**Tag suggestion and text classification**

Aoshi Li, Yuanyi Yang

**Abstract**

In this paper, we test several combinations of feature extraction, feature selection, classification method, several NLP tools and parameters tuning. We also tried simple rule based system to understand why it is necessary to use machine learning. After that, we tried a more challenging task to extract tags from the given material.

**keywords:** social tagging, text classification, machine learning, web search

**1 Introduction**

As the information grows in an exploding speed, the demand for acquiring useful information is also steadily increasing. Yet due to the large number of ambiguities and varieties, it can be rather hard to get what we want. Merely searching by keywords will not completely solve this problem because searching simply by keywords is affected by a lot of factors such as synonyms, user’s unfamiliarity with domain keywords, the ranking method and so on.

To partly solve this problem, we can think of text classification as a part in the vertical search. Of course this is better than searching with combination of keywords and the representative words of the keywords’ domain. But the procedure to be taken in order to get the category of one document is a fairly long pipeline. The main components of it in sequence is: web crawler, main content and tags extraction, tokenization, stemmer, stop words and punctuations removal, feature extraction, feature selection, and classification. Among them, feature extraction and classification methods are especially important to the system performance. A large number of studies have been conducted under this domain. However, up to the writer’s knowledge, no paper claims to have discovered the strict relationship between the different ways of handling data and the final output. So instead of using a steady method, people try out different combinations of the modern systems and compared their efficiency.

Mckerlich, etc. from New York Univeristy[1] published a paper in 2013 describing the influences of different choices of training methods and features. Their result suggested several good combinations. But the most important conclusion they made is that there is no universal true text classification method. Data plays a very important role here. So it appears that we have nothing to do except for trial and error.

Tag recommendation is another way to increase information retrieval efficiency, in a much more personal way. Think in this way, if the set of tags are limited. Tag recommendation is just text classification. Indeed, given a large enough corpus, we could simply train it by aggregating all the labels, and do the traditional classification. However this methodology suffers from several cons. First, it could not recommend tags apart from the given tags. Second, some tags are of rather low frequency. The traditional way of training will inevitably suffers for this “long tail”. Finally, the corpus should be huge to contain a reasonable number of tag keywords. As a result, the training time would be so long that it is almost impossible to do such experiment, let alone the challenge of finding such corpus.

Therefore, instead of doing tag recommendation by brute force. Several researchers tried some new ways around. They turn the problem into finding high frequency words of the corpus, graph problem, and recommendation system problem. [2] But different from text classification, the difficulties and possibilities of tag recommendation is tremendous. Different users have different tagging habits, making it almost impossible to get a good model. We propose a simple rule inference algorithm based on personalized tagging habits and data.

**2 Data and Methods**

**2.1 Text classification**

The dataset we use is 8298 articles crawled from website, “the verge”. Most of the articles on that website belongs to several main categories such as technology, culture, transportation, etc. For each category, there are some sub-categories which are also assigned to articles.

**2.1.1** **Preprocess data**

Preprocessing data is not an easy task. There are several issues to be solved before getting a clear, easy to train data.

1) Crawling the website.

This task is not as simple as it sounds. Even in a single website, it can be difficult to extract just the information we want. We used jsoup to write regular expression and identify where the titles and tags are. To find out the main text, we have to use another java library called boilerpipe since there is no explicit signs to identify where the body of the article is, this library guesses where it is. Then we save the crawled data to a json dictionary format for future processing.

2) Clean the text.

There are several traditional cleaning steps to be taken before training. Among them is tokenization, stop words removing, and stemming. (Stemming is the process of transforming a word to its simplest form. e.x. “bought” to “buy”). These are done using python nltk library.

3) Feature extraction.

Then we need to decide which feature to use. We cant expect any algorithm to work by just giving it a whole block of text. We have tried bag of words and tf-idf. Since one feature of tf-idf is just a number if that word appear in the text and zero if not. This model works also for bag of words since the model of bag of words treats each word as either appear or not. Thus we use the function provided by sklearn to turn the whole corpus into a vector of idf value. Then during training, we multiply the term frequency with the idf value of that word and produce a vector of each word’s tf-idf value.

4) Feature selection.

