Text Mining Project(732A92)

TEXT CLASSIFICATION OF HEAVILY IMBALANCED DATA USING SIMILARITY BASED APPOACHES

Abhinay Krishna Vellala

LINKÖPING UNIVERSITY

Contents

Abstract

Solving classification problem in Text Mining has it's unique ways where two documents are combined together and then classified. However, for the cases where comparing two documents and classifying for a relation(label to be specific) remained the same. Even though the classifiers are performing well with the combination of texts, the curiosity of solving the comparison based problems by a using similarity approaches is the intention of this project.

In the Machine Learning world particularly while solving classification problems, it is important to have a good amount of data for each class for a Machine Learning model to give better predictions. But, Data extracted from various sources is improper most of the times and will not have enough number of rows per each class all the time. Inspite of imbalance in data, there is a high chance that Machine Learning models fail to generate better predictions. One way to solve the problem is by removing few rows and balance the data. This might lead us to loose some valuable information. There are various other techniques like Downsampling, resampling, generating synthetic data [1] and so on. Most of these techniques lead to either loose data or generate new data. This paper addresses the problem of classifying data without any type of resampling. With text data, there are many other ways to be considered to extract required information and classify the text based on that. This paper is an attempt to solve the problem of imbalanced data and classify the text based on Similarity rather than depending on Machine Learning algorithms.

Introduction

In contrast with other data types, text data has an advantage of dealing with context of the document. In this paper, experiments are done based on context and topic to generate similarity between the two documents. The idea is to compare two documents for similarity and check if they belong to same label based on similarity score. This project focuses on using Cosine similarity where it takes two vectors and calculates the distance between two vectors based on cosine index.

$$CosineSimilarity(A, B) = \frac{A.B}{||A|| * ||B||}$$

The threshold of similarity is considered as hyper parameter to determine if both the documents are similar or not. If they are similar, they belong to same class. Else, they are different.

Data

To experiment the mentioned theory, data is taken from Kaggle competition "WSDM - Fake News Classification"[2]. The competition is meant to run classification algorithms and determine the best approach to classify this dataset with the given labels. The dataset comprises of two documents, one is the given news article and other is real news article. The labels are 'agreed', 'disagreed' and 'unrelated'.

Given the title of a fake news article A and the title of a coming news article B, participants are asked to classify B into one of the three categories.

agreed: B talks about the same fake news as A

disagreed: B refutes the fake news in A

unrelated: B is unrelated to A

This data is heavily imbalanced where there are more rows that have class 'unrelated'. The description of imbalance is shown in figure1.

```
The datapoints per each category are
unrelated    219313
agreed        92973
disagreed      8266
Name: label, dtype: int64
```

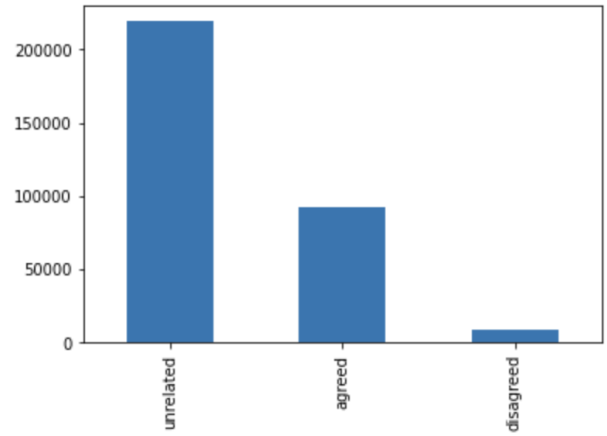


Figure 1: Imbalance in data

This data is indeed suitable to run the similarity based experiments as it has two documents and a label which says if they are unrelated or related. If related, agreed or disagreed. Further preprocessing is done to make it suitable for this project. Importantly, this dataset is heavily imbalanced and the experiments performed shows some alternative ways to do text classisifaction

