Introduction

In recent years, authorship attribution has attracted much attention due to its useful applications in fields like privacy defenses, plagiarism detection, and some other areas. In this report, the task is to analyze different algorithms and different ways to represent data in order to properly predict the most likely author given the relevant paper information.

The dataset contains the paper information which includes the authors of the paper, the year and venue published also the list of keywords in the title and abstract. In order to apply the machine learning algorithms, the data has been transformed and labeled to be ready for applying algorithms in a supervised fashion. The transformation includes for each paper, one of the authors in the group has been decided to become the label also called the target author, and the remaining authors are chosen to be coauthors. The year, venue, and coauthor information are all categorical variables and need to be transformed using the one-hot encoding. The "keywords" vector, however, has been decided to use various ways to encode it using 4 different ways: one-hot encoding, TF-IDF, word2vec, and using graph embeddings. The supervised machine learning algorithms used are Naïve Bayes and logistic regression.

Methods

- One hot encoding

The easiest way and most straightforward way to represent a "keywords" vector is using the one-hot encoding. In the example of the keyword, the input is a list of integers and the output is a vector of 0 and 1s with dimensions the same as the number of distinct elements in the keywords data. This method is fast and popular with to handle the categorical data. In this dataset, all the features are categorical, thus all of them need to be transformed using one-hot encoding. The target label doesn't need to use this transformation.

- Tf-idf

The TF-IDF method, which means the "term frequency and inverse document frequency", is a popular method in the information retrieval field. One of the drawbacks of one-hot encoding is it's hard to calculate the similarity between different vectors, and thus can't be used to interpret the significance of each element. TF-IDF however, can handle this issue. The intuition behind this is to consider the importance of each keyword in two parts: one is term frequency and document frequency. For each paper, we have a collection of keywords and if each word has multiple occurrences, the term frequency is high, and consider this keyword has larger importance. The inversed document frequency means if a keyword has occurred in many papers, then it's considered less important. TD-IDF is widely used on test data, however, in this example, it seems that the keywords in each paper are distinct and the term frequency for all elements are the same. The dimension of this encoding depends on the number of distinct keywords in the training set.

- Word2vec

Word2vec, developed by Tomas Midolov in 2013[1], is a popular way to obtain the embedding of a word in a given corpus. This method is an unsupervised learning technique and receives the input of a collection of sentences, in this example, is a collection of keywords list, and outputs a continuous representation for each keyword. A continuous representation catches the significance and importance of each keyword better than both using the one-hot and TF-IDF approaches. In this task, the method for achieving word2vec uses a method called skip-gram. One of the assumptions for word2vec is that the meaning of a keyword can be also represented using its surrounding keyword, that is, if a set of keywords co-occur in a paper, then their meanings are likely to close with each other. The skip-gram receives the input of each keyword to predict the

context of this keyword. The word2vec and TF-IDF are widely used in the NLP field. We use python's built-in function to achieve this.

- Node2vec

Another way to think of the set of keywords as an undirected graph. The node of this graph is each keyword in the dataset and the edge between two nodes represents the two keywords that have co-occur in a paper. Node2vec uses the method called Deepwalk. This algorithm is to choose a starting node and randomly selected its neighbors, moves to its neighbour, and continues this process repeatedly. The path of the visited node is a random walk. After applying random walk, we can get a collection of paths or a collection of a list of keywords and use word2vec to convert it to embeddings. The intuition for this algorithm is the nodes which close to each other on the graph are also close to each other in the embedding space. The Deepwalk can be achieved using the karate club library in python.

Results

	ML	AUC on validation	AUC on Kaggle (public)
One-hot	NB	0.78380	-
	LR	0.94719	0.91538
TF-IDF	NB	0.86252	-
	LR	0.94706	0.88607
Word2vec	NB	0.74360	-
	LR	0.9336	0.85378
Node2vec	NB	0.74473	-
	LR	0.92350	0.83491
One-hot+Node2vec	LR	0.95391	0.91872

The table shows the AUC on both validation sets and on the Kaggle platform with different combinations of encoding methods and machine learning algorithms, in which NB represents multinomial Naive Bayes and LR represents logistic regression. For One-hot encoding, the Multinomial Naive Bayes algorithm was chosen, and Gaussian Naïve Bayes is used for the rest of the encoding algorithms. The validation set contains 80% of the data. Since the evaluation metric used is AUC, the positive samples are the same as the original validation set and the negative samples for the validation set are using the feature set as a positive sample but labels are just random numbers.

From the table, we can find the logistic regression generally outperformed Naïve Bayes. This could be due to the naive assumption of independence may not hold in this case. Combinations with the best performance on the validation set are submitted to Kaggle. In general, the performance on Kaggle is not as good as the one on the validation set. The AUC calculated using all combinations with logistic regression can reach at least 0.92 on the validation set, but can't reach 0.92 on the Kaggle platform. The result may indicate there is potential overfitting or probably due to the imprecise way we used when generating negative samples.

The combination with one-hot with node2vec gives the best AUC on Kaggle, which is only slightly higher than using purely one-hot encoding.

Discussions & Analysis

The performance for combinations using logistic regression with different encodings, however, does not show the pattern we expect. The word2vec and node2vec which are used to generate continuous embeddings should have higher performances than TF-IDF which should also be better than using one-hot. The result is the reverse of our expectation, which AUC using one-hot with logistic regression is surprisingly the best encoding algorithm compared with others in this task. Node2vec is the worst in this case.

As covered before, the keywords may have similar term frequency for the TF-IDF method, the only difference is the reversed document frequency, which means the encoding using a TF-IDF is just a scalar product of one-hot encoding for each keyword. The logistic regression is expected to be insensitive to this kind of change and always return to the same result. The little change in their AUC is due to the dimension of keywords using TF-IDF being only 490 due to some unseen words in the training set.

The reason for word2vec does not obtain a satisfactory result could be the assumption fails. The keywords listed in each paper are not ordered, which means the keyword may not have the same meaning as its surrounding keywords. This may be solved using a larger window size. Window size decides for a certain keyword depending on how many surrounding keywords. A large window size, however, could lead to different keywords will have a large intersection of context and lead to the result that the node can't be distinguished from each other and thus generate bad embeddings.

Node2vec has the worst performance may be due to the graph having too large edges. This graph contains only 500 nodes but has 117438 edges. This shows that the majority of keywords are interconnected, so each node probably contains huge amount neighbors, and the neighbors for different nodes may have a large intersection. This is the same issue as word2vec with a large window size which also leads to bad performance.

Conclusions & Improvements

In this report, we have analyzed different combinations of encoding methods and machine learning algorithms for the task of authorship attribution. One-hot encoding is surprisingly the best encoding method with logistic regression. Combined with node2vec gives the highest AUC of 0.918 on the Kaggle. The TF-IDF in this task is equivalent to the one-hot encoding due to the structure of the dataset. Other encoding algorithms, like word2vec or node2vec, are not working very well. These two encoding methods use similar ideas but the keywords data in this example, in which most of the keywords are interconnected, is different from the text data in which the word only depends on a certain context used. In the future, the graph embedding can be built on the co-author vector and investigate the importance and similarity between the different authors. This is more helpful than building embedding on the keywords vector because the author can only be connected with a certain amount of authors and therefore can use to generate better embeddings.

Reference

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.