2019

Improving Document Representation Using Retrofitting

Zeeshan Mansoor
University of Windsor

Improving Document Representation Using Retrofitting

By

Zeeshan Mansoor

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2019

Improving Document Representation Using Retrofitting

by

Zeeshan Mansoor

APPROVED BY:

G. Zhang
Department of Mechanical, Automotive and Materials Engineering

---

Z. Kobti
School of Computer Science

J. Lu, Advisor
School of Computer Science

April 16, 2019

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Data-driven learning of document vectors that capture linkage between them is of immense importance in natural language processing (NLP). These document vectors can, in turn, be used for tasks like information retrieval, document classification, and clustering. Inherently, documents are linked together in the form of links or citations in case of web pages or academic papers respectively. Methods like PV-DM or PV-DBOW try to capture the semantic representation of the document using only the text information. These methods ignore the network information altogether while learning the representation. Similarly, methods developed for network representation learning like node2vec or DeepWalk, capture the linkage information between the documents but they ignore the text information altogether. In this thesis, we proposed a method based on Retrofit for learning word embeddings using a semantic lexicon, which tries to incorporate both the text and network information together while learning the document representation. We also analyze the optimum weight for adding network information that will give us the best embedding. Our experimentation result shows that our method improves the classification score by 4% and we also introduce a new dataset containing both network and content information.

AKNOWLEDGEMENTS

I want to thank my supervisor Dr.Jianguo Lu for his guidance and support throughout my Master's program. This thesis would not have been possible without his help and mentorship.

I also want to thank my committee members Dr. Ziad Kobti and Dr. Guoqing Zhang for their valuable opinions and suggestions.

I also want to thank Yi Zhang for his assistance in analyzing retrofitting.

In the end, I want to thank my parents for their consistent support, guidance and help.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Obtaining good representation of documents is crucial for different machine learning tasks like classification, clustering and information retrieval. These tasks require the input to be of short length vector. Recently, the most common method for learning short length vector representation of text is Paragraph Vector (PV) [27]. Based on the distributed representation for words [31], PV proposed two models: Paragraph Vector-Distributed Memory (PV-DM) and Paragraph Vector-Distributed Bag of Words (PV-DBOW) with negative sampling and hierarchal softmax and the input for these methods is only the text information as shown in Figure 1. These methods assume that the documents are independent of each other.

Generally, documents are linked together in the form of hyperlinks in case of web pages and citation in case of academic papers. Researchers tried to capture linkage information while learning the representations like in node2vec [25]. These methods ignore text information altogether during this process. The limitation of the methods like PV-DBOW and node2vec is that they take information only from either the content or network view alone. Intuitively, the vector obtained from using content and network information together will give us a better vector representation.

Recently, the focus is shifted to learn multi-view representations [40]. For example, learning document representation of academic papers from multiple views. We can learn the representation using both the content and citation network by concatenating or averaging these embeddings. The aim is to improve the document embedding by

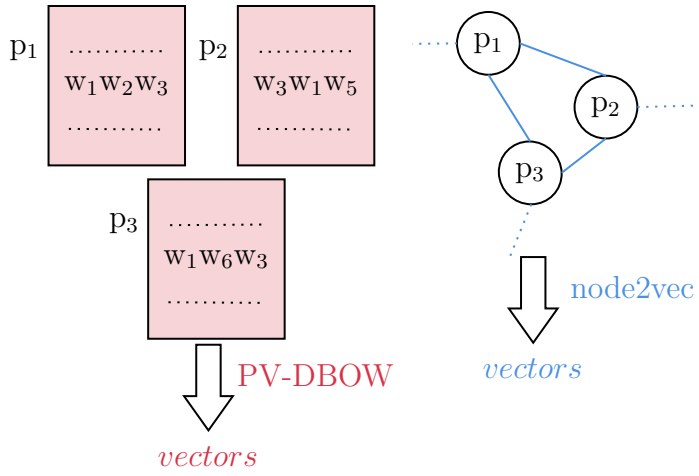


FIGURE 1: Learning Vector Representation from Content and Network

capturing different aspects of the documents and when combined will give a better result for downstream tasks like classification than their single view counterparts.

In this thesis, we focus on improving the document representation of academic papers using multiple views. Academic papers are more complex than plain texts or networks. Papers contain text and are linked together through citations or references. We can create a citation graph by connecting the paper together with the papers that it is citing. The linkage is done based on the intuition that a paper cites another paper which has a common topic or method. These network representation learning methods, do not capture the content information, so they regard all the papers connected in the graph equal irrespective of their topics as shown in Figure 1. For example, a paper in the domain of machine learning may cite papers related to math and machine learning area. If we want similar papers, the network representation methods will consider papers belonging to math and machine learning equal as it does not have the content information. Similarly, for methods like PV-DBOW will output papers pertaining to the machine learning domain but it will ignore the linkage of these papers with the math papers.

The contribution of this thesis is a multi-view based learning technique for document embeddings using citation and content information. In contrast to the previous

work, our method is applied as a post-processing step by running it on pre-trained document vectors obtained from any method. The proposed method encourages the new vectors to be similar to vectors which are linked together in the citation graph. This process is fast and takes about 5 seconds for a graph of 10,000 documents and vector length 100.

Our method is inspired by the existing method called Retrofit [22] for word embeddings. In retrofitting, they proposed a method in which they brought word embeddings closer together in the vector space based on the lexicons; meaning having the same semantics. We modified the algorithm to extend it to document embeddings and brought documents closer together based on the citation graph. Our experiments show that our method improves the classification performance when compared with existing methods.

1.1 Contributions

In this thesis we make the following contributions:

- Introduce a new algorithm for adding network information to the embeddings obtained using content information in the form of retrofit.
- Show how the addition of network information improves the document embedding through different methods.
- Analyze the reasons why when adding information from network and content fails for some methods or datasets.
- Introduce a new dataset called Arxiv which contains both linkage and content information for experimentation.

In the following sections, we will cover related work, problem statement, algorithm, experimentation, results and conclusion.

CHAPTER 2

Review of the Literature

This section will give a detailed analysis of the existing algorithms which try to capture linkage information along with the content in the document embedding. For each algorithm, we will explain the method and give a detailed description of the dataset used for experimentation.

2.1 Concatenating Document and Network Embeddings

2.1.1 Learning Document Embedding

Paper [27] introduces Paragraph Vector (PV), an unsupervised algorithm that learns fixed-length feature representation from variable length pieces of text such as sentences, paragraphs or documents. The vectors are represented as dense vectors and are trained to predict words in the document.

The PV have two models for learning document representation. We will explain

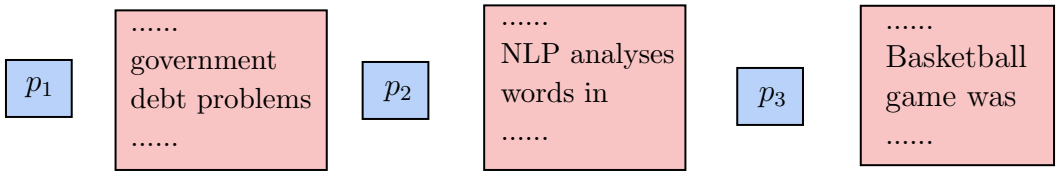


FIGURE 2: Each paragraphID will represent a unique paragraph

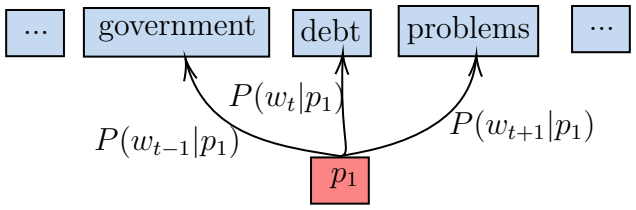


FIGURE 3: PV-DBOW Overview

Paragraph Vector-Distributed Bag of Words (PV-DBOW) model since we used this method in our experimentation

Figure 2 shows different paragraphs containing some text and each paragraph is represented by a unique ID as $p1$, $p2$ or $p3$. The model will take paragraph ID and content as input and will return the vector representation of all the paragraphs based on content only.

Figure 3 gives an overview of the PV-DBOW model. For paragraph $p1$ we want to maximize the probability of the words occurring in the paragraph with respect to all the words in the vocabulary. This way the paragraph vector $p1$ will capture semantic representation of this paragraph based on the content information.

$$J(\theta) \;=\; -\frac{1}{T} log(L(\theta)) \;=\; -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m<s<m \\ j\neq 0}} logP(w_j|p_t;\theta) \tag{1}$$

Equation 1 captures the vector representation of the paragraphs. θ are all the variables that we want to optimize. T is the total number of documents in our corpus. m is the window size and s is a word drawn randomly from that window. P is the probability which we want to maximize when we have paragraph vector as an input and want to predict the words present in that paragraph.

$$p(o|c) \;=\; \frac{exp\left(u_o^T v_c\right)}{\sum_{w\in V} exp\left(u_w^T v_c\right)} \tag{2}$$

The $P(w_j|p_t;\theta)$ is a softmax as defined in 2. In this case θ are the weight matrix $D^{d\times n}$ and $W^{n\times V}$ where d is the total number of documents, n is the total number

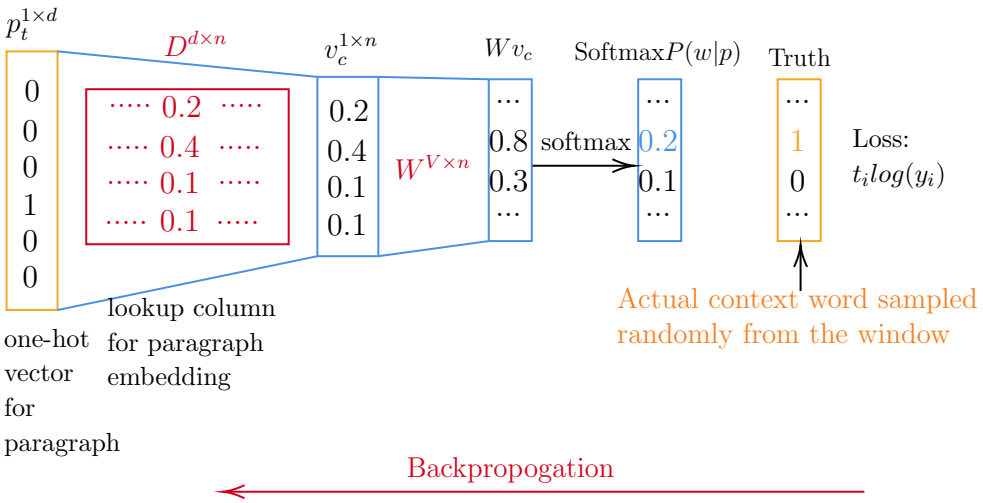


FIGURE 4: PV-DBOW Model

of hidden neurons or vector dimension and V is the total number of words in the vocabulary. The vector v_c is obtained by multiplying one hot paragraph vector p_c with D. This will select a column in D and we represent this vector as v_c. Similarly, u_o vector is obtained by the multiplication of v_c with W. u_o is the vector representation of the context word that we are trying to predict. To normalize the score, we will multiply our paragraph vector v_c with all the word vectors u_w in the vocabulary set V.

Figure 4 explains the PV-DBOW model using graphical representation. In the input, we pass one hot vector p_t and multiply it with the weight matrix D. It will select t^{th} column of the weight matrix D and we will pass this column to the hidden layer. The hidden layer is represented as a vector $v_c^{1\times n}$ where n is the number of hidden neurons. We will then multiply v_c with the words weight matrix W which will give us u_o vector containing the score in the output layer. Then we will pass u_o to the softmax which will give us a probability distribution of the words based on the input vector. To calculate the loss we will use one hot vector representation of the label word that we are trying to predict and multiply it with our softmax vector as shown in Equation 3 . To minimize this loss we will do backpropogation of loss with

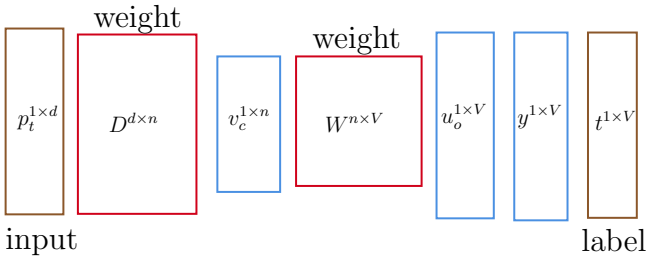


FIGURE 5: Weight Update

respect to the weight matrix D.

$$SLoss = t_i log(y_i) \tag{3}$$

$$\frac{Loss}{\partial D} = \frac{Loss}{\partial y}.\frac{\partial y}{\partial u_o}.\frac{\partial u_o}{\partial v_c}.\frac{\partial v_c}{\partial D} \tag{4}$$

Figure 5 gives a more abstract view of the PV-DBOW model and Equation 4 outlines the series of partial derivative that we have to do to update the weight parameter D where y is softmax, $v_c = Dp_t$ and $u_o = v_c W$.

The problem with the above model is that for each iteration we will have to calculate softmax and for its calculation we need to normalize the score for all the words in the vocabulary which can be 10^5 to 10^7.

$$J(\theta) = log\ \sigma(u_o v_c) + \sum_{k=1}^{K} E_j \in P(w)\ [log\ \sigma(-u_j v_c)] \tag{5}$$

To optimize this problem, the authors used Negative Sampling in which they tried to maximize the probability of the input and context vector and tried to minimize the probability between k random words drawn from the corpus and the input vector. The equation 5 defines negative sampling where σ is a sigmoid function defined as $\frac{1}{1+e^{-x}}$ and $P(w) = U(w)^{\frac{3}{4}}/Z$ is a unigram distribution for word w in the training set.

Dataset Used

For Sentiment Analysis experimentation, they used Stanford Sentiment Treebank Dataset [Sta]. This dataset contains 11855 sentences taken from movie review site Rotten Tomatoes. Each sentence in the dataset has a label ranging from very positive to very negative in the scale from 0.0 to 1.0. The sentences are further divided into subphrases and each subphrase is labeled. Human annotators did the labeling, and there are $239,232$ labeled phrases in the dataset.

For experimenting with paragraphs and documents, they used IMDB [IMD] dataset as a benchmark for sentiment analysis. The dataset contains $100,000$ movie reviews taken from IMDB. Each movie review consists of multiple sentences and reviews are labeled as positive or negative.

For the information retrieval experiment they used a dataset of paragraphs consisting of the first ten results returned by a search engine given each of $1,000,000$ most popular queries. To test vector representation, they created triplet of paragraphs such that the two paragraphs are results of the same query and the third paragraph is a randomly sampled paragraph from the rest of the collection.

During training, they learned the vector representation using the training dataset only and then passed the representation to learn Logistic Regression classifier for the prediction task. Once the model is trained, they tested it on the training dataset. They first froze the model and obtained vector representation of the test sentences and then passed this representation to the trained Logistic Regression model.

Sentiment Analysis

For sentiment analysis experiment with Stanford Sentiment dataset, they used two approaches for classification [27]. First, they proposed a 5-way fine-grained classification task where the labels are from very negative, negative, neutral, positive and very positive. In the second approach, they used a 2-way classification task where

labels are either positive or negative. Each phrase is treated as an independent sentence, and after learning vector representation, they are fed into logistic regression to predict the movie rating. The window size used is 8, and vector representation has a dimension of 400. The results show that their method outperforms the other baseline methods like the bag of words, bag of n-words or Naive Bayes by relative improvement of 16% in terms of error rates. Error rates calculates the percentage error by subtracting the original value with the predicted value and dividing it by the original value.

In the case of IMDB dataset, they learned vector representation through a neural network with one hidden layer with 50 units and a logistic classifier to learn to predict the sentiment. They used the same hyperparameters as they used in Stanford Dataset except the window size is 10. The result shows that the bag of words performed better than in the previous experiment because of long documents. Overall, paragraph vector again outperforms the other methods as it achieves 7.42% error rate which is the least when compared with the other techniques [27].

Information Retrieval

The objective of this experiment is to find which of the three paragraphs are from the same query in the dataset they created. To achieve this, they used paragraph vectors and computed distances between them. A better representation is one, which produces a small distance for pairs of paragraphs of the same query and a large distance for a paragraph from a different query. They used bag of words, bag of bigrams and averaging word vectors as a benchmark. They recorded the number of times where each method produces smaller distance for the paragraphs from the same query and error was made if the method does not provide a desirable result. The result showed that Paragraph Vector gave 32% relative improvement in terms of error rate [27] suggesting that the proposed method is useful for capturing the

semantics of the text.

Conclusion

This paper introduced an unsupervised learning algorithm that learns vector representation for a variable length of texts. In this method, it is assumed that documents(phrases) are independent of each other, but in the real world, it is the opposite. Documents are linked with each other in the form of links or citation in case of web pages or academic papers respectively. The document embeddings are not able to capture this linkage information by default; as a result, this method ignores this linkage information altogether. In the experimentation, it is also advised to use PV-DM with concatenating vectors and a window size of 5 to 12.

2.1.2 Learning Network Embedding using Node2vec

Paper [25] proposes an algorithm for learning continuous feature representation for nodes in a network by mapping them to low dimensional space of features that maximize the likelihood of preserving network neighborhoods of nodes.

Algorithm

This algorithm uses network information to learn the feature embeddings. Let u be the source node and $N_s(u)$ be the neighbourhood of nodes connected with u using sampling strategy like DeepWalk [33] or Node2Vec [25]. The set of sequence of nodes produced from this sampling strategy can be considered as a sentence and each node in the sequence as a word as shown in Figure 6. Then we can use a model similar to the skip-gram [31] to learn the feature embedding of u by predicting it's neighbor node in the output.

The process of learning the feature embedding is the same as described in the PV-DBOW. Only change is the input to the model as described in the objective function

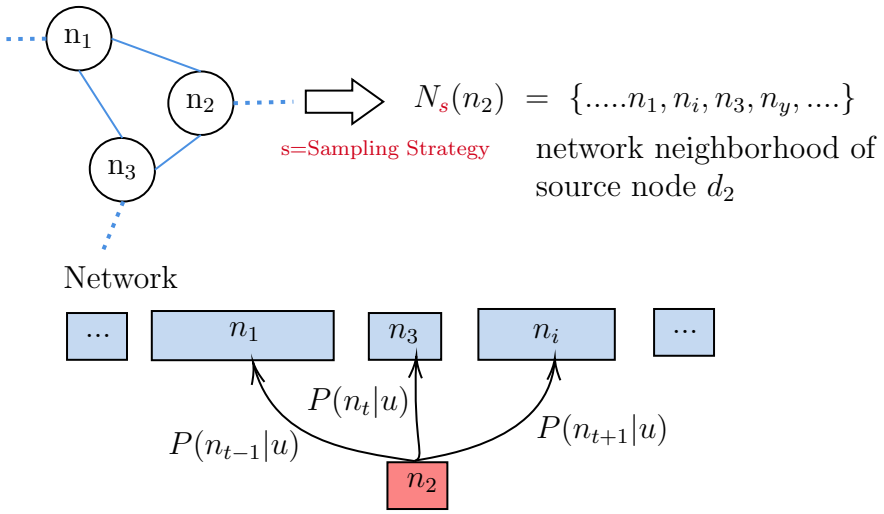


FIGURE 6: lnput to node2vec

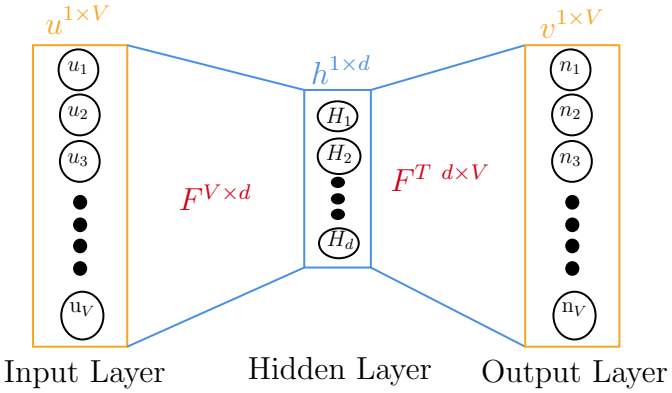


FIGURE 7: Node2vec Model

6 and shown in Figure 7.

$$max_f \sum_{u \in V} log\ P\ (N_s(u) \mid f(u)) \tag{6}$$

where f is a matrix of size $|V| \times d$ and V is the total number of nodes and d is the number of hidden neurons.

In node2vec, they use 2nd order random walk algorithm to find the neighboring nodes in the network. They define two hyperparameters p and q which tries to find nodes sharing a similar community and similar structure through breadth-first and depth-first search respectively. Figure 8 describes this concept in more detail.

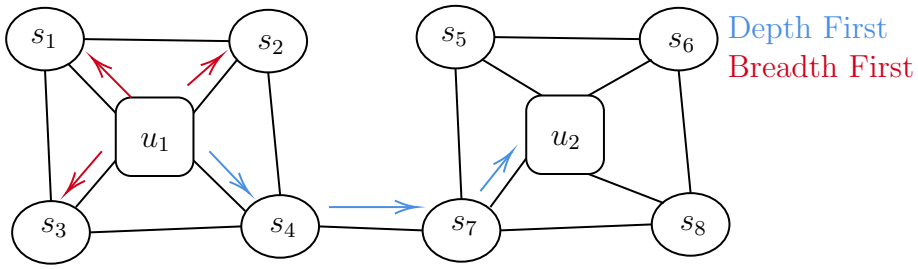


FIGURE 8: Graph Structure in node2vec

If we want to find nodes belonging to the similar community as the source node u, we will increase the value of q, so that breadth-first search is given more weight. Similarly to find nodes sharing a similar structure, we will increase the value of p to promote depth-first search in the random walk.

Dataset Used

This paper uses four different datasets for experimentation. For learning feature representation for a node in a network, they used Les Miserable Network [26], where nodes correspond to characters in the novel and edges connect coappearing characters. This network has 77 nodes and 254 edges. In multi-label classification experiments, they used three different relatively large datasets. First is BlogCatalog [43] dataset. It is a network consisting of the social relationship between bloggers. This network

has $10,312$ nodes and $333,983$ edges. Second is Protein-Protein Interaction [PPI] [34] which contains subgraph of the PPI network of Homo Sapiens. This network has $3,890$ edges and $76,584$ nodes. Lastly, they used Wikipedia [wik] dataset containing co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. This network has $4,777$ nodes and $184,812$ edges.

For link prediction experiment, two additional datasets are used; Facebook [28] and arXiv ASTRO-PH [arx]. In facebook, nodes represented users and edges represented friendship relation between them. The network has $4,0396$ nodes and $88,234$ edges. ArXiv is a collection of network generated from papers submitted to the e-print arXiv. In this dataset, nodes represent scientists, and an edge is present if two nodes co-authored in a paper. This network has $18,722$ nodes and $198,110$ edges.

Learning Edge Features

For experimentation, they used node2vec to learn feature representation for nodes in a network. They observed that Breath First Search (BFS) and Depth First Search (DFS) strategies both represent extreme ends on the spectrum of embedding nodes based on the principle of homophily (similar network communities) and structural equivalence (structural role of nodes). They demonstrated that node2vec could obtain embeddings that obey both the principles. They set $d = 16$ and ran node2vec on the Les Miserables dataset. Then the feature representations were clustered using k-means. Then they visualize the original network in two dimensions with nodes now assigned colors based on their clusters. They set $p = 1$ and $q = 0.5$ to capture nodes which interact more often with each other hence same community. This characterization relates to homophily. To discover structure equivalence they set $p = 1$ and $q = 2$ as node2vec hyperparameters. This setting captured nodes which have the same structural role.

Multi-label Classification

In multi-label classification experiment, every node was assigned one or more labels from a finite set. During training, they used a certain fraction of nodes and all their labels. The objective of the experiment is to predict the labels for the remaining nodes. They used BlogCatalog, PPI and Wikipedia dataset for this experiment. They compared node2vec against Spectral clustering [38], DeepWalk [33] and Line [36] for evaluating the performance. The node features were passed to a one-vs-rest logistic regression classifier with L2 regularization. They used Macro-$F_1 score$ for comparing the performance. Node2vec outperformed other methods because of the added flexibility in the algorithm by fixing hyperparameters p and q to low values. This setting enabled node2vec to capture homophily and structural equivalence in the network.

Link Prediction

In link prediction, first, they removed 50% of edges randomly from the network while making sure that the residual network remains connected after these edges are removed. The objective of this experiment is to predict the missing edges. They used Facebook, PPI and arXiv dataset for this experiment. They compared node2vec and other benchmarking algorithms with some popular heuristic scores that achieve good performance in link prediction. Node2vec outperformed DeepWalk and Line in all networks with gains up to 3.8% and 6.5% respectively using AUC scores.

Conclusion

This paper focuses on learning feature representation of a network as a search based optimization problem. It explains the classic search as a trade-off between exploration and exploitation. It shows that BFS can explore only limited neighbourhoods and it is suitable for characterizing structural equivalences in the network. Whereas, DFS can

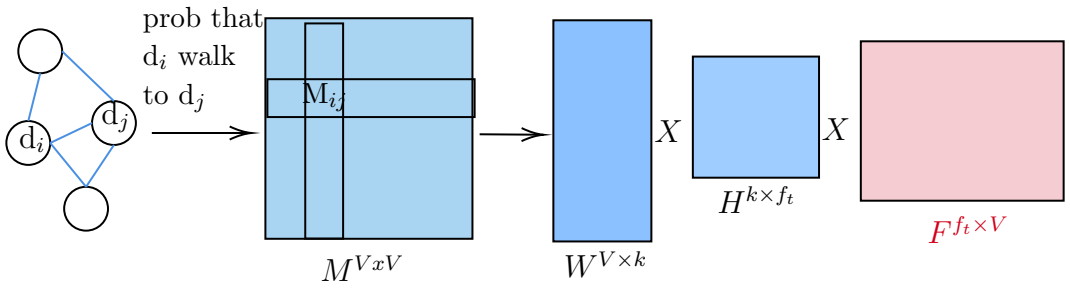


FIGURE 9: TADW Model

freely explore the network and can discover homophilous communities in the network. It is assumed while factorizing the likelihood that the observed nodes are independent of each other while learning the feature representation. Just like in the PV-DBOW model, the context nodes are considered to have no relationship with each other. In this algorithm, the content of the nodes is totally ignored. Therefore, the produced representation of the documents does not capture the content of the document.

Concatenating PV-DBOW and Node2vec Embeddings

To capture both the content and link information for learning document embedding, we can concatenate the embeddings produced from the above methods for the downstream tasks like classification. This method will not capture the complex relationship between the documents regarding content and linkage. After concatenation, the vector dimension will double, and we can use dimensionality reduction methods like PCA [41] to reduce the dimensions. During this process, we will lose some feature information which might downgrade our classification result.

2.2 Text Associated DeepWalk (TADW)

Paper [42] proposes an algorithm to include text information to the network representation learning (NRL) method like DeepWalk [33] using matrix factorization.

Algorithm

TADW algorithm tries to capture the linkage and content information together when learning the document representation. First, the content information is converted into vector representation using TF-IDF matrix or binary vector and represented as $F^{f_t \times V}$ where f_t is the number of dimensions we want to represent each word in the vocabulary V. Then, the model will construct matrix $M^{V \times V}$ where each entry M_{ij} is a probability of walking from the document d_i to d_j based on the sequence generated using sampling strategy like DeepWalk [33] as shown in Figure 9.

The goal is to approximate matrix $W^{V \times k}$ and $H^{k \times f_t}$ where k is the dimension of the vector we want to learn using matrix M and F through inductive matrix factorization [32].

$$min_{W,H} \sum_{i,j\in} \left(M_{ij} \; - \; \left(W^T HT\right)\right)^2 \; + \; \frac{\lambda}{2} \left(||W||_F^2 + ||H||_F^2\right) \tag{7}$$

The equation 7 shows that we try to approximate the matrix M through inductive matrix factorization. We try to learn the matrix W and H and use the feature matrix F to obtain better representation. λ is a regularizer which prevents the model from overfitting. Computation of M is very expensive and can take $O\left(|V|^2\right)$ times where V is the total number of documents [42].

Dataset Used

For experimentation, they used three datasets:

- Cora: It contains $2,708$ machine learning documents with $5,429$ citation links. Each document is represented by a binary vector of $1,433$ dimensions indicating the presence of the corresponding word
- Citeseer: It contains $3,312$ documents with $4,732$ citation links. Each document is again represented by a binary vector 0f $3,703$ dimension where each position

indicates the presence of the word

- Wiki: This dataset contains $2,405$ documents with $17,981$ links. The document is represented as TD-IDF matrix with $4,973$ columns

Experimentation

As an input for the experimentation, they reduced the vector dimensions by using SVD [23] decomposition and obtained a vector dimension of 200 for the TADW method. For classification, they used an SVM classifier. While training they used one-vs-rest classifier for each class and selected the class which has the maximum score. The experimentation result showed that TADW consistently performs much better than the other baseline methods and can also work well when there is a significant amount of noise in the graph.

Conclusion

This paper gives a proof that the DeepWalk algorithm is equivalent to the matrix factorization and using this concept they tried to incorporate the text information in the network representation learning method DeepWalk. The proposed method does not support online and distributed learning for large scale networks. It is also computationally expensive as the procedure of computing M takes $O|V^2|$, TADW, takes into account both the text and network information for learning the document embedding. Instead of just concatenating the vectors, it proposed a method of jointly modeling feature combination via matrix factorization. As a result, for paper similarity experiment they showed that when finding similar papers related to the category of theory. TADW found all papers belonging to the same topic whereas DeepWalk returned paper belonging to different class labels. Though the original query paper cited all the returned papers. This was because DeepWalk considered all the papers cited by the original query paper as equal as it did not have the text information.

