Large-scale Social Tag Prediction

Lorenzo Lucido
Stanford University
SCPD - Computer Science Department
Remotely from : Sheung Wan, HONG KONG
llucido@stanford.edu

ABSTRACT

Social tagging, also known as social bookmarking or "folksonomy", is the process of annotating a text published on the web, usually on community forums, with specific keywords related to the underlying topic of the document.
In this paper, we re-visit the famous task of social tag prediction with two perspectives. First, we discuss in depth what challenges one has to face while trying to predict social tags, both in theory and practice. Then, we take a practical and -most importantly- scalable approach of the task, with the potential objective to develop a high-performance automated tagging software. We propose an efficient algorithm based on linear SVM and ensemble methods and we test it on a real machine learning competition. Finally, from a more research-oriented point of view, we analyze how deep learning applied to natural language processing can be used in order to approach the end goal of social tagging, topic discovery.

1. INTRODUCTION

The dataset used in this paper is issued from the Kaggle Facebook competition named "Keyword Extraction"[1]. The training set consists in approximately 6 millions messages provided by the well-known Stack Exchange community[2]. Each message includes a title part, a body part, and of course the associated tags. The testing set, which is an additional 2 millions untagged messages, serves as benchmark for the competition. The number of different tags in the dataset is greater than $40,000$ with names as various as *python*, *astronomy* or *stanford-nlp*. It is important to mention that while such massive dataset certainly makes data manipulation relatively more complicated, it also constitutes a very valuable asset.
The performance metric used in order to assess the predictions is the mean F_1 score (also known as micro-averaging). This measure[3], which is common in information retrieval, is essential in any classification task -such as social tag prediction- that involves imbalanced classes (which reduces most of the time to a low number of positive examples in binary classification) and has been subject to a large amount of research due to the fact that it is not directly maximizable. On balanced datasets, maximizing the accuracy is equivalent to maximizing F_1 score.
Most practitioners tend to train their algorithms on accuracy performance and use cross-validation in order to maximize the F_1 score.

The particular task of social tag prediction been approached in many ways. For example, *Heymann, Ramage & Garcia-Molina (2008)* have trained a SVM algorithm on a bag-of-words feature model, while *Budura & al. (2009)* have looked into a PageRank-like algorithm, using tags as links between documents. Our implementation can be seen as an improvement of the former, however we do consider the interesting fact that the latter model is *class-free* and therefore not limited to predict a finite number of tags.

2. LARGE-SCALE TAG PREDICTION

The first challenge in our task is to define and extract a set of features that would be directly used to train the algorithm. In our case, we adopted a similar view as *Heymann, Ramage & Garcia-Molina (2008)* and used a bag-of-words feature model. We have drawn a particular attention to the title of each message and used a technique called "parameter tying". In other words, we upweighted words appearing in the title, by a factor of 3 in our case. We also normalized the word counts by the total number of words in each forum post, in get comparable features regardless of the size of the message.
This brings us to the second challenge, which is the number of features and how they can fit in memory. The training set has more than 197 millions words which generate more than 10 millions features. While most, if not all, are dealing with this problem by using mutual information (or similar tests) to perform feature selection, we chose to keep them all. We justify our choice by the necessity to have a unique set of features (for all tags and all weak learners - as we discuss later) which allows to get a fast-running algorithm. We can realize this by, first, exploiting the sparsity of the features (much less than 0.1% of the total number of features used per training example) and second, using *feature hashing*[4].
Social tag prediction is an application of text classification that involves, as we mentioned, highly imbalanced datasets.

[1] *Lorenzo's comment* : please note that any information in this document must remain confidential in order for my participation to comply with the competition rules.
[2] `www.stackexchange.com`
[3] For more details, the reader can refer to the Mean F-score explanation page on Kaggle website.

[4] Feature hashing is the process of assigning hash values to features -words here-, potentially generating collisions, but with the noticeable advantage of avoiding the necessity to keep a large vocabulary into memory.

StackExchange forums most frequent tags					
c#	7.6%	*jquery*	5.1%	*mysql*	2.9%
java	6.8%	*c++*	3.3%	*html*	2.8%
php	6.5%	*python*	3.1%	*.net*	2.7%
javascript	6.1%	*iphone*	3.0%	*ios*	2.3%
android	5.3%	*asp.net*	2.9%	*objective-c*	2.2%

Table 1: % of positive training examples

Indeed, *Table (1)* illustrates the percentage of positive examples across the popular tags, and even the tag the most frequently used, namely *c#*, does not have more than 10% of positive examples.

