Alcatel·Lucent Enterprise

Openclassrooms - Parcours Data Scientist

Projet N° 6

Catégorisez automatiquement des questions

Classify questions automatically

DOCUMENT	
Reference	**File Name**
	Openclassroms Data Scientist projet 6 rapport Christian Muths.docx

Versions	
Date	**Reason for change**

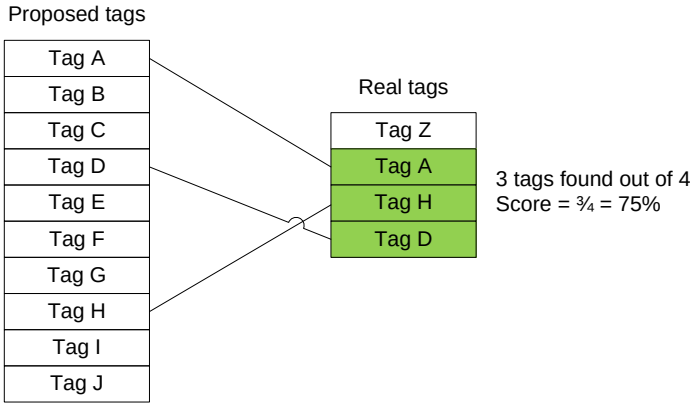
Table of contents

0. Introduction

The target of this project is to propose to a user creating a question on the stackoverflow.com site, a set of relevant tags.

The algorithms propose 5 to 10 tags of which the user is invited to pick and choose some of them to add to the question. Performance has been evaluated both with 5 and with 10 tags proposed. In the case tags are proposed to the user to choose from 10 tags make sense. If the system should automatically assign tags without user intervention, 5 tags looks more relevant.

Whatever the algorithm, the performance measurement is based on a comparison with the real tags the user has given. Out of the real tags, we measure how many we guessed. For example, if the user has set 4 tags, and if in the proposed list 3 of them were proposed, the score for this question is 75%



The overall performance indicator is the relative number of questions for which the proposed tags contain at least 50% of the tags the user had chosen.

1. Principle of tag recommendation system based on non-supervised algorithm

This algorithm is based on a non-supervised topic extraction, associated with a statistical approach to get a list of the most likely tags to propose.

Based on a list of questions having titles, descriptions and human-selected tags, several data structures are being built:

1.1 List of stems

A stem is the root of a word, regardless of the suffixes, conjugation, gender, etc... This reduces the number of values and helps grouping similar notions. However, it may lose some useful information.

A column is added into the questions dataset with the list of stems for each question. Stems are extracted both from the title and the body.

The HTML content of the description gets its HTML tags cleaned out, then a stemmer is used to extract the useful words into stems.

The steps to get the list of stems for each question are:

- Concatenate title and body
- Use BeautifulSoup to remove HTML tags
- Convert into lower case
- Tokenize the resulting string into an array of words
- Remove stopwords using the nltk.stowords list for English language.
- Convert remaining words into stems using nltk SnowballStemmer

- Append the list of stems into a new column of the questions dataframe.

Out of 15.000 questions, we got a total of approx. 1 Million stem occurrences, with 40.000 unique stems.

The set of stems gets vectorized. Each question is a line of the matrix and each stem is a column. The values are the number of occurrences of each stem in each question. Any word that is used in more than 95% of the questions or less than 5 times across all questions is removed, because its either too common or too specific to be used for the topic determination.

The result is the stem_matrix which has 5.755 stems in columns, and 15.000 lines. Only approx. 25% of the initial stems were kept. The figures given here are about monograms. Same vectorization could be done with multigrams.

5700 stems as columns

15.000 questions as lines $\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & e \\ d & 0 & 0 & 0 \\ 0 & b & 0 & 0 \end{pmatrix}$

stem_matrix

```
Maximum number of stems per question: 2386
Minimum number of stems per question: 2
Average number of stems per question: 56
Average number of unique stems per question: 32
```

1.2 Matrix of topics per question

1.2.1 Topic extraction

The method used to extract topics is the Latent Dirichlet Allocation. An optimization is done on the number of topics as a parameter.

The measurement of score and Perplexity shows that:

- The score doesn't vary much with the number of topics
- the perplexity strictly increases with the number of topics when tested between 2 and 90. It almost doubles with each additional topic. It's close to 1000*2^(ntopics-1).

These measurements aren't useful to choose the optimal number of topics. The final evaluation will be based on the ratio of correct tags proposed.

