Predicting Stack Overflow Question Tags

Jan Viilma

School of Information Technologies
Tallinn University of Technology, Estonia
janviilma@hotmail.com

Abstract

The aim of this project was to predict the key words of a text in a larger text corpus. Specifically the Stack Overflow user questions dataset [1]. These key words are have been assigned by human users. Different options and their viabilities for solving this problem are discussed and compared. Mostly unsupervised models, such as latent Dirichlet allocation [2], are going to be covered in the experiments section below. It will be interesting to see how well these mathematical models can predict the key words that people have assigned. Finally, one of the unsupervised methods is going to be implemented in a supervised learning model.

1. Introduction

In machine learning and natural language processing (NLP) this kind of keyword prediction problem is usually referred to as topic modeling, which is an umbrella term for a variety of text mining and pattern recognition methodologies [3]. Text mining or data mining is essentially the construction of a statistical model that best describes this data [4]. Topic modeling can either use unsupervised probabilistic algorithms or supervised learning algorithms. One of the most popular unsupervised approaches is latent Dirichlet allocation. LDA also has a further developed analog, sLDA [5], which uses a supervised training approach.

2. Background

Topic modeling could be described as working through an article with a set of different colored highlighters. As you come upon a key word you highlight that word with one of the colors in such a manner that the key words in a similar theme have the same color. In practice this highlighting is determined by mathematics and a lot of different algorithms exist for that purpose.

2.1. Term frequency-inverse document frequency

Term Frequency times Inverse Document Frequency (TFIDF) is a measurement of how concentrated in a relatively few texts are the occurrences of a given word. The idea here is that if a text in a larger text corpus is about a certain topic, then there are a few infrequent words related to this discrete topic a lot more densely packed together than in other parts of the corpus [4]. It is usually calculated as follows,

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{1}$$

where f_{ij} is the frequency of a term (word) i in text j and f_{kj} is the most often occurring term in the same text, meaning that $TF_{kj} = 1$. From this follows,

$$IDF_i = \log_2(N/n_i) \tag{2}$$

where N is the number of texts in the corpus and n_i is the number of texts that contain the term i. The score of a term i in text j is then $TF_{ij} \times IDF_i$. The terms with the highest scores in a text are now most likely going to represent this text. With this formula both the occurrences of a term in the whole corpus and every text individually are taken into account. This results in a term-by-text matrix X, where every element is the score of a single term in a single text, thus converting these texts into fixed-length lists.

2.2. Latent semantic analysis

In *latent semantic analysis* (LSA) or *latent semantic indexing* (LSI), a large matrix of term-text association data is used to create a "semantic" space wherein terms and texts that are more closely related are more close together [6]. The main problems that this method tackles are *synonymy* and *polysemy*. *Synonymy* means that multiple words can describe the same thing and *polysemy* refers to the fact that a single word can have multiple meanings. Mathematically this is done by for example using the TDIDF term-by-text matrix and applying singular value decomposition on it, which allows us to build a new matrix where terms and texts that are more similar are closer together in space (vectors that represent them have a smaller cosine).

The problem with this approach is that the algorithm doesn't differentiate well between terms and texts where the term could have multiple meanings, resulting in some texts being close to a set of terms that don't really describe those texts.

2.3. Latent Dirichlet allocation

Latent Dirichlet allocation is an unsupervised probabilistic topic model. The intuition of this model is that first we assume some number of topics for our text corpus. These topics are the distributions over all of the words. First we choose the distribution of chosen topics. Next we iterate over every word and see which topic it represents [7].

2.4. Hierarchical Dirichlet process

Another powerful topic model is *hierarchical Dirichlet process* (HDP) can be used [8]. It is a nonparametric mixed-membership model for unsupervised analysis of grouped data. This means that unlike LDA or LSA, HDP can infer the number of topics from the data.

3. Approach

The approaches that are used in this project will be considering the properties and idiosyncrasies of the Stack Overflow dataset. Meaning that when a different type of dataset is used, another approach could give more precise results. The size of the used Stack Overflows tags dataset is $62.4\,\mathrm{MB}$ and consists of $3\,750\,994$ entries with two values, the corresponding question id and value of the tag (Table 1),

Table 1: *Stack Overflows dataset of tags.*

Id	Tag
80	flex
80	actionscript-3
80	air
90	svn
...	...

When counting only the unique values, this count drops to $37\,034$. As we can see, this is a lot of keywords and a simple classification model training won't be enough. This is why we are going to use some of the mentioned unsupervised topic models and compare their results. This means that the tags are only going to be used in evaluation after training the model.

