Precise Question classification for applications in customer service

Khizr Ali Pardhan (kpardhan@sfu.ca)
Cognitive Science Program
Simon Fraser University, 8888 University Drive
Burnaby, BC, V5A 1S6 Canada

Abstract

Automatically annotating documents with keywords is important for enabling users to retrieve the information they need. This concept can take many forms, for example, labeling questions with “tags”. A user will not necessarily know the appropriate terms to google, let alone to “tag” their question. This core concept can be readily applied in industry, in the first point of customer service, which is a major expense for businesses. This paper explores precisely this real-world use case. I compare a variety of models ranging in theoretical grounding and number of parameters to achieve a precise multi-label classification of a given question. That is, the output will classify the user’s question with up to five predetermined labels, referred to as "tags".

1 Motivation

Tagging is one of the most common ways to organize and search content online, spanning across domains and online communities. One such example is Stack Overflow, a question and answer community, which allows the asker of a question as well as community editors to assign up to five tags to categorize a question. This is necessary to asset the search function. Tags are of varying granularity, covering small differences such as sql and sql-server-2008, and also several tags relating to Microsoft’s .net framework. I will use the "StackSample" Stack Overflow Dataset (sta, 2016) to categorize questions to their corresponding, predetermined, tags. Suppose you have a question with a title and body of text, my model outputs which tags should be associated with the question.

This Task of Multi-label classification of users’ questions is of particular importance in the domain of customer service, precisely categorizing customers’ questions can route them to the next automated service or directly to a human. Currently automated customer service agents, "Bots", are only ideal for trivial inquiries, which coincidentally are often common. My work experience and investment background are focused on telecommunications. My recent employer, one of the major Canadian providers, uses minimal user-facing automation to ensure the integrity of the customer experience. Conversely, my mobile service provider, a smaller budget alternative, has heavily automated common processes despite the user experience being far from perfect. Such systems classify text to predetermined options, ranging from automated tasks to direct connection to a customer service agent, in a large variety of departments.

A special feature of such automated systems is that they can detect a cryptic web server error, and label the user’s query as labels to customer service departments, but also mapped to action items. For example, first, the customer can be routed to the more sophisticated, level 2 tech support specific to a particular application. Second, an automated vigorous test can be run to confirm the error is server-side. In a final step, if needed, the web-portals development team can be notified. This helps bridge the gaps between tech support and developers, enabling faster action on issues in a production system and well control over the Sensitivity (recall) & Specificity, as user-level support inquiries and issues are not only displeasurable but also unrelated to the work of developers. Yet in case of an outage, it is crucial that issues are addressed with urgency. At a business level, an outage can mean lost business and hence, lost revenue. There are few monopolies, and the market encourages "perfect competition". On the topic of customer psychology, rarely will a user complain when it’s faster and the same prices go to a competitor, so it is important to consider whether the underlying issue is a technical failure. In the absence of automation, there is a strong reliance on level 1 tech support to propagate technical issues to the relevant departments – let alone human errors in the communication chain and the spamming of emails to an internal-support mailing list consisting of many developers.

Customer service is the inspiration behind my work in question tagging. I will classify programming-related questions into one or more predetermined categories, "tags”. I have elected to focus on software development-related questions as the varying granularity and similarity of tags make this problem more challenging. The data was readily available as part of a Kaggle challenge but was encoded in HTML[1]. Though Stack Overflow questions are a different domain, the task is the same, and the challenges are not presumed to be less challenging. In the context of Stack Overflow, my work is useful for suggesting tags and flagging the misuse of a tag. To ensure real-world relevance, F-score is the primary metric rather than accuracy which neglects how the tags are distributed. This project is a comparative study comparing the F beta of various methods. The central research question is whether automated tagging can be accurately done in a data-driven manner, and on the computational costs associated with the modeling.

2 Prior Related work

This project is based on a dataset featured in a Kaggle challenge, which would typically enable direct comparison, however this is not the case for a variety of reasons. After finishing the data preprocessing I compared my method against the top 7 voted Kaggle notebooks. From analyzing the work of several others, I found some did not do classification or machine learning and hence did not deal with labels. Furthermore, all remaining I analyzed, including the top voted submission, combined all labels ("tags") into a single label. This means other authors elected to handle multiple labels by combining them into a single ordered label delimited by the space character. I found it to be wholly unprofessional and suspect it would severely reduce both accuracy and F-score.