This experiment proved that adding text information to the network embedding will learn better representation.

2.3 Linked Document Embedding for Classification

Paper [39] introduces an algorithm which tries to incorporate the linkage and label information between the documents along with the content information to the document embeddings.

Algorithm

In the algorithm, they try to capture the linkage and content information together while learning the document embedding. The algorithm consists of two parts:

1. learning word-word embedding for the documents using content only
2. learning document-document embeddings using the network only

$$max_{WD} \frac{1}{|P|} \sum_{w_i, w_j, d_k \in P} log\ P\ (w_j \mid w_i, d_k) \tag{8}$$

In the first part, for each target word w_i they extract c neighbours as w_j based on the window size. They also store the information from where this word was extracted by adding document ID. Together the pair of words and the document form a triplet (w_i,w$_j$, d_j) and stored in the set P. Objective function 8 tries to maximize the probability of the neighbor words w_j given the target word w_i and the document vector d_k. $W^{M \times s}$ and $D^{N \times s}$ are the weight matrices that we want to learn. M is the total number of words in the vocabulary, and N is the total number of documents

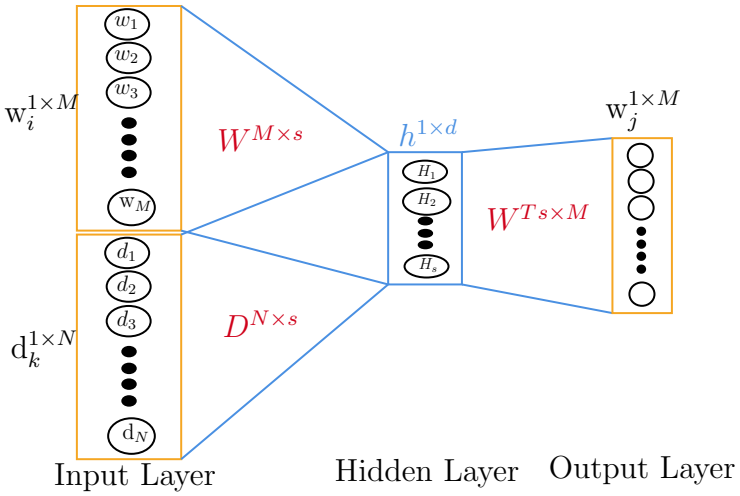


FIGURE 10: Learning Document Representation using Content Information

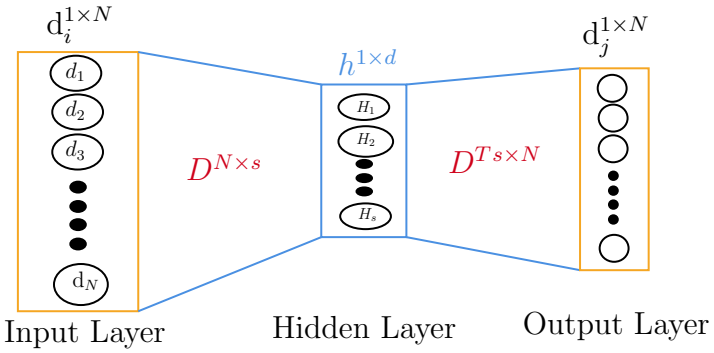


FIGURE 11: Learning Document Representation using Network Information

in the corpus. s is the dimension of the vectors. Figure 10 shows an outline of the model.

$$max_D \frac{1}{|E|} \sum_{i=1}^{N} \sum_{j:e_{ij}=1} log\ P\ (d_j \mid d_i) \tag{9}$$

In the second part, they learn the document embeddings using the graph $G(N, E)$ where N is a total number of documents and E represents an edge between d_i and d_j. The objective is to maximize the probability of the neighbouring nodes in G when the source node d_i is given. The equation 9 defines this concept where E is the set containing document-document pair based on citation information and $e_{ij} = 1$ only if d_i cites d_j. Figure 11 gives an overview of the model.

$$\frac{1}{|P|} \sum_{(w_i,w_j,d_k)\in P} log\ P\ (w_j|w_i,d_k)\ +\ \frac{1}{|E|} \sum_{i=1}^{N} \sum_{j:e_{ij}=1} log\ P\ (d_j|d_i)\ -\gamma(W,D) \quad (10)$$

Equation 10 gives an overview of the model. In the first part, we try to learn the document representation using only the content information, and in the second part, we use network information to learn the document representation. γ is a regularizer so that the model does not overfit.

For experimentation, we use source code build by Yi Zhang as the authors did not provide the code.

Dataset Used

For experimentation, they used two datasets DBLP and BlogCatalog. They extracted only title and abstract as content information for the DBLP dataset and chose six categories for the experimentation. Each group consisted of $2,550$ samples and in total $15,300$ documents were used with $36,359$ total links.

In the case of the BlogCatalog dataset, they used text description in each blog as the content and determined links according to the related blogs mentioned by the website. They also assumed the presence of the link between two blogs if the authors of the blogs followed each other. In total, the dataset contained $62,652$ documents with $378,161$ links. This dataset is unbalanced.

Experimentation

For experimentation, they used $F1micro$ and $F1macro$ as an evaluation metrics. They compared the proposed method with the other existing methods like PV-DM, PV-DBOW, and PTE [35]. Their experiment consisted of two-phase; the representation learning phase and the classification phase. During the representation learning

phase, they used the whole dataset to train word embeddings and document embeddings. Label information of the training data is also used for learning whereas the labels of the testing data are not included during the learning phase. The used dimension size of 100 and window size of 7 and the number of negative samples equal to 7.

Conclusion

The result of experimentation proves that the inclusion of link information to the document embeddings will improve the classification result. They also experimentally showed the effect of link density on the classification task, and as the density increases, the classification accuracy also increased. In this paper, they did not test the embeddings against other baseline experiment methods like document recommendation or embedding visualization. In the proposed method, it is also assumed that the weight of links is uniform for all the edges. In the real world, it is not the case, as the documents might have weighted edges between them in the form of page rank values. As a result, this method will not be able to capture that information.

CHAPTER 3

Retrofit Algorithm

In this section, we will formally define the problem statement and explain our algorithm in detail.

3.1 Problem Statement

Given the content based embedding of the document $\hat{d_i}$, we want to learn the new embedding of document d_i such that it is close to its original embedding and also to the embeddings of the documents that it is linked with.

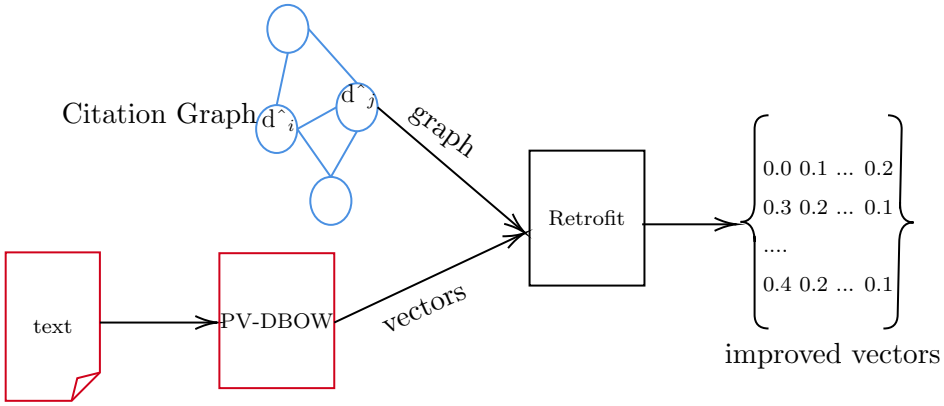


FIGURE 12: Retrofit Input

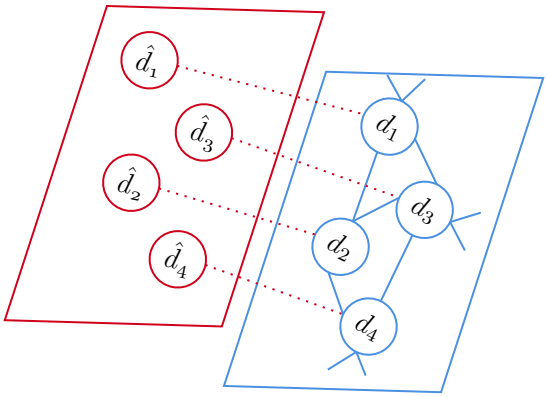


FIGURE 13: Retrofit Algorithm

3.2 Proposed Algorithm

The proposed method Retrofit will try to bring document embeddings close together based on citation and network information. The Retrofit takes vectors produced from PV-DBOW [27] as an input. The vectors from PV-DBOW lack knowledge from the citation graph. The Retrofit will also take the citation graph as input and will add network information to the PV-DBOW vectors as shown in Figure 12.

Let $V = \{d_1, d_2, ..., d_n\}$ be a set of n documents and G be a graph that captures citation links between the documents in V. We represent G as an undirected graph (V, E) with one vertex for each document type and edge $(d_i, d_j) \in E \subset V \times V$ indicating citation relationship between the two documents.

Let $\hat{\mathbf{D}} = (\hat{\mathbf{d}}_\mathbf{1}, \hat{\mathbf{d}}_\mathbf{2}, ..., \hat{\mathbf{d}}_\mathbf{n})^\mathbf{T}$ be the matrix for the vector representation of each document $\hat{\mathbf{d}}\_\mathbf{i} \in \Re^k$ for each $d_i \in V$ learned using standard document embedding representation methods where k is the dimension of the vector. Our aim is to learn matrix $\mathbf{D} = (\mathbf{d}_\mathbf{1}, \mathbf{d}_\mathbf{2}, ..., \mathbf{d}_\mathbf{n})$ such that d_i is close to $\hat{d}_i$ and to the citing documents in G.

The objective is to minimize the distance between the document d_i, and the edges it is connected with in the citation graph $(d_i, d_j) \in E$. The document $\hat{d}_i$ represents the original embeddings, and we want to bring the retrofitted embedding close to the original embedding and to its neighbor as shown in Figure 13. Since there is no closed form solution [22] of the objective function, we will use an iterative method

to solve this problem. This is because D can contain $10,000,000$ documents and a dense 100 dimensional vector can represent each document. For finding a closed-form solution, we need to store DD^T matrix which will require 1×10^{10} floating point numbers. At 8 bytes per number, this comes to $80GB$ which is impractical to store on anything but a supercomputer. Furthermore, computing inverse of this matrix would also be very expensive. We have two variants of the retrofit algorithm called retrofit1 and retrofit2. The difference between these two methods is how we add the citation information to the document embeddings.

$$J(D) \; = \; \sum_{i=1}^{N} \left[(1 \; - \; \beta) || \; \mathbf{d_i} \; - \; \hat{\mathbf{d_i}} \; ||_2^2 \; + \beta \sum_{(i,j) \; \in \; E} ||\mathbf{d_i} \; - \; \mathbf{d_j} \; ||_2^2 \right] \tag{1}$$

Retrofit1

Equation 1 defines the objective function of retrofit1 where β is a hyperparameter. The boldface lower case letters represent vector representation of the documents. J in Equation 1 is a convex in D [22]. The hyperparameter β controls the relative strength of association in Equation 1. This objective function gives more weight to nodes with high degrees. The weight beta loops over the links and adds them to the document vector. This weight controls how much we want the document embedding to come close to its neighbors. This approach is modular meaning it can be applied to any document vector representation obtained from any model. We use Squared Euclidean distance (SED) [22] to define the distance between the pair of vectors. First, we shall define the Euclidean distance [21] between $\mathbf{d_i}$ and $\hat{\mathbf{d}}_\mathbf{i}$ in Equation 2. k is the dimension of the vector.

$$||\mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i} \; ||_2 \; = \; \sqrt{(\mathbf{d_{i1}} - \hat{\mathbf{d}}_\mathbf{i1})^2 + (\mathbf{d_{i2}} - \hat{\mathbf{d}}_\mathbf{i2})^2 \; + \; ..... \; + \; (\mathbf{d_{ik}} - \hat{\mathbf{d}}_\mathbf{ik})^2} \tag{2}$$

Equation 2 can be further simplified to 3.

$$||\mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i}||_2 \;=\; \sqrt{\sum_{x=1}^{k} \left(\mathbf{d_{i_x}} - \hat{\mathbf{d}}_{\mathbf{i_x}}\right)^2} \tag{3}$$

We use SED to remove the square root and make calculations less expensive. SED is a smooth and strictly convex function of the two points and is preferred in optimization theory [20]. Equation 4 measures the similarity between retrofitted embedding $\mathbf{d_i}$ and initial content-based vector $\hat{\mathbf{d}}_\mathbf{i}$ using SED.

$$||\mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i}||_2^2 = \sum_{x=1}^{k} \left(\mathbf{d_{i_x}} - \hat{\mathbf{d}}_{\mathbf{i_x}}\right)^2 \tag{4}$$

The objective function 1 refers to the task of minimizing the distance of document vectors based on content and citation network. The derivative of the objective function will give us an updating equation for the change in vector $\mathbf{d_i}$ to reduce the distance between the content based vector $\hat{\mathbf{d}}_\mathbf{i}$ and the citing papers $\mathbf{d_j}$, $\forall j$ where $(i, j) \;\in E$ [24]. This derivative specifies how much change in the input will scale to the change in the output. The derivative is useful in determining the change in $\mathbf{d_i}$ to make a small improvement in the output. Since our objective function has multiple variables, we take the partial derivative to measure the change concerning the specific component. Now we take the partial derivative of our objective function 1 with respect to d_i and obtain the updating equation for the new vector $\mathbf{d_i}$.

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; \sum_{i=1}^{N} \left[(1-\beta)||\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2 \;+\beta \sum_{(i,j)\;\in\; E} ||\mathbf{d_i} \;-\; \mathbf{d_j} \;||_2^2 \right] \tag{5}$$

First we take the derivative of $||\mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i}||_2^2$ and $\sum_{(i,j)\;\in\; E} ||\mathbf{d_i} \;-\; \mathbf{d_j} \;||_2^2$ with respect to

$\mathbf{d_i}$

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; \sum_{i=1}^{N} \left[(1-\beta) \frac{\partial}{\partial \mathbf{d_i}} || \;\mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2 \;+ \beta \frac{\partial}{\partial \mathbf{d_i}} \sum_{(i,j) \;\in\; E} ||\mathbf{d_i} \;-\; \mathbf{d_j} \;||_2^2 \right] \tag{6}$$

We will use chain rule [19] to take the derivative of $|| \;\mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2$ which can be written as $h(g(x))$ where h is the $||_2^2$ as defined in 4 and $g(x) = \mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i}$. First we will take the derivative of the $||_2^2 4$

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; \sum_{i=1}^{N} [2(1-\beta) \left(\;\mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;\right) \; \frac{\partial}{\partial \mathbf{d_i}} \left(\;\mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;\right) + 2\beta \sum_{(i,j) \;\in\; E} (\mathbf{d_i} \;-\; \mathbf{d_j} \;) \frac{\partial}{\partial \mathbf{d_i}} (\mathbf{d_i} \;-\; \mathbf{d_j} \;)] \tag{7}$$

and then the derivative of $g(x)$.

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; 2 * (1-\beta) \left(\mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i} \right) (1) \;+\; 2\beta \sum_{(i,j) \;\in\; E} (\mathbf{d_i} - \mathbf{d_j}) \tag{8}$$

Now lets suppose $\frac{\partial J(D)}{\partial \mathbf{d_i}} = 0$:

$$(1-\beta)(\mathbf{d_i}) - (1-\beta) \left(\hat{\mathbf{d}}_\mathbf{i} \;\right) \;+\; \beta \sum_{(i,j) \;\in\; E} (\mathbf{d_i}) - \beta \sum_{(i,j) \;\in\; E} (\mathbf{d_j}) \;=\; 0 \tag{9}$$

We will move all the $\mathbf{d_i}$ terms to one side and all the other terms to another side.

$$(1-\beta)(\mathbf{d_i}) \;+\; \beta \sum_{(i,j) \;\in\; E} (\mathbf{d_i}) = \beta \sum_{(i,j) \;\in\; E} (\mathbf{d_j}) \;+ (1-\beta) \left(\hat{\mathbf{d}}_\mathbf{i} \;\right) \tag{10}$$

Then we will take $\mathbf{d_i}$ as common and divide its coefficient where $\sum_{(i,j) \in E} \mathbf{d_i} = deg(d_i)(\mathbf{d_i})$

$$\mathbf{d_i} \;=\; \frac{\beta \sum_{(i,j) \;\in\; E} (\mathbf{d_j}) \;+\; (1-\beta) \left(\hat{\mathbf{d}}_\mathbf{i} \;\right)}{(1-\beta) \;+\; \beta \times deg(d_i)} \tag{11}$$

Input	Output
$f(x_1^0, x_2^0, x_3^0, ..., x_n^0)$	x_1^1
$f(x_1^1, x_2^0, x_3^0, ..., x_n^0)$	x_2^1
$f(x_1^1, x_2^1, x_3^0, ..., x_n^0)$	x_3^1
.....	.....
$f(x_1^{m+1}, x_2^{m+1}, x_3^{m+1}, ..., x_n^m)$	x_3^{m+1}

TABLE 1: Example of the Gauss-Seidel method. m is the number of step. The complete table shows one iteration. For each step in a single iteration, we will always use the latest value

Equation 11 is an online update equation we use in each iteration to obtain the vector representation of d_i. For updating d_i, we use an iterative updating method as used in [20]. In [20] they try to build a graph for data points containing labeled and unlabeled points. Known labels are used to propagate information in the graph in order to label all the nodes using edges as a similarity. They also used an iterative updating method to update the label information in the graph. Our approach uses Gauss-Seidel as implemented in [20] to solve the system of linear equations. The system of linear equations is the updating equation for each d_i in D. The Gauss-Seidel method works by using the latest value in D for each iteration. For example, $x = (x_1, x_2, ...x_n)$ is the true solution of x. If x_1^{m+1} is better then x^m to approximate $x_2, x_3, ..., x_n$, then we will use the new value x_1^{m+1} rather than x^m. m is the number of steps. For finding x_2^{m+1}, instead of using the old value x_1^m, we will use x_1^{m+1} and the subsequent old values $x_3^m, x_4^m, ..., x_n^m$. Similarly, for finding x_3^{m+1}, we will use x_1^{m+1} and x_2^{m+1} and the subsequent $x_4^m, x_5^m, ..., x_n^m$. Table 1 shows how the latest value of x is used in each step for a single iteration.

Algorithm 1: Retrofit1

Result: Adds citation information to the content based vector representation of documents in the matrix $\hat{\mathbf{D}}$

Input : $\hat{\mathbf{D}}$, $G = (V, E)$, *iterations*, β

Output: $\mathbf{D}$

1 $\mathbf{D} = \hat{\mathbf{D}}$
2 **for** *iter=0* **to** *iterations* **do**
3 **foreach** $\mathbf{d_i}$ *in* $\mathbf{D}$ **do**
4 $\mathbf{v_i} = (1\text{-}\beta) * deg(\mathrm{d}_i)$ * $\hat{\mathbf{d}}_\mathbf{i}$
5 **foreach** $\mathbf{d_j}$ *in* $E(d_i)$ **do**
6 $\mathbf{v_i} = \mathbf{v_i} + \beta$ * $\mathbf{d_j}$
7 $\mathbf{d_i} = \mathbf{v_i}$ / (2*deg(d_i))
8 **end**
9 **end**
10 **end**

Algorithm 1 links the objective function 1 and shows how we implemented the updating step 11 in the code. In algorithm 1 $\hat{\mathbf{D}}$ is a matrix representing content based vector representation of N documents. $E(d_i)$ will give us a list of all the documents that are connected with d_i in the network graph. *Iteration* is a hyper-parameter which will determine how many times we want to repeat this process. $deg(d_i)$ is the degree of the document d_i in the citation graph. Line 4 of the algorithm 1 will add the information from content based embedding and line 6 will add information from the network view. Line 7 takes the average of the content and network based embedding for each d_j. The computational complexity of retrofit1 is (tn) where t is the total number of neighbours and n is the total number of documents in the dataset.

```
def retrofit1(D_hat, G, iterations, beta):

  D = D_hat
```

```
for it in iterations:
   for d_i in D:
      numNeighbours = len(G(d_i))
      if numNeighbours == 0:
         continue
      v_i = (1-beta)*numNeighbours*D_hat[d_i]
      for d_j in G(d_i):
         v_i += beta*D[d_j]
         D[d_i] = v_i / (2*numNeighbours)
return D
```

Listing 3.1: Retrofit1 Code. Complete code can be found at `https://github.com/ZeeshanMansoor260/Masters/blob/master/Retrofit/retrofit_document.py`

Code 3.1 shows the implementation of retrofit1 in Python. The complete code can be accessed at `https://github.com/ZeeshanMansoor260/Masters/blob/master/Retrofit/retrofit_document.py` and we originally obtained this code from [22]. G is the citation graph and $G(d_i)$ will return the list of documents that are connected with d_i in the network.

Retrofit2

$$J(D) = \sum_{i=1}^{N} \left[(1 - \beta) || \mathbf{d_i} - \hat{\mathbf{d_i}} ||_2^2 + \beta || \mathbf{d_i} - \frac{1}{deg(d_i)} \sum_{(i,j) \in E} \mathbf{d_j} ||_2^2 \right] \quad (12)$$

Another variant of the retrofit algorithm called retrofit2 is defined in 12. Retrofit2 first takes an average of all the documents that it is citing and then add that to the content using β as the weight for network information and $deg(d_i)$ is the degree of the document d_i. We use SED 4 to calculate the distance between the embeddings.

For retrofit2, we will again take the first derivative of the objective function 12 and equate it to zero to obtain the equation for updating the vectors as done for

retrofit1.

The objective function for retrofit 2 is defined as:

$$J(D) \;=\; \sum_{i=1}^{N} \left[(1-\beta) \;||\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2 \;+\beta||\; \mathbf{d_i} \;- \frac{1}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;||_2^2 \right]$$

We take the partial derivative of $J(D)$ with respect to $\mathbf{d_i}$

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; \sum_{i=1}^{N} \left[(1-\beta) \;||\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2 \;+\beta||\; \mathbf{d_i} \;- \frac{1}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;||_2^2 \right] \tag{13}$$

Now we take the derivative of $||\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2$ and $||\mathbf{d_i} \;-\; \frac{1}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;||_2^2$

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; \sum_{i=1}^{N} \left[(1-\beta) \frac{\partial}{\partial \mathbf{d_i}} ||\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2 \;+\beta \frac{\partial}{\partial \mathbf{d_i}} ||\mathbf{d_i} \;-\; \frac{\partial}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;||_2^2 \right] \tag{14}$$

Again $||\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;||_2^2$ can be considered as $h(g(x))$ and we will apply chain rule to take it derivation. First we take the derivative of the $||_2^2$ 4.

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} \;=\; \sum_{i=1}^{N} [2(1-\beta) \left(\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;\right) \; \frac{\partial}{\partial \mathbf{d_i}} \left(\; \mathbf{d_i} \;-\; \hat{\mathbf{d}}_\mathbf{i} \;\right)$$
$$+2\beta \left(\mathbf{d_i} \;-\; \frac{1}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;\right) \frac{\partial}{\partial \mathbf{d_i}} \left(\mathbf{d_i} \;-\; \frac{1}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;\right)]$$

Then we take the derivative of the $g(x)$

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} = 2 * (1-\beta) \left(\mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i} \;\right) (1) \;+\; 2\beta \left(\mathbf{d_i} \;-\; \frac{1}{deg(d_i)} \sum_{(i,j)\;\in\; E} \mathbf{d_j} \;\right) (1) \tag{15}$$

After that we will cancel out 2

$$\frac{\partial J(D)}{\partial \mathbf{d_i}} = (1-\beta)\left(\mathbf{d_i} - \hat{\mathbf{d}}_\mathbf{i}\right) + \beta\left(\mathbf{d_i} - \frac{1}{deg(d_i)}\sum_{(i,j)\in E}\mathbf{d_j}\right) \tag{16}$$

Now lets suppose $\frac{\partial J(D)}{\partial \mathbf{d_i}} = 0$

$$(1-\beta)(\mathbf{d_i}) - (1-\beta)\left(\hat{\mathbf{d}}_\mathbf{i}\right) + \beta(\mathbf{d_i}) - \frac{\beta}{deg(d_i)}\sum_{(i,j)\in E}\mathbf{d_j} = 0 \tag{17}$$

We will move all the $\mathbf{d_i}$ terms to one side

$$(1-\beta)(\mathbf{d_i}) + \beta(\mathbf{d_i}) = (1-\beta)\left(\hat{\mathbf{d}}_\mathbf{i}\right) + \frac{\beta}{deg(d_i)}\sum_{(i,j)\in E}\mathbf{d_j}$$

After simplifying the left hand side will obtain the following update equation

$$\mathbf{d_i} = (1-\beta)\left(\hat{\mathbf{d}}_\mathbf{i}\right) + \frac{\beta}{deg(d_i)}\sum_{(i,j)\in E}\mathbf{d_j} \tag{18}$$

We will use Gauss-Seidel [20] method for updating the vectors. For each $\mathbf{d_i}$ we

will use the latest vectors in $\mathbf{D}$ and after 10 iterations it will converge [22].

Algorithm 2: Retrofit2

Result: Adds citation information to the content based vector representation of documents in the matrix $\hat{\mathbf{D}}$

Input : $\hat{\mathbf{D}}$, $G = (V, E)$, *iterations*, β

Output: $\mathbf{D}$

```
1  D = D̂
2  for iter=0 to iterations do
3      foreach d_i in D do
4          netvec = 0
5          foreach d_j in E(d_i) do
6              netvec += d_j
7          end
8          netvec = netvec / deg(d_i)
9          d_i = (1 − β) d̂_i + β(netvec)
10     end
11 end
```

We will now link algorithm 2 with the update equation 19.

$$\mathbf{d_i} = (1-\beta)\left(\hat{\mathbf{d}}_\mathbf{i}\right) + \frac{\beta}{deg(d_i)} \sum_{(i,j) \in E} \mathbf{d_j} \tag{19}$$

In 19 $(1-\beta)\hat{\mathbf{d}}_\mathbf{i}$ is obtaining information from the content view and implemented in line 9 of the algorithm 2. $\frac{\beta}{deg(d_i)} \sum_{(i,j) \in E} d_j$ is getting information from the network view and implemented from line 5 to 8 of the algorithm 2. Information from both the views is then combined in line 9 as shown in the algorithm 2. The computational complexity of retrofit2 is (tn) where t is the total number of neighbours and n is the total number of documents in the dataset.

```
def retrofit2(D_hat, G, iterations, beta):
   D = D_hat
   for it in iterations:
      for d_i in D:
         numNeighbours = len(G(d_i))
         if numNeighbours == 0:
            continue
         netvec = np.zeros(D.dim)
         for d_j in G(d_i):
            netvec += beta*D[d_j]
         netvec = netvec / (numNeighbours)
         D[d_i] = (1-beta)*numNeighbours*D_hat[d_i] + netvec
   return D
```

Listing 3.2: Retrofit2 Code

The code 3.2 is the implementation of the retrofit2 algorithm in Python. The complete code can be found at `https://github.com/ZeeshanMansoor260/Masters/blob/master/Retrofit/retrofit_document_adv.py`. G is the citation graph and $G(d_i)$ will return the list of documents that d_i is connected with in the network.

3.2.1 Code Flow

This section will explain the flow of code from the point of learning embeddings from text to classifying evaluation. The underlying retrofitting code is obtained from [22]. We converted the code from word embeddings to the document embeddings. The code architecture is decoupled into several components. A single program called Run.py launches all components. Upon running this file, it will trigger the content based embedding learning module. This module uses PV-DBOW to learn vector representation from the content only. Run.py will also launch the network module

to obtain the linkage information from the network. The information from both the network and the content will then be passed to the retrofit module. The retrofit will try to add the linkage information to the content based embeddings using the algorithms defined above. Once the embeddings are retrofitted, then they will be passed to the classification and visualization module for evaluation. This code can be accessed at `https://github.com/ZeeshanMansoor260/Masters`.

CHAPTER 4

Datasets

This section will explain the four datasets we used for our experimentation in detail.

4.1 DBLP

First, we will outline the details regarding the Digital Bibliography and Library Project DBLP dataset [dbl]. For initial experimentation, we used a small dataset and classified the papers based on the conferences. We extracted documents from DBLP which is a computer science bibliography provided by the University of Trier in Germany. From DBLP published venue metadata, we can find papers that belong to specific conferences. We used three popular conferences: VLDB, SIGMOD, and ICSE. The details of the three conferences are listed below:

- VLDB: The International Conference on very Large Databases constitutes of research papers focusing on research in the field of database management [vld].
- ACM SIGMOD: The International Conference on Management of Data [sig] contains research papers focusing on principles, techniques and applications of database management systems and data management technology.
- ICSE: The International Conference on Software Engineering [ics] comprises of all the papers that contain most recent innovation, trends, experiences and concerns in the field of software engineering.