It is important to note that training a machine learning algorithm with imbalanced classes is still a challenge today, the usual techniques to cope with such imbalances are based on dataset resampling, either by under-sampling (removing samples from the majority class) or over-sampling (duplicating samples from the minority class), in order to restablish balance between both classes. We, again, chose a different path by leveraging on our large labeled dataset. Instead, we are using an aggregation of classifiers with almost no regularization in order to make sure each of them outputs a very high F_1 score, even with imbalanced data.
Finally, we split the social tag prediction into two consecutive subtasks. The first one, named *popular tag classification*, is the prediction of the most popular tags with the usual bag-of-words feature model, each one of the prediction is considered independent of the others, therefore, we train as many binary classifiers as the number of tags we want to predict. In the second subtask, *individual tag extraction*, we need to predict non-popular tags (since popular ones have been handled in the first subtask), and therefore we assume that the tags to predict are likely to appear in the related message. Thus, we want to use a completely different set of features.
In order to isolate the popular tags, we sorted them by frequencies and retained the n top ones. We then studied the *maximum achievable mean F_1 score* (denoted $\hat{\mathcal{F}}_1$) as a function of n, defined more formally as :

$$\hat{\mathcal{F}}_1(n) = \frac{\mathcal{P}(n)}{\mathcal{P}(n) + \frac{\mathcal{N}(n)}{2}}$$

Where $\mathcal{P}(n)$ is the number of positive tags within the set of tags we are trying to predict for the full training set and $\mathcal{N}(n)$ is the number of positive tags not included in this set of tags (directly counted as false negatives). One can easily verify that $\hat{\mathcal{F}}_1(n)$ is equivalent to the usual micro-averaged F_1 score with perfect predictions on the first n tags and no prediction at all on the remaining ones.
While we could only focus on predicting existing tags for the purpose of the competition[5], we consider that trying to match human inputs - although being an interesting challenge - diverges slightly from the actual topic discovery purpose of social tagging. Indeed, an adequate set of tags should give useful information about the topic of the message but also avoid useless and misleading information. While we, as humans, usually tag words which are relevant, we tend to make two mistakes. First, we omit very important tags, as

[5]Top competitors have achieved a mean F_1 score as high as 80% only by predicting existing tags

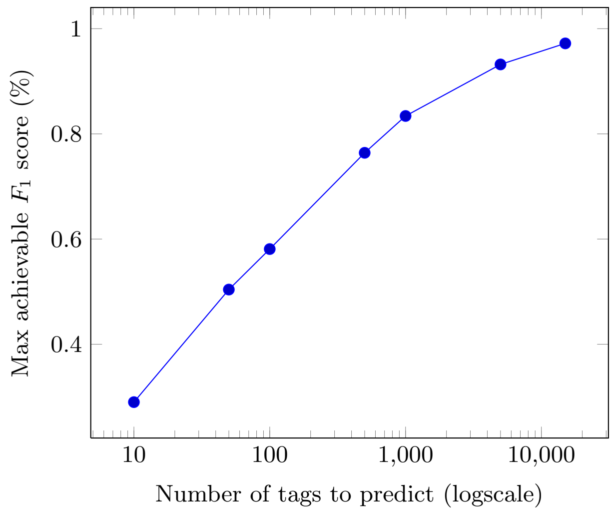

Figure 1: max F_1 vs number of tags

an example, we have in our dataset a message tagged with *boost*, which is a famous C++ library, but not *c++* (see the message extract). Second, we often do not consider word-sense disambiguations as shown by the popularity of the tag *post* (used in more than 13,000 messages) which has many different meanings.
This argument favors our two-step approach. In the first one, we use the bag-of-words features and tag labels to get a first set of tags for each message. Then, we use this representation and a different set of features to construct our final set of tags.
Also, we believe that setting $n = 500$ is a good trade-off between performance and efficiency, although it is for sure up to discussion and might need to be optimized at a later stage.

Message Extract :

Compiling with Boost C++
I'm using Boost for the first time and having a problem compiling my program. So far all I have done is include the tokenizer.hpp header file and it's failing with a load of errors. I'm using using g++ 4.4.7, Boost 1.54 on CentoS 6.4 32Bit. Here is the output: (...)

