Automatic Question Tagging using k-Nearest Neighbors and Random Forest

Virik Jain, Jash Lodhavia

Department of Computer Engineering, VJTI, Mumbai, India
vrkgajani@gmail.com, jashlodhavia15@gmail.com

***Abstract*—Stack Overflow is one of the most widely used platforms for asking questions and queries on topics related to computer science, software development and general computer programming. Tagging of the questions is particularly useful for indexing information based on the tags. Currently, a user enters the tag manually for a question asked by him/her. The question should contain at least one tag manually typed by the user. It can be seen that most of the questions asked should either have more tags associated with it or aren't tagged accurately and appropriately. Since there are a huge number of tags, the process of searching through all the tags manually and find relevant ones can be cumbersome and is therefore overlooked by most of the users asking the questions. This research is focused on exploring methods for developing an autonomous tagging system using Machine learning methods like k-Nearest Neighbors and Random Forest along with some crucial data preprocessing steps like Stemming, Tokenization and removing Stop words. The dataset for the above research is taken from kaggle.com which has a 10% Stackoverflow question dataset open for all. The results of the following proposed system for automatic tagging were satisfactory. Random Forest gave an average percentage accuracy of 70% across all the tags while k-Nearest Neighbors performed slightly better giving an accuracy of 75%.**

***Index Terms*—Machine Learning, Automatic Question Tagging, k-Nearest Neighbors, Random Forest, Natural Language Processing**

I. Introduction

The task that we try to propose a solution for is that of tagging(or labelling) some given piece of text, specifically questions. With the ever increasing number of users using the internet, there has been a tremendous growth of the Q&A websites. Q&A stands for Questions and Answers. Q&A sites like Quora, Stackoverflow and forums of other non Q&A sites receive a large number of questions each day. If the questions are not segregated or classified then they will get lost in the large pool of unanswered questions. For this reason, questions are classified by assigning them tags or labels. Assigning tags helps in clubbing similar questions. Also assigning proper tags will make it more likely for the question to reach the proper target audience and thus that question will have a higher chance to be answered [2]. One choice is to have employees to tag these questions. But there are several major drawbacks of such manual tagging. First of all, the scale of the Q&A websites is so large that it is practically infeasible for all of the questions to be tagged manually by employees. Secondly, even if we somehow overcome this then there is the issue of human bias that can lead to incorrect tagging of questions. Since manual tagging is so laborious and error-prone there is a need for a more robust solution. This is where automatic tagging comes in. In an automatic tagging system, we feed the question to the model that then assigns tags to the question. Such a system overcomes the major drawbacks of manual tagging as it is not just quick and automatic but also less error-prone since it is free from any human bias. The system relies on past data to make its decisions and today we have access to a large amount of data that we can build a model that can confidently make tagging decisions. Since a question can have more than one tag, there are multiple ways to tackle this problem. One is to build a binary classifier for each of the possible target tags and second is to use multi tag classifier algorithms. In this project, we focus on the data from Stack Overflow. Stack Overflow is a popular Q&A website for programming questions. Since the total data is very large, we use a small subset of the data that is available on Kaggle website, 10% of Stack Overflow QA.

Main objectives of this paper are:

- Providing a way to efficiently pre-process text to facilitate effective classification.
- Providing the algorithm for classification that is sufficiently accurate as well as computationally efficient.

II. Recent Work

A lot of research has already been done on automatic tagging. Some popular websites limit the number of tags to be used which is not a good solution to the problem. TagIt [8] is a system which uses Naive Bayes text classification method. The drawback to this system is that any new tag generated must be present in the training data and the training data contains on 330 tags thus, limiting the number of tags. Another approach was in [9] which proposes an automatic recommendation algorithm called TagCombine. The paper presents a method which has three systems namely, multi-label ranking component, similarity based ranking component and tag-term based ranking component. In the tag-term based component, bag of words are converted into TF.IDF (Term-frequency, Inverse-document frequency). Another approach is mentioned in [10] called EnTagRec. EnTagRec computes tag probability scores using two separate methods, Bayesian Inference and

978-1-7281-8041-0/20/$31.00 ©2020 IEEE

Frequentist Inference, and then take the weighted sum of their probability scores.

III. PROPOSED METHODOLOGY

We propose our method for autonomous tagging system in this module. The data used to test our method was taken from kaggle.com . The following steps illustrates our method in detail. The flowchart presented in Fig. 1. explains our proposed method in brief.

Fig. 1. Pre-Processing Steps

A. Data Preprocessing

Before we could use the data to train our model, there was some preprocessing that needed to be done. In this section we describe the different stages of text preprocessing performed before extracting features to be used for training.

