Job Ad Classification Using a TD.IDF Bag-of-words Vector Space

February 15, 2016

Clemens Westrup

1 Introduction

This document describes and implements a prototype to classify job ads using the TF.IDF technique to create a vector space of words. In this space the distance between classes can be measured and new documents are classified by mapping their bag-of-words representation into a vector in the same space with TF.IDF and computing the cosine or jaccard distance to the nearest classes.

This ipython/jupyter notebook can be downloaded and used interactively and contains all code ready to run. As prerequisites python3 must be installed with the packages listed below:

```
In [1]: import logging
        logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

```
In [2]: import requests
        import json
        import numpy as np
        import scipy
        import datetime
        import sklearn.metrics
        import sys
        import random
        import math
        import re
        from pprint import pprint
```

2 Methods

2.1 Dataset: English Job Ads from Oikotie

In order to retrieve labels for the data I downloaded around 6000 english job ads from the Oikotie job platform.

2.2 Preprocessing the Data

These were then split into sections to be classified. The splitting criteria were the character sequences: "
", "</p>" and the unix linebreak character. The code for preprocessing the ads can be found here https://github.com/cle-ment/thesis-tagger/blob/master/pre-processing.ipynb.

2.3 Labelling the Data with a Little Help of My Friends

I then build a tool to ask others to help my manually tag job ads. This tool consists of a Node.js server with using MongoDB for persistence and a HTML5/JS interface. The code and description can be found on GitHub: https://github.com/cle-ment/thesis-tagger, and the interface is online here: http://thesis.cwestrup.de/jobad-tagger/.

The server offers a simple REST API do send and retrieve the job ad data and tags which is documented here: http://thesis.cwestrup.de/jobad-tagger/apidoc/ and can be accessed via http://thesis.cwestrup.de/api.

At the point of writing there were a total of 60 submitted labelled job ads resulting in 304 tags.

2.3.1 Downloading the resulting labelled data

The resulting labelled data can be downloaded using the API:

```
In [3]: def downloadTaggedChunks():
            print("Retrieved 0 tagged chunks. Working...", end="\r")

            url = ("http://cwestrup.de:8082/api/tags/populated");
            file = "data/tagged_chunks"+str(datetime.datetime.now())+".json"
            last_batch_received = False
            size = 100
            page = 1

            with open(file, 'w', encoding='utf8') as json_file:
                json_file.write("[")

            while not last_batch_received:
                params = {'size': size, 'page': page}
                response = requests.get(url, params=params)
                if response.status_code != 200:
                    print("Request not successful")
                    break
                # count items
                json = response.json()
                if (len(json) < size): last_batch_received = True
                data = response.text.lstrip("[").rstrip("]")
                with open(file, 'a', encoding='utf8') as json_file:
                    json_file.write(data)
                    if not last_batch_received:
                        json_file.write(', ')
                        print("Retrieved " + str(page*size) + " tagged chunks. Working...", end="\r")
                        page += 1
                    else:
                        json_file.write("]")
                        print("Retrieved " + str((page-1)*size + len(json)) + " tagged chunks. Done!")
            return file
```

```
In [4]: file = downloadTaggedChunks()
```

```
Retrieved 316 tagged chunks. Done!
```

Let's load the data we just downloaded:

```
In [5]: with open("./"+file) as data_file:
            data = json.load(data_file)
```

We can verify the number of chunks we downloaded:

```
In [6]: len(data)
```

```
Out[6]: 316
```

Let's take a look at all the tags the participants used:

```
In [7]: for document in data:
            sys.stdout.write(document['content'] + ", ")
```

```
employer's information, language requirement, job id, website of employer, start of list, functions, tra
```

2.4 Preprocessing the Labelled Data

We first convert the data to a dictionary in the correct format for further processing

```
In [8]: data_dict = {}
        for document in data:
            for chunk in document['chunks']:
                try:
                    data_dict[chunk['_chunk']['content']].add(document['content'])
                except KeyError:
                    data_dict[chunk['_chunk']['content']] = set([document['content']])
```

Then we'll define a list with stopwords taken from http://www.ranks.nl/stopwords

```
In [9]: stopwords = ("a about above after again against all am an and any are aren't as at be "
                     + "because been before being below between both but by can't cannot could "
                     + "couldn't did didn't do does doesn't doing don't down during each few for "
                     + "from further had hadn't has hasn't have haven't having he he'd he'll he's "
                     + "her here here's hers herself him himself his howhow's i i'd i'll i'm i've "
                     + "if in into is isn't it it's its itself let's me more most mustn't my myself "
                     + "no nor not of off on once only or other ought our ours ourselves out over "
                     + "own same shan't she she'd she'll she's should shouldn't so some such than "
                     + "that that's the their theirs them themselves then there there's these they "
                     + "they'd they'll they're they've this those through to too under until up very "
                     + "was wasn't we we'd we'll we're we've were weren't what what's when when's "
                     + "where where's which while who who's whom why why's with won't would wouldn't "
                     + "you you'd you'll you're you've your yours yourself yourselves - . :").split()
```

And we need a function to "lemmatize" or process each word (very simple so far):