It is impractical to use all the words as training features. That number is 40093, too slow for training. After feature selection, there are only 7003 words left. Also, feature selection can help improve the accuracy by eliminating non-representative words, which will only bring noise to the model if treated improperly. The method we use to select features is chi-2. Basically, it calculates the appearance of each word among its category and compare it to the whole corpus. If that word is distributed uniformly within the corpus. Then we should treat it as a stop word.

**2.1.2 Training method selection**

A lot of works have been done in text classification training method. And we are not so ambitious to suggest a better one. Instead, we would like to see the results of different choices of training methods and try to find a best one to suit our need. Hopefully, we can also learn something during this process. The method we choose is listed in the results.

We shall briefly talk about the methods we use here.

1) Naive bayes: naive bayes has two different ways of calculating the conditional probabilities of each feature. One way is to treat all the words in the corpus as the sample space. The other considers multiple occurence in a document of a single word as only one word. This is the most important difference between these two models. The second works better for small document because a second appearance of a word in a document might be just purely by chance and reflect nothing about relationship between that word and the possibility of the corresponding label.

2) Support vector machine: support vector machine is usually considered one of the simplest and most efficient algorithms when considering classification. Here we use words and their tf-idf values to represent the whole model, hoping there can be a high dimensional plane to separate the two categories.

3) Logistic regression: logistic regression’s idea is also very simple. Assign a weight to each of the feature, multiple to each word, add them together, and turn them into a range between -1 and 1. A loss function is used to calculate the weight aiming at maximizing accuracy of prediction.

**2.2 Keyword extraction and tag recommendation**

Another way of thinking about this problem, doubtfully the more interesting aspect, is to extract keywords from article and recommend corresponding tags. To recommend more concrete tags instead of general category, we cannot follow the old strategy because the potential tag pool is infinite. The words or phrases that could be used as tags are constantly growing, making it impossible to use a fix set leave-one-out training method to recommend tags. To overcome this difficulty, we have tried both Chinese and English data under this direction. For English version, we use MIT news as dataset. For Chinese version, I use my own Evernote as dataset.

**2.2.1 Preprocessing data**

English version: We crawled all the tags used in MIT news, 81 pages of tags. Notice that we could simply add more words into this tag pool if needed. The added time cost is small. Each tag has a number attached to it indicating how many articles are labeled under this tag in MIT news. This is an important information that we will use later on.

Chinese version: Evernote’s notebook can be exported as its own .enex format. This format is actually written in xml. Therefore, the first thing to do is to traverse each note, parse xml and extract title, text, and tags for each one of them. After that, we combined wordnet and keyword extraction to suggest potential tags for users.

**2.2.2 Recommend tags**

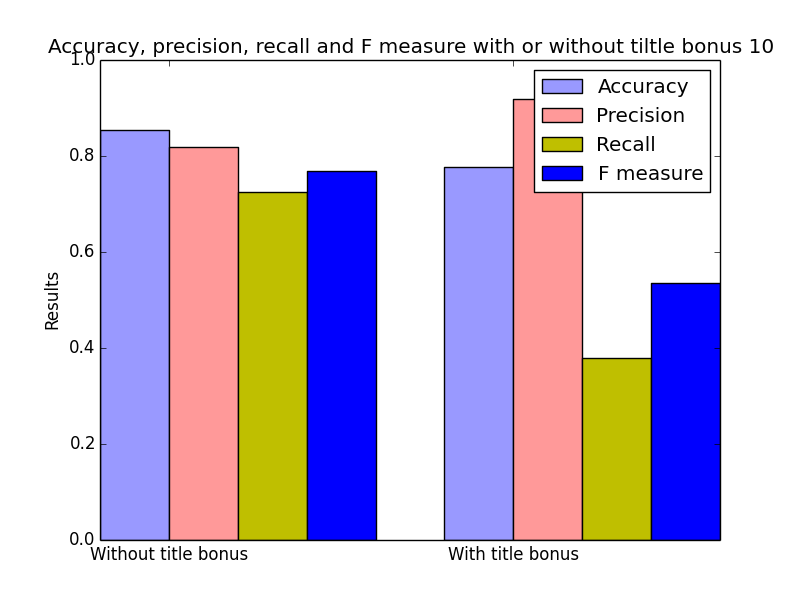
English version: There is a classic algorithm called RAKE doing keyword extraction. We tried to combine this with wordnet. First we scan the input articles, extract keywords and find their synonyms in the wordnet. Together, they are the potential tags that user can choose from. The reason we use synonyms is to increase the chance of hitting the words used in the actual tag pool. This procedure will inevitably lowers precision, but since its contribution to recall, we provide this feature as an additional field.

