", "ex
    cel",
531.       #                           "förvaltningsledare","jira", "mongodb", "mysql", "nodejs
    ", "nosql",
532.       #             "express", "pentaho", "pl/sql", "postgresql", "power", "prediktiv", "p
    ython", "qlikview",
533.       #             "sql", "ssas","ssis","vba", "visual", "analytics", "weblogic", 'pow',
    'förbättringsarbet',
534.       #             'jir', 'oracl', 'releas',  'analytic',  'cogno',  'förvaltningsled',
    'warehous',
535.       #             "data warehouse", "plsql", "dw", "nodej", 'projektled', 'rele']) ##Def
    ining Words
536.       #
537.       # ================================================================================
538.
539.
```

```
540.
541.       storePath = "C:/Users/Carles/Desktop/MasterStatistics-
    MachineLearning/Master_subjects/Text_Mining/Labs/Project/DataResults/"
542.       paramOpt = np.asarray(yRes, np.unicode_)
543.       paramRange = np.asarray(paramRange, np.unicode_)
544.
545.       np.savetxt(storePath+'ParamOpt.txt', paramOpt, delimiter=',', fmt="%s")
546.       np.savetxt(storePath+'ParamRange.txt', paramRange, delimiter=',', fmt="%s")
547.       # np.savetxt(storePath+'Cossim.txt', np.around(result[2],3), delimiter=';', fmt="%s"
    )
548.       # np.savetxt(storePath+'BestReccommendations.txt', result[1], delimiter=';', fmt="%s
    ")
549.
550.
551.
552.
553.       # ================================================================================
554.
555.       # #numpymycvs = np.asarray(mycvs, np.unicode_)
556.       # EmailsList= np.asarray(myemailslist, np.unicode_)
557.       #
558.       # #np.savetxt(storePath+'cvNames.txt', numpymycvs, delimiter=',', fmt="%s")
559.       # np.savetxt(storePath+'EmailList.txt', EmailsList, delimiter=',', fmt="%s")
560.       # np.savetxt(storePath+'BestReccommendations.txt', data2.bestRec, delimiter=',',fmt
    = '%i')
561.       # np.savetxt(storePath+'cosSim.txt', data2.cosSim, delimiter=',', fmt = "%10.5f")
562.       #
563.       # ================================================================================
```

R Code (graphic display)

```
#############Comparison of data
storePath = "C:/Users/Carles/Desktop/MasterStatistics-
MachineLearning/Master_subjects/Text_Mining/Labs/Project/DataResults/"


# EmailList<-read.table(paste0(storePath,"EmailList.txt"), sep = "\t")
# EmailList<- lapply(EmailList, as.character)
# BestRecom<-read.table(paste0(storePath,"BestReccommendations.txt"), sep = ",")
ParamOpt<-read.table(paste0(storePath,"ParamOpt.txt"), sep = ",")
ParamRange<-read.table(paste0(storePath,"ParamRange.txt"), sep = ",")

library(dplyr)
library(tidyr)
library(ggplot2)
theme_set(theme_bw())  # pre-set the bw theme.

mydata<- data.frame(x = ParamRange, y = ParamOpt)
colnames(mydata) <- c("x","y")

mydata %>% ggplot(aes(x = x, y = y))+
 geom_line()+
 geom_point(aes(x = x[which.max(y)], y =y[which.max(y)]), col = "red")+
 labs(subtitle="parameter variable optimization",
     y="% equal to Gold Standard classif. (from 0-1)",
     x="parameter values taken",
     title="Optimization of the CV Inf. Retr. by important words")
```