Learning to Label Stack Exchange Questions

Rafael Leano Jose Picado Zahra Iman
Oregon State University
Corvallis, OR 97333

Abstract

Many websites now allow users to create content for others to see. This results in content being added at a high rate making it difficult to navigate. To address this, websites have opted to allow tagging mechanisms to capture relevant topics or keywords of the content and attaching it. However, this brings new problems as users have to deal with the tagging, which becomes more difficult the more tags there are. Hence, research has focused on developing approaches that try to predict the tags that a given content may have. In particular, we use data from a Kaggle competition from StackOverflow to predict the tags of questions. We treat the data to improve our results (e.g. no stopwords, no HTML tags, use a specific wordvec set); and we use 2-layer and 3-layer LSTM architectures and evaluate our results with further restrictions on the input data like titles only or cropping questions at a given word-size. We are able to achieve a subset accuracy of 0.6557 that is significantly better than a baseline uni-gram and bi-gram approach that attains a 0.518 accuracy.

1. Introduction

In today's information era, much of the content is created and maintained collaboratively by the users. For example, websites like Wikipedia [1], DevianArt [2], Twitter [3] or even GitHub [4] simply provide a platform for a specific type of content (articles, images, comments, source code) and it is the users who share their information for others to see.

However, the vast amount of content shared soon becomes difficult to navigate. Users need to be able to sort, categorize, filter, and/or search this content to find what they are looking for. To address this issue, many information hubs have used *tagging*. In tagging, content is enriched with *tags* that represent the most relevant topics or keywords of that content.

[1]http://www.wikipedia.org
[2]http://www.deviantart.com
[3]http://www.twitter.com
[4]http://www.github.com

Nonetheless, *tagging* brings new problems to consider. Tying users down to a set of limited tags can be too coarse-grained and restrictive to users, limiting the usefulness of tagging. On the other hand, allowing users to create tags makes the number of tags to grow to levels where it is difficult for users to know which tags to use (e.g. maybe there is a more specific tag for my content but I am unaware of it), or to create synonym tags; that is, tags that refer to the same concept but use different words, or different spellings or abbreviations (e.g. *db* vs. *database*).

In particular, StackOverflow is a Q&A website where users can post questions and get answers from the community. Questions are tagged with as many concepts as desired to help users find the questions and provide answers. The user who posted a question can then come back and choose an answer as the *preferred* answer. Doing so marks the question as closed and gives *reputation* point to the user to gave the answer. StackOverflow uses *reputation* to handle the problem of restricting users to a limited set of tags or having an uncontrollable set of tags. Only users with a given reputation may suggest new tags, or may suggest a *synonym* relationship between two existing tags. Even so, the number of tags in the size currently sits around 44,000 tags [5].

In conclusion, while tags are a good way of classifying content and make it easier to navigate, they become another responsibility for the content creator (i.e. the user). Saha et al. [2] showed that correctly tagging content makes the content to be seen more, and in the case of StackOverflow, that also increases the chances of a question having more answers and having a *preferred* answer. Hence, being able to automatically tag questions has positive effects on the site, its content, and its users, by: (1) suggesting tags to users for their questions and facilitating their work, (2) preventing the discriminate creation of synomyn tags (e.g. *mysql*, *my-sql*, *MySQL*, *mysql-db*, etc.), and (3) warning a user of a potentially incorrect label they are using.

In this report, we evaluate a deep learning approach that uses *Long Short-Term Memory* (LSTM) Networks to predict the labels of a question in StackOverflow given the title

[5]http://data.stackexchange.com/stackoverflow/query/new

and content of the question only. We make use of *wordvectors* to represent each word and baseline our results with a *uni-gram* and *uni-gram and bi-gram* model approach with logistic regression. Our results show that our LSTM architecture is able to obtain a *subset accuracy* of 0.6567 with our data. In comparison, the uni-gram and bi-gram model's accuracy peaks at 0.518. Also, between the different configuration of LSTM architecture and input data we determined that more samples allow better results as opposed to more features, a deeper architecture, or limiting to just the title of the questions.