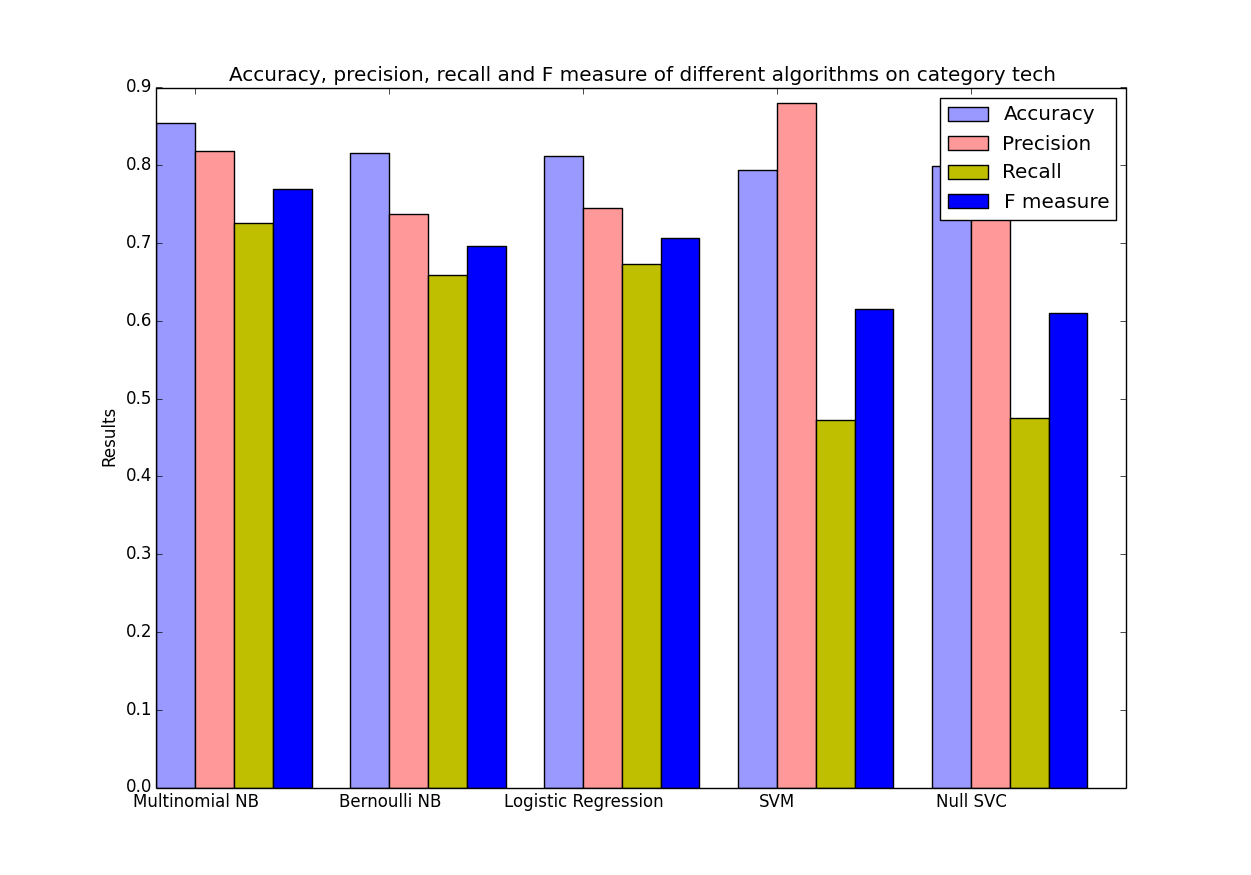
It is reasonable to sort the list of recommended tags before given to user. Otherwise, the number of potential tags is too large for user to get what he wants. We use a trick here similar to tf-idf. Remember we have previously recorded the number of occurrence of documents for each tag in the tag pool. Multiplying this number with the score of the extracted keyword, we get a good estimation of how well this keyword will be the actual tags. Only the keyword that actually appear in the final tag pool is used.

