Automated Stack Overflow Question Tagger

Authors
Jaewoo Kim / 20170149
Taeyoon Kim / 20150860
Lee, Jun Hyeong / 20150608
Jun Seonyoung / 20140507

Abstract

We performed a preliminary experiment on our project of developing an automated Stack Overflow tagger. For every question, the significance of each word in the text were identified with the "tf-idf" algorithm. The significant words were then compared with the actual tags to check how they overlap. There was a high correlation between the words but they did not coincide as much as we hoped. We propose some additional tuning for finding significant words and a method to find significant sentences to improve the results.

1 Introduction

Stack-overflow is a popular community among developers for asking and answering questions about various fields of computer science. One feature of this community is tags that are attached to each post. These tags enable the classification of questions, thus allowing users to search questions that contain specific tags. However, users sometimes fail to attach appropriate tags to their questions, due to lack of experience in the community. We strive to develop an algorithm that analyzes and suggests appropriate tags for each question.

2 Problem Statement

Tags are useful for users who are not familiar with their own research area, and they find it difficult to identify appropriate tags for their own posts. Our goal is to develop an algorithm that analyzes the text of the question and suggest appropriate tags. We would also like to investigate which tags lead to most answers and which tags are commonly used together. We use the API of stack-overflow to collect data of existing posts. We process the text of each post to find suggested tags, and compare those with the actual tags for evaluation.

3 Technical Approach and Models

We will combine two kinds of models to solve our problem. First one is finding important words(words that contain lots of information about the paragraph) and the other one is finding the most important sentence.

A. Finding important words.

We will use a technique called tf-idf, which stands for term frequency and inverse document frequency. Here are the basic ideas of this approach. First, important words tend to appear relatively more than other words compared to words that aren't important. This is what term-frequency analyzes. Second, important words for that paragraph tend to appear only in that paragraph. In other words, if some word is 'unique' for that paragraph, such word is likely to serve an important word in that paragraph. Tf-idf model is a combination of these two. This approach can help us to find important words contained in the paragraph. Before analyzing words using tf-idf, we will remove stopwords, which means words that are contained in the sentence for correct grammar but have no useful meaning. Also, we want to perform stemming so identical words with different formats are not treated as two different words. If we require more accuracy, we are also considering pos-tagging and lemmatizing the words instead of stemming.

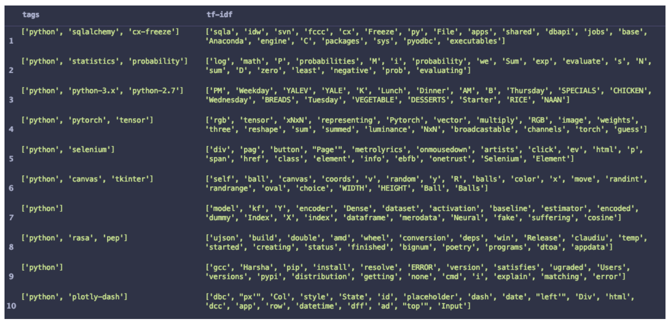
B. Finding important sentences

We will find important sentences out of the paragraph using the corpus of important words we've got in part A. We can analyze the importance of a sentence using two approaches. First, we can analyze the length of the sentence compared to other sentences in the paragraph. Longer sentences tend to contain more information compared to others. Second, we can see how many important words are contained in a sentence compared to other sentences. We're planning to use these two approaches to score the importance of each sentence. We have not yet set the exact parameters and methodology to combine these two, but we will tune them and find the optimal solution as we research on this topic.

Now, what we want to do with these models is to find ‘tags’ that represent the question. We can now set priorities of each word as a tag, by combining score by tf-idf, and score of the sentences that the word is contained in. The way we combine the score of two models will depend on the accuracy of each model. We expect two models will cover weaknesses of each other and find optimal tags out of the question.

4 Intermediate/Preliminary Experiments Result

We collected 7512 newest StackOverflow questions through crawling to use as the corpus. The fields collected for each of them are the id, title, link, content, tags, last activity date, and the number of answers. For the preliminary experiment we used the tf-idf method to calculate the importance of each word, in the hopes that it will coincide with the tags attached to each question. In the processing of the data, words were tokenized with non-alphabetic characters as delimiters. Thus, words like “python-3.x” or “cx-freeze” were not analyzed as one word. Also, stemming or lemmatizing was not performed to keep the experiment simple. This issue will be addressed later through the project. The image below shows the comparison between tags and words with top tf-idf scores for questions that include “python” as its tag. Tf-idf words are quite related to the real tags, but in many of the cases real tags do not end up as tf-idf words or do not even appear in the content of the question. Also, not many of the tf-idf words include the tag “python” which could be said as the most obvious one. It seems this is because people do not mention the word “python” often when asking a question that involves the python language. We are aiming to address this issue by analyzing the code part of the question separately to identify in which language it is written. We evaluated the results qualitatively since it is a preliminary experiment, but it was sufficient to identify the shortcomings of the approach and improve for the next steps.