Pre-processing

The data comes in two files(train.csv, test.csv). Train file has the shape (320552, 8). From all the rows, first 30000 rows are used to run the experiments.The data has two columns where text is in Chinese. Those columns are dropped and only columns where text is in English are considered. Labels are further reduced to two classes instead of three because we are interested only to check if they are related(similar) or unrelated. All the experiments here focuses on checking the similarity rather than determining if they are agreeing or disagreeing. The head of the data after pre-processing in Figure 2. A plot of histogram of labels after the pre-processing is showin in figure 3.

	id	tid1	tid2	title1_en	title2_en	label
0	0	0	1	There are two new old-age insurance benefits f...	Police disprove "bird's nest congress each per...	unrelated
1	3	2	3	"If you do not come to Shenzhen, sooner or lat...	Shenzhen's GDP outstrips Hong Kong? Shenzhen S...	unrelated
2	1	2	4	"If you do not come to Shenzhen, sooner or lat...	The GDP overtopped Hong Kong? Shenzhen clarifi...	unrelated
3	2	2	5	"If you do not come to Shenzhen, sooner or lat...	Shenzhen's GDP topped Hong Kong last year? She...	unrelated
4	9	6	7	"How to discriminate oil from gutter oil by me...	It took 30 years of cooking oil to know that o...	agreed

Figure 2: Head of pre-processed data

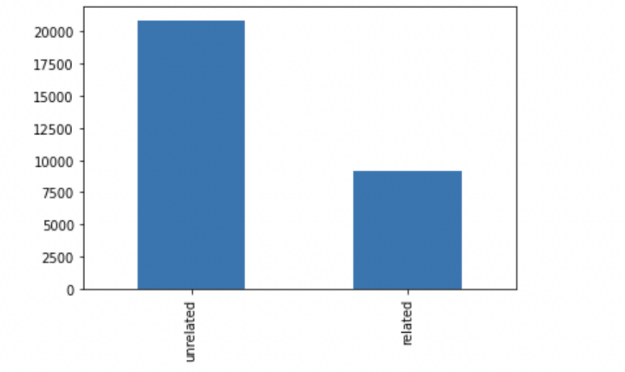


Figure 3: Labels of pre-processed data

Approch

Initial check is done to see how this data is performing on Bernoulli Naivebayes classifier. The two documents are initially combined to a single column and then extracted a vector of Tf-Idf values. The vectorized matrix is then used to classify the data. Here are the results

	precision	recall	f1-score	support
related	0.58	0.71	0.63	30180
unrelated	0.85	0.76	0.80	65986
accuracy			0.74	96166
macro avg	0.71	0.73	0.72	96166
weighted avg	0.76	0.74	0.75	96166

Figure 4: NaiveBayes result on data

The results look promising. It has got some good values of precision, recall and f1-score and the model has achieved 74% accuracy. The method used to classify is to combine two documents, later vectorized and then classified. Even with Word embeddings or transformers, we combine two documents together and perform classification(Source: 732A92 - LiU Text Mining lab2, lab 3, lab 4).

This project is an attempt to find alternate ways to classify the data by comparing two documents separately. Approaches used here are:

1. Context based Similarity - Topic Modelling
2. Semantic based similarity - Embeddings

In Context based similarity methods, each article from 'title1_en' are clustered independently. Similarly, each article from 'title2_en' are clustered independently. If both articles from a single row belong to same cluster, then they are related. Else, they are unrelated. Here, articles are clustered based on distance between word in it's respective vectorized form, which is tells about the similarity. If article from 'title2_en' also fall in same cluster, it means that the distance between two articles should less in the vector space and the two articles are 'related'. This idea is carried forward and clustering approaches like K-means, Gaussian Mixture Model and Latent Dirichlet Allocation are used to model the topic(Cluster) and later compared with each other.

