Paper-Author Identification Based on Topic Modeling with Word Vectors

Team: Five Idiots & One Genius

Members:

Hanlin Zhu	*604738818*	*zhuxx544@ucla.edu*
Yuling Liu	*504777221*	*lylwill@ucla.edu*
Zeyu Li	*704759914*	*lizeyu_cs@foxmail.com*
Chen Shao	*704760776*	*shaoc2@g.ucla.edu*
Kaiyi Wu	*004761345*	*kaiyiwu@ucla.edu*
Ke Xu	*604761427*	*xuke931226@ucla.edu*

Abstract

Paper-author identification problem was a challenge given at KDD 2013 competition. Most of the competition teams proposed their models based on a heavy feature-engineering process. In recent years, deep neural networks have proven its power in feature learning given a specific task. Therefore, in this project, we aim to use deep learning techniques both to save the feature engineering labor work and to find better latent features for paper-author data. Specifically, due to limited time we have for this project, we only considered natural language data from the given dataset and designed our model based on word vectors learned from deep neural networks. In this paper, we started with a brief review of the paper-author identification problem. Then, we provided our implementation and evaluation of the project. After the evaluation section, we analyzed our result and pointed out future directions for improvement. A conclusion of this project is made at the end of this paper.

1. Introduction

The paper-author identification problem was proposed for the following scenario. Given a pair of an author and a list of papers, the task is to rank those papers in the order from the most likely written by the author to the least likely written by the author. It is a common problem we encounter when we try to search papers or authors on Google or other search engines, since they need to decide which paper should appear before other papers.

This problem was the challenge given at KDD 2013 competition for $2,500 to the top winner. By studying the works of those competition teams, we found that almost all teams proposed their model based on a heavy feature-engineering process. Such a process is usually tedious and requires a good instinct of picking the right feature for the given task. For example, the first place winner from National Taiwan University engineered 39 features manually from raw dataset before they apply their ranking model.

Thus, based on the previous observation, in this project, we are interested to see whether the feature learning technique from the deep learning community could help relieve the burden of feature engineering while at the same time provide a reasonable performance in terms of benchmark. Although we want to avoid feature engineering, it is inevitable to still consider features of different types. For example, natural language based features are different from coauthor based features. Hence, when we say feature learning, we refer to learn latent features of a specific type of data. Due to the limited time we have for this project, we only considered natural language data from the data set, such as titles, keywords, etc. Specifically, we aim to use word vectors learned from deep networks to build a score function for ranking papers in the validation set. The detailed implementation of our model is given in the following section.

2. Implementation

We attempted three different ways to compose a score function based on word vectors. In the rest of this paper, we will call them topic score, average score, and max score respectively. One important note is that although we propose the topic score model in this section, it is not included in the evaluation section for the following reason. Firstly, we tested this model to see the final score of a confirmed paper and a deleted paper. As a result, this model returns almost the same number for both papers, and therefore is considered not helpful for this task. We include it in this section purely for the purposes of completeness of our project progress.

The idea of our implementation is fairly simple. Given an author name and a set of papers, we designed a score function and use it to score each (author, paper) pair. The score function will return a high value if it is highly likely the paper is actually written by the author, and vise versa. Based on the score of each pair, we rank the whole paper set from high score to low score, and evaluate the mean average precision of this ranking.

In the following three subsections, we provide the formulation of our three score functions in the order mentioned before.

2.1 Topic Score Model

In this topic score model, we aim to treat an author as a word, and the same is true for each paper. This is done by encoding the id number of an author or a paper into a alphabetical sequence. The coding process is done by mapping a numerical value into a letter by ASCII code. The coding between author id and paper id are shifted by an offset of 9, so that there is no overlapping between author code and paper code. We will refer to those encoded id as code words in the rest of this paper.

Then, we search from the raw database to find context words for each code word. The context of a code word is defined as the words that describe the content of a set of papers. For an author code word, its context would be all the titles and keywords of the papers this author confirmed as publications. For a paper code word, its context would just be its title and keywords. We represent the context of a code word as a bag-of-word model, in which each paper title is tokenized into words and added to a set of words.

As a result, we obtain a training data in the format of (code word, context word) pairs. We feed this data into a feed-forward neural network, with a hidden layer of dimension 2048. After training, we extract the vector representation of each code word from the neural network, and treat it as a topic of the code word.

The intuition behind this model is that an author tend to publish new papers in similar or the same field of his past study. Thus, we use a vector to represent an author’s past work, and use a vector to represent a paper’s content. Both vectors are trained based on the titles and keywords of paper contents. The score function is defined as the dot product of an author vector and a paper vector to measure the similarity of an author’s past publication and this new paper in terms of the topic they represent.