Zangerle(Zangerle et al., 2011) focused on Twitter hashtag recommendation, given a tweet. This was a fundamentally similar problem, though the authors used a non-machine learning approach. Wang(Wang and Davison, 2008) explored tagging in the context of web pages and limited the scope to 140 tags, the authors claim superior performance to major search engines. Finally, Beyer(Beyer and Pinzger, 2015) covered the topic of suggesting synonyms for stack overflow tags, with statistical methods. A key focus of this work was redundant tags, as they diminish the user’s ability to search and retrieve information. For example, it may not be necessary to have a tag for every version of C++ and Java

3 Data

Originally there were 1,264,216 unique questions and 32057 unique tags. Due to computation constraints, multi-label classification with 32057 categories was unmanageable, and the target per train test split would be a 1,264,216 by 32,057 matrix. Let alone the lack of training samples for the rarest tags. Given the distribution of the data, this was turned into a 101 multi-label classification. This reduction was done by requiring a minimum of 2000 occurrences of a given tag. I believe this design decision is appropriate as the dataset consists of over 1.2 million samples. After these steps, only 0.3151% of the tag categories were remaining, and 498,996 questions, which is 39.47% of the original. The vocabulary size after prepossessing is 309411, which is about three times greater in the unaltered dataset. Lemmatization would at most reduce the vocabulary by 5%. I had considered whether it would be advisable to require questions to have a minimum score of 3. This is because it stands to reason the quality of data will be higher, as there have been more views and hence, community more edits to ensure the data is closer to the ground truth. This additional criterion resulted in a sister dataset with 160123 questions, 12.67% of original, but also has 101 labels. The vocabulary size was 121824, less than half. This illustrates the higher-voted questions are fundamentally different.

In terms of processing the question, the HTML was decoded, and I removed punctuations, numbers, single characters, and both trailing and multiple spaces. The tags were encoded as one-hot vectors which in turn were summed on the question axis, to be a multi-label encoding. The result was a 498,996 by 101 matrix, about a tenth of the size had there been no data removal and filtering. The most common single label on the dataset is “Java” at over 6.6% occurrence, followed very closely by “JavaScript” and “C#”. This can be observed in figure 1.

There were three encoding methods for questions. Option 1: naively tokenize each word to an

[1] https://www.kaggle.com/stackoverflow/stacksample

integer and pad the sequences. As expected there was a massive right shew in the length of sequences, the ten longest questions were between 3000 to 3500 words. Given that models need consistent sizing, the next step was to pad the sequences. Originally, I had used the smaller of 200 and the median with 1.5 times the interquartile range. After further considerations, this was changed to the mean length of a question, about 150 words. Option 2: Glove Embedding. There is only 17.86% vocabulary found in the pre-trained embedding. This means the embedding layer should be trained upon, which was I elected to do mid-training procedure. It is worthy of mention that in the sister dataset with a minimum score of 3, there was a 27.08% portion of vocabulary in the Glove embedding. Recall that this sister dataset had less than half the diversity in vocabulary. The embedding matrix can be initialized with normally distributed vectors, as initialization with zeros may be harder to train. Option 3: TF-IDF converts text into fixed-length numeric vectors while considering frequency. The equation is as follows, $\text{tf-idf}_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$ There were originally 374,513 features, and hence that number of dimensions. By requiring a term to be in a minimum of five questions, there were only 108, 967 features. Likewise, minimum occurrences in questions of 200 and 1000 resulted in 8908 and 3385 dimensions respectively. With this filtering, there would be 3385 features in the TF-IDF vectors.

The most common single label on the dataset is “Java” at over 6.6% occurrence, followed very closely by “JavaScript” and “C#”. This can be observed in figure 1. My baseline model is simply predicting every question to have the single most common combination of tags, which is the single tag of Java. Unexpectedly, this yields an Accuracy, Recall, Precision, and F1 score of 0.