Semantic based similarity is a distance or cosine based similarity index but here, each word of the document or each document is embedded and then vectorized. This approach focuses on representing data as a Continuous Bag of words where each document from article A and article B is transformed as a vector and check the

Cosine Similarity between them. Two documents are 'related' when cosine similarity is high. Else they are 'unrelated'. This idea is checked with Simple similarity calculation of Word2Vec, Transformer which sums up all the word vectors and Doc2Vec approaches.

Context based Similarity - Topic Modelling

In this approach, before classifying the two documents, it is important to cluster them based on their topics. This classifies every article based on context. Later, the articles are checked if they belong to same topic. Here is the pipeline:

1. Cluster the article A and article B independently.
2. Check if both articles belong to same cluster.

Hyperparameter : Number of topics/clusters.

From the data, it is unclear the number of topics. So, Unsupervised learning approaches are used with the number of topics/clusters as hyperparameter.

K-means Clustering.

K-means clustering is one of the simplest approaches to determine the pattern in the data and classify into clusters based on distance. The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-of-squares criterion [3]. Given a vector like Tf-Idf vector of all the words, it randomly chooses k number of centroids and calculates distance between all the vectors from centroids in a multi-dimensional space. The clusters are then updated according to the within-cluster sum-of-squares. This approach is useful in determining closest documents and create a topic based on that. Since the number of clusters the most important aspect here, clusters from 5, 10, 15, 20.

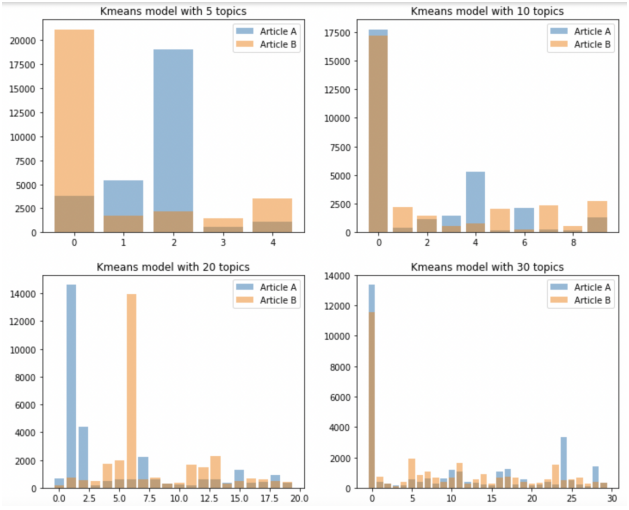


Figure 5: Classification using kmeans

The results shows even the accuracy is above 50% all the cases the precision, recall values says that the the model has done well only with the majority class. The accuracy is based only on 'unrelated' class. This means most of the times, the two articles are not in the same cluster. There is a high rate of False Positives and True Negatives and it is well explained in precision, recall.

K_value	Accuracy
5	0.63
10	0.68
20	0.67
30	0.68

K_value	Labels	Precision	Recall	f1.score
5	related	0.32	0.20	0.24
5	unrelated	0.70	0.82	0.75
10	related	0.27	0.04	0.07
10	unrelated	0.69	0.95	0.80
20	related	0.25	0.05	0.08
20	unrelated	0.69	0.94	0.80
30	related	0.25	0.03	0.05
20	unrelated	0.69	0.96	0.81

Gaussian Mixture Model

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.[4]

To perform GMM, the dimensions of the vectorizers are reduced using t-distributed Stochastic Neighbor Embedding. GMM uses Expectation-Maximization algorithm to get the marginal probability distribution of each cluster.[5]

GMM clustered the data better than K-mean. Even though there wasn't great improvement in accurary, the weightage on minority class('related') has been improved compared to K-means model.