3.1. Preparing the data

When examining a few questions in the Stack Overflow dataset (Table 2) and comparing them to their corresponding tags, it can be seen that the "*Body*" or "*Title*" columns individually will not have enough information for creating an accurate model. All of the assigned tags will rarely be in a single column. In addition, all of the *topic models* that were discussed and even most that weren't mentioned above depend heavily on the density of rare words. This is why the input corpus is generated by merging the "*Title*" and "*Body*" columns.

Table 2: *Stack Overflows dataset of questions.*

Id	OwnerUserId	CreationDate	ClosedDate	Score	Title	Body
90	26	2008-08-01T13:57:07Z	NaN	26	*body of text*	*longer body of text*
...	...	...	...	...	...	...

Before starting any topic modeling the text corpus has to go through certain processes (assuming that the text corpus is a list of texts):

1. All of the texts have to be tokenized into lists of words.
2. It is recommended to filter out more often recurring "*stop words*" (*e.g.* "a", "the", "and") because they don't carry any real meaning about the topic. Other kinds of filtering methods could be used here. The end target here is to have as little unnecessary data as possible, considering the *garbage in, garbage* out principle [9].
3. A dictionary of the tokenized corpus has to be created where every word comes up only once and has a discrete index value.
4. The text that is used for model creation is now converted to *bag of words* representation, which means that every word is now represented by a vector of its index and number of occurrences.

Now the data is ready for further model training.

3.2. Model creation

The questions dataset size is $1.79\,\mathrm{GB}$ and it consists of 69 entries. It has some unnecessary columns in it, which we are going to ignore. The most important columns for us are the "*Title*" and the "*Body*". After throwing out all the unnecessary columns, the dataset is still quite large. This means that we have to be extra careful before doing any model training, because the training times are going to be long and take up a lot of resources.

4. Experiments

All of the code was written in Python with *Jupyter Notebook*. They are made to be easily comprehensible and re-usable by making use of IPythons widget API. The application is divided into four main parts – data analysis, data preparation, model creation and model analysis. The main library for data loading and structure handling is done with *pandas* library. The most important library that all of the model building and data preparation is done with is *Gensim* [10]. Gensim is a Python library that has a wide variety of powerful tools for *topic modeling* and *document indexing* that can handle huge bodies of texts. It also has the tools for preparing your datasets for different deep learning algorithms if necessary. It has good community support and comprehensive well maintained documentation. *Gensim* also has wrappers for more popular external libraries, *e.g. keras*, *scikit-learn*.

4.1. Data preprocessing

Data preparation is partly done with *Gensim*'s built-in preprocessing methods. This will filter out all of the HTML tags, *stop words*, punctuations and redundant white-spaces. Next it will turn the text into lowercase, stem it and finally tokenize it into list of words. After that a custom filter is used to filter out words that are shorter than three characters, except a few that are sensitive for correct keyword detection – "c", "c#", "r", "3d", "2d", "1d", "7z", "qt".

The next step is to create a dictionary from the tokenized text corpus. This is going to map all of the words with a letter. In the hopes of getting more interesting results, two dictionaries are going to be created. One of the will use the tokenized questions text corpus but the other will use only the tags dataset. It will be interesting to see how the results will differ when creating the *bag of words* data representations from both of these dictionaries separately. This will also mean that the tokenization has to be done twice because word stemming and removal of short words could actually hinder the results when using the tags dictionary.

For supervised learning, the *scikit-learn* [11] library was used. *Scikit-learns* training input requires a little different data form. The input corpus is going to be a list of texts, not a list of tokenized text. To not lose any information, the words won't be stemmed. This time also the labels are going to be necessary. They are going to be converted with the multi-label binarizer which transforms between a list of lists and a binarized format. For example $[(1, 2), (3,)] < - > [[1, 1, 0], [0, 0, 1]]$.

4.2. Model creation

The simplest model that we will be creating is the TFIDF model. Its only required input is the text corpus in the *bag of words* form. We will also add the optional dictionary that was used when creating *bag of words* to make use of other helpful model functionalities.

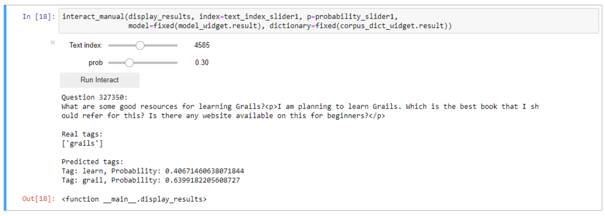
Next on our list will be the LDA model. We will make use of the multi-core model option to use all of the CPU's cores. Besides the corpus *bag of words*, this model also needs to know the number of topics before hand. This doesn't mean that we have to give it the number of different unique tags (37 034 topics would need an absurd amount of RAM for training). These are going to be the topics that are the distributions over all of the words. When detecting the key word of an observed text, it will still assign it a word that best describes it according to the mathematical model. Our model is going to try to divide all of the words into 50 topics. This is still quite a large number, considering the size of the dataset and will take some time to calculate. The model size is going to be "8 B×*num_terms*×*num_topics*".