Tags : linux, boost, gcc4.7

3. POPULAR TAG CLASSIFICATION

We now focus on our first subtask in more details. Let us denote m_{test} the number of messages for which we are trying to predict the tags, and $T = \{t_1, t_2, \ldots, t_n\}$ the set of n most popular tags to be predicted. We then try to build the m_{test}-by-n matrix $\hat{Y}$ which values are either 0 (predicting absence of tag) or 1 (presence of tag). This matrix is, again, sparse since the average number of tags per message is about 3.

Our first approach was to build a simple Naive Bayes model in order to get a first idea of what percentage out of the *max achievable F_1 score* is reasonable. Using only the first 100 tags and cross-validation to set the Laplace smoothing parameter, we managed to get an overall performance of 29.8%

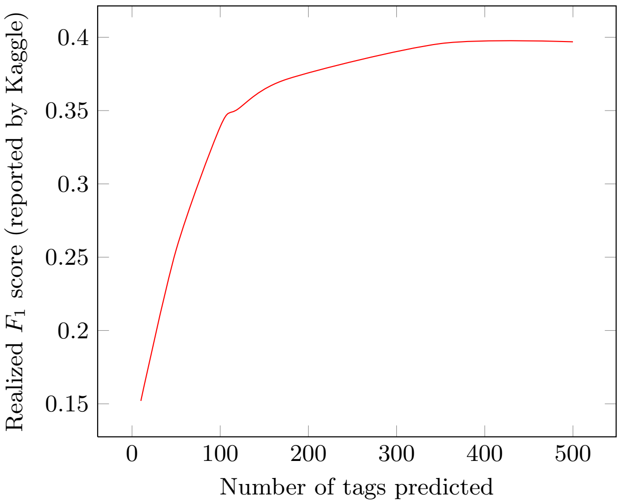

Figure 2: Learning curve

out of 58.1%. This result shows that a discriminative model -such as SVM-, should be reasonably able to get more than 50% of *max achievable F_1 score* even when training substantially more classes, given the large size of the training set. It is also important to note that Naive Bayes has the advantage of being gradually parameterized by using subsets of the training set that can fit into memory. In our implementation, each subset S_i is represented as a sparse m_S-by-N matrix, m_S being the size of each subset (we are using 200,000 messages by subset) and N the number of features[6].

Our proposed algorithm, is the combination of Linear SVM and *bagging*. Bagging, or bootstrap aggregating, is an ensemble algorithm usually employed to build a set of classifiers by randomly sampling a subset of the training set. It has a lot in common with k-fold cross validation, except that in cross-validation, we try to estimate of the generalization error, while in bagging, we average the predictions[7] across all the classifiers -also called *weak learners*- in order to obtain a more robust model overall. Each of our weak learner $\mathcal{C}_{jk}$ is a linear SVM model trained to predict a tag $t_k \in T$ on a subset S_j of the training set. Inspired by the *Wisdom of the crowd* and again taking advantage of abundant labeled data, each subset S_j is disjoint, therefore guaranteeing the diversity of the classifiers. The regularization parameter of each Linear SVM is set to an extremely low level in order to guarantee a high F_1 on the training set even with unbalanced data. We then perform a majority vote.
Our resulting implementation has several strengths :

- easy to implement
- highly parallelizable : horizontally (tags) and vertically (disjoint subsets)
- low memory consumption (adjustable with the size of the subsets)
- on-the-go learning curve as the population of weak learners increases

Using our algorithm to predict the first 500 most popular tags and 29 weak-learner for each tag, we managed to reach almost 40% on the mean F_1 score for the competition with the computing power of a single laptop[8]. We report the learning curve of our algorithm as a function of n, the number of predicted tags in *Fig. 2*, the number of weak learners (29) corresponds to the full training set, excluding the cross-validation set. Also, *Algorithm 1* gives a pseudocode of our implementation :

Data: Training set : Bag-of-Words features X, tags Y
Result: Tag predictions on Test set $\hat{X}$
Split X and Y into M subsets $S_j = [X_j, Y_j]$
for k = *1 To number of tags to predict (n)* **do**
  y = k-th column of Y_j
  for j = *1 To number of subsets (M)* **do**
    train linear SVM classifier $\mathcal{C}_{jk}(X_j, y)$
    predict on Test set $\mathcal{C}_{jk}(\hat{X})$
  end
  majority vote across predictions
end

Algorithm 1: SVM - Bagging for popular tag prediction

Both loops are parallelizable in the above algorithm.