However, after testing several examples, we found that this model couldn't provide informative result since the raw dataset is too sparse in the sense that lots of papers does not have keywords in them, so the cardinality of each bag-of-word is small for training a good vector embedding model. As a result, we turned our attention to word vectors that have been pre-trained on large corpus, and this leads to the following two score functions.

2.2 Average Score Model

In this average score model, we used pre-trained word vectors from Google, which can be downloaded from the URL in the read me file. Instead of modeling each code word as a dense vector to represent a topic, in this model we combine bag-of-words and pre-trained word vectors to compute a score that measures the content similarity between two bag-of-words.

Specifically, for each (author, paper) pair, we extract the bag-of-words representation of their corresponding contexts as what we did in the topic model. In other words, the bag-of-words for an author would be tokenized titles and keywords of his confirmed publication, and the bag-of-words for a paper would be its title and keywords.

Then, a score of this pair is computed by the following formula:

$$\text{score(author,paper)} = (1/\ \#\text{pairs}) \sum \sum \text{<author\_vec>*<paper\_vec>}$$

The idea is to find the inner product between each vector pairs in two bag-of-words, and then average them to get a score for measuring the content similarity between two bag-of-words.

2.3 Max Score Model

Similar to the idea in the average score model, the only difference between max score model and average score model is how they compute the final score from all possible vector products. In average score model, we computed a final score by averaging over all vector inner products, while in max score model, we compute a final score by picking the maximal value among those inner products.

Formally speaking, the score function can be expressed as the following:

$$\text{score(author,paper)} = \max(\text{<author\_vec>*<paper\_vec>}), \text{ for all possible pairs}$$

3. Evaluation

In evaluation, we mainly talk about the MAP metric, present our result and then state the whole process of our implementation.

The evaluation criterion in this project is mean average precision (MAP), which is commonly used for ranking problems and also the direct result to evaluate the whole classification algorithm.

An author profile contains an author and a set of papers that may or may not have been written by that author. The goal is to rank the papers so that instances that the papers written by the author come before the other instances. This ranking is evaluated using average precision. This means that we calculate the precision at each point when an instance that paper written by the author occurs in the ranking, and then take the average of those values. Say an author profile contains 3 papers P1, P2, P3. Papers P1, P3 are written by the author, while P2 not. Say that the algorithm returns the ranked list P1 P2 P3 then the AP is (1/1+2/3)/2 = 5/6. And the average of these average precisions over all author profiles is calculated as the final result.

For our MAP score, we list them as the table 1 below.

table 1. final scores

Model	Co-author based benchmark	Random benchmark	Average score model	Max score model
score	0.866956028603	0.689617536816	0.687049534869	0.689456459048

It can be seen that for our 2 proposed models, the max score model works a little better than the average score model. The two benchmarks are provided by the competition website and are regarded as the baseline of the training. We simply import these two sets of trained data and compute their MAP scores respectively to compare them to our results.

To implement our two score functions proposed in the implementation section, which are average score and max score function, we need to firstly preprocess the raw data. We import “Author.csv” and “Paper.csv” which contain all the basic information of each author and paper in our dataset, transfer their IDs respectively as stated in the implementation section and do some pruning such as deleting the prepositions in titles of papers to have all the words in titles containing useful information and generate the bag-of-words mentioned in the implementation section. Then we import “Train.csv” to add context from training data and generate each tuple of author and paper that contains the topic and code word and finally output them as paper-topic-pairs and author-topic pairs for further score computation.

To combine bag-of-words and pre-trained word vectors from Google to compute the score, we implement the data which includes word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a Google News dataset. The vector length is 300 features. As an interface to word2vec, we simply go with a Python package called gensim that appears to be a popular NLP package.

In score computing process, we firstly import the “PaperAuthor.csv” for acquiring the actual corresponding information of each paper and its authors as well as generating several features, then we import the author-topic pairs and paper-topic pairs to be trained utilizing feed-forward neural network to get the corresponding word vector for each code word. Before implementing two score functions respectively, what we need to do is load the pre-trained word vector from Google to combine it with our own.

For average score, we get the inner product between each vector pairs in two bag-of-words, and then average them to get a score for measuring the content similarity between two bag-of-words. For max score, we directly regard the maximal one of the inner products as the score. Then generate the score-data matrix in which each data point means the similarity score of a certain author-paper pair. Finally, based on this score-data matrix and the ground-data matrix which only contains “1”s and “-1”s that represent whether actually a certain paper belongs to an author (“1”) or not (“-1”), we implement the MAP metric to get the final result of our two score algorithms listed in table 1.