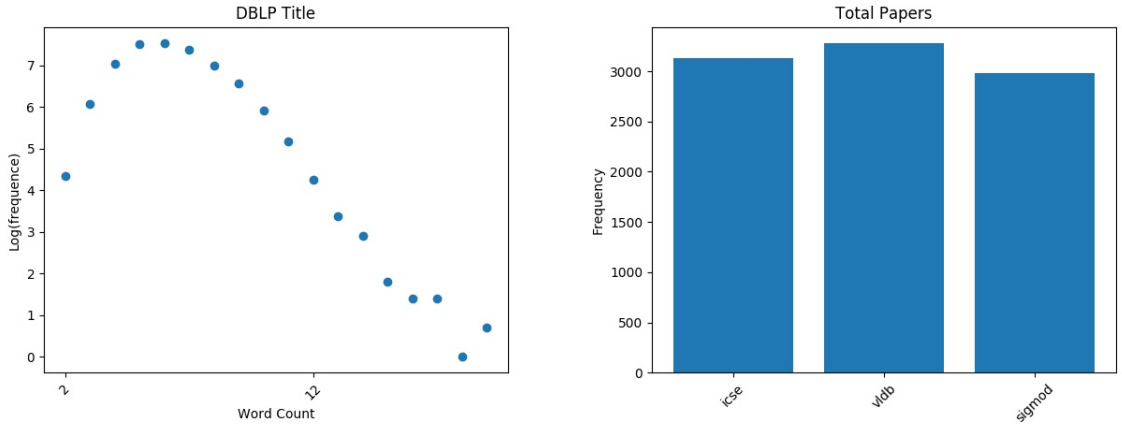


FIGURE 14: Document size and classes distribution in the DBLP dataset

In the rest of our discussion, we will use DBLP to refer to the combined dataset consisting of the above three conferences.

The DBLP dataset consisted of title only, and the average text length is 6.4 words per document. For experimentation, we wanted to have more content and see the effect of adding network information to the vector representation consisting of content information only. We combined the DBLP dataset with the Aminer [37] to increase the content. The Aminer dataset has more details like title, abstract, authors and references. This dataset contains citation information extracted from the Microsoft Academic Graph (MAG), ACM and DBLP. We used version 10 of the Aminer dataset which includes 3 million papers. Then we merged the DBLP dataset with the Aminer dataset using the title to link between these two datasets. Once connected, we added abstract information to the DBLP dataset and called this dataset as $DBLP_{adv}$. We were able to link 6704 documents, and the average text length increased to 100.5.

Figure 14 shows the document size and label statistics of the DBLP dataset. We can see that all the 3 classes are uniformly distributed and most of the documents have title length of 6.4.

Figure 15 shows the distribution of words and labels for DBLPadv dataset. In Figure 15 we can observe that the average document length is 100 words and the

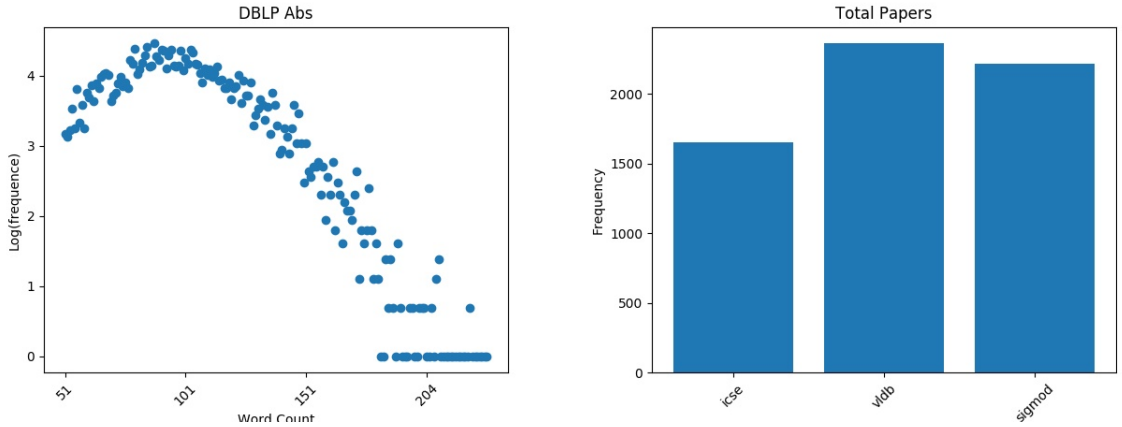


FIGURE 15: Document size and classes distribution in the DBLPadv dataset

class distribution of the ICSE documents has decreased. This is because we could not link some of the documents with the Aminer dataset when trying to extract abstract information.

4.2 ArXiv

The following sections explain how we created the arXiv [arx] dataset and the method we used for adding reference information.

4.2.1 ArXiv Content Extraction

arXiv [arx] is an open-access containing papers in the domain of Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance, Statistics, Electrical Engineering, and Economics. arXiv dataset stores the source code of the documents in the form of Latex format.

We are interested in the source code as it gave us more freedom to extract the different type of information like titles or abstract from each document. We downloaded the Latex files from 1997 to 2018 from the Amazon S3 servers. Each tar file consisted of around $500MB$, and there were total 190 tar files. The total size of the

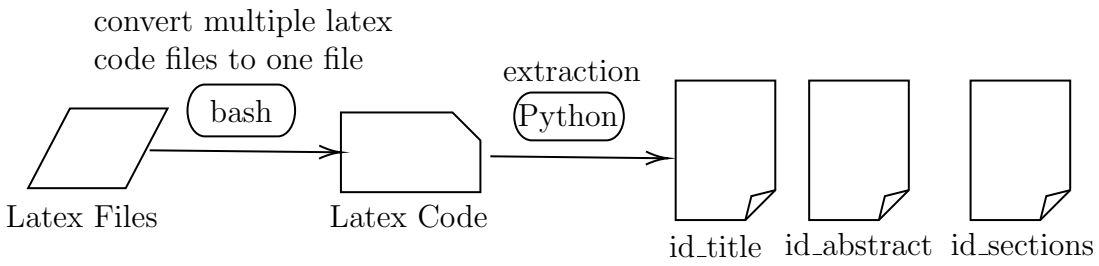


FIGURE 16: arXiv Extract FlowChart

complete dataset is approximately $600GB$.

The tar files are named using the following convention:

yymm.chunk

where yy represents the year, mm represents the month in which the paper was published and chunk as the number of the tar file. For example, the tar file named 1401.01 means that the papers in this tar were published in January 2014. The chunk 01 means the number of this tar file as there are several chunks of each tar files.

Once extracted, each document's filename is it's paperID, and the complete source code is present inside that file. For papers which contains several files, the directory's name is the paperID, and all of its files are present in that directory.

Initially, to extract title and paperID from this dataset, we developed a bash script in Linux. A bash script is a text file which can contain Linux commands and will execute them when running. This bash script will search for the title in all the files using an awk command and stores it in another file containing paperID and title only. Similarly, we extended this method for extracting abstract, introduction section, middle section and last section from all the papers and linked them with their paperIDs. This method we found to be very slow and subsequently, we modified the bash scripts such that, it only finds the tex file which contains the main tex code for all the documents and copies it to another directory. Then a Python script will search for title, keywords, abstract and introduction section in the tex source code

Title Extraction

\title [class] % comments { title \textbf{ something to bold } remaining title \}

FIGURE 17: Title Format in Latex

Title Example

Scenario 1: \title {Numerical Methods for Quasi-Periodic Incident Fields Scattered by Locally Perturbed Periodic Surfaces}
Scenario 2: { \textbf{On the diameter and incidence energy of iterated total graphs}}
Scenario 3: \title{Dynamics of Nonlinear Random Walks on Complex Networks \thanks{Submitted to the editors DATE.

FIGURE 18: Different scenarios for extracting title

using regex expressions and store them in a separate file according to the following format

PaperID Title

Within the title, there might be other latex tags which we do not want to acquire like comments or class as shown in figure 17. To counter that, we used a stack to identify the beginning of the title using {. Whenever we saw { we increased our stack and whenever we saw } we decreased our stack until it is empty. As a result, we were able to capture the title of the document only and ignore other information like comments. For example the figure 18 shows different scenarios from which we need to extract title. Scenario 1 is the most straightforward situation. Using regex expression, we detect \title and add { to the stack and start reading. When we see } we remove it from the stack, and if the stack is empty, we stop reading. Scenario 2 is more complicated. Instead of \title, it uses \textbf, and we also search for this alternative. Situation 3 is the most complicated. It does not have a } bracket at the end. The stack will not be empty, and we will continue to read assuming it is a title.

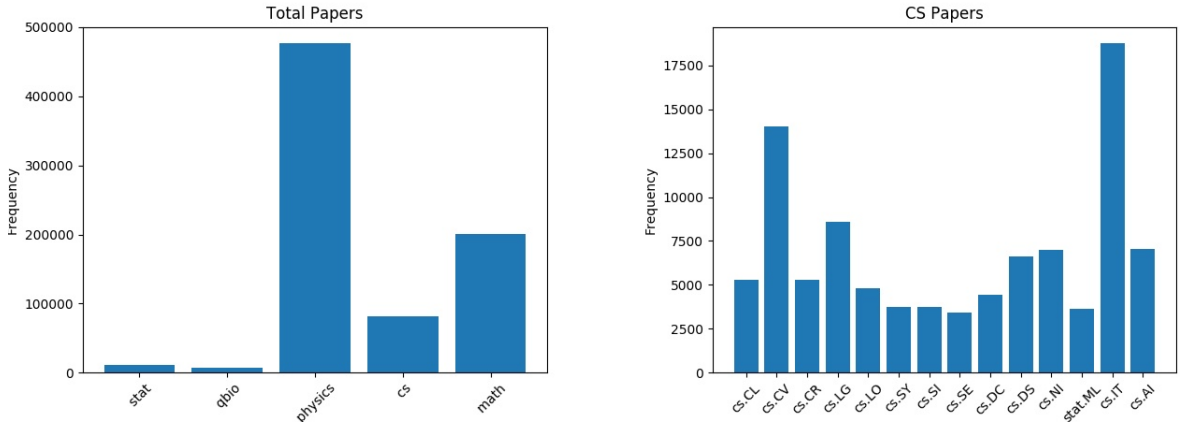


FIGURE 19: Statistics of classes from different domains and within CS papers

Our algorithm assumes that the title is not more than 3 lines and if we reach that limit we stop reading.

From this dataset, we were able to obtain title, abstract, keywords and the complete content of documents. We were also able to recover label information for the documents.

The label information is stored in a separate file in the following format

PaperID Label

The labels cover topic like Physics, Computer Science and also subject classes covering specific topic, for example information theory or computer vision in a separate file.

Number of Documents	Number of Documents with Lables	Number of Documents with Keywords	Year Span
1.3 Million	0.33 Million	0.36M	1990-2018

TABLE 2: ArXiv Dataset Statistics

Table 2 gives an overview about the whole dataset. We found 1.3M documents which has title and abstract along with the content information. We were able to link 0.36M documents which has keyword information. 0.33M documents has label information which can be used for classification and training the document embeddings. Figure 19 gives statistics of classes of the papers ranging from different domains like

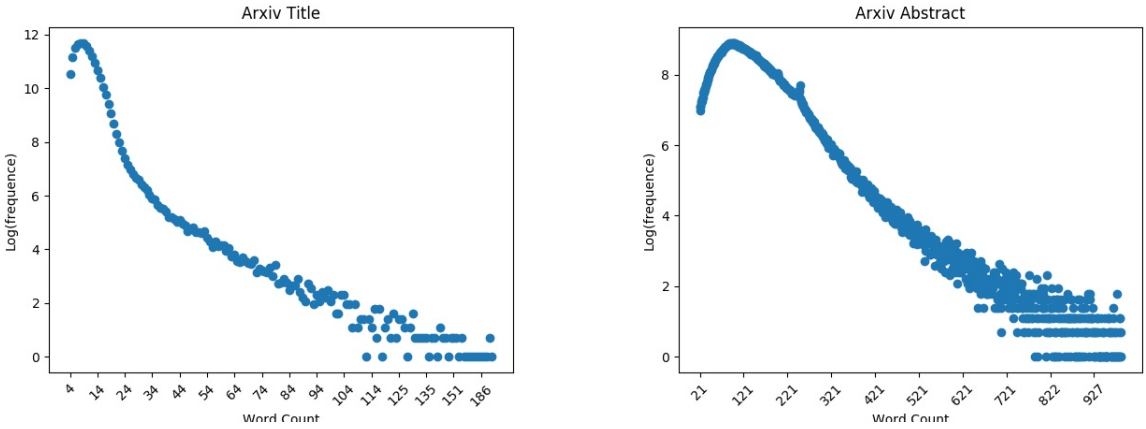


FIGURE 20: Document Size Distribution

Physics or CS. It also shows the class statistics for different topics within the CS papers.

Figure 20 shows the document size distribution. On x-axis is the word count and y-axis is log of the number of documents. In arXiv we can observe majority of the documents have length from 7 to 10 whereas in documents with abstract have mostly 111 total words.

Format	**Average Length Words**
title	9
abstract	173
intro	636

TABLE 3: ArXiv Text Length

The table 3 gives average text length of title, abstract and introduction section. The values present are calculated before stemming and tokenizing the text.

4.2.2 arXiv References

From the content of the documents it was very difficult to link the documents based on the references due to the following reasons

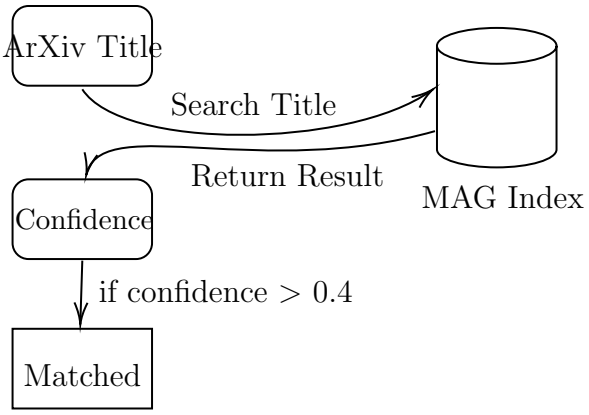


FIGURE 21: Matching Architecture for arXiv and MAG

- Different papers follow different reference formats
- Can not link the paper by just obtaining the title from the reference tags. Must use other information like authors or venue for the linkage
- Usually last names of the authors are written in references and difficult to detect it

Due to these reasons we decided to use Open Academic Graph (OAG) [37] dataset which contains $116M$ documents. In this dataset, they have title, abstract, authors, reference information for all the papers. This dataset contains paper from the Microsoft Academic Graph and Aminer. The paper covers a wide range of topics ranging from Biology to Physics. Our focus is only to get Computer Science papers so we wanted to get linkage information from the OAG dataset for the arXiv dataset. The figure 21 gives an outline of the matching architecture we used between the arXiv and MAG dataset for reference. We first extracted the title from the OAG dataset and then indexed it using Lucene [Luc] along with their paperID. Then for each arXiv document we searched it in the OAG index and got only the topmost result. Then we used sets to compare the search result with the original arXiv title and calculated the confidence of our matching based on the equation 1

$$confidence = \frac{searchResult.size()}{arXivTitle.size()} \quad (1)$$

We used confidence=0.4 because we did not tokenize or stemmed the title. The reason behind is that in the arXiv title, there were many latex tags present in the title and after preprocessing we would have lost much information in the title. We tried removing the latex tags like $or % using Detex [det] library but in some cases, it removed the whole title altogether. The main objective of most of these libraries is to remove latex tags from the whole document and these libraries used latex hierarchy to understand the architecture. In our case, we only wanted to remove tags from the title so these libraries did not function properly because they could not find the latex hierarchy. We also did not remove stops words because in this case, they played a pivotal role in identifying the strings. The main words in the topic might be different because we did not stem or tokenized them but the stops words remained the same in both the datasets so when calculating the confidence they proved to be a very good indicator. The output from this process is in the following format

MAGID arXivID Confidence

The intuition of adding confidence in the output is that if we want papers which are correctly matched with confidence 1.0, we can distinguish them.

Confidence 1.0

arXiv Title Thermal entanglement of qubit pairs on the Shastry-Sutherland lattice

OAG Title Thermal entanglement of qubit pairs on the Shastry-Sutherland lattice

The matching result with confidence 1 shows the output of our matching algorithm has a perfect match. Whereas with confidence 0.4, we are still able to match successfully though we are missing some part of the title in the arXiv.

Confidence 0.4

arXiv Title on punctured pragmatic space-time codes

OAG Title On Punctured Pragmatic Space-Time Codes in **Block Fading Channel**

Confidence	Nodes	Edges
0.8−1.0	534466	2913316
0.4−1.0	650797	4254848

TABLE 4: arXiv Network Statistics

We created two types of citation graphs based on the confidence of the documents matched between arXiv and MAG dataset. In the first type, we used documents with a confidence level of 0.8 and above. This citation graph is more accurate but contains only 534466 nodes and 2913316 edges as stated in the table 4. In the second citation graph, the confidence constraint is more relaxed. We selected documents whose matching confidence was 0.4 or more. In this graph, there are 650797 nodes and 4254848 edges. Both the graphs are undirected graphs and retrieved from the MAG dataset.

4.2.3 arXiv Labels

Dataset	Classes
ArxivTitle,ArxivAbstract	maths,physics,stat,cs
ArxivCSTitle,ArxivCSAbstract	Information Theory (cs.IT), Distributed Computing (cs.DC), Machine Learning (cs.LG), Computer Vision (cs.CV), Computational Complexity (cs.CC)

TABLE 5: arXiv Dataset Labels

For experimentation, we created two types of the dataset based on the labels.

The first dataset is called Arxiv which is based on documents belonging to different domain areas like physics or maths. The table 5 outlines the types of documents we have in the Arxiv dataset. Another dataset, that we created is called ArxivCS. In this dataset, we selected all the documents belonging to the Computer Science area and explicitly belonging to the topics as outlined in the table 5. The reason we selected these particular classes was based on the number of documents each class had in the dataset.

4.3 Dataset Overview

Dataset	**# Documents**	**Vocab Size**	**Avg Text Length**	**Avg Degree**
DBLP	9396	8036	6.4	8.0
DBLP_{adv}	6704	15932	100.5	7.7
ArxivTitle	40000	20737	7.0	7.2
ArxivAbstract	40000	68798	102.0	7.2
ArxivCSTitle	37000	18525	7.0	4.8
ArxivCSAbs	37000	68001	116.0	4.8

TABLE 6: Datasets Overview

Table 6 gives a detailed description of all the datasets that we used for experimentation. DBLP contains 9396 documents with only title information and average text length is 6.4 words with total vocabulary size 8036. It has a dense citation graph as the average degree of the network is 8.0 meaning each document on average is citing 8 other papers. $DBLP_{adv}$ contains only 6704 documents with title and abstract information, as a result, the average text length increased to 100.5 words with total vocabulary size 15932. Again, the citation graph is dense with the average degree count of 7.7. ArxivTitle and ArxivAbstract both contain $40,000$ documents with $10,000$ documents per class. The average text length of ArxivTitle and ArxivAb-

stract is 7 and 102 respectively. The density of the citation graph is less as compared to the DBLP dataset as the average degree count is 7.2. In the case of ArxivCSTitle and ArxivCSAbstract, we randomly selected $37,000$ documents and the average text length for the title is 7 followed by 116 for abstract. The density of the citation network is 4.8 which is the least in all of the datasets.

CHAPTER 5

Experimentations

In this section, we will describe the parameters we used for our experiments and also how we obtained our results.

5.0.1 Evaluation Benchmarks

Doc2Vec

We used PV-DBOW [27] to obtain embeddings of the dataset implemented in Gensim [gen]. We set vector size equal to 100 with window size 5 for the datasets with title only and 8 for those datasets which has abstract too. We used negative sampling with k = 15 and min_count = 10 and used iterations = 10.

Node2Vec

For node2vec [25], we use the source code [n2v] from Github provided by the authors. We set p and q equal to 1 and walk length to 80. The vector dimension was set to 100 and after obtaining the embeddings we concatenated them with the PV-DBOW embeddings.

TADW

TADW [42] is implemented in Matlab. The authors provided the code in mexa64 file format. The authors did not provide the source code of these libraries and hence

we could not use them as we needed the source code for detailed experimentation. Instead we used the code provided by [18] for our experimentation. We set lambda equal to 0.2 and used edgelist as the graph format.

LDE

The authors [39] did not provide the source code for this method. We used the code provided by Yi Zhang for our experimentation. We used the default parameters in our experiment.

Evaluation Method

The experimentation is conducted in two phases:

- Representation Learning
- Embedding Evaluation

We used the complete dataset for learning the embeddings in the document representation learning phase and for the evaluation of the embeddings we selected randomly 4000 samples from each class in Arxiv and ArxivCS dataset. For DBLP and DBLP_adv we used the complete dataset for evaluation too. The reason behind partial evaluation of the Arxiv and ArxivCS dataset was that these datasets were massive and would take a lot of time for evaluation.

For the evaluation of the embedding, we performed the following experiments

- Classification
- Clustering

To verify the results, we also potted our data to determine the correctness of our results visually.

Document Classification

To evaluate our results, we passed the document embedding to the classifier and compared the predicted class with the actual class. We used cross-validation, which means separate the training data into two parts. One part is used for training the classifier, and the other part is used for testing the classifier. We used 10-fold cross validation meaning we trained the classifier ten times using different training set and tested the data on a different test dataset each time. After every experimentation, we obtained confusion matrix which is used for measuring the performance of the classifier. This matrix shows the number of observation that belongs to a specific class but how the classifier predicted it into another class. Based on this matrix we can obtain a true positive (TP), true negative (TN), false positive (FP) and false negative (FN). TP means the number of documents that actual class is a true class and predicted class is also a true class. Similarly, TN means that actual class is false and predicted class is also false. FN shows that the number of documents that actual class is true but predicted class is false. FP is the number of documents that actual class is false but predicted to be true. Precision is the fraction of relevant instances among the retrieved instance while recall is the fraction of relevant instances that have been retrieved over the total number of relevant instance. Precision tells us how useful our classifier is and recall tells us how complete our result is. The F1 measure defines a weighting between recall and precision. It is a harmonic average of the precision and recall where the F1 score of 1 means the best result and 0 the worst result. A higher value of F1 ensure efficiency of the classifier and can be calculated using equation 3

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{1}$$

Where precision and recall are defined in equation 4 and 5 respectively

$$Precision = \frac{TP}{TP + FN} \tag{2}$$

$$Recall = \frac{TP}{TP + FP} \tag{3}$$

We used F1 micro and F1 macro as an evaluation metric for multi-label classification. In F1 micro, we sum up the individual true positives, false positives and false negatives of the system for different sets and then apply them to the equation 4 and 5 respectively. For F1 macro, we take the average of the precision and recall produced in each iteration. F1micro is useful when the dataset varies in size whereas F1 macro tells us the overall performance of the system across the sets of data. We used Logistic Regression as the Classifier implemented in Scikit [LR]

Document Clustering

We used clustering as an another form of evaluation. We used KMeans implemented in Scikit [kme]. For training we used 80% of the data and once the model is trained we used the remaining 20% to predict the labels. To measure the efficiency of the clustering algorithm we used Purity [30] and Silhouette score implemented in Scikit [sil].

Purity

Purity [30] measures clustering quality by assigning each cluster the most frequent class in that cluster and then dividing it by the total number of documents as defined in 4. N and k denotes the total number of documents and clusters respectively. c_i is a cluster in C and t_j is the classification which has the max count for cluster c_i.

$$Purity \ = \ \frac{1}{N} \sum_{i=1}^{k} max_j |c_i \cap t_j| \tag{4}$$

Purity score is in the range of $[0, 1]$ where 1 is the best score, and 0 is the worst

score. To calculate purity, we first create a confusion matrix. To do so, we get cluster labels for our test samples using our model. Then using *metrics.cluster.contingency_matrix* implemented in Scikit [pur] we pass test labels and true prediction values for our test samples to get the confusion matrix. Then we take the sum of the max number of documents in the class and divide it by the total number of documents as shown in 4

Silhouette Score

This score is calculated using mean intra-cluster distance as a and the mean nearest cluster distance as b for each sample. a is defined in the following equation:

$$a(i) = \frac{1}{C_i - 1} \sum_{j \in C_i \ j \neq i} (d_i, d_j)$$

where i is a datapoint in the cluster C_i. $d(i, j)$ is the distance between the points i and j.

b is defined as:

$$b(i) \ = \ \frac{min}{i \neq j} \frac{1}{C_j} \sum_{j \in C_j} d(i, j)$$

and Silhouette score for each point then can be obtained as

$$s(i) \ = \ \frac{a(i) \ - \ b(i)}{max\{a(i), b(i)\}}$$

The best value is 1 and worst value is -1. Value near 0 means that the clusters are overlapping and the negative value suggest that the sample has been assigned a wrong cluster.

Document Visualization

For document visualization we projected the embedding obtained using t-SNE[29] to see how similar documents are clustered together in the vector space. This technique

is a variation of Stochastic Neighbor Embedding and can reveal structure of the data at many different scales. For reducing the dimension of document embeddings in t-SNE we used preplexity = 20, n_components = 2 and init = PCA. We also used PCA [41] to project multi-dimensional vectors into 2 dimensional vector space for evaluation.

Dataset	PVDBOW	Node2Vec	DBOW-Node2Vec	LDE	TADW	Retrofit1	Retrofit2
DBLP	0.623	0.679	0.684	0.655	0.676	0.671	0.692
DBLP_adv	0.647	0.655	0.681	0.744	0.664	0.66	0.668
Arxiv	0.768	0.597	0.797	0.691	0.806	0.791	0.784
ArxivAbs	0.859	0.595	0.872	0.721		0.865	0.862
ArxivCS	0.819	0.844	0.893	0.850	0.879	0.868	0.873
ArxivCSAbs	0.893	0.848	0.915	0.877		0.914	0.906

TABLE 7: Classification Result

5.1 Results

5.1.1 Classification Result

This section will discuss the summary of the classification result obtained for all the methods using all six datasets. Table 7 shows the result of classification. In the experiment, we used Logistic Regression implemented in Scikit [LR] as the classifier with default parameters. The output from each method produced a 100-dimensional vector and passed to the classifier.

Figure 22 shows the classification result of all the methods for each datasets. The x-axis represents different methods, and the y-axis shows the classification score using $F1micro$. For DBLP, we can see retrofit2 performed the best because it has a very good network graph and small text. In DBLPadv, LDE outperformed all the methods, and this is because it is the smallest dataset we have therefore LDE can capture information from content and network view successfully.

For ArxivTitle, TADW performed the best but gave memory error for ArxivAbs. TADW is not scalable for large datasets. In ArxivAbs concatenation gave us the best result closely followed by retrofit1 and retrofit2.

In ArxivCS and ArxivCSAbs, concatenation performed the best but closely followed by retrofit1 and retrofit2.

Figure 23 is plotted to visualize the classification result from a global perspective. On x-axis are the datasets and on the y-axis is the $F1score$. Each method has a

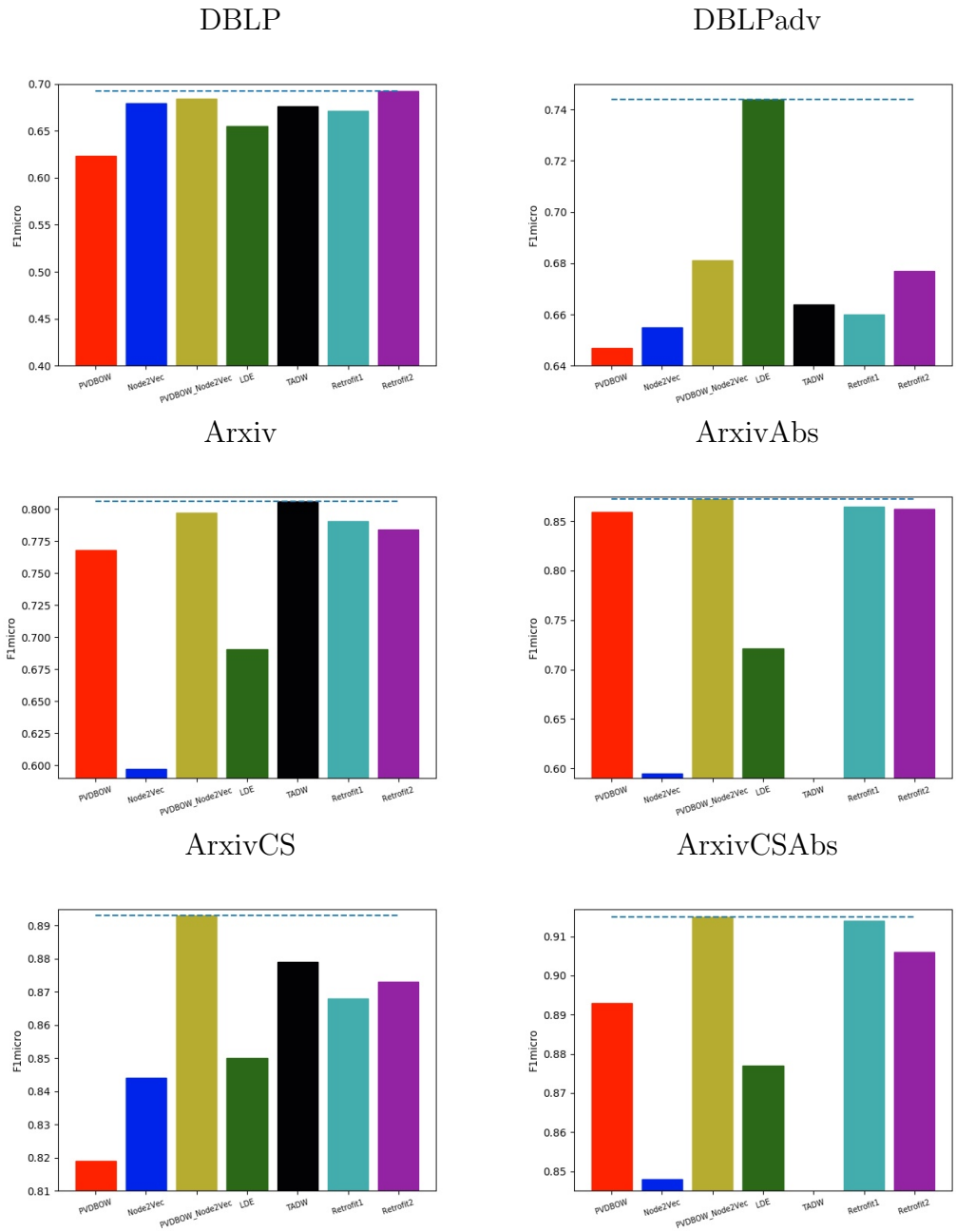


FIGURE 22: Classification Result for all the methods for individual datasets. Retrofit2 performed the best for DBLP dataset. LDE outperformed all the methods in DBLPadv. TADW performed the best for Arxiv but gave memory error for ArxivAbs and ArxivCSAbs. For rest of the datasets, concatenation performed the best followed by either retrofit1 or retrofit2

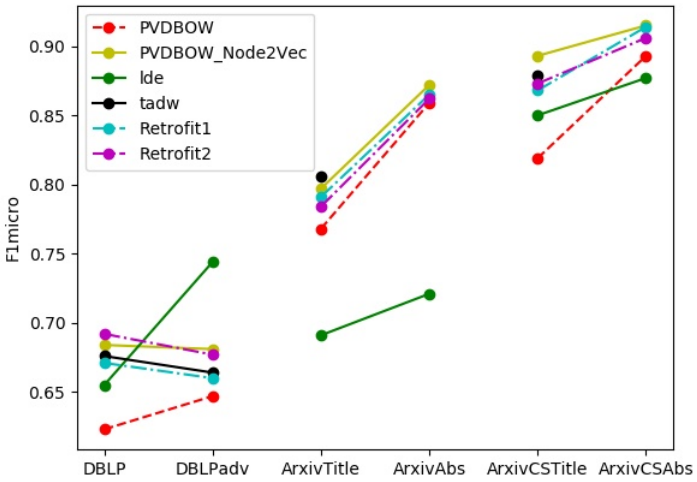


FIGURE 23: Classification Result for all the methods across different datasets to give a global overview. As content increases, overall $f1score$ of all the methods also increases. Addition of network information has little improvement when the content is more.

unique color, and the dots represent the $F1score$ of the method for each dataset. The connection between the dots means that the dataset has more content from left to right. For example, DBLP has title only, and DBLPadv has title+abstract. For each method, we used the parameters outlined in the Evaluation Benchmark section 5.0.1. The purpose of this experiment is to study the effect of more content when network information is added.