As an example, with 15 topics, here are the top words for each of them. Some of them are intuitive (Topic 0 = git, Topic 1 = Python, Topic 3 = numbers, dates and times, Topic 6 = SQL databases, Topic 11 = CSS), while some others are less meaningful in a human perspective.

```
Topic #0: file git command use directori commit run chang project branch local
work path want window folder line repositori tri script

Topic #1: python test line number print import self py modul def virtualenv get
key return code str file function tri call

Topic #2: java class public method object new string static void return thread
privat call except system null properti code use get

Topic #3: date 00 10 11 05 12 format 15 time 01 02 03 datetim 24 androidruntim
19 13 20 26 18
```

```
Topic #4: use differ code understand googl mean question map read vs oper
implement data time one warn could applic compil know

Topic #5: div use control set class html work item id model tag element name
scope like get properti content ng want

Topic #6: tabl column databas key sql select null queri data mysql row id valu
insert creat name server updat set db

Topic #7: function var name js javascript json valu node php type foo npm option
script data input get return jqueri text

Topic #8: android instal version org error id packag build com lib eclips
librari app activ java xml usr project view depend

Topic #9: error user app get server http request use applic com system url web
tri connect log run messag client password

Topic #10: int array return valu function std type size integ byte swift length
fals true doubl element char convert const number

Topic #11: text color css imag background style width left center border 100
right height box margin size pad align red label

Topic #12: button page view click event jqueri want use div li chang imag class
show element like disabl load work tab

Topic #13: net framework td asp height mvc entiti tr use width font differ size
linq space pixel content scale pdf vs

Topic #14: string use way like list would one want someth know need exampl
variabl get best two case look code question
```

1.2.2 Question-topic probability

The result of the LDA transformation lda.fit_transform(stem_matrix) is the matrix of probability for a document to belong to a topic.

15.000 questions as columns

15 topics as lines $\begin{pmatrix} 0.15 & 0.75 & 0.05 & 0.20 \\ 0.55 & 0.02 & 0.15 & 0.45 \\ 0.10 & 0.03 & 0.55 & 0.25 \\ 0.15 & 0.05 & 0.15 & 0.05 \\ 0.05 & 0.15 & 0.1 & 0.05 \end{pmatrix}$

lda_corpus.T

1.3 Tag decomposition

First step is to vectorize the list of tags assigned by the author. The result is a sparse matrix with

5400 tags as columns

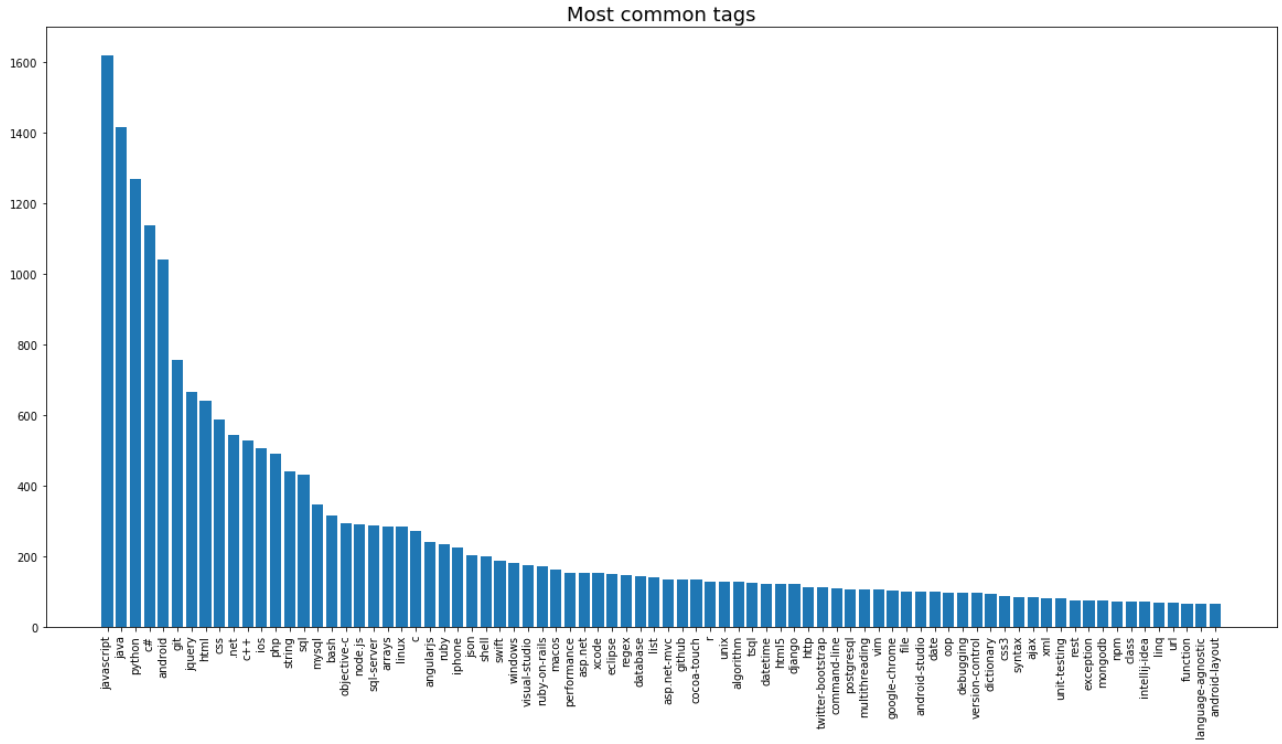
15.000 questions as lines $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

