Classifying Wikipedia People Into Occupations

CS229 Final Report

Aleksandar Gabrovski (sasho@stanford.edu)

Introduction

Wikipedia articles are classified by collaborative authors in categories. Each category generally covers a single aspect of an article - for example common categories are “person”, “woman”, “1989 births”, etc. Given the crowd-sourced nature of Wikipedia, it is not surprising that the categorization of articles is sometimes incomplete, if not outright misleading.

Wikipedia categories form hierarchical graphs. For example, an article could be marked as in the category Category:Russian_female_boxers, which implies that the article is about a boxer, a woman and a Russian - each of which is a separate category. Indeed, Category:Russian_female_boxers is a subcategory with three parent categories. This is challenging since any automated process on top of an article’s categories needs to reason about the specific category assigned to the article, as well as all category parents of that category (each of which can have its own multiple parents).

Wikipedia categories are extremely useful for automatically generating lists of articles that share a common trait. A case in point is occupations: if we were interested in finding out more about Italian architects, we could go to Category:Italian_architects and discover a very rich listing of architects in Italy over the ages. Unfortunately, if we were just interested in architects, we would have to go to Category:Architects which contains the misleading category Category:Wikipedia_categories_named_after_architects - a subcategory about inanimate objects which are clearly not architects!

The following paper explores the possibility of auto-tagging Wikipedia articles with some categorical metadata - in this case the focus is on occupational categories. An accurate occupational classifier could ensure that all Wikipedia person articles are appropriately tagged with the profession of the person in question, making her discoverable from automatically generated lists by category. We present a classifier on top of a limited set of occupations - 6 total. We achieve 90% accuracy with our final dataset of 2000 articles per label.

Dataset

To get reliable labels on occupations, we use the automatically generated page Lists_of_people_by_occupation[1] as a seed to a recursive web scraper. In order to avoid blacklisting the author’s IP address from Wikipedia, we use Wikimedia’s pywyikibot[2] for downloading Wikipedia pages with automatic throttling of traffic.

Starting at the seed page, we proceed to download all links on that page, and then recursively download all of their links up to a total depth of 3 hops. As illustrated with the Category:architects example, this inevitably leads us to download many pages that are about non-persons. To detect which article is in fact about a person, we rely on the existence of a category of the type “\d\d\d\d births”, where the “\d” represents a single digit in base 10.

In order to avoid the complications of reasoning about a category’s parents, we use a simple heuristic: we assume that a link on the seed page Lists_of_people_by_occupation is an actual occupational label. For example, if

[1] https://en.wikipedia.org/wiki/Lists_of_people_by_occupation
[2] http://www.mediawiki.org/wiki/Manual:Pywikibot

Lists_of_people_by_occupation has a link called “architects” all articles downloaded from “architects” are marked as occupation “architects”.

The first overnight run of the recursive scraper yielded 123 labels, each label having between 30 to more than two thousand person articles. This dataset proved too big for the author’s laptop to handle. So for the majority of this paper we used only 6 occupations with roughly 2000 articles each. These six occupations are: architect, engineer, scientist, writer, actor and musician. We briefly explore adding more labels at the end of the report.

Features And Preprocessing

Each Wikipedia article contains plain text, links and categories metadata. We start by stripping out all categories from an article to avoid including the label we are trying to classify in the text of the article. Then, we proceed to extract all links from an article. Finally, we apply a standard stemmer[3] to all plain text words in the article.

Words

The plain text words contain a lot of stop words such as “a”, “as”, “the”. The author considered several approaches to removing the stop words. First, a basic heuristic was used where top N words were removed from the plain text of each article. Unfortunately, the accuracy of the trained models seemed to depend heavily on the value of N if the number of articles per label was less than 200.

So, the author experimented with using TF*IDF [4]for automatically weighing each word according to its relative importance. The original TF*IDF algorithm involves computing the equation:

$$tf_i^j \times log \frac{N}{df_i}$$

where tf_i^j is the frequency of word i in article j, N is the number of articles in the entire corpus and df_i is the frequency of word i in the entire corpus.

In practice, the second term of the above equation led to numbers less than 10^{-5} involving severe loss of precision. In fact, using the TF*IDF weighting led to lower accuracy than running the same models with all stop words included.

After some experimentation, the author modified TF*IDF as follows, leading to much bigger weights for the words and reduced precision error:

$$tf_i^j \div log\ df_i$$

A comparison table of not removing stop words and using the modified TF*IDF approach with a 1-vs-1 multi-class SVM is presented below:

Results for 6 occupations	Accuracy for each dataset		
Model	100 articles/label	400 articles/label	1000 articles/label
1-vs-1 SVM	66.28%	77.20%	80.11%
1-vs-1 SVM modified TF*IDF	74.86%	79.02%	80.78%

[3] https://pypi.python.org/pypi/stemming/1.0

[4] Manning, C. D.; Raghavan, P.; Schutze, H. (2008). "Introduction to Information Retrieval". p. 100.