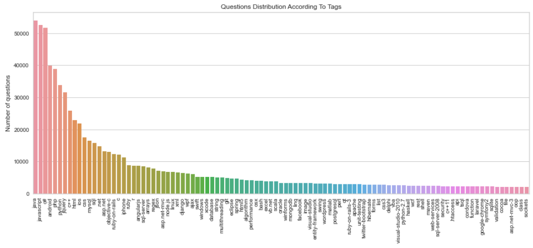

Figure 1: The distribution of labels ("tags")

4 Models & Experiments

The main libraries I used were sklearn(Pedregosa et al., 2011) and TensorFlow(Abadi et al., 2015) and its associated subpackage, Keras. The model I used was a 300-dimension glove(Pennington et al., 2014) embedding, followed by a bidirectional LSTM, very light non-recurrent dropout, LSTM, ReLu activated dense, and finally, a sigmoid activated dense layer. The padded sequence to a length is about 150, though these are large dimensions, it was tractable, and the model trained. The loss function was binary cross-entropy, which is probabilistic in nature.

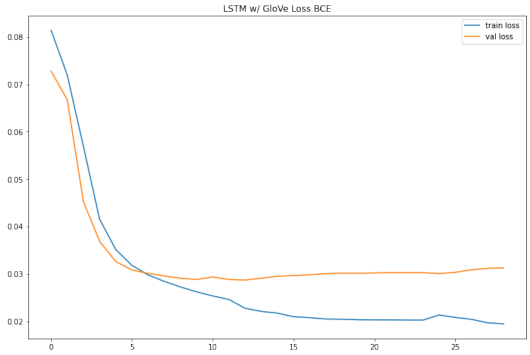

Figure 2: BCE Loss of a Glove LSTM

The sklearn models: ExtraTrees, RandomForest, and Multilayer Perceptron models were also tested, though they performed poorly. Despite programmatically selecting the top few predictions for each model, to ensure the models did not predict null output, there was still a minimal score across all metrics, on the tokenized inputs, as observed in table 1. Conversely, the aforementioned models paired with TF-IDF worked significantly better, per the case of table 2. It is most surprising that the Multilayer Perceptron performs significantly better than the other two sklearn models. In fact, the performance was nearly on par with the LSTM with embedding.

The best Multilayer Perceptron on TF-IDF vectors had an F1 score of 0.661, which was superior to any non-neural network. Only an LSTM with retrained GloVe embedding was able to outperform, with an F1 score of 0.7544.

Metrics are a crucial discussion topic because they also form quantified compassion, but they are often unintuitive. The F1 score is the harmonic mean of precision and recall. It is important to note that a high precision implies low false positives, while a high recall (sensitivity) implies few false

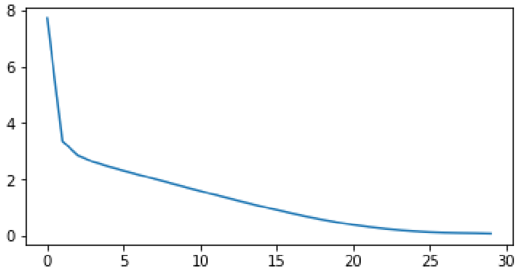

Figure 3: Loss of a Canonical MLP

negatives. It is a good balance as it in a sense is two metrics in one. Though this metric is not without criticism, as it presumes every misclassification is equal, though, in industry, some misclassifications might be of significant consequence. F1 score is extended with the notion macro and micro averaging(Gupta, 2016), as a means to handle class imbalance. Given that my data has a class imbalance, I choose to use the F1 score, over other metrics such as accuracy.

I have found that the majority of the work that needs to be done is centered on data preprocessing and memory-related challenges. There were originally 32057 tags. I have reduced the output to be a multi-label classification between 101 tags to elevate this issue and as well as stick to the target multiclass use case of assisting a customer service representative, as they would not select from such a high number. Many models, such as my preferred, XGBoost(Chen and Guestrin, 2016), are more compatible with multi-label classification problems, though they work with multiclass classification. In this case, the "One Vs Rest" classification method could have been employed, though the great computational cost, by nature of the strategy, as it consists of fitting one classifier per class.

Due to computational costs, I key point of the investigated dimensionality reduction on both encoding methods. Under the working hypothesis that there would be a minimal decrease in F1 score, I attempted a variety of algorithms(Pedregosa et al., 2011), principal component analysis (PCA)(Ljungberg, 2017) and singular value decomposition (SVD). SVD when applied to TF-IDF vectors is known in natural language processing as latent semantic analysis(Ljungberg, 2017). Every experiment attempted around dimensionality reduction resulted in relative inferior performance and longer training time, despite having about half the dimensions and an explained variance ratio between 0.66 and 0.85.