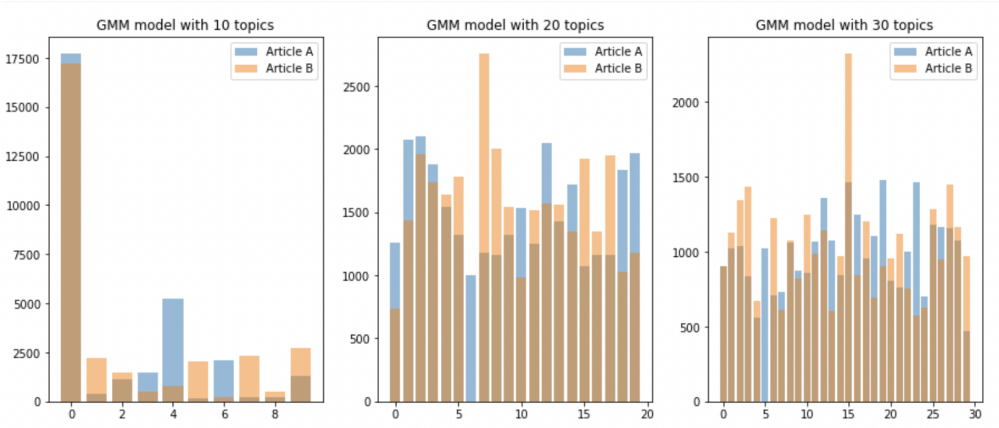


Figure 6: Classification using Gaussian Mixture

Gaussians	Accuracy
10	0.66
20	0.68

Gaussians	Accuracy
30	0.68

Gaussians	Labels	Precision	Recall	f1.score
10	related	0.33	0.11	0.17
10	unrelated	0.70	0.90	0.79
20	related	0.37	0.07	0.11
20	unrelated	0.70	0.95	0.81
30	related	0.11	0.01	0.01
20	unrelated	0.69	0.97	0.81

Latent Dirichlet Allocation

In LDA, each document may be viewed as a mixture of various topics where each document is considered to have a set of topics that are assigned to it via LDA. This is identical to probabilistic latent semantic analysis (pLSA), except that in LDA the topic distribution is assumed to have a sparse Dirichlet prior.[6]

While LDA gives weight of all the topics for given document, the topic with maximum weight is assigned to the document. However, it has given 1 topic maximum number of times. From the precision, recall values, the model is not good with minority class.

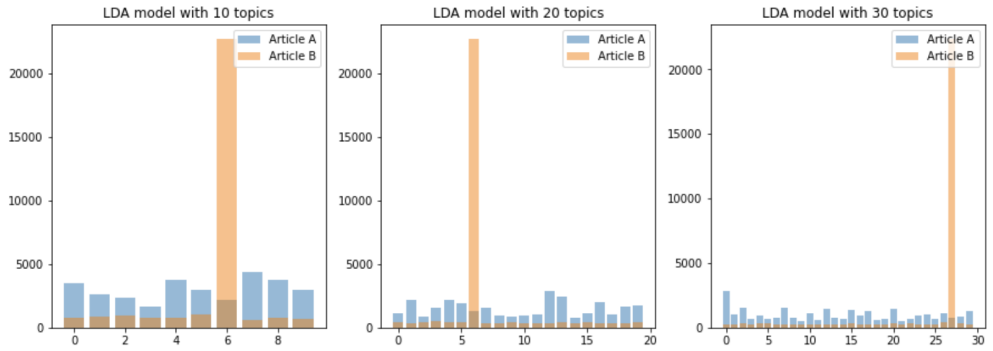


Figure 7: Classification using LDA

Topics	accuracy
10	0.67
20	0.67
30	0.69

Topics	Labels	Precision	Recall	f1.score
10	related	0.32	0.08	0.13
10	unrelated	0.70	0.93	0.80
20	related	0.27	0.04	0.07
20	unrelated	0.69	0.95	0.80
30	related	0.41	0.04	0.07
20	unrelated	0.50	0.98	0.82

Clustering approaches couldn't solve the problem of imbalance. Most of the time, title1 and title2 fall into different cluster and increase the number of 'Unrelated' in the prediction. So, even when accuracy is around 60, the rate of false positives and true negatives are too high and led to bad classification for imbalanced data.