For all the dataset, we can observe that the addition of network information to PV-DBOW improves the classification score.

For DBLP, retrofit2 gave the best score and LDE in DBLPadv. In DBLPadv, we lost some of the documents while getting more content. Hence the citation graph changed. As a result, the addition of network information to the document embeddings decreased for all the methods when compared with DBLP.

In ArxivTitle, TADW gave the best score and PVDBOW_Node2vec in ArxivAbs. The gap between content based (PV-DBOW) and network+content based methods

has also decreased because of the poor citation graph. LDE performance is worse than PV-DBOW because it is susceptible to both network and content when learning. Since the network is not good, LDE is not able to capture the document semantics successfully. TADW failed to compute the embeddings for ArxivAbs because it is not scalable.

PVDBOW_Node2vec gave the best score for ArxivCS, and ArxivCSAbs followed closely by retrofit2 in ArxivCSAbs. TADW again failed to capture the embedding for ArxivCSAbs because it could not handle a large amount of data.

5.2 Parameter Tuning

5.2.1 Retrofit

Dataset	**0**	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1**
DBLP	0.619	0.629	0.644	0.659	0.672	0.662	0.659
DBLPadv	0.644	0.647	0.666	0.660	0.653	0.650	0.641
Arxiv	0.764	0.768	0.773	0.777	0.782	0.771	0.764
ArxivAbs	0.864	0.865	0.864	0.864	0.863	0.857	0.858
ArxivCS	0.813	0.827	0.845	0.856	0.863	0.848	0.829
ArxivCSAbs	0.891	0.900	0.907	0.910	0.906	0.892	0.881
CS3	0.914	0.918	0.926	0.933	0.935	0.924	0.914
CS3Abs	0.963	0.964	0.969	0.970	0.964	0.961	0.948
CS6	0.734	0.749	0.778	0.793	0.788	0.765	0.756
CS6Abs	0.832	0.839	0.846	0.853	0.850	0.827	0.809

TABLE 8: Hyper-parameter β tuning for the retrofit1. β will determine how much weight to give to content or network information. When $\beta = 0$, retrofit is biased towards content and will ignore network information. When $\beta = 1$, the retrofit will ignore the content information and give all weight to network information

In this experiment, we varied the hyper-parameter β for the both retrofit1 and retrofit2 to see how the addition of content and network information effects the classification score respectively. When $\beta = 0$, it means that we are ignoring the network information altogether. As we increase β, we give more weight to network and less weight to content. At $\beta = 0.5$, the weight for network and content information is equal, and we expect to receive the best classification score at this point. When

Dataset	**0**	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1**
DBLP	0.627	0.638	0.652	0.672	0.673	0.668	0.661
DBLPadv	0.647	0.649	0.660	0.67	0.659	0.654	0.651
Arxiv	0.773	0.779	0.782	0.790	0.789	0.785	0.779
ArxivAbs	0.858	0.861	0.865	0.868	0.861	0.858	0.856
ArxivCS	0.818	0.832	0.856	0.869	0.867	0.853	0.842
ArxivCSAbs	0.893	0.904	0.912	0.913	0.910	0.893	0.883
CS3	0.919	0.922	0.934	0.940	0.939	0.930	0.918
CS3Abs	0.963	0.966	0.972	0.970	0.968	0.954	0.949
CS6	0.732	0.756	0.783	0.797	0.793	0.781	0.765
CS6Abs	0.823	0.836	0.858	0.861	0.844	0.832	0.815

TABLE 9: Hyper-parameter β tuning for retrofit2. β will determine how much weight to give to content or network information

$\beta = 1$, we are ignoring all the content information and using the information only from the network. We used logistic regression as a classifier. Table 8 and 9 shows the classification score for retrofit1 and retrofit2 respectively.

Figure 24 shows the classification score for all DBLP and Arxiv dataset. On x-axis are different values of β and on the y-axis are the $F1scores$. In DBLP, we can observe that as we increase β, the classification score improves. We report the best $F1score$ when $\beta = 0.7$ for both retrofit1 and retrofit2, meaning we are taking 70% of information from the network and 30% from content respectively. This is because the network of DBLP is excellent and with the addition of only 30% content it gives us the best result. In DBLPadv, we observe a similar pattern but note we obtain the best classification score when $\beta = 0.7$ for retrofit2 and $\beta = 0.3$ for retrofit1 respectively. This is because DBLPadv contains more content (title + abstract); therefore, the content information alone is sufficient. We have to give more weight to content 70% and network 30% so that the overall vector learned is the best in case of retrofit1.

In Arxiv and ArxivAbs, we observe a bell-shaped curve, suggesting that we obtain the best f1-score when $\beta = 0.5$. That means we give equal weight to content and network information while retrofitting. ArxivIntro contains the introduction section of the papers meaning it has even more text. Still we observe $\beta = 0.5$ the best for classification. This also reinforces our hypothesis that the addition of network

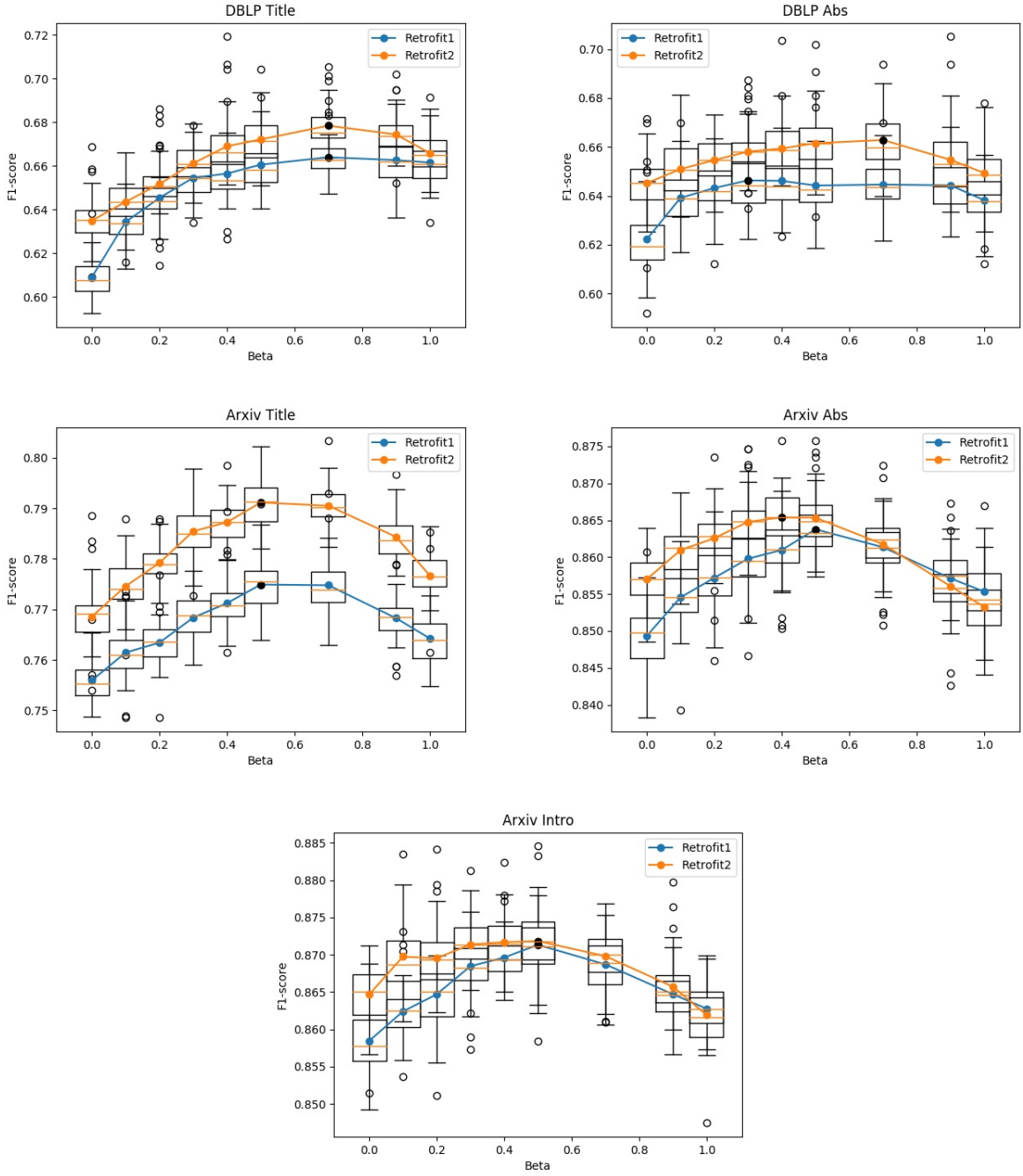


FIGURE 24: Tuning of the β parameter for DBLP and Arxiv dataset. In DBLP more weight should be given to network as the quality of the citation graph is good. In Arxiv, as we increase content length, the $f1score$ for $\beta < 0.5$ starts to improve, suggesting more weight should be given to the content.

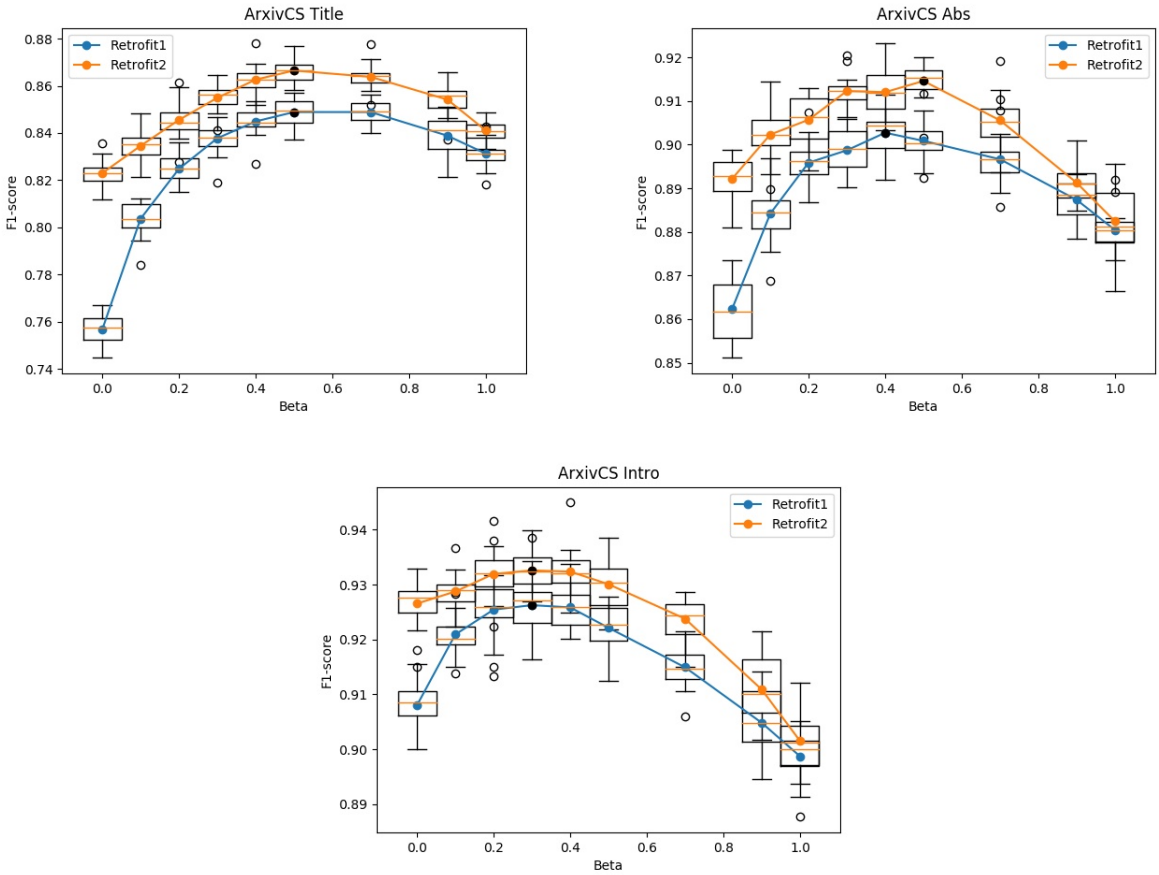


FIGURE 25: Tuning of the β parameter for CS papers. As content length increases, $f1score$ starts to improve again for $\beta < 0.5$. The best β also shift towards 0.3 as the introduction section is added.

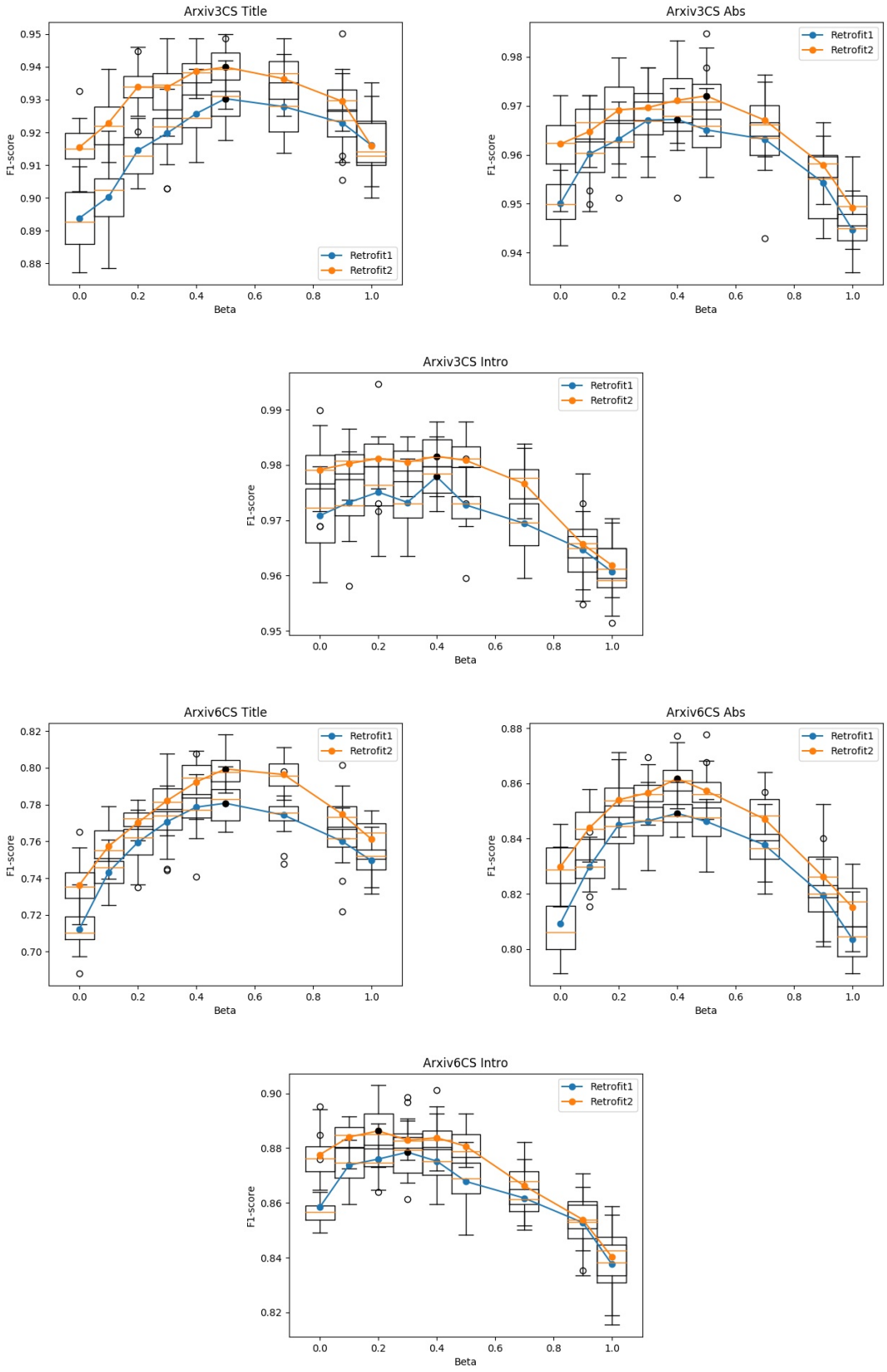


FIGURE 26: Tuning of β parameter for the additional CS papers.

information to the content based embeddings will improve the classification score.

Dataset	**Classes**
3CS	computer vision, cryptography and database area
5CS	Information Theory, Distributed Computing, Machine Learning, Computer Vision, Compu- tational Complexity
6CS	artificial intelligence, computation and language, data structure and algorithm, computer science and game theory, logic in computer science, and social and information networks

TABLE 10: Additional CS datasets

In CS papers, we performed classification on different set of classes as shown in Table 10. Set with three classes contains papers belonging to computer vision, cryptography and database area. Set containing five classes is a default dataset we used in the experimentation and denoted as ArxivCS in the previous sections. Set with six classes contains documents in the area of artificial intelligence, computation and language, data structure and algorithm, computer science and game theory, logic in computer science, and social and information networks. Each class consist of 1000 documents. The motivation to form these new dataset was to study the pattern of β in different datasets and generalize our findings.

For ArxivCS and ArxivCSAbs, we observe a similar pattern as shown in Figure 25. We get the best $F1score$ when $\beta = 0.5$ in the case of title only. With the addition of abstract, $\beta = 0.5$ and $\beta = 0.4$ gives us the best classification score for both the methods. Addition of the introduction section further inclines the algorithm to favor more content as we observe the best $F1score$ when $\beta = 0.3$ for both the methods. Figure 26 also shows the similar pattern. When we have less content, both the method gives us the optimal result when $\beta = 0.5$. As we increase content, we need to give

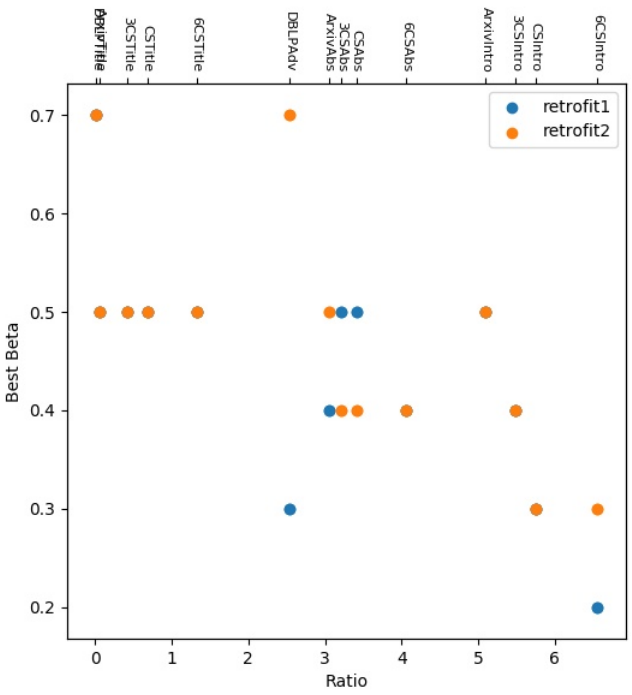


FIGURE 27: Relationship of content and network with β using multi-label classification

more weight to the text view to receive the best result.

Dataset	Ratio	Retrofit1	Retrofit2
DBLP(Title)	8/64=1.25	0.7	0.7
Arxiv(Title)	1/7.429=0.135	0.5	0.5
ArxivCS(Title)	4/7.91=0.507	0.5	0.5
Arxiv3CS(Title)	5/.7575=0.660	0.5	0.5
Arxiv6CS(Title)	2/7.512=0.266	0.5	0.5
DBLPadv	8/100.5=0.08	0.3	0.7
ArxivAbs	1/148=0.006	0.4	0.5
ArxivCSAbs	4/121=0.033	0.5	0.4
Arxiv3CSAbs	5/124=0.040	0.5	0.4
Arxiv6CSAbs	2/116=0.017	0.4	0.4
ArxivIntro	1/1149=0.0008	0.5	0.5
ArxivCSIntro	4/1258=0.003	0.3	0.3
Arxiv3CSIntro	5/1204=0.004	0.4	0.4
Arxiv6CSIntro	2/1401=0.001	0.2	0.3

TABLE 11: Experimental results for predicting the best possible value of β

$$ratio \ = \ \log \frac{avg\ text\ length}{avg\ deg} \tag{5}$$

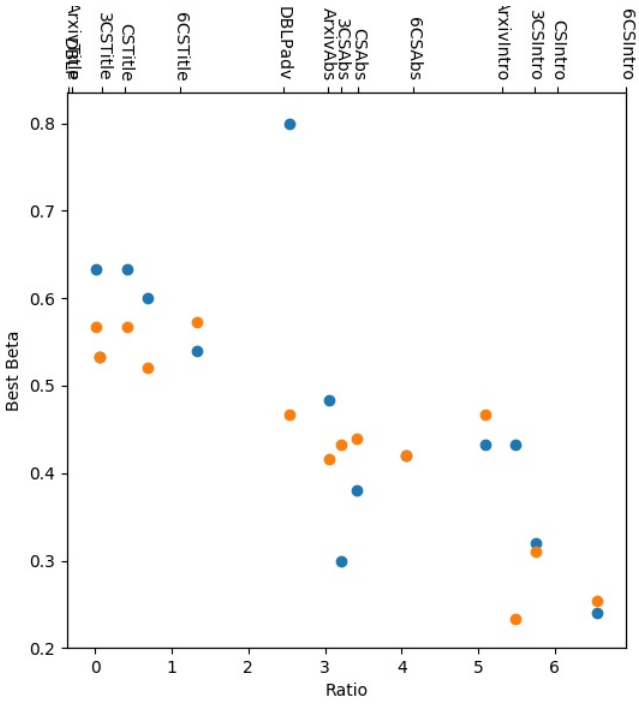


FIGURE 28: Relationship of content and network with β using binary classification

For predicting the best possible value of β, we analyzed the content and network information and found a relationship between them. We used the average degree of the network to measure the quality of the graph quantitatively. And for content, we used average text length to calculate the quality of the text. Equation 5 defines the ratio that will give us a rough estimate of the best possible β value. We used log to spread out ratios for the dataset with the introduction sections. Table 11 shows the best β value for different ratios. Figure 27 shows that when we have datasets with small text like title , then $beta > 0.5$ gives us the best result. β fluctuates between 0.5 and 0.4 for the data with abstract. This is because at this ratio, all the documents contain more content and we need to adjust the β value based on the network quality. When dataset have more text like introduction, then the β fluctuates between 0.5 and 0.2 based on our experimentation.

The above experiment 27 is based on multi-label classification. To understand this pattern in more depth, we performed a binary classification for each dataset.

For example, Arxiv3CS has 3 classes, so we performed a binary classification on all the combinations resulting in 6 samples. We repeated this experiment for all variants of Arxiv3CS which includes title, abstract and introduction. This process was implemented for all the datasets; as a result, we got several samples for each ratio. Figure 28 shows the binary classification result. Y-axis is an average of the best β value for different binary classifications. We can observe a pattern when the ratio is less meaning content is less, then $\beta > 0.5$ will give us a better result. When $2 < ratio < 4$ like for the datasets with the abstract, we observe that the best value of β is in the range from 0.5 to 0.3. When the $ratio > 5$, we see the β value decreases and gives us the best value when the $\beta < 0.4$. This is because either the network is not good or we have a large amount of content. $\beta = 0.8$ at $ratio = 2.8$ is an outlier. This is the binary classification result of sigmod and icses class.

To further understand, what is the best possible value of beta for different datasets, we analyzed the citation graph in depth. Each dataset has multiple classes. For each class, we counted how many times it cites documents belonging to the same class and different classes. For example, DBLP has three classes; icse, sigmod, and vldb. We determined the number of times icse documents cited papers belonging to the icse class and also the other classes. The table 12 shows the statistics. The quality column is calculated by multiplying the percentage of documents citing within their own class. Quality=1 means that the network is very good and all the documents cite within their own class.

The figure 29 visualizes the findings in the table 12. The network quality of DBLP and DBLPadv is relatively good. ICSE documents are citing papers mostly within their own class. VLDB and Sigmod papers are linked together which decreases the overall quality of the graph. In Arxiv, cs, stat and maths class is linked together which decreased the overall quality of the graph. We observe the best network in 3CS dataset which reports the highest quality of 0.281.

dataset	classes	percentage	quality
DBLP			0.0814
	icse	97	
	sigmod	52	
	vldb	48	
DBLPadv			0.0820
	icse	97	
	sigmod	51	
	vldb	49	
Arxiv			0.0685
	cs	77	
	stat	40	
	math	98	
	physics	88	
3CS			0.281
	DB	96	
	CR	88	
	CV	99	
ArxivCS			0.0687
	CC	88	
	DC	70	
	CV	87	
	LG	65	
	IT	97	
6CS			0.0522
	AI	65	
	DS	93	
	GT	83	
	LO	86	
	CL	89	
	SI	79	

TABLE 12: Quality of the citation graph

In terms of beta, we can estimate the best value by determining the quality of the graph and content length. In case of retrofit1, when there is less content, and the network quality is good, $\beta = 0.7$ will give us the best result. As we increase content while keeping the network constant, $\beta = 0.5$ will produce the best result. We observe the same pattern for all the datasets.

Retrofit2 is more robust and less sensitive to the quality of the citation graph. It give us the best result when $\beta = 0.5$.

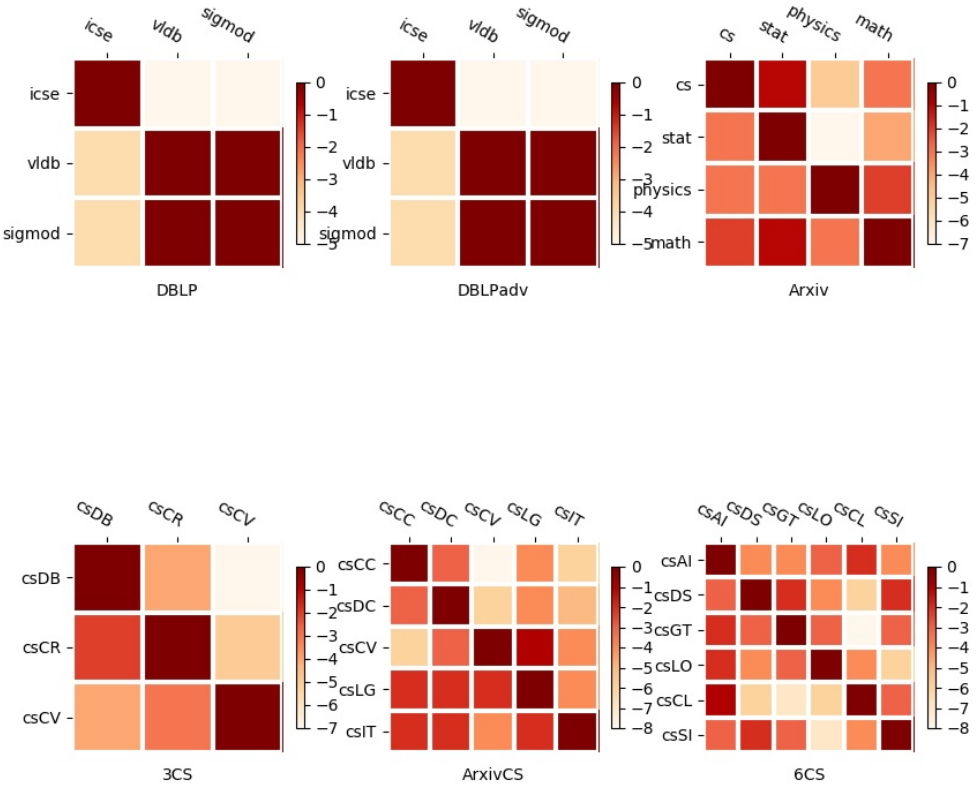


FIGURE 29: Analysis of the citation graph. Dark color demonstrates high number of documents citing that class. Y-axis shows different classes and x-axis shows the breakdown of the classes with respect to the number of documents it is citing. Ideally, all the dark red colors should be in a diagonal.