2. Architecture

For this study, we used the architecture like the one depicted in Fig. 1. Vertically (i.e., each row) represents a layer of LSTM and horizontally (i.e., each column) the time sequence. That is, at each time $t_1, t_2, ...t_n$, a *word* is given as input to our architecture, which uses the layer LSTM, to learn and predict the possibility of all the potential *tags* to belong to the question or not. After the LSTM layers, we have a fully-connected layer that averages the results from all the time sequences ($t_1 - t_n$) and uses a *softmax* activation function to output the possibility ($[0, 1]$) of how likely is each tag to belong to the given question.

In our case, LSTM are useful since they keep in mind the order of the words. We posit that this is important since questions in StackOverflow initially begin by giving a context. This context is likely to contain important words that allow us to define good tags for the question. For example, for a question that starts by saying *"I am using Eclipse Indigo for Android development. The problem I face is that it..."* has the tags $< eclipse, android >$. Specifically, we used the *vanilla* variant of LSTM [1]. the ellipses (...) in Fig. 1 denote that we have as many columns (time sequences) as required by the questions. In the next section we will describe the data we used and how it was treated prior to feeding it to our architecture.

Finally, our architecture used *categorical cross entropy* as the *LOSS* function to allow multiple tags per question. We optimized the model using *RMSprop*.

3. Data

We employed the Stack Exchange dataset provided in the "Facebook Recruiting III - Keyword Extraction" Kaggle competition[6]. The dataset originally contains the text and title for 6,034,195 questions, as well as the tags assigned to each question (Table 1) Question are extracted from Stack Exchange sites and contain a mix of both technical and non-technical questions. In the original dataset, there are 42,048 tags. Each question is assigned between 1 and 5 tags, and the average number of tags per question is 2.89.

[6]https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction

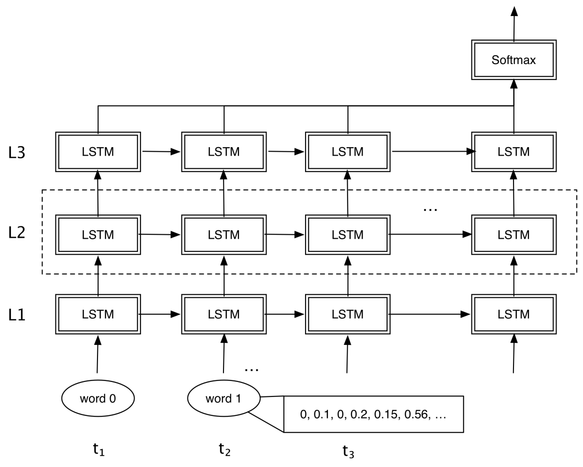


Figure 1. our LSTM architecture

As this is a very big dataset, due to computational constraints and time, in this study we reduced the number of tags and number of questions considered. First, we considered only the top 10 tags, as well as the top 100 co-occurring tags with each top tag. This resulted in reducing the number of tags to 573. Likewise, we considered only questions that contained tags in the set of chosen tags, which were 1,220,004 questions. We called this dataset *Filtered-1* (Table 1)

Additionally, we needed a way to represent the words in our LSTM, one way of doing this was to have perpendicular vectors so that each word would be independent of one another. Nonetheless, research has shown that it is good to allow related words to not be perpendicular with the idea of *word vectors* that allow words to be added, subtracted, and more to leverage this relations (e.g. the word $W(Queen)$ can be produced from $W(King) + W(Woman) - W(Man)$). We decided to take advantage of this relations in our predictions.