Semantic based Approaches

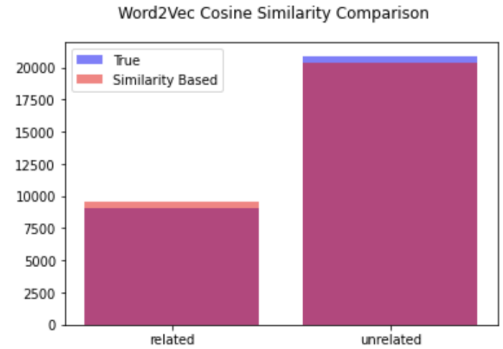
Clustering approaches failed to classify data properly as the ground truth of classification creteria was weak in determining classes if two documents fall into same cluster. Semantic based approaches deal with the relationship between the words. One of the best things happened in Natural Language processing is Word2Vec. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.[7] This concept is further extended to Deep Learning where many modern architectures like LSTM, Transformer uses RNNs to improve the relation between word vectors and derive usefulness. There are many popular architectures and pre-trained models that are built on this idea. In this project, experiments uses the Semantic relations between word vectors and determine the similarity between two documents to solve text classification problem.

Word2Vec - Semantic similarity

Word embedding is build on the concept of Continuous bag of words and one hot vector. This made Word2Vec most efficient in terms of determining the similarity by calculating cosine distance[8]. Similarity is determined by comparing word vectors or word embedding, multi-dimensional meaning representations of a word. Word vectors can be generated using an algorithm like word2vec[9].

In this approach, a vector of every word in each document is considered as whole and cosine similarity is calculated between them. A threshold value is used a hyperparameter and the classification is done based on it.

When threshold value = 0.9, this approach has shown great result by classifying the data.



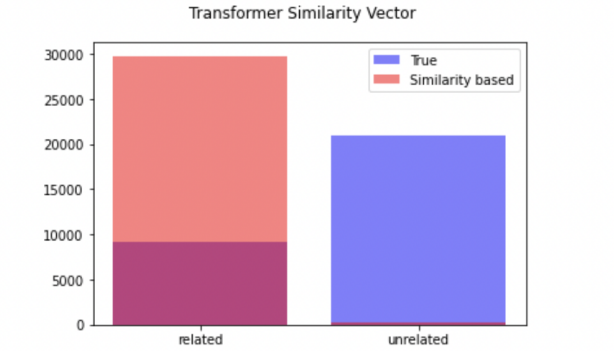
	precision	recall	f1-score	support
related	0.58	0.61	0.59	9102
unrelated	0.82	0.80	0.81	20898
accuracy			0.74	30000
macro avg	0.70	0.71	0.70	30000
weighted avg	0.75	0.74	0.75	30000

Figure 8: Word2Vec -Cosine similarity, threshold = 0.9

The results are promising as the precision and recall values have improved.

Using Transformer

The inspiration for using Transformer is from LiU course: Text Mining(732A92) lab 4. The idea is to use Transformer that sums all the word vectors in each article. The cosine similarity is then calculated between the summed vectors. The hyperparameter is used as threshold value. In this experiment, most of the times the cosine similarity was above 0.90. Even when threshold value is 0.95, the model still produced poor classification. It is indeed interesting to see the precision and recall values here. The transformer considered every document is similar and has classified less number of 'unrelated' labels.



	precision	recall	f1-score	support
related	0.30	0.99	0.47	9102
unrelated	0.77	0.01	0.02	20898
accuracy			0.31	30000
macro avg	0.54	0.50	0.24	30000
weighted avg	0.63	0.31	0.15	30000