The last unsupervised model is going to use the HDP algorithm. Just like the TFIDF model, its only required input is the *bag of words* text corpus. The down side of this model is that the training time will be a lot longer than it is for the other models. It will be interesting to see how well it will do and how many different topics it infers from the corpus.

For supervised learning the model pipeline starts with a TFIDF vectorizer, which was covered in section 2.1. After that its making use of a regular stochastic gradient descent (SGD) classifier and finally the *One-vs-the-rest* (OvR) multiclass/multilabel strategy. OvR fits a classifier for every class where every class is fitted against every other class.

4.3. TFIDF model

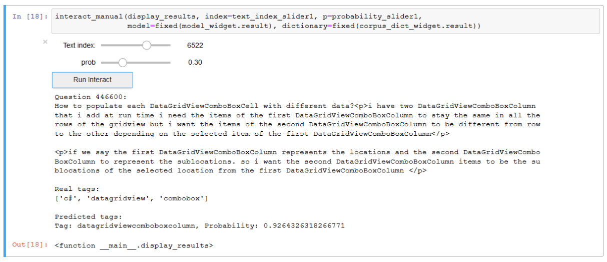
Lets take a look at an example of assigning the tags for a random question (Fig. 1),



```
In [18]: interact_manual(display_results, index=text_index_slider1, p=probability_slider1,
                    model=fixed(model_widget.result), dictionary=fixed(corpus_dict_widget.result))

         Text index: 4585
         prob        0.30
         Run Interact

         Question 327350:
         What are some good resources for learning Grails?<p>I am planning to learn Grails. Which is the best book that I sh
         ould refer for this? Is there any website available on this for beginners?</p>

         Real tags:
         ['grails']

         Predicted tags:
         Tag: learn, Probability: 0.40671460638071844
         Tag: grail, Probability: 0.6399182205608727

Out[18]: <function __main__.display_results>
```

Figure 1: *Example of analyzing a single question with TFIDF model.*

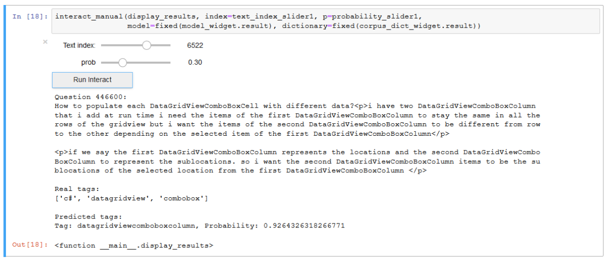
The probability here actually just refers to the TFIDF score (covered in section 2.1) of this the tag in the context of the analyzed text. From the slider we can choose a random question and a filtration value for the key word score. Although the last example was a pretty good guess, most aren't that good. For example on Fig. 2 we can see that the guess is actually very good but when doing a validation run, it will be cleared as completely wrong.



```
In [18]: interact_manual(display_results, index=text_index_slider1, p=probability_slider1,
                    model=fixed(model_widget.result), dictionary=fixed(corpus_dict_widget.result))

         Text index: 6522
         prob        0.30
         Run Interact

         Question 446600:
         How to populate each DataGridViewComboBoxCell with different data?<p>i have two DataGridViewComboBoxColumn
         that i add at run time i need the items of the first DataGridViewComboBoxColumn to stay the same in all the
         rows of the gridview but i want the items of the second DataGridViewComboBoxColumn to be different from row
         to the other depending on the selected item of the first DataGridViewComboBoxColumn</p>

         <p>if we say the first DataGridViewComboBoxColumn represents the locations and the second DataGridViewCombo
         BoxColumn to represent the sublocations. so i want the second DataGridViewComboBoxColumn items to be the su
         blocations of the selected location from the first DataGridViewComboBoxColumn </p>

         Real tags:
         ['c#', 'datagridview', 'combobox']

         Predicted tags:
         Tag: datagridviewcomboboxcolumn, Probability: 0.9264326318266771

Out[18]: <function __main__.display_results>
```

Figure 2: *Another example of analyzing a single question with TFIDF model, that should be partly correct.*