```
In [10]: def lemmatize(word):
             # TODO: do a bit more sophisticated processing here
             return word.lower().strip().strip(".").strip(",").strip(":").strip("-")
```

2.5 Building a Vocabulary for Words and Tags

We'll build a vocabulary for all tags and words in the data and assign each of them to a unique id (the position in the array). At the same time the words are "lemmatized" or processed.

```
In [11]: def buildVocab(data_dict):
             vocab_words = [] # array to store all words (and their indices)
             vocab_tags  = [] # array to store all tags (and their indices)

             for chunk, tags in data_dict.items():
                 if re.match(r'^\s*$', chunk): continue # skip empty chunks
                 words = [lemmatize(word) for word in chunk.split() if not lemmatize(word) in stopwords]
                 for word in words:
```

```
                # add word to vocabWords
                try:
                    word_id = vocab_words.index(word)
                except ValueError:
                    vocab_words.append(word)
                    word_id = len(vocab_words)

            for tag in tags:
                # append tag to tag vocab
                try:
                    vocab_tags.index(lemmatize(tag))
                except ValueError:
                    vocab_tags.append(lemmatize(tag))

        return vocab_words, vocab_tags
```

```
In [12]: vocab_words, vocab_tags = buildVocab(data_dict);
```

```
In [13]: len(vocab_words), len(vocab_tags)
```

```
Out[13]: (3165, 298)
```

We'll need a couple of functions to convert the id's and strings for words and tags into one another

```
In [14]: def word2id(word, vocab_words):
             return vocab_words.index(word)

         def id2word(id, vocab_words):
             return vocab_words[id]

         def tag2id(tag, vocab_tags):
             return vocab_tags.index(tag)

         def id2tag(id, vocab_tags):
             return vocab_tags[id]
```

2.6 Constructing a Wordcount Matrix

We'll go through the dataset and create a sparse matrix of the form *words* × *tags* with the frequency for each word in with each tag. In information retrieval terms the words here are terms and each tag represents a document (or rather the words assigned to it).

```
In [15]: def buildWordcountMatrix(data_dict, vocab_words, vocab_tags):
             wordcounts = scipy.sparse.lil_matrix((len(vocab_words), len(vocab_tags)), dtype=int)
             for chunk, tags in data_dict.items():
                 words = [word for word in chunk.split() if not lemmatize(word) in stopwords]
                 for word in words:
                     for tag in tags:
                         tag_id = tag2id(lemmatize(tag), vocab_tags)
                         word_id = word2id(lemmatize(word), vocab_words)
                         wordcounts[word_id, tag_id] += 1

             return wordcounts
```

```
In [16]: wordcounts = buildWordcountMatrix(data_dict, vocab_words, vocab_tags)
```

2.7 Computing the Tf.Idf Matrix

Now we'll convert the wordcount data into a TF.IDF matrix as in [1, section 1.2.1] with

$$TF_{ij} = \frac{f_{ij}}{max_k f_{kj}}$$

$$IDF_i = log_2(N/n_i)$$

```
In [17]: def computeTFIDF(wordcounts):
             max_term_freqs = np.amax(wordcounts.A, axis=0)
             total_tags = wordcounts.shape[1]
             total_words = wordcounts.shape[0]

             # compute TF
             # term/doc frequency / maximum frequency of all terms in that document
             tf = wordcounts.tocsr().multiply(scipy.sparse.csr_matrix(1 / max_term_freqs))

             # compute IDF
             # total documents / num of documents with term i
             idf = scipy.sparse.csr_matrix(np.log2(total_tags / wordcounts.tocsr().getnnz(axis=1)))

             # compute TF.IDF
             tfidf_matrix = tf.transpose().multiply(idf).transpose()

             return tfidf_matrix;
```

```
In [18]: tfidf_matrix = computeTFIDF(wordcounts)
```

We can also use this matrix to find the most similar words for each tag and vice versa and to find the most similar tags by computing the distance between them. Both of these side experiments can be found in the appendix.

2.8 Classifying New Texts

To classify new texts we'll compute the TF.IDF vector for the new chunk and return the most similar tag vectors