5.2.2 Node2vec

For node2vec [25], our experiments are focused on the following key points:

- analysis of how the classification score changes when the weight β of obtaining information from different views is varied
- effect of hyper-parameter p and q on classification score after concatenating with PV-DBOW
- impact of vector dimension on the classification score for node2vec and concatenated node2vec with PVDBOW

Effect of β on Concatenation

We introduced β to control the weight for obtaining information from the text and network view using concatenation. The idea behind β is similar to the retrofit algorithm. The value of β ranges from 0 to 1. We use PV-DBOW to capture the vector representation of text, and node2vec to learn the vector representation for the network. We use β to determine the weight for obtaining information from each view. When $\beta = 0$, we are considering the information only from the content view and when $\beta = 1$ we are considering only the network. To introduce β in concatenation, we first normalize the PV-DBOW and node2vec vectors. After normalization, we multiply β with the vector obtained from node2vec and $1 - \beta$ with PV-DBOW and then concatenate these vectors. At extreme ends of β, we use the respective vectors alone without concatenation. For example $\beta = 0$ we use only PV-DBOW vector and when $\beta = 1$ we use only node2vec vector. For node2vec, hyper-parameter p and q is equal to 1 and walk-length = 80. For PV-DBOW we use window size=5, number of negative samples=25 and iteration=10 The vectors are then evaluated using logistic regression with ten cross-validations. We repeat the experiment three-time and plot the result using a box plot.

DBLP DBLPadv

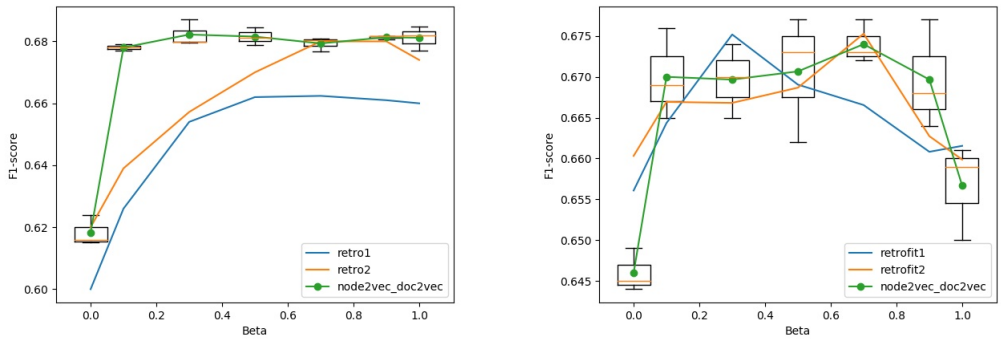


FIGURE 30: Effect of β on concatenation using DBLP. Change in β has little effect on concatenation while retrofit2 becomes equal to concatenation as β increases to 0.9.

Arxiv (Title) Arxiv Abs

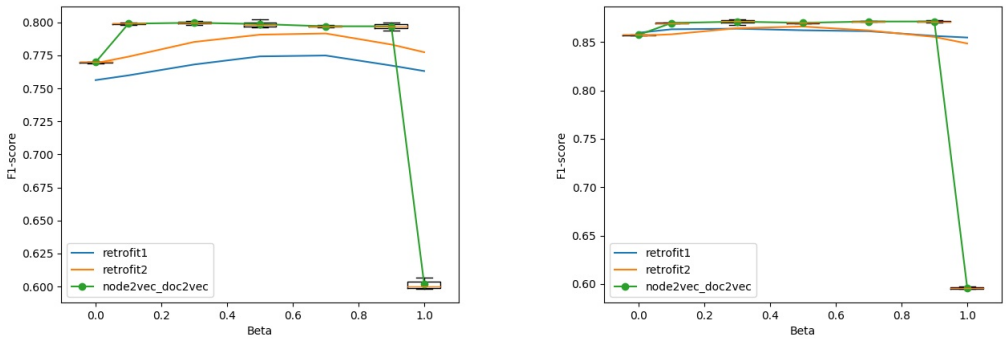


FIGURE 31: Effect of β on concatenation using Arxiv. β again did not change the classification score for concatenation which shows it is difficult to control how much information we want from different views.

Figure 30 shows the effect of varying β on concatenation using classification. X-axis represent different β values and y-axis shows the $F1-score$ for logistic regression. For DBLP dataset, we can see β did not have much effect on the concatenation. $\beta = 0.1$ used only 10% information from the network and gave almost the same score as when $\beta = 0.9$ which was using 90% of information from the network. On the other hand, retrofit2 was almost equal to concatenation when $\beta = 0.7$. We can also see both variants of retrofit are more sensitive to β than concatenation.

The effect of β on concatenation for Arxiv is shown in the figure 31. For the

ArxivCS (Title) ArxivCS Abs

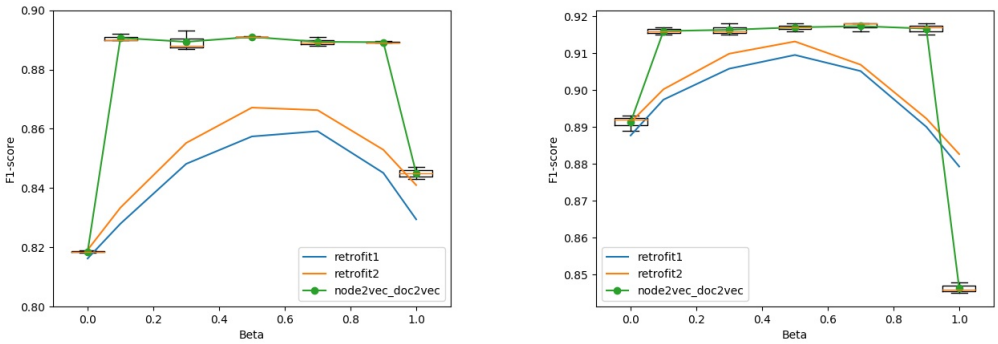


FIGURE 32: Effect of β on concatenation using ArxivCS. We observe the similar pattern for concatenation as we changed β.

Arxiv title and abstract, we can observe that β again did not have much effect on the concatenation. When $\beta = 1$ classification score for the node2vec immensely decreased because of the poor network in Arxiv Title and Arxiv Abs. Figure 31 also shows that retrofit2 performed almost equal to the concatenation when $\beta = 0.7$ and $\beta = 0.5$ for Arxiv Title and Arxiv Abs respectively.

Figure 32 shows the effect of β on concatenation for ArxivCS. With the title only, concatenation outperforms both variants of retrofit, but with the addition of abstract, we can observe that retrofit2 is near to the concatenation.

Dataset	**0**	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1**
DBLP	0.646	0.670	0.680	0.686	0.684	0.678	0.686
DBLPadv	0.621	0.657	0.654	0.652	0.657	0.651	0.650
Arxiv	0.490	0.565	0.593	0.607	0.606	0.610	0.618
ArxivAbs	0.499	0.567	0.588	0.601	0.613	0.614	0.618
ArxivCS	0.766	0.829	0.835	0.839	0.841	0.842	0.842
ArxivCSAbs	0.765	0.828	0.838	0.842	0.839	0.842	0.844

TABLE 13: $F1score$ of node2vec when the hyper-parameter p and q is varied. When $p = 0.1$, then $q = (1 - p)$

Dataset	0	0.1	0.3	0.5	0.7	0.9	1
DBLP	0.664	0.680	0.683	0.686	0.686	0.682	0.689
DBLPadv	0.659	0.663	0.665	0.660	0.667	0.665	0.659
Arxiv	0.794	0.804	0.808	0.806	0.805	0.803	0.808
ArxivAbs	0.867	0.869	0.868	0.872	0.870	0.869	0.871
ArxivCS	0.868	0.887	0.885	0.890	0.893	0.891	0.889
ArxivCSAbs	0.908	0.915	0.917	0.917	0.916	0.918	0.916

TABLE 14: $F1score$ of concatenated node2vec and PV-DBOW when the hyper-parameter p and q is varied. When $p = 0.1$, then $q = (1 - p)$

Hyper-parameter p and q Training

For the hyper-parameter p and q, we merged them such that when $p = 0.1$ then $q = 0.9(1 - p)$. If we set p to 0.9, then the random walk will be biased towards the nodes which are away from the source node. The node2vec will identify nodes which have similar structure if p is large. Similarly, if we set p to 0.1, then the random walk will favor nodes near to the source node. As a result, these nodes will be belonging to the same community.

Table 13 shows the classification score for all the datasets as we increase the value of p for node2vec. Table 14 is obtained after concatenating node2vec vectors from 13 with PV-DBOW. The $F1score$ is obtained by passing node2vec and node2vec_doc2vec to the logistic regression.

Figure 33 combines the result from the Table 13 and 14. The x-axis represent different values of p and y-axis represent the $F1score$. The node2vec is labelled in the form of blue line and concatenation of node2vec with PV-DBOW as orange. For DBLP and DBLPadv we can observe that when p is small node2vec_doc2vec performed better. As p increases node2vec score improves but node2vec_doc2vec score relatively decreases or stays constant. We can observe similar pattern for Arxiv and ArxivCSAbs dataset. In ArxivCS and ArxivAbs, we can clearly observe that when p is small, node2vec_doc2vec performance was better. As we increase p, node2vec performance improved but node2vec_doc2vec performance decreased.

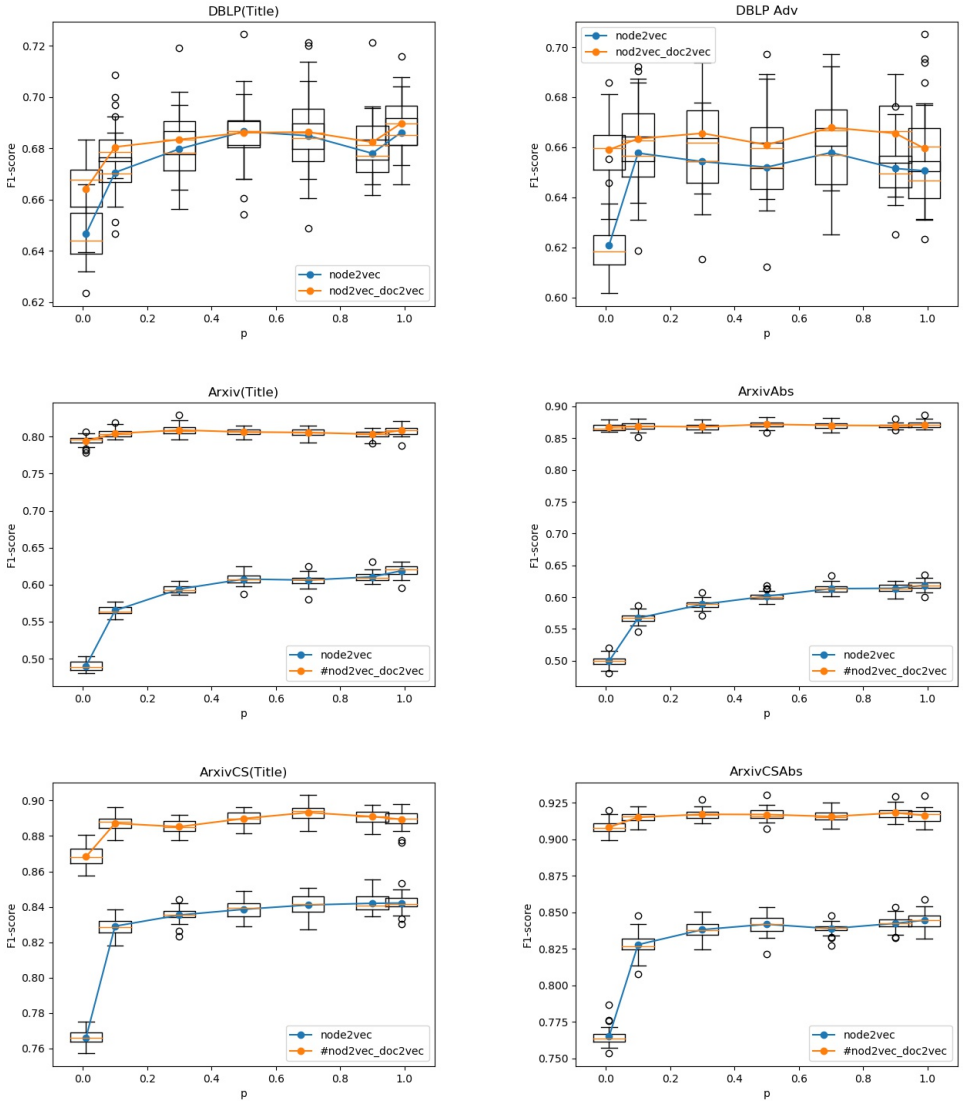


FIGURE 33: Determines the effect on the classification result, when hyper parameter p and q is changed in node2vec.

The reason behind this phenomena is that when p is small, nodes sampled in the random walk are from the same community as the source node. Therefore, they share similar content. When we concatenate vectors, the information in node2vec amplifies the information present in PV-DBOW. As a result, the overall classification score improves. When p is large, then the nodes sampled are away from the source node. These nodes though have a similar structure to the source node, but their content is different. So when we concatenate, the information in node2vec does not help to improve the classification score.

Dataset	dim=50	dim=100	dim=150	dim=200	dim=400
DBLP	0.684	0.681	0.685	0.682	0.689
DBLP_adv	0.648	0.657	0.652	0.657	0.665
Arxiv	0.579	0.587	0.596	0.602	0.617
ArxivAbs	0.576	0.592	0.597	0.606	0.615
ArxivCS	0.829	0.842	0.840	0.851	0.852
ArxivCSAbs	0.828	0.842	0.842	0.844	0.853

TABLE 15: Node2vec classification score using different vector dimensions

Dataset	**dim=50**	**dim=100**	**dim=150**	**dim=200**	**dim=400**
DBLP	0.620	0.634	0.635	0.630	0.628
DBLP_adv	0.647	0.634	0.644	0.635	0.645
Arxiv	0.759	0.774	0.775	0.776	0.778
ArxivAbs	0.858	0.855	0.855	0.862	0.861
ArxivCS	0.809	0.823	0.831	0.829	0.838
ArxivCSAbs	0.893	0.893	0.891	0.891	0.896

TABLE 16: PV-DBOW classification score using different vector dimensions

Dataset	dim=100	dim=200	dim=300	dim=400	dim=800
DBLP	0.689	0.687	0.6884	0.681	0.696
DBLP_adv	0.675	0.660	0.6605	0.666	0.666
Arxiv	0.788	0.805	0.809	0.807	0.813
ArxivAbs	0.871	0.868	0.872	0.868	0.874
ArxivCS	0.881	0.885	0.894	0.896	0.895
ArxivCSAbs	0.915	0.915	0.92	0.915	0.916

TABLE 17: Classification score after concatenating node2vec 15 and PV-DBOW 16. The dimension is doubled because half of the features are from node2vec and the other half from PV-DBOW

Impact of Vector Dimension on the Classification Score

For determining the effect of vector dimension on classification score, we changed the vector dimension of node2vec and PV-DBOW from 50 to 400. Table 15 and 16 shows the classification score of different vector dimension for node2vec and PV-DBOW respectively. In node2vec we set p and q equal to 1. We passed these vectors to Logistic Regression for classification.

Similarly, Table 17 shows the classification result, after concatenating node2vec and PV-DBOW. The vector dimension doubles because 50 features are taken from node2vec and and 50 from PV-DBOW when $dim = 100$.

Figure 34 shows that when vector size is small vectors produced alone from node2vec did not capture much information from the network alone as they reported the lowest $F1score$. After concatenating the node2vec vector with the PV-DBOW vector which contains only content information enhanced the classification result drastically. This shows that the addition of network information to the content information will improve the vector representation of the document especially when the vector size is small. As we increased the vector dimension, node2vec performance remains stable whereas node2vec_pvdbow performance first decreased and then become equal to node2vec. This is because as the vector length increases, there is more space for the information to be stored in case of node2vec. When we concatenate node2vec with PV-DBOW, the node2vec vector already contains some of the information found in PV-DBOW. Hence the resultant vector might overfit and decrease the overall quality of the vector representation.

5.2.3 LDE

To do an in-depth analysis of the LDE model, we varied the weights of network and content information to check how the model responds. The weight β represent how much we want our model to sample the data from content or network information

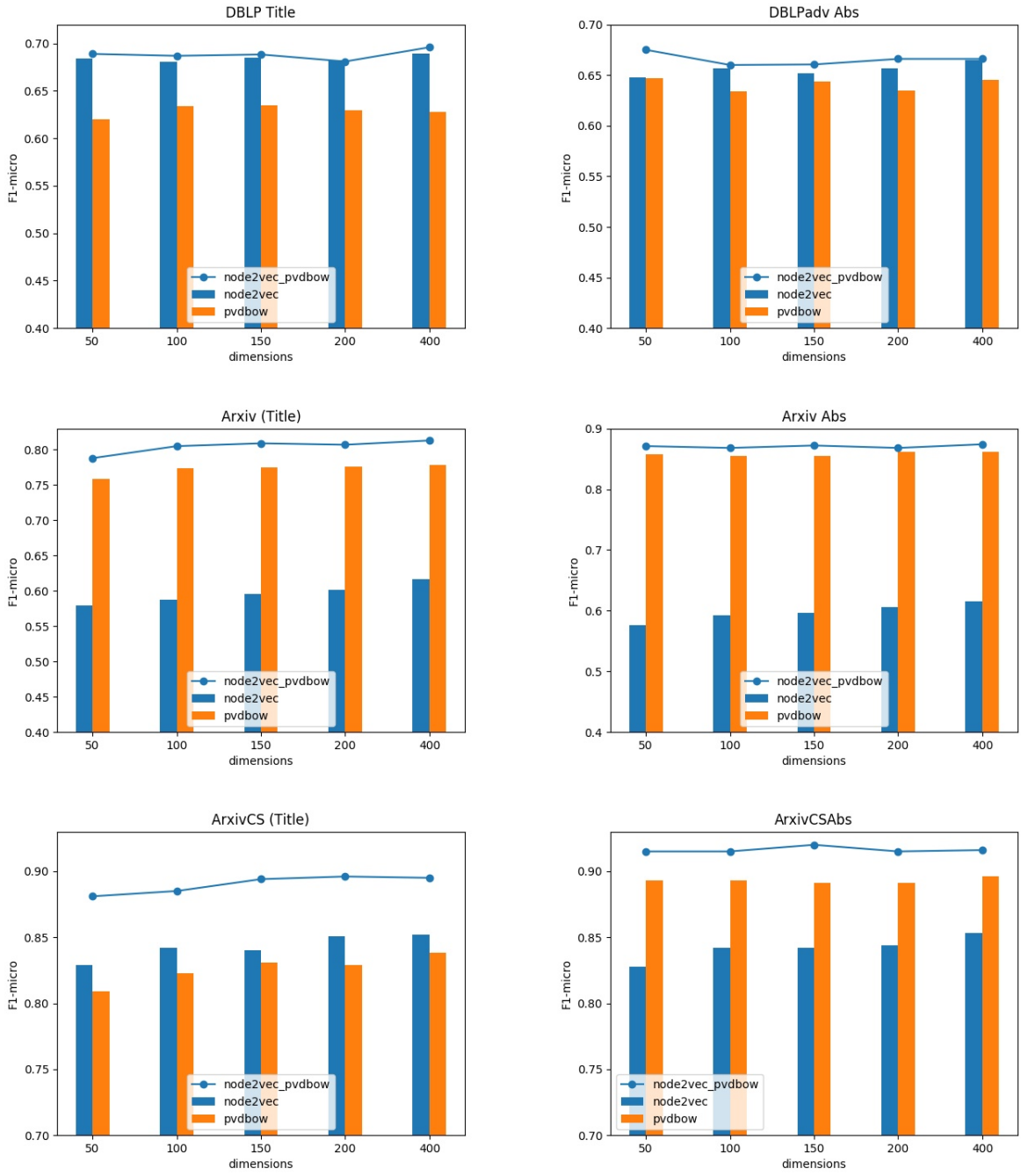


FIGURE 34: Concatenating PV-DBOW and node2vec with different vector dimensions

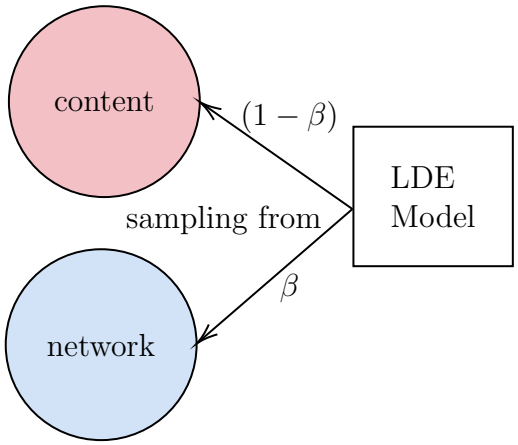


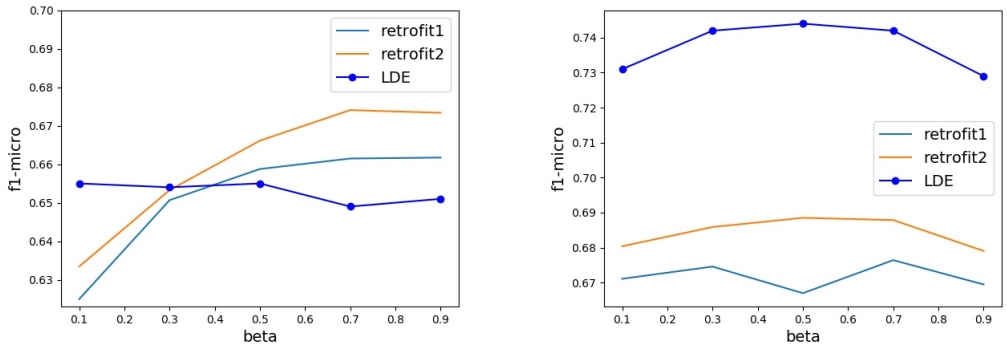
FIGURE 35: We use β in LDE to adjust the weights for content and network information respectively. If we set $\beta = 0.1$ that means we are taking more information from the content and ignoring most of the network information. Similarly, if we set $\beta = 0.9$ then the model will take more information from the network and ignore the content information.

respectively as shown in Figure 35. By default $\beta = 0.5$, meaning samples are drawn equally from content and network samples respectively.

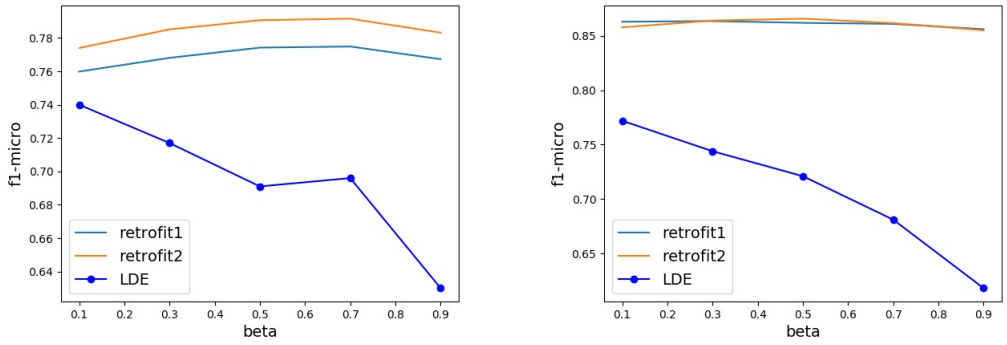
Dataset	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
DBLP	0.655	0.654	0.655	0.649	0.651
DBLPadv	0.731	0.742	0.744	0.742	0.729
Arxiv	0.740	0.717	0.691	0.696	0.630
ArxivAbs	0.772	0.744	0.721	0.681	0.618
ArxivCS	0.837	0.851	0.850	0.840	0.792
ArxivCSAbs	0.884	0.886	0.877	0.862	0.789

TABLE 18: LDE Parameter Tuning for β

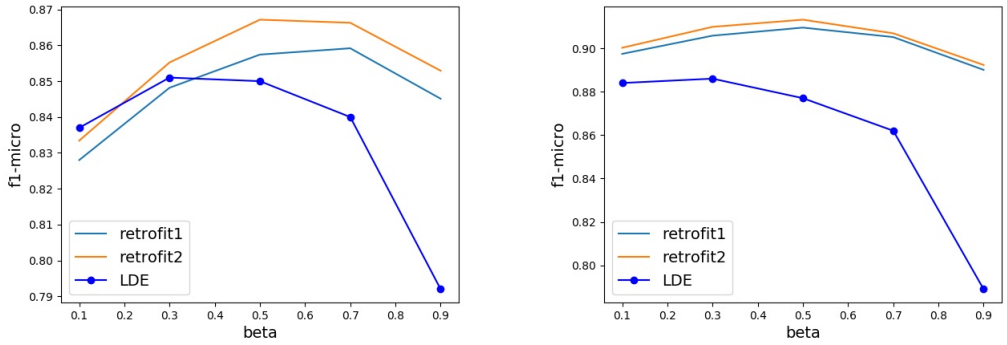
Table 18 show the classification result for different β values. We produced 100-*dimensional* vectors from LDE, and these vectors were then passed to logistic regression for classification. Figure 36 obtained from Table 18 shows how the models respond if we give more weight to the content information and network information respectively. When we set $\beta = 0.5$ means equal weight is given to both network and content information. When $\beta = 0.1$, more weight is given to the content information, and we obtain a high classification score for all the datasets. As β increases to 0.5, LDE performance starts to decrease for DBLP, Arxiv, ArxivABs and ArxivCSAbs



(a) DBLP **(b)** DBLPadv



(c) Arxiv **(d)** ArxivAbs



(e) ArxivCS **(f)** ArxivCSAbs

FIGURE 36: Parameter Tuning for the best β. When $\beta < 0.5$, the LDE model performed the best for all the datasets except for DBLPadv. LDE is not able to learn network and content information together successfully.Retrofit1 and Retrofit2 always outperformed LDE except for DBLPadv

dataset. When we continue to increase β, we give more weight to the network information, and we see the classification score decreases for all the datasets. This shows that the LDE model is not able to combine content and network view successfully together while learning the embeddings. For DBLPadv, we observe LDE outperformed retrofit1 and retrofit2 and also all the other methods. This is because DBLPadv is the smallest dataset we have consisting of only 6704 documents. Therefore, the LDE model can learn the information from both the views successfully.

5.2.4 Analysis of Adding Network Information to PV-DM Based Embeddings

Dataset	PV-DM(dm)	dm _node2vec	Retrofit1	Retrofit2
DBLP	0.619	0.690	0.666	0.687
DBLP_adv	0.655	0.678	0.668	0.671
Arxiv	0.618	0.722	0.662	0.672
ArxivAbs	0.838	0.86	0.849	0.842
ArxivCS	0.723	0.879	0.805	0.826
ArxivCSAbs	0.877	0.919	0.912	0.901

TABLE 19: Classification Result when the content vector is obtained from PV-DM

In this section, we will analyze the effect of adding network information to the embeddings obtained from PV-DM [27] method. We obtained the PV-DM vectors using Gensim [gen] and then added network information through the concatenation of node2vec and retrofit respectively. For PV-DM we used $windowsize = 10$, $negative\text{-}samples = 15$ and $iteration = 10$. In node2vec we set $p = 1$ and $q = 1$. β in retrofit1 and retrofit2 were both equal to 0.5. Table 19 shows the classification result obtained and Figure 37 visualizes that result. The bar plots in the figure represent the classification score of each method for all the datasets. The bars behind the plot shows the classification score obtained using PV-DBOW. Figure 37 shows that the addition of network information improves the classification score for all the datasets. We can also see PV-DM method did not perform as well as the PV-DBOW for both Arxiv and ArxivCS dataset. Consequently, the addition of network information to PV-DM also did not perform as well as those based on PV-DBOW.

Figure 38 shows that PV-DM failed to separate the classes for Arxiv and ArxivCS dataset. We can conclude from this experiment that embedding produced from PV-DBOW are more susceptible to adding information from the network view.