- **Strip HTML Tags:** The default text in the data is in HTML format. Since we would like to get rid of the tags as they do not necessarily add anything to understanding the context of the question. We use the Beautiful Soup library in Python for this.
- **Lowercase:** We then convert text to lowercase as case sensitivity doesn't add to the understanding of the question. Eg: "Python" and "python" both are equal and uppercase or lowercase letters don't make a difference to the question.
- **Remove Numbers:** While numbers can be useful in understanding some type of questions, for the most part they do not play any role in our task of determining the labels and so the numbers are removed from the text.
- **Remove Punctuation:** Punctuation too does not add any information to the context of the question and so the text is stripped of punctuation.
- **Remove Whitespaces:** After performing the previous steps, our text may end up having some unnecessary whitespaces. We thus remove these whitespaces and replace them with a single space.
- **Tokenization:** It is the process of breaking down a bigger piece of text into smaller "tokens". The tokens can be individual characters, words, sentences or even paragraphs. In our application, we use word tokenization i.e we extract one word at a time from our text.
- **Remove stop words:** Stop words are words that do not add any information to the system and may even skew our text analysis. They are generally the frequently occuring words in a language. For example, in English, some stop words could be "the", "a", "and", etc. We discuss feature extraction in detail in the next section where we use the most frequent 100 words as features for a tag. If we didn't remove the stop words then due to their high frequency they would end up being the features for every tag.
- **Stemming:** It is the process of reducing the words to their root, or base word. It can be seen as removing a suffix (possibly empty) from the word, or equivalently, replacing a word with one of its non-empty prefixes. It is done in order to reduce the similar words that convey the same information to equivalent terms thereby reducing the size of feature space. For example, the words "want", "wants", "wanted" are essentially the same for understanding the context of a text and so the stemmer might decide to reduce each one of them to the word "wan".

After successfully applying these preprocessing steps on the question title and body, we can then move on to extract features for different tags.

B. Feature Extraction

This is one of the most important steps in building the automatic tagging system. The feature extraction process from the question takes place after the data-preprocessing step. The questions in the database were converted to words by following the pre-processing steps. The questions were first converted to units of words, also known as tokenization. After tokenization, stop words such as "in", "as", "the", "and", "a", "an" were removed as they are insignificant to the question and don't carry much information as such. The final step is Stemming and we used Porter Stemmer to stem the remaining words. Feature extraction is then done as follows:

- First, top hundred tags by frequency are extracted from the database. These are the most frequently appearing tags that are present in most of the stack overflow questions.
- For each of these hundred tags, we select thousand questions (positive examples) and eight-hundred negative examples to have a balance between positive and negative examples. This is done as SVM (Support vector Machine) performs poorly on unbalanced data as cited in [4] and thus, to remove the bias, we provide the algorithm with a good balance of both negative an positive examples.
- We just chose the thousand positive examples for each tag and extract the top hundred frequently occuring words in the questions. For all hundred tags, there may be many overlapping words but on the worst case the dimension of our feature vector will be (n*10,000), where n is the number of data rows which the model will be trained on.

C. Prediction Models

We used the classifiers from python's scikit-learn library, more specifically, we used k-Nearest Neighbors (kNN) and Random Forest (RF) algorithms on our data set.

k-Nearest Neighbours (kNN) We use a kNN (k-Nearest Neighbors) Classifier for our classification task. It is a simple and easy to understand algorithm. It can also be used for Regression problems but we use it for Classification. We

explain the kNN algorithm in this section. kNN is a supervised learning algorithm i.e it requires labelled data. The basis on which the algorithm relies is as follows: similarly classified data points will be close to each other in the feature space that is, they will be clustered together. Thus when we need to classify a new data point, we find its 'k' nearest neighbors, where 'k' is a parameter of the model. The new data point is then assigned the class label of that class which appears for the maximum number of times in those 'k' nearest neighbors. For example, if k=5, and the 5 nearest neighbors for a data point are 0, 0, 1, 0, 1 then the class of 0 will be assigned to the point. For kNN to work well, the data should support its basis, that is, the data should follow the assumption that similarly classified data points are closer to each other. To define this 'closeness', or similarity between two data points, we use the distance measure in the feature space. More specifically, a point is closest to another if the Euclidean distance between them is lesser compared to all other points in the dataset. The Euclidean distance is measured in the feature space, thus in our case, with 100 features, it will be measured in a 100-dimensional space with each feature acting as a dimension. In our dataset, we use frequently appearing words for a tag as our features. Thus, in our case many similarly classified questions will have similar values for the features as frequently appearing words are more likely to appear in positive examples and less likely to appear in negative examples. We also find that from our results that kNN provides a satisfactory result. The parameter of the model is 'k'. For our application, we used the value of k=5. We found that the results were satisfactory as well as computationally efficient.

The table-1 indicates the percentage accuracy of k-NN on the top five tags (sorted by frequency) on the train and test data. As we expected, javascript is the most frequent tag followed by java.

Random Forest: We also used Random Forest Classifier for our classification task. We tuned the maximum depth parameter in a range of 3 to 8 and found that at maximum depth = 5, the Random forest classifier performed the best. The Random Forest Classifier gave us an accuracy of 72%. We explain the Random Forest Classifier as follows: The Random Forest is an ensemble machine learning method, which uses a number of decision trees to predict/classify the given data. We are using python's famous library sci-kit learn where the random forest for the dataset can be built efficiently. The hyper parameters for building the random forest were tested multiple and the below mentioned hyper parameters gave the most promising results:

- n_estimators = 150 (Determines the number of trees in the forest).
- max_depth = 5 (Determines the maximum depth of each decision tree).