tag_matrix

Some statistics about tags distribution shows that some tags are much more used than the average. This has the impact on the performance of the algorithm, because most used tags have a very high probability to be proposed, whatever the question content.

It also gives the number of how many tags should be proposed to the user to pick and choose from.

```
Maximum number of tags per question: 5
Minimum number of tags per question: 1
Average number of tags per question: 3
```



1.4 Tag weight per topic

A matrix that gives the weight of each tag for each topic is created by multiplying the lda_corpus matrix by the tag_matrix. Weight can be greater than 1 if a tag is used several times in a topic.

$$\begin{pmatrix} 0.15 & 0.75 & 0.05 & 0.20 \\ 0.55 & 0.02 & 0.15 & 0.45 \\ 0.10 & 0.03 & 0.55 & 0.25 \\ 0.15 & 0.05 & 0.15 & 0.05 \\ 0.05 & 0.15 & 0.1 & 0.05 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0.35 & 0.05 & 0.75 \\ 1.00 & 0.15 & 0.02 \\ 0.35 & 0.55 & 0.03 \\ 0.20 & 0.15 & 0.05 \\ 0.10 & 0.1 & 0.15 \end{pmatrix}$$

lda_corpus.T tag_matrix topic_tag_weight

1.5 New questions classification

New questions are classified into topics using the classifier which was trained on the train set. The result is the vector of the question's probabilities to belong to each topic.

The steps followed to reach this are very similar to the process done on the known questions:

- Concatenate title and body
- Use BeautifulSoup to remove HTML tags
- Convert into lower case
- Tokenize the resulting string into an array of words

- Remove stopwords using the nltk.stowords list for English language.
- Convert remaining words into stems using nltk SnowballStemmer
- Vectorize the list of stems using the vectorizer which was fitted with the training set. This ensures to stick to the same set of stems.
- Transform the stem matrix using the LDA transformer which was fitted with the training set. This ensures to use the same exact topics and gets the probabilities for each new question to belong to each topic.

1.6 Tag suggestions for new questions

The probability matrix for each new question to belong to a topic is multiplied by the matrix of tag weight per topic. This gives a distribution of tags with their weight for each question. The 5 to 10 questions having the highest weight are proposed.

1.7 Optimizations

1.7.1 Normalize the matrix of weights

Some tags are over-represented, which makes them appear almost all the time, even if they are not relevant. To reduce this effect, the matrix of weights is normalized on the topics axis.

This change increases the performance by 3 to 6 points

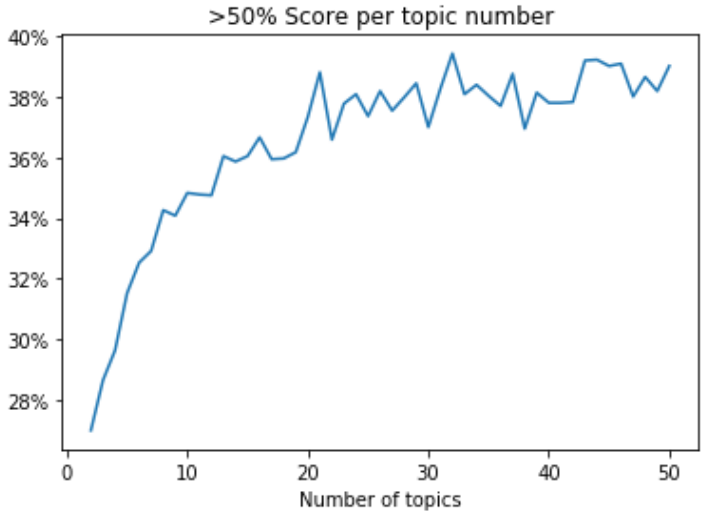
1.7.2 Add bigrams

A try has been made to include bigrams in addition to monograms. The total number of monograms + bigrams is 25500 compared to 5700 monograms. With this algorithm, there is no improvement doing so.

1.7.3 Grid search to find the optimum number of topics

For each case mentioned above, a search of the best number of topics is done.

