COMP90051 Report

Introduction

Authorship attribution is the process of identifying the author of a certain document. It has received substantial research in the areas of machine learning, natural language processing, linguistics, and privacy research and is regularly applied in real-world settings. In this project, we are given training data containing information including the title of the paper, the abstract of the paper, the authors, the year it was published and the venue it was published in. The test data is a list of 800 papers published after the training period. The top 100 (0-99) are prolific authors. Our task is to predict which prolific authors are the true authors for the paper. The result could be zero, one or many of these authors.

Feature engineering

First, we start with inspecting the training data. All information is given randomly assigned IDs, except the year of publication.
we start to process features. There are five columns in the training dataset, “authors”, “year”, “venue”, “title”, “abstract”, which gives us five corresponding features.

Label

To train a model, we need to get the labels, so we start with extracting the labels for training data. The goal is to predict the prolific author of the paper, so the first step is to split authors in the training dataset. We know that there are 100 prolific authors and some of the papers don’t have a prolific author, so we decide to do a one-hot encoding which is an array with length 101 with the last index representing no prolific author. For example, if one paper is written by author 0 and 1, the label is [1,1,0,0,...,0,0]. If there is no prolific author, we set the last element of the label to 1, e.g. [0,0,0,...0,1].

Text

After getting the labels, we start by with title and abstract, since we think the context is the most relevant feature. One researcher would be strongly related with some specific field mentioned as keywords in text features. We first try to use TF-idf (term frequency-inverse document frequency) which is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. However, this doesn’t give us the desired result, only [] f1 score. So we try to implement another approach, Word2Vec. Word2vec is another of the frequently used word embedding techniques. The entire corpus is scanned, and the vector creation process is performed by determining which words the target word occurs with more often. In this way, the semantic closeness of the words to each other is also revealed. So we not only pay attention to the frequency of the appearance of words. Using word2vec gives us a slightly better result. Then we apply doc2vec which is the generalization version of word2vec. It represents a document as a vector while word2vec computes a vector for every word. In word2vec,we train to find word vectors and then run similarity queries between words. In doc2vec, you tag your text and you also get tag vectors. Using doc2Vec gives us a significant improvement in the prediction f1 score as it extracts more meaningful sentence information from the text.

Venue

For the "venues" attribute, we decide to use this data since authors are more likely to publish their paper in the same venue. We simply scale it using a standard scaler and add this to our feature array since we can't think of a better way to encode this data.

Authors

For the "authors" attribute, we can't really think of a good way to represent them, so we decide to use one-hot encoding. For authors, we only consider those who have co-written some papers with prolific authors. For papers with no such coauthors, we simply set the last element to 1 like we did for the labels.

Year

For the "years" attribute, we thought it was not very useful, since the year of all test data is 19. And the training data only contains papers before 19. Another reason for not using the year attribute is that people don't usually publish multiple papers in one year which means we can't learn much information by looking at the year attribute.

Feature: Final concatenation

For the input for the model, we concatenate all the preprocessed features to get a single array containing all the features. For the model to learn better, we then scale the data using StandardScaler so that they have mean = 0 and standard deviation = 1.

Training models and Learners

We have experimented with a variety of data engineering methodologies and problem modeling approaches in this project.

Classifier chain

At first, we intended to create 100 binary classifiers for every author that needed to be identified in relation to the paper. In each model, we converted the feature dictionary into a single hot sparse-matrix for the training feature and did the same for the label, which indicates whether a given author is the actual author of the paper in the given context. The recall rate $\frac{tp}{tp+fn}$ can reach very high levels that are all above 95% when testing with a single author classifier. However, when combining the results from each classifier, the final result is only 35% accurate, which is not satisfactory. This is likely due to the inaccurate results from each model potentially influencing one another and ultimately producing the unsatisfactory result in each paper in the test set.

Neural network

We then try a deep learning neural network model. To get started we tried three fully connected layers with default settings. The first layer input dimension matches with our feature dimension, about 6777 (6576+200+1) parameter, the second layer shrinks to 2048 neurons then the third layer with 1024 neurons and final layer with 512 neurons and output dimension is 101 which represents the probability of the existense of the prolific author. After trying different activation functions “relu”, “sigmoid”, “tanh” and several loss functions "binary_crossentropy", "categorical_crossentropy". We found out that using “tanh” in the hidden layer and “sigmoid” in the final layer and “binary_crossentropy” loss function when fitting the model gives the best f1-score 60%. Activation function “tanh” outpurange lies in (-1, 1) gives more information in the middle of training, while “sigmoid” gives range (0, 1) to simulate the probability of author existence.

Balancing dataset

The test data set has around 25% of articles with no prolific authors, whereas the training set contains 71% of articles with no prolific authors. This is something we have noticed. In order to balance the difference between the training set and the test set, we therefore develop a number of data generating strategies. One of the generating strategies is to exclude some of the training set's articles while making sure that the training set's processed version includes every word from the abstract and title. However, this tactic might result in severe data loss, which would reduce the classification model's accuracy noticeably. The addition of repetition data to the original training set is the next data production approach we'll cover in order to perform the comparison. Our team first lists every article index with a prolific author, then stores all the processed features in a total list, continually concatenates the repeated data, combines it with the original training set, and simultaneously adds relative labels to the label set.

When these two balancing strategies are applied to the same model, data repetition consistently outperforms data reduction, which results in a f1 score of 41.64% while reduction’s f1 score is 38.57% for Classifier chain approach. Because there is no data loss when the data set is reduced, more feeding data indicates that the model will be more applicable to the real data environment. Data repetition would yet lead to some degree of overfitting because a significant percentage of the data has exactly the same value. However, the f1-score on validation dataset using ANN approach reaches 80% which is unrealistic. The possible reason is that we duplicated data to balance the dataset. After splitting the dataset into train and validation, we may possibly see the exact same instance in the both validation dataset and train dataset and this will result in a fake high f1-score. To overcome this issue, we use class weight when fitting models to change the weight of classes to have a similar proportion as the test data. And to simulate the test dataset, we set the weight of no prolific author instance to 0.25.

Conclusion

To conclude, we have implemented comprehensive feature engineering also utilizing and fine-tuned both classical models and neural network models. We have critically analyzed the pros and cons of our selected feature, hyperparameter and explained the challenge we face as well as how we overcome them. For further improvement, we may choose to convert our feature in the nodes and apply a Graphic Neural Network (GNN). Graph Neural Networks are able to learn graph structures for different data sets, which means they can generalize well to new datasets. In our case, the cooperation of A and B may infer the future cooperation of B and C. These kinds of relation information are hard to learn from our model but work well on GNN.