5 Results

I elected to use an LSTM based model(Thangaraj and Sivakami, 2018) in Keras. Neural network models are not only better but can achieve a higher F1 score in less time. In the case of the MLP classifier, it was in sklearn and hence on the CPU, furthermore, it was single-threaded. The MLP classifier out only outperformed nonneutral network models, but also was the most computationally efficient. To answer my hypothesis, neural networks are successful in question tagging, by qualitative analysis, and in computational efficiency.

Regarding the LSTM model, it was interesting that the loss curve was particularly steep, particularly in the other non-reported architectures, which had fewer parameters and trained with lower batch size. Both these factors caused more learning per epoch, causing the loss to have relatively little reductions after the first epoch. This is partially due to the very large size of the dataset.

The sklearn models(Pedregosa et al., 2011), ExtraTrees, RandomForest, though both multithreaded, took exceedingly long to train and took a significant amount of time to run, perhaps over 10GB. Sklearn lacks a partial fit method, which may be beneficial in situations where the dimensionality of the data is very high. One point of contention left unexplained is why the sklearn models on the tokenized representations of words performed so poorly, even were forced to output the two most probable predictions.

Given the high computational cost of computing, and other factors, such as controlling for the quality of data, I hypothesized, what would result if I discarded questions with a low score, as those may be inherently low-quality questions and may not have been edited to reflect proper details, let alone that stack overflow users can actually tag others' posts, in effect ensuring tagging correctness. Tables 3 & 4 showcase the results of every classifier being rerun on the sister dataset. In analysis table 2 & 4, it is observable that the MLP models offer higher recall than other Sklearn models. If there were multiple models performing well, a natural next step could be to have an ensemble of models.

6 Limitations

Given time and computational limitations, there were several issues. First, my models which trained

Table 1: Questions Tokenized

	precision	recall	f1
ExtraTrees	0.000	0.000	0.000
RandomForest	0.000	0.000	0.000
MLP (4 layer)	0.000	0.000	0.000
LSTM	0.757	0.762	0.754

Table 2: TF-IDF

	precision	recall	f1
TF-IDF 3385 dim			
ExtraTrees	0.798	0.337	0.446
RandomForest	0.795	0.425	0.521
MLP (3 layer)	0.752	0.604	0.661
MLP (4 layer)	0.727	0.615	0.658
TF-IDF 8908 dim			
ExtraTrees	0.794	0.278	0.387
RandomForest	0.797	0.352	0.454
MLP (4 layer)	0.752	0.540	0.619

Table 3: Tokenized min score 3

	precision	recall	f1
ExtraTrees	0.159	0.000	0.000
RandomForest	0.237	0.000	0.001
MLP (4 layer)	0.049	0.001	0.002
LSTM	0.713	0.722	0.715

Table 4: TF-IDF min score 3

	precision	recall	f1
TF-IDF 3378 dim			
ExtraTrees	0.794	0.319	0.422
RandomForest	0.768	0.381	0.480
MLP (4 layer)	0.650	0.594	0.619
TF-IDF 4519 dim			
ExtraTrees	0.798	0.296	0.406
RandomForest	0.776	0.379	0.481
MLP (4 layer)	0.685	0.591	0.685

successfully would often suggest perfect plausible tags, but the model did not know how many terms to predict. This limitation was addressed, but it involved looking at the number of terms in the target, though not the contents. Before this, it was found that selecting all tags with a minimum probability of 0.3 yields a greater F1 score than selecting the top N tags. This was perhaps due to the model learning that the number of tags is partially a function of the given question. Table 5 shows the top five predictions and the labeled target data.

As in many machines learning experiments, there were many design choices made. It is unclear whether training on low-quality data is beneficial, though it is technically valid, its low quality might add noise and hence detriment learning. Higher voted data could potentially be trusted to be more precisely and completely labeled. Lemmatization could not be done due to computational limitations. After training on a small sample of the data, there was a 5% reduction in the vocabulary. I suspect the rest of the data would disproportionately differ due to technology-related nouns, meaning I expect a less than 5% reduction in vocabulary. Given the nature of TF-IDF, it was necessary to discard a significant amount of vocabulary, this led to the hypothesis where extra vocabulary was beneficial, and lead to the modeling being retrained on higher dimensional TF-IDF and lead to a decrease in F score on every classifier. This experiment resulted in an overall decrease in F1 Score across all tested models, as seen in table 2, TF-IDF with 3385 and 8908 dimensions. The precision was minimally impacted but the recall is decreased with a richer vocabulary. I found these results counterintuitive. Perhaps more data or oversampling technique is needed, as that added vocabulary is less occurring.A potential limitation was the loss of valuable information in the preprocessing stage, and it may have been better to retain single-character tokens, numbers, and symbols. A separate experiment would need to be conducted. It is possible that, If punctuations were maintained, then the code of the form “object.attribute” would be a single token and perhaps act as an n-tuple, which could allow a model to gain a deeper understanding. Though, conversely, there would be an increase in vocabulary.