4. INDIVIDUAL TAG EXTRACTION

Our second task is now to improve the results previously obtained by looking to extract the words within the message which are relevant to the topic of the message. Indeed, having predicted the tags mostly used, we consider that this new representation gives a first idea of the message content. However, it is only based on human inputs, which, as discussed above, are relevant for the Kaggle competition, but not ideal for summarizing the topic of each message.
In this particular step, we are looking to enhance the set of tags by adding relevant words. These words must be either in the body or in the title of the message in order to be selected as tags. Therefore, while still not exactly "summarizing" the content using new words, this approach has the advantage of choosing words which have potentially never been tagged before but still gives useful information.

We introduce a modified version of the TF-IDF statistic, called TF-ICF (Term frequency - Inverse class frequencies) and defined by :

$$\log \mathcal{T}(t,d) = n \log\left(\frac{t_d}{|d|}\right) \sum_{i=1}^{n} \log \frac{|C^{(i)}|}{t_{C^{(i)}}}$$

Where t_d is the frequency of the term t in document d, $|d|$ is the total number of terms within the document d, $t_{C^{(i)}}$ is the number of documents of class i where the term t appears, $|C^{(i)}|$ is the frequency of the class i with the corpus of documents. The classes here correspond to the tags predicted in the previous step.
This statistic is designed to be, for each class, increasing with the probability of the term t within the document d

[6]In practice, N is the maximum number of hash values resulting from feature hashing - 15 millions here.
[7]In classification, each weak learner "votes" to either predict a positive or negative label.
[8]For the latest rankings, please check the Kaggle Facebook rankings

and decreasing with the probability of the term t within the class i. It is closely linked to the Multinomial Naive Bayes log-likelihood, due to the assumption that the classes are independent.

The TF-ICF statistic allows us to rank all the term in all the documents relatively to the additional information they bring on top of the general idea of the topic predicted in the first step. That is, for example, knowing that the message seems to be about *java* and *eclipse* but not *linux*, is the word *void* relevant to the overall topic of the message ? Is the word *mahout* relevant ?

The tags predicted in the previous step are here very important, because for a new untagged message, we need to provide our algorithm with a basic set of tags. Also, the computation of TF-ICF can be implemented very efficiently using some optimized vector operation library and is much simpler than a PageRank-like algorithm.
The only parameter in this second step is in fact the unique threshold used to select words that should be flagged as tags. One way would be to set it in order to maximize the mean F_1 score on a cross-validation set, but since we consider that human inputs might not be ideal, supervised learning should not be part of this step. We suggest instead to target a *mean number of tags* around 5. Similarly, it is also fairly difficult to assess the quality of topic discovery.

Another choice for the second step could have been to consider building a totally new set of tags, i.e. considering the predicted tags from SVM - Bagging only as classes. These classes would then be used to compute a slightly different TD-IDF-like statistic that emphasizes on inter-class relevance (rather than intra-class).
An actual implementation with examples of the second step will be subject of further work.

5. DISCUSSION

Being able to automatically summarize a message, either by selecting words relevant to the topic, or building short sentences, is still today one of the most difficult task in Machine Learning, and more specifically Natural Language Processing due to several reasons, namely :

- the quality of the output is very subjective and hard to assess quantitatively
- due to unreliable human labelling, supervised learning is not sufficient
- the features used can be considered as very noisy, since several people can write the same idea in different ways, and sometimes mispelling words

Kaggle is definitely the kind of initiative that can really help the development of Machine Learning. It is indeed very interesting to compete against the best practitioners, in a result-driven challenge. As for our participation in the Facebook competition, we have carefully looked into explaining the top score of 80%, while our actual score is about 40%. The top scorer announced that he has achieved it using only popular tag prediction (first step of our method), it means