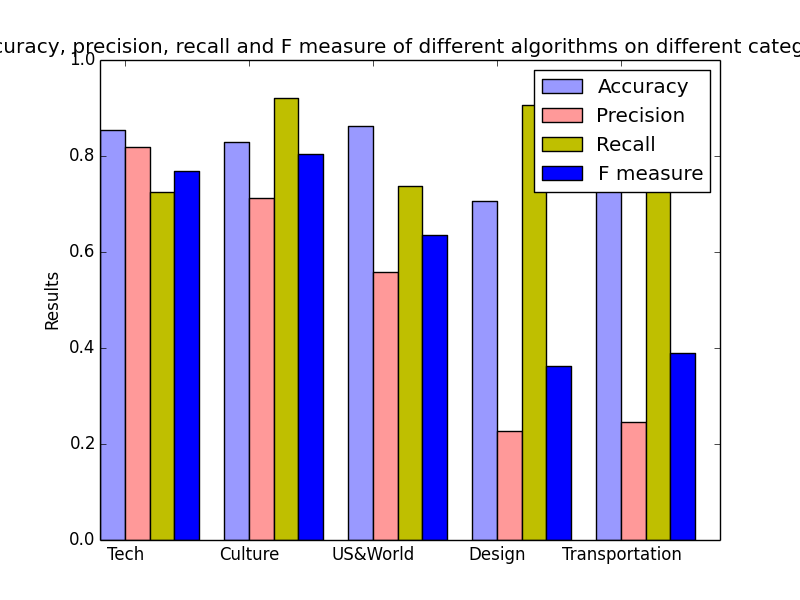
For keywords extracted from wordnet, we use the same score as the parent keyword each word is derived from, and repete the procedure. They are offered separately due to their low precision.

Chinese version: Since the majority of the data in my Evernote is written in Chinese. We borrowed a Chinese keyword extraction system called jieba[3]. It is able to run through a given text segment and recommend keywords based on the occurrence of the frequent words while also considering the co-occurence of other words in the same sentence with the frequent words and make suggestion based on that. But there is no Chinese wordnet good enough to be used. Indeed, there is such an attempt at Taiwan, but the number of words in that set is too small to be usable.

**3 Results**

Due to the time, we haven’t got time to test tag recommendation based on keyword extraction. Hence the charts shown here are all from text classificaiton. The training set is comprised of 7800 documents. The test set is composed of 800 documents. Below is the result:





One test result of our tag recommendation based on keyword extraction is as follows:

Article is “Rumors have it” from MIT news: <http://newsoffice.mit.edu/2015/correcting-political-rumors-0505>

The tags shown on the web page is: SHASS; Political science; Communications; Research; Politics

Our tag recommendation is: (with score indicating how possible this word is a tag)

[[u'research', 5430.75], [u'political science', 1328.5384615384617], [u'politics', 154.0], [u'public opinion', 16.0], [u'democracy', 9.0], [u'psychology', 9.0], [u'led', 6.0]]

Feel free to try out on our website: <http://linserv2.cims.nyu.edu:40010>

All you need to do is create a document, fill it with an article you are interested in. and see the miracle!

But notice that since the function of recommend tags based on keywords draws conclusion from MIT news, if you would like to test this feature, please upload a document talking about technology to get the best performance.

**4 Discussion**

Text classification is a field which has been extensively studied. Numerous methods exist on the web. We do not intend to beat the best benchmark result of text classification. Instead, we tried to understand better about the techniques behind. Below are some of the interesting results we find out:

1) Surprisingly, naive bayes turns out to work pretty good. Although the model is quite naive. It still captures some of the key intrinsic properties. This method simplifies problem to comparing probabilities between two classes. If the model is not simplified so much, of course it is a fairly precise algorithm. However, it ignores the dependence among the features. Most importantly, it treats several appearances and single appearance of a term completely the same way. Yet due to its direct representation of some of the features of the text handling, it is still a good choice.