```
In [19]: def classifyChunk(text, tfidf_matrix, vocab_tags, vocab_words,
                           distance_threshold=0.9, return_max=5):
             total_tags = len(vocab_tags)
             total_words = len(vocab_words)
             # collect frequencies of known words
             occurrences = np.zeros((total_words,1))
             words = text.split()
             words_found = False
             for word in [lemmatize(word) for word in words]:
                 try:
                     word_id = word2id(word, vocab_words)
                     occurrences[word_id] += 1
                     words_found = True
                 except ValueError:
                     continue
             # if no words found to classify return empty result
             if not words_found: return []
             # calculate TF.IDF
```

```
        max_term_freq = occurrences.max()
        num_of_docs_with_term = tfidf_matrix.getnnz(axis=1)
        tf = occurrences / max_term_freq
        idf = scipy.log2(total_tags / num_of_docs_with_term)
        tfidf_vec = tf.reshape(-1) * idf
        # compute distances to all tag vectors
        distances = sklearn.metrics.pairwise_distances(tfidf_matrix.transpose(),
                                                       tfidf_vec.reshape(1, -1), metric="cosine")
        #
        distance_tuples = [(id2tag(id, vocab_tags), item[0])
                           for id, item in enumerate(distances) if item[0] < distance_threshold]
        return sorted(distance_tuples, key=lambda x:x[1])[0:return_max]
```

Let's try it out:

```
In [20]: text = ("Do you want to join our team? Please leave your application and CV no later "
                 + "than 20.9.2015: http://www.fennovoima.fi/en/jobs/open-positions. For further "
                 + "information, please contact Psycon consultant Rauna Kautto tel. 020 7101 242 "
                 + "on Wednesday 16.9. at 9-10 or on Friday 18.9. at 14-15. More information about "
                 + "Fennovoima you will find on our website: www.fennovoima.fi.")
```

```
In [21]: classifyChunk(text, tfidf_matrix, vocab_tags, vocab_words)
```

```
Out[21]: [('application submission', 4.4408920985006262e-16),
          ('application instructions', 0.29764818254826508),
          ('contact information', 0.56496033241087573),
          ('contact', 0.60759132544406558),
          ('further info', 0.79354848870005745)]
```

2.9 Random baseline

We'll also need a baseline for comparision. This will be a function that randomly guesses n labels out of the available labels.

```
In [22]: def classifyChunkRandomGuess(vocab_tags, return_max=5):
             return np.random.choice(vocab_tags, return_max)
```

```
In [23]: classifyChunkRandomGuess(vocab_tags)
```

```
Out[23]: array(['bonus skills', 'signature', 'detailed information on productline',
                'company culture', 'unit of the company'],
               dtype='<U45')
```

3 Experiments

Finally to analyze the results let's define a function for cross-validating with the dataset. We'll predict max n (3 by default) labels with the algorithm and interpret the result as a correct classification if there is an intersection between the predicted labels and the true labels. Also the crossvalidation can run the random guesser as a baseline

```
In [24]: def crossvalidate(data_dict, tfidf_matrix, vocab_tags, vocab_words, folds=5,
                           return_max=3, baseline=False, verbose=False):
             correct_total = 0
             test_size_total = 0
```

```
for i in range(folds):
    # training set size 80%
    train_set_size = math.floor(len(data_dict)*0.7)
    test_set_size = len(data_dict) - train_set_size

    correct = 0

    # shuffle data and select subsets
    keys = list(data_dict.keys())
    random.shuffle(keys)
    train_keys = keys[0:train_set_size]
    train_data_dict = { key:value for key,value in data_dict.items() if key in train_keys }
    test_keys = keys[train_set_size+1:len(keys)]
    test_data_dict = { key:value for key,value in data_dict.items() if key in test_keys }

    # run the whole training pipeline
    vocab_words, vocab_tags = buildVocab(train_data_dict);
    wordcounts = buildWordcountMatrix(train_data_dict, vocab_words, vocab_tags)
    tfidf_matrix = computeTFIDF(wordcounts)

    # classify all chunks in the test set and compute the error for each
    for chunk, tags in test_data_dict.items():
        if (baseline):
            predictions = classifyChunkRandomGuess(vocab_tags, return_max=return_max)
            break
        else:
            predictions = classifyChunk(chunk, tfidf_matrix, vocab_tags,
                                        vocab_words, return_max=return_max)

        # skip if no predictions made
        if predictions == []: continue

        # find intersection of predicted and true tags
        pred_tags = [tag[0] for tag in predictions]
        tags_intersect = set(tags).intersection(pred_tags)

        # if tags were correctly predicted increase correctly classified item count
        if tags_intersect != set(): correct += 1