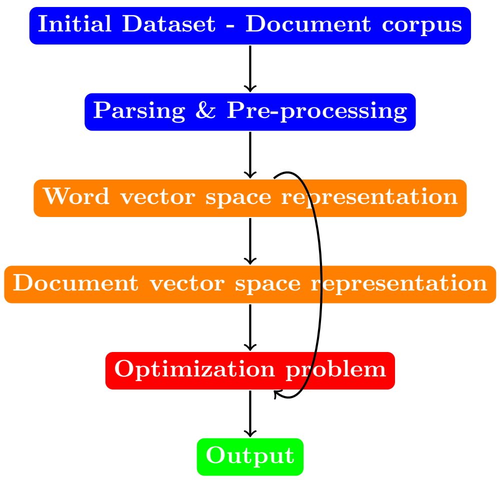

Figure 3: Deep Learning pipeline for topic discovery

that his solution lies between predicting perfectly 500 tags and predicting 45000 tags with a weighted 80% F_1. We think the score difference, and in the meantime, the key ideas in order to improve our score, can be explained by :

- a significantly higher number of predicted tags
  => better predictions
- a better parsing script with manual inputs related to main programming languages (should show significant improvement, due to the tag popularity of such languages)
  => better features
- a faster implementation in a programming language suitable for large datasets, allowing for example ensemble methods algorithms to achieve a lower generalization error
  => better training algorithm

We have then highlighted the fact that trying to match human inputs may not be the end goal of social tag prediction. Indeed, we consider the task successful when the tags associated to a text are giving a "good" overview of its content[9]. Individual tag extraction, through TF-ICF statistic, presents a way to select words eligible to be tags within the associated text.

Finally, we open the discussion on how deep learning could be applied for social tag prediction, and especially how the previous work from *Socher, Pennington, Huang, Ng & Manning (2011)* connects to this specific task. Deep learning, although very demanding in terms of computing power[10], seems to be the next step towards better social tagging.

[9]Additionally, we could say that proper tags are to be reused across the corpus, and therefore must be both words relevant to the topic and popular within the community.
[10]Note : it did not seem to be wise training a sparse autoencoder on a 6 millions documents dataset with a single laptop, which is why we looked into more computationally efficient algorithms, but the initial subject was "Deep learning for social tag prediction"

More specifically, we can train a deep learning algorithm to map words and documents onto the same vector space, in which a large Euclidean distance between two words or documents translates to very different meanings, and inversely a close distance into similar meanings. Then, after a clustering step that will help us define the tags, i.e. words with close meanings are grouped under the same tag, the social tagging, or more generally, topic discovery task reduces -informally- to the constrained optimization problem:

- *Minimize the number of tags*
- *s.t.* $||$*Sum of tags - target document*$|| <$ *Meaning dist*

Where $||.||$ is the L_2-norm and "Meaning dist" is a constant that defines how close to the actual meaning of the document the tag summarization should be. That is, we want to restrict the meaning of the selected tags to be within a "meaning range", which, in the word vector space, is simply the Euclidean distance. This pipeline is fully unsupervised, in order to be consistent with the fact that human social tagging is not necessarily relevant. An overview of the pipeline is given in *Fig. 3*.
An extensive study about this application of deep learning will be the subject of a further work.

6. REFERENCES

[1] P. Heymann, D. Ramage and H. Garcia-Molina, *Social Tag Prediction* [2008]

[2] A. Burdura, S. Michel, P. Cudre-Mauroux and K. Aberer, *Neighborhood-based Tag Prediction* [2009]

[3] D. Yin, Z, Xue, L. Hong and B. Davison, *A Probabilistic Model for Personalized Tag Prediction* [2010]

[4] A. Hotho, R. Jaschke, C. Schmitz and G. Stumme *Information Retrieval in Folksonomies: Search and Ranking* [2006]

[5] A. Ng, *CS229 Lecture Notes* [2012]

[6] C. Manning, P. Raghavan and H. Schutze, *Introduction to Information Retrieval* [2008]

[7] S. Bird, E. Loper and E. Klein, *Natural Language Processing with Python* [2009]

[8] A. Rajaraman and J. Ullman, *Mining of Massive Datasets* [2011]

[9] M. Byrne, *Predicting Tags for StackOverflow Posts* [2013]

[10] H. Guan, J. Zhou and M. Guo, *A Class-Feature-Centroid Classifier for Text Categorization* [2009]

[11] R. Socher, J. Pennington, E. Huang, A. Ng and C. Manning, *Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions* [2011]

[12] E. Bauer and R. Kohavi, *An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants* [1998]