Table 1. Different datasets used

Dataset	Questions	Tags	Max Length
Original	6,034,195	42,048	2,694
Filtered-1	1,220,004	573	2,694
Filtered-2	200,000	72	400
Filtered-3	100,000	72	800

3.1. Word Vectors

We found different sets of word vectors (wordvecs) available to use[7]. However, to be able to have a vector for a particular word it was required that the word to be contained in the dictionary used for the word vectors. Because we were looking at questions from StackOverflow, we expected that many important terms were not everyday words that could be found in a regular dictionary or by analyzing regular news. As expected, the first dictionaries that used Google News and Freebase lacked many programming-specific words like particular APIs, technology names, and so on. Thankfully, the models trained with Wikipedia did have these terms and we opted for the smallest version of it. That is, the *Wikipedia+Gigaword 5* model with 50 dimensions (numbers) per word.

3.2. Data Treatment

The questions in the original dataset did not have any pre-treatment at all. That is, these questions contained everything "as is" from the dataexchange dump. Questions contained all the HTML tags and newlines used to decorate an organize the text. Though we initially though that HTML tags would be useful to weight words in the question (e.g., bolded words or code could contain more meaningful words). However, we soon realized that these tags were not present on any on the wordvecs available. Hence, before encoding all questions to word vectors, we cleaned the HTML tags from the questions since tags are not really words and then they would not be found on the wordvec model. For example $< p >$ or $< br >$ would not add anything new to the prediction and, instead.

Similarly, another common technique used in Natural Language Processing is *stopword* removal. Questions were stripped of common words like 'a', 'then', 'when', etc. that are deemed as *stopwords* that are present in sentences as grammatical aides but do not convey any significant idea. Removing these words is useful to simplify the model by reducing the length of the questions.After removing stopwords, HTML tags, punctuation and numbers from titles and body of questions. The maximum length of a question was 2,694.

Finally, because to input questions to the LSTM we had to make all the questions to have the same lenght, we use *left padding* to left fill with empty words questions until they all reached the maximum size. Hence making the input data to be a 3-D matrix with the three dimensions being: (1) the number of questions, (2) the length of each word vector (50 in the selected wordvec), and (3) the number of words per question.

[7]https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models

Table 2. Top 10 tags and some top co-occurring tags.

Top 10 tags	Co-occuring tags	
c#	wpf	winforms
java	eclipse	spring
php	html	json
javascript	html5	ajax
android	android-layout	activity
jquery	jsp	css
c++	c	string
python	numpy	python-3.x
iphone	objective-c	ios5
asp.net	asp.net-mvc	wcf

4. Experiments

In this section we present the results of question labeling using the model described in Section 2 and the data from the previous section (Section 3). We also conducted experiments with traditional methods in order to compare (baseline) the performance of deep models with the performance of these methods.

Because the number of questions and the maximum length of each question were very big, we first tried to process the questions in batches of 10,000 questions. However, we did not see any improvement in the training of our models after each batch (accuracy would start from almost 0 for each batch). Therefore, we decided to reduce data even more so that we could fit all the data in main memory and eliminate the batches.

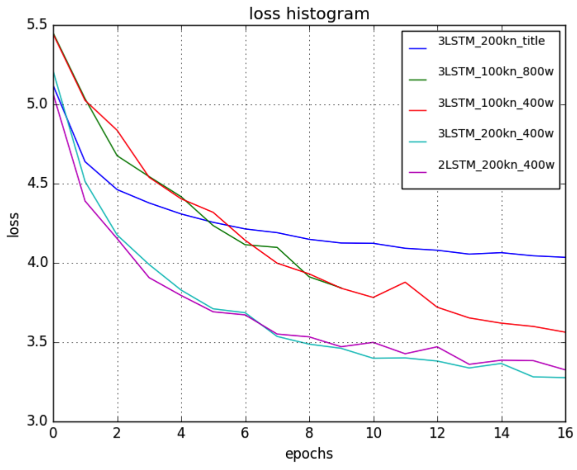
Our final approach considered only the top 10 tags, as well as the top 10 co-occurring tags with each top tag. This resulted in 72 tags. The top 10 tags along with some co-occurring tags are shown in Table 2. This also reduced even more the number of questions available (only those with tags from the new subset of 72). However, we were still experiencing memory errors due to the size. Finally, we were able to reduce our data even more by randomly selecting a number of questions from the remaining data, and limiting the words considered for each question. This resulted in two different datasets: *Filtered-2* had more questions, but with a 400-word limit; *Filtered-3* had half the questions, but considered twice the words in each question (See Table 1). The length of the question included the question's title and description.