        # print true labels and predictions
        if verbose:
            print("true labels: " + str(tags))
            print("predictions: " + str(predictions))
            print("intersection: " + str(tags_intersect))

    correct_total += correct
    test_size_total += test_set_size

    # print info on current iteration
    print("[" + str(i+1) + "/" + str(folds) + "]"
          + " Correctly predicted: " + str(correct) + " of " + str(test_set_size)
          + " (" + "{:.2f}".format(correct/test_set_size) + ")")
```

```
        print("\n" + "In total (" + str(folds) + "-fold CV):")
        print("Correctly predicted: " + str(correct_total) + " of " + str(test_size_total)
              + " (" + "{:.2f}".format(correct_total/test_size_total) + ")")
```

3.1 Setting 1: Predicting a Maximum of 5 Tags

Now let's run 10-fold CV with our classifier while allowing our classifier to predict a maximum of 5 tags

```
In [25]: crossvalidate(data_dict, tfidf_matrix, vocab_tags, vocab_words, folds=10, return_max=5)
```

```
[1/10] Correctly predicted: 83 of 291 (0.29)
[2/10] Correctly predicted: 90 of 291 (0.31)
[3/10] Correctly predicted: 89 of 291 (0.31)
[4/10] Correctly predicted: 90 of 291 (0.31)
[5/10] Correctly predicted: 92 of 291 (0.32)
[6/10] Correctly predicted: 92 of 291 (0.32)
[7/10] Correctly predicted: 97 of 291 (0.33)
[8/10] Correctly predicted: 93 of 291 (0.32)
[9/10] Correctly predicted: 89 of 291 (0.31)
[10/10] Correctly predicted: 104 of 291 (0.36)

In total (10-fold CV):
Correctly predicted: 919 of 2910 (0.32)
```

Next we'll run 10-fold CV the random baseline for comparision

```
In [26]: crossvalidate(data_dict, tfidf_matrix, vocab_tags,
                       vocab_words, folds=10, return_max=5, baseline=True)
```

```
[1/10] Correctly predicted: 0 of 291 (0.00)
[2/10] Correctly predicted: 0 of 291 (0.00)
[3/10] Correctly predicted: 0 of 291 (0.00)
[4/10] Correctly predicted: 0 of 291 (0.00)
[5/10] Correctly predicted: 0 of 291 (0.00)
[6/10] Correctly predicted: 0 of 291 (0.00)
[7/10] Correctly predicted: 0 of 291 (0.00)
[8/10] Correctly predicted: 0 of 291 (0.00)
[9/10] Correctly predicted: 0 of 291 (0.00)
[10/10] Correctly predicted: 0 of 291 (0.00)

In total (10-fold CV):
Correctly predicted: 0 of 2910 (0.00)
```

3.2 Setting 2: Predicting a Maximum of 3 Tags

Now we'll do the same thing using a maximum of 3 allowed tags for prediction

```
In [27]: crossvalidate(data_dict, tfidf_matrix, vocab_tags, vocab_words, folds=10, return_max=3)
```

```
[1/10] Correctly predicted: 65 of 291 (0.22)
[2/10] Correctly predicted: 91 of 291 (0.31)
[3/10] Correctly predicted: 66 of 291 (0.23)
[4/10] Correctly predicted: 69 of 291 (0.24)
[5/10] Correctly predicted: 83 of 291 (0.29)
[6/10] Correctly predicted: 64 of 291 (0.22)
[7/10] Correctly predicted: 62 of 291 (0.21)
```

```
[8/10] Correctly predicted: 72 of 291 (0.25)
[9/10] Correctly predicted: 72 of 291 (0.25)
[10/10] Correctly predicted: 64 of 291 (0.22)

In total (10-fold CV):
Correctly predicted: 708 of 2910 (0.24)
```

```
In [28]: crossvalidate(data_dict, tfidf_matrix, vocab_tags,
                       vocab_words, folds=10, return_max=3, baseline=True)
```

```
[1/10] Correctly predicted: 0 of 291 (0.00)
[2/10] Correctly predicted: 0 of 291 (0.00)
[3/10] Correctly predicted: 0 of 291 (0.00)
[4/10] Correctly predicted: 0 of 291 (0.00)
[5/10] Correctly predicted: 0 of 291 (0.00)
[6/10] Correctly predicted: 0 of 291 (0.00)
[7/10] Correctly predicted: 0 of 291 (0.00)
[8/10] Correctly predicted: 0 of 291 (0.00)
[9/10] Correctly predicted: 0 of 291 (0.00)
[10/10] Correctly predicted: 0 of 291 (0.00)

In total (10-fold CV):
Correctly predicted: 0 of 2910 (0.00)
```

3.3 Setting 3: Predicting Only the Most Likely Tag

And again with only the most likely tag