Notably, as the dataset grows the removal of the stop-words from the text seems to matter less and less. This makes intuitive sense - the more data the model has to train on, the more it is able to figure out that stop-words are not a useful signal.

Using the above preprocessing, we generate a feature mapping from plain text words to sparse vectors of modified TF*IDF scores for each word occurring in an article.

Links

The second raw feature we use are the links a Wikipedia article contains. We borrow the concept of the random surfer from PageRank[5] - intuitively, articles on the same occupation likely point to similar articles. A random surfer on an article about a boxer is likely to visit other articles about boxers. If we can approximate the probability of the random surfer visiting a page tagged with a given label, we would have a useful mapping.

Following this logic, we introduce the following feature mapping: all links from articles in our training set with label L are tagged with label L. A link can be tagged with more than one label. For each article we introduce the mapping s_L^j - the number of links tagged with label L that article j contains. We generate a vector with these weights for each known label and append it to the word vector described in the previous section. When we get a new article we haven't seen before we compute the s_L^j weights for it by seeing how many of its links were tagged in the training set and append the weights to its word vector.

A comparison table of the feature mapping without the links and with the links is shown below:

Results for 6 occupations	Accuracy for each dataset		
Model	100 articles/label	400 articles/label	1000 articles/label
1-vs-1 SVM modified TF*IDF	74.86%	79.02%	80.78%
1-vs-1 SVM modified TF*IDF + links	83.43%	87.41%	87.91%

Clearly the links provide a very good indicator for an article's label.

Models

We train multi-class SVM and logistic regression on the above dataset. In each case we use both 1-vs-1 and 1-vs-all (aka 1-vs-rest)[6] multi-class classification. We extend appropriately the logistic regression provided by liblinear[7] to 1-vs-all classification, where the model with the highest probability label for a test article is selected as the label selector. Similarly, we extend libsvm[8] for the SVM implementation of 1-vs-all classification, again selecting the model with the highest confidence (biggest margin) to determine the label.

---

[5] Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999) The PageRank Citation Ranking: Bringing Order to the Web. Technical Report. Stanford InfoLab.

[6] K. Crammer and Y. Singer. On the algorithmic implementation of multi-class kernel-based vector machines. Machine Learning Research, 2:265–292, 2001.

[7] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification, Journal of Machine Learning Research 9(2008), 1871-1874. Software available at http://www.csie.ntu.edu.tw/~cjlin/liblinear

[8] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

For SVM, we determine the value of the regularization parameter C using simple model selection on the dataset with size of 400 articles per label. We test the values 0.001, 0.1, 1, 10, 50, 100, 200, 500, 800, 1000, 10000 for C. The value of 800 renders the best accuracy for 1-vs-1 SVM. For 1-vs-all we choose the value 50.

We train the above models on datasets of sizes 100, 400 and 1000 articles per label using simple cross-validation with a 70-30 split of the data.

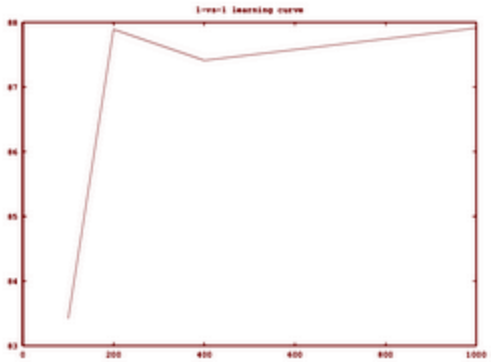
Results

Following is a table that compares the above described models and features for the previously listed 6 labels:

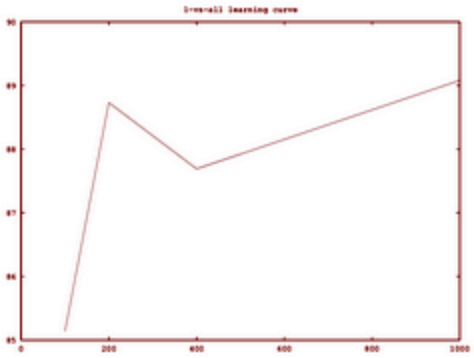
Results for 6 occupations	Accuracy for each dataset		
Model	100 articles/label	400 articles/label	1000 articles/label
1-vs-1 SVM modified TF*IDF + links	83.43%	87.41%	87.91%
1-vs-all SVM modified TF*IDF + links	85.14%	87.69%	89.08%
1-vs-1 Log reg mod TF*IDF + links	87.43%	87.69%	87.85%
1-vs-all Log reg mod TF*IDF + links	87.43%	87.69%	87.85%