4.1. Baseline

We tested two traditional methods:

1. Multi-label classification with logistic regression on uni-grams

2. Multi-label classification with logistic regression on uni-grams and bi-grams



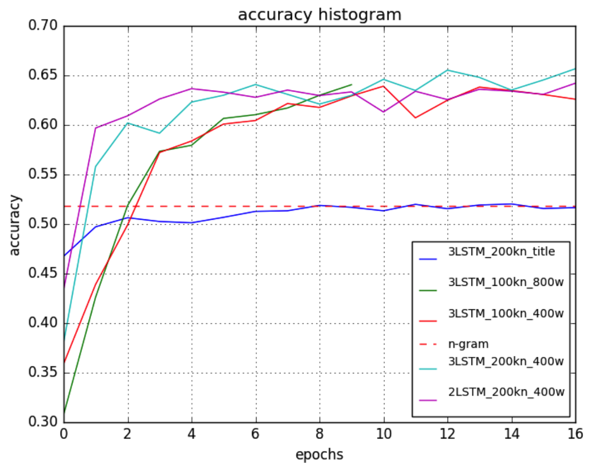


Figure 2. Loss histogram (left) and accuracy histogram (right) for various models. Legend has format ⟨architecture⟩ _ ⟨number of questions⟩ _ ⟨maximum question length⟩.

This multi-label classification method is also called *binary relevance methods* [3], in which one binary classifier is independently trained for each label. For a test question, the combined model predicts all tags for which the respective classifiers predict a positive result. The base estimator used for this purpose is *Logistic Regression* in order to correspond to our LSTM approach as defined in Section 2.

Uni-grams and bi-grams were extracted from the title and body of the question. The text of title and body are cleaned and limited with the same process defined in Section 3 to ensure comparability to the *LSTM* model. This results is $1,171,964$ unigrams, and $5,551,226$ uni-grams and bi-grams in total. The multi-label classification with logistic regression has been implemented using *OneVsRestClassifier* method from Scikit-Learn toolbox[8].

4.2. Experimental results

As mentioned before, the *Filtered-2* dataset consisted of 200k questions, and each question was limited to have a length at most 400. We divided the dataset into training and testing sets by keeping 90% of the questions for training and the remaining 10% for testing.

We implemented the deep models using the Keras[9] toolbox. We used a batch size of 128, and trained for 16 epochs. We used *rmsprop* as the optimizer. The variant of LSTM we used is the common “vanilla” architecture [1]. As the number of classes that our model must predict is greater than 2, we used categorical loss entropy as our loss function. We also report the *subset accuracy* to compare our models. In this metric, a *true positive* occurs only when the set of predicted labels *exactly* matches the true set of labels. Results are presented in Table 3.

Table 3. Experimental results.

Method	Categorical loss entropy	Subset accuracy
1-gram + Logistic Regression	16.66	0.487
{1,2}-gram + Logistic Regression	16.09	0.518
2-layer LSTM (title+body)	3.33	0.642
3-layer LSTM (title+body)	**3.27**	**0.657**
3-layer LSTM (title only)	4.01	0.524

As expected, the deep models outperformed traditional models. The 3-layer LSTM that took as input both title and body of questions performed best. The 2-layer LSTM with the same input was a close second. This shows that using deep models certainly results in better performance. However, for this problem, it is not necessary to have very deep models to achieve good performance. Interestingly, the 3-layer LSTM that took as input titles only also outperformed traditional methods. This shows the power of deep models, which although taking a much more restricted input, are able to compute way better features than n-grams. Nevertheless, deep models require long training times and specialized hardware, i.e. GPUs. Each epoch may take between 1 to 12 hours, and according to our experiments, at least 10 epochs are required to see some convergence. On the other hand, training traditional models took at most 2 and a half hours, running in a personal computer without a GPU.