```
In [29]: crossvalidate(data_dict, tfidf_matrix, vocab_tags, vocab_words, folds=10, return_max=1)
```

```
[1/10] Correctly predicted: 41 of 291 (0.14)
[2/10] Correctly predicted: 31 of 291 (0.11)
[3/10] Correctly predicted: 37 of 291 (0.13)
[4/10] Correctly predicted: 41 of 291 (0.14)
[5/10] Correctly predicted: 42 of 291 (0.14)
[6/10] Correctly predicted: 35 of 291 (0.12)
[7/10] Correctly predicted: 38 of 291 (0.13)
[8/10] Correctly predicted: 33 of 291 (0.11)
[9/10] Correctly predicted: 36 of 291 (0.12)
[10/10] Correctly predicted: 36 of 291 (0.12)

In total (10-fold CV):
Correctly predicted: 370 of 2910 (0.13)
```

```
In [30]: crossvalidate(data_dict, tfidf_matrix, vocab_tags,
                       vocab_words, folds=10, return_max=1, baseline=True)
```

```
[1/10] Correctly predicted: 0 of 291 (0.00)
[2/10] Correctly predicted: 0 of 291 (0.00)
[3/10] Correctly predicted: 0 of 291 (0.00)
[4/10] Correctly predicted: 0 of 291 (0.00)
[5/10] Correctly predicted: 0 of 291 (0.00)
[6/10] Correctly predicted: 0 of 291 (0.00)
[7/10] Correctly predicted: 0 of 291 (0.00)
[8/10] Correctly predicted: 0 of 291 (0.00)
[9/10] Correctly predicted: 0 of 291 (0.00)
```

```
[10/10] Correctly predicted: 0 of 291 (0.00)

In total (10-fold CV):
Correctly predicted: 0 of 2910 (0.00)
```

4 Results

The first learning was that people don't follow instructions and e.g. empty sections were tagged, leading to zero division errors due to the word frequencies being zero. Also tags were partially done in Finnish and were sometimes not seperated by comma but rather marked with the hash sign (e.g. #finlandjob). Some of these mistakes were directly corrected in the preprocessing while others weren't yet (seperating tags is not done yet).

Given the intersection of tags is a proper metric to measure performance of a multiclass predictor the results are surprisingly good using the rather simple TF.IDF method. While the random baseline does not get a single guess right, the classifier predicts matching classes in around on third of the cases for the first setting with 5 output classes. Even when allowing only 1 output class it gets a match in 12% of the cases.

This result is especially good considering that tags are not processed in any way, meaning that they the predicted tags have to match literally ("languages" does not match "language").

5 Conclusion and Future Work

While the experiment gave good results with the chosen metrix, several issues or potential improvements can be pointed that could be further developed.

5.1 Improved Preprocessing for Reducing Effects of Human Error

In not so fancy words this means that some tags were not seperated by comma as mentioned in the result section and instead tagged by a hash character. This and other obvious and common mistakes should be resolved in the preprocessing step.

5.2 Metrics for multiclass classification

The metric for multiclass classification is somewhat questionable and should be defined better. One possible way to design the metric could be to output confidence results by taking the proportion of tags that match. On the other hand there is also a lot of noise in the data and it is questionable if the true labels given can be seen as a real ground truth to be learned as closely as possible or rather a noisy dataset to generalize from.

5.3 Removing Redundancy

Many words and tags have redundant representations in this application context ("language" and "languages" refer to the same concept). This makes it harder for the classification to predict the correct results and could be improved with the following approaches:

1. Thresholding frequencies: First words and tags that only appear n times (e.g. only once) could be removed as they don't carry much information.

2. Lemmatization: So far there was no real lemmatizing of words and tags was done (apart from lower-casing them). Doing this would greatly help to reduce the amount of redundancy in the mapping and almost certainly improve the results significantly. This could also involve resolving synomyms using e.g. WordNet.

3. Using Relations in the Vector Space: As shown in Appendix C the TF.IDF vector space can be used to identify very similar tags. This could help to even further reduce redundant tags that are not captured by any lemmatization (e.g. synomnyms like "skills" and "requirements"). This needs some

more experimentation though as the distance is not the only relation between the tag vectors (think of subsets of words describing tag that belongs to a sub-hierarchy of another).

An interesting side-effect was that since some people did not read the instructions and tagged in Finnish the mapping effectively acts as a translation (see Appendix C). This could be further looked into.

6 References

1. Leskovec J, Rajaraman A, Ullman JD (2014) Mining of massive datasets. Cambridge University Press.

7 Appendix

7.1 A: Best of Tags

Just for fun here's a list with the most interesting or surprising tags that participants used:

1. 90% bullshit (I wonder how that is measured)
2. bullshit
3. selling the job
4. empy (yes "empy", there's also empty which doesn't make sense either since empty sections should be ignored)
5. "#kemira #finlandjob", (tags not seperated by comma and use the #)
6. useless info
7. crap
8. kuvaus (etc. somebody tagged in Finnish)

Conclusion: People don't follow (or read) instructions. Nevertheless I am of course extremely thankful for all the participants :)

7.2 B: Finding the Most Likely Words for Each Tag and vice Versa

We can compute a distance matrix between the tag vectors (classes) to find very similar classes:

```
In [31]: def getMostLikelyWords(tag, tfidf_matrix, vocab_words, vocab_tags):
             tag_id = tag2id(tag, vocab_tags)
             nonzeros = tfidf_matrix[:,tag_id].nonzero()[0]
             return_tags = {}
             for word_id in nonzeros:
                 return_tags[id2word(word_id, vocab_words)] = tfidf_matrix[word_id, tag_id]
             return sorted(return_tags.items(), key=lambda x:x[1], reverse=True)
```

```
In [32]: def getMostLikelyTags(word, tfidf_matrix, vocab_words, vocab_tags):
             word_id = word2id(word, vocab_words)
             nonzeros = tfidf_matrix[word_id,:].nonzero()[1]
             return_tags = {}
             for tag_id in nonzeros:
                 return_tags[id2tag(tag_id, vocab_tags)] = tfidf_matrix[word_id, tag_id]
             return sorted(return_tags.items(), key=lambda x:x[1], reverse=True)
```

Let's test this:

```
In [33]: getMostLikelyTags("finnish", tfidf_matrix, vocab_words, vocab_tags)
```

```
Out[33]: [('language', 3.4642810182986929),
          ('language skill', 3.4642810182986929),
          ('languages', 3.4642810182986929),
          ('language requirements', 3.4642810182986929),
          ('skill exception', 3.4642810182986929),
          ('actual requirements', 3.4642810182986929),
          ('job requirements', 2.3095206788657952),
          ('important requirements', 1.7321405091493465),
          ('level', 1.7321405091493465),
          ('common requirements', 1.7321405091493465),
          ('tech', 1.7321405091493465),
          ('innovation', 1.7321405091493465),
          ('contact agent', 1.7321405091493465),
          ('habits', 1.7321405091493465),
          ('language skills', 1.3857124073194773),
          ('requirements', 1.3857124073194773),
          ('expectations', 1.1547603394328976),
          ('about', 1.1547603394328976),
          ('language requirement', 0.86607025457467324),
          ('skill', 0.76984022628859838),
          ('summary', 0.74234593249257697),
          ('skills', 0.70860293556109633),
          ('role', 0.69285620365973866),
          ('requirement', 0.57738016971644879),
          ('responsibilities', 0.53296631050749121),
          ('experience', 0.33525300177084122),
          ('tasks', 0.28869008485822439)]
```

```
In [34]: getMostLikelyWords("language", tfidf_matrix, vocab_words, vocab_tags)
```

```
Out[34]: [('norwegian', 7.2191685204621612),
          ('speaking', 7.2191685204621612),
          ('writing', 6.6342060197410051),
          ('fluency', 4.8972404255747994),
          ('least', 4.7597369018248639),
          ('fluent', 3.8268510976834014),
          ('finnish', 3.4642810182986929),
          ('english', 2.971241007018576)]
```

7.3 C: Computing a Distance Matrix Between Tags

We can use the TF.IDF matrix to compute the most likely words given a tag and vice versa:

```
In [35]: jaccard_kernel = sklearn.metrics.pairwise_distances(tfidf_matrix.transpose().A,
                                                             metric="jaccard")
```

Let's try to find similar tags for each tag. (Note: the distance is given in brackets, not the similarity):

```
In [36]: similarTags = []
         for i, row in enumerate(jaccard_kernel):
             similarTags.append([])
             for (j, value) in [(tag_id, value)
                                    for tag_id, value in enumerate(jaccard_kernel[i,:]) if value < 0.88]:
                 if (j == i): continue
                 similarTags[i].append(j)
```

```
            if similarTags[i] != []:
                sys.stdout.write(id2tag(i, vocab_tags) + ": ")
                for tag in similarTags[i]:
                    sys.stdout.write(id2tag(tag, vocab_tags) + "(" + "{:.2f}".format(value) + "), ")
                sys.stdout.write("\n")
```

