

Machine Discovery-Final Project

Tags Prediction on Stack Exchange

網媒所 R05944014 何文琦

化工系 B01504044 宋易霖

資工系 B04902072 陳威嘉

I 、 Abstract

On the Internet, lots of people like to share and exchange their experiences to others. However, the database is too large to search, we like to give some tags to those paragraphs and make a classification for others can later use tags to find the proper information.

In this project, given lots of paragraphs, we would like to discovery the tags in each paragraph, we applied three methods: TF-IDF, Textrank and RAKE to find tags and use Word2Vec [1] try to improve the result.

II 、 Data

Our data are from **Kaggle**. There are several fields of datasets like cooking, traveling and physics. Each input file includes many paragraphs. Each paragraph is composed of “id”, “title”, “content” and “tags” rows. In row “title”, there is a question about this field and the answers of this question are in row “content”. The row “tags” shows the tags of this sub data. We only use row “title” and content” to train our model, the row “tags” is used to evaluate our results.

III 、Method



Figure 1. Our process

First, we need to transform all verbs and nouns in the data set to their “simplest form”. For example, “is” will be transformed to “be”, “cookies” will become “cookie”. This step will let us regard these “actually same” words the same. We use “Polyglot” and “Pattern”, which are tools implemented by python, to help us do preprocessing. Polyglot can recognize part of speech of each words in a sentence, and then use Pattern to transform “nouns” and “verbs” to their original type.

In following algorithm, we will remove stop words and punctuations. The stop words are something like “I”, “You”, “What”, which cannot increase the recognition of each paragraph. The result after preprocessing show as Figure 2.

```
3
What is the difference between white and brown eggs?
3
What be the difference between white and brown egg?
3
difference white brown egg
```

Figure 2. Example sentence after preprocessing

After preprocessing, we start to apply three algorithms: TF-IDF, Textrank and RAKE to find tags in paragraph.

TF-IDF

For the first method, we use the simplest method TF-IDF to calculate words score in each paragraph. The value increases proportionally to the number of times a

word appears in the document, but is offset by the frequency of the word in the whole corpus, which helps to adjust for the fact that some words appear more frequently in general.

TF (Term frequency): the number of times that word occurs in document.

IDF (Inverse document frequency): measure of how much information the word provides, that is, whether the term is common or rare across all documents.

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

- t : word, d : document, N : total number of documents in the corpus
- $|\{d \in D : t \in d\}|$: number of documents where the term t appears

TF-IDF is calculated as

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

A high weight in TF-IDF is reached by a high term frequency and a low document frequency of the term in the whole collection of documents. Hence, the higher TF-IDF is, the more possible to be chosen as tags.

Textrank

For the second method, we use the Textrank algorithm to extract keywords from a paragraph. Given a paragraph, we construct a graph as follows. Each word is represented a vertex in the graph, and two vertexes are connected if there are corresponding words both appear within a window of the paragraph. For example, if the paragraph is 'systems linear constraints' and the length of the window is two, then there are edges between 'systems' and 'linear' and between 'linear' and 'constraints'.

Compatibility of systems of linear constraints over the set of natural numbers. Criteria of compatibility of a system of linear Diophantine equations, strict inequations, and nonstrict inequations are considered. Upper bounds for components of a minimal set of solutions and algorithms of construction of minimal generating sets of solutions for all types of systems are given. These criteria and the corresponding algorithms for constructing a minimal supporting set of solutions can be used in solving all the considered types systems and systems of mixed types.

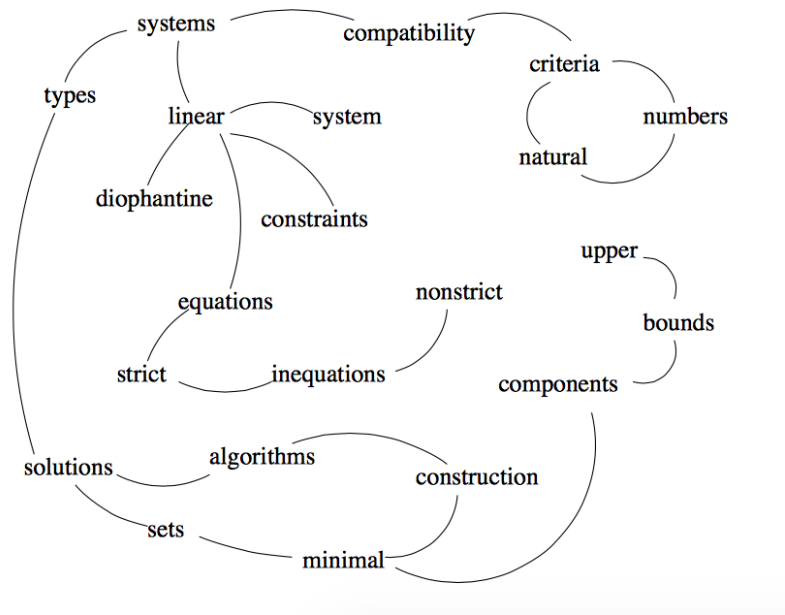


Figure 3. Graph illustration

Given a graph $G = (V, E)$, let v be a vertex, we compute the score of v as:

$$S(v) = (1 - d) + d * \sum_{u \in \text{deg}(v)} \frac{S(u)}{|\text{deg}(u)|}$$

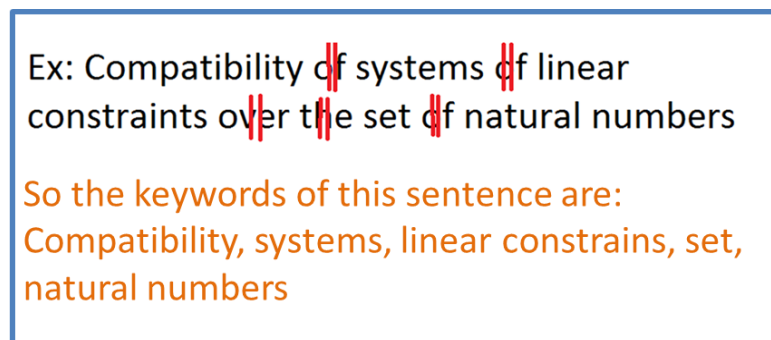
where $d \in [0, 1]$ is a damping factor and $\deg(\cdot)$ is the degree of a vertex. We find out the keywords of a paragraph as follows:

1. Assign arbitrary values to each vertex in the graph.
2. Iterate each node – compute their scores until convergence.
3. Select top T^{th} words as keywords of this paragraph.

RAKE

The third algorithm is Rapid Automatic Keyword Extraction (RAKE)[3], which can extract key words from single paragraph, too.

RAKE will treat every segment which formed after we remove all stop words as a candidate of keyword. And then use $\frac{\text{degree}(\text{word})}{\text{frequency}(\text{word})}$ as score function to evaluate keywords. Figure 4. and 5. is an example.



Ex: Compatibility of systems of linear constraints over the set of natural numbers

So the keywords of this sentence are:
Compatibility, systems, linear constrains, set, natural numbers

Figure 4. Example of RAKE

Frequency(word) means how many times does “word” appear, while degree(word) means that how “word” interact with other keywords. Ex: linear constraints=> degree(linear) = 2 (interact with “constraints” and itself). After defining frequency and degree, the final score can be computed: $\text{score}(\text{word}) = \frac{\text{degree}(\text{word})}{\text{frequency}(\text{word})}$, and $\text{score}(\text{complex word}) = \sum_{x=\text{each words}} \text{score}(x)$. Actually, these scores can be computed according to a table, just like Table 1. When a word appears, just add 1 at the table[word][word]. If there is complex word, add 1 at table[word][each other word]. Finally, the diagonal is words’ frequency, and the sum of each rows are words’ degree.

Table 1. Compute the frequency and degree

	compatibility	system	linear	constraint	set	natural	number
compatibility	1						
system		1					
linear			1	1			
constraint			1	1			
set					1		
natural						1	1
number						1	1

Score of each words: compatibility (1), system (1), linear (2), constraints (2), set (1), natural (2), number (2).

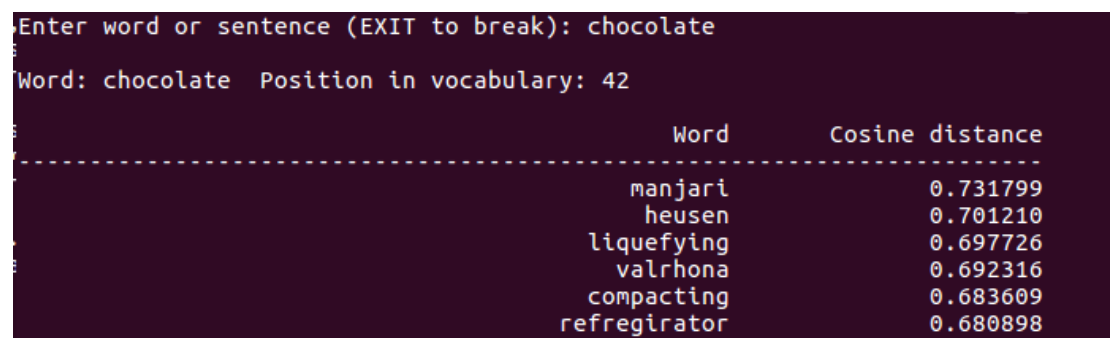
And the score of each candidate is: linear constraint (4), natural number (4), compatibility (1), system (1), set (1).