2) The second thing we didn't expect is the effect of tf-idf model turns out to be not so influential. At first we thought that if a term appear multiple places in an article, then it is a good proof that the article is talking about this term. However, since tf-idf combined with svm cannot directly reflect the properties of a document. (It maps one document to a high dimensional space.) So unlike naive bayes, the classification suffers from outliers. After all, you cannot expect there is a clear boundary between the tf-idf terms of documents belonging to two different categories.

3) The third interesting result we find out is the effect of title bonus. It is very natural to imagine that since we human beings decide whether a document belongs to a certain category mainly by looking at title, machine should be able to do the same. So we test our classifiers with title bonus. The way we do it is simple, just repeat every word in one document’s title several times and add them to the main body of this document, together they form a single element of our training set.

But the result shows that this model is terrible at accuracy and recalls. Apart from that, the whole training time is also much longer than we don’t use title bonus. These facts are not so hard to explain. With a large bonus in the title words, a little noise can cause the system to be not robust. In other words, some irrelevant words are assigned great weights therefore influencing the result. The feature selection step is also forced to choose more features and makes the training process longer.

The pros of using title bonus is the great improvement in precision. This can also be explained. Some words that really describe what this document is talking about sometimes occur less often in that particular document. Yet the title tends to represent what the document is talking about, and usually uses words that directly refers to the topics it is discussing about. With this being said, we are not entirely sure why the precision is increased and recall is decreased.

**5 Conclusion**

This article discusses about text classification in a specific scenario, assigning tags to articles from **[verge.com](http://verge.com)**. We tested several ways of feature selection and training method, and got a reasonable result. Next, we tried to automatic generate tags from articles, which turned out to be a difficult task.

Some of the difficulties we encountered are discussed below. Future work can be conducted according to these.

1) The rarity of data remains to be a big problem. Categories with only a few articles are very hard to classify using traditional method. Because the rare category’s frequency is too low compared to other categories, their possibility returned by algorithm tends to be too small to be recognized. Yet in a lot of cases, many rare categories indeed exist such as terminologies or topic specific terms. The discovery and classification of these categories remain to be a hard problem.

2) The speed of the algorithm is too slow. Dealing with text is way much time consuming than we previously estimated. The average running time of a single run is between ten minutes and twenty minutes. As discussed above, the time length is also directly influenced by title bonus.

3) The connection between keywords and real tags is hard to identify. There is a huge semantic gap lies between them. For instance, if you randomly clicked into a question on “stackoverflow”, then chances are no word of tags appear in the question at all. The question is talking about a specific topic, but in order to get that, you have to first understand what this question is talking about. Yet this as the writer considers, is way beyond our current technology.

4) Another difficulty is the rarity of the dataset. Delicious used to be the leading website in terms of social tagging. But it is composed of recommended urls and their corresponding tags. To infer tags from website is obviously more difficult than from articles. To make the situation even worse, the website appears to be half abandoned. Many functions are not available anymore. Of course there remained to be some datasets of delicious crawled by researchers years ago. But these datasets are mainly targeting at a different strategy than ours, to use collaborative filtering as the way to recommend tags. What we want, is given an article, we can recommend tags based on the content of this article. Apart from this website where researchers traditionally use to recommend tags. Few other websites able to offering the data we are interested in exist. Stack overflow can be considered as one of them, but it is too technical to be used.

**6 Project implementation**

We use Python/AngularJS/Java to implement this project. We utilize Python and its NLTK package to do document classification and tag suggestion, utilize AngularJS to build user friendly front-end. User could upload or drag and drop a text file to do tag suggestion. The application would handle uploading and return classification results. In content search, we utilize our MapReduce framework (implemented as assignments in Prof. Doherty's course) to build inverted index for online information retrieve. The front-end serves as load balance and a total of 10 back-ends serves as inverted-index server.

To start the application, change working directory to `Search\_Project/final\_server' and use command `python starter.py' to start the application. The application would start smoothly on linserv2.cims.nyu.edu. The front-end port will be logged and displayed.