We performed further experiments to find out what is more useful: longer questions, more questions, or deeper models. This was done using the *Filtered-2* and *Filtered-3* datasets. We also optionally could use only the words from

[8]http://scikit-learn.org/
[9]http://keras.io/

the title or all the words (title + description) We compared the following models:

- 3-layer LSTM with 200,000 questions, titles only (Filtered-2 dataset, 400-word limit)
- 3-layer LSTM with 100,000 questions, title+body (Filtered-3 dataset, 800-word limit)
- 3-layer LSTM with 100,000 questions, title+body (Filtered-3 dataset, **400**-word limit)
- 3-layer LSTM with 200,000 questions, title+body (Filtered-2 dataset, 400-word limit)
- 2-layer LSTM with 200,000 questions, title+body (Filtered-2 dataset, 400-word limit)
- N-gram + Logistic Regression (Filtered-2 dataset, 400-word limit)

The categorical loss entropy and validation accuracy histograms are presented in Fig. 2 As can be seen, the models that performed best were the ones that took more questions, even though the maximum length was shorter. These models, *3LSTM_200kn_400w* and *2LSTM_200kn_400w*, learned faster and achieved higher accuracy and lower loss than other models. Interestingly, model *2LSTM_200kn_400w* learned faster than model *3LSTM_200kn_400w* in the first epochs. However, the latter caught up and even outperformed the former. As can be seen by models *3LSTM_100kn_800w* and *3LSTM_100kn_400w*, there was not much difference in restricting the maximum length of the questions. Both models learned in a very similar fashion and achieved very similar results. This is probably because of two reasons: (1) the information to determine the tags of a question usually comes in the title and first sentences of the question, (2) not many questions are longer than 400 words.

Furthermore, we did not find evidence of overfitting in any of our configurations. Fig. 3 to 6 show the individual accuracy for each configuration (comparing training set, validation set, and baseline). In every case, the accuracy of the validation set was similar to the testing or above, not significantly below.

Finally, it is interesting to see that the model that took as input only titles, *3LSTM_200kn_title*, started with a better accuracy than any other model, and better loss than the models that took 100,000 questions as input. However, it was quickly outperformed by other deep models, and was only able to achieve a performance comparable to the baseline. From these observations we can conclude that, for this problem, the order of importance is: (1) more questions, (2) deeper models, (3) length of questions.

5. Future Work

There is still much to be done to enhance our model. For starters, we only experimented with 2-layer LSTM and 3-layer LSTM. Though we are already doing better than traditional, *n-gram* approaches, we have yet to experiment with more hyper-parameters such as the *batch size*, the number of *inner nodes* inside the LSTM, etc.

Additionally, to be able to compare our approach with others in the Kaggle competition or with Saha et al. [2]; we need to be able to use get closer to use the original dataset. At least, to consider all the questions and tags to see how our approach compares to theirs.

6. Conclusion

We developed deep learning models to predict tags of Stack Exchange questions, given the title and body of questions. Results showed that deep models can reasonably predict *exactly* which tags should be assigned to each question, as opposed to traditional models. Results also showed that, as it is common the case, more data is better than having more features (words of the questions) or deeper models. As future work, it would be interesting to feed models with even more data to find out the true potential of these models. It would also be interesting to modify data by: keeping HTML tags so that deep models create features for text inside tags, e.g. programming code, cleaning data even more, among others. We used word embeddings with vectors of dimension 50. It would be interesting to use word vectors of bigger dimension. Finally, we used subset accuracy to compare our models, which may be a harsh metric. Therefore, it would be interesting to use other metrics that reward a model for predicting some tags correctly, even though not all of them are correct.