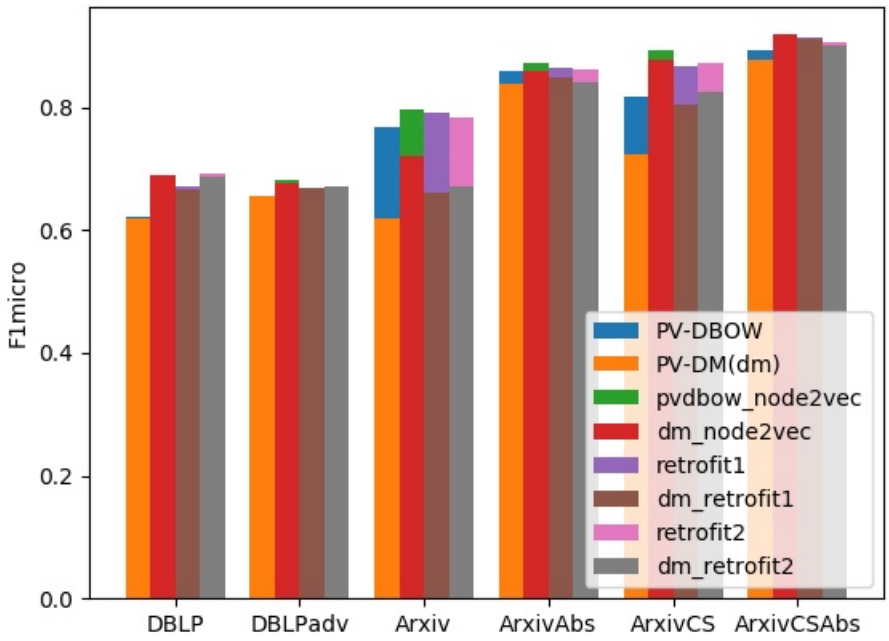
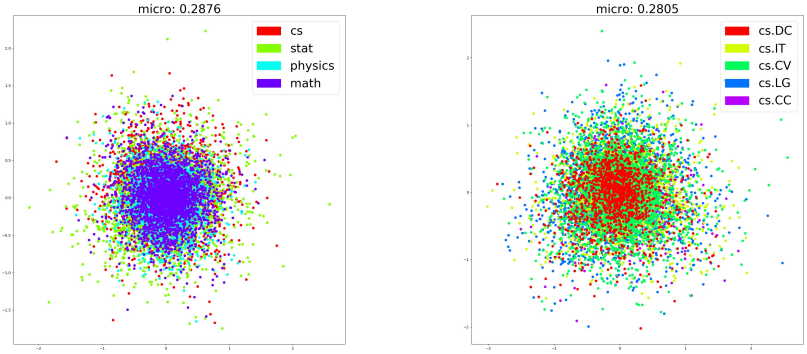


FIGURE 37: Classification result for PV-DM. For DBLP and DBLPadv dataset, PV-DM vector did not change the result much whereas, for Arxiv and ArxivCS, PV-DM decreased the f1score as compared to PV-DBOW

5.2.5 TADW

TADW [42] has two hyper-parameters; regularizer and feature vector size. Since this approach is based on matrix factorization, we cannot introduce β to control how much information we want from text and network view respectively. For experimentation, we studied the effect of regularization and feature vector size on the DBLP dataset only. Figure 39 shows the result of classification when we varied the regularization. We changed regularization from 0.1 to 1 and we observed that as we increased the regularization the model performance improved. TADW gave the best result when regularization was set to 0.75 for the DBLP dataset. For this experiment, we set feature size to 200.

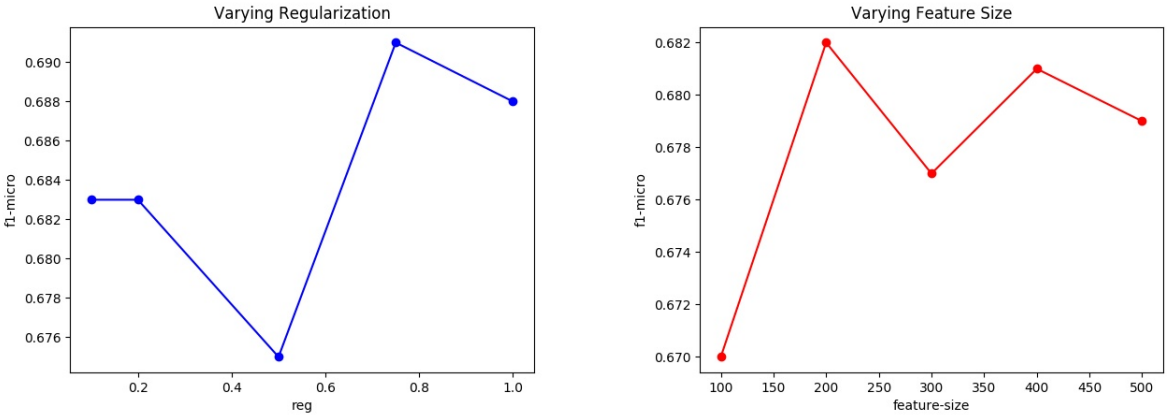
Another hyperparameter we experimented on was the feature size. By default, the model always obtained feature vector of 200 dimension but we made it a hyper-



(a) Arxiv **(b)** ArxivCS

FIGURE 38: Embedding visualization using PV-DM for Arxiv and ArxivCS dataset.

parameter and experimented with different lengths of vectors. This feature vector is obtained after passing binary TFIDF vector to the SVD [32]. Figure 39 shows that the feature vector of 200 dimension gave the best result of 0.682 $f1score$. As we increased the feature size, the model performance decreased because the model was now overfitting. We used regularization=0.2 for our experiment.



a Regularization Tuning **(b)** Feature Vector Size Tuning

FIGURE 39: Parameter Tuning for **TADW** **(a)** In this experiment we changed the regularization from 0.1 to 1. As we increased the regularization the model performance improved and gave the best f1score of 0.691 when regularization was 0.75. **(b)** Change in feature size allows the model to use content information from different length of vectors obtained after passing binary vector to the SVD. As we increase the feature vector size, the f1 score improves. The best results were obtained when we set the feature vector to 200 dimension.

5.3 Clustering

This section will evaluate the embeddings obtained from different methods using clustering. We use K-means clustering implemented in Scikit-Learn [kme] and use Purity [30] and Silhouette score [sil] for evaluation. Purity is a simple evaluation measure that calculates the percentage of the total number of objects that were classified correctly. We compare the ground truth labels of each data point in a cluster and count its frequency. Then we take the maximum count of label frequency in each cluster and divide it by the total number of data points. This calculation requires true label values for each point, and the purity value ranges from 0 to 1. Purity score of 1 means that each cluster contains the data points of the same label. Silhouette score is another evaluation metric we used for our experimentation. This score determines how similar an object is within its cluster and how dissimilar it is to the other clusters. Silhouette score of 1 is the best meaning the object is well

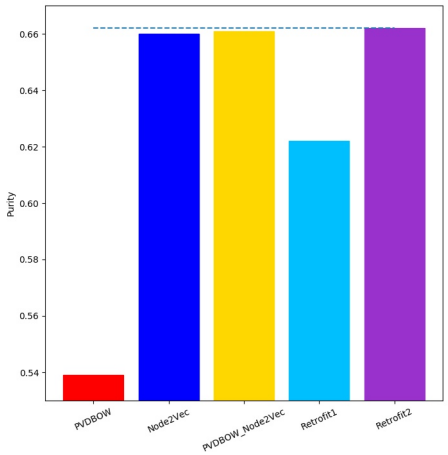
matched within its cluster and the least score of -1 indicating it is mismatched within its cluster and has more similarity with the other clusters. The aim is to get the Silhouette score as close as to 1.

Figure 40 shows the purity score of all the methods for all the datasets. In DBLP, retrofit2 gave the best score followed closely by concatenation. For DBLPadv dataset, retrofit1 and retrofit2 outperformed all the other methods. In Arxiv dataset, concatenation performed the best, and for the remaining datasets, retrofit1 outperformed all the other methods. Retrofit1 performed better than the other methods because it can adjust the weight of network information well and did not bring all the citing documents to close together. For example in Arxiv and ArxivCS datasets, the network quality is not good because different papers are citing documents from different classes. Addition of network information will bring different classes closer together. As a result, these classes will be clustered together in a single cluster and will reduce the purity score. In the case of retrofit1, through β we can adjust the weight of adding network information. As a result, we bring the citing document embeddings closer together but not that close that the documents from the different classes are grouped in a single cluster.

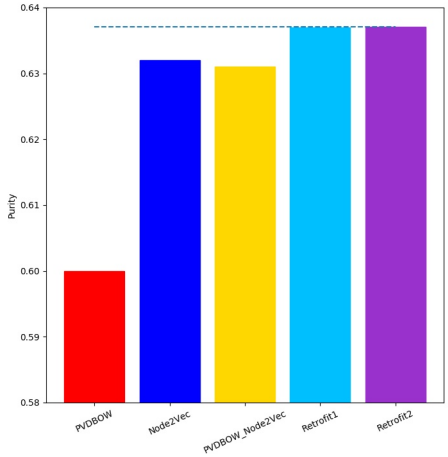
In the next experiment, we will use the Silhouette score to analyze this observation in more detail. The purpose of the Silhouette score is to calculate the similarity of the data point within a cluster, and it's dissimilarity to the nearest cluster. We do not know the ground labels and use cosine similarity to calculate the distance between points. To link this experiment with our previous observation, what we want to see is that Silhouette score should be in the range of 0.4 to 0.6. That means that points in the cluster are relatively close to each other and are also dissimilar to their nearest clusters.

Figure 41 shows the Silhouette plot for the DBLP dataset using PV-DBOW, node2vec, and concatenation of PV-DBOW and node2vec. The figure on the left

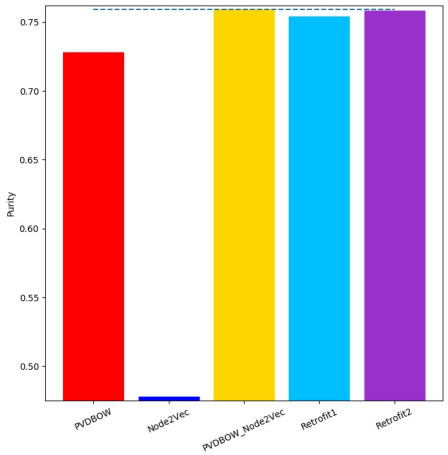
DBLP



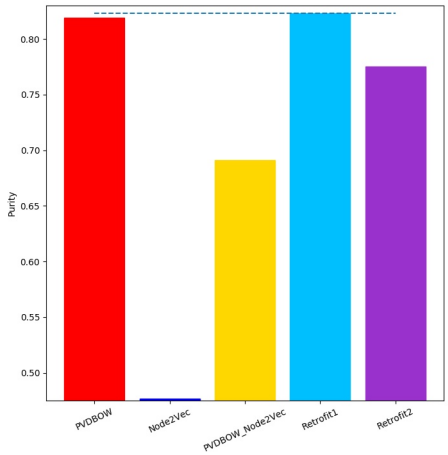
DBLPadv



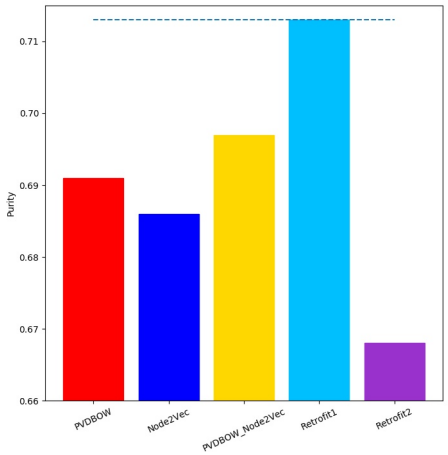
Arxiv



ArxivAbs



ArxivCS



ArxivCSAbs

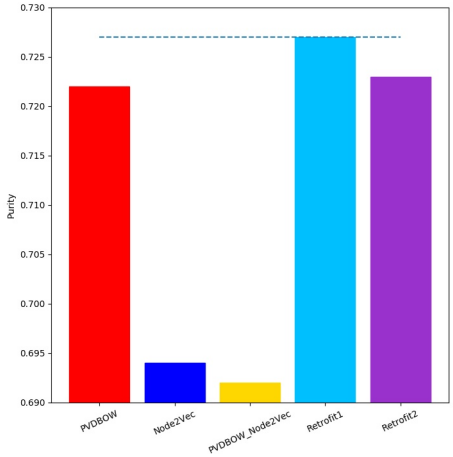
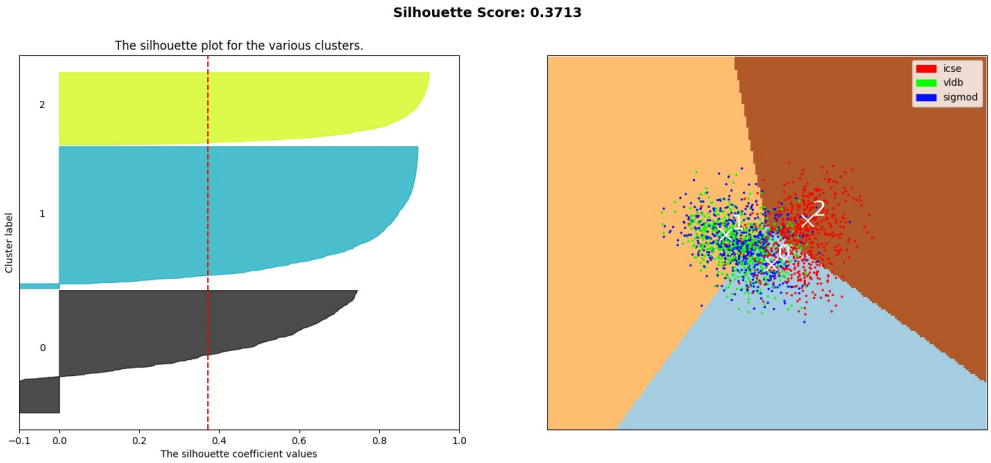
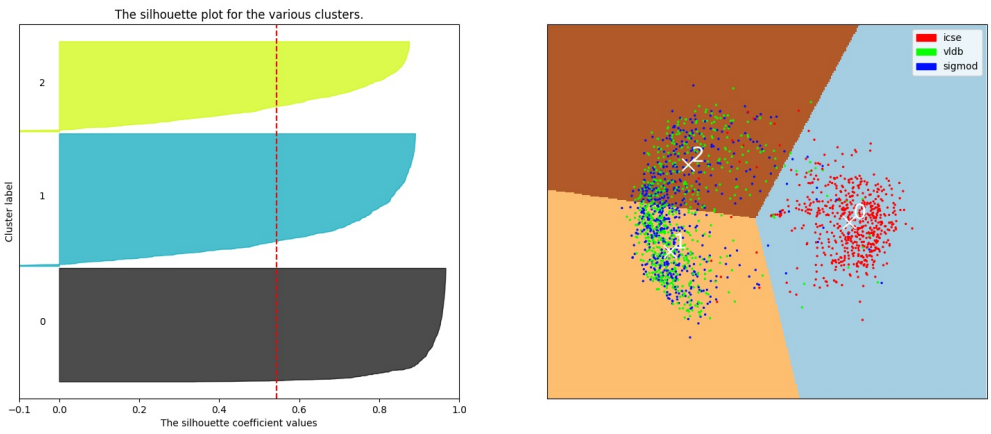


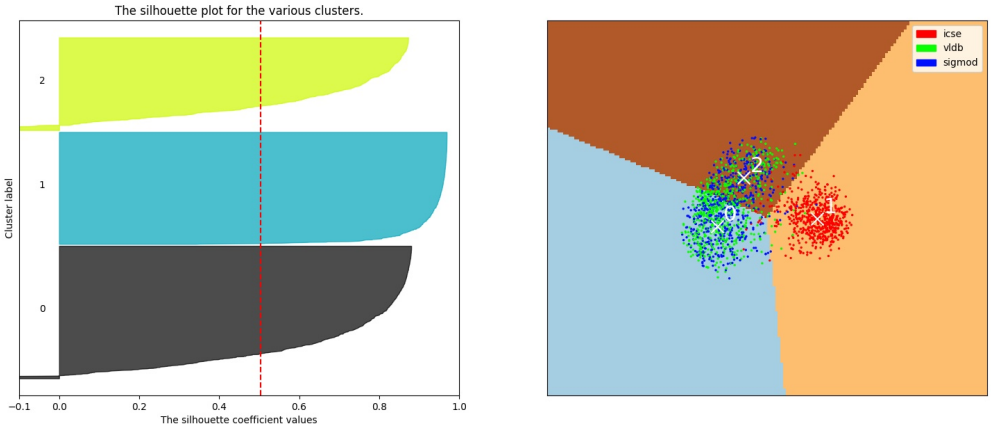
FIGURE 40: Purity score for all the methods for individual datasets. Retrofit1 performed the best in general as compared to concatenation.



(a) **PV-DBOW**



(b) **Node2vec**



(c) **Concatenation of PV-DBOW and Node2vec**

FIGURE 41: Silhouette score of DBLP dataset for PV-DBOW, node2vec and concatenation of PV-DBOW and node2vec

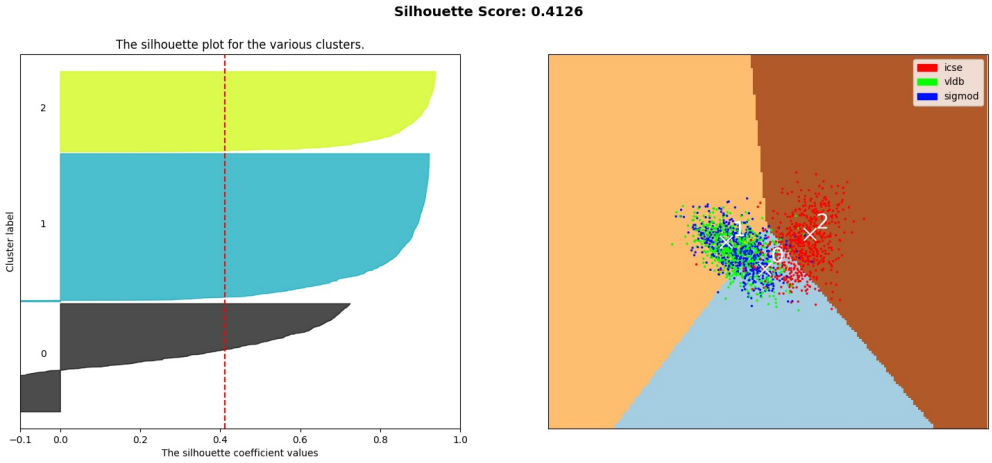
shows the Silhouette coefficient values. The red dotted line is the mean score. We want this line to be as close as to the 1. The colored region is the Silhouette score of each point within the clusters. We want each cluster plot to be above the mean value as much as possible, and the width of these plots should be uniform for all the clusters. On the right is the visualization of the clusters using PCA [41]. Each point represents different documents, and the color of points are its labels. Separate sections of colors in the plot shows the region of each cluster. X mark represents the cluster center and the value associated with it links it with the Silhouette coefficient plot on the left.

Part a of figure 41 shows that clustering did not perform well as the distribution of clusters is not uniform, and for the yellow cluster, some of the points are even negative meaning they are not similar to the assigned cluster. Part b shows improvement as the distribution is relatively more uniform, but still, some of the black and yellow points are in negative. Concatenation increases the negative points in the yellow cluster, and the distribution is not uniform showing that the clustering did not perform well.

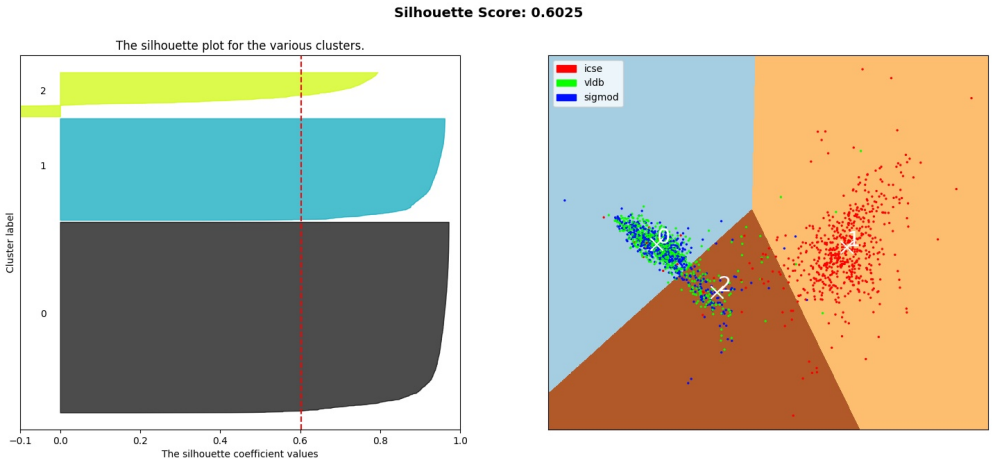
Figure 42 shows the Silhouette plot for the retrofit1 and retrofit2 for DBLP dataset. We can observe that retrofit2 has the best mean score. Though there are some points in the negative for the yellow cluster but for the other two cluster we have a wide distribution.

Figure 43 shows the clustering result for DBLPadv dataset. For PV-DBOW, we can see that many points are in the negative and the mean score is also minimal. Node2vec has better distribution, and the mean score is 0.5197. We can still observe points in the cluster 0 and 1 in negative because Sigmod and VLDB documents are both from the database class, so it is difficult to separate them in a cluster. Concatenation of node2vec and PV-DBOW increases the mean Silhouette score, and still we some points in the negative for cluster 1 and 2.

Figure 44 shows the clustering result for the retrofit1 and retrofit2 using DBLPadv

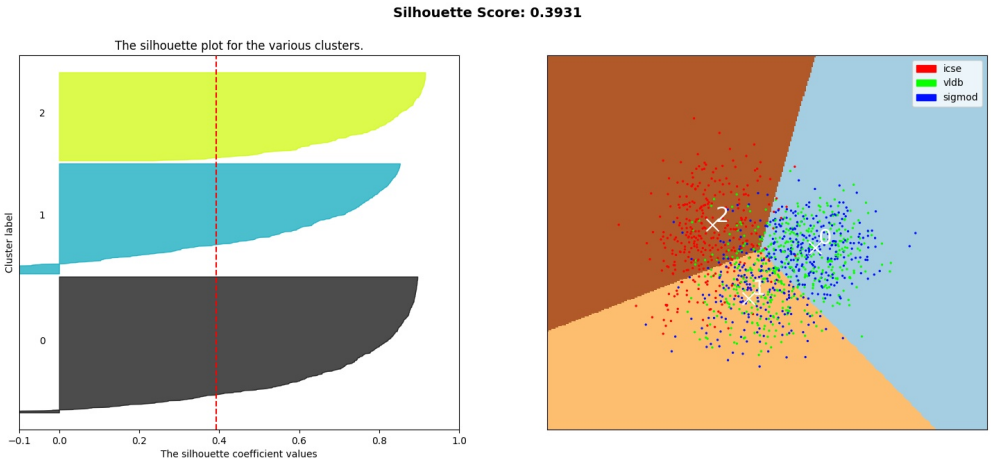


(a) Retrofit1

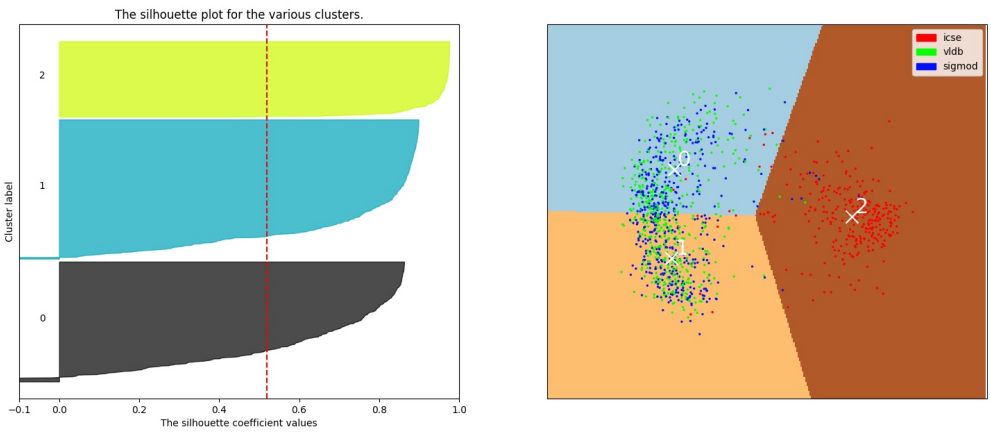


(b) Retrofit2

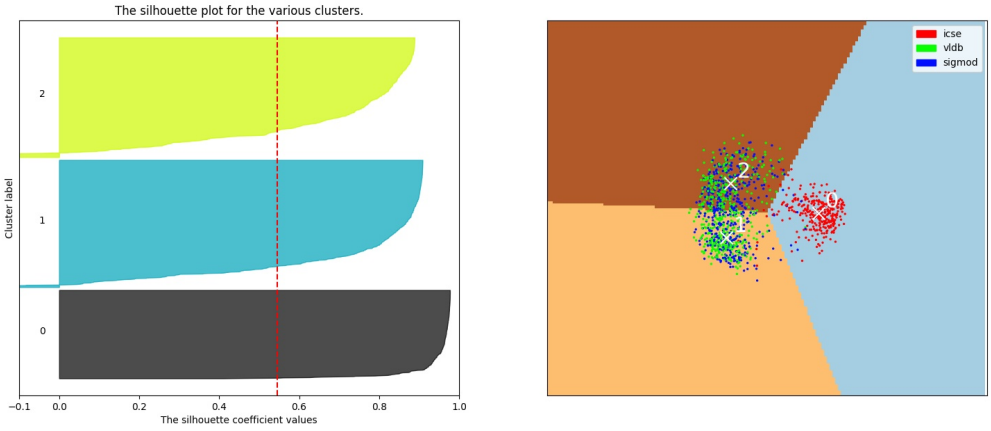
FIGURE 42: Silhouette score of DBLP dataset for retrofit1 and retrofit2



(a) **PV-DBOW**

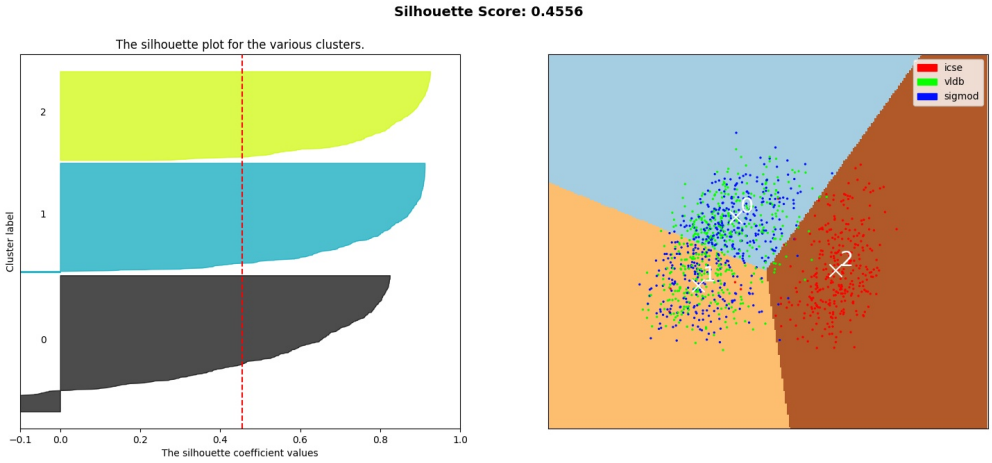


(b) **Node2vec**

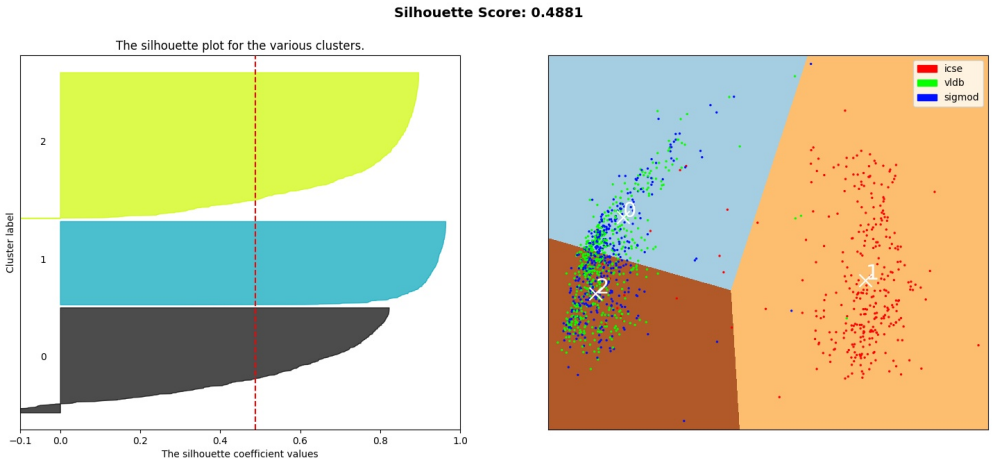


(c) **Concatenation of PV-DBOW and Node2vec**

FIGURE 43: Silhouette score of DBLPadv dataset for PV-DBOW, node2vec and concatenation of PV-DBOW and node2vec



(a) **Retrofit1**



(b) **Retrofit2**

FIGURE 44: Silhouette score of DBLPadv dataset for retrofit1 and retrofit2

dataset. We can observe that retrofit1 did not perform well because there are many points in the negative for the cluster 0 and 1. Retrofit2 is much better, as the mean Silhouette score is 0.488 and the cluster distribution is above the mean score and is uniformly distributed.

We observe a similar pattern in other datasets for the Silhouette score. The plots for those datasets can be accessed in the Appendix 6.

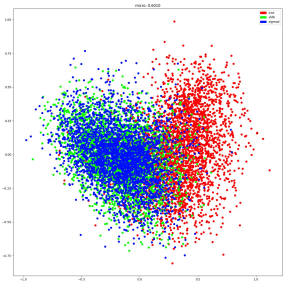
5.4 Document Visualization

In this section, we used PCA [41] to project the embeddings produced from different methods into 2-dimensional vector space with the objective that the papers belonging to the same class should be clustered together. If the distance between different classes is more that means the embeddings were able to capture the semantics of the documents successfully and were able to differentiate different topics in the dataset. If the classes are merged, that means these classes are linked together through network and content information.