In our case, our application works on port 40010 of linserv2 server of NYU cims machine. Url is given: <http://linserv2.cims.nyu.edu:40010/>

7 **Acknowledgement**

We would like to deliver our thankfulness to Prof. Davis and Prof. Doherty. Prof. Davis helps a lot from the very beginning by excluding several impractical proposal ideas. Without these valuable suggestions, the whole project will be aiming at a vague destination. Afterwards, he points out several valuable key points such as the importance of keyword extraction. Before that, we were only considering tag recommendation, and didn't realize that problem can be reduced to this simple.

We would also like to thank Prof. Doherty. Part of the code that comprises the whole project is borrowed from his great work. Besides, we learned a lot from his genius talent and experience in handling practical problems.

There are a lot of things that we owe to these two great teachers. Simple words are too pale to express the lessons we learn from them. Thanks!

**8 References**

[1] McKerlich, R., Ives, C., & McGreal, R. (2013). Measuring use and creation of open educational resources in higher education. International Review of Research in Open and Distance Learning, 14(4), 90–103. **<http://doi.org/10.1002/asi>**

[2] Data, M. T. (n.d.). A survey of text classification algorithms.

[3] jieba: **<https://github.com/fxsjy/jieba>**

[4] Rendle, S., & Schmidt-Thieme, L. (2010). Pairwise interaction tensor factorization for personalized tag recommendation. Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM ’10), 81–90. **<http://doi.org/10.1145/1718487.1718498>**

[5] Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. Machine Learning-International Workshop Then Conference-, 412–420. **<http://doi.org/10.1093/bioinformatics/bth267>**

[6] Foreman, G. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification. Journal of Machine Learning Research, 3, 1289–1305.

**9 Appendix**

**Proposal:**

**We propose to build a personal knowledge base consisted of four parts:**

1. Auto tagging: Assign tags to an article.
2. Clustering: Cluster the data into hierarchy form. (Demo:http://[bonda.lti.cs.cmu.edu](http://bonda.lti.cs.cmu.edu):8002/gc.html)
3. Structure: Build the inner structure between the nodes.
4. Online Learning: So that the system can remember user’s behavior and suggest better tags and clustering results.

**We think the proposal maybe too ambitious, so we will try to accomplish as many goals listed above as possible in sequence.**

**Data Source:**

<http://www.tagora-project.eu/data/#delicious>

<http://www.markusstrohmaier.info/datasets/>

Main data source Delicious

As you can see, there are tags attached there by user.

**Technique**:

Tokenization (NLP)

Named Entity Recognition (NLP)

Classification (Machine Learning)

Inverted Index and Page Rank (Web Search)

Clustering (Web Search)

**Software Package:**

Stanford NLP tools: <http://nlp.stanford.edu/software/>

**Output of the System:**

1. For auto tagging: tags for a given article.
2. For clustering: Different clusters.
3. For structure: An interactive picture grouped by clusters. User can search by click.

**Evaluation**:

The main evaluation part is auto tagging. F-measure, accuracy, and recall of our system’s output compared to the real user tagging.

**Why this is interesting**

Information is growing in an exploding speed. We cannot possibly grab every new technology and idea that occurs. However, what we can do is that we could

1. Understand the things that don’t change. For example, there are hundreds of programming language, but almost all of them have a way to represent data structure, to deal with loops. The details can be acquired later, but it is important to know the basic so that we can decide which language to use, or how to learn something quickly.
2. Store the knowledge that we have learned in a structured, self-explanatory way. Often after we learn something, we forget it. This is like after we finished a complicated programming project without any annotation, it is hard for us to understand what the hack were we doing back at that time some time later. The structure aspect is much more interesting in the respect of research. We could not rely on computers to structure the knowledge in a graph to us, we can only use it to help. But we can use online learning to teach them how to better give us results that we are interested in.

**How am I gonna do this**

By “RE”search:

1. NYU related courses
2. Open course
3. Other students’ related projects
4. Text books
5. Wiki pedia
6. Blogs
7. Q&A forum (such as stack overflow, and Quora)

**Related areas:**

1. Clustering
2. Knowledge representation
3. Online learning
4. Reinforcement learning