Figure 9: Transformer - Cosine similarity, threshold = 0.95

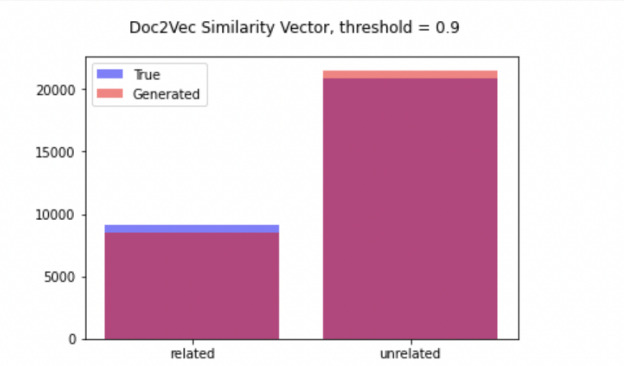
Using Doc2Vec

The objective of Doc2Vec is to create the numerical representation of documents unlike word2vec that computes a feature vector for every word in the corpus, Doc2Vec computes a feature vector for every document in the corpus. The vectors generated by doc2vec can be used for tasks like finding similarity between documents[10]

In order to build Doc2Vec, a vocabulary should be created with the documents we have and the model should be trained using the same. It is build from a sequence of sentences. This represents the vocabulary or Dictionary of the model which keeps track of all unique words [10].

The requirements are fulfilled using gensim package and dictionary is created using all the preprocessed words in the data. A model is trained for Doc2Vec. The implementation is based on demo code from gensim documentation[11].

With a threshold value of 0.9, Doc2Vec approach has classified the model above 55% accuracy.Plot looks it has estimated the model well, but from precision and recall values says, it has high amount of False positives and True negatives. This is the reason for low accuracy.



	precision	recall	f1-score	support
related	0.25	0.23	0.24	9102
unrelated	0.67	0.69	0.68	20898
accuracy			0.55	30000
macro avg	0.46	0.46	0.46	30000
weighted avg	0.55	0.55	0.55	30000

Figure 10: Doc2Vec - Cosine similarity, threshold = 0.95

Conclusion

Even when clustering failed in classifying the data, it was a good starting point for the similarity based classification experiments performed in this project. The results of word embedding are satisfactory and has good f1-score, recall, precision values. This can be further improved by using Deep Learning techniques like LSTM or BERT. However, this project addresses a major issue of imbalanced data. With the above approaches, size of the data does not impact the performance of the model. These approaches also helps in dealing with the data where labels are not available in Unsupervised Learning context.

Code

Github repository link: https://github.com/abhi-vellala/Text_mining_project

References

- 1 Methods for dealing with imbalanced data https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18
- 2WSDM - Fake News Classification: https://www.kaggle.com/c/fake-news-pair-classification-challenge/overview
- 3 Scikit learn K-means clustering documentation: https://scikit-learn.org/stable/modules/clustering.html#k-means
- 4 Scikit learn GMM documentation: https://scikit-learn.org/stable/modules/mixture.html#mixture
- 5 Gaussian Mixture model: https://towardsdatascience.com/gaussian-mixture-modelling-gmm-833c88587c7f
- 6 LDA Wikipedia: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
- 7 Word2Vec Wikipedia: https://en.wikipedia.org/wiki/Word2vec
- 8 From Count Vectors to Word2Vec: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
- 9 Spacy Linguistic Features: https://spacy.io/usage/linguistic-features#vectors-similarity
- 10 Sentence Similarity in Python using Doc2Vec: https://kanoki.org/2019/03/07/sentence-similarity-in-python-using-doc2vec/
- 11 Gensim Doc2Vec Demo: https://radimrehurek.com/gensim/models/doc2vec.html

Other inspiring references

- Compare documents similarity using Python: https://dev.to/coderasha/compare-documents-similarity-using-python-nlp-4odp
- Estimate Degree of similarity between two texts: https://medium.com/@adriensieg/text-similarities-da019229c894