	tags	tf-idf
1	['python', 'sqlalchemy', 'cx-freeze']	['sqla', 'idw', 'svn', 'fccc', 'cx', 'Freeze', 'py', 'File', 'apps', 'shared', 'dbapi', 'jobs', 'base', 'Anaconda', 'engine', 'C', 'packages', 'sys', 'pyodbc', 'executables']
2	['python', 'statistics', 'probability']	['log', 'math', 'P', 'probabilities', 'M', 'i', 'probability', 'we', 'Sum', 'exp', 'evaluate', 's', 'N', 'sum', 'D', 'zero', 'least', 'negative', 'prob', 'evaluating']
3	['python', 'python-3.x', 'python-2.7']	['PM', 'Weekday', 'YALEV', 'YALE', 'K', 'Lunch', 'Dinner', 'AM', 'B', 'Thursday', 'SPECIALS', 'CHICKEN', 'Wednesday', 'BREADS', 'Tuesday', 'VEGETABLE', 'DESSERTS', 'Starter', 'RICE', 'NAAN']
4	['python', 'pytorch', 'tensor']	['rgb', 'tensor', 'xNxN', 'representing', 'Pytorch', 'vector', 'multiply', 'RGB', 'image', 'weights', 'three', 'reshape', 'sum', 'summed', 'luminance', 'NxN', 'broadcastable', 'channels', 'torch', 'guess']
5	['python', 'selenium']	['div', 'pag', 'button', "Page'", 'metrolyrics', 'onmousedown', 'artists', 'click', 'ev', 'html', 'p', 'span', 'href', 'class', 'element', 'info', 'ebfb', 'onetrust', 'Selenium', 'Element']
6	['python', 'canvas', 'tkinter']	['self', 'ball', 'canvas', 'coords', 'v', 'random', 'y', 'R', 'balls', 'color', 'x', 'move', 'randint', 'randrange', 'oval', 'choice', 'WIDTH', 'HEIGHT', 'Ball', 'Balls']
7	['python']	['model', 'kf', 'Y', 'encoder', 'Dense', 'dataset', 'activation', 'baseline', 'estimator', 'encoded', 'dummy', 'Index', 'X', 'index', 'dataframe', 'merodata', 'Neural', 'fake', 'suffering', 'cosine']
8	['python', 'rasa', 'pep']	['ujson', 'build', 'double', 'amd', 'wheel', 'conversion', 'deps', 'win', 'Release', 'claudiu', 'temp', 'started', 'creating', 'status', 'finished', 'bignum', 'poetry', 'programs', 'dtoa', 'appdata']
9	['python']	['gcc', 'Harsha', 'pip', 'install', 'resolve', 'ERROR', 'version', 'satisfies', 'ugraded', 'Users', 'versions', 'pypi', 'distribution', 'getting', 'none', 'cmd', 'i', 'explain', 'matching', 'error']
10	['python', 'plotly-dash']	['dbc', "px'", 'Col', 'style', 'State', 'id', 'placeholder', 'dash', 'date', "left'", 'Div', 'html', 'dcc', 'app', 'row', 'datetime', 'dff', 'ad', "top'", 'Input']

```
def tf(bag):
    bag = list(filter(lambda w: re.match('[a-zA-Z]+', w), bag))
    words = dict.fromkeys(set(bag), 0)
    for word in bag:
        words[word] += 1
    bag_size = len(bag)
    return dict(map(lambda w: (w[0], w[1] / bag_size), words.items()))


def idf(bag):
    bag = list(filter(lambda w: re.match('[a-zA-Z]+', w), bag))
    document_count = len(data)
    return dict(map(lambda w: (w, math.log(document_count / len(data[data.apply(lambda x: w in x['content'], axis=1)]))), bag))


def tf_idf(tf, idf, scores):
    words = sorted(list(map(lambda kv: (kv[0], kv[1] * idf[kv[0]]), tf.items())), key=itemgetter(1), reverse=True)
    if scores:
        return words
    else:
        return list(map(lambda w: w[0], words[:20]))
```