Random forest doesn't just average the results of all decision trees. The two important concepts used to build the forest are:

- Random Sampling of training dataset.
- Random features considered when splitting the node.

The table-2 indicates the percentage accuracy of Random forest on the top five tags (sorted by frequency).

TABLE I
ACCURACY OF THE TOP FIVE TAGS USING K-NN

Tags	Train set	Test set
javascript	83.38	78.25
java	81.06	70.25
c#	76.31	65.75
php	86.00	79.00
android	88.12	77.75

TABLE II
ACCURACY OF THE TOP FIVE TAGS USING RANDOM FOREST

Tags	Train set	Test set
javascript	88.78	76.00
java	79.06	68.25
c#	86.31	75.75
php	81.00	78.73
android	89.12	71.75

TABLE III
F1 SCORE OF THE TOP FIVE TAGS USING PREDICTIONS MADE BY KNN

Tags	F1 Score
javascript	0.8348
java	0.8104
c#	0.7868
php	0.8266
android	0.9052

TABLE IV
F1 SCORE OF THE TOP FIVE TAGS USING PREDICTIONS MADE BY RANDOM FOREST

Tags	F1 Score
javascript	0.8078
java	0.8144
c#	0.7989
php	0.8254
android	0.7912

IV. RESULTS

Evaluation Metric: Accuracy

We tested two machine learning algorithms on the kaggle dataset and the results were as follows:

- k-Nearest Neighbours (kNN) algorithm gave an accuracy of 75% on the test set, which was created using python's train_test_split on the kaggle's dataset.
- Random Forest algorithm performed a bit worse, giving an accuracy of 70% on the same test set as that of kNN.

Evaluation Metric: F1 Score

We used another evaluation metric on our model to further analyze the results. Sci-kit learn provides us with an in-built f1_score method for calculating the F1 Score. The table III

and table IV indicated the F1 scores when calculated on the two models.

The average F1 score across all tags for Random Forest was 0.8 while KNN performed slightly better with an average F1 Score of 0.84.

V. Conclusion and Future Scope

This paper proposed an efficient method for autonomous tagging of questions for forum websites such as stack overflow. The obtained results are quite good with Random Forest classifier giving an accuracy of 70% while the k-Nearest Neighbour algorithm giving an accuracy of 75%. We have used traditional Machine Learning Classification algorithms namely k-Nearest Neighbors(kNN) and Random Forest(RF). While they have provided satisfactory results, there is always scope for improvement. Both the industry and research community is moving towards using Deep Learning and Neural Networks for modern problems instead of traditional Machine Learning Algorithms. While there has been some work in Automatic Question Tagging using Neural Networks, the majority of work has been focused on using traditional algorithms like Support Vector Machines(SVM) and Naive Bayes as we saw in the Previous Work section. We believe that Deep Neural Networks have great potential and can thus be used for the task of Automatic Question tagging.

References

[1] Charte, F., Rivera, A. J., del Jesus, M. J., and Herrera, F. (2015). QUINTA: A question tagging assistant to improve the answering ratio in electronic forums. IEEE EUROCON 2015 - International Conference on Computer as a Tool (EUROCON).

[2] Saha, A. K., Saha, R. K., and Schneider, K. A. (2013). A discriminative model approach for suggesting tags automatically for Stack Overflow questions. 2013 10th Working Conference on Mining Software Repositories (MSR).

[3] Gonzalez, J. R. C., Romero, J. J. F., Guerrero, M. G., and Calderon, F. (2015). Multi-class multi-tag classifier system for StackOverflow questions. 2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC).

[4] Saini, T., and Tripathi, S. (2018). Predicting tags for stack overflow questions using different classifiers. 2018 4th International Conference on Recent Advances in Information Technology (RAIT).

[5] Sahu, T. P., Thummalapudi, R. S., and Nagwani, N. K. (2019). Automatic Question Tagging Using Multi-label Classification in Community Question Answering Sites. 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom).

[6] J. B. Lovins, Development of a stemming algorithm. MIT Information Processing Group, Electronic Systems Laboratory, 1968

[7] J. Wang and B. D. Davison, "Explorations in tag suggestion and query expansion," in Proceedings of the 2008 ACM workshop on Search in social media, ser. SSM '08. New York, NY, USA: ACM, 2008, pp. 43–50.

[8] Sood S, Owsley S, Hammond KJ, Birnbaum L. TagAssist: Automatic Tag Suggestion for Blog Posts. InICWSM 2007 Mar 26.

[9] Wang, X., Xia, X. Lo, D. TagCombine: Recommending Tags to Contents in Software Information Sites. J. Comput. Sci. Technol. 30, 1017–1035 (2015).

[10] S. Wang, D. Lo, B. Vasilescu, A. Serebrenik. EnTagRec: An Enhanced Tag Recommendation System for Software Information Sites. ICSME, 2014.