Figure 5. Example of RAKE (cont'd)

Enhance – Word2Vec

After applying above methods, we could only extract the tags which are actually in each paragraph. Those tags don't appear in the paragraph are impossible to be found, but in our project, these "hidden" tags are important. For example, we can imagine that some words are highly correlated with other words, like cookies and bake: we won't "fry" cookies or "boil" cookies, so maybe we tag both "cookies" and "bake" can help us to classify articles. Therefore, we used Word2Vec to enhance our results.

Word2Vec is a tool which provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. We download this tool from Google and took the whole text corpus as input then produced the word vectors as output. After training, if we enter 'chocolate'; it will display the most similar words and their distances to 'chocolate', which should look like Figure 6.

A screenshot of a terminal window with a dark purple background and light purple text. The prompt 'Enter word or sentence (EXIT to break):' is followed by the input 'chocolate'. Below this, it says 'Word: chocolate Position in vocabulary: 42'. Then, a table is displayed with two columns: 'Word' and 'Cosine distance'. The table lists six words and their corresponding cosine distances to 'chocolate'.

Word	Cosine distance
manjari	0.731799
heusen	0.701210
liquefying	0.697726
valrhona	0.692316
compacting	0.683609
refregirator	0.680898

Figure 6. The most similar words to chocolate

Therefore, with this trained vector, we can use the tags found in one paragraph to get other related tags which didn't show up in that paragraph. Nevertheless, we found out sometimes the closest words are not common in whole corpus and are not suitable to be the tags. To solve this problem, we set a threshold to determine whether the words are able to become tags, we define a function of the words frequency and their distance, and use this function to decide which word will be chosen.

Beside only use one method to find original tags, we tried to combine all tags which are found in four methods (including baseline tags) and applied Word2Vec. In this part, Word2Vec is used in different way if our tag candidates are too much; we wouldn't find more tags but try to remove the most unrelated word in combined tags. Word2Vec can use the distance between words to find out which word doesn't match other words. For example, given a list with ['breakfast', 'cereal', 'dinner', 'lunch'], the Word2Vec function "*doesn't_match*" will return 'cereal' for other words are more

similar to classify into a category. With this function, we can remove the most different tags in combined tags and increase the accuracy.

For evaluation, we apply F1 score to measure our accuracy. It considers both the precision and the recall of the test to compute the score. Precision is the number of correct positive results divided by the number of all positive results, and recall is the number of correct positive results divided by the number of positive results that should have been returned. The F1 score can be interpreted as a weighted average of the precision and recall.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

tp: # of true positive instances

fp: # of false positive instances

$$\text{Recall} = \frac{tp}{tp + fn}$$

fn: # of false negative instances

Our Baseline is defined as choosing the nouns which appear both in “title” and “content” to become tags. And the baseline of Cooking filed is about 0.299.

IV 、Result

Cooking

Tags number	1	2	3	4	5	6
Textrank	0.26047	0.32279	0.30537	0.28787	0.26875	0.25180
TF-IDF	0.19733	0.25564	0.26847	0.26227	0.24918	x
RAKE	0.163	0.201	0.277	0.229	x	x
Textrank + Word2Vec	0.2432	0.2652	0.2334	0.2685	0.27	0.2623

Baseline: 0.299

We can find out that using Textrank will have the best result and the maximum is 0.3227, which is higher than baseline. Later, we apply Word2Vec on Textrank result to find more tags, yet the result is not well in finding only 1~4 tags. We considered the reason is Word2Vec has find out too much wrong tags so the f1 score decrease. But, while finding 5 and 6 tags in each paragraph, Word2Vec indeed improved the result. Hence, Word2Vec really can find out some right tags but we still cannot successfully detect the wrong tags and dispose them.

Traveling

Tags number	1	2	3	4	5	6
Textrank	0.0934	0.14156	0.16381	0.17257	0.17327	0.17227
TF-IDF	0.08829	0.13392	0.15721	0.16862	0.17314	0.1734
RAKE	0.036	0.054	0.064	0.069	x	x

Physics

Tags number	1	2	3	4	5	6
Textrank	0.13858	0.18041	0.19326	0.19760	0.19499	0.18877
TF-IDF	0.05348	0.08667	0.10077	0.11274	0.11698	0.11744
RAKE	0.065	0.094	0.111	0.118	x	x

Combined Tags

Our policy to combine tags which generated by all methods is simple: if the tags occur in more than 2 methods, we considered it has high probability to be real tag.

datasets	Cooking
Combined tag	0.3374

We can find that combined tags have little improvement, and then we try apply Word2Vec to it.