DBLP Visualization

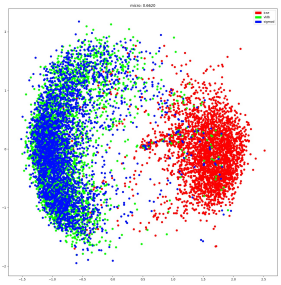
Figure 45 gives the visualization of 3000 vector representation of documents per class for the DBLP dataset. We can observe that the ICSE class was able to separate successfully for all the methods whereas Sigmod and VLDB failed to separate. Sigmod and VLDB both represent documents related to the database management and ICSE contains documents related to software engineering. Since VLDB and Sigmod contains same kind of words related to databases as a result they could not be separated. Retrofit2 performed the best as it got the classification score of 0.668 for the 2-dimensional vector representation. Vector representation obtained from node2vec produced classification score of 0.662 and the concatenation of PV-DBOW

$F1_{micro}$:**0.601**



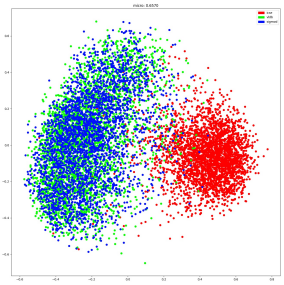
(a)PV-DBOW

$F1_{micro}$:**0.662**



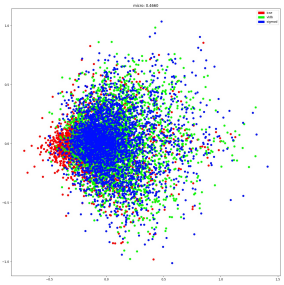
(b)Node2vec

$F1_{micro}$:**0.657**



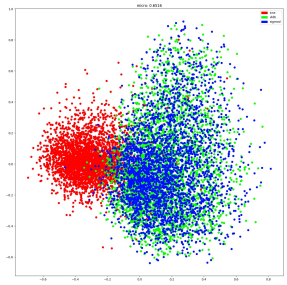
(c)PVDBOW_Node2vec

$F1_{micro}$:**0.466**



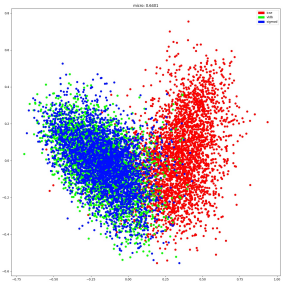
(c)LDE)

$F1_{micro}$:**0.652**

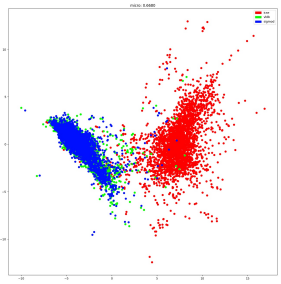


(d)TADW

$F1_{micro}$:**0.668**



(e)Retrofit1



(f)Retrofit2

Key: vldb sigmod icse

FIGURE 45: **Dataset Used**: DBLP **Key**: Red:ICSE Green:VLDB Blue:Sigmod

with node2vec produced 0.657 score for the classification.

DBLPadv Visualization

Visualization of the vector representation obtained from the DBLPadv dataset also produced a similar result as shown in Figure 46. ICSE papers were able to get separated for all the methods whereas VLDB and Sigmod failed to get separated. Retrofit2 performed the best followed by concatenation of PV-DBOW with node2vec.

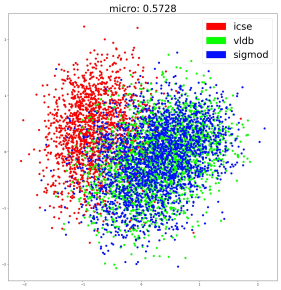
Arxiv Visualization

For Arxiv, we can see in Figure 47 that adding network information did not improve the results much. Papers belonging to physics and maths are together, and computer science and statistics documents are clustered together in the 2-dimensional vector space for Node2Vec. The classes are better separated for the content based method like PVDBOW. When $\beta = 0.5$ we are giving equal weight to content and network view for learning the embeddings in retrofit2. Since the network is not good therefore the classes are not able to get separated. With $\beta = 0.25$ retrofit2 performed the best followed by retrofit1 and concatenation of PV-DBOW and node2vec. This is because we give more weight to the content than the network.

ArxivCS Visualization

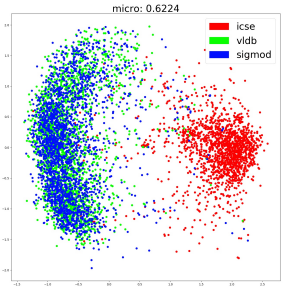
We observe the same pattern for ArxivCS in Figure 48. Node2vec alone performed poorly as machine learning, and computer vision papers are clustered together, and computation complexity is clustered together with the distributed computing papers. Retrofit2 again produced the best classification score of 0.641 with $\beta = 0.25$ followed by retrofit1 and concatenation of PV-DBOW and node2vec.

$F1_{micro}$:**0.573**



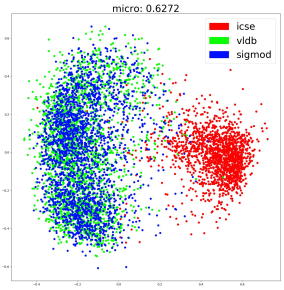
(a)PV-DBOW

$F1_{micro}$:**0.6224**



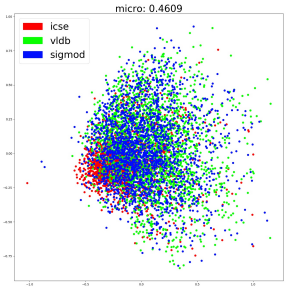
(b)Node2vec

$F1_{micro}$:**0.627**



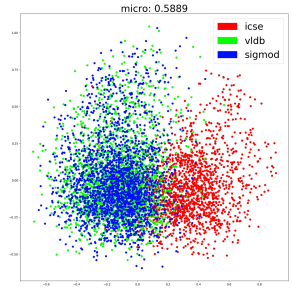
(c)PVDBOW-Node2vec

$F1_{micro}$:**0.461**



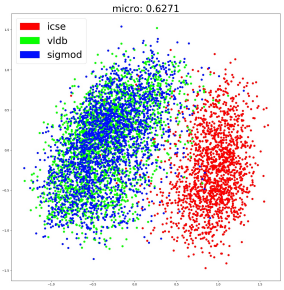
(c)LDE

$F1_{micro}$:**0.589**



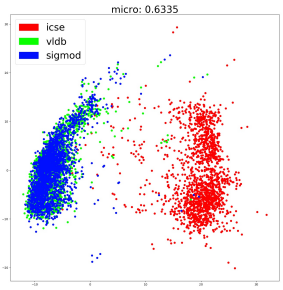
(d)TADW

$F1_{micro}$:**0.627**



(e)Retrofit1

$F1_{micro}$:**0.634**

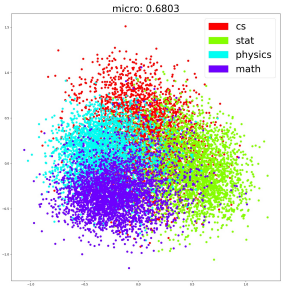


(f)Retrofit2

Key: vldb sigmod icse

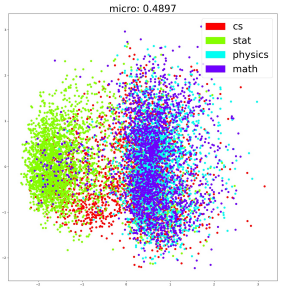
FIGURE 46: **Dataset Used**: DBLPAdv **Key**: Red:ICSE Green:VLDB Blue:Sigmod

$F1_{micro}$:**0.680**



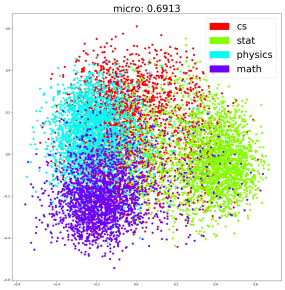
(a)PV-DBOW(d2v)

$F1_{micro}$:**0.450**



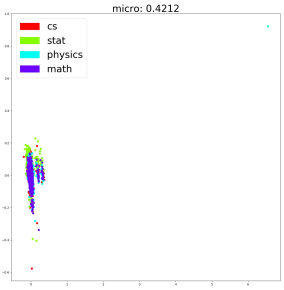
(b)Node2vec

$F1_{micro}$:**0.691**



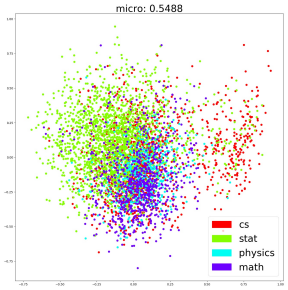
(c)PVDBOW-Node2vec

$F1_{micro}$:**0.422**



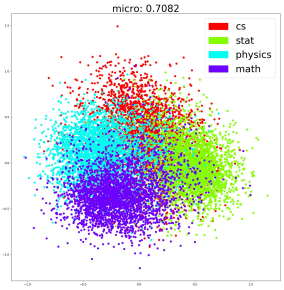
(c)LDE

$F1_{micro}$:**0.549**



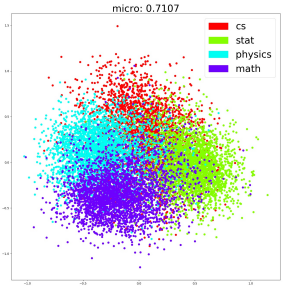
(d)TADW

$F1_{micro}$:**0.708**



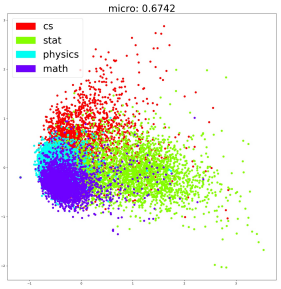
(e)Retrofit1

$F1_{micro}$:**0.711**



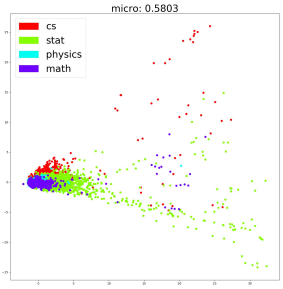
(f)Retrofit2 β 0.25

$F1_{micro}$:**0.674**



(g)Retrofit2 β 0.375

$F1_{micro}$:**0.580**

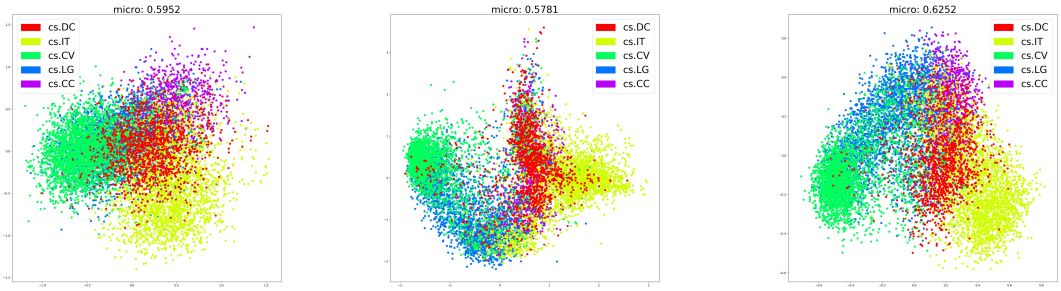


(h)Retrofit2 β 0.5

Key: stat physics cs math

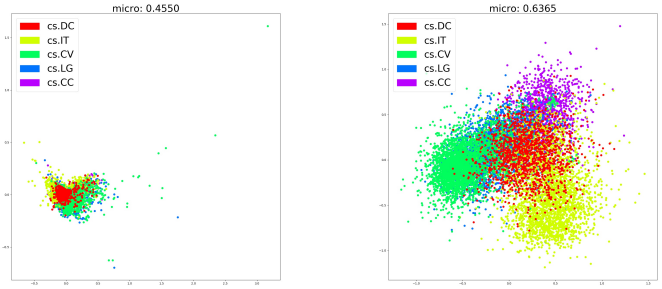
FIGURE 47: **Dataset Used**: Arxiv **Key**: Red:CS Green:Stat Blue:Phy Purple:Math

$F1_{micro}$:**0.595** $F1_{micro}$:**0.578** $F1_{micro}$:**0.625**



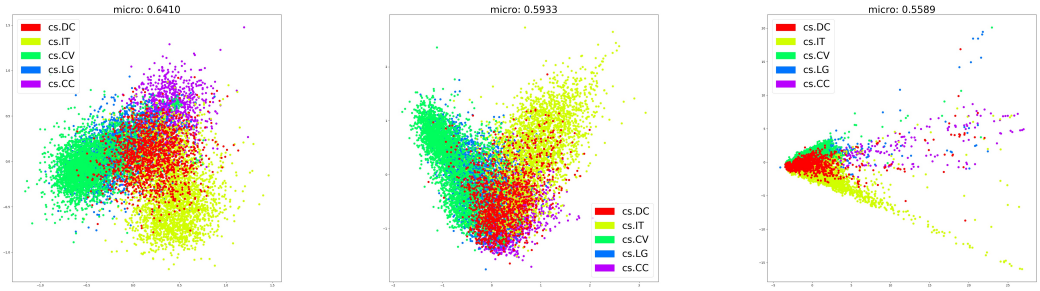
(a)PV-DBOW **(b)Node2vec** **(c)PV-DBOW-Node2vec**

$F1_{micro}$:**0.455** $F1_{micro}$:**0.636**



(c)LDE **(e)Retrofit1**

$F1_{micro}$:**0.641** $F1_{micro}$:**0.593** $F1_{micro}$:**0.559**



(f)Retrofit2 β 0.25 **(g)Retrofit2 β 0.375** **(h)Retrofit2 β 0.5**

Key: Information Theory Distributed Computing Computational Complexity Computer Vision Machine Learning

FIGURE 48: **Dataset Used**: ArxivCS **Key**: Red:DC Green:CV Blue:LG Purple:CC Yellow:IT

Retrofit Animation

To understand the effect of β on retrofit, we created animations showing how the documents in the two-dimensional vector space move when the network information is added. The animation can be accessed at `https://github.com/ZeeshanMansoor260/Masters/tree/master/DataVisualization`. The animations show that the addition of network information will move the different classes away from each other as we increase β. When $\beta = 1$, different classes start to come closer together as we are using the information only from the network.

ID	Title	Ref
ICSE:0	Lightweight adaptive filtering for efficient learning and updating of probabilistic models	Cites 6 icse papers
ICSE:1	Consistent group membership in ad hoc networks	Cites 1 icse paper
ICSE:2	Stochastic Modelling of Seasonal Migration Using Rewriting Systems with Spatiality	Cites 2 icse paper
ICSE:3	Stochastic Modelling of Seasonal Migration Using Rewriting Systems with Spatiality-HOLMES: Effective statistical debugging via efficient path profiling	Cites 3 icse paper
VLDB:6	Probabilistic nearest neighbor queries on uncertain moving object trajectories	Cites 4 sigmod and 2 vldb papers
VLDB:7	REFEREE: an open framework for practical testing of recommender systems using ResearchIndex	Cites 2 vldb papers
VLDB:8	ZOO: A Desktop Experiment Management Environment	Cites 1 vldbl paper
Sigmod:4	The test data challenge for database-driven applications	Cites 2 icse, 1 vldb and 1 sigmod paper
Sigmod:9	Clio grows up: from research prototype to industrial tool	Cites 12 vldb and 7 sigmod paper

TABLE 20: Paper Label Details

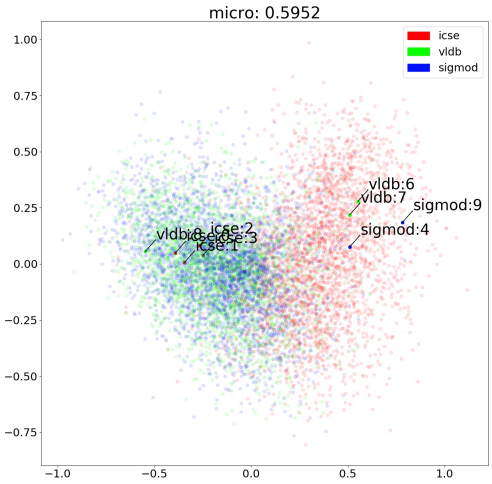
5.4.1 Labels Plots

In this experiment, we wanted to visualize the effect of adding network information to the documents independently. We used PCA [41] to project 100-dimensional vectors obtained from PV-DBOW to 2-dimension vectors. Then we identified some documents which were placed with the wrong classes in the visualization. For example, Sigmod or VLDB documents grouped with ICSE documents and similarily ICSE documents placed with VLDB.

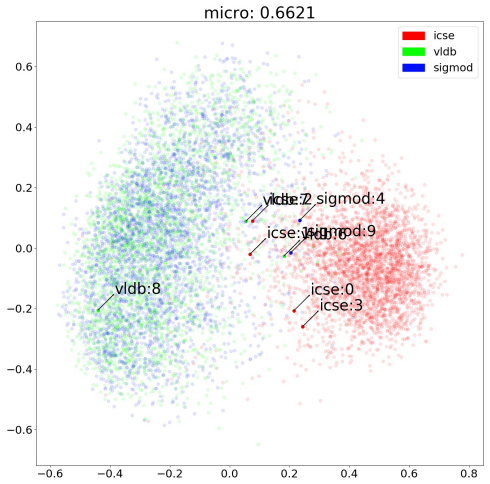
Table 20 gives detail about the selected documents. We selected four ICSE documents with content similar to the VLDB and Sigmod class. For example, these documents contain words like model, migration, and group membership which can be

found in high probability in the database documents. Similarly, VLDB and Sigmod documents have content similar to ICSE because they contain words like framework, recommender system, test-data, and application. These words are more likely to be found in the software engineering documents. We also list the class of documents these papers are citing. Most of the documents are citing papers within their class.

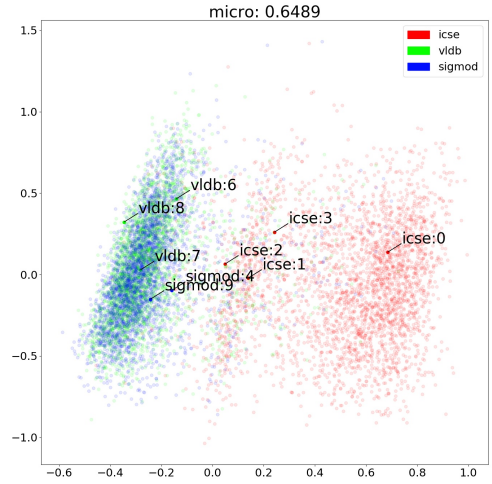
The figure 49 (a) shows the visualization of the 100-dimensional vector obtained from PV-DBOW after passing it through PCA. The document outlined in table 20 are identified in the plot. Initially, the documents are placed with the wrong classes because of their content information. Part (b) shows the visualization of the embedding obtained after concatenating node2vec with PV-DBOW and passing it through PCA. We can see with the addition of network information, identified documents brought closer to their respective classes, but VLDB and Sigmod documents are still far from their true classes. Part (c) is obtained from LDE. Identified VLDB and Sigmod documents are placed with the other documents in their class. For ICSE, it is forming a new cluster which is close to the database documents because of their content. Also, LDE failed to place identified VLDB documents close together. Finally, part (d) represent retrofit2. We can see that the ICSE documents are placed correctly with the other documents in their class. And VLDB and Sigmod are also separated, as identified VLDB and Sigmod documents are clustered together respectively.



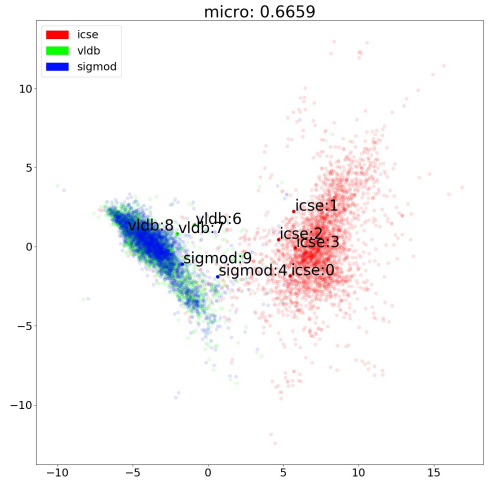
(a)PV-DBOW



(b) Concatenated PV-DBOW and Node2vec



(c)LDE



(d) Retrofit2

FIGURE 49: Visualization of how the documents are separated based on the content and network information. In PV-DBOW, ICSE documents are wrongly placed with Sigmod and VLDB documents because of their titles and same applies for Sigmod and VLDB documents. After adding network information documents start to come closer together to their respective classes. For LDE and concatenated PV-DBOW and node2vec these documents are placed between the classes. But in retrofit2, these document are placed in the correct classes and also VLDB and Sigmod documents are brought closer.

CHAPTER 6

Conclusion

In this thesis, we studied the effect of adding network information to the embeddings obtained from content only. We wanted to improve the embeddings obtained from methods like PV-DBOW which uses only the text information to train the model. These methods assume documents to be independent of each other. The reality is contrary to this assumption. Documents are linked together in the form of hyperlinks or references for webpages or academic papers respectively. These content-based methods ignore the linkage information altogether. Likewise, network representation learning methods like node2vec will learn network information. These methods ignore the text attribute of documents in the network and focus only on the linkage part. The method proposed in this thesis try to incorporate linkage information to the embeddings obtained using just content.

For experimentation, we use three main datasets: DBLP, Arxiv, and ArxivCS. Each dataset contains two variants, less content (title only), and more content (abstract). We created both variants of Arxiv, ArxivCS, and DBLPadv dataset. We wanted to do experimentation on different networks with the varying size of the content.

In the experimentation, we show the classification score of different methods using various datasets. The results show that concatenating node2vec with PV-DBOW gives the best result followed by retrofit2. However, concatenation does not allow to control the weight of adding network information to the document embeddings.

Moreover, concatenation doubles the vector length; as a result, the resultant vector has more room to store information. LDE did not perform well on all the datasets. TADW faced scalability issues and gave memory error for ArxivAbs and ArxivCSAbs dataset.

The experimentation showed that when adding network information to the embedding obtained from larger text gives little improvement as compared to the less content. This is because more text contains partially the same type of information as in the network. The embeddings will capture the semantic of documents which contains information present in the network too. When the content size is small, the model does not have enough information to train and looses some of the aspects of the documents. On the other hand, the citation graph contains different aspects of the same documents. When we combine these views, the resultant vector will have information from both the aspects and will produce a better classification score.

To analyze the classification score in depth, we identified the reasons why different methods fail for various datasets. We performed rigorous experimentation to find the best parameters for different methods using all the datasets. In concatenation, we observe the best classification score when sampled nodes are near to the source node. This is because nodes near to the source node are directly linked through citation and share a similar topic. In the retrofit, we observe the best classification score is obtained when $\beta = 0.5$. It means we are using an equal amount of information from network and content. LDE also follow a similar pattern, but the experimentation result shows that LDE is a difficult model to train. It is very sensitive to network and content information and is not able to use the information optimally when training.

To understand the classification score in more depth, we visualized the embeddings from all the methods. The objective was to visualize the effectiveness of different methods and to study the structure of the datasets. The goal is to bring documents of different classes closer together in the vector space. In general, retrofit2 performed

the best in separating the documents based on their classes. We also analyzed the effect of adding network information on individual documents by tracking them using different methods. We show that retrofit2 was successfully able to bring the identified documents back to their respective classes and also brought similar documents together within the same class.

Limitations

The limitation of our method is that we need a vector representation of the documents as an input. We cannot add citation information to the documents in their raw text form. Secondly, our method is dependent on the quality of the network. If the network is citing documents from different classes, then the addition of network information will not separate the classes. Also, we can not add citation information to the document embedding if it is not present in the network graph.

Future Work

Our research focuses on the linkage of document embeddings using the citation graph only. But these documents can be linked together based on different relationships like similar keywords and authors. In the future, it would be interesting to see how our model responds when we try to add network information based on the common keywords and authors. It will also be interesting to analyze the significance of β when used in a different scenario like keywords.

In the end, we can conclude that the addition of network information to the document embedding improves the classification score in general. This improvement is dependent on the quality of the citation graph and the text length. We can also conclude that retrofit significantly improves the classification score in all the dataset.

References

[dbl] Dblp: Computer science bibliography. https://dblp.uni-trier.de. Accessed 16-April-2019.

[wik] M. mahoney. large text compression benchmark. www.mattmahoney.net/dc/textdata. Accessed 16-April-2019.

[pur] More details about contingency matrix https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cluster.contingency_matrix.html. Accessed 16-April-2019.

[ics] More details about icse can be found on the website. https://dblp.uni-trier.de/db/conf/icse/. Accessed 16-April-2019.

[IMD] More details about imdb dataset can be found on the website. http://ai.Stanford.edu/amaas/data/sentiment/index.html.

[kme] More details about kmeans implementation https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. Accessed 16-April-2019.

[LR] More details about logistic regression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[gen] More details about pv-dbow implementation using gensim https://radimrehurek.com/gensim/models/doc2vec.html. Accessed 16-April-2019.

[sig] More details about sigmod can be found on the website. `https://dblp.uni-trier.de/db/conf/sigmod/`. Accessed 16-April-2019.

[sil] More details about silhouette score `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html`. Accessed 16-April-2019.

[Sta] More details about stanford sentiment dataset can be found on the website. `http://nlp.Stanford.edu/sentiment/`. Accessed 16-April-2019.

[arx] More details about the arxiv dataset can be found on this link. `https://arxiv.org`. Accessed 16-April-2019.

[det] More details about the detex can be found on this link. `https://code.google.com/archive/p/opendetex/`. Accessed 16-April-2019.

[Luc] More details about the lucene can be found on this link. `http://lucene.apache.org`. Accessed 16-April-2019.

[n2v] More details about the node2vec source code. `https://github.com/aditya-grover/node2vec`. Accessed 16-April-2019.

[vld] More details about vldp can be found on the website. `https://dblp.uni-trier.de/search?q=vldb`. Accessed 16-April-2019.

[17] Tadw code in matlab `https://github.com/albertyang33/TADW`. Accessed 16-April-2019.

[18] Tadw code in python `https://github.com/thunlp/OpenNE`. Accessed 16-April-2019.

[19] Ambrosio, L. and Dal Maso, G. (1990). A general chain rule for distributional derivatives. *Proceedings of the American Mathematical Society*, 108(3):691–702.

[20] Bengio, Y., Delalleau, O., and Le Roux, N. (2006). 11 label propagation and quadratic criterion.

[21] Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1.

[22] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (2014). Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*.

[23] Golub, G. H. and Reinsch, C. (1971). Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer.

[24] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[25] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM.

[26] Knuth, D. E. (1993). *The Stanford GraphBase: a platform for combinatorial computing*. AcM Press New York.

[27] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.

[28] Leskovec, J. and Krevl, A. (2015). {SNAP Datasets}:{Stanford} large network dataset collection.

[29] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[30] Manning, C., Raghavan, P., and Schütze, H. (2010). Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103.

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

[32] Natarajan, N. and Dhillon, I. S. (2014). Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 30(12):i60–i68.

[33] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM.

[34] Stark, C., Breitkreutz, B.-J., Reguly, T., Boucher, L., Breitkreutz, A., and Tyers, M. (2006). Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl_1):D535–D539.

[35] Tang, J., Qu, M., and Mei, Q. (2015a). Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174. ACM.

[36] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015b). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee.

[37] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. (2008). Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM.

[38] Tang, L. and Liu, H. (2011). Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478.

[39] Wang, S., Tang, J., Aggarwal, C., and Liu, H. (2016). Linked document embedding for classification. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 115–124. ACM.

[40] Wang, W., Arora, R., Livescu, K., and Bilmes, J. (2015). On deep multi-view representation learning. In *International Conference on Machine Learning*, pages 1083–1092.

[41] Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52.

[42] Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. (2015). Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[43] Zafarani, R. and Liu, H. (2009). Social computing data repository at asu.

Appendix: Clustering Results

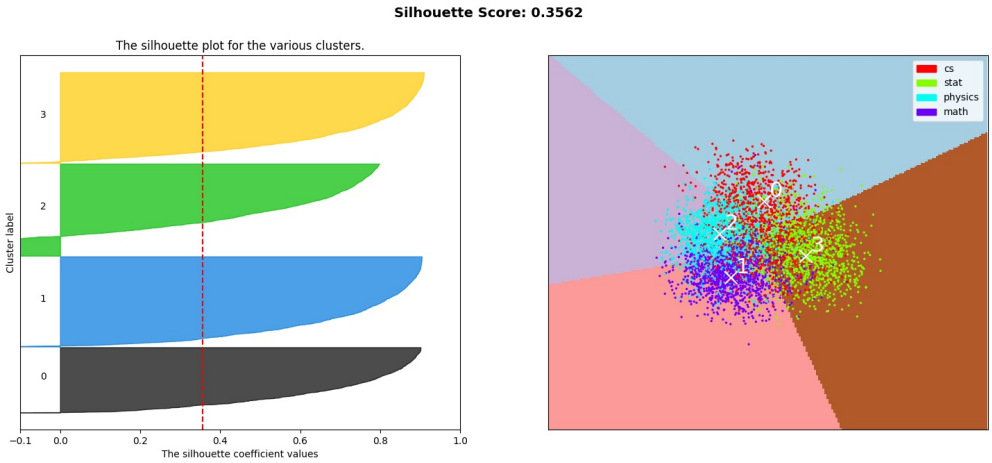
Following figures shows the Silhouette score for Arxiv and ArxivCS datasets.

The clustering result for the Arxiv dataset is shown in the figure 50 using PV-DBOW, node2vec and concatenation. Clustering result of PV-DBOW gave us a Silhouette score of 0.3562 because many points in the cluster 2 are negative. Also the distribution of points is not uniform. Node2vec improved the Silhouette score but the overall distribution is the same and again many points in the cluster 0 are in negative. Concatenation, decreased the Silhouette score to 0.3989 and cluster 3 has many points in the negative.