A guess of how well the model does was calculated by checking a large number of question and seeing how many had tags were observed correctly. This is displayed on Fig. 3. We can see that for 6522 questions with a filtration score of 0.30, 4307 tags were assigned correctly out of 20 442. This is only 21 %.



```
In [18]: interact_manual(display_results, index=text_index_slider1, p=probability_slider1,
                    model=fixed(model_widget.result), dictionary=fixed(corpus_dict_widget.result))

         Text index: 6522
         prob        0.30
         Run Interact

         Question 446600:
         How to populate each DataGridViewComboBoxCell with different data?<p>i have two DataGridViewComboBoxColumn
         that i add at run time i need the items of the first DataGridViewComboBoxColumn to stay the same in all the
         rows of the gridview but i want the items of the second DataGridViewComboBoxColumn to be different from row
         to the other depending on the selected item of the first DataGridViewComboBoxColumn</p>

         <p>if we say the first DataGridViewComboBoxColumn represents the locations and the second DataGridViewCombo
         BoxColumn to represent the sublocations. so i want the second DataGridViewComboBoxColumn items to be the su
         blocations of the selected location from the first DataGridViewComboBoxColumn </p>

         Real tags:
         ['c#', 'datagridview', 'combobox']

         Predicted tags:
         Tag: datagridviewcomboboxcolumn, Probability: 0.9264326318266771

Out[18]: <function __main__.display_results>
```

Figure 3: *Example of analyzing a single question with TFIDF model.*

When altering the filtering score, 0.3 seemed to give the best results. Raising the number of questions will only make it load for a longer time but won't really have much of an effect on the results.

By using the model that was trained by the tags dictionary, a slight increase in the predictions was seen. With 5269 questions, 4280 tags were predicted correctly out of 18 248 – that is an accuracy of 23 %. Another increase was seen when using the model only on the titles of the questions. This way, with the model based on tags, the prediction accuracy rose to 29 %. With the questions model, it even reached an accuracy of 33 %. These results are shown in the Table 3 below.

Table 3: *Results of the TFIDF model with different setups.*

Model dictionary	Target	Accuracy
Questions	Title & body	21 %
Questions	Title	33 %
Tags	Title	29 %
Tags	Title & body	23 %

4.4. Other unsupervised models

In addition to the TFIDF model, models were made with the LSI, LDA and HDP algorithms in various combinations. All of these, compared to TFIDF, took hours to train, were extremely large in size and gave very poor results. None of the models even reached a 1 % accuracy.

4.5. Supervised model

Because the dataset consists of a large number of elements and there is no way to use better resources for model creation, only 140 000 questions are used for training and 70 000 for evaluation. With this in mind the supervised model performed slightly better than unsupervised models. The results were evaluated with F-score, achieving the accuracy of 0.36, which is pretty close to one of the TFIDF model accuracies. However the evaluation of TFIDF was not as in-depth, so comparing them is a bit unfair. Also when considering that this result is achieved by using the model on the whole question we should actually compare it with the lower results that we got with the TFIDF model. Because these models take a long time to train we are going to accept these results. To gain a further increase in accuracy the questions titles could be used with a higher importance.

5. Conclusion

This kind of unsupervised topic modeling can be very good in cases where there is no classifying data for the model training. As we can see from the results these mathematical models do not "think" like human users. Even the TFIDF, which is ironically the simplest one out of all the models, doesn't predict the tags with a considerable accuracy. The supervised models accuracy can be increased with further tweaking of input data or hyper-parameters. An increase could also be gained when using the whole dataset. In conclusion, when considering the size of the dataset and the number of labels used, these model definitely predict something and don't just make completely random guesses.

6. References

[1] Stack Overflow, "Stacksample: 10% of Stack Overflow Q&A," `https://www.kaggle.com/stackoverflow/stacksample`, 2016, [Online; accessed 04-June-2018].

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, Jan. 2003.

[3] Megan R. Brett, "Topic modeling: A basic introduction," *Journal of Digital Humanities*, vol. 2, no. 1, 2012.

[4] Anand Rajaraman and Jeffrey David Ullman, "Data mining," in *Mining of Massive Datasets*, pp. 1–17. Cambridge University Press, 2011.

[5] David M. Blei and Jon D. McAuliffe, "Supervised topic models," 2010.

[6] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, sep 1990.

[7] David M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, pp. 77, apr 2012.

[8] Chong Wang, John Paisley, and David Blei, "Online variational inference for the hierarchical dirichlet process," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Geoffrey Gordon, David Dunson, and Miroslav Dudk, Eds., Fort Lauderdale, FL, USA, 11–13 Apr 2011, vol. 15 of *Proceedings of Machine Learning Research*, pp. 752–760, PMLR.

[9] Revi Parikh, "Garbage in, garbage out: How anomalies can wreck your data," 2014.

[10] Radim Řehůřek and Petr Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010, pp. 45–50, ELRA, `http://is.muni.cz/publication/884893/en`.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.