```
wished skills: start of skills list(0.88), heading(0.88), department(0.88), useless info(0.88), ask(0.88
position type: extent(0.75), employment type(0.75), job form(0.75),
extent: position type(0.75), employment type(0.75), job form(0.75),
header: section title(0.53),
details: about job agent(0.84),
offering: employer's information(0.60), company(0.60),
title: position level(0.34), task titles(0.34), team title(0.34),
contact ddetails: contact details(0.83), job desription(0.83),
job benefits: travel requirements(0.73), oppoturnity(0.73),
additional job opportunities: application deadline(0.85),
dealine: about job agent(0.86),
previous experience: background(0.83), job specific tag(0.83),
product line: company details(0.00),
company details: product line(0.00),
#kemira #finlandjob: department(0.83), what(0.83), where(0.83), who(0.83), need(0.83), ask(0.83), search
about: unit description(0.77),
unit description: about(0.00),
location of job: job classification(0.86),
actual requirements: languages(0.88),
application information: application deadline(0.78), website contact info(0.78), application practicali
habits: important requirements(0.88), technical skills(0.88),
important requirements: habits(0.88), technical skills(0.88),
start of skills list: wished skills(0.00),
employer listing: further information(0.85),
further information: employer listing(0.00),
requirements title: additional information(0.88), prerequisite studies(0.88), application procedure(0.88
communication skill: 90% bullshit(0.60), communication(0.60),
employer's information: offering(0.56), company(0.56),
company: offering(0.00), employer's information(0.00),
about job agent: details(0.00), dealine(0.00),
crap: job contact information(0.75), job number(0.75), number(0.75), travel requirements(0.75), tracking
heading: wished skills(0.75), department(0.75), useless info(0.75), ask(0.75), search(0.75),
language skill: bonus skills(0.79), languages(0.79), skill exception(0.79), language(0.79),
attachments: application procedure(0.86), deadline for application(0.86), how to apply?(0.86), applicati
bonus skills: language skill(0.00),
contact details: contact ddetails(0.83), job desription(0.83),
competence: communication skills(0.75), 90% bullshit(0.75), communication(0.75),
closing date: opening date(0.83), dates(0.83),
additional information: requirements title(0.67), prerequisite studies(0.67), further info(0.67), more i
prerequisite studies: requirements title(0.83), additional information(0.83), start of list(0.83),
responsibilities heading: checklist for job description(0.67),
department: wished skills(0.81), #kemira #finlandjob(0.81), heading(0.81), what(0.81), where(0.81), who
contact person title: employer description(0.83),
innovation: tech(0.00),
tech: innovation(0.00),
time commitment: time(0.80), employment type(0.80), info(0.80), job classification(0.80),
time: time commitment(0.80), employment type(0.80), info(0.80), job classification(0.80),
```

employment type: position type(0.62), extent(0.62), time commitment(0.62), time(0.62), info(0.62), job i
info: time commitment(0.80), time(0.80), employment type(0.80), job classification(0.80),
what: #kemira #finlandjob(0.82), department(0.82), where(0.82), who(0.82), need(0.82), ask(0.82), searc
where: #kemira #finlandjob(0.82), department(0.82), what(0.82), who(0.82), need(0.82), ask(0.82), searc
who: #kemira #finlandjob(0.82), department(0.82), what(0.82), where(0.82), need(0.82), ask(0.82), searc
conditions: information(0.58),
application deadline: additional job opportunities(0.87), application information(0.87), website contac
communication skills: competence(0.71), 90% bullshit(0.71), communication(0.71),
further info: additional information(0.00), more info(0.00), contact practicalities(0.00),
job contact information: crap(0.75), job number(0.75), number(0.75), travel requirements(0.75), tracking
job number: crap(0.75), job contact information(0.75), number(0.75), travel requirements(0.75), tracking
job id: website of employer(0.50), application procedure(0.50), deadline for application(0.50),
website of employer: job id(0.50), application procedure(0.50), deadline for application(0.50),
employer info: job related field(0.00),
job related field: employer info(0.00),
more info: additional information(0.80), further info(0.80), contact practicalities(0.80),
work experience: background(0.83), job specific tag(0.83),
opening date: closing date(0.83), dates(0.83),
need: #kemira #finlandjob(0.83), what(0.83), where(0.83), who(0.83), job duration(0.83), work title(0.83
section title: header(0.00),
background: previous experience(0.88), work experience(0.88), job specific tag(0.88), programming skill
website contact info: application information(0.78), application deadline(0.78), application practicali
encouragement: application deadline(0.88), invitation(0.88), job classification(0.88), application prac
invitation: application deadline(0.00), encouragement(0.00),
personal: company culture(0.00),
company culture: personal(0.00),
non-discrimination: equal opportunity(0.00), company policy(0.00),
equal opportunity: non-discrimination(0.00), company policy(0.00),
company policy: non-discrimination(0.00), equal opportunity(0.00),
compensation: marketing of the company(0.88), treats(0.88),
marketing of the company: compensation(0.88), treats(0.88),
languages: actual requirements(0.70), language skill(0.70), skill exception(0.70), language(0.70),
number: crap(0.75), job contact information(0.75), job number(0.75), travel requirements(0.75), tracking
langugage skills: technical skills(0.07),
useless info: wished skills(0.50), heading(0.50), department(0.50), ask(0.50), search(0.50),
time frame: job(0.88), #clinicalcontractcoordinator #bayerrecruiting(0.88),
job: time frame(0.00),
job type: kuvaus(0.67),
analysis: mobile development(0.46),
job duration: need(0.86), practicalities(0.86),
practicalities: job duration(0.84), introduction to organization(0.84),
java: programming skills(0.80),
90% bullshit: communication skill(0.86), competence(0.86), communication skills(0.86), communication(0.8
#clinicalcontractcoordinator #bayerrecruiting: time frame(0.00),
job specific tag: previous experience(0.86), work experience(0.86), background(0.86), programming skill
technical skills: habits(0.00), important requirements(0.00), langugage skills(0.00),
skill exception: language skill(0.88), languages(0.88), language(0.88), job desription(0.88),
introduction to organization: practicalities(0.00),
language: language skill(0.00), languages(0.00), skill exception(0.00),
job form: position type(0.00), extent(0.00), employment type(0.00),
applicant's personality: applicant's abilities(0.00),
applicant's abilities: applicant's personality(0.00),
employer description: contact person title(0.00),
job classification: location of job(0.00), time commitment(0.00), time(0.00), info(0.00), encouragement