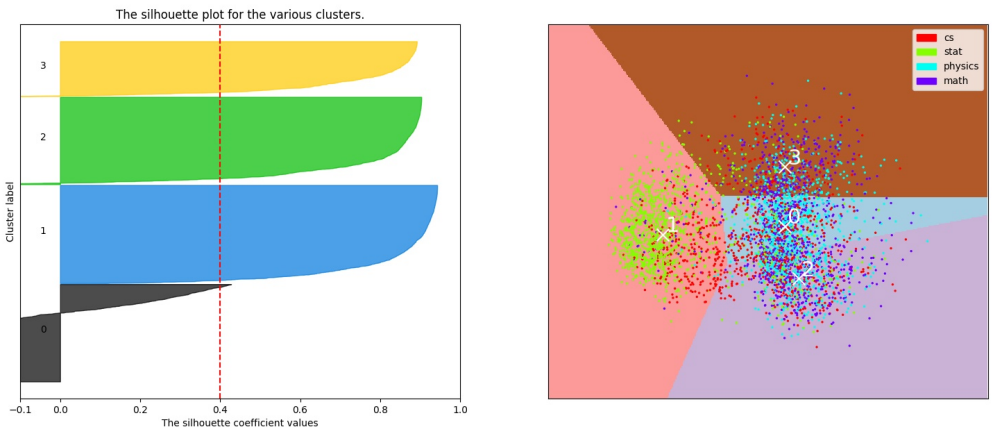
Figure 51 shows the clustering result for both variants of the retrofit. Retrofit1 performed better than the other as the distribution is more uniform and Silhouette mean is 0.3819.

For ArxivAbs, the clustering result for PV-DBOW, node2vec and concatenation is shown in Figure 52. PV-DBOW did not perform well as it achieved the Silhouette score of 0.4154. Also the cluster 1 has some points in the negative. Node2vec gave a Silhouette score of 0.4051, but the distribution is not uniform and cluster 3 has many points in the negative. Concatenation, decreases the mean Silhouette score to 0.388 and has many points in negative for the cluster 0.

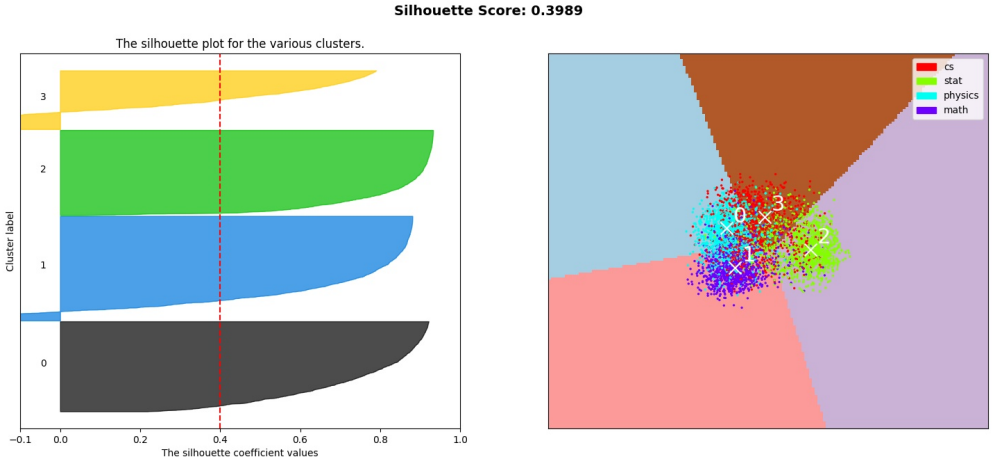
Clustering result of both variants of retrofit is shown in the figure 53. Retrofit1, gave the best Silhouette score of 0.4269. Retrofit2 performed relatively better, but still cluster 2 and 1 have points in the negative.



(a) **PV-DBOW**

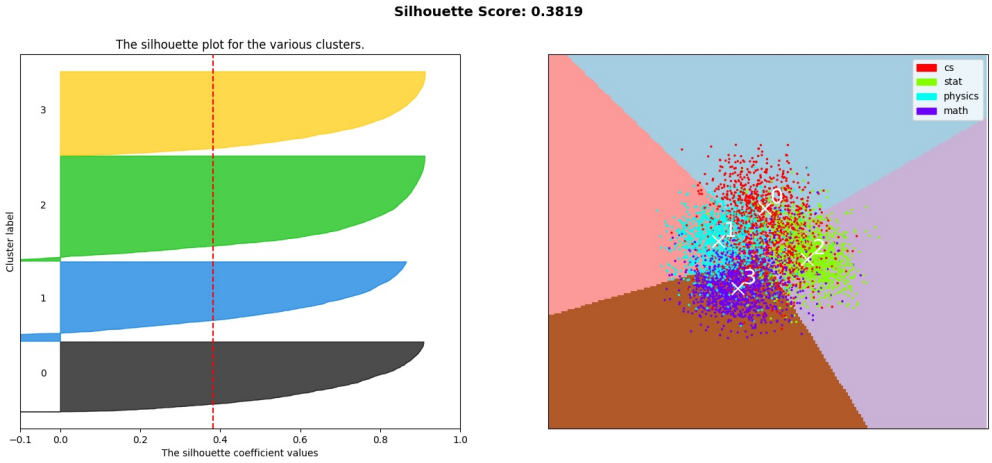


(b) **Node2vec**

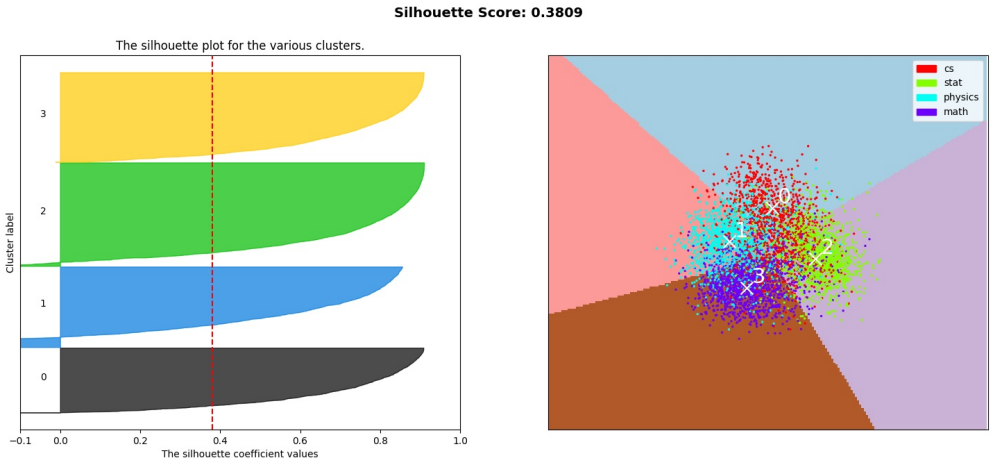


(c) **Concatenation of PV-DBOW and Node2vec**

FIGURE 50: Silhouette score of Arxiv dataset for PV-DBOW, node2vec and concatenation of PV-DBOW and node2vec

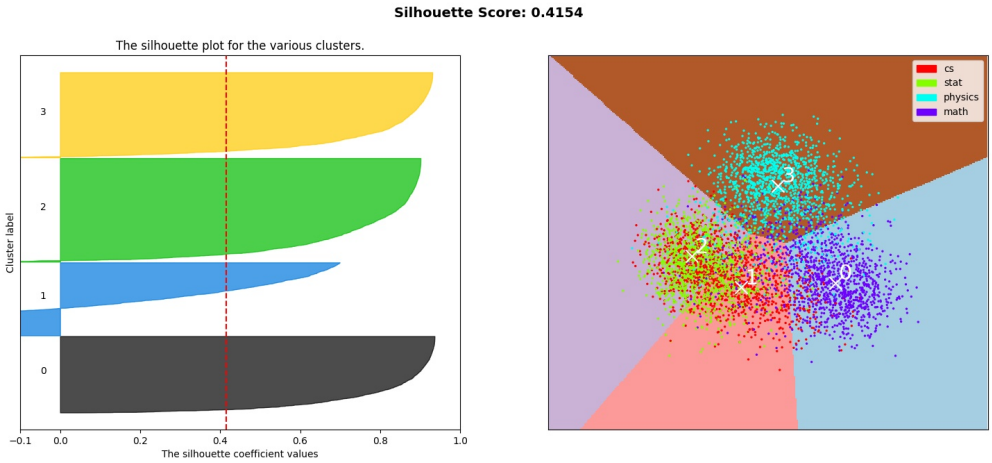


(a) Retrofit1

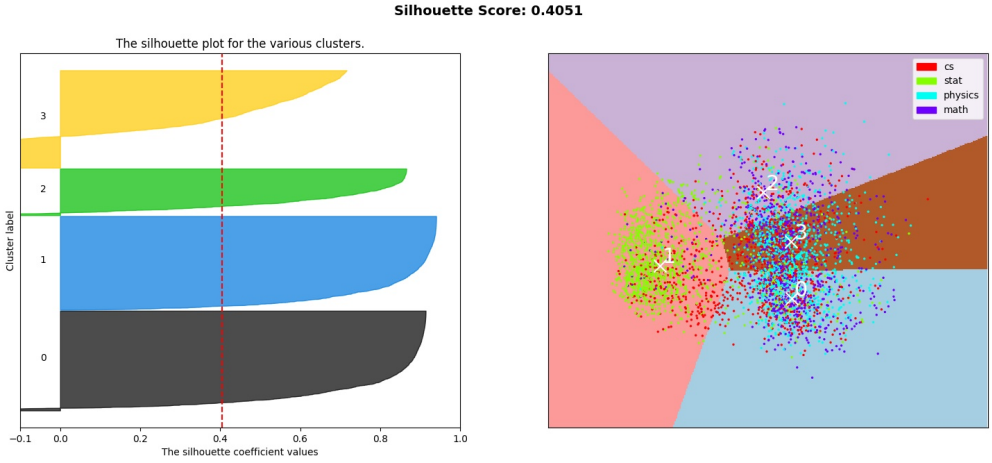


(b) Retrofit2

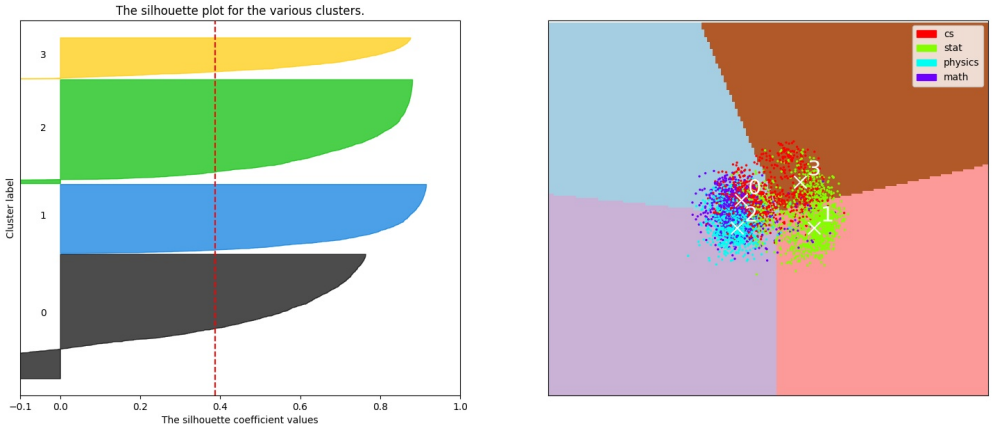
FIGURE 51: Silhouette score of Arxiv dataset for retrofit1 and retrofit1



(a) PV-DBOW

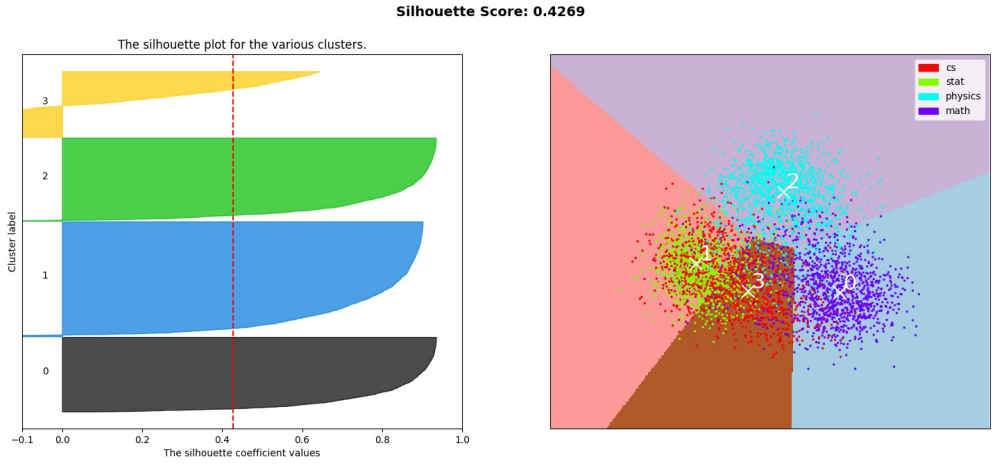


(b) Node2Vec

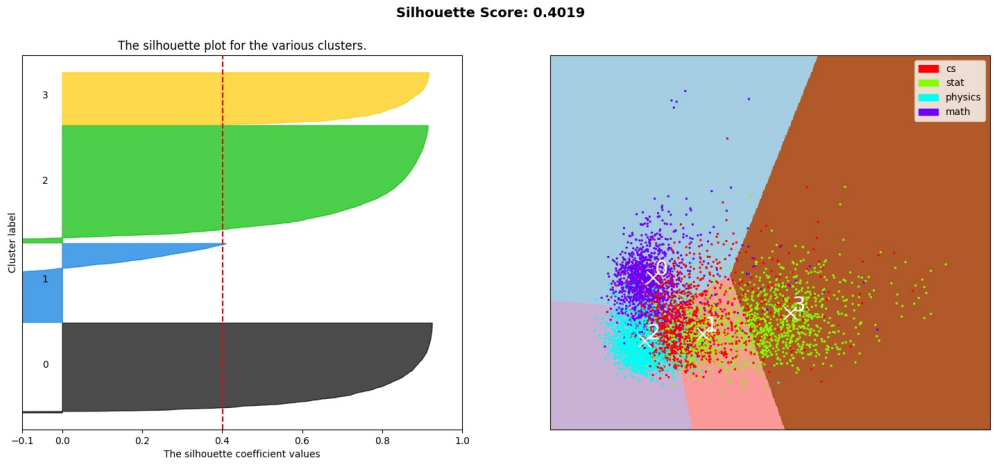


(c) Concatenation of PV-DBOW and Node2vec

FIGURE 52: Silhouette score of ArxivAbs dataset for PV-DBOW, node2vec and concatenation of PV-DBOW and node2vec

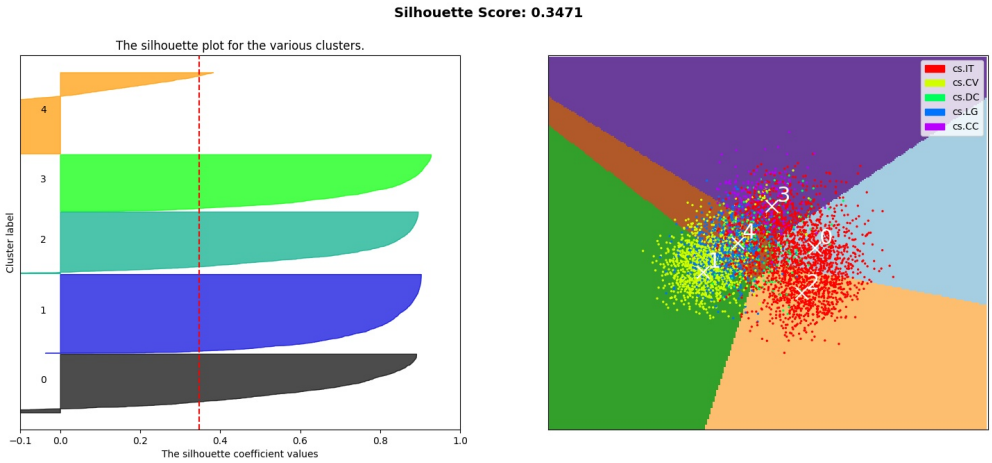


(a) Retrofit1

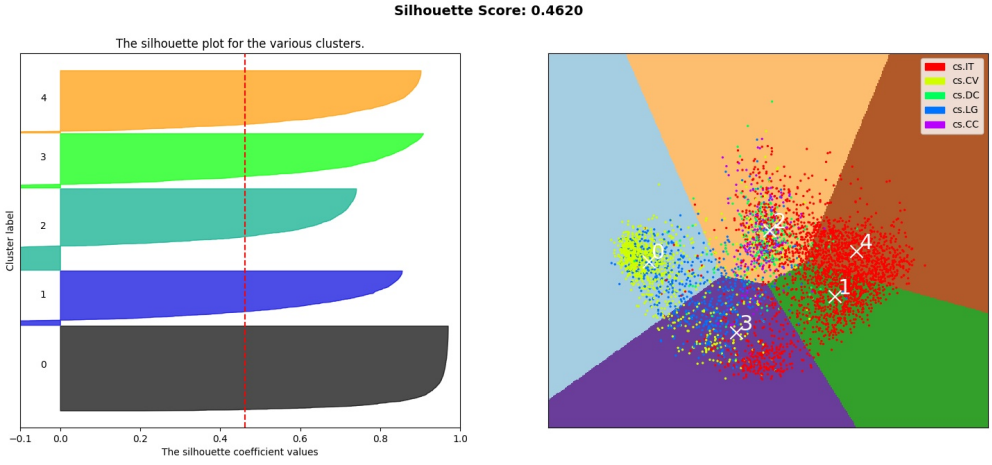


(b) Retrofit2

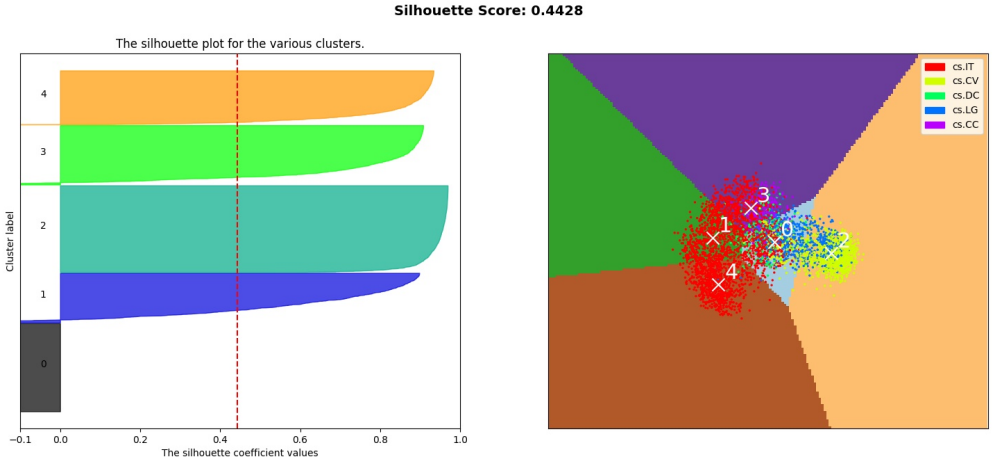
FIGURE 53: Silhouette score of ArxivAbs dataset for retrofit1 and retrofit2



(a) **PV-DBOW**

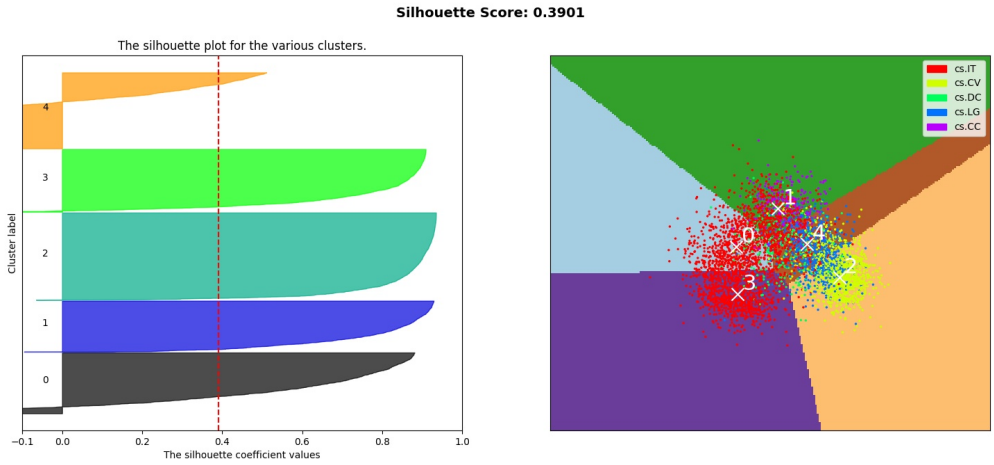


(b) **Node2vec**

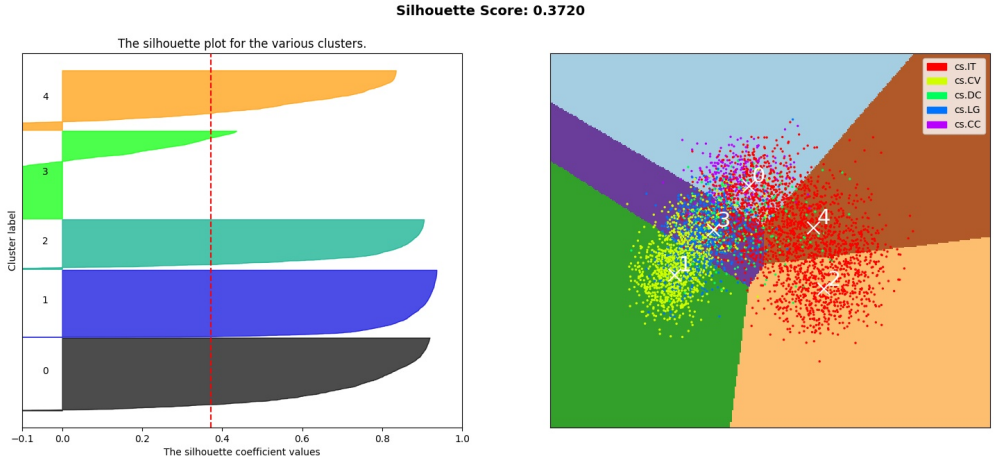


(c) **Concatenation of PV-DBOW and Node2vec**

FIGURE 54: Silhouette score of ArxivCS dataset for PV-DBOW, node2vec and concatenation of PV-DBOW and node2vec



(a) Retrofit1

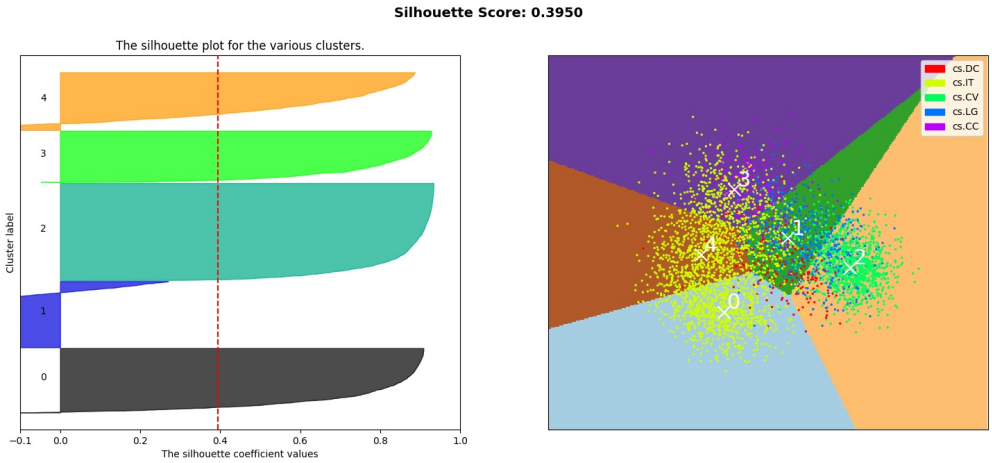


(b) Retrofit2

FIGURE 55: Silhouette score of ArxivCS dataset for retrofit1 and retrofit2

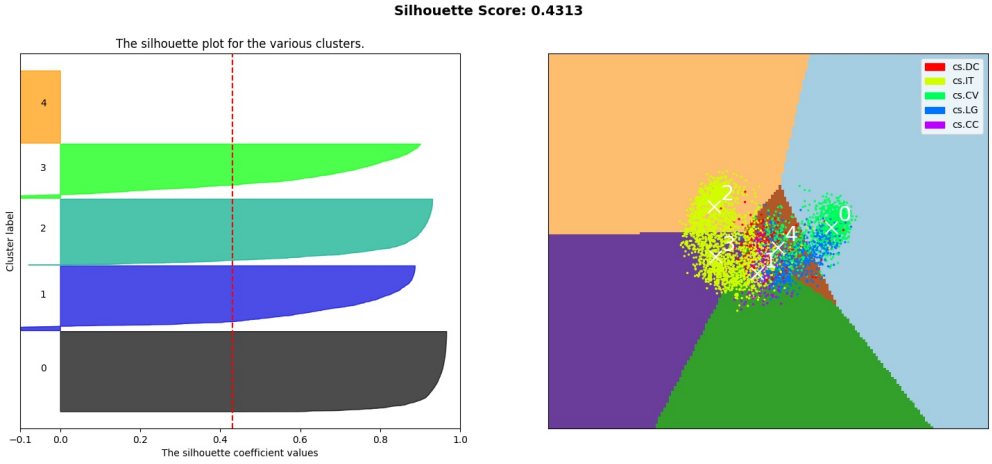
The clustering the results of CS papers is shown in Figure 54 for PV-DBOW, node2vec and concatenation. Node2vec achieved the mean Silhouette score of 0.4620. After concatenation, Silhouette score increased, and the distribution of cluster 0 became all negative.

Clustering result of both retrofit variant is shown in Figure 55. Retrofit1 gave much better Silhouette score of 0.301 but it has some points in negative for the cluster 4. Retrofit2, has a smaller Silhouette score of 0.372.



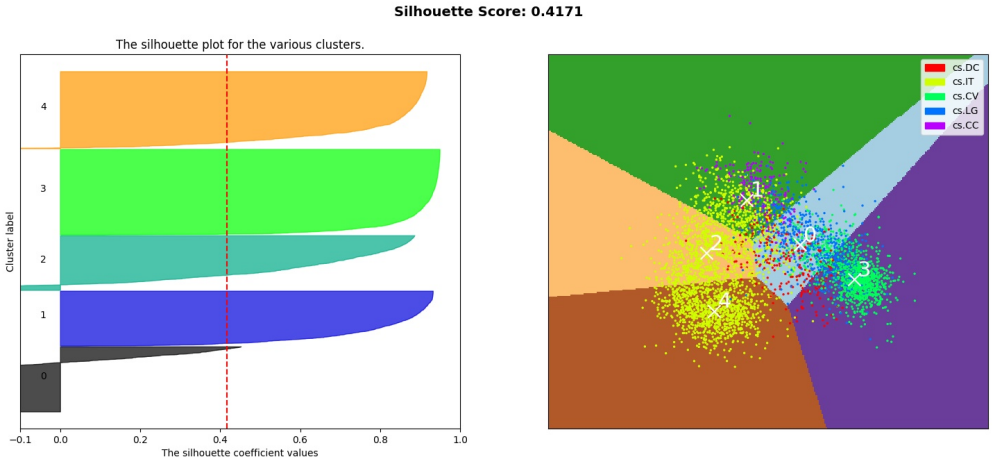
(a) **PV-DBOW**

(b) **Node2vec**

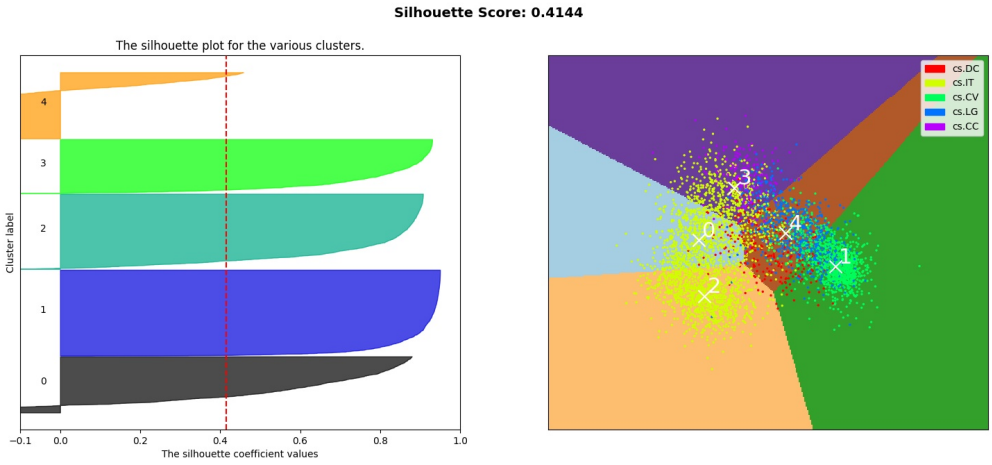


(c) **Concatenation of PV-DBOW and Node2Vec**

FIGURE 56: Silhouette score of ArxivCSAbs dataset for PV-DBOW, Node2Vec and concatenation of PV-DBOW and Node2Vec



(a) Retrofit1



(b) Retrofit2

FIGURE 57: Silhouette score of ArxivCSAbs dataset for retrofit1 and retrofit2

Vita Auctoris

NAME:	Zeeshan Mansoor
PLACE OF BIRTH:	Gujranwala, Punjab, Pakistan
YEAR OF BIRTH:	1993
EDUCATION:	National University of Science and Technology, Bachelors in Software Engineering, Islamabad, Pakistan, 2017