Throughout this work, computation abilities were stretched. Google Collaboratory was often inadequate in terms of CPU and memory, while my local machine was often inadequate in terms of GPU memory. This issue caused several OS level blue screen crashes of my local machine, conversely, CPU-related crashes are often confined to the program level. My local machine is an old i7 4790k, 24GB ram, and a new-architecture medium-range RTX 3060 TI 8GB. The CuDNN backend was used with the higher level Tensorflow(Abadi et al., 2015). A bidirectional LSTM would not work on Collaboratory GPU, this was unexpected as the local GPU has less memory.

There were some model-related limitations, not related to computational capabilities. There is no comparison with an LSTM without a Glove, which would demonstrate the impact of embeddings in natural language processing. It is possible to leverage a pertained BERT(Devlin et al., 2018) model for encoding, in lieu of GloVe. On the topic of GloVe, the embedding used may have benefited from a more sophisticated training manner on more relevant data, sure as the stack overflow answers dataset which was not used in this paper. This is needed as the pretraining is mainly on disjoint vocabulary, there, in all test cases, was less than 30% coverage of the dataset vocabulary. Lastly, the LSTM with GloVe embedding I trained was overfitting, I did not have the time to address it fully. This can be seen in figure 2. There was only the addition of light classic, non-recurrent, dropout, and did not attempt of layer normalization, which is more suitable for an LSTM than batch normalization. I read others' works only after completing my project, in an effort to be honest. It appears that another metric that fits multi-label classification is the Jaccard score, which in computer vision is known as the Intersection over Union (IoU) metric. The inclusion of this metric may have been insightful. Similarly, the Hamming loss could have also been an insightful metric. Though, the F1 score is by no means inadequate. Choosing a primary metric is subjective at best.

7 Conclusion

Data preprocessing, data engineering, and dimensionality reduction play a significant role in model performance. It was found that Neural networks, even when non-sequential are superior to other tested models. It is clear they are worth the necessity of a GPU from processing. Additionally, it was found the TF-IDF, though significantly more sparse and memory-intensive is superior to tokenization. The only case in which tokenization was found to be beneficial was when used in conjunction is a GloVe embedding, which is a significant mutation. Tagging question plays a key role in serving customers, in this context a "tag" can be the triggering of a test, sending of a report, or categorization of a query. Tagging is a complex task as benefits greatly from data preparation and models.

References

2016. Stacksample 10 precent of stack overflow qa.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Stefanie Beyer and Martin Pinzger. 2015. Synonym suggestion for tags on stack overflow. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 94–103.

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Shashank Gupta. 2016. Micro average vs macro average performance in a multiclass classification setting.

Benjamin Fayyazuddin Ljungberg. 2017. Dimensionality reduction for bag-of-words models: Pca vs lsa.

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. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

M Thangaraj and M Sivakami. 2018. Text classification techniques: A literature review. *Interdisciplinary Journal of Information, Knowledge & Management*, 13.

Jian Wang and Brian D Davison. 2008. Explorations in tag suggestion and query expansion. In *Proceedings of the 2008 ACM workshop on Search in social media*, pages 43–50.

Eva Zangerle, Wolfgang Gassler, and Günther Specht. 2011. Using tag recommendations to homogenize folksonomies in microblogging environments. In *International conference on social informatics*, pages 113–126. Springer.

Table 5: Top 5 predictions in alphabetical order

Prediction	css	html,	html5	javascript	jquery
Target	css	forms	html	javascript	
Prediction	.net	c	c#	c++	c++17
Target	c#				
Prediction	algorithm	android	image	java	xml
Target	android				
Prediction	database	mysql	postgresql	sql	sql-server
Target	mysql	sql			