```
programming skills: background(0.00), java(0.00), job specific tag(0.00),
ask: wished skills(0.86), #kemira #finlandjob(0.86), heading(0.86), department(0.86), what(0.86), where
search: wished skills(0.86), #kemira #finlandjob(0.86), heading(0.86), department(0.86), what(0.86), wh
checklist for job description: responsibilities heading(0.75), start of list(0.75),
signal for interview question: background(0.88), job desription(0.88),
travel requirements: job benefits(0.83), crap(0.83), job contact information(0.83), job number(0.83), n
mobile development: analysis(0.00),
communication: communication skill(0.88), competence(0.88), communication skills(0.88), 90% bullshit(0.
application procedure: requirements title(0.88), attachments(0.88), job id(0.88), website of employer(0
position level: title(0.00), task titles(0.00), team title(0.00),
task titles: title(0.00), position level(0.00), team title(0.00),
team title: title(0.00), position level(0.00), task titles(0.00),
soft skills: wished skills(0.00), communication(0.00),
start of list: requirements title(0.00), additional information(0.00), prerequisite studies(0.00), chec
treats: compensation(0.00), marketing of the company(0.00),
oppoturnity: job benefits(0.00),
tools: wished skills(0.83), background(0.83), job specific tag(0.83), design(0.83),
performance: 90% bullshit(0.00), communication(0.00),
deadline for application: requirements title(0.88), attachments(0.88), job id(0.88), website of employe
dates: closing date(0.00), opening date(0.00),
transition: expected values(0.88), job position(0.88), application practicalities(0.88),
design: tools(0.00),
istqb: communication(0.00),
traveling requirements: employee qualities(0.00),
employee qualities: traveling requirements(0.00),
job desription: contact ddetails(0.00), contact details(0.00), skill exception(0.00), signal for interv
field of operations: unit of company(0.00),
unit of company: field of operations(0.00),
tracking: crap(0.00), job contact information(0.00), job number(0.00), number(0.00), travel requirement
expected values: transition(0.00),
how to apply?: attachments(0.00),
contact practicalities: additional information(0.00), further info(0.00), more info(0.00),
job position: transition(0.86), introductory phrase(0.86),
kuvaus: job type(0.00),
introductory phrase: job position(0.00),
project management: bugtracking tool(0.00),
bugtracking tool: project management(0.00),
information: conditions(0.00),
location data: ccompany details(0.00),
ccompany details: location data(0.00),
application practicalities: application information(0.00), attachments(0.00), website contact info(0.00
work title: #kemira #finlandjob(0.00), department(0.00), what(0.00), where(0.00), who(0.00), need(0.00)
```

We can see that in many cases semantically similar concepts were mapped close to each other in the vector space which is a very good sign. An side-effect is that also some Finnish tags (while they should not exists) are effectively translated this way, e.g. “kuvaus” meaning “description” maps to “job type”.