References

[1] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. 2, 4

[2] A. K. Saha, R. K. Saha, and K. A. Scheneider. A Discriminative Model Approach for Suggesting Tags Automatically for Stack Overflow Questions. *MSR '13 Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 73–76, 2013. 1, 5

[3] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*, 2006. 4

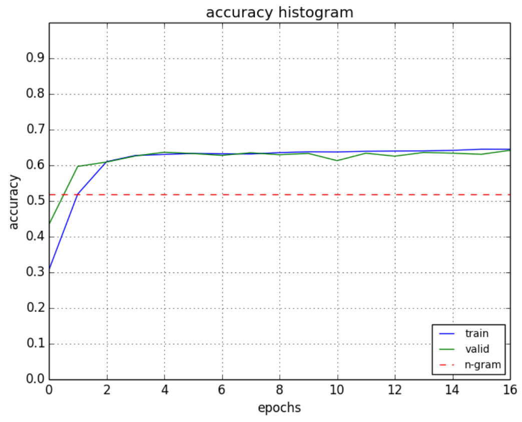


Figure 3. acc-2LSTM-200ks-400w

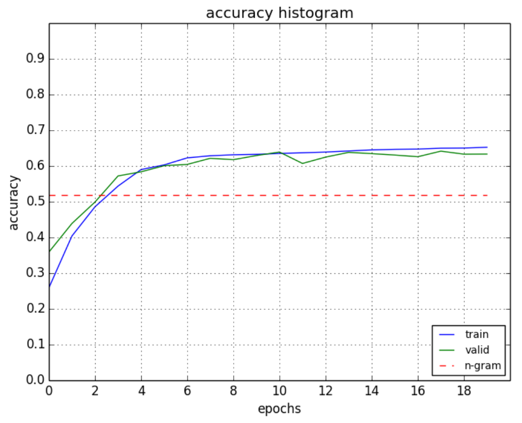


Figure 4. acc-3LSTM-100ks-400w

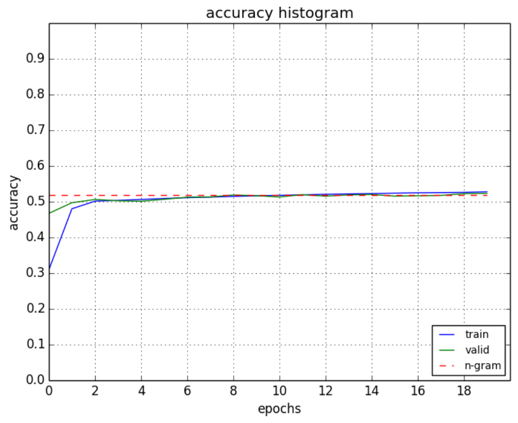


Figure 6. acc-3LSTM-200ks-titles

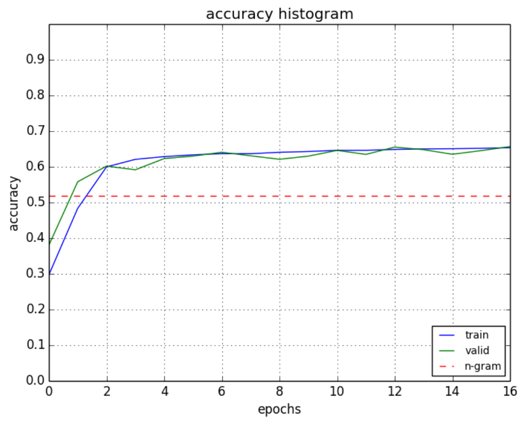


Figure 5. acc-3LSTM-200ks-400w