	5 tags	10 tags
Monograms without normalization	27% (20 topics)	36% (21 topics)
Monograms with normalization	29% (43 topics)	39% (32 topics)
Bigrams without normalization	24% (43 topics)	33% (43 topics)
Bigrams with normalization	28% (49 topics)	39% (50 topics)



Example of performance increase depending on number of topics (monograms only, normalized weights).

2. Principle of tag recommendation system based on supervised algorithm

2.1 List of stems

This algorithm uses the same matrix of stems as the non-supervised algorithm, both for monograms only as well as bigrams added. → X matrix

As a reminder

The steps to get the list of stems for each question are:

- Concatenate title and body
- Use BeautifulSoup to remove HTML tags
- Convert into lower case
- Tokenize the resulting string into an array of words
- Remove stopwords using the nltk.stowords list for English language.
- Convert remaining words into stems using nltk SnowballStemmer
- Append the list of stems into a new column of the questions dataframe.

The Python code for this part is

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from bs4 import BeautifulSoup

# Take only alphanumeric words, no punctuation signs
tokenizer = nltk.RegexpTokenizer('\w+')

# Prepare set of stopwords
stopWords = set(stopwords.words('english'))

# Define stemmer
snowball_stemmer = SnowballStemmer("english")

wordsFiltered = []
wordsArray = []

for html_text in df_questions['body'] + " " + df_questions['title']:
    soup = BeautifulSoup(html_text, "lxml").get_text()
    words = tokenizer.tokenize(soup.lower())
    his_words = ''
    for w in words:
        if w not in stopWords:
            stem = snowball_stemmer.stem(w)
            wordsFiltered.append(stem)
            his_words = his_words + ' ' + stem
    wordsArray.append(his_words)
```

Count vectorizer is used for monograms and bigrams. This procedure has many options which would allow much less preprocessing. The raw text could be passed to the vectorizer, and options set to get a similar result. However, to be sure to compare the algorism on the same basis, the preprocessing has been maintained the same.

```
from sklearn.feature_extraction.text import CountVectorizer
stem_vectorizer = CountVectorizer(lowercase = True, ngram_range=(1, 2),
max_df=0.95, min_df=5)
stem_matrix = stem_vectorizer.fit_transform(df_questions['words'])
```

Tags are vectorized using a similar process, but without removing any data.

```
# Vectorize all the tags used in the training set
tag_vectorizer = CountVectorizer(lowercase = True, max_df=1.0, min_df=0,
token_pattern = '[^<>]+')
tag_matrix = tag_vectorizer.fit_transform(df_questions['tags'])
```

2.2 Multi-class SVM

A multi-class SVM is used to get the probability of each tag to belong to a question. This algorithm is first fit on the training set. This results in training as many SMV classifiers as lablels to predict.

- Matrix of monograms/bigrams for each question → X matrix
- Labels to predict are the tags given by the users. → Y matrix

A multi-class SVM is trained on the X/Y values.

The trained classifier is used to classify new questions with the predict_proba method. The result is the probability for each of this new question to use one of the tags.

The 5 to 10 tags having the highest probability are proposed to the user to qualify his question.

Results are much better than the non-supervised methodology.

Using bigrams in addition to monograms increases the score.

	5 tags	10 tags
Monograms	58%	68%
Monograms + Bigrams	63.7%	71.5%

2.3 API development

The training of the multi-label SVM takes a long time. With 25000+ bigrams, it takes about 7 hours on a standard laptop to fit. This process is done offline.

The trained classifier is dumped into a pickle file to be used in the API. This pickle file is not small. For 25000+ bigrams it's 270MB big.

A new question requires a title and a description body. Both contents are sent to the API using POST request to avoid encoding special characters in the URL. The processing of these data is

- Concatenate title and body
- Use BeautifulSoup to remove HTML tags
- Convert into lower case
- Tokenize the resulting string into an array of words
- Remove stopwords using the nltk.stowords list for English language.
- Convert remaining words into stems using nltk SnowballStemmer
- Use the classifier's predict_proba method to get a vector of probability for each tag to be associated to this question.
- Select the 5 to 10 tags having the highest probability and display them to the user.

3. Conclusion

The supervised algorithm gives much better results. It yields more than 70% of performance with the defined measurement method, whereas the non-supervised algorithm remains below 40%

The fitting time of the supervised algorithm is very long, but it can run off-line and will not harm the user.

The time to classify one new question with the fitted classifier is fast enough to be used on a web site.

Globally, even with the better results of the multi-class algorithm, it's not realistic to build a complete automatic classification program. The ratio of wrong tags would still be too high. With the current solution, it's better to propose the list of tags to the user, and that the user chooses the tags most relevant to him.

END OF DOCUMENT