1. Remove tags if the amount of tags is larger than k:

k	4	5	6
F1 score	0.3018	0.317	0.3224

From the above table we can found that we use Word2Vec to remove the most different tags is not work well. Because if our recall is 0.3 and we predict 3 tags, it means only 1 tag is right while other 2 tags might not alike the right tag, so we will remove the right tag when we use Word2Vec. This mention us finding better keywords in each paragraph has large impact to the performance of Word2Vec.

2. Add Tags by Word2Vec

After we combined the tags, then each tags are very like to be the real tag or alike the real tag, so we can use these tags to predict some “hidden” tag. This method also induced a worse result, but it sometimes can really predict “hidden” tags.

Show some results as following table.

Predict tags	Real tags	Result (top 20 alike)
Cookies, chip	Cookies, bake(not in the paragraph), texture(not in the paragraph).	<pre>[('cookie', 32.82515295692881), ('chocolate', 26.576515951269574), ('bake', 26.062151218168534), ('recipe', 24.375136780370635), ('make', 23.710536306874065), ('butter', 21.72938481986751), ('time', 21.077480667609745), ('add', 20.867892608755675), ('sugar', 20.718035564450286), ('baking', 19.698447185308577), ('flmy', 19.39686713644808), ('cook', 19.16348545607328), ('cake', 18.846577992992668), ('temperature', 17.942906578773965), ('bread', 17.781507590469435), ('texture', 17.76638226083076), ('give', 17.690253387056245), ('dough', 17.64317147067233), ('soft', 17.583630380930106), ('mix', 17.58025980554535)]</pre>

Bacon, oven, cook	Bacon, oven, cook, time	[('cooking', 40.12933597311298), ('time', 38.69417079086883), ('make', 37.85600728352511), ('pan', 35.359513553832876), ('bake', 34.09959705608397), ('meat', 33.4611175456202), ('good', 31.591361448351755), ('minute', 31.335652236656113), ('long', 31.15089171833207), ('put', 31.002237710315402), ('recipe', 30.948408662716087), ('heat', 30.829655010949725), ('temperature', 30.00208758626384), ('turn', 29.53337950556877), ('thing', 29.070785200269672), ('add', 28.70496599315215), ('work', 27.663726941690317), ('dish', 27.31617072730367), ('hour', 27.21676057410397), ('eat', 27.07264508932171)]
Mother, bread, culture	Bake, bread, yeast (not in the paragraph)	[('make', 25.47095406794962), ('bake', 24.992571092511625), ('yeast', 24.422003023523377), ('flour', 23.266094995529212), ('time', 22.950044862332707), ('add', 22.806543036574094), ('good', 22.189691703818635), ('start', 21.472871433624363), ('mix', 21.328554098504103), ('recipe', 20.88246716470687), ('rise', 20.857748162358178), ('sourdough', 20.75041902347124), ('starter', 20.46855578938263), ('dough', 20.383988549446222), ('loaf', 20.35161761314173), ('work', 20.14210821017458), ('question', 20.016495685927378), ('give', 19.81206249443809), ('butter', 19.572097003706926), ('cook', 19.506279576086563)]
Meat, rest, cold	Meat, temperature	[('cook', 1065.940343645877), ('put', 841.0429413657383), ('hour', 773.2172403484759), ('long', 667.6443849373065), ('add', 659.2773414090246), ('time', 658.0238109795231), ('fridge', 647.9300747230467), ('temperature', 624.4453333017682), ('water', 573.9342832353653), ('heat', 520.856810238986), ('make', 516.5555475590637), ('hot', 506.8121684315101), ('cooking', 464.8101276160044), ('chicken', 452.80417272731955), ('good', 452.29933664284897), ('minute', 440.6868892677706), ('work', 436.9843763852456), ('day', 428.51965665502706), ('start', 410.59460478534425), ('cut', 368.45186586238947)]

According to the result show above, we still possible to find tags by using Word2Vec, but we need to find a better score function and extract better keywords in former step so that we can find more alike keywords by Word2Vec.

V 、 Conclusion

In this task we can find tags from each paragraph with about 0.34 F1 score, which is 0.05 higher than our baseline (17% improvement). Although Word2Vec might not work very well as our expectation, it still can find some “hidden” tags or tags we didn’t extract from paragraph at first. So our method might work better after some modifications.

VI 、 Contribution

何文琦 : TF-IDF, Word2Vec

宋易霖 : RAKE, preprocessing.

陳威嘉 : Textrank.

VII 、 Reference

- [1] <https://radimrehurek.com/gensim/models/word2vec.html>
- [2] [TextRank: Bringing Order into Texts](#)
- [3] [Automatic Keyword Extraction from Individual Documents](#)