The learning curves for the above models are:

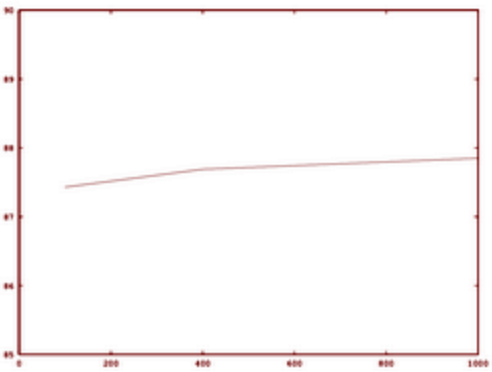
1-vs-1 SVM modified TF*IDF + links



1-vs-all SVM modified TF*IDF + links



Log reg mod TF*IDF + links

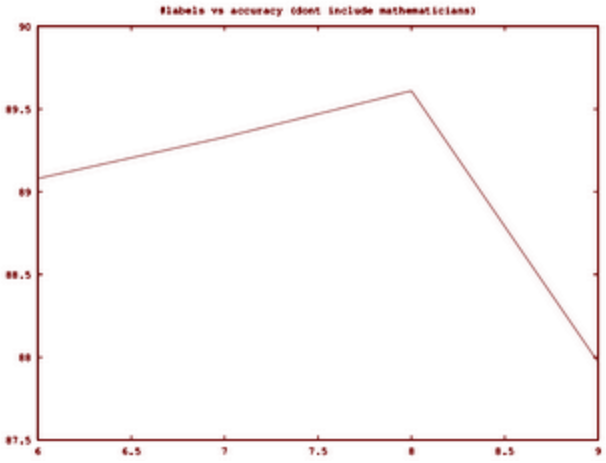


Discussion

Interestingly, the 1-vs-all extension of logistic regression had no impact on the accuracy of the model, despite the increase in the dataset each binary model is running on in 1-vs-all setting. A likely explanation for this is that the linear separator for logistic regression does not have the same flexibility to adapt to the data as SVM. This view is further supported by the fact that exponential increases in the dataset size barely affect the performance of logistic regression.

From the learning curves it looks like the 1-vs-all SVM has the steepest learning slope, suggesting that this model has the best odds of improving with an increase in the dataset. Running the same model against a dataset with 2000 articles per label renders accuracy of **90.32%** suggesting the further improvements might indeed be possible with that model.

Finally, we address some of the concerns regarding the restriction of our labels to 6 occupations only. We proceed to gradually increase the labels to 7, 8 and 9 (adding biologists, chemists and mathematicians respectively), testing the 1-vs-all SVM on these models with datasets of 400 articles per label. To the left we've plotted accuracy relative to the number of labels. Note that the y-axis ranges from 87.5% to 90%, i.e. we do not see a dramatic change in the accuracy. This makes the author hopeful that the SVM 1-vs-all model can scale OK with the number of labels.



Interestingly, the accuracy only drops once we include mathematicians as an occupation. This is likely due to the fact that mathematicians, engineers, scientists and architects all do math, potentially leading to similar language used across these articles.

Conclusions

We managed to reach 90% accuracy with the 1-vs-all SVM model against our dataset of 2000 articles per label. We also showed that the accuracy of the model does not vary too much with the number of labels used. Both of these results suggest that productionizing this model to actual Wikipedia could lead to very favourable results. Unfortunately, the 1-vs-all SVM model takes quite a while to train and performance is heavily dependent on having many articles per label. It seems that if we are willing to slightly reduce our accuracy to 87.85% for a big boost in performance, logistic regression is the a better model to use at scale. Such tradeoffs are of course determined by the business need and infrastructure limitations of the actual product.

Future

The above analysis shows promising results for classifying new and existing Wikipedia articles on people into occupations with up to 9 labels. A natural extension would be to expand the number of labels to something more realistic and see how that performs. Furthermore, the current models only focus on the English Wikipedia - it would be great to see how this scales across languages. Finally, for such a project to succeed in the field, serious thought needs to be given to scaling our models out to millions of articles with a massive feature mapping for each article.

References

1. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification, Journal of Machine Learning Research 9(2008), 1871-1874. Software available at http://www.csie.ntu.edu.tw/~cjlin/liblinear
2. Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
3. Manning, C. D.; Raghavan, P.; Schutze, H. (2008). "Introduction to Information Retrieval". p. 100.
4. Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999) The PageRank Citation Ranking: Bringing Order to the Web. Technical Report. Stanford InfoLab.
5. K. Crammer and Y. Singer. On the algorithmic implementation of multi-class kernel-based vector machines. Machine Learning Research, 2:265–292, 2001.
6. Stijn van Dongen, Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, May 2000