4. Discussion

As we observed from the evaluation section, although word vector has shown its power in many natural language processing tasks, it did not provide us a satisfying performance in this task. In this section, we aim to discuss our model and result from three perspectives. The goal is to analyze the reason why word vector fails to give a satisfying performance in this project.

4.1 Raw Dataset Perspective

By introspecting the raw dataset, we concluded that the given dataset is not a language rich dataset for two reasons. Firstly, there are only two type of natural language related information we could extract for representing the content of a paper. They are keywords and titles. Also, if we exclude functional words, such as prepositions, articles, and so on, the number of informative words we could extract from a paper title is very limited. Usually, we could not have more than ten informative words from a paper title. Another point is that most of the papers do not have keywords in the raw dataset. This is the key observation that leads us to conclude the above conclusion.

One key question is how does language-limited dataset could affect our model so substantially in this project. To address this question, we will analyze three of our models separately in the remaining of this section.

For the topic model we proposed, language-limited dataset would not be a good support for a neural network model, which, we believe, is the main reason why this model could not even be considered as a good feature. For a neural network model to perform accurately in a specific task, one requirement is data rich. Since neural network is a data-driven method for discriminative tasks, the lack of language-rich data directly implies the failure of our topic model for learning vector representation of authors or papers.

Both average score model and max score model use a pre-trained word vector to relieve the issue caused by language-limited dataset. As a result, this modification indeed boost the performance to the basic baseline. However, it does not provide us more information other than a random shuffle could provide. Since both models compute their score based on the words in bag-of-words extracted from raw dataset, and the raw dataset is very limited in its ability to provide informative words to represent paper content, we believe the reason why pre-trained word vectors do not further boost model performance is also affected by the language-limitedness of the raw dataset.

4.2 Task Nature Perspective

Besides the point of view from raw dataset, we analyzed our result in terms of the nature of our task. The question to be answered is whether language-related features could provide significant information in terms of paper-author identification. We analyzed this question in the following two main points.

We first considered the benchmark given in the competition. The benchmark that gives a 0.86 mean average precision is a model based on random forest and co-author related features. Also, based on the paper from the first place winner, the most significant feature in this task which could potentially boost mean average precision up to 0.98 is a co-author based feature. This two observations lead us to think that co-author based features might be a more useful feature for this task.

On the other hand, intuition tells us that language related features should be an important measure in this task. Following this intuition, we checked with the paper from the first place winner, and found that language related features indeed affect the final result, but not as crucial as co-author related features. Our conclusion on this point is that the significance of language features might be reduced by the language-limited dataset.

4.3 Model Design Perspective

As a final perspective in this section, we attempt to interpret our result in terms of model design. In this part of the section, we start to review what word vector is really powerful at, and attempt to account the question whether it is our model that did not express the full power of word vectors.

Word vectors are dense representations that encodes the co-occurrence statistics in natural language corpus. It is very useful in comparing word similarity based on inner product in an induced vector space. Hence, it should provide a good measure of content similarity between papers. The reason why it did not do so in both our case and the case of first place winner is the preprocessing step.

In our case, average model and max model give about the same performance, so it is not the way how we compute the final score affect the performance. We think it is because the bag-of-words used to represent the content of a paper does not contain crucial or representative information that actually represent paper content. For example, in our model, bag-of-words only contains some nouns and verbs from title and keywords, and does not filter out nouns that are too common to be considered as a paper content. We think that the performance might be boosted if we consider phrase based vector instead of pure word vectors.

5. Future Improvements

Based on our discussion in previous sections, we suggest several future directions for improvement on natural language based method in author-paper identification problem.

The first direction is to go beyond word based features. As we pointed out in the previous section, a single word is not good enough to give solid information on paper content. One possible improvement is to considered phrase-based features since phrase can express more information than a single word. A simple way of doing this is to consider features like document vector or sentence vector which are extension of word vector model in natural language processing field.

Another direction is to improve dataset. Since the given dataset only contains two kind of language data, which are title and keywords respectively, it is hard for language related features to show their true power. One challenge in this direction might be that most language data are unstructured and huge in size. Hence, we think a good direction is to study low dimensional representation and dimension reduction techniques for storing language related data.

A final remark on direction of improvement is about the model design itself. One language feature might be a weak classifier in this task, but a collection of weak classifiers could potentially constitute a strong classifier. This idea of building strong classifiers from weak classifiers is the core idea of boosting techniques, such as Adaboost.

6. Conclusion

In conclusion, our word vector based language feature did not provide much useful information for the paper-author identification task. We believe the performance should be boosted if we have a language rich dataset. We also proposed three possible directions for future improvements.

Although we failed at using word vector to propose simple models for this task, we still believe language-based features could provide a simple model for